Query Rewriting Under Ontology Contraction

Eleni Tsalapati, Giorgos Stoilos, Giorgos Stamou, and George Koletsos

School of Electrical and Computer Engineering,
National Technical University of Athens,
Zographou Campus, 15780, Athens, Greece

Abstract. Conjunctive query (CQ) answering is a key reasoning service for ontology-based data access. One of the most prominent approaches to conjunctive query answering is query rewriting where a wide variety of systems has been proposed the last years. All of them accept as input a fixed CQ q and ontology $\mathcal{O}$ and produce a rewriting for $q, \mathcal{O}$. However, in many real world applications ontologies are very often dynamic—that is, new axioms can be added or existing ones removed frequently. In this paper we study the problem of computing a rewriting for a CQ over an ontology that has been *contracted* (i.e., some of its axioms have been removed) given a rewriting for the input CQ and ontology. Our goal is to compute a rewriting directly from the input rewriting and avoid computing one from scratch. We study the problem theoretically and provide sufficient conditions under which this process is possible. Moreover, we present a practical algorithm which we implemented and evaluated against other state-of-the-art systems obtaining encouraging results. Finally, axiom removal can also be relevant to ontology design. For each test ontology we study how much the removal of an axiom affects the size of the rewriting and the performance of systems. If the removal of a single axiom causes a significant decrease either in the size or in the computation time then this part of the ontology can be re-modelled.

Keywords: Ontologies, Query Rewriting, Ontology contraction, Axiom Removal.

1 Introduction

A recent application of ontologies that continuously gains momentum is *ontology-based data access* (OBDA) [17]. Ontologies aim to provide a formal semantically rich conceptualization of the (possibly distributed) data layer, thus simplifying numerous data management problems such as information integration [13], data exchange [9], data warehousing [23] and more. The main advantages of OBDA are that, firstly, the conceptual data description does not directly reflect the specific system/storage specifications and restrictions and, secondly, the data access can be performed by answering *conjunctive queries* (CQs) expressed in terms of the ontology [17], which is usually more intuitive for the user.

Unfortunately, the problem of answering conjunctive queries over ontologies expressed using expressive ontology languages (like those underpinning the Web

Ontology Language OWL 2) has been proved to be very difficult [14]. Consequently, less expressive languages (like those underpinning the OWL 2 QL and OWL 2 EL fragments) have been proposed in the literature. For these languages query answering is tractable [5, 16, 11] and thus efficient systems can be implemented. One of the widely used methods to query answering over such languages is *query rewriting*. Given a query q and an ontology $\mathcal{O}$, a rewriting r of $q, \mathcal{O}$ is a set of clauses (usually Datalog rules or unions of CQs) such that for any database the answers of q over the database and the ontology coincide with the answers of the rewriting over the database and discarding the ontology. Thus, r can be used for finding the answers by translating it into an (recursive) SQL query.

So far many algorithms and systems for computing the rewriting of a query over an ontology have been developed in the literature [5, 16, 19, 11, 6, 15, 22]. Several of them, like Presto, Quest,[1] Nyaya,[2] Rapid,[3] and IQAROS[4] employ sophisticated optimisations in order to reduce either the size of the computed rewriting or the computation time. Despite very encouraging results it is clear that the problem of query rewriting remains open since there are several problematic cases for which either the computation time or the computed rewriting are quite large. The latter is a significant problem as it is well-known that large rewritings are likely to cause a problem when evaluated over a database (a large rewriting implies a large SQL query with many unions and joins) [11].

All the aforementioned systems assume a fixed query and ontology and for this input they employ a set of 'rewriting' rules in order to compute the target rewriting. However, in many applications ontologies are very often dynamic and can change in time [7, 3, 18]. More precisely, an ontology can be extended by adding new axioms or be *contracted*—that is, some of its axioms might be removed because they no longer hold. For example, the NCI ontology (a well-known medical ontology) has been updated more than 85 times [10]. In such scenarios all aforementioned algorithms would compute a rewriting for the input (fixed) query over the updated ontology from scratch discarding any information previously computed, although it is expected that the new rewriting has a significant overlap with the initial one. In the current paper we study the following problem: Given a query q, an ontology $\mathcal{O}$, a rewriting r for $q, \mathcal{O}$ and a set of axioms A, compute a rewriting for $q, \mathcal{O} \setminus \mathsf{A}$ 'directly' from r and by avoiding using any of the known rewriting algorithms. Firstly, we study the problem theoretically to investigate its feasibility. We thus develop sufficient conditions that if satisfied by r then this process is possible. Subsequently, we present a practical algorithm for computing the rewriting of a query over a contracted ontology. Finally, we have implemented our algorithm and we have conducted an experimental evaluation using the evaluation framework proposed in [16]. We compared the performance of our system to the performance of cutting-edge query rewriting systems and we obtained encouraging results.

[1] `http://obda.inf.unibz.it/protege-plugin/quest/quest.html`
[2] `http://mais.dia.uniroma3.it/Nyaya/Home.html`
[3] `http://www.image.ece.ntua.gr/~achort/rapid.zip`
[4] `http://code.google.com/p/iqaros/`

To the best of our knowledge there is no previous study of the problem of query rewriting over contracted ontologies. We believe that such an algorithm can be helpful in cases where computing the rewriting for a large and complex ontology is time consuming. In such cases an initial rewriting can be computed once while then rewritings for contractions of the input ontology can be computed using a lightweight algorithm. Additionally, ontology contraction can also be interesting in designing ontologies for practical applications. More precisely, given an ontology $\mathcal{O}$ and query q the proposed method can be used to investigate which of the axioms of $\mathcal{O}$ affect the size of the rewriting for $q, \mathcal{O}$. More precisely, if r is a rewriting for $q, \mathcal{O}$ while for some $\alpha \in \mathcal{O}$ r' is a rewriting for $q, \mathcal{O} \setminus \{\alpha\}$ that is significantly smaller than r, then we can deduce that the presence of α makes rewriting particularly 'hard' and hence should be revised. The specific idea has lately gained attention in the area of terminological reasoning over expressive DLs with important theoretical and practical results [10]. However, as far as we know it has not been studied in the area of query rewriting. Finally, our techniques are also relevant to the problem of ontology repairing for incomplete reasoners [21], which provides an alternative and very promissing way to scalable ontology-based data access. More precisely, our methods can be used to compute a repair for a contracted ontology by avoiding re-computing one from scratch.

2 Preliminaries

We use standard notions of first-order constants, variables, function symbols, terms, substitutions, predicates, atoms, (ground) formulae, sentences, and entailment ($\models$). A *fact* is a ground atom and an *instance* is a finite set of facts. A tuple (vector) of variables (constants) is denoted by $\vec{x}$ ($\vec{a}$). For ϕ a formula, with $\phi(\vec{x})$ we denote that $\vec{x}$ are the free variables of ϕ, while for σ a substitution, $\phi\sigma$ is the result of applying σ to ϕ. Satisfiability and entailment are defined as usual.

Existential Rules An *existential rule* [2, 4], often called *axiom*, is a sentence of the form

$$\forall \vec{x}.\forall \vec{z}.[\phi(\vec{x}, \vec{z}) \rightarrow \exists \vec{y}.\psi(\vec{x}, \vec{y})] \tag{1}$$

where $\phi(\vec{x}, \vec{z})$ and $\psi(\vec{x}, \vec{y})$ are conjunctions of atoms and $\vec{x}, \vec{y}$ and $\vec{z}$ are pair-wise disjoint. Formula ϕ is the *body*, formula ψ is the *head*, and universal quantifiers are often omitted. If $\vec{y}$ is empty, the rule is called *datalog*. An *ontology* $\mathcal{R}$ is a finite set of existential rules.

Many popular Description Logics, such as $\mathcal{ELHI}$ [16], as well as database constraint languages, such as *tuple generating dependencies* [1], can be captured by existential rules.

Queries A *datalog query* $\mathcal{Q}$ is a tuple $\langle Q_P, P\rangle$, where Q_P is a query predicate and P is a set of datalog rules such that the body of each clause in P does not contain Q_P. A datalog query $\mathcal{Q} = \langle Q_P, P\rangle$ is called a *union of conjunctive queries* (UCQ) if Q_P is the only head predicate in the head of the rules in

P; furthermore, $\mathcal{Q}$ is a *conjunctive query* (CQ) if it is a union of conjunctive queries and P contains exactly one rule—that is, $\mathcal{Q} = \langle Q_P, \{Q_C\}\rangle$ and Q_C is a datalog rule with Q_P as a head predicate. We often abuse notation and write $\mathcal{Q} = Q_C$ if $\mathcal{Q}$ is a CQ in which case the head of Q_C is the query predicate. A tuple of constants $\vec{a}$ is a *certain answer* of a datalog query $\mathcal{Q} = \langle Q_P, P\rangle$ over an ontology $\mathcal{R}$ and an instance I if and only if $\mathcal{R} \cup I \cup P \models Q_P(\vec{a})$. We denote with $\mathsf{cert}(\mathcal{Q}, \mathcal{R} \cup I)$ the set of certain answers of the datalog query $\mathcal{Q}$ over $\mathcal{R} \cup I$.

Given two datalog rules r_1, r_2 we say that r_2 *subsumes* r_1 if there exists a substitution σ such that $(r_2)\sigma \subseteq r_1$.

Query Rewriting Intuitively, a *rewriting* of $\mathcal{Q}$ w.r.t. an ontology $\mathcal{R}$ is another query that captures all the information from $\mathcal{R}$ relevant for answering $\mathcal{Q}$ over $\mathcal{R}$ and an arbitrary instance I. Today several query rewriting algorithms for many ontology languages such as DL-Lite, $\mathcal{ELHI}$, *linear*-TGDs and many more have been presented [5, 16, 11, 15]. For all these works, UCQs and datalog are common target languages for computing a query rewriting.

Definition 1. *A* datalog rewriting *of a conjunctive query $\mathcal{Q} = \langle Q_P, \{Q_C\}\rangle$ w.r.t. an ontology $\mathcal{R}$ is a datalog query $\mathcal{Q}' = \langle Q_P, P\rangle$ such that the following properties hold:*

- *each $r \in P$ either does not mention Q_P or contains Q_P only in the head,*
- *for each $r \in P$ we have $\mathcal{R} \cup \mathcal{Q} \models r$,*
- *for each instance I using only predicates from $\mathcal{R}$ we have $\mathsf{cert}(\mathcal{Q}, \mathcal{R} \cup I) = \mathsf{cert}(\mathcal{Q}', I)$.*

If the rewriting $\mathcal{Q}'$ is a UCQ then it is called UCQ rewriting.

Many state-of-the-art systems often normalise the input ontology $\mathcal{R}$ by introducing *new* (*fresh*) predicates that do not appear in $\mathcal{R}$ and then compute a rewriting using the normalised ontology. For example, the ontology $\mathcal{R} = \{R(x, y) \wedge C(y) \wedge D(y) \rightarrow A(x)\}$ would usually be normalised to $\mathcal{R}' = \{R(x, y) \wedge A_0(y) \rightarrow A(x), C(x) \wedge D(x) \rightarrow A_0(x)\}$, where A_0 is a new predicate. For such systems the second condition of Definition 1 is likely not to hold. However, the rules of the rewriting that contain such fresh predicates can be eliminated by 'unfolding' the definition of the fresh symbols creating new rules for which the condition holds.

3 Rewriting Reduced TBoxes

In this section we study the problem of computing a rewriting for a conjunctive query $\mathcal{Q}$ over an ontology $\mathcal{R}'$ given a rewriting for $\mathcal{Q}$ and an ontology $\mathcal{R} \supseteq \mathcal{R}'$—that is, given a rewriting for $\mathcal{Q}$ over a superset of $\mathcal{R}'$. Since rewriting over large ontologies can be a rather time consuming process our motivation is to avoid computing the new rewriting from scratch using any of the standard algorithms, but instead to re-use the previously computed information as much as possible. In the following, we first study the problem at a theoretical level providing illustrative examples that highlight important technical points and motivate several assumptions that are required and, then, we present the algorithm in detail.

Example 1. Consider the ontology $\mathcal{R}_1 = \{\alpha_1, \alpha_2, \alpha_3\}$, where α_1, α_2, and α_3 are defined as follows:

$$\begin{aligned}\alpha_1 &= \mathsf{Painting}(x) \rightarrow \mathsf{ManMadeObject}(x),\\ \alpha_2 &= \mathsf{isSimilarTo}(x,y) \wedge \mathsf{Painting}(y) \rightarrow \mathsf{Painting}(x),\\ \alpha_3 &= \mathsf{isCopyOf}(x,y) \rightarrow \mathsf{isSimilarTo}(x,y)\end{aligned}$$

The ontology states that a painting is a man made object, that if some object is similar to a painting then it is also a painting and anything that is a copy of an entity is also similar to this entity. Consider now the CQ $\mathcal{Q} = Q(x) \leftarrow \mathsf{ManMadeObject}(x)$. The datalog query $\mathcal{Q}' = \langle Q, P\rangle$, where P is the program consisting of the rules defined below, is a datalog rewriting of $\mathcal{Q}$ over $\mathcal{R}_1$:

$$q = Q(x) \leftarrow \mathsf{ManMadeObject}(x) \tag{2}$$

$$q_1 = Q(x) \leftarrow \mathsf{Painting}(x) \tag{3}$$

$$r_1 = \mathsf{Painting}(x) \leftarrow \mathsf{isSimilarTo}(x,y) \wedge \mathsf{Painting}(y) \tag{4}$$

$$r_2 = \mathsf{Painting}(x) \leftarrow \mathsf{isCopyOf}(x,y) \wedge \mathsf{Painting}(y) \tag{5}$$

This rewriting can be computed by any state-of-the-art query rewriting system that at-least supports the DL language $\mathcal{ELHI}$.

Assume now that we remove axiom α_3 from $\mathcal{R}_1$ obtaining the new ontology $\mathcal{R}_1' = \{\alpha_1, \alpha_2\}$. A new rewriting for $\mathcal{Q}$ and $\mathcal{R}_1$ can be computed using again the same algorithm; the rewriting would consist of rules (2)–(4). ♢

Although the new rewriting can be computed using again our rewriting system we can see that when applied over $\mathcal{Q}$ and $\mathcal{R}_1'$ this system would re-compute the rules (2)–(4). Moreover, we can see that one can compute a rewriting directly from $\mathcal{Q}'$ simply by removing rule (5) from the program P. Intuitively, this rule is produced by resolving rule (4) with axiom α_3 which has been removed from the initial ontology. Hence, this rule cannot be produced using the axioms of $\mathcal{R}_1'$. This suggests that if one has additionally annotated the elements of a rewriting with the subset of the ontology that is required to generate them, then a new rewriting would be easily computable.

Definition 2. *Let $\mathcal{Q} = \langle Q_P, \{Q_C\}\rangle$ be a CQ, let $\mathcal{R}$ be an ontology, let $\mathcal{Q}_D = \langle Q_P, P\rangle$ be a datalog rewriting of $\mathcal{Q}$ w.r.t. $\mathcal{R}$ and let some $r \in P$. We say that $\mathcal{R}' \subseteq \mathcal{R}$ is* minimal *for r if the following conditions hold:*

1. *$r \in P'$ for some $P' \subseteq P$ s.t. $\langle Q_P, P'\rangle$ is a datalog rewriting for $\mathcal{Q}$ w.r.t. $\mathcal{R}'$.*
2. *For all $\mathcal{R}'' \subset \mathcal{R}'$ condition 1 does not hold.*

Intuitively, $\mathcal{R}'$ is minimal for some r if r occurs in some rewriting for $\mathcal{Q}$ and $\mathcal{R}'$, but if we remove any rule from $\mathcal{R}'$ then r no longer occurs in *any* rewriting for $\mathcal{Q}$ and the modified ontology.

In our running example (Example 1) we have the following minimal sets for each element of P:

$$\begin{aligned}\mathcal{R}_q &:= \emptyset && \text{is minimal for } q\\ \mathcal{R}_{q_1} &:= \mathcal{R}_q \cup \{\alpha_1\} && \text{is minimal for } q_1\\ \mathcal{R}_{r_1} &:= \{\alpha_2\} && \text{is minimal for } r_1\\ \mathcal{R}_{r_2} &:= \mathcal{R}_{r_1} \cup \{\alpha_3\} && \text{is minimal for } r_2\end{aligned}$$

Hence, since the minimal subset for r_2 contains the axiom α_3 that was previously removed from $\mathcal{R}_1$ to obtain $\mathcal{R}'_1$, we can deduce that r_2 cannot be part of a rewriting for $\mathcal{Q}, \mathcal{R}'_1$.

Note here that for a rule r of a rewriting for some query and ontology $\mathcal{R}$ there may be many minimal subsets. For example, for ontology $\mathcal{R} = \{A(x) \rightarrow B(x), C(x) \rightarrow B(x)\}$ and CQ $\mathcal{Q} = \langle Q(x), \{Q(x) \leftarrow A(x), B(x), C(x)\}\rangle$ the rewriting would contain the rule $r = Q(x) \leftarrow A(x), C(x)$ and both $\{A(x) \rightarrow B(x)\}$ and $\{C(x) \rightarrow B(x)\}$ are minimal for r. Hence, for each $r \in P$ the rewriting needs to contain all minimal subsets for a member of the rewriting.

Definition 3. *Let $\mathcal{Q} = \langle Q_P, \{Q_C\}\rangle$ be a CQ and let $\mathcal{R}$ be an ontology. A* labelled *datalog rewriting is a triple $\langle Q_P, P, \rho\rangle$ where $\langle Q_P, P\rangle$ is a datalog rewriting for $\mathcal{Q}$ w.r.t. $\mathcal{R}$ and ρ is a mapping from P to sets of subsets of $\mathcal{R}$ such that for each $r \in P$, $\rho(r)$ contains all $\mathcal{R}' \subseteq \mathcal{R}$ that are minimal for r.*

Note that to compute a labelled rewriting for a CQ $\mathcal{Q} = \langle Q, \{Q_C\}\rangle$ over an input set $\mathcal{R}$ one has to modify the internals of the used rewriting algorithm. This can be done easily by initialising the empty set $\emptyset$ as the minimal set for Q_C and the singleton set $\{\alpha\}$ for each axiom $\alpha \in \mathcal{R}$ and then track the axioms that are used to generate the elements of the output rewriting. For example, if r' is produced by resolving rule r with axiom α, then $\rho(r') = \rho(r') \cup \{\rho(r) \cup \rho(\alpha)\}$.

An important technical question at this point is whether we can compute a rewriting for a CQ over a reduced ontology $\mathcal{R}'$ given any rewriting for the input ontology $\mathcal{R} \supseteq \mathcal{R}'$. As the following example shows, this is not always possible.

Example 2. Consider the following ontology $\mathcal{R}_2$ and CQ:

$$\mathcal{R}_2 = \{\mathsf{Creator}(x) \rightarrow \mathsf{Agent}(x)\}$$
$$\mathcal{Q} = Q(x) \leftarrow \mathsf{Creator}(x), \mathsf{Agent}(x)$$

The tuple $\mathcal{Q}_1 = \langle \mathcal{Q}(x), \{r, r_1\}\rangle$, where $r = Q(x) \leftarrow \mathsf{Creator}(x), \mathsf{Agent}(x)$ and $r_1 = Q(x) \leftarrow \mathsf{Creator}(x)$ is a rewriting for $\mathcal{Q}, \mathcal{R}_2$. However, r_1 subsumes r, hence $\mathcal{Q}_2 = \langle \mathcal{Q}(x), \{r_1\}\rangle$, is also a rewriting for $\mathcal{Q}, \mathcal{R}_2$.

Assume now that axiom $\alpha_1 = \mathsf{Creator}(x) \rightarrow \mathsf{Agent}(x)$ is removed from $\mathcal{R}_2$ obtaining a new ontology $\mathcal{R}'_2$. A rewriting for $\mathcal{Q}, \mathcal{R}'_2$ consists of the query $\mathcal{Q}_3 = \langle Q(x), \{r\}\rangle$. However, it is quite clear that we cannot compute $\mathcal{Q}_3$ from the (non-redundant) rewriting $\mathcal{Q}_2$, as it does not contain the rule r at all. Instead, $\mathcal{Q}_3$ can be computed from $\mathcal{Q}_1$ that contains rule r simply by removing r_1. ♢

Intuitively, the issue in the previous example is that although r is redundant in $\mathcal{Q}_1$ the query that subsumes it (r_1) is not part of all rewritings for $\mathcal{Q}, \mathcal{R}'_2$ because the axiom that is used to generate it (i.e., α_1) has been removed. Hence, r is no longer redundant in rewritings of $\mathcal{Q}, \mathcal{R}'_2$.

As we will show next, the following condition that was first introduced in [8], provides a sufficient condition for computing a rewriting for an ontology $\mathcal{R}'$ given a rewriting for an ontology $\mathcal{R} \supseteq \mathcal{R}'$.

Definition 4. *A datalog rewriting $\langle Q_P, P\rangle$ of a CQ $\mathcal{Q} = \langle Q_P, \{Q_C\}\rangle$ w.r.t. an ontology $\mathcal{R}$ is* subset-closed *if for each $\mathcal{R}' \subseteq \mathcal{R}$ there exists $P' \subseteq P$ such that $\langle Q_P, P'\rangle$ is a datalog rewriting for $\mathcal{Q}$ w.r.t. $\mathcal{R}'$.*

Example 3. Consider the ontology $\mathcal{R}_2$ and CQ $\mathcal{Q}$ from Example 2. For $\mathcal{R}_2'' = \emptyset$ no subset of $\mathcal{Q}_2$ is a datalog rewriting for $\mathcal{Q}, \mathcal{R}_2''$. Instead, for the rewriting $\mathcal{Q}_1$ the query $\mathcal{Q}_1' = \langle Q(x), \{r\}\rangle$ is a datalog rewriting for $\mathcal{Q}, \mathcal{R}_2''$. Therefore, $\mathcal{Q}_1$ is subset-closed while $\mathcal{Q}_2$ is not. ♢

As noted in [8], however, from a practical point of view subset-closed rewritings are not straightforward to compute. As also illustrated by the above example, to compute such rewritings one would typically need to disable (at least partially) subsumption-based optimisations, whereas many rewriting systems are optimised hence their output is typically not subset-closed. However, on the one hand, there exist highly efficient algorithms and systems that compute subset-closed rewritings [22] and on the other hand, we argue that one can compute a subset-closed rewriting once as an off-line procedure and then a lightweight algorithm can be used to compute rewritings for the revised ontologies.

Concluding our presentation of the technical issues of the algorithm we show that there are certain kinds of dependencies between the elements of a labelled rewriting which the algorithm can exploit in order to compute the new rewriting more efficiently.

Example 4. Consider our running example (Example 1) and assume that instead of α_3 we remove axiom α_2 creating the new ontology $\mathcal{R}_1'' = \{\alpha_1, \alpha_3\}$. A rewriting for $\mathcal{Q}, \mathcal{R}_1''$ consists only of rules (2) and (3). The algorithm can compute this by checking whether for all $\mathcal{R}_{r_1} \in \rho(r_1)$ we have $\alpha_2 \in \mathcal{R}_{r_1}$, which holds hence r_1 is removed, and then the same for r_2, which again holds hence r_2 is also removed obtaining finally the correct rewriting.

However, the latter check can be avoided if we order the elements of the rewriting according to the order induced by their minimal sets in ρ. More precisely, in our running example the (only) minimal set for r_2 is a superset of the (only) minimal set for $\rho(r_1)$; hence if r_1 is removed because $\alpha_2 \in \mathcal{R}_{r_1}$ then all rules produced "after" r_1 (i.e., r_2) can be discarded from further processing. ♢

To exploit the above idea the algorithm introduced in the next section first orders the elements of a rewriting according to their minimal sets. This is performed using the function order that is defined next.

Definition 5. *Let $\mathcal{Q}_D = \langle Q_P, P, \rho\rangle$ be a labelled rewriting for a CQ $\mathcal{Q}$ w.r.t. an ontology $\mathcal{R}$. The function $\mathsf{order}(\mathcal{Q}_D)$ returns a directed graph $\mathcal{G} = \langle P, \mathcal{H}\rangle$ where $\langle r_1, r_2\rangle \in \mathcal{H}$ iff for all $\mathcal{R}_1 \in \rho(r_1)$ there exists $\mathcal{R}_2 \in \rho(r_2)$ such that $\mathcal{R}_1 \subset \mathcal{R}_2$ and no $r' \in P$ exists such that for some $\mathcal{R}' \in \rho(r')$ we have $\mathcal{R}_1 \subset \mathcal{R}' \subset \mathcal{R}_2$.*

In our running example the function $\mathsf{order}(\mathcal{Q}')$ would return $\mathcal{G} = \langle P, \mathcal{H}\rangle$ where $\mathcal{H} = \{\langle q, q_1\rangle, \langle r_1, r_2\rangle\}$.

Algorithm 1 DELETE($\mathsf{A}, \mathcal{Q}_D$)

Input: $\mathcal{Q}_D = \langle Q_P, P, \rho \rangle$ is a labelled datalog rewriting and A a set of axioms.

1: $\mathcal{G} :=$ order($\mathcal{Q}_D$)
2: Initialise a triple $\mathcal{Q}'_D = \langle Q_P, P', \rho' \rangle$, where $P' = \emptyset$ and ρ' is an empty mapping
3: Initialise a stack S to contain all vertices r of $\mathcal{G}$ s.t. $\nexists r'.\langle r', r \rangle \in \mathcal{G}$
4: **while** $S \neq \emptyset$ **do**
5: Pop an element r from S
6: **if** $\mathcal{R}_i \in \rho(r)$ exists s.t. $\mathsf{A} \cap \mathcal{R}_i = \emptyset$ **then**
7: Add r to P'
8: **if** $\rho'(r)$ is undefined **then**
9: Initialise $\rho'(r) := \emptyset$
10: **end if**
11: **for all** $\mathcal{R}_j \in \rho(r)$ **do**
12: **if** $\mathsf{A} \cap \mathcal{R}_j = \emptyset$ **then**
13: $\rho'(r) = \rho'(r) \cup \{\mathcal{R}_j\}$
14: **end if**
15: **end for**
16: Push all r' such that $\langle r, r' \rangle \in \mathcal{G}$ to S
17: **end if**
18: **end while**
19: **return** Q'_D

3.1 The Delete Algorithm

As described previously it is possible to compute a rewriting for a CQ $\mathcal{Q}$ and an ontology $\mathcal{R}'$ from some subset-closed labelled rewriting for $\mathcal{Q}$ and a superset of $\mathcal{R}'$ without relying at all on traditional rewriting algorithms. Such a detailed algorithm is depicted in Algorithm 1.

Algorithm Delete accepts as input a labelled datalog rewriting $\mathcal{Q}_D$ for a query $\mathcal{Q}$ and ontology $\mathcal{R}$ and a set $\mathsf{A} \subseteq \mathcal{R}$ of axioms to be removed from $\mathcal{R}$ and produces a new datalog rewriting for $\mathcal{Q}, \mathcal{R} \setminus \mathsf{A}$. First, the algorithm calls function order to sort the elements of $\mathcal{Q}_D$ and create a directed graph $\mathcal{G}$ (line 1) while then it initialises a new labelled datalog rewriting $\mathcal{Q}'_D$ which will be the output of the algorithm. Then $\mathcal{G}$ is traversed in a depth-first manner (using a stack S) and checks if for some element r of the graph there exists a minimal subset in $\rho(r)$ that does not contain any element of A. This implies that r can be generated by not using any of the removed axioms and hence should be in the output of the algorithm. Thus, r is added to the new rewriting (line 7) and $\rho'(r)$ is set to all minimal subsets of r that do not contain any axiom from A (lines 11–15). Finally, all successor nodes of r in the graph are added to the stack (line 16).

Example 5. Consider the ontology $\mathcal{R}_1$ of the running example (Example 1) extended by the set of axioms $\{\alpha'_1, \alpha'_2, \alpha'_3\}$, where α'_1, α'_2, α'_3 are defined as follows:

$$\begin{aligned} \alpha'_1 &= \mathsf{Potrait}(x) \rightarrow \mathsf{Painting}(x) \\ \alpha'_2 &= \mathsf{Fossil}(x) \rightarrow \mathsf{ManMadeObject}(x), \\ \alpha'_3 &= \mathsf{ResinFossil}(x) \rightarrow \mathsf{Fossil}(x) \end{aligned}$$

The new ontology $\mathcal{R}'_1 = \mathcal{R}_1 \cup \{\alpha'_1, \alpha'_2, \alpha'_3\}$ additionally states that a potrait is a painting, a fossil is a man made object and a resin fossil is a fossil. Consider again the CQ of Example 1. The query $\mathcal{Q}' = \langle Q, P', \rho \rangle$, where $P' = P \cup \{q'_1, q'_2, q'_3\}$ and q'_1, q'_2, q'_3 are defined as follows, is a labelled subset-closed datalog rewriting of $\mathcal{Q}, \mathcal{R}'_1$:

$$q'_1 = Q(x) \leftarrow \mathsf{Potrait}(x) \tag{6}$$

$$q'_2 = Q(x) \leftarrow \mathsf{Fossil}(x) \tag{7}$$

$$q'_3 = Q(x) \leftarrow \mathsf{ResinFossil}(x) \tag{8}$$

Assume now that we remove the axiom α'_2. We will show how Algorithm 1 will compute a rewriting for $\mathcal{Q}, \mathcal{R} \setminus \{\alpha'_2\}$.

The algorithm would first initialise a rewriting $\mathcal{Q}'_D = \langle Q_P, P', \rho' \rangle$, with $P' = \emptyset$ and ρ' an empty mapping. Then, it would execute the function $\mathsf{order}(\mathcal{Q}')$ which would return the directed graph $\mathcal{G} = \langle P, \mathcal{H} \rangle$, where

$$\mathcal{H} = \{\langle q, q_1 \rangle, \langle q_1, q'_1 \rangle, \langle q, q'_2 \rangle, \langle q'_2, q'_3 \rangle, \langle r_1, r_2 \rangle\}$$

Then, initially S would contain q and r_1. Suppose that q is popped. Since $\mathcal{R}_q = \emptyset$ CQ q would be added to P' and $\rho'(q)$ is set to $\emptyset$. Since $\langle q, q_1 \rangle$, $\langle q, q'_2 \rangle \in \mathcal{H}$, the CQs q_1, q'_2 are pushed in the stack. Suppose that q_1 is popped from the stack next. Since $\mathcal{R}_{q_1} = \{\alpha_1\}$ the condition in line 6 is satisfied and q_1 would also be added to P' while the algorithm sets $\rho'(q_1) = \{\alpha_1\}$. Similarly, q'_1 is added to P' and so far we have $P' = \{q'_1, q_1, q\}$.

Now since there is no q' s.t. $\langle q'_1, q' \rangle \in \mathcal{H}$ nothing is pushed in the stack. Next, q'_2 is popped from the stack. Since $\mathcal{R}_{q'_2} = \{\alpha'_2\}$ the condition of line 6 is not satisfied; therefore the algorithm continuous with the next element of the stack which is r_1. Following the same process as before the algorithm would add r_1 and r_2 to P', hence we will have $P' = \{r_1, r_2, q'_1, q_1, q\}$. It can be verified that the datalog rewriting $\langle Q(x), P' \rangle$ returned by the algorithm is a rewriting for $\mathcal{Q}, \mathcal{R} \setminus \{\alpha_2\}$. ♢

Next we show correctness of Algorithm 1.

Theorem 1. *Let $\mathcal{R}$ be an ontology, let $\mathcal{Q} = \langle Q, P_0 \rangle$ be a CQ and let $\mathcal{Q}_D = \langle Q, P, \rho \rangle$ be a labelled datalog rewriting for $\mathcal{Q}, \mathcal{R}$ that is subset-closed. Let also A be a subset of $\mathcal{R}$. When applied to A and $\mathcal{Q}_D$ Algorithm 1 terminates. Let $\mathcal{Q}'$ be the tuple produced by the algorithm; then, $\mathcal{Q}'$ is a rewriting for $\mathcal{Q}, \mathcal{R} \setminus \mathsf{A}$ that is subset-closed.*

Proof. First we show termination. Let $\mathcal{G}$ be the graph computed at line 1 of Algorithm 1. First, we show that $\mathcal{G}$ is a directed acyclic graph. Assume that there is a cycle in $\mathcal{G}$—that is, there exist vertices r_1, r_2 such that r_2 is reachable from r_2 and r_1 from r_2. By Definition 5 we have that for all $\mathcal{R}_1 \in \rho(r_1)$ there exists $\mathcal{R}_2 \in \rho(r_2)$ s.t. $\mathcal{R}_1 \subset \mathcal{R}_2$. Let an arbitrary $\mathcal{R}_1$ and $\mathcal{R}_2$. From the latter we also get that for this specific $\mathcal{R}_2$ there exists $\mathcal{R}'_1 \in \rho(r_1)$ s.t. $\mathcal{R}_2 \subset \mathcal{R}'_1$. Hence, we have that $\mathcal{R}_1 \subset \mathcal{R}'_1$ which contradicts the assumption that $\mathcal{R}'_1$ is minimal for r_1.

Now, since $\mathcal{G}$ is a directed acyclic graph and Algorithm 1 performs a standard depth-first traversal of $\mathcal{G}$ the algorithm clearly terminates.

Next we show that $\mathcal{Q}' = \langle Q, P' \rangle$ computed by the algorithm is a rewriting for $\mathcal{Q}, \mathcal{R} \setminus \mathsf{A}$. In order to show this it suffices to show that for all instances I we have $\mathsf{cert}(\mathcal{Q}, \mathcal{R} \setminus \mathsf{A} \cup I) = \mathsf{cert}(\mathcal{Q}', \mathcal{A})$.

First, we show that $\mathsf{cert}(\mathcal{Q}, \mathcal{R} \setminus \mathsf{A} \cup I) \supseteq \mathsf{cert}(\mathcal{Q}', I)$. By construction, a rule r is in P' only if there exists $\mathcal{R}' \in \rho(r)$ such that $\mathcal{R}' \subseteq \mathcal{R} \setminus \mathsf{A}$. This implies that $\mathcal{R} \setminus \mathsf{A} \models \mathcal{R}'$. Moreover, by definition of ρ we have that r belongs in some rewriting for $\mathcal{Q}, \mathcal{R}'$ hence we have that $\mathcal{R}' \cup \mathcal{Q} \models r$. From both conditions and monotonicity it follows that $\mathcal{R} \setminus \mathsf{A} \cup \mathcal{Q} \models r$. This holds for all members of P' hence we have $(\mathcal{R} \setminus \mathsf{A}) \cup \mathcal{Q} \models P'$ which implies that for any I the answers of $\mathcal{Q}'$ over I are also answers of $\mathcal{Q}$ over $(\mathcal{R} \setminus \mathsf{A}) \cup I$.

Second, we show that $\mathsf{cert}(\mathcal{Q}, \mathcal{R} \setminus \mathsf{A} \cup I) \subseteq \mathsf{cert}(\mathcal{Q}', I)$. Since $\mathcal{Q}_D = \langle Q, P \rangle$ is subset-closed there exists $P'' \subseteq P$ such that $\langle Q, P'' \rangle$ is a rewriting for $\mathcal{R} \setminus \mathsf{A}$. Clearly for each $r \in P''$ and for each $\mathcal{R}' \in \rho(r)$ we have $\mathcal{R}' \subseteq \mathcal{R} \setminus \mathsf{A}$. Hence, it follows easily by construction of P' that it contains r and thus we have $P'' \subseteq P'$; hence, $P' \models P''$ and it follows that any answer of $\mathcal{Q}$ over $(\mathcal{R} \setminus \mathsf{A}) \cup I$ is also an answer of $\mathcal{Q}'$ over I.

Finally, we show that $\mathcal{Q}' = \langle Q, P' \rangle$ is subset-closed. Consider some arbitrary subset $\mathcal{R}_s \subseteq \mathcal{R} \setminus \mathsf{A}$. Clearly, $\mathcal{R}_s \subseteq \mathcal{R}$ and since $\mathcal{Q} = \langle Q, P \rangle$ is subset-closed then there exists some $P_s \subseteq P$ s.t. $\langle Q, P_s \rangle$ is a rewriting for $\mathcal{Q}, \mathcal{R}_s$. By the latter we get that for all $r_s \in P_s$ we have $\mathcal{R}_s \cup \mathcal{Q} \models r_s$, hence there exists $\mathcal{R}'_s \in \rho(r_s)$ that is minimal for r_s. It follows easily that $r_s \in P'$. Since r_s is an arbitrary rule we have that $P_s \subseteq P'$. Moreover, also note that $\mathcal{R}_s$ is arbitrary. Hence, it follows that $\mathcal{Q}'$ is subset-closed. □

The performance of Algorithm 1 can be further improved if one additionally has pre-computed and stored the subsumption relations between the elements of the input rewriting $\mathcal{Q}_D$. This can be accomplished by executing the standard subsumption checking algorithm over $\mathcal{Q}_D$ and creating an additional mapping λ such that for a clause r, $\lambda(r)$ contains the subsumers of r in $\mathcal{Q}_D$. Algorithm 1 uses λ as follows:

- When it selects a new clause r in line 5 it proceeds in processing r only if $\lambda(r) = \emptyset$ or none of the clauses in $\lambda(r)$ is already in P'; otherwise r and all the clauses that are "after" r in the graph can be discarded by continuing with the next element in the stack.
- In line 16 it checks whether for some clause r_k such that $\langle r, r_k \rangle \in \mathcal{H}$ we have $r_k \in \lambda(r)$. In such case, only r_k is pushed to the stack.

The correctness of this optimisation is a straightforward consequence of the correctness of subsumption for First-Order logic. More precisely, if a clause r subsumes a clause r', then any resolution inference using r' will produce clauses that are subsumed by clauses produced using resolution over r. Hence, if r is already in P', then both r' and all descendant rules are redundant and can be discarded from the output. Note, however, that the output of this optimised algorithm is clearly not guaranteed to be subset-closed.

Concluding this section we comment on the problem of query answering. Note that a subset-closed rewritings $\mathcal{Q}$ for a query $\mathcal{Q}_0$ and ontology $\mathcal{R}$ can typically be much larger than an equivalent non-redundant one. Hence, for an instance I it would not be very practical to use $\mathcal{Q}$ to compute the answers of $\mathcal{Q}_0$ over $\mathcal{R} \cup I$. In such setting one should use $\mathcal{Q}$ to compute a rewriting for further constructions of $\mathcal{R}$ while to evaluate the computed rewriting a non-redundant one should be computed from $\mathcal{Q}$. Note that given λ this is a fairly easy task.

4 Evaluation

We have developed a prototype tool for computing the rewriting of a conjunctive query w.r.t. a contracted ontology based on Algorithm 1. Our implementation is based on the query rewriting system ProgRes [20]—that is, we have modified ProgRes to extract subset-closed labelled rewritings.[5] Then, Algorithm 1 is executed over the subset-closed rewriting and a set of axioms. We have developed two versions of the algorithm; an unoptimised one, called Del, and one that uses the optimisations outlined at the end of Section 3, called DelOpt.

We have compared our implementations against the standard (non-modified) version of ProgRes and a recently developed highly-optimised query rewriting system IQAROS which has been shown to outperform many existing rewriting systems [12]. For the evaluation we used the framework proposed in [16]. It consists of nine test ontologies together with a set of five hand-crafted test queries for each of them. All experiments were conducted on a Intel(R) Core (TM) with a 3.20GHz processor and 4GB of RAM.

For each test ontology and query we compute a subset-closed labelled rewriting and then execute Del and DelOpt by selecting one axiom of the input ontology. Finally, we remove the subsumed (redundant) clauses. This process is repeated for all axioms of the ontology. For ProgRes and IQAROS we measure the time to compute the rewriting for the respective contracted ontology from scratch. Table 1 shows the average computation time for each ontology and query. Note that all four tools returned rewritings of the same size so for brevity reasons we do not present these numbers.

Comparing Del with DelOpt we see that DelOpt is in most cases faster than Del. This is due to the optimisations that have been implemented which prune the search space of Algorithm 1 significantly by discarding parts of the graph $\mathcal{G}$ that are redundant, e.g., because for some r in line 5 we have $\lambda(r) \neq \emptyset$. However, in ontology $P5X$ queries Q_4 and Q_5, Del performs better than DelOpt. We concluded that this is due to the overhead of the implemented optimisations of DelOpt. More precisely, DelOpt needs to perform several checks over potentially large sets in order to decide whether a selected clause can be skipped. However, as shown by the table this is noticeable only in these two queries.

Compared to ProgRes and IQAROS, both Del and DelOpt are faster in the vast majority of cases. Actually, in most ontologies and queries DelOpt can compute

[5] However, we plan to use other systems as well in the future.

Table 1. Performance results for Del, DelOpt, ProgRes, and IQAROS

	V					S				
	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5
Del	0	0	0	0	0	0	1	7	8	88
DelOpt	0	0	0	0	0	0	0	3	3	26
ProgRes	2	2	11	36	15	1	5	23	25	157
IQAROS	1	1	2	3	3	0	1	8	6	98

	P5					P5X				
	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5
Del	0	0	0	0	0	1	1	8	152	3014
DelOpt	0	0	0	0	0	0	1	8	158	3384
ProgRes	3	9	4	5	6	3	11	131	2891	123094
IQAROS	0	0	0	2	12	0	3	13	280	8431

	U					UX				
	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5
Del	0	0	1	9	39	0	0	3	30	95
DelOpt	0	0	0	3	10	0	0	1	13	25
ProgRes	1	4	7	73	46	1	4	30	223	141
IQAROS	1	2	4	9	10	1	2	7	13	14

	A					AX				
	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5
Del	1	0	0	1	1	1	2	15	26	-
DelOpt	0	0	0	0	1	1	2	16	17	-
ProgRes	34	9	30	122	604	117	2587	44104	41154	-
IQAROS	3	4	32	11	204	8	59	559	405	31125

a new rewriting almost instantaneously in less than 10 milliseconds. A large difference compared to these systems can be noticed in $P5X$ query Q_5 as well as in ontologies A and AX which are particularly hard for them. However, note that we were not able to obtain results for ontology AX query Q_5 as the ProgRes implementation that we based Del did not terminate after 9 hours. Moreover, Del was slower than IQAROS in ontologies U, UX. By investigating these cases we concluded that this is due to the large size of the rewriting $\mathcal{Q}_D$ that is given as an input to Algorithm 1 as well as that for α the removed axiom and for most CQs q in the rewriting we have $\alpha \notin \rho(q)$; hence, the algorithm also produces a large output. For instance, for ontology UX and query Q_5 the graph contains on average 6622 queries most of which also belong in the rewriting $\mathcal{Q}'_D$ computed by Algorithm 1. But subsequently, most of them are redundant and need to be removed. In contrast due to the optimisations implemented in DelOpt the algorithm is able to identify on the fly many redundant CQs and avoid traversing this large graph that is given as an input.

The goal of our second experiment was to assess and interpret the extent to which an axiom of an ontology affects the size and computation time of a computed rewriting for fixed queries. If by removing an axiom the size of the rewriting or the computation time is significantly smaller than the original one

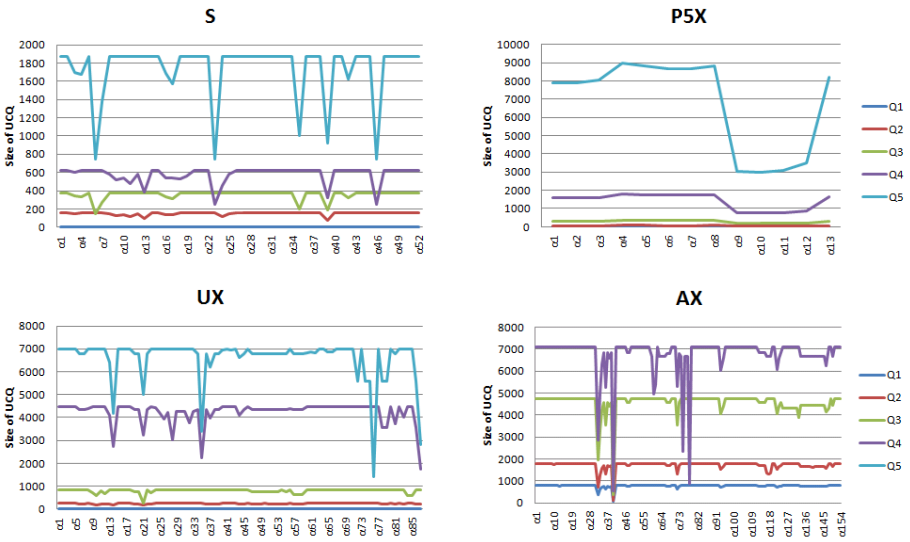

Fig. 1. Size of rewriting for ontologies $S, P5X, UX$, and AX for CQs Q_1–Q_5 when axiom α_i is removed.

then we can conclude that its existence in the specific ontology is particularly 'problematic' for the rewriting systems and hence in practical settings one would probably need to revise it. Note that even if the rewriting can be computed fast the database system would likely not be able to answer it as large rewritings imply large complex SQL queries.

For this experiment we proceeded as follows: for each ontology and query we removed iteratively each axiom and measured the size of the resulting UCQ using our system Del. We did not eliminate subsumed clauses in order to have a better picture of the number of clauses that could be produced during a rewriting process. Then, we drew plots of rewriting size vs. removed axiom in order to see for which and how many axioms there is a significant reduction in the size of the rewriting. Figure 1 presents the plots for ontologies $S, P5X, UX$, and AX which according to Table 1 are the ones that are the most difficult for the systems.

A first interesting observation is that indeed there are axioms that affect the size of the rewriting significantly. For example, in AX the removal of one of the axioms $\alpha_{32}, \alpha_{40}, \alpha_{72}$ and α_{78} causes the size of the rewriting to drop to less than half. Especially, if we remove axiom α_{40} then the rewriting of Q_4 drops from 7000 CQs to just 528 CQs. Similar observations can be made for the other ontologies as well. A second interesting observation is that for all ontologies the set of axioms that demonstrates the largest reduction is the same regardless of which query we examine. This shows that most queries are interrelated (i.e., they mention the same predicates) and that there are usually specific points in the ontology that are hard for a given query. A third interesting observation is that

for all ontologies and queries the number of axioms that affect the size of the rewriting is usually small. More precisely, for each ontology there are usually less than five axioms which if removed the size of the rewriting drops significantly.

Subsequently, we wanted to investigate the reason why these axioms affect the size of the rewriting. For AX one such axiom is $\alpha_{40} = \mathsf{AssistiveDevice}(x) \rightarrow \mathsf{Device}(x)$ ($\mathsf{AssistiveDevice} \sqsubseteq \mathsf{Device}$ in DL notation). By inspecting manually the ontology we concluded that concept Device appears very high in the hierarchy of this ontology,[6] it has many descendant concepts (that is, there are many unary predicates A in the ontology such that $AX \models A(x) \rightarrow \mathsf{Devise}(x)$), and finally it appears in all test queries. In contrast, although axiom $\alpha_{47} = \mathsf{VisualDisability}(x) \rightarrow \mathsf{Disability}(x)$ also refers to $\mathsf{Disability}$ that is also high in the hierarchy it does not affect the size of the rewriting as the hierarchy below it is rather 'shallow'. After examining all ontologies we concluded that this is a main reason for hardness. However, note that in many cases this is not immediately obvious by inspecting the ontology. For example, in case an axiom involves binary predicates the interpretation is more difficult since these can participate in axioms with unary predicates (e.g., in axioms of the form $C(x) \rightarrow R(x, f(x))$ or $R(x, y) \rightarrow C(y)$) which are not reflected in the hierarchy.

Finally, we also wanted to check whether a large reduction in the size of a rewriting also implies a large reduction in computation time for each of the tested systems. Indeed the computation time decreases in a similar way as the size of the rewriting. An interesting case is the system $\mathsf{ProgRes}$ and query Q_5 of ontology AX. Although the system is not able to terminate when processing the original input ontology even after several hours, by removing axiom α_{40} (i.e., one of the problematic ones) it can compute a rewriting for $AX \setminus \{\alpha_{40}\}$ in 16 seconds. Hence, we see that this analysis can indeed be very helpful when designing an ontology for practical applications.

5 Conclusions

In the current paper we present and study a novel problem in the area of query rewriting. More precisely, we have studied query rewriting of fixed queries over contracted ontologies—that is, over ontologies for which one or more axioms have been removed. We presented a practical algorithm which, given a rewriting $\mathcal{Q}'$ for the input query $\mathcal{Q}$ and ontology $\mathcal{O}$ (that satisfies certain conditions) and a set of axioms A to be removed from $\mathcal{O}$ it computes a rewriting $\mathcal{Q}''$ for $\mathcal{Q}, \mathcal{O} \setminus \mathsf{A}$ directly from $\mathcal{Q}$. We have implemented and evaluated the algorithm over state-of-the-art rewriting systems and have obtained encouraging results. Moreover, we have used the algorithm to analyse the role that each axiom of an ontology plays to the 'complexity' of the final rewriting. More precisely, we have measured how much the size of a rewriting is reduced if one removes an axiom of the ontology making interesting observations.

[6] That is, there are few (if any) unary predicates A in the ontology such that $AX \models \mathsf{Devise}(x) \rightarrow A(x)$.

Regarding future work we plan to investigate the same problem under ontology extensions—that is, when new axioms are added to the ontology, further optimise and evaluate our algorithm and also delve more into the role that each axiom plays in the rewriting.

Acknowledgments This work was partially supported by the European commission project 'Linked Heritage" (contract Number: ICT-PSP-270905). Giorgos Stoilos is supported by a Marie Curie Career Reintegration Grant within European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement 303914.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artificial Intelligence 175(9–10), 1620–1654 (2011)
3. Booth, R., Meyer, T., Varzinczak, I.J.: First steps in EL contraction. In: Proceedings of the Workshop on Automated Reasoning about Context and Ontology Evolution (ARCOE 2009) (2009)
4. Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., Pieris, A.: Datalog+/-: A family of logical knowledge representation and query languages for new applications. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LICS 2010). pp. 228–242 (2010)
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Journal of Automated Reasoning 39(3), 385–429 (2007)
6. Chortaras, A., Trivela, D., Stamou, G.: Optimized query rewriting in OWL 2 QL. In: Proceedings of the 23rd International Conference on Automated Deduction (CADE 23), Polland. pp. 192–206 (2011)
7. Cuenca Grau, B., Kharlamov, E., Zheleznyakov, D.: Ontology contraction: Beyond propositional paradise. In: Alberto Mendelzon International Workshop on Foundations of Data Management (AMW). Ouro Preto, Brazil (Jun 2012)
8. Cuenca Grau, B., Motik, B., Stoilos, G., Horrocks, I.: Completeness guarantees for incomplete ontology reasoners: Theory and practice. Journal of Artificial Intelligence Research 43, 419–476 (2012)
9. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. In: Proceedings of the 9th International Conference on Database Theory (ICDT). pp. 207–224 (2003)
10. Gonçalves, R.S., Parsia, B., Sattler, U.: Categorising logical differences between OWL ontologies. In: Proceedings of the 20th ACM Conference on Information and Knowledge Management (CIKM 2011). pp. 1541–1546 (2011)
11. Gottlob, G., Orsi, G., Pieris, A.: Ontological queries: Rewriting and optimization. In: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011 (2011)
12. Imprialou, M., Stoilos, G., Grau, B.C.: Benchmarking ontology-based query rewriting systems. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2012). AAAI Press (July 2012)

13. Lenzerini, M.: Data integration: a theoretical perspective. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 233–246. PODS '02, ACM, New York, NY, USA (2002)
14. Lutz, C.: The complexity of conjunctive query answering in expressive description logics. In: Proceedings of the 4th International Joint Conference on Automated Reasoning, IJCAR 2008. pp. 179–193 (2008)
15. Orsi, G., Pieris, A.: Optimizing query answering under ontological constraints. Proceedings of the VLDB Endowment 4(11), 1004–1015 (2011)
16. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. Journal of Applied Logic 8(2), 186–209 (2010)
17. Poggi, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Linking data to ontologies. JoDS X, 133–173 (2008)
18. Ribeiro, M., Wassermann, R., Antoniou, G., Flouris, G., Pan, J.: Belief contraction in web-ontology languages. In: Proceedings of the 3rd International Workshop on Ontology Dynamics (IWOD-09) (2009)
19. Rosati, R., Almatelli, A.: Improving query answering over DL-Lite ontologies. In: Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR 2010) (2010)
20. Stamou, G., Trivela, D., Chortaras, A.: Progressive semantic query answering. In: Scalable Semantic Web Knowledge Base Systems Workshop (SSWS 2010), Shanghai, China, 7-8 November 2010 (2010)
21. Stoilos, G., Cuenca Grau, B., Motik, B., Horrocks, I.: Repairing ontologies for incomplete reasoners. In: Proceedings of the 10th International Semantic Web Conference (ISWC-11), Bonn, Germany. pp. 681–696 (2011)
22. Venetis, T., Stoilos, G., Stamou, G.: Incremental query rewriting for OWL 2 QL. In: Proceedings of the 25th International Workshop on Description Logics (DL 2012), Rome, Italy (2012)
23. Widom, J.: Research problems in data warehousing. In: Proceedings of International Conference on Information and Knowledge Management. pp. 25–30 (1995)