Rewriting Minimisations for Efficient Ontology-Based Query Answering

Tassos Venetis
Athens University of Economics and Business
76 Patision Street
Athens, Greece
avenet@aueb.gr

Giorgos Stoilos
Athens University of Economics and Business
76 Patision Street
Athens, Greece
gstoil@aueb.gr

Vasilis Vassalos
Athens University of Economics and Business
76 Patision Street
Athens, Greece
vassalos@aueb.gr

ABSTRACT

Computing a (Union of Conjunctive Queries - UCQ) rewriting $\mathcal{R}$ for an input query and ontology and evaluating it over the given dataset is a prominent approach to query answering over ontologies. However, $\mathcal{R}$ can be large and complex in structure hence additional techniques, like query subsumption and data constrains, need to be employed in order to minimise $\mathcal{R}$ and lead to an efficient evaluation. Although sound in theory, how to efficiently and effectively implement many of these techniques in practice could be challenging. For example, many systems do not implement query subsumption. In the current paper we present several practical techniques for UCQ rewriting minimisation. First, we present an optimised algorithm for eliminating redundant (w.r.t. subsumption) queries as well as a novel framework for rewriting minimisation using data constraints. Second, we show how these techniques can also be used to speed up the computation of $\mathcal{R}$ in the first place. Third, we integrated all our techniques in our query rewriting system IQAROS and conducted an extensive experimental evaluation using many artificial as well as challenging real-world ontologies obtaining encouraging results as, in the vast majority of cases, our system is more efficient compared to the two most popular state-of-the-art systems.

1. INTRODUCTION

Query rewriting is a prominent approach to answering queries over data described using ontologies [17]. In such a setting the input ontology $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ and query $\mathcal{Q}$ are transformed into a union of conjunctive queries ($\mathcal{R}$), called *UCQ rewriting*, which captures all certain answers of $\mathcal{Q}$ over any possible dataset $\mathcal{A}$—that is, $\mathcal{R} \cup \mathcal{A}$ returns the same answers to $\mathcal{Q}$ as $\mathcal{T} \cup \mathcal{A}$. The motivation behind this approach is that since a rewriting is a UCQ one can use a relational database system to evaluate $\mathcal{R}$ and compute the answers of $\mathcal{Q}$ over $\mathcal{T} \cup \mathcal{A}$.

The last decade a spate of algorithms and systems for computing UCQ rewritings have been presented in the literature [4, 14, 15, 21, 20, 12, 22]. Although one would expect that the main difficulty in this approach is how to efficiently compute a rewriting, it has been stressed that how to actually evaluate it using the underlying data management system can also pose significant challenges [9, 19, 20, 16] as the UCQ rewriting can contain an exponential number of queries some of which might be much larger than the input query (they can contain a large number of conjuncts).

To solve the above issue many techniques for minimising and simplifying the computed rewriting have been proposed [14, 19, 21, 20, 16, 11, 15]. Perhaps the first suggested technique was query subsumption—that is, removing a query from $\mathcal{R}$ if some other exists that contains/subsumes it [14, 1]. Another powerful technique employed in all systems in [19, 20, 16, 11] is to assume that the dataset satisfies certain type of dependencies, e.g., assume that whenever we have $A \sqsubseteq B \in \mathcal{T}$ and assertion $A(o) \in \mathcal{A}$ we also have $B(o) \in \mathcal{A}$. If this is the case and $\mathcal{R}$ contains queries $\mathcal{Q}_1 = B(x) \to Q(x)$ and $\mathcal{Q}_2 = A(x) \to Q(x)$, then $\mathcal{Q}_2$ can be discarded, since due to the assumption all instances of A would be retrieved by $\mathcal{Q}_1$. Another approach to minimise $\mathcal{R}$ is to remove queries that contain atoms that have no instances [15]. Clearly, these queries would yield an empty answer if evaluated over the dataset.

Although sound and possibly very effective, how to implement and exploit many of these techniques in practice poses many challenges. For example, to the best of our knowledge, many existing systems do not implement the query subsumption technique since checking query containment can be a very time-consuming process. Moreover, assuming that the data satisfy certain closure conditions can be a very strong assumption for certain applications especially for (Semantic Web) applications dealing with semi-structured, Big, or streaming data, but even if they do not, how to discover them by data analysis is not trivial and is tantamount to database dependency induction.

In the current paper we present efficient practical algorithms and novel frameworks for effectively minimising a computed rewriting. More precisely, first, we present several non-trivial heuristic optimisations for speeding up the execution of the subsumption-based redundancy elimination algorithm. Interestingly, our evaluation showed that these refinements can improve the execution of the standard implementation in most cases by several times or even up to one order of magnitude. Second, we argue and verify experimentally that the idea about the emptiness of atoms

© 2017 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

can be extended to emptiness of joins of atoms (e.g., emptiness of expressions $A(x) \wedge R(x, y)$) and this can reduce the size of the rewriting quite significantly without significant pre-processing penalties in discovering such emptiness information. Third, we show how the inference (data saturation) capabilities of scalable and mature OWL 2 RL reasoners, like GraphDB and RDFox, can be used to further minimise the computed UCQ rewriting. Fourth, besides techniques for minimising the computed UCQ rewriting, we show how many of them can be used to also speed up its computation in the first place by pruning the search space of the query rewriting system.

We have implemented all of the above techniques into our query rewriting system IQAROS [23] which we connected with both RDBMS systems (PostgreSQL and MySQL) as well as the OWL 2 RL system RDFox in order to apply our OWL 2 RL-based rewriting minimisation technique mentioned above. To evaluate their effectiveness we conducted an extensive experimental evaluation using two well-known benchmarks like UOMB, LUBM, the newly proposed NPD benchmark [10] as well as many real-world ontologies like Fly Anatomy, Reactome and Uniprot and compared against two available state-of-the-art query rewriting systems, namely Ontop [20] and Stardog [15]. Our results show that, in the vast majority of cases, our approach significantly outperforms both of these systems as it computes much smaller rewritings.

Full proofs of our results and the implementation can be found on-line.[1]

2. PRELIMINARIES

2.1 OWL 2 DL and OWL 2 RL

We assume familiarity with the OWL 2 DL ontology language and its profiles like OWL 2 RL.[2]

For brevity, throughout the paper we will use the Description Logic (DL) notation to write down OWL 2 DL axioms. The reader is referred to [3] for an overview of the relationship between DLs and OWL. As usual in the literature, we distinguish between the *schema* of an ontology, called *TBox* and denoted by $\mathcal{T}$, and the *data* called *ABox* and denoted by $\mathcal{A}$. For $\mathcal{L}$ a fragment of OWL 2 DL we use the notation $\mathcal{T}|_{\mathcal{L}}$ to denote all $\mathcal{L}$-axioms of $\mathcal{T}$, i.e., those axioms of $\mathcal{T}$ that are expressed in the fragment $\mathcal{L}$. We also use $\mathsf{Sig}(\mathcal{T})$ to denote all atomic concepts and roles that appear in $\mathcal{T}$.

OWL 2 RL is a prominent profile of OWL 2 DL. Each OWL 2 RL TBox can be translated into an equivalent datalog program using simple syntactic transformations. For example, an axiom of the form $A \sqcap B \sqsubseteq C$ corresponds to the datalog rule $A(x) \wedge B(x) \rightarrow C(x)$ and the axiom $\exists R.A \sqsubseteq B$ to $R(x, y) \wedge A(y) \rightarrow B(x)$. In contrast $A \sqsubseteq B \sqcup C$ and $A \sqsubseteq \exists R.\top$ are not OWL 2 RL axioms since they correspond to the (non-datalog) clauses $A(x) \rightarrow B(x) \vee C(x)$ and $A(x) \rightarrow R(x, f(x))$ for f a skolem function.[3] Consequently, query answering in OWL 2 RL can be realised by mature datalog reasoning techniques and optimisations. Several scalable and highly efficient OWL 2 RL systems have been implemented in the past, like GraphDB (formerly OWLim), Oracle's Semantic Graph, and RDFox.

2.2 Conjunctive Queries

We use the standard notions of function-free concept atom and role atom, variable and substitution from First-Order Logic. A *conjunctive query* (or simply query) (CQ) $\mathcal{Q}$ is an expression of the form $\alpha_1 \wedge \ldots \wedge \alpha_m \rightarrow Q(\vec{x})$ where $\vec{x} = (x_1, \ldots, x_n)$ is a tuple of variables called *distinguished* (or *answer*) and each α_i is an atom called *body atom*. Every variable in $\vec{x}$ appears in at least one body atom while n is called the *arity* of $\mathcal{Q}$; all other variables of $\mathcal{Q}$ are called *existential*. For σ a substitution, $\mathcal{Q}\sigma$ denotes the query $\alpha_1\sigma \wedge \ldots \wedge \alpha_m\sigma \rightarrow Q(\vec{x})\sigma$, where $\alpha_i\sigma$ is the result of applying σ to α_i. For a tuple of constants $\vec{a}$ with the same arity as $\mathcal{Q}$, we use $\mathcal{Q}(\vec{a})$ to denote the CQ obtained from $\mathcal{Q}$ by replacing all its answer variables with $\vec{a}$. A union of conjunctive queries (UCQ) is a set of queries $\{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_\ell\}$ with the same arity.

For a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, a tuple of constants $\vec{a}$ is a *certain answer* of a conjunctive query $\mathcal{Q} = \alpha_1 \wedge \ldots \wedge \alpha_m \rightarrow Q(\vec{x})$ over $\mathcal{T} \cup \mathcal{A}$ if $\mathcal{T} \cup \mathcal{A} \models \mathcal{Q}(\vec{a})$. We denote with $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$ all the certain answers of $\mathcal{Q}$ over $\mathcal{T} \cup \mathcal{A}$. Moreover, by a slight abuse of notation, for $\mathcal{R}$ a UCQ we write $\mathsf{cert}(\mathcal{R}, \mathcal{T} \cup \mathcal{A})$ to denote all $\vec{a}$ such that $\mathcal{T} \cup \mathcal{A} \models \mathcal{Q}_i(\vec{a})$ for some $\mathcal{Q}_i \in \mathcal{R}$.

2.3 Query Rewriting

Query rewriting is a widely-used technique for query answering over ontologies. Intuitively, a *rewriting* for $\mathcal{Q}, \mathcal{T}$ is special structure (in many cases a UCQ) that captures all the information from $\mathcal{T}$ relevant for answering $\mathcal{Q}$ over $\mathcal{T}$ and *any* ABox $\mathcal{A}$ [4, 14].

DEFINITION 2.1. *Let $\mathcal{Q}$ be a CQ and let $\mathcal{T}$ be an OWL 2 DL-TBox. A* UCQ rewriting *(or simply* rewriting*) $\mathcal{R}$ for $\mathcal{Q}, \mathcal{T}$ is a UCQ with the same arity as $\mathcal{Q}$ such that for each ABox $\mathcal{A}$ using only predicates from $\mathcal{T}$ we have*

$$\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) = \mathsf{cert}(\mathcal{R}, \mathcal{A})$$

EXAMPLE 2.2. *Consider the following TBox $\mathcal{T}$ and query $\mathcal{Q}$:*

$$\begin{aligned}
\mathcal{T} &= \{\mathsf{Bolt} \sqsubseteq \exists \mathsf{isPartOf}.\mathsf{Engine}, \\
&\quad\ \mathsf{Engine} \sqsubseteq \exists \mathsf{hasPart}.\mathsf{Piston}, \\
&\quad\ \mathsf{isPartOf}^{-} \sqsubseteq \mathsf{hasPart}\} \\
\mathcal{Q} &= \mathsf{isPartOf}(x, y) \wedge \mathsf{hasPart}(y, z) \wedge \mathsf{Piston}(z) \rightarrow Q(x)
\end{aligned}$$

A rewriting for $\mathcal{Q}, \mathcal{T}$ computed using systems like Requiem, Rapid, IQAROS, and more consists of $\mathcal{R} = \{\mathcal{Q}, \mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4\}$ where $\mathcal{Q}_1$–$\mathcal{Q}_4$ are as defined next:

$$\begin{aligned}
\mathcal{Q}_1 &= \mathsf{isPartOf}(x, y) \wedge \mathsf{Engine}(y) \rightarrow Q(x) \\
\mathcal{Q}_2 &= \mathsf{isPartOf}(x, y) \wedge \mathsf{Piston}(x) \rightarrow Q(x) \\
\mathcal{Q}_3 &= \mathsf{Bolt}(x) \wedge \mathsf{Piston}(x) \rightarrow Q(x) \\
\mathcal{Q}_4 &= \mathsf{Bolt}(x) \rightarrow Q(x)
\end{aligned}$$

Note that $\mathcal{R}$ captures any *answer over $\mathcal{T}$ given* any *ABox. For example, for $\mathcal{A}_1 = \{\mathsf{Bolt}(b)\}$ we have $\mathcal{T} \cup \mathcal{A}_1 \models \mathcal{Q}(b)$ and for $\mathcal{Q}_4 \in \mathcal{R}$ we have $\mathcal{A}_1 \models \mathcal{Q}_4(b)$.* ♢

In case the dataset is stored into a relational database to compute the answers one needs to translate the UCQ rewriting into an SQL query by (properly) mapping the atoms of

[1]https://github.com/iqaros/
[2]http://www.w3.org/TR/owl2-overview/
[3]As we mentioned, the check is purely syntactic discarding the rest of the TBox axioms. For example, we do not check if atom B is possibly unsatisfiable in which case one could argue that $A \sqsubseteq B \sqcup C$ is *semantically* equivalent to $A(x) \rightarrow C(x)$.

the query to database tables/columns. This connection is dictated by a set of *mappings* like the following one, which populates the concept GradSt with the proper ids from the table *Persons*:

$$\begin{aligned}&\text{select } P.id \text{ from } Persons \text{ as } P\\ &\text{where } P.type = \text{‘GradStudent’} \rightsquigarrow \mathsf{GradSt}(P.id)\end{aligned}$$

During the last decade a wealth of algorithms and systems for computing UCQ rewritings have been developed. To name a few we note Ontop [20], Mastro [16], Prexto [21], IQAROS [23], Requiem [14], Rapid [22], Nyaya [7], and Kyrie2 [12].

3. MINIMISING THE SIZE OF A REWRITING

Minimising the size of a computed rewriting $\mathcal{R}$ and possibly simplifying its structure are clearly the key objectives for speeding up the subsequent evaluation of $\mathcal{R}$ over the given dataset. In the current section we present several efficient and practically effective approaches for rewriting minimisation. Some of these techniques are (non-trivial) refinements and improvements of previously proposed techniques, like query subsumption studied in Section 3.1, however, others are highly novel, like that presented in Section 3.3.

3.1 Efficient Query Subsumption

The first method for rewriting minimisation appeared in [14] and was based on the well-established notion of query containment from database theory [1] defined next.

Definition 3.1. *Let $\mathcal{Q}_1$, $\mathcal{Q}_2$ be CQs. We say that $\mathcal{Q}_1$* subsumes *$\mathcal{Q}_2$ if there exists a mapping σ from the terms of $\mathcal{Q}_1$ to those of $\mathcal{Q}_2$ such that every atom in $\mathcal{Q}_1\sigma$ appears in $\mathcal{Q}_2$. Moreover, let $\mathcal{R}$ be a set of queries. We say that some CQ $\mathcal{Q} \in \mathcal{R}$ is* redundant *in $\mathcal{R}$ if $\mathcal{Q}$ is subsumed by some other query in $\mathcal{R}$.*

In Example 2.2, query $\mathcal{Q}_3$ is redundant in $\mathcal{R}$ since it is subsumed by $\mathcal{Q}_4$. Clearly, for any ABox $\mathcal{A}$ such that $\mathcal{A} \models \mathcal{Q}_3(a)$ we will also have $\mathcal{A} \models \mathcal{Q}_4(a)$, hence $\mathcal{Q}_3$ can be removed from $\mathcal{R}$.

As shown by several evaluations [14, 23, 8], discarding subsumed queries can significantly decrease the size of a rewriting. Surprisingly, however, many systems like Nyaya [7] and Clipper [6] do not apply it. The reason is that to identify and remove the redundant queries one has to perform a nested for-loop over the (potentially very large) input rewriting $\mathcal{R}$ and check for query subsumption which, for conjunctive queries, is an NP-complete problem. In more detail, one needs to pick two different queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ from $\mathcal{R}$ and construct (guess) some mapping σ such that every atom in $\mathcal{Q}_1\sigma$ appears in $\mathcal{Q}_2$ (or vice versa). If this is the case, then $\mathcal{Q}_1$ (resp. $\mathcal{Q}_2$) is marked for removal. This *basic* algorithm was implemented in the Requiem system and both Rapid and IQAROS used it.

This basic algorithm, however, can be significantly optimised by exploiting the following intuitive heuristic. We have observed that, in the vast majority of cases, if $\mathcal{Q}_1$ subsumes $\mathcal{Q}_2$ then $\mathcal{Q}_1$ usually has fewer atoms than $\mathcal{Q}_2$ (although this is not necessarily the case in theory). Hence, before iterating over $\mathcal{R}$ it would be beneficial to order the queries according to an increasing number of body atoms. Then, after the first few iterations of the outer for-loop it is very likely that we have discovered all (or at least most) redundant queries in $\mathcal{R}$.

Algorithm 1 querySubsumption($\mathcal{R}$)

```
1:  $\mathcal{R}_<$ := orderByNumberOfAtoms($\mathcal{R}$)          //Additional line
2:  SubsumedCQs := $\emptyset$, NonSubsumed := $\emptyset$
3:  for all $\mathcal{Q} \in \mathcal{R}_<$ do
4:      if $\mathcal{Q} \notin$ SubsumedCQs then                 //Additional line
5:          for all $\mathcal{Q}' \in \mathcal{R}_<$ s.t. $\mathcal{Q}' \neq \mathcal{Q}$ and $\mathcal{Q}' \notin$ SubsumedCQs do
6:              if $\mathcal{Q}$ subsumes $\mathcal{Q}'$ then
7:                  add $\mathcal{Q}'$ to SubsumedCQs
8:              else if $\mathcal{Q}'$ subsumes $\mathcal{Q}$ then
9:                  add $\mathcal{Q}$ to SubsumedCQs
10:                 break
11:             end if
12:         end for
13:         if $\mathcal{Q} \notin$ SubsumedCQs then
14:             add $\mathcal{Q}$ to NonSubsumed
15:         end if
16:     end if
17: end for
18: return NonSubsumed
```

A second improvement stems from the transitivity of the "subsumes" relation. More precisely, if $\mathcal{Q}_1$ subsumes $\mathcal{Q}_2$ and $\mathcal{Q}_2$ subsumes $\mathcal{Q}_3$, then $\mathcal{Q}_1$ also subsumes $\mathcal{Q}_3$. Hence, when a query $\mathcal{Q}_2$ in $\mathcal{R}$ is redundant, besides marking it for removal we also add it to a set called SubsumedCQs. During the iterations, we do not check if redundant queries subsume other queries in $\mathcal{R}$. As explained this does not harm completeness of the algorithm due to transitivity.

The above observations lead to the optimised subsumption algorithm depicted in Algorithm 1. Compared to the basic algorithm this one differs in Lines 1 and 4. As discussed, first, the algorithm orders the queries and, second, it stores in a separate set (SubsumedCQs) those queries that have already been marked as subsumed with the intention to skip them in the for-loops.

Proposition 3.2. *Given a set of queries $\mathcal{R}$, Algorithm 1 returns a set $\mathcal{R}'$ that contains no redundant queries and for any ABox $\mathcal{A}$ we have $\mathsf{cert}(\mathcal{R}, \mathcal{A}) = \mathsf{cert}(\mathcal{R}', \mathcal{A})$.*

Note that, by replacing queries with arbitrary (even disjunctive) clauses, Algorithm 1 becomes applicable even to first-order logic.

3.2 Emptiness of Queries

Another conceptually appealing way to reduce the size of the rewriting is by discarding those queries that would have an empty answer set if evaluated over the data. In [15] the authors noticed that even if the ontology has a large number of concepts and roles, usually very few of them participate in assertions in the dataset. Queries that contain such concepts and roles can clearly be discarded. In addition to that, we also argue that rewritings contain many queries with *conjunctions* of atoms that have no matching assertions in the dataset.

Example 3.3. *Consider the following TBox and query about students and courses:*

$$\begin{aligned}\mathcal{T} &= \{\mathsf{GradSt} \sqsubseteq \mathsf{St},\\ &\quad\ \mathsf{takesGrC} \sqsubseteq \mathsf{takesC},\\ &\quad\ \mathsf{takesUnderGrC} \sqsubseteq \mathsf{takesC}\}\\ \mathcal{Q} &= \mathsf{St}(x) \wedge \mathsf{takesC}(x, y) \rightarrow Q(x)\end{aligned}$$

A rewriting for $\mathcal{Q}, \mathcal{T}$ consists of the set $\mathcal{R} = \{\mathcal{Q}, \mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4, \mathcal{Q}_5\}$ where $\mathcal{Q}_1$–$\mathcal{Q}_5$ are as follows:

$$\begin{array}{lcl} \mathcal{Q}_1 & = & \mathsf{St}(x) \wedge \mathsf{takesGrC}(x,y) \to Q(x) \\ \mathcal{Q}_2 & = & \mathsf{St}(x) \wedge \mathsf{takesUnderGrC}(x,y) \to Q(x) \\ \mathcal{Q}_3 & = & \mathsf{GradSt}(x) \wedge \mathsf{takesC}(x,y) \to Q(x) \\ \mathcal{Q}_4 & = & \mathsf{GradSt}(x) \wedge \mathsf{takesGrC}(x,y) \to Q(x) \\ \mathcal{Q}_5 & = & \mathsf{GradSt}(x) \wedge \mathsf{takesUnderGrC}(x,y) \to Q(x) \end{array}$$

As noticed in [15] in a real-world scenario it is expected that every student is either classified as a graduate or an undergraduate one, hence, the ABox will not explicitly contain any assertions for the student concept, i.e., assertions of the form $\mathsf{St}(a)$. *If this is the case, then queries $\mathcal{Q}, \mathcal{Q}_1, \mathcal{Q}_2$ are* not *going to return any tuples when evaluated over $\mathcal{A}$; the same holds for the role* takesC *and query $\mathcal{Q}_3$. Consequently, all these queries can be removed from $\mathcal{R}$.*

By a closer look to the above TBox we can see that it is reasonable to extend this heuristic to conjunctions of atoms. For example, in a real-world setting it is also likely that graduate students do not take undergraduate courses. If this is the case, then the join between the atoms $\mathsf{GradSt}(x)$ *and* $\mathsf{takesUnderGrC}(x,y)$ *will also be empty and as a consequence query $\mathcal{Q}_5$ is not going to return any values. All in all, only query $\mathcal{Q}_4$ is going to return some answers.* $\diamondsuit$

To implement the above approach one should at a pre-processing step compute the (conjunctions of) atoms that have no instances in the dataset. The system we report in the evaluation section uses the approach formalised next to prune queries; other combinations could perhaps be possible.

Definition 3.4. *For a concept C and roles R, S, S^- (S^- denotes the InverseOf S) let $\mathcal{Q}^C, \mathcal{Q}^{C,S}$, and $\mathcal{Q}^{R,S}$, denote the queries $C(x) \to Q(x), C(x) \wedge R(x,y) \to Q(x,y)$ and $R(x,y) \wedge S(x,y) \to Q(x,y)$, respectively. For some TBox $\mathcal{T}$ and ABox $\mathcal{A}$ let the following set of queries:*

$$\begin{array}{lcl} \mathsf{Empty}^{\mathcal{T}}_{\mathcal{A}} & := & \{\mathcal{Q}^C \mid C \in \mathsf{Sig}(\mathcal{T}), \mathsf{cert}(\mathcal{Q}^C, \mathcal{T} \cup \mathcal{A}) = \emptyset\} \cup \\ & & \{\mathcal{Q}^{C,S} \mid \{C,S\} \subseteq \mathsf{Sig}(\mathcal{T}), \mathsf{cert}(\mathcal{Q}^{C,S}, \mathcal{T} \cup \mathcal{A}) = \emptyset\} \cup \\ & & \{\mathcal{Q}^{C,S^-} \mid \{C,S\} \subseteq \mathsf{Sig}(\mathcal{T}), \mathsf{cert}(\mathcal{Q}^{C,S^-}, \mathcal{T} \cup \mathcal{A}) = \emptyset\} \cup \\ & & \{\mathcal{Q}^{R,S} \mid \{R,S\} \subseteq \mathsf{Sig}(\mathcal{T}), \mathsf{cert}(\mathcal{Q}^{R,S}, \mathcal{T} \cup \mathcal{A}) = \emptyset\} \cup \\ & & \{\mathcal{Q}^{R,S^-} \mid \{R,S\} \subseteq \mathsf{Sig}(\mathcal{T}), \mathsf{cert}(\mathcal{Q}^{R,S^-}, \mathcal{T} \cup \mathcal{A}) = \emptyset\} \end{array}$$

Proposition 3.5. *Let $\mathcal{R}$ be a rewriting for some CQ $\mathcal{Q}$ and TBox $\mathcal{T}$, and let $\mathcal{A}$ be some ABox. Let $\mathcal{R}^-$ be the subset of $\mathcal{R}$ after removing all $\mathcal{Q}' \in \mathcal{R}$ such that some $\mathcal{Q}'' \in \mathsf{Empty}^{\mathcal{T}}_{\mathcal{A}}$ and substitution σ exist such that all body atoms in $\mathcal{Q}''\sigma$ appear in $\mathcal{Q}'$. Then, $\mathsf{cert}(\mathcal{R}, \mathcal{T} \cup \mathcal{A}) = \mathsf{cert}(\mathcal{R}^-, \mathcal{T} \cup \mathcal{A})$.*

As we will see in the evaluation section, rewritings for real-world ontologies do contain several queries that are empty due to *conjunctions* of atoms. Moreovoer, computing $\mathsf{Empty}^{\mathcal{T}}_{\mathcal{A}}$ at pre-processing and $\mathcal{R}^-$ at query time are not particularly time consuming although the number of queries of the form $\mathcal{Q}^{C,R}$ and $\mathcal{Q}^{R,S}$ is quadratic in the size of $\mathcal{T}$ and constructing $\mathcal{R}^-$ requires checking a limited form of query subsumption.

3.3 OWL 2 RL Systems and Data Dependencies

A powerful idea for query optimisation is to exploit any (possibly existing) data dependencies/constraints specified in the schema. This idea has already been proposed and exploited in many areas like XML query minimisation and optimisation using DTDs [18, 2, 24] and has also been proposed in ontology-based query rewriting [19, 21, 16, 11].

Consider, for example, the axiom $\mathsf{Manager} \sqsubseteq \mathsf{Person}$ and assume that as a result of a database trigger, every manager is also explicitly recorded to be a person in the database. In this case, for every assertion $\mathsf{Manager}(a)$ we also explicitly have $\mathsf{Person}(a)$ in the dataset and the aforementioned axiom actually behaves like a *dependency/constraint* [1]. Consider now query $\mathcal{Q} = \mathsf{Person}(x) \to Q(x)$. In theory, a rewriting for $\mathcal{Q}$ over the above axiom should contain $\mathcal{Q}$ and $\mathcal{Q}' = \mathsf{Manager}(x) \to Q(x)$, however, due to the data constraint, instances of $\mathsf{Manager}$ would be retrieved by $\mathcal{Q}$, hence we can actually discard $\mathcal{Q}'$.

This approach works potentially well if the data originate from a relational database where dependencies are encoded in the form of triggers or foreign key constraints, however, in many (Semantic Web) applications that deal with semi-structured, Big, or streaming data, such constraints are unlikely to be satisfied. Interestingly, in Semantic web applications the data are very often stored in triple-stores or OWL 2 RL systems which, given a TBox $\mathcal{T}$ and ABox $\mathcal{A}$, "saturate" $\mathcal{A}$ using some syntactic fragment $\mathcal{T}_{\mathcal{L}}$ of $\mathcal{T}$. In our previous example given $\mathsf{Manager}(a)$ and $\mathsf{Manager} \sqsubseteq \mathsf{Person}$ OWL 2 RL systems like GraphDB and RDFox would compute $\mathcal{A}_s = \mathcal{A} \cup \{\mathsf{Person}(a)\}$. Consequently, after saturation all axioms in $\mathcal{T}_{\mathcal{L}}$ will eventually behave like dependencies.

In the following we show how the saturation performed by OWL 2 RL systems can be exploited to significantly minimise a rewriting. First, in order to abstract away from the specifics of each system we recall the notion of an OWL 2 RL ABox-saturation system [5].

Definition 3.6. *An* ABox-saturation *system* ans *is an algorithm which given some OWL 2 DL-TBox $\mathcal{T}$ and ABox $\mathcal{A}$ computes (using $\mathcal{T} \cup \mathcal{A}$) some ABox $\mathcal{A}_s \supseteq \mathcal{A}$, called* saturation. *Moreover, given some query $\mathcal{Q}$ it returns those answers of $\mathcal{Q}$ over $\mathcal{A}_s$ containing only individuals from $\mathcal{A}$, i.e., it returns* $\mathsf{cert}(\mathcal{Q}, \mathcal{A}_s)|_{\mathsf{ind}(\mathcal{A})}$.

Let $\mathcal{L}$ be a fragment of OWL 2 DL. We say that ans *is* complete for *$\mathcal{L}$ if for every CQ $\mathcal{Q}$, TBox $\mathcal{T}$, and ABox $\mathcal{A}$ the system returns* $\mathsf{cert}(\mathcal{Q}, \mathcal{T}|_{\mathcal{L}} \cup \mathcal{A})$. *In this case $\mathcal{A}_s$ is computed using $\mathcal{T}|_{\mathcal{L}} \cup \mathcal{A}$, hence* $\mathsf{cert}(\mathcal{Q}, \mathcal{T}|_{\mathcal{L}} \cup \mathcal{A}) = \mathsf{cert}(\mathcal{Q}, \mathcal{A}_s)$.

Most OWL 2 RL systems and triple-stores known to us, like GraphDB, Oracle's Semantic Graph, RDFox, and more, can be captured by the above definition.

Example 3.7. *Let* ans *be an ABox-saturation system complete for OWL 2 RL. Let also the TBox $\mathcal{T} = \{A \sqsubseteq \exists R.\top, \exists R.\top \sqsubseteq B\}$ and the ABox $\mathcal{A} = \{A(a), R(c,d)\}$. Since for $\mathcal{L}$ = OWL 2 RL we have $\mathcal{T}|_{\mathcal{L}} = \{\exists R.\top \sqsubseteq B\}$, then* ans *would compute the saturation $\mathcal{A}_s = \mathcal{A} \cup \{B(c)\}$.*

Now consider the query $\mathcal{Q} = B(x) \to Q(x)$ for which $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) = \{a, c\}$ *and consider also the rewriting $\mathcal{R} = \{\mathcal{Q}, \mathcal{Q}_1, \mathcal{Q}_2\}$ for $\mathcal{Q}, \mathcal{T}$, where $\mathcal{Q}_1 = R(x,y) \to Q(x)$ and $\mathcal{Q}_2 = A(x) \to Q(x)$. Clearly, we can evaluate $\mathcal{R}$ over $\mathcal{A}_s$ to compute the answers, however, due to the saturation $\mathcal{A}_s$, query $\mathcal{Q}_1$ can be discarded. Indeed* $\mathsf{cert}(\mathcal{R} \setminus \{\mathcal{Q}_1\}, \mathcal{A}_s) = \{a, c\}$. $\diamondsuit$

In the above example it is easy to see that $\{\exists R.\top \sqsubseteq B\} \cup \mathcal{Q} \models \mathcal{Q}_1$—that is, $\mathcal{Q}_1$ follows by $\mathcal{Q}$ and an axiom that falls in the language for which ans is complete. Consequently, the rewriting need only contain those queries that follow by axioms outside $\mathcal{T}_{\mathcal{L}}$. We now formalise our technique.

Table 1: Comparison of query subsumption algorithm; times are in milliseconds.

$\mathcal{T}$	$\mathcal{Q}$	$\sharp\mathcal{R}_{\mathsf{in}}$	$\sharp\mathcal{R}_{\mathsf{out}}$	t_{basic}	t_{opt}
P5X	$\mathcal{Q}_4$	1953	179	1.35	0.10
	$\mathcal{Q}_5$	9766	718	47.47	2.14
U	$\mathcal{Q}_4$	1628	2	0.50	0.26
	$\mathcal{Q}_5$	2960	10	1.71	0.02
UX	$\mathcal{Q}_3$	1008	12	0.22	<0.01
	$\mathcal{Q}_4$	5000	5	4.22	1.20
	$\mathcal{Q}_5$	8000	25	13.73	0.15
S	$\mathcal{Q}_3$	960	4	0.11	<0.01
	$\mathcal{Q}_4$	2880	8	0.18	<0.01
	$\mathcal{Q}_5$	2880	8	3.55	0.10
AX	$\mathcal{Q}_2$	1656	1431	0.62	0.26
	$\mathcal{Q}_3$	4752	4466	5.71	3.18
	$\mathcal{Q}_4$	4984	3159	4.24	1.89
	$\mathcal{Q}_5$	76032	32921	1826.83	722.43

DEFINITION 3.8. *Let $\mathcal{T}$ be an OWL 2 DL-TBox, let $\mathcal{Q}$ be a CQ, let $\mathcal{R}$ be a UCQ rewriting for $\mathcal{Q}, \mathcal{T}$, and let $\mathcal{L}$ be some fragment of OWL 2 DL. We say that a query $\mathcal{Q}_1 \in \mathcal{R}$ is ($\mathcal{L}$-)*derived *by some other query $\mathcal{Q}_2$ in $\mathcal{R}$ if some minimal w.r.t. set inclusion ($\mathcal{L}$-)TBox $\mathcal{T}' \subseteq \mathcal{T}$ exists such that the following condition holds: for $\mathcal{R}'$ some UCQ rewriting for $\mathcal{Q}_2, \mathcal{T}'$ we have $\mathcal{Q}_1 \in \mathcal{R}'$. Finally, let $\mathcal{R}^{\mathcal{T}}_{\mathcal{L}}$ denote the smallest subset of $\mathcal{R}$ containing $\mathcal{Q}$ and all queries $\mathcal{Q}_1 \in \mathcal{R}$ that are not $\mathcal{L}$-derived by any other $\mathcal{Q}_2 \in \mathcal{R}$.*

LEMMA 3.9. *Let $\mathcal{T}$ be a OWL 2 DL-TBox, let $\mathcal{Q}$ be a CQ, let $\mathcal{R}$ be a UCQ rewriting for $\mathcal{Q}, \mathcal{T}$, let* ans *be a query answering system complete for some fragment $\mathcal{L}$ of OWL 2 DL, and let $\mathcal{R}^{\mathcal{T}}_{\mathcal{L}}$ be as defined in Definition 3.8. Then, for every $\mathcal{A}$ we have $\mathsf{cert}(Q, \mathcal{T} \cup \mathcal{A}) = \mathsf{cert}(\mathcal{R}^{\mathcal{T}}_{\mathcal{L}}, \mathcal{A}_s)$, where $\mathcal{A}_s$ is the saturation computed by* ans *for $\mathcal{T} \cup \mathcal{A}$.*

EXAMPLE 3.10. *Consider the TBox $\mathcal{T}$, query $\mathcal{Q}$, and rewriting $\mathcal{R}$ of Example 3.7. For this rewriting we have that $\mathcal{Q}$ OWL 2 RL-derives $\mathcal{Q}_1$ due to $\exists R.\top \sqsubseteq B$ but $\mathcal{Q}_2$ is not OWL 2 RL-derived by any query of $\mathcal{R}$; it is derived by $\mathcal{Q}_1$ due to $A \sqsubseteq \exists R.\top$ which is not an OWL 2 RL axiom. Hence, as shown in Example 3.7, we can remove $\mathcal{Q}_1$ from $\mathcal{R}$ but not $\mathcal{Q}_2$.*

To implement the above technique one has to modify the internals of a rewriting algorithm in order to check what kind of axioms are used to derive some query during the construction of a rewriting. If the axioms used are expressed in the language $\mathcal{L}$ then the query can be marked for removal from the final rewriting.

4. SPEEDING UP QUERY REWRITING COMPUTATION

Besides minimising the size of the rewriting, many of the previous techniques can be used to possibly speed up its computation in the first place.

EXAMPLE 4.1. *Consider the query $\mathcal{Q} = A(x) \wedge D(x) \rightarrow Q(x)$ and the TBox $\mathcal{T} = \{B_1 \sqsubseteq D, \ldots, B_n \sqsubseteq D\}$ for $n > 0$ some natural number. Any UCQ rewriting for $\mathcal{Q}, \mathcal{T}$ should contain all queries $\mathcal{Q}_i = A(x) \wedge B_i(y) \rightarrow Q(x)$ for $1 \leq i \leq n$. However, assume that we know beforehand that atom $A(x)$ is empty. In that case, appart from discarding all queries $\mathcal{Q}_i$, it would be beneficial to avoid generating them in the first place.* ♢

The above situation is actually not uncommon even for real-world ontologies. How to exploit emptiness information to speed up the computation of a rewriting clearly depends on how each query rewriting algorithm/system works and we cannot go into detail for every existing one. We provide a short note about how these could be exploited by IQAROS and Rapid which demonstrates that similar extensions could be incorporated to more systems.

EXAMPLE 4.2. *Consider the query $\mathcal{Q} = C(x) \wedge D(x) \rightarrow Q(x)$, the TBox $\mathcal{T}' = \mathcal{T} \cup \{A \sqsubseteq C\}$ where $\mathcal{T}$ is the TBox from Example 4.1, and assume also that concept A has no instances.*

For the above scenario IQAROS will perform the following steps: first, it will consider the two atoms of $\mathcal{Q}$ in separation and compute a rewriting for query $\mathcal{Q}_\alpha = C(x) \rightarrow Q(x)$ which consists of $\mathcal{R}_\alpha = \{\mathcal{Q}_\alpha, A(x) \rightarrow Q(x)\}$ and a rewriting for $\mathcal{Q}_{\alpha'} = D(x) \rightarrow Q(x)$ which consists of $\mathcal{R}_{\alpha'} = \{\mathcal{Q}_{\alpha'}, B_1(x) \rightarrow Q(x), \ldots, B_n(x) \rightarrow Q(x)\}$ and, second, it will construct the rewriting of $\mathcal{Q}$ by "joining" the body atoms of the queries in the partial rewritings $\mathcal{R}_\alpha$ and $\mathcal{R}_{\alpha'}$. More precisely, by joining $A(x)$ with each $B_i(x)$ it computes all queries $\mathcal{Q}_i = A(x) \wedge B_i(x) \rightarrow Q(x)$.

However, by using the emptiness information for A, we can skip joining atom $A(x)$ from query $A(x) \rightarrow Q(x) \in \mathcal{R}_\alpha$ with any atom of the queries in $\mathcal{R}_{\alpha'}$. This could avoid generating a significant number of queries that would eventually have an empty answer set. ♢

Modulo handling of existential restrictions, Rapid has some similarities to IQAROS. Given a query with atoms α_i, Rapid first computes so-called *unfolding sets* US_{α_i} for each one of them and then combines (joins) elements from each US_{α_i} to compute the rewriting of the input query. In Example 4.2 the unfolding set of $C(x)$ would consist of the set $\{C(x), A(x)\}$ and the unfolding set of $D(x)$ would consist of $\{B_1(x), \ldots, B_i(x), D(x)\}$. Clearly, by picking one atom from each (unfolding) set and putting them together (joining them) we can re-construct the queries $\mathcal{Q}_i$. However, by exploiting the emptiness information we can remove atom $A(x)$ from the first unfolding set. Hence, in the combination phase we will avoid generating all $\mathcal{Q}_i$ queries.

5. EVALUATION

5.1 Evaluation of Subsumption-Based Redundancy Elimination

First, we evaluated the efficiency of Algorithm 1 and compared it to the standard implementation found in the original Requiem system [14] in order to assess the impact of our heuristics. For the evaluation we used the same benchmarking setting (ontologies and test queries) as the one originally proposed in [14].

Table 1 presents the results excluding cases that both algorithms require less than 50msec to prune the redundant queries. The table shows the size of the rewriting before

Table 2: Results for ontologies LUBM, UOBM, Reactome, and Uniprot; $\sharp\mathcal{R}_*$ denotes the size of the query computed by some of the considered systems; times are presented in milliseconds.

		UCQ Rewritings in SQL				**Evaluation Times**				
$\mathcal{O}$	$\mathcal{Q}$	$\sharp\mathcal{R}_{\mathsf{Inc}}$	$\sharp\mathcal{R}_{\mathsf{Inc}_{\mathsf{db}}}$	$\sharp\mathcal{R}_{\mathsf{Inc}_{\mathsf{rl}}}$	$\sharp\mathcal{R}_{\mathsf{Ontop}}$	Inc	$\mathsf{Inc}_{\mathsf{db}}$	Ontop	$\mathsf{Inc}_{\mathsf{rl}}$	Stardog
LUBM	$\mathcal{Q}_2$	4	1	1	4	1245	950	2175	367	182
	$\mathcal{Q}_4$	18	6	6	13122	9388	4862	-	132	260
	$\mathcal{Q}_8$	8	1	1	36	1087	819	6915	174	74
	$\mathcal{Q}_9$	2	1	1	9	3824	3473	19363	115	41
	$\mathcal{Q}_{14}$	1	1	1	1	1683	1512	4611	163	10
UOBM	$\mathcal{Q}_1$	1	1	1	1	670	676	958	2	206
	$\mathcal{Q}_2$	18	11	1	23324	10690	8501	-	268	1078
	$\mathcal{Q}_7$	114	77	48	114	2065	1269	2408	4	718
	$\mathcal{Q}_{11}$	513	354	47	522	180457	153546	309137	-	2098
Reactome	$\mathcal{Q}_2$	32	2	1	128	395	258	1755	11	717
	$\mathcal{Q}_3$	32	2	1	128	268	144	2360	15	451
	$\mathcal{Q}_4$	32	2	1	128	234	47	2370	8	511
	$\mathcal{Q}_6$	32	0	1	0	344	53	36	35	2655
	$\mathcal{Q}_7$	32	0	1	0	273	48	44	62	2229
Uniprot	$\mathcal{Q}_1$	22	20	9	0	100	93	206	55	709
	$\mathcal{Q}_4$	11	9	9	0	13	15	14	6	118
	$\mathcal{Q}_6$	11	0	0	0	12	3	5	3	111

Pre-processing Times				
$\mathcal{O}$	$\mathsf{Inc}_{\mathsf{db}}$	Ontop	$\mathsf{Inc}_{\mathsf{rl}}$	Stardog
LUBM	109938	5321	114924	56391
UOBM	323387	7231	242789	122791
Reactome	39400	4258	33000	39292
Uniprot	75577	13159	75800	34576

($\mathcal{R}_{\mathsf{in}}$) and after ($\mathcal{R}_{\mathsf{out}}$) subsumption. As can be seen in most cases the modified algorithm is several times faster than the standard one and in some cases for even up to two orders of magnitude (e.g., ontology U query $\mathcal{Q}_5$ and UX query $\mathcal{Q}_5$). This is particularly the case when there are few non-redundant queries that subsume all the rest which, as we argued, the algorithm discovers in a very early stage due to the ordering of queries.

5.2 Evaluation of Rewriting Minimisation and Query Answering

In order to evaluate our rewriting minimisation techniques we implemented them into our query rewriting system IQAROS.[1] For data storage and UCQ rewriting evaluation we used both RDBMS systems (PostgreSQL and MySQL) as well as the OWL 2 RL system RDFox [13]. When using an RDBMS, IQAROS uses only the emptiness information from (conjunctions of) atoms to minimise the computed rewriting while when using RDFox, it discards both empty queries as well as all OWL 2 RL-derived queries using the technique outlined in Section 3.3. In the following, the former setting is called $\mathsf{Inc}_{\mathsf{db}}$ and the latter $\mathsf{Inc}_{\mathsf{rl}}$. Both of them use the optimised subsumption-based redundancy elimination algorithm as well as the rewriting optimisations presented in Section 4.

For the evaluation we used the well-known benchmarks LUBM and UOBM, for which we generated an ABox containing 30 universities, the NPD Benchmark [10] as well as the real-world ontologies, Reactome, Uniprot and Fly Anatomy. Reactome and Uniprot have a medium sized TBox (600 and 259 axioms, respectively) but a large ABox (over 1 million assertions) and the Fly Anatomy has a very large and highly challenging TBox (23,467 axioms) but a small ABox (2,500 assertions); all ontologies come with real-world queries. In the PostgreSQL RDBMS systems, used for all except the NPD benchmark, ABoxes were stored using one table per concept (role) and one (two) column(s) for storing the individuals [4]. For the NPD benchmark we used a MySQL database with a realistic set of mappings between the ontology and the database.

We compared our implementations against Ontop [20] v1.15[4] (in RDBMS mode) and Stardog [15] v4.0 under the in-memory mode. Hence, one should compare $\mathsf{Inc}_{\mathsf{db}}$ against Ontop and $\mathsf{Inc}_{\mathsf{rl}}$ against Stardog. During query answering we set a time-out of 20 minutes. For brevity and space limitations we will present the results only for some representative test queries, however, full results can be found on-line.[1]

Table 2 presents the results for LUBM, UOBM, Reactome and Uniprot. The table also includes results for IQAROS using just query subsumption and none of the rewriting minimisations presented in Sections 3.2 and 3.3, denoted by Inc. In the table, $\sharp\mathcal{R}_*$ denotes the size of the SQL/SPARQL query computed by each system and evaluated over the underlying data management system; for Stardog it was not possible to obtain this via its API. Finally, the right part of the table presents the total running time, i.e., computing the rewriting, translating to SQL/SPARQL and evaluating while the lower part the pre-processing and loading times required by each system.

As can be seen, $\mathcal{R}_{\mathsf{Inc}_{\mathsf{db}}}$ is usually much smaller than $\mathcal{R}_{\mathsf{Inc}}$, hence the relatively simple emptiness technique from Section 3.2 can already minimise a rewriting significantly. Interestingly, about 60% of the queries that are pruned are due

[4]This was the latest stable version that we were able to run.

Table 3: Results for the Fly Anatomy ontology; $\sharp\mathsf{cert}_*$ denotes the number of answers returned by a system; times are presented in seconds.

		UCQ Rewritings			Certain Answers			Evaluation Times			
$\mathcal{O}$	$\mathcal{Q}$	$\sharp\mathcal{R}_{\mathsf{Inc}_{\mathsf{db}}}$	$\sharp\mathcal{R}_{\mathsf{Ont}}$	$\sharp\mathcal{R}_{\mathsf{Inc}_{\mathsf{rl}}}$	$\sharp\mathsf{cert}_{\mathsf{Inc}}$	$\sharp\mathsf{cert}_{\mathsf{Ont}}$	$\sharp\mathsf{cert}_{st}$	$\mathsf{Inc}_{\mathsf{db}}$	Ont	$\mathsf{Inc}_{\mathsf{rl}}$	Star
Fly	$\mathcal{Q}_1$	803	7936	410	803	803	0	495.32	73.41	427.17	375.77
	$\mathcal{Q}_2$	-	-	-	-	-	0	-	-	-	374.96
	$\mathcal{Q}_3$	803	7936	847	803	803	402	56.49	26.38	30.78	428.80
	$\mathcal{Q}_4$	803	7936	410	803	803	0	727.77	62.82	705.84	395.03
	$\mathcal{Q}_5$	803	7936	409	803	803	0	185.15	66.80	152.52	381.21

Pre-processing Times			
$\mathsf{Inc}_{\mathsf{db}}$: 57.5	Ontop: 2,741.4	$\mathsf{Inc}_{\mathsf{rl}}$: 155.2	Stardog: 81.0

Table 4: Results for the NPD Benchmark; $\sharp\mathcal{R}_*$ is like in Table 2 and times are presented in milliseconds.

		UCQ Rewritings in SQL				Evaluation Times				
$\mathcal{O}$	$\mathcal{Q}$	$\sharp\mathcal{R}_{\mathsf{Inc}}$	$\sharp\mathcal{R}_{\mathsf{Inc}_{\mathsf{db}}}$	$\sharp\mathcal{R}_{\mathsf{Inc}_{\mathsf{rl}}}$	$\sharp\mathcal{R}_{\mathsf{Ontop}}$	Inc	$\mathsf{Inc}_{\mathsf{db}}$	Ontop	$\mathsf{Inc}_{\mathsf{rl}}$	Stardog
NPD	$\mathcal{Q}_1$	1	1	1	1	90	156	701	122	298
	$\mathcal{Q}_3$	12	12	12	1	157	223	22	121	88
	$\mathcal{Q}_7$	1	1	1	1	31	39	135	148	42
	$\mathcal{Q}_{11}$	1	1	1	24	393	86	533	88	249
	$\mathcal{Q}_{22}$	3	3	3	4	112	86	339	49	43

Pre-processing Times			
$\mathsf{Inc}_{\mathsf{db}}$: 266408	Ontop: 7693	$\mathsf{Inc}_{\mathsf{rl}}$: 105987	Stardog: 52321

to empty *conjunctions* of concepts, hence our extensions of the original technique [15] (Section 3.2) have practical consequences. $\mathcal{R}_{\mathsf{Inc}_{\mathsf{db}}}$ is also in almost all cases smaller than $\mathcal{R}_{\mathsf{Ontop}}$. As a consequence $\mathsf{Inc}_{\mathsf{db}}$ is also usually much faster than Ontop with most notable cases queries $\mathcal{Q}_4, \mathcal{Q}_8$ and $\mathcal{Q}_9$ over LUBM, queries $\mathcal{Q}_2$ and $\mathcal{Q}_{11}$ over UOBM and the first three queries over Reactome. Actually in $\mathcal{Q}_4$ over LUBM and $\mathcal{Q}_2$ over UOBM, Ontop timed-out. Surprisingly, in many cases Ontop computes rewritings that are similar or even larger in size compared to the unoptimised system Inc and can actually be even slower than it. The only case that Ontop computes smaller rewritings was in Uniprot. However, in this ontology Ontop computes rewritings of size 0 and consequently an empty set of answers whereas IQAROS (and Stardog) computed some answers which shows that Ontop is incomplete in this ontology.

In the case that both emptiness information and OWL 2 RL-derived queries are considered (i.e., system $\mathsf{Inc}_{\mathsf{rl}}$) even smaller rewritings are computed.[5] $\mathsf{Inc}_{\mathsf{rl}}$ and Stardog are faster than both the DB approaches, which is expected since both are in-memory systems, and $\mathsf{Inc}_{\mathsf{rl}}$ is generally faster than Stardog with the exception of a few queries in LUBM and $\mathcal{Q}_{11}$ in UOBM where it did not manage to terminate. Since $\mathsf{Inc}_{\mathsf{rl}}$ computed the rewriting of $\mathcal{Q}_{11}$ in just 329 milliseconds this time-out was caused by RDFox. Stardog was especially slow in queries $\mathcal{Q}_6$ and $\mathcal{Q}_7$ over Reactome and $\mathcal{Q}_7$ over UOBM. Ragarding loading, IQAROS and Stardog have similar times (a few minutes) whereas Ontop was the fastest (under a minute in all ontologies).

Table 3 presents our results for the Fly Anatomy ontology. As can be seen this ontology is quite challenging for all systems. More precisely, $\mathsf{Inc}_{\mathsf{db}}, \mathsf{Inc}_{\mathsf{rl}}$ and Ontop timed-out in query $\mathcal{Q}_2$, whereas Stardog was the only system that managed to terminate in all queries, however, it returned less answers than both IQAROS and Ontop; actually it returned 0 answers in all but query $\mathcal{Q}_3$. We have verified that all queries have a non-empty set of answers hence Stardog is incomplete. Consequently, it is a bit doubtful if it terminated with a correct result in query $\mathcal{Q}_2$ as again it returned 0 answers. This ontology is the only that Ontop performs better than $\mathsf{Inc}_{\mathsf{db}}$ and returns the same sets of answers. We attribute this to the many existentials of this ontology and the tree-witness rewriting of Ontop that particularly aims to deal with existentials efficiently. However, note that Ontop required 45 minutes to pre-process this ontology, apparently in order to compute all the tree-witnesses of the TBox. Moreover, over this ontology Inc (not shown in the table) managed to compute a UCQ rewriting only for query $\mathcal{Q}_3$ which illustrates the importance of the rewriting optimisations outlined in Section 4 integrated in $\mathsf{Inc}_{\mathsf{db}}$ and $\mathsf{Inc}_{\mathsf{rl}}$.

Finally, Table 4 presents our results using the NPD Benchmark. We can observe that both Inc and $\mathsf{Inc}_{\mathsf{db}}$ outperform Ontop in almost all queries. Note that in most queries the rewriting consists (only) of the original query, hence the evaluation times and the rewriting sizes of Inc and $\mathsf{Inc}_{\mathsf{db}}$ are about the same which demonstrates that this ontology is not particularly challenging and our minimisation techniques just introduced an unnecessary overhead. As expected, the in-memory systems $\mathsf{Inc}_{\mathsf{rl}}$ and Stardog perform faster than the RDBMS-based systems and $\mathsf{Inc}_{\mathsf{rl}}$ slightly outperforms Stardog while the pre-processing times of IQAROS and Stardog are similar and Ontop was the fastest.

6. CONCLUSIONS

We have studied the problem of efficient query answering over lightweight ontologies using query rewriting. Towards our goal we have designed and implemented several tech-

[5]Note that the saturation performed by RDFox in $\mathsf{Inc}_{\mathsf{rl}}$ alters the emptiness of concepts and roles (since the saturation $\mathcal{A}_s$ is a superset of $\mathcal{A}$) hence it is not necessarily the case that $\mathcal{R}_{\mathsf{Inc}_{\mathsf{rl}}}$ is a subset of $\mathcal{R}_{\mathsf{Inc}_{\mathsf{db}}}$.

niques for minimising the size of a computed rewriting in order to make its evaluation over the data management system as efficient as possible.

First, we presented an efficient subsumption-based redundancy elimination algorithm that is able to scale even over large UCQ rewritings. To accomplish this it employs several non-trivial heuristics over the standard algorithm implemented in the Requiem system [14]. To the best of our knowledge, similar heuristics have never been presented neither in the ontology nor in the automated-theorem proving literature where subsumption is a central optimisation technique. Moreover, our algorithm is applicable to any set of (first-order disjunctive) clauses and not just UCQ rewritings (sets of CQs) hence it is also of general interest.

Second, we investigated and verified that extending the idea of atom emptiness to conjunctions of atoms has many practical benefits as even in practical real-world scenarios rewritings do contain many queries that are empty due to empty conjunctions.

Third, we designed and formalised a novel UCQ minimisation technique which is based on the inference capabilities of OWL 2 RL systems. Using such systems to evaluate UCQ rewritings is certainly not a new idea (Stardog is an example), however, formalising the notion of $\mathcal{L}$-derived concepts and showing how to prune the rewriting has not been shown before.

Fourth, we have also shown how many of these techniques can be used to speed up the computation of the rewriting in the first place. As shown by our evaluation these refinements enabled IQAROS to compute a rewriting for all except one query over the highly challenging Fly Anatomy ontology.

Finally, we have integrated all techniques into our system IQAROS and conducted an extensive experimental evaluation using both well-known benchmarks, the newly proposed NPD Benchmark as well as several real-world (including challenging) ontologies. The experiments show that our system is quite robust, in the vast majority of cases it computes the smallest rewritings and hence is also usually the fastest, and computes the right answers in all tests. Moreover, our pre-processing times were always at the order of a few minutes and we feel that this extra penalty is worthwhile compared to the benefits of the much more important on-line query answering time.

Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 604102 (Human Brain Project), the Research Centre of the Athens University of Economics and Business, in the framework of Research Funding at AUEB for Excellence and Extroversion" and a EU FP7 Marie Curie Career Reintegration Grant with grant agreement 303914.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] P. Aravogliadis and V. Vassalos. On Equivalence and Rewriting of XPath Queries Using Views under DTD Constraints. In *Proc. of the 22nd Int. Conference on Database and Expert Systems Applications (DEXA 2011)*, pages 1–16, 2011.

[3] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[4] 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 Autom. Reasoning*, 39(3):385–429, 2007.

[5] B. Cuenca Grau and G. Stoilos. What to ask to an incomplete semantic web reasoner? In *Proc. of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 2226–2231. AAAI Press, 2011.

[6] T. Eiter, M. Ortiz, M. Simkus, T.-K. Tran, and G. Xiao. Query Rewriting for Horn-SHIQ Plus Rules. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*, 2012.

[7] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proceedings of the 27th International Conference on Data Engineering (ICDE 2011)*, pages 2–13, 2011.

[8] M. Imprialou, G. Stoilos, and B. Cuenca Grau. Benchmarking ontology-based query rewriting systems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2012)*. AAAI Press, 2012.

[9] S. Kikot, R. Kontchakov, and M. Zakharyaschev. On (In)Tractability of OBDA with OWL 2 QL. In *Proceedings of the 24th International Workshop on Description Logics (DL 2011)*, 2011.

[10] D. Lanti, M. Rezk, G. Xiao, and D. Calvanese. The NPD benchmark: Reality check for OBDA systems. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015.*, pages 617–628, 2015.

[11] C. Lutz, I. Seylan, D. Toman, and F. Wolter. The combined approach to OBDA: taming role hierarchies using filters. In *12th International Semantic Web Conference (ISWC 2013)*, pages 314–330, 2013.

[12] J. Mora, R. Rosati, and Ó. Corcho. kyrie2: Query rewriting under extensional constraints in *ELHIO*. In *Proceedings of the International Semantic Web Conference (ISWC 14)*, pages 568–583, 2014.

[13] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee. Rdfox: A highly-scalable RDF store. In *Proceedings of the 14th International Semantic Web Conference (ISWC 2015)*, pages 3–20, 2015.

[14] H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient Query Answering for OWL 2. In *Proceedings of the International Semantic Web Conference (ISWC2009)*, pages 489–504, 2009.

[15] H. Pérez-Urbina, E. Rodríguez-Díaz, M. Grove, G. Konstantinidis, and E. Sirin. Evaluation of query rewriting approaches for OWL 2. In *Joint Workshop on Scalable Semantic Web Systems and High Performance Semantic Web Systems*, 2012.

[16] F. D. Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *Proc. of the 16th International Conference on Extending Database Technology (EDBT 2013)*,

pages 561–572, 2013.

[17] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *Journal on Data Semantics*, X:133–173, 2008.

[18] P. Ramanan. Efficient algorithms for minimizing tree pattern queries. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD '02)*, pages 299–309. ACM, 2002.

[19] M. Rodriguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work in practice. In *Proc. of the 5th Alberto Mendelzon Int. Workshop on Foundations of Data Management (AMW 2011)*, 2011.

[20] M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontology-based data access: Ontop of databases. In *Proceedings of the 12th International Semantic Web Conference (ISWC 2013)*, pages 558–573, 2013.

[21] R. Rosati. Prexto: Query rewriting under extensional constraints in DL-Lite. In *Proceedings of the 9th Extended Semantic Web Conference*, pages 360–374, 2012.

[22] D. Trivela, G. Stoilos, A. Chortaras, and G. Stamou. Optimising resolution-based rewriting algorithms for OWL ontologies. *Journal of Web Semantics*, 33:30–49, 2015.

[23] T. Venetis, G. Stoilos, and G. Stamou. Query extensions and incremental query rewriting for OWL 2 QL ontologies. *Journal on Data Semantics*, 3(1):1–23, 2014.

[24] P. T. Wood. Minimising Simple XPath Expressions. In *Proc. of the 4th Int. Workshop on the Web and Databases (WebDB 2001)*, pages 13–18, 2001.