Query Rewriting Under Ontology Evolution*

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. One of the most prominent reasoning techniques for query answering is query rewriting. The last years a wide variety of query rewriting systems has been proposed. All of them accept as input a CQ $\mathcal{Q}$ and a fixed ontology $\mathcal{O}$ and produce a rewriting for $\mathcal{Q}$, $\mathcal{O}$. 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 *evolved* (i.e., a set of new axioms has been added to $\mathcal{O}$), by exploiting the information computed for the extraction of the initial rewriting. Our goal is to avoid computing the target rewriting from scratch or repeating computations that have already been conducted for the initial rewriting. We study the problem theoretically and we specify the form that the input information must have for the efficient computation of the new rewriting based on some of the known query rewriting systems (QuOnto, Requiem, Rapid, Naya). Moreover, we present a practical algorithm which we implemented and evaluated against the query rewriting system Requiem obtaining encouraging results.

Keywords: Ontologies, Query Rewriting, Ontology Evolution, Axiom Addition.

1 Introduction

Conjunctive query answering constitutes a central problem of many applications which involve managing datasets consisting of very large sets of data assertions. The last years conjunctive query answering has become one of the most prominent reasoning tasks in Description Logics. This is witnessed by both the high number of publications on this research area, to mention only few [15, 9, 10], and the increasing number of query answering engines such as [5, 15, 18], in the last decade.

Unfortunately, it is proved that the problem of answering conjunctive queries over ontologies expressed using expressive ontology languages (like those underpinning the Web Ontology Language OWL 2) is of high computational complexity [12]. Hence, languages of low expressivity (like those underpinning the OWL 2 QL and OWL 2 EL fragments) have been proposed in the literature for which

* Work was supported by 'Linked Heritage" and an FP7 Marie Curie CIG grant

query answering is tractable w.r.t. the size of data [5, 15, 9] and efficient systems can be implemented. One of the most prominent techniques for query answering over such languages is *query rewriting*. Given a query $\mathcal{Q}$ and an ontology $\mathcal{O}$, a rewriting $\mathcal{R}$ of $\mathcal{Q}$ w.r.t. $\mathcal{O}$ is a set of clauses (usually Datalog rules or unions of CQs) such that for any database the answers of $\mathcal{Q}$ over the database and the ontology coincide with the answers of the rewriting $\mathcal{R}$ over the database. Thus, $\mathcal{R}$ can be used for finding the answers by translating it into a (recursive) SQL query.

Hitherto, numerous query rewriting systems have been developed in the literature [5, 15, 18, 9, 6, 13, 20]. Prominent examples include QuOnto[1], Requiem[2], Quest[3], Nyaya[4], Rapid[5] and IQAROS[6] while several of them employ sophisticated optimisations in order to reduce either the size of the computed rewriting or the computation time. Despite the very encouraging results, a drawback of these systems is that they assume that the input ontology over which, given a CQ they compute the target rewriting, is fixed. However, in many applications ontologies are dynamic, in the sense that they can change in time [7, 3, 16]. More precisely, a new set of axioms can be incorporated into an existing ontology or be contracted. For example, the NCI Thesaurus[7] is maintained by a multidisciplinary team of editors, who add about 900 new entries each month. While, at the same time, between two consecutive versions of the ontology 220,000 axioms were deleted [8]. In such scenarios all aforementioned systems would compute a rewriting for the input query over the new ontology from scratch discarding any information previously computed, although it is expected that the new rewriting has a significant overlap with the initial one.

Recently we studied the problem of computing a query rewriting of a CQ w.r.t. an ontology that has been contracted directly from a rewriting of the CQ w.r.t. the initial ontology [19]. In the current paper we study the following problem: Given a query $\mathcal{Q}$, an ontology $\mathcal{O}$, a set $\mathcal{R}$ computed for the extraction of a rewriting of $\mathcal{Q}$ w.r.t. $\mathcal{O}$ and a set of axioms $\mathcal{O}_N$, compute a rewriting of $\mathcal{Q}$ w.r.t. $\mathcal{O} \cup \mathcal{O}_N$ from $\mathcal{R}$ and $\mathcal{O}$ without repeating the same computations[8]. Firstly, we study the problem theoretically and we specify the form that the input $\mathcal{R}$ must have according to query rewriting system used (Requiem, QuOnto, Naya and Rapid). Next, we provide a novel algorithm that computes a rewriting for the extended ontology which is compatible with the inference systems underpinning these rewriting systems. We prove that our algorithm is correct in the sense that when evaluating the computed rewriting over the system's data we obtain the whole set of correct answers.

[1] `http://www.dis.uniroma1.it/quonto/`
[2] `http://www.cs.ox.ac.uk/isg/tools/Requiem/`
[3] `http://ontop.inf.unibz.it/`
[4] `http://mais.dia.uniroma3.it/Nyaya/Home.html`
[5] `http://www.image.ece.ntua.gr/~achort/rapid.zip`
[6] `http://code.google.com/p/iqaros/`
[7] `https://wiki.nci.nih.gov/display/VKC/NCI+Thesaurus+Terminology`
[8] Note that we assume $\mathcal{O} \cup \mathcal{O}_N$ to be consistent w.r.t. the input data and we don't handle cases where inconsistencies need to be resolved

Finally, we have optimised and implemented our algorithm using Requiem and conducted an experimental evaluation using both benchmark ontologies and an $\mathcal{ELHI}$ fragment of the realistic medical ontology GALEN. We compared the performance of our system to the performance of the query rewriting system Requiem and we obtained encouraging results.

The algorithm proposed in this paper can be exploited by cutting-edge technologies like ontology-based data access and ontology-based data integration. For instance, an innovative algorithm for ontology-based data integration under ontology evolution has recently been suggested in the literature [11], in which every new rewriting is computed from scratch, hence our algorithm could further improve its efficiency.

2 Preliminaries

In this section we introduce all necessary terminology and demonstrate the relevant definitions assuming that the reader is familiar with first-order logic.

We use standard notions of first-order constants, variables, function symbols terms, substitutions, predicates, atoms, (ground) formulae, sentences, clauses 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] 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 $\psi = \bot$ then the rule is called *constraint*. If $\vec{y}$ is empty, the rule is called *datalog*.

Many popular Horn ontology languages, such as DL-Lite$_R$ [5], $\mathcal{ELHI}$ [15], as well as Datalog$^{\pm}$ [4] can be captured by existential rules. So in the context of this paper we define an *ontology* $\mathcal{O}$ as a finite set of existential rules. An $\mathcal{ELHI}$ or a DL-Lite$_R$ ontology can be transformed into *normal form* by systematically replacing conjunctions of atoms with atomic ones along the lines of [15].

Every existential rule of a normalised $\mathcal{ELHI}$ or DL-Lite$_R$ ontology can be translated to at most two clauses. We often make no distinction between an existential rule and its equivalent representation as one or two clauses. Also, as query rewriting algorithms operate over clauses, for the rest of this work we assume that ontologies are sets of clauses.

Given two clauses c_1, c_2 we say that c_1 *subsumes* c_2 if there exists a substitution σ such that every literal in $c_1\sigma$ appears in c_2.

Queries A *query* $\mathcal{Q}$ is a finite set of sentences containing a distinct query predicate Q. A tuple of constants $\vec{a}$ is an answer to $\mathcal{Q}$ w.r.t an ontology $\mathcal{O}$ and

instance $\mathcal{I}$ if the arity of $\vec{a}$ agrees with the arity of Q and $\mathcal{O} \cup \mathcal{I} \cup \mathcal{Q} \models Q(\vec{a})$. We denote with $\mathsf{cert}(\mathcal{Q}, \mathcal{O} \cup \mathcal{I})$ the answers to $\mathcal{Q}$ w.r.t. $\mathcal{O} \cup \mathcal{I}$. A query $\mathcal{Q}$ is a *union of conjunctive queries* (UCQ) if it is a set of datalog rules containing Q in the head but not in the body. A UCQ is a *conjunctive query* (CQ) if it has exactly one rule. To simplify the presentation, we often abuse notation and identify a CQ with the only rule it contains.

Query Rewriting Intuitively, a *rewriting* of $\mathcal{Q}$ w.r.t. $\mathcal{O}$ is another query that captures all the information from $\mathcal{O}$ relevant for answering $\mathcal{Q}$ over an arbitrary instance $\mathcal{I}$ [5, 15, 9]. UCQs and datalog are common target languages for query rewriting.

Definition 1. *A* datalog rewriting *of a CQ $\mathcal{Q}$ w.r.t. an ontology $\mathcal{O}$ is a set of datalog rules $\mathcal{R}$ s.t. for each instance $\mathcal{I}$ using only predicates from $\mathcal{O}$ we have*

$$\mathsf{cert}(\mathcal{Q}, \mathcal{O} \cup \mathcal{I}) = \mathsf{cert}(\mathcal{R}, \mathcal{I})$$

The rewriting $\mathcal{R}$ is a UCQ rewriting *if it is a UCQ with query predicate Q whose body atoms contain only predicates from $\mathcal{O}$.*

Inference Rules An *inference rule* is a n+1 relation on clauses. We denote the elements of such a relation as $\langle c_1, \dots c_n, c \rangle$, and we call them *inferences*. We say that an inference $\langle c_1, \dots c_n, c \rangle$ is sound if $\{c_1, \dots c_n\} \models c$. An *inference system* Γ is a collection of inference rules. *Proof* of a clause c from a set of clauses N w.r.t. Γ is a sequence of clauses $c_1, \dots, c_m$, s.t. $c = c_m$ and each c_i is either an element of N or else the conclusion of an inference by Γ from $N \cup \{c_1, \dots, c_{i-1}\}$. For a set of clauses N and a clause c, we denote with $N \vdash_\Gamma c$, if there exists a proof of c from N using the rules of the inference system Γ. A set of clauses N is called *saturated* with respect to Γ, and we denote it as N_Γ, if the conclusion of any inference from N by Γ is an element of N [1].

3 Rewriting Extended Ontologies

In this section we present an algorithm for computing a rewriting of a query $\mathcal{Q}$ w.r.t. an ontology $\mathcal{O}'$, given a set $\mathcal{R}$ computed for the extraction of the initial rewriting of $\mathcal{Q}$ w.r.t. an ontology $\mathcal{O}$ such that $\mathcal{O} \subseteq \mathcal{O}'$. We specify the nature of $\mathcal{R}$ according to the inference system used. Our goal is to reuse as much of the information in $\mathcal{R}$ as possible, focusing on computing only the new elements that are due to the information in $\mathcal{O}' \setminus \mathcal{O}$. The input data are assumed to be consistent w.r.t. the input ontology $\mathcal{O}'$ and under this assumption one can ignore the constraint rules.

The next example illustrates the key ideas behind the algorithms we will present next.

Example 1. Consider an ontology $\mathcal{O}$ consisting of the following clauses:

$$c_1 = \quad P(x, y) \leftarrow S(x, y)$$
$$c_2 = S(x, f(x)) \leftarrow A(x)$$

and consider also the query $\mathcal{Q} = Q(x) \leftarrow S(x, y) \wedge P(x, y)$. Applying QuOnto or Rapid to compute a rewriting for $\mathcal{Q}$ w.r.t. $\mathcal{O}$ will produce the following inferences:

$$\langle \mathcal{Q}, c_1, \mathcal{Q}_1 \rangle, \text{ where } \mathcal{Q}_1 = Q(x) \leftarrow S(x, y)$$
$$\langle \mathcal{Q}_1, c_2, \mathcal{Q}_2 \rangle, \text{ where } \mathcal{Q}_2 = Q(x) \leftarrow A(x)$$

The set of queries $\mathcal{R} = \{\mathcal{Q}, \mathcal{Q}_1, \mathcal{Q}_2\}$ consists a rewriting of $\mathcal{Q}$ w.r.t. $\mathcal{O}$.

Assume now that the original ontology $\mathcal{O}$ is revised by a domain expert who adds the new clause $c_3 = S(x, f(x)) \leftarrow B(x)$, obtaining the new ontology $\mathcal{O}' = \mathcal{O} \cup \{c_3\}$. A rewriting for $\mathcal{Q}$ w.r.t. $\mathcal{O}'$ consists of the set $\mathcal{R}' = \mathcal{R} \cup \{\mathcal{Q}_3\}$, where $\mathcal{Q}_3 = Q(x) \leftarrow B(x)$ which can again be computed using any state-of-the-art algorithm.

However, in the process of computing $\mathcal{R}'$ all systems we are aware of would recompute also all queries in $\mathcal{R}' \setminus \{\mathcal{Q}_3\}$ although these have been computed before. Instead, $\mathcal{R}'$ can be computed from $\mathcal{R}$ by applying the inference rules of a rewriting algorithm using only the newly added clauses. For example, by the inference $\langle \mathcal{Q}_1, c_3, \mathcal{Q}_3 \rangle$, we can obtain $\mathcal{Q}_3$ in one step, without recomputing $\mathcal{Q}_1$ and $\mathcal{Q}_2$. ♢

The previous example suggests that to compute a rewriting for an extended ontology we can only perform the new inferences that are "activated" by the newly added clauses. Initially, these will be the inferences that contain in their premises a clause from the new set of clauses. However, these inferences will trigger new ones since their conclusions along with clauses from $\mathcal{O}'$ may infer new clauses, which again, in the same way, may "activate" new inferences, and so forth until no new inferences can be triggered.

The ability to efficiently compute a rewriting for an extended ontology given a precomputed rewriting depends greatly on the inference system used to compute the input rewriting. Many inference systems, like the ones that underpin Rabid and QuOnto, consist of inference rules whose conclusion is always a CQ. Next, we define formally this class of inference systems.

Definition 2. *A* query-oriented inference rule *is an inference rule of the form* $\langle \mathcal{Q}, c_1, \ldots, c_n, \mathcal{Q}' \rangle$*, where* $\mathcal{Q}$*,* $\mathcal{Q}'$ *are CQs and* $c_1, \ldots, c_n$ *are clauses. A* query-oriented inference system *is an inference system that consists of query-oriented inference rules.*

However, the rewriting algorithms that rely on query-oriented inference systems can only support ontologies of low expressivity, like DL-Lite$_R$ ontologies. For more expressive ontologies, such as $\mathcal{ELHI}$-ontologies, rewriting algorithms that rely on different inference systems are required. As the following example illustrates, one such system whose inference system does not satisfy Definition 2 is Requiem [14].

Example 2. Consider the ontology $\mathcal{O}$ of Example 1 and the query $\mathcal{Q} = Q(x) \leftarrow R(x, y) \wedge P(x, y)$. Requiem will initially compute the saturation of $\mathcal{Q}$, $\mathcal{O}$ by

conducting the following inferences:

$$\langle c_1, c_2, c_4 \rangle, \text{ where } c_4 = P(x, f(x)) \leftarrow A(x), \tag{2}$$

$$\langle \mathcal{Q}, c_4, \mathcal{Q}' \rangle, \text{ where } \mathcal{Q}' = Q(x) \leftarrow R(x, f(x)) \wedge A(x) \tag{3}$$

obtaining the saturation set $\mathcal{R}_s = \{c_4, \mathcal{Q}, \mathcal{Q}'\} \cup \mathcal{O}$. Then, Requiem will return the rewriting $\mathcal{R} = \{\mathcal{Q}, c_1\}$ of $\mathcal{Q}$ w.r.t. $\mathcal{O}$ by extracting the function-free subset of $\mathcal{R}_s$. Assume now that the knowledge in $\mathcal{O}$ is revised and we add the clause $c_5 = R(x, y) \leftarrow P(x, y)$ hence obtaining the new ontology $\mathcal{O}' = \mathcal{O} \cup \{c_5\}$. A rewriting for $\mathcal{Q}$ w.r.t. $\mathcal{O}'$ consists of the set $\mathcal{R}' = \{\mathcal{Q}, \mathcal{Q}_1, c_1, c_5\}$, where $\mathcal{Q}_1 = Q(x) \leftarrow A(x)$.

Suppose that we attempt to compute $\mathcal{R}'$ from $\mathcal{R}$ and $\mathcal{O}'$. Then, according to the inference system that underpins Requiem, the inferences (2), (3) and the inferences:

$$\langle c_5, c_4, c_6 \rangle, \text{ where } c_6 = R(x, f(x)) \leftarrow A(x) \tag{4}$$

$$\langle \mathcal{Q}, c_6, \mathcal{Q}'' \rangle, \text{ where } \mathcal{Q}'' = Q(x) \leftarrow A(x) \wedge P(x, f(x)) \tag{5}$$

$$\langle \mathcal{Q}', c_6, \mathcal{Q}_1 \rangle \quad \text{and} \tag{6}$$

$$\langle \mathcal{Q}'', c_4, \mathcal{Q}_1 \rangle, \text{ where } \mathcal{Q}_1 = Q(x) \leftarrow A(x) \tag{7}$$

will be performed to compute the new rewriting. In contrast to the previous example, we observe that $\mathcal{R}'$ cannot be computed from $\mathcal{R}$, $\mathcal{O}'$ without repeating the intermediate inferences (2) and (3).

However, we can avoid repeating these inferences by applying the inference rules of Requiem to the set $\{c_5\} \cup \mathcal{R}_s$. In this way, Requiem will compute $\mathcal{R}'$ by performing only the inferences (4), (5), (6) and (7). ♢

The previous example suggests that there are query rewriting algorithms that compute a rewriting by producing new intermediate clauses which may be discarded from the output. It also illustrates that for such algorithms, the rewriting of an extended ontology can be efficiently computed if, instead of the initial rewriting $\mathcal{R}$ for a CQ $\mathcal{Q}$ and an ontology $\mathcal{O}$, the saturation of the set $\mathcal{Q} \cup \mathcal{O}$ by the inference system of the algorithm is provided. Intuitively, this is a reasonable claim since a clause with functionals, which is not contained in the final rewriting, and a new clauses can trigger new inferences, crucial for the final form of the rewriting.

Next we define formally the class of inference systems that these algorithms are relied on.

Definition 3. *An* axiom-oriented inference rule *is an inference rule of the form* $\langle c_1, \ldots, c_n, c \rangle$*, where the set of clauses* $\{c_1, \ldots, c_n\}$ *contains at most one query and* c *is either a query or a clause. An* axiom-oriented inference system *is an inference system that consists of axiom-oriented inference rules.*

3.1 The Add Algorithm

As described previously, it is possible to compute a rewriting of a query $\mathcal{Q}$ w.r.t. an ontology $\mathcal{O}'$, given an inference system Γ and a set of clauses $\mathcal{R}$ generated

Algorithm 1 $\mathsf{Add}(\mathcal{O}, \mathcal{R}, \mathcal{O}_N, \Gamma)$

Input: an ontology $\mathcal{O}$, a set of clauses $\mathcal{R}$, the set of new clauses $\mathcal{O}_N$, Γ an inference system.

Output: $\mathcal{R}'$ new rewriting

1: Initialise $\mathcal{R}_{new} := \emptyset$
2: **repeat**
3: Pick clauses $c_1, \ldots, c_{i-1} \in \mathcal{R}$, clauses $c_i, \ldots, c_n \in \mathcal{O}_N$ s.t. $\langle c_1, \ldots, c_n, c \rangle$ is an inference by Γ.
4: Add c to $\mathcal{R}_{new}$
5: Pick clauses $c'_1, \ldots, c'_{j-1} \in \mathcal{R}$, clauses $c'_j, \ldots, c'_{k-1} \in \mathcal{O}_N$, clauses $c'_k, \ldots, c'_m \in \mathcal{O}$ s.t. $\langle c'_1, \ldots, c'_m, c' \rangle$ is an inference by Γ.
6: Add c' to $\mathcal{R}_{new}$
7: Pick clauses $c''_1, \ldots, c''_{s-1} \in \mathcal{R}_{new}$, clauses $c''_s, \ldots, c''_t \in \mathcal{O}_N \cup \mathcal{O} \cup \mathcal{R}$ s.t. $\langle c''_1, \ldots, c''_t, c'' \rangle$ is an inference by Γ.
8: Add c'' to $\mathcal{R}_{new}$
9: **until** no new clauses up to variable renaming are added to $\mathcal{R}_{new}$
10: **return** $\mathcal{R}' = \{c \in \mathcal{R}_{new} \cup \mathcal{R} \mid c \text{ is function-free}\}$

from $\mathcal{Q}$ and an ontology $\mathcal{O}$, with $\mathcal{O} \subseteq \mathcal{O}'$, by performing only the necessary new inferences. The nature of the set $\mathcal{R}$ depends on the type of the inference system. If Γ is query-oriented then $\mathcal{R}$ constitutes a rewriting of $\mathcal{Q}$ w.r.t. $\mathcal{O}$, while if Γ is axiom-oriented then $\mathcal{R}$ is the set $(\mathcal{Q} \cup \mathcal{O})_\Gamma$. It is reasonable to store $\mathcal{R}$, as for large ontologies its recomputation from scratch can be a very time consuming process.

Such a detailed algorithm that relies on a query-oriented or an axiom-oriented inference system is depicted in Algorithm 1. Algorithm Add accepts as input the ontology $\mathcal{O}$, the set of clauses $\mathcal{R}$, the set of new clauses $\mathcal{O}_N = \mathcal{O}' \setminus \mathcal{O}$ and a sound inference system Γ. Then, the algorithm proceeds as follows. First, it initialises a new set $\mathcal{R}_{new}$ which will contain the new inferred clauses. Next, it attempts to produce a new clause from $\mathcal{R}$ and $\mathcal{O}_N$ via the inference system Γ (line: 3). The new clause is added to the set $\mathcal{R}_{new}$. However, the inference of the first step may not produce new clauses, as clauses from $\mathcal{O}$ do not necessarily appear in its premises, although they may be required for this purpose. For instance, if $\mathcal{R}$ is a UCQ rewriting then the clauses of $\mathcal{O}$ are not contained in $\mathcal{R}$. For this reason, the algorithm conducts an extra inference which contains in its premises clauses from $\mathcal{R}$ and from *both* $\mathcal{O}$ and $\mathcal{O}_N$ (line: 5). The algorithm adds the produced clause to $\mathcal{R}_{new}$. Then, the algorithm adds to $\mathcal{R}_{new}$ the conclusion of any inference that has in its premises clauses from $\mathcal{R}_{new}$ and from any of the sets $\mathcal{O}_N$, $\mathcal{O}$, $\mathcal{R}$ (line: 7). The algorithm iterates over these three types of inferences until no new clauses can be produced up to variable renaming. When this process terminates, the algorithm returns the function-free subset of $\mathcal{R}_{new} \cup \mathcal{R}$.

It is easy to see that the worst-case complexity of Add is the same with the complexity of the query rewriting system based on the inference system Γ, since Add performs only inference rules of Γ. Next we show the correctness of Algorithm 1.

Theorem 1. *Let $\mathcal{O}$ be an ontology, let $\mathcal{Q}$ be a CQ and Γ be a sound query (resp. axiom)-oriented inference system. Let $\mathcal{R}$ be a UCQ rewriting for $\mathcal{Q}$, $\mathcal{O}$ (resp. $\mathcal{R} = (\mathcal{Q} \cup \mathcal{O}_N)_\Gamma$) and let $\mathcal{O}_N$ be a finite set of clauses. When applied to $\mathcal{O}$, $\mathcal{R}$, $\mathcal{O}_N$ and Γ Algorithm 1 terminates. Let $\mathcal{R}'$ be the set of clauses produced by the algorithm; then, $\mathcal{R}'$ is a UCQ (resp. datalog) rewriting for $\mathcal{Q}, \mathcal{O} \cup \mathcal{O}_N$.*

Proof. First we will show that the algorithm terminates. Since the rewriting of $\mathcal{Q}$ w.r.t. $\mathcal{O} \cup \mathcal{O}_N$ is finite and Γ, on which Algorithm 1 is based on, is sound we conclude that the algorithm does not produce infinite set of clauses up to variable renaming. Also, given that it stops when no new clauses up to variable renaming can be produced we conclude that the algorithm will terminate.

It suffices to prove for each istance $\mathcal{I}$ that $\mathsf{cert}(\mathcal{R}', \mathcal{I}) = \mathsf{cert}(\mathcal{Q}, \mathcal{O} \cup \mathcal{O}_N \cup \mathcal{I})$. It is obvious that $\mathsf{cert}(\mathcal{R}', \mathcal{I}) \subseteq \mathsf{cert}(\mathcal{Q}, \mathcal{O} \cup \mathcal{O}_N \cup \mathcal{I})$, since the algorithm is based on a sound inference system and hence, produces only sound clauses.

Now we will prove that $\mathsf{cert}(\mathcal{Q}, \mathcal{O} \cup \mathcal{O}_N \cup \mathcal{I}) \subseteq \mathsf{cert}(\mathcal{R}', \mathcal{I})$. Let $\mathcal{R}''$ be a UCQ (resp. datalog)-rewriting for $\mathcal{Q}$, $\mathcal{O} \cup \mathcal{O}_N$ resulted from a rewriting algorithm that relies on the query (axiom)-oriented inference system Γ. Since $\mathsf{cert}(\mathcal{Q}, \mathcal{O} \cup \mathcal{O}_N \cup \mathcal{I}) = \mathsf{cert}(\mathcal{R}'', \mathcal{I})$, it suffices to prove that for every clause c s.t. $\mathcal{R}'' \vdash_\Gamma c$ it holds that $\mathcal{R}' \vdash_\Gamma c$. Since $\mathcal{R} \subseteq \mathcal{R}'$, we will prove this only for the clauses resulted from a sequence of inferences that at least one of them contains in its premises clauses from $\mathcal{O}_N$. The proof will be by induction.

Suppose that c is produced by one inference, i.e. that $c_1, \ldots, c_{i-1} \in \mathcal{R}$, $c_i, \ldots, c_{j-1} \in \mathcal{O}$, $c_j, \ldots, c_n \in \mathcal{O}_N$ exist s.t. $\langle c_1, \ldots, c_i, \ldots, c_j, \ldots, c_n, c \rangle$ is an inference by Γ. Then, it is obvious from the structure of the algorithm that either $c \in \mathcal{R}'$ or there is a $c' \in \mathcal{R}'$ s.t. c' is equivalent up to variable renaming with c, i.e. $\mathcal{R}' \vdash_\Gamma c$. Suppose that c is produced after k inferences, i.e. $\mathcal{Q} \cup \mathcal{O} \cup \mathcal{O}_N \vdash_\Gamma c_k$ and $c_1, \ldots, c_{n-1} \in \mathcal{O}$, $c_n, \ldots, c_m \in \mathcal{O}_N$ exist s.t. $\langle c_k, c_1, \ldots, c_n, \ldots, c_m, c \rangle$ is an inference by Γ. Since $c_k \in \mathcal{R}'$, from induction hypothesis, it is easy to conclude from the structure of Algorithm 1 that either $c \in \mathcal{R}'$ or there is a $c' \in \mathcal{R}'$ s.t. c' is equivalent up to variable renaming with c, i.e. $\mathcal{R}' \vdash_\Gamma c$ □

A potential performance bottleneck of Algorithm 1 is that it may perform unnecessary inferences. For example, if a clause c produced from the inference of line 3 is subsumed from another clause of $\mathcal{R}$ or $\mathcal{R}_{new}$ then every inference that has in its premises c or any of its descendants is unnecessary since both c and its descendants are redundant. Hence, in each of the lines 4, 6, 8 a produced clause should be added to $\mathcal{R}_{new}$ only if it is not subsumed from another clause of $\mathcal{R}$ or $\mathcal{R}_{new}$. Also, the performance of the algorithm will be further optimised if we execute the standard subsumption checking algorithm over the input set $\mathcal{R}$ and then serve the resulted set, instead of $\mathcal{R}$, as input to the algorithm.

4 Evaluation

We have implemented Algorithm 1 enhanced with the optimisations described in the previous section in a prototype tool called AddOnto. AddOnto also performs subsumption checking before returning the final rewriting. Our implementation

Table 1. Performance results of AddOnto and Requiem for DL-Lite$_R$ ontologies

	V					S				
	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5
AddOnto	0	0	2	29	6	0	0	0	0	1
Requiem	0	16	32	47	16	1	67	452	904	16,240
	P5					P5X				
	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5
AddOnto	0	0	0	37	696	0	0	33	852	26,481
Requiem	10	20	50	520	13350	10	30	230	5,370	176,774
	U					UX				
	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5
AddOnto	0	0	0	28	1	0	1	1	46	2
Requiem	16	47	94	1,404	5,584	0	78	921	13,291	41,994
	A					AX				
	Q1	Q2	Q3	Q4	Q5	Q1	Q2	Q3	Q4	Q5
AddOnto	3	7	19	39	289	8	1,435	16,085	6,894	-
Requiem	47	47	47	109	452	62	1,888	19,765	15,319	-

relies on the query rewriting system Requiem[9]. We have evaluated AddOnto tool using both benchmark ontologies expressed in DL-Lite$_R$ and a realistic ontology expressed in $\mathcal{ELHI}$.

Regarding DL-Lite$_R$, we used the framework proposed in [14], which consists of eight test ontologies together with a set of five hand-crafted test queries for each ontology. Concerning $\mathcal{ELHI}$, we used an $\mathcal{ELHI}$ fragment (149 rules) of the medical ontology GALEN (39,243 rules), together with five manually constructed queries. All experiments were conducted on an Intel(R) Core (TM) with a 3.20GHz processor and 4GB of RAM.

For the evaluation of our tool we proceeded as follows. Initially we removed a rule r from each test ontology for every query. Next, we transformed the rules of the contracted ontology to clauses obtaining the set $\mathcal{O}$ and we computed the saturation $\mathcal{R}$ (since the inference system Γ that underpins Requiem is axiom-oriented) of $\mathcal{Q} \cup \mathcal{O}$ via Γ. Then, we executed AddOnto with input $\langle \mathcal{O} \setminus \mathcal{O}_N, \mathcal{R}', \mathcal{O}_N, \Gamma \rangle$, where the set $\mathcal{O}_N$ contains the clauses that correspond to the rule r and $\mathcal{R}'$ is $\mathcal{R}$ after the subsumption checking and is computed off-line. This process was repeated for each rule of the ontology, and we recorded the mean time among the total set of its rules. Tables 1, 2 show the average computation time (in ms) for every ontology and query. Since our implementation is based on Requiem we compare our implementation against the standard (non-modified) version of Requiem. For Requiem we measured the time to compute the rewriting of each query w.r.t. the whole respective ontology from scratch. Note that both tools returned rewritings of the same size so we do not present these numbers for brevity.

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

Table 2. Performance results of AddOnto and Requiem for the $\mathcal{ELHI}$ fragment of GALEN ontology

	AddOnto	Requiem
$\mathcal{Q}_1 : Q(x) \leftarrow \mathsf{ErythrocyteCount}(x)$	711	1299
$\mathcal{Q}_2 : Q(x) \leftarrow \mathsf{ErythrocyteSedimentation}(x)$	478	827
$\mathcal{Q}_3 : Q(x) \leftarrow \mathsf{HollowStructure}(x)$	494	883
$\mathcal{Q}_4 : Q(x) \leftarrow \mathsf{HaemoglobinConcentrationProcedure}(x)$	773	895
$\mathcal{Q}_5 : Q(x) \leftarrow \mathsf{WestergrenESRProcedure}(x)$	3,473	14,109

From the results demonstrated in the Tables 1 and 2, we note that in most cases the time required to calculate the new rewriting is negligible. In half of the DL-Lite$_R$ ontologies and queries (Table 1) AddOnto computes a new rewriting almost instantaneously in less than 0.05 seconds. The results obtained for the GALEN ontology were also encouranging (Table 2) and especially for the query $\mathcal{Q}_5$ compared to the corresponding performance of Requiem. Concerning the DL-Lite$_R$ ontologies, a large difference compared to Requiem can be noticed in $P5X$ as well as in ontologies UX and AX which are particularly hard. Note, however, that we were not able to obtain results for ontology AX query Q_5 as Requiem that we based AddOnto on did not terminate after 9 hours. It is clear from both tables that AddOnto outperforms Requiem. This is an expected result since AddOnto is provided with the saturation of $(\mathcal{O} \setminus \{r\}) \cup \mathcal{Q}$ and has to compute only the new inferences resulted from the addition of r.

Concerning the case of the ontology AX and the query Q_5 we noticed in [19] that there is a specific rule ($r_{98} = \mathsf{AssistiveDevice}(x) \rightarrow \mathsf{Device}(x)$) in AX whose removal reduces the size of the rewriting significantly. Hence, we removed this rule from the ontology and we calculated with Requiem a rewriting of Q_5 w.r.t. $AX \setminus \{r_{98}\}$. The output minimal rewriting was computed in 29 seconds and contained 2,546 CQs. Then, we executed AddOnto with $\mathcal{O}_N = \{r_{98}\}$ obtaining, in 17 minutes, the final minimal rewriting of Q_5 w.r.t. AX, which contains 32,921 CQs. This result leads us to the conclusion that the rule r_{98} causes explosion of the size of the rewriting. Also, we conclude that the order by which the rules are taken during the resolution process is crucial for the efficiency of the algorithm.

Although in all cases AddOnto outperforms Requiem, we notice that there are some relatively hard cases for AddOnto. Such cases are the computation of the rewriting of the queries Q_4, Q_5 w.r.t. ontology $P5X$ and of the rewriting of the queries Q_2, Q_3, Q_4 w.r.t. ontology AX. We will further investigate the reasons for these time performances with graphs of the time performance vs the added rule, in order to observe the behaviour of each rule separately.

The graph of $P5X$ ontology that appears in Fig. 1 shows that the time performance increases significantly when the rules r_4 ($\mathsf{AUX}_0(x, y) \rightarrow \mathsf{edge}(x, y)$), r_9 ($\mathsf{AUX}_1(x, y) \rightarrow \mathsf{edge}(x, y)$) and r_{13} ($\mathsf{AUX}_2(x, y) \rightarrow \mathsf{edge}(x, y)$) are added. This happens because, as it was pointed in [19] these rules are crucial for the final form of the output rewriting, as when any of them was removed the size of the rewriting was greatly decreased. This can be easily explained by observing that

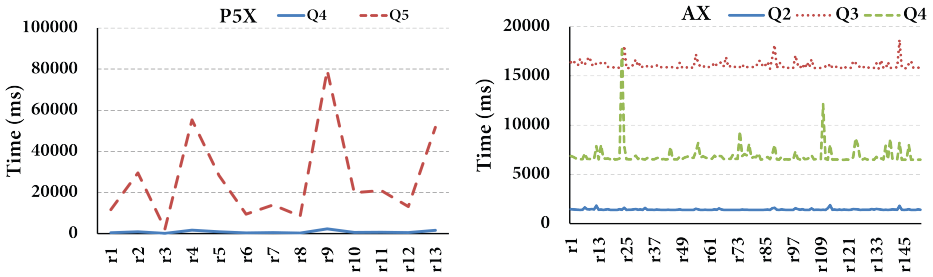

Fig. 1. Time performance of AddOnto for the computation of the minimal rewriting for ontologies $P5X$ and AX for CQs Q_4, Q_5 and Q_2–Q_4 respectively when an rule r added.

the atoms $\mathsf{AUX}_0(x, y)$, $\mathsf{AUX}_1(x, y)$, $\mathsf{AUX}_2(x, y)$ have many descendants in the ontology, while the atom $\mathsf{edge}(x, y)$ appears in all test queries.

Regarding AX ontology, we notice from the corresponding graph of Fig. 1 that for every added rule and every query, the time performance of AddOnto is above a certain value. For instance, in the case of query $\mathcal{Q}_3$ the algorithm requires more than 15 seconds to compute the new rewriting, for any $\mathcal{O}_N$. This is a surprising result since it was shown in [19] that there are rules in AX, like the rule $r_{151} = \mathsf{Ability}(x) \rightarrow \exists y.\mathsf{AUX}_0(x, y))$, that do not affect the rewriting. However, after further investigation of the time results of each subprocess, we noticed that the process of the subsumption checking that is performed in the end of AddOnto is mostly responsible for this behaviour. For instance, in the case of Q_4, when rule r_{151} is added, then 5,386 ms are required only for the subsumption checking process. This can be explained by the large size (3,163 CQs) of the input set to the subsumption checking process.

5 Conclusions

In the current paper we have presented and studied a novel problem in the area of query rewriting. More precisely, we have studied query rewriting of fixed queries over evolved ontologies—that is, over ontologies for which one or more axioms have been added. We presented a practical algorithm which, given a set of clauses $\mathcal{R}$ generated from an input query $\mathcal{Q}$ and an ontology $\mathcal{O}$, and given a set of axioms $\mathcal{O}_N$ to be added to $\mathcal{O}$, computes a rewriting $\mathcal{R}'$ for $\mathcal{Q}$, $\mathcal{O} \cup \mathcal{O}_N$ without recomputing the same clauses. Our algorithm is compatible with rewriting calculi such as those underpinning Requiem, QuOnto, Naya and Rapid. We have implemented and evaluated the algorithm over the rewriting system Requiem and have obtained encouraging results.

Regarding future work we plan to investigate the same problem under the assumption that the new set of axioms may introduce inconsistencies in the Knowledge Base. We also plan to further evaluate our algorithm firstly by using larger and more realistic ontologies and secondly against other state-of-the-art rewriting systems.

References

1. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson and Voronkov [17], pp. 19–99
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., Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The dl-lite family. J. Autom. Reason. 39(3), 385–429 (Oct 2007), http://dx.doi.org/10.1007/s10817-007-9078-x
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. Gonçalves, R.S., Parsia, B., Sattler, U.: Analysing multiple versions of an ontology: A study of the nci thesaurus. In: Rosati, R., Rudolph, S., Zakharyaschev, M. (eds.) Description Logics. CEUR Workshop Proceedings, vol. 745. CEUR-WS.org (2011)
9. Gottlob, G., Orsi, G., Pieris, A.: Ontological queries: Rewriting and optimization. In: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011 (2011)
10. 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)
11. Kondylakis, H., Plexousakis, D.: Ontology evolution without tears. Web Semantics: Science, Services and Agents on the World Wide Web 19(2) (2013)
12. 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)
13. Orsi, G., Pieris, A.: Optimizing query answering under ontological constraints. Proceedings of the VLDB Endowment 4(11), 1004–1015 (2011)
14. Pérez-Urbina, H., Horrocks, I., Motik, B.: Efficient query answering for OWL 2. In: Proc. of the Int. Semantic Web Conference (ISWC2009). Chantilly, VA, USA. (October 2009)
15. 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)
16. 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)
17. Robinson, J.A., Voronkov, A. (eds.): Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press (2001)

18. 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)
19. Tsalapati, E., Stoilos, G., Stamou, G.B., Koletsos, G.: Query rewriting under ontology contraction. In: Krötzsch, M., Straccia, U. (eds.) RR. Lecture Notes in Computer Science, vol. 7497, pp. 172–187. Springer (2012)
20. 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)