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: a query posed over an ontology is first rewritten using the conceptual part of the ontology and then the evaluation of the rewritten query is delegated to a (deductive) database where the instance data resides. In our previous work we presented a rewriting algorithm for OWL QL that can also deal with most of the QL profile. In order to test the likely practicality of our rewriting algorithm, 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 evaluation, and discuss various optimization techniques aimed at reducing the size of the rewritings. Moreover, we present some preliminary results from an ongoing empirical evaluation of REQUIEM in which we consider 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, e.g., information integration [2]; the use of data repositories to store instance data is becoming increasingly important due to the widespread use of ontologies, e.g., in the semantic Web, and the scalability requirements of many applications.

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 is designed such that queries against an OWL 2 QL ontology and a set of instance data stored in a data repository can be answered by using the ontology to *rewrite* queries such that their evaluation can be delegated to the data repository. 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. [1]. 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. 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 $\mathcal{O}$ [1]. 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 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; the algorithm can therefore be seen as a generalization and extension of CGLLR.

In order to test the likely practicality of query rewriting, and the performance of the different rewriting techniques, 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 would 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—we discuss this optimization in more detail in Section 3. Our empirical evaluation showed that the query subsumption check can significantly reduce

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

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 we describe, in Section 3, an optimization technique that exploits the relation between the ontology and the database where the instance data resides in order to further reduce the size of the rewritings.

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 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 describe an optimization that can be used to reduce the size of the datalog queries obtained by RQR. Moreover, we characterize a case in which such datalog queries can always be transformed into unions of conjunctive queries. Furthermore, we present preliminary results of an ongoing empirical evaluation of REQUIEM in which we consider realistic queries posed over real OWL 2 EL ontologies and show that, in many cases, these can be transformed into unions of conjunctive queries.

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, 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 OWL 2 QL ontology $\mathcal{O}$ to describe DB:[4]

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

$$\exists \text{teaches} \sqsubseteq \text{Teacher} \tag{2}$$

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

Axiom (1) states that professors teach at least someone, axiom (2) states that the domain of the property teaches is Teacher, 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. The advantages of this are that, on the one hand, queries can be posed in terms of the conceptual structure of the data rather than its arrangement in the database, while 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 $\mathcal{M}$ between $\mathcal{O}$ and DB is defined as follows:

$$\text{Professor} \mapsto \texttt{SELECT Name FROM Professor}$$
$$\text{hasTutor} \mapsto \texttt{SELECT Name, Tutor FROM Student}$$

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—so-called *rewriting*—and second, the mappings are used to transform the rewriting into an SQL query that is then evaluated using the RDBMS 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.

For example, consider the query $Q = Q(x) \leftarrow \text{Teacher}(x)$ posed over $\mathcal{O}$. A piece of relevant information encoded in $\mathcal{O}$ for answering Q is, for instance, that all professors are teachers, and 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.

As mentioned in the introduction, 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. Even though 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 unions of conjunctive queries when rewriting queries w.r.t. OWL 2 QL ontologies. An advantage of using RQR is that in addition to OWL 2 QL, it can handle various more expressive language fragments, including most of the fragment captured by the OWL 2 EL profile. For ontologies expressed in such fragments, however, the rewriting might be a datalog query, and thus require the use of a deductive database system for its evaluation.

Given the inputs Q and $\mathcal{O}$ as above, either algorithm will produce the following rewriting $Q_\mathcal{O}$:

$$Q(x) \leftarrow \text{Teacher}(x) \tag{4}$$
$$Q(x) \leftarrow \text{teaches}(x, y) \tag{5}$$
$$Q(x) \leftarrow \text{Professor}(x) \tag{6}$$
$$Q(x) \leftarrow \text{hasTutor}(y, x) \tag{7}$$

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 (4) and (5) can be discarded. It is easy to see that, as a result,

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

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

3 Applying 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 $\mathcal{O}$ [1]. 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 $Q = Q(x_0) \leftarrow R_n(x_0, x_1) \wedge ... \wedge R(x_{m-1}, x_m)$ posed over $\mathcal{O}$. The rewriting $Q_{\mathcal{O}}$ of Q w.r.t. $\mathcal{O}$ will contain n^{m-1} 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. 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.

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 $Q_1\sigma \subseteq Q_2$. For instance, given $\sigma = \{z \mapsto x, w \mapsto y\}$, the query $Q(z) \leftarrow \text{isTutorOf}(z, w)$ subsumes $Q(x) \leftarrow \text{isTutorOf}(x, y) \wedge \text{Undergraduate}(y)$. 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 implemented RQR in a system that we call REQUIEM. As mentioned in the introduction, we recently conducted an empirical evaluation [6] of REQUIEM. We also implemented a version of RQR that reduces the size of the rewritings using the query subsumption check described previously; we call this implementation REQUIEM-SC.

Our test set mainly consisted 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

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

Fig. 1. OWL 2 QL Evaluation Results

		Queries		Symbols		Time	
O	Q	R	RSC	R	RSC	R	RSC
V	1	15	15	454	454	16	31
	2	10	10	762	762	16	47
	3	72	72	6,525	6,525	31	62
	4	185	185	11,911	11,911	62	141
	5	30	30	3,761	3,761	31	63
S	1	6	6	158	158	15	16
	2	160	2	11,422	154	78	109
	3	480	4	56,536	488	438	1,062
	4	960	4	111,092	468	828	2,171
	5	2,880	8	466,896	1,320	9,829	34,681
U	1	2	2	118	118	16	31
	2	148	1	10,378	68	47	93
	3	224	4	29,376	516	94	203
	4	1,628	2	113,270	124	797	4,093
	5	2,960	10	279,266	932	3,234	15,262
A	1	402	27	21,933	901	94	265
	2	103	50	7,122	3,783	47	78
	3	104	104	10,108	10,108	78	93
	4	492	224	33,454	16,069	156	422
	5	624	624	70,320	70,320	328	1,031

data access [8]. `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 [5]. The number of classes (C), properties (P) and axioms (A) are as follows:

	V	S	U	A
C	194	18	34	74
P	10	12	26	5
A	222	51	127	137

The results of our evaluation indicated that, while REQUIEM produced significantly smaller rewritings than the CGLLR algorithm in most cases, with the query subsumption optimisation the two techniques produced almost identical rewritings. Unfortunately, our evaluation showed that, even when using REQUIEM-SC, the rewritings can be extremely large.

Figure 1 shows the results of our evaluation for REQUIEM and REQUIEM-SC. 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, and the column 'Time' shows the number of milliseconds taken to compute the rewritings.

As can be seen, our results suggest that typical queries over realistic ontologies can lead to REQUIEM producing very large rewritings. In the fifth queries over `S` and `U`, for instance, REQUIEM produced nearly 3,000 queries. If we turn

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

our attention to REQUIEM-SC, even though query subsumption significantly reduced the size of some rewritings, REQUIEM-SC still produced rewritings containing hundreds of queries (e.g., last three queries over `A`).

Once the rewriting has been computed and reduced using query subsumption, 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

As mentioned in Section 2, an advantage of using RQR is that it can handle fragments of OWL 2 that go beyond the QL profile. In fact, the algorithm supports ontologies expressed in $\mathcal{ELHIO}^{\neg}$—a description logic that captures most of the EL profile of OWL 2.

Unlike CGLLR, RQR can handle axioms containing conjunction on the left-hand side, e.g.,

$$\text{Student} \sqcap \exists \text{hasSupervisor} \sqsubseteq \text{GraduateStudent};$$

qualified existential restrictions on the left-hand side, e.g.,

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

and a limited use of nominals—classes with only one instance—e.g.,

$$\text{OxfordStudent} \sqsubseteq \exists \text{studiesAt.}\{\mathsf{OxfordUniversity}\}.$$

When dealing with OWL 2 EL ontologies, however, RQR is no longer guaranteed to produce a union of conjunctive queries; instead, the rewriting $Q_{\mathcal{T}}$ might be a datalog query—that is, a set of *Horn clauses* [3]. 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 $\mathcal{T}$ containing the following axiom:

$$\exists \text{hasClassmate.Student} \sqsubseteq \text{Student} \tag{8}$$

Axiom (8) states that anybody who has a classmate who is a student is also a student.

On input Q and $\mathcal{T}$, RQR will produce the following rewriting $Q_{\mathcal{T}}$:

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

$$\text{Student}(x) \leftarrow \text{hasClassmate}(x, y) \wedge \text{Student}(y) \tag{10}$$

Intuitively, the reason RQR produced a datalog query is that clause (10) is recursive; therefore, $Q_{\mathcal{T}}$ cannot be transformed into a union of conjunctive queries.

There is a straightforward a posteriori optimization that can be applied to reduce the size of the datalog query $Q_{\mathcal{T}}$. Such an optimization is based on the notion of the so-called *dependency graph*. (Note that the optimizations discussed in Section 3 can also be applied to reduce the size of $Q_{\mathcal{T}}$.)

Given a set of clauses LP, the *dependency graph* $\mathcal{G}(LP)$ of LP is constructed as follows: every predicate P occurring in LP is a node in $\mathcal{G}(LP)$ and there is an edge from a predicate P_1 to a predicate P_2 if there is a clause $C \in LP$ such that P_1 occurs in the head of C and P_2 occurs in the body of C or viceversa. Intuitively, we can use the dependency graph $\mathcal{G}(LP)$ of a given set of clauses LP to identify the set of clauses $LP' \subseteq LP$ that are "relevant" in order to compute the extension of a given predicate P. For example, consider the following set of clauses LP:

$$\text{Teacher}(x) \leftarrow \text{Professor}(x) \tag{11}$$
$$\text{Student}(x) \leftarrow \text{hasTutor}(y, x) \tag{12}$$

Clearly, since neither Student, nor hasTutor are reachable from the predicate Teacher in $\mathcal{G}(LP)$, we can immediately deduce that clause (12) is not relevant for the computation of the extension of Teacher.

By computing the set of predicates that are reachable from the head predicate of the query Q in $Q_{\mathcal{T}}$, one can immediately identify the set of clauses that are "relevant" for Q and discard the set of clauses that correspond to the unreachable parts of the graph.

4.1 Using Databases for OWL EL

As shown in the last section, when rewriting queries w.r.t. OWL 2 EL ontologies, RQR might produce a cyclic or recursive datalog query that cannot be transformed into a union of conjunctive queries. However, examination of a large corpus of ontologies [4] suggests that in many realistic cases ontologies do not contain (or imply) cyclic axiom such as (8). In these cases, the datalog query obtained by RQR can still be transformed into a union of conjunctive queries.

Suppose, for example, that RQR computed the following rewriting $Q_{\mathcal{T}}$:

$$Q(x) \leftarrow \text{Teacher}(x) \tag{13}$$
$$\text{Teacher}(x) \leftarrow \text{teaches}(x, y) \wedge \text{Student}(y) \tag{14}$$

It is not difficult to see that by *unfolding* (14) into (13), such a datalog program can be transformed into the following union of conjunctive queries:

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

Fig. 2. OWL 2 EL Evaluation Results

O	Q	Clauses	Symbols	Time
NCI	1	35	1,235	133,141
	2	171	4,975	3,110
	3	220	14,508	213,706
	4	9,109	936,485	253,659
	5	176	12,171	334,522
U	1	24	3,532	78
	2	11	1,005	328
	3	37	4,577	188
	4	36	4,800	1,922
	5	1	158	188

We have conducted a preliminary empirical evaluation of REQUIEM in which we consider real acyclic OWL 2 EL ontologies—that is, ontologies that do not contain (or imply) axioms of the form of (8). Our test data includes the well-known ontology `NCI`[7] and `U`—an $\mathcal{ELHI}$ version of the university bechmark ontology[8] developed at Lehigh University. The number of classes (C), properties (P) and axioms (A) are as follows:

	NCI	U
C	27,652	43
P	70	25
A	395,124	171

Figure 2 shows preliminary results of our evaluation using REQUIEM-SC enhanced with the dependency graph optimization and the greedy unfolding discussed in this section. As before, 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, and the column 'Time' shows the number of milliseconds taken to compute the rewritings.

In all cases, the resulting rewriting was a union of conjunctive queries. As can be seen, the running time was significant in the case of `NCI`, which is mostly due to its size. More importantly, and similarly to our evaluation using OWL 2 QL ontologies, REQUIEM-SC produced relatively large rewritings in some cases (e.g., fourth query over `NCI`).

The preliminary results presented in this section are encouraging since they suggest that it is possible to use an off-the-shelf RDBMS for query answering over OWL 2 EL ontologies in many realistic scenarios. However, the average size of the rewritings obtained suggests that a further optimization based on the mappings as described in Section 3 might be critical in order to successfully use existing database technology in this setting.

[7] `http://www.mindswap.org/2003/CancerOntology/`
[8] `http://swat.cse.lehigh.edu/projects/lubm/`

5 Future Work

We plan to extend our empirical evaluation of REQUIEM w.r.t. OWL EL to include cyclic ontologies, and to analyze the type of datalog queries that arise in such cases in order to determine the most promising strategy for further optimization and evaluation.

Additionally, we plan to implement an OWL 2 QL ontology-based data access system using REQUIEM enhanced with various optimizations. Based on our results, we expect the system to be useful for answering may realistic queries even over OWL 2 EL ontologies. 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 the 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 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. Finally, we plan to extend the system to support all of OWL 2 QL, which mainly involves adding support for datatypes.

References

1. 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.
2. 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.
3. M. Fitting. *First-order logic and automated theorem proving (2nd ed.)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.
4. T. Gardiner, D. Tsarkov, and I. Horrocks. Framework for an automated comparison of description logic reasoners. volume 4273, pages 654–667, 2006.
5. 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.
6. H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient Query Answering for OWL 2. In *In Proc. ISWC 2009, Chantilly, Virginia, October 2009*, 2009. To appear.
7. H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable Query Answering and Rewriting under Description Logic Constraints. *J. of Applied Logic*, 2009. To appear. `http://web.comlab.ox.ac.uk/people/publications/date/Hector.Perez-Urbina.html`.
8. 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.