Query Rewriting Under Query Refinements

Tassos Venetis[a,*], Giorgos Stoilos[a,*], Giorgos Stamou[a]

[a]School of Electrical and Computer Engineering, National Technical University of Athens, Iroon Polytexneiou 9, 15780 Zografou, Greece

Abstract

Ontologies expressed in Description Logics or extensions of Datalog are gradually used for describing the domain of many research and industrial strength applications. They provide a formal semantically rich and data-independent layer over which user queries can be posed. A prominent technique for query answering in ontology-based applications is *query rewriting*, where the given user query $\mathcal{Q}$ and ontology $\mathcal{O}$ are transformed into a (datalog) program $\mathcal{R}$ that captures the answers of $\mathcal{Q}$ over $\mathcal{O}$ and every database $\mathcal{D}$. In realistic scenarios it is quite often the case that users refine their original query by adding or removing constraints until they produce a final one. In such scenarios, however, all existing systems would compute a new rewriting $\mathcal{R}_i$ for each refined query $\mathcal{Q}_i$ from scratch, discarding any information possibly computed previously. In this paper we study the problem of computing a rewriting for a query $\mathcal{Q}'$ which is a "refinement" of a query $\mathcal{Q}$ by exploiting as much as possible information possibly computed previously for $\mathcal{Q}$. We investigate whether such information is usable when computing a rewriting for $\mathcal{Q}'$ and present detailed algorithms. Finally, we have implemented all proposed algorithms and conducted an extensive experimental evaluation.

Keywords: Ontologies, Description Logics, Query Rewriting, Query Refinement.

1. Introduction

The use of ontologies in research as well as in industrial strength applications is gradually gaining momentum [21, 26, 33]. In such scenarios the data reside in secondary data management systems while ontologies provide a formal specification of the intentional level (knowledge/schema) of the application domain. Then, access to the data is performed via *conjunctive queries* (CQs) and the computed answers reflect both the stored data as well as the knowledge in the ontology providing so-called Ontology Based Data Access (OBDA) [34]. Ontologies also play an important role in a number of different scenarios like data integration [7, 22], biomedical applications [15, 12], and more.

An important family of formalisms for constructing ontologies, primary due to fact that they constitute the logical underpinnings of the Web Ontology Languages OWL [19] and OWL 2 [11] are Description Logics (DLs) [3]. Other prominent ontology languages motivated mostly from the area of deductive databases are fragments of Datalog$^{\pm}$ [6]. Query answering over such ontology languages has extensively been studied in the literature [23, 14, 29] and today there exist languages that are specifically purposed for efficient data access. Prominent examples are DL-Lite [8] and EL [4], which constitute the logical underpinnings of OWL 2 QL [25] and OWL 2 EL [25], respectively.

An important approach for query answering in these languages is via a technique called *query rewriting* [8, 2, 32]. According to this technique a query $\mathcal{Q}$ and an ontology $\mathcal{O}$ are transformed into a program $\mathcal{R}$, typically a datalog program called a *rewriting*, such that the answers of $\mathcal{R}$ over any input data $\mathcal{D}$ (discarding the ontology) are precisely the answers of $\mathcal{Q}$ over $\mathcal{D}$ and $\mathcal{O}$. Such an approach to query answering is interest-

*Corresponding author

Email addresses: avenet@image.ece.ntua.gr (Tassos Venetis), gstoil@image.ece.ntua.gr (Giorgos Stoilos), gstam@cs.ntua.gr (Giorgos Stamou)

ing from a practical perspective because after computing $\mathcal{Q}$ the problem of query answering can be delegated to efficient and scalable (deductive) database and datalog evaluation systems by either directly evaluating $\mathcal{R}$ using off-the-shelf systems [13], by implementing customised engines [27], or by integrating $\mathcal{R}$ in an optimal way into data saturation engines [38].

In the last years many algorithms and systems for computing rewritings have been presented, such as QuOnto [1], Requiem [31], Presto [37], Nyaya [16], Quest [35], Rapid [10], IQAROS [41], and Clipper [13]. All of them will compute for a given fixed query a rewriting by applying (usually in a brute-force manner) a certain set of rewriting rules discarding any information possibly computed for previously processed queries. However, it is quite often the case that user queries alter their initial queries producing new ones which have small differences [30, 20]. Consequently, a query can be refined several times until the user (possibly) finds the intended information.

For example, a user might initially ask to retrieve from a student database all those students that take a specific course using the following conjunctive query:

$$\mathsf{Student}(x) \wedge \mathsf{takesC}(x, y) \rightarrow \mathcal{Q}_A(x)$$

where x, y are variables that need to be matched to actual students and courses from the database, respectively, while Student is a concept atom (unary predicate) and takesC is a role atom (binary predicate). Moreover, the predicate $\mathcal{Q}_A$, called *head* of the query, specifies which matches should be returned to the user. In this query, only the matched students would be returned.

Subsequently, the user might desire to also retrieve the course that each student takes, hence posing the following query:

$$\mathsf{GStudent}(x) \wedge \mathsf{takesC}(x, y) \rightarrow \mathcal{Q}_A(x, y)$$

where now variable y also appears in the predicate $\mathcal{Q}_A$. Yet, our hypothetical user might want to further refine the query and retrieve all those that take a course without necessarily being students, hence, posing the following query:

$$\mathsf{takesC}(x, y) \rightarrow \mathcal{Q}_A(x, y)$$

Finally, he/she can again relax the constrains and retrieve only the people that take a course:

$$\mathsf{takesC}(x, y) \rightarrow \mathcal{Q}_A(x)$$

In our previous work [40, 41] we have studied the problem of computing a rewriting for queries that have been refined by extending them with new atoms. More precisely, given a DL-Lite obtology $\mathcal{O}$, a query $\mathcal{Q}$, a rewriting $\mathcal{R}$ computed for $\mathcal{O}$ and $\mathcal{Q}$ and a new atom α with which the user wants to "extend" $\mathcal{Q}$, we have studied how to compute a rewriting for the extended query by reusing as much as possible the information that has been pre-computed in $\mathcal{R}$. Our study gave rise to a novel query rewriting system based on incremental processing of the query atoms and experimental evaluation showed that the new algorithm is currently one of the most efficient ones.

In the current paper we study all other types of refinements as illustrated previously in our running example. More precisely, for an ontology $\mathcal{O}$, a query $\mathcal{Q}$, and a rewriting $\mathcal{R}$ possibly computed previously for $\mathcal{Q}$ and $\mathcal{O}$ we study how to compute a rewriting for a new query that is obtained from $\mathcal{Q}$ by removing some specific atom of $\mathcal{Q}$, or by removing some of the head variable of $\mathcal{Q}$, or by extending its head variables with new ones, again by exploiting as much as possible the information in previously computed rewriting $\mathcal{R}$.

To the best of our knowledge there exist no previous study of the above problems in the presence of ontologies. A related problem studied in the field of databases is *view adaptation* [18, 24], where the problem is to compute the materialisation of a *re-defined* materialised view. However, in both works, the focus is in updating the data (the materialisation of the view) and, moreover, there are no database constraints (ontological axioms) present.

The rest of the paper is organised as follows. In Section 2 we recapitulate all the necessary terminology as well as relevant definitions that will help us with the rest of the paper.

Subsequently, in Sections 3–5 we study the aforementioned refinement problems. More precisely, in Section 3 we study query rewriting when some head variables have been removed; in Section 4 we study query rewriting when new variables have been added to the head of the current query; and finally, in Section 5 we study query rewriting when an atom from the query has been removed. For each studied problem we present motivating examples, detailed algorithms, and proofs. Subsequently, in Section 6 we study for each refinement problem the possiblity of some optimisations. Next, in Section 7 we present two implementations of all our algorithms, one using the system Rapid and one using Requiem. We also present results of a detailed experimental evaluation comparing them against the classical Rapid and Requiem implementation. Finally, in Section 8 we conclude the paper.

2. Preliminairies

2.1. Existential Rules

We use standard notions of (function-free) term, variable, substitution, ground atom, formula, and entailment (denoted as $\models$) from First-Order Logic [9]. An *instance* I is a finite set of ground atoms. For a finite set of atoms $\{\alpha_1, \ldots \alpha_n\}$, we define $\bigwedge\{\alpha_1, \ldots \alpha_n\}$ to be the formula $\alpha_1 \wedge \ldots \wedge \alpha_n$. For α an atom and σ a substitution, the result of applying σ to α is denoted as $\alpha\sigma$. Also, every substitution σ induces a directed graph $G = \langle V, E\rangle$, where $t \in V$ iff t is a term in σ and $\langle x, t\rangle \in E$ iff $x \mapsto t \in \sigma$.

A *concept atom* is of the form $A(t)$ with A an atomic concept and t a term. A *role atom* is of the form $R(t, t')$ for R an atomic role, and t, t' terms. An *existential rule* r (or just *rule*) [5, 6], often called *axiom* or *clause*, is a sentence of the form:

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

where $\phi(\vec{x},\vec{z})$ and $\psi(\vec{x},\vec{y})$ are conjuncts of function-free 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. Note that, by definition, existential rules are safe–that is, all variables in $\vec{x}$ occur both in the body and the head. If $\vec{y}$ is empty, the rule is called *datalog*. For r a datalog rule, we denote with $\mathsf{bd}(r)$ the set of body atoms of r, and by $\mathsf{hd}(r)$ the single head atom of r. A *datalog program* is a finite set of datalog rules.

Many popular Horn ontology languages, such as DL-Lite$_R$ [8] and $\mathcal{ELHI}$ [32] as well as Datalog$^{\pm}$ [6] can be captured by existential rules. So, in the context of this paper, we will define an *ontology* $\mathcal{O}$ as a finite set of existential rules.

2.2. Queries

A query $\mathcal{Q}$ is a finite set of sentences containing a distinct query predicate Q_A in the head atom. A tuple of constants $\vec{a}$ is a *certain answer* to $\mathcal{Q}$ w.r.t. ontology $\mathcal{O}$ and instance I if the arity of $\vec{a}$ agrees with the arity of Q_A and $\mathcal{O} \cup I \cup \mathcal{Q} \models Q_A(\vec{a})$. We denote with $\mathsf{cert}(\mathcal{Q}, \mathcal{O} \cup I)$ all answers to $\mathcal{Q}$ w.r.t. $\mathcal{O} \cup I$. A query $\mathcal{Q}$ is a *union of conjunctive queries* (UCQ) if it is a set of datalog rules containing Q_A in the head but not in the body. A UCQ $\mathcal{Q}$ is called a *conjunctive query* (CQ) if it has exactly one rule; in this case with $\mathcal{Q}$ we denote the single rule in the CQ.

All the variables that appear in the head of a conjunctive query are called *distinguished* (or *answer*) and are denoted by $\mathsf{avar}(\mathcal{Q})$. For an atom α we use $\mathsf{var}(\alpha)$ to denote the set of its variables; var can be extended to queries in the obvious way. Let $\phi(\vec{x},\vec{z}) \rightarrow Q_A(\vec{x})$ be a CQ, where $\vec{x} = (x_1, \ldots, x_n)$ and let also a vector $\vec{y} = (y_1, \ldots, y_m)$ of variables. Then, by $\phi(\vec{x},\vec{z}) \rightarrow Q_A(\vec{x},\vec{y})$ we denote the new query $\phi(\vec{x},\vec{z}) \rightarrow Q_A(\vec{z})$, where $\vec{z} = (x_1, \ldots, x_n, y_1, \ldots, y_m)$.

Let $\mathcal{Q} = \phi(\vec{x},\vec{z}) \rightarrow Q_A(\vec{x})$, where $\vec{x} = (x_1, \ldots, x_n)$. Let also $j_1, \ldots, j_m$ be a non-empty sequence of positive integers such that $n \geq \max\{j_1, \ldots, j_m\}$. The *projection* of $\mathcal{Q}$ over $j_1, \ldots, j_m$, denoted by $\pi_{j_1,\ldots,j_m}(\mathcal{Q})$, is the new query $\phi(\vec{x},\vec{z}) \rightarrow Q_A(\vec{w})$ where $\vec{w} = (x_{j_1}, \ldots, x_{j_m})$, and $\pi_{j_1,\ldots,j_m}$ is called a *projection operator*.

Given CQs $\mathcal{Q}_1, \mathcal{Q}_2$ with distinguished variables $\vec{x}$ and $\vec{y}$, respectively, we say that $\mathcal{Q}_2$ *subsumes* $\mathcal{Q}_1$ (or that $\mathcal{Q}_2$ is a *subsumer* of $\mathcal{Q}_1$), if there exists a substitution σ from $\mathsf{var}(\mathcal{Q}_2)$ to $\mathsf{var}(\mathcal{Q}_1)$ such that every atom in $\mathcal{Q}_2\sigma$ also appears in $\mathcal{Q}_1$. For a rewriting $\mathcal{R}$ we say that $\mathcal{Q}_1$ is *redundant* in $\mathcal{R}$ if a different query $\mathcal{Q}_2 \in \mathcal{R}$ exists that subsumes it; otherwise it is called *non-redundant* in $\mathcal{R}$. Finally, $\mathcal{Q}_1$ is called *non-subsumer* if it

does not subsume any other query in $\mathcal{R}$.

A CQ $\mathcal{Q}$ is called *connected* if, for all terms t, t', there exists a sequence $t_1, \ldots, t_n$ such that $t_1 = t$, $t_n = t'$ and, for all $1 \leq i < n$, there exists a role R such that $R(t_i, t_{i+1}) \in \mathcal{Q}$. Without loss of generality, we assume that CQs are connected.

2.3. Inferences and Inference Rules

We use standard notions of most general unifier (mgu), (hyper-)resolvent, and clause subsumption from First-Order resolution [9]. A resolution inference rule is an $n + 2$ relation on clauses annotated with a substitution and a list of the relevant sets of literals. The elements of such a relation with the corresponding sets and substitution are written as follows:

$$\frac{C \quad C_1 \ldots C_n}{C'}[\sigma, \Upsilon]$$

where C is a clause called *main premise*, $C_1, \ldots, C_n$ are distinct clauses called *side premises*, and C' is called the *conclusion*. An inference is also denoted by a tuple of the form $\langle C, C_1, \ldots, C_n, C' \rangle$. Given an inference $\langle C, C_1, \ldots, C_n, C' \rangle$, we require C' to be a (hyper-)resolvent of C with $C_1, \ldots, C_n$ via the composition of an mgu θ and a renaming ρ. This effectively ensures that C' does not share variables with any of the premises. Moreover, σ is a mapping from the variables of C to those of C' that establishes a connection between them. More precisely, it is defined as the subset of assignments in $\theta \circ \rho$ mapping variables of C to function-free terms of C'. Furthermore, Υ is a list of pairs of sets of the form $\langle \mathcal{A}_i, \mathcal{B}_i \rangle$ that contains the relevant literals of the inference. More precisely, $\mathcal{A}_i$ are all the literals of C that are unified with literals from the i-th side premise C_i, while $\mathcal{B}_i$ are the literals of C_i that are introduced to C' (after also application of $\theta \circ \rho$). Formally, $C' = [(C \setminus \bigcup_i \mathcal{A}_i)\theta \circ \rho \cup \bigcup_i \mathcal{B}_i]$, where $\mathcal{B}_i \subseteq C_i\theta \circ \rho$. An *inference system* Γ is a collection of inference rules. For a set of clauses N we denote with $\Gamma(N)$ the set of all inferences by Γ having all their premises in N.

2.4. 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 I [8, 32, 16].

Definition 1. A *datalog rewriting* of a CQ $\mathcal{Q}$ with query predicate Q_A w.r.t. ontology $\mathcal{O}$ is a datalog program that can be partitioned into two disjoint sets $\mathcal{R}_D \uplus \mathcal{R}_Q$ such that $\mathcal{R}_D$ does not mention Q_A and $\mathcal{R}_Q$ is a UCQ with query predicate Q_A, and where for each I using only predicates from $\mathcal{O}$ we have

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

In the following we freely use $\mathcal{R}_D$ ($\mathcal{R}_Q$) to denote the datalog (UCQ) part of some rewriting $\mathcal{R}$.

Several techniques and algorithms for computing a rewriting have been presented in the literature [8, 32, 16, 28]. Many of them have been implemented in state-of-the-art systems. Thus, to abstract from the specifics of each system we introduce an abstract notion that can capture most algorithms proposed in the literature.

Definition 2. A *rewriting algorithm* rew for a class $\mathcal{L}$ of existential rules is an algorithm that for each $\mathcal{L}$-ontology $\mathcal{O}$ and CQ $\mathcal{Q}$ proceeds in three phases:

1. It transforms $\mathcal{O} \cup \mathcal{Q}$ into a set of clauses N_1.
2. It uses an inference system Γ to compute a sequence $\langle N_1, \sigma_1, \Upsilon_1 \rangle \triangleright \cdots \triangleright \langle N_i, \sigma_i, \Upsilon_i \rangle$ of tuples that consists of a set of clauses N_i, a substitution σ_i and a list of pairs Υ_i such that for $i = 1$ we have $\sigma_1 = \emptyset$, $\Upsilon_1 = \{\langle \emptyset, \emptyset \rangle\}$. Moreover, for each $i \in \{1, \ldots, m-1\}$, N_{i+1} extends N_i with a conclusion of an inference in $\Gamma(N_i)$ and σ_{i+1}, Υ_{i+1} are the annotations of the specific inference.
3. It returns a tuple $\mathsf{rew}(\mathcal{Q}, \mathcal{O}) = \langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$, where $\mathcal{R} = \mathcal{R}_D \uplus \mathcal{R}_Q$ is a subset of N_m s.t. it is a rewriting for $\mathcal{Q}$ w.r.t. $\mathcal{O}$, and $\rightsquigarrow_{\mathcal{R}}$ is the relation defined as follows: for $x \in \mathsf{var}(\mathcal{Q})$, $\mathcal{Q}' \in \mathcal{R}_Q$ and $y \in \mathsf{var}(\mathcal{Q}')$, we have $x \rightsquigarrow_{\mathcal{R}} y$ iff $x(\sigma_1 \circ \ldots \circ \sigma_j) = y$, where j is the smallest integer such that $\mathcal{Q}' \in N_j$.

A rewriting algorithm is called *Q-oriented* if all $\mathcal{Q}_i \in \mathcal{R}_Q$ different from $\mathcal{Q}$ are produced from some $\mathcal{Q}_j \in \mathcal{R}_Q$ by an inference of the form $\langle \mathcal{Q}_j, C_1, \ldots, C_n, \mathcal{Q}_i \rangle$.

Most existing rewriting algorithms are either resolution-based or can be cast in the framework of resolution, and hence

they can be captured by our definition. Throughout the paper we will additionally assume that rewritings are computed by Q-oriented rewriting algorithms. Most well-known systems, like PerfectRef, Nyaya, and Rapid, are based on Q-oriented algorithms, hence our results are quite general. However, there are systems, like Requiem, that are not. For example, in Requiem a query $\mathcal{Q}_i \in \mathcal{R}_Q$ might be computed from a query $\mathcal{Q}_j \in \mathcal{R}_Q$ using more than one inferences involving intermediate clauses that contain functional terms. For example, for $\mathcal{O} = \{A(x) \rightarrow \exists y.R(x,y) \wedge C(y), R(x,y) \rightarrow S(x,y)\}$ and $\mathcal{Q} = S(x,y) \wedge C(y) \rightarrow \mathcal{Q}_A(x)$ it will proceed as follows:

- First, it will transform $\mathcal{O}$ into the following set of clauses:

$$A(x) \rightarrow R(x, f(x)) \quad (1)$$
$$A(x) \rightarrow C(f(x)) \quad (2)$$
$$R(x,y) \rightarrow S(x,y) \quad (3)$$

  where f is a new skolem function.

- Then, it will apply (binary) resolution with free selection and one possible derivation could be the following: from $\mathcal{Q}$ and (2) derive $\mathcal{Q}^1_f = S(x, f(x)) \wedge A(x) \rightarrow \mathcal{Q}_A(x)$; from $\mathcal{Q}^1_f$ and (3) derive $\mathcal{Q}^2_f = R(x, f(x)) \wedge A(x) \rightarrow \mathcal{Q}_A(x)$; and, from $\mathcal{Q}^2_f$ and (1) derive $\mathcal{Q}' = A(x) \rightarrow \mathcal{Q}_A(x)$. The rewriting for $\mathcal{Q}$ w.r.t. $\mathcal{O}$ is the set $\mathcal{R} = \{\mathcal{Q}, \mathcal{Q}'\}$.

Clearly, Requiem is not Q-oriented as $\mathcal{Q}'$ is derived from $\mathcal{Q}$ by several steps involving intermediate non-function-free clauses.

Our results can be applied to systems such as Requiem in the following way: We can modify all algorithms and definitions to accept as input the limit N_m of Definition 2 instead of just the clauses in the function-free subset of N_m. For example, in the previous case the input should be the set $N_3 = \{\mathcal{Q}, \mathcal{Q}^1_f, \mathcal{Q}^2_f, \mathcal{Q}'\}$. With this modification all our results apply.

3. Rewriting Under Answer Variable Removals

In this section we study the problem of computing a rewriting for a query whose distinguished variables have been reduced, by exploiting as much as possible a precomputed rewriting for the original query. The following example illustrates the key ideas behind the algorithm we will present next.

Example 3. Consider the following ontology $\mathcal{O}$ which describes courses and professors:

$$\mathsf{teachesGradC}(x,y) \rightarrow \mathsf{teachesC}(x,y) \quad (4)$$
$$\mathsf{Professor}(x) \rightarrow \exists y.\mathsf{teachesGradC}(x,y) \quad (5)$$

Consider also the query $\mathcal{Q}_1 = \mathsf{teachesC}(x_1, y_1) \rightarrow Q_A(x_1, y_1)$ which retrieves pairs of tutors and courses. The set $\mathcal{R} = \{\mathcal{Q}_1, \mathcal{Q}_2\}$, where $\mathcal{Q}_2 = \mathsf{teachesGradC}(x_2, y_2) \rightarrow Q_A(x_2, y_2)$ is a rewriting of $\mathcal{Q}_1$ w.r.t. $\mathcal{O}$. More precisely, $\mathcal{Q}_2$ is the conclusion of an inference with premises query $\mathcal{Q}_1$ and clause (4), and annotation $\Upsilon = \{\langle\{\mathsf{teachesC}(x_1, y_1)\}, \{\mathsf{teachesGradC}(x_2, y_2)\}\rangle\}$. Moreover, $\rightsquigarrow_{\mathcal{R}} = \{\langle x_1, x_2\rangle, \langle y_1, y_2\rangle\}$ that states that during computation of $\mathcal{R}$ variable x_1 was renamed to x_2 and y_1 to y_2.

Assume that after evaluating $\mathcal{Q}_1$ we want to retrieve only those that teach some course without also retrieving the specific course, i.e., we pose the query $\mathcal{Q}'_1 = \mathsf{teachesC}(x_1, y_1) \rightarrow Q_A(x_1)$ which is the projection of $\mathcal{Q}_1$ over variable x_1 (the first variable), i.e., $\mathcal{Q}'_1 = \pi_1(\mathcal{Q}_1)$. The set $\mathcal{R}' = \{\mathcal{Q}'_1, \mathcal{Q}'_2, \mathcal{Q}'_3\}$, where $\mathcal{Q}'_2 = \mathsf{teachesGradC}(x_2, y_2) \rightarrow Q_A(x_2)$ and $\mathcal{Q}'_3 = \mathsf{Professor}(x_3) \rightarrow Q_A(x_3)$, is a rewriting of $\mathcal{Q}'_1$ w.r.t. $\mathcal{O}$, and $\rightsquigarrow_{\mathcal{R}'} = \{\langle x_1, x_2\rangle, \langle y_1, y_2\rangle, \langle x_1, x_3\rangle\}$.

The rewriting $\mathcal{R}'$ can be computed using any state-of-the-art system. However, each such system would apply its inferences rules over $\mathcal{Q}'_1$ and $\mathcal{O}$ from scratch without taking into account that a significant amount of information has already been computed for query $\mathcal{Q}_1$, and $\mathcal{Q}'_1$ differs only by one variable. In fact, $\mathcal{Q}'_2$ (a member of $\mathcal{R}'$) can be computed directly from $\mathcal{R}$ as the projection of $\mathcal{Q}_2$ over the first variable, i.e., $\mathcal{Q}'_2 = \pi_1(\mathcal{Q}_2)$. Actually, if $\mathcal{Q}_1$ and $\mathcal{Q}_2$ have been evaluated over the database then we can also directly compute the evaluation of $\mathcal{Q}'_1$ and $\mathcal{Q}'_2$ by a projection over the results returned for $\mathcal{Q}_1$ and $\mathcal{Q}_2$, respectively.

However, we can observe that query $\mathcal{Q}'_3$ of $\mathcal{R}'$ cannot be computed using the above simple operation. Instead, one needs to exploit the inference rules of a rewriting system to compute it. More precisely, $\mathcal{Q}'_3$ can be computed by performing an inference using, e.g., systems like PerfectRef or Requiem, with

premises query $\mathcal{Q}'_2$ and clause (5). Intuitively, this is because after projection, variable y_2 appears only once and hence new inferences are "activated". ◊

The above example suggests that given a query $\mathcal{Q}$ and a rewriting $\mathcal{R}$ of $\mathcal{Q}$ w.r.t. some ontology $\mathcal{O}$, a rewriting for a query $\mathcal{Q}'$ that is a projection of $\mathcal{Q}$ (i.e., $\mathcal{Q}' = \pi_{j_1,\ldots,j_n}(\mathcal{Q})$ for some $\pi_{j_1,\ldots,j_n}$) can be computed from $\mathcal{R}$ by performing two key operations: first, project all queries in $\mathcal{R}$ using $\pi_{j_1,\ldots,j_n}$, and, second, possibly apply additional inferences over the projected queries in order to compute the missing ones.

In general, we can apply the inference rules of a rewriting system exhaustively over all projected queries, however, this could potentially affect performance. In contrast our algorithm checks whether a new inference has been activated using the following function.

Definition 4. Let $\mathcal{O}$ be an ontology, let $\mathcal{Q}$ be a conjunctive query, and let $\pi_{j_1,\ldots,j_n}$ be a projection operator. Then function $\mathsf{needsRewriting}(\mathcal{Q}, \pi_{j_1,\ldots,j_n}, \mathcal{O})$ returns true if there exists an inference with query $\pi_{j_1,\ldots,j_n}(\mathcal{Q})$ as a main premise and annotations σ_π, Υ_π and there exists $\langle \mathcal{A}_\pi, \mathcal{B}_\pi \rangle \in \Upsilon_\pi$ such that for every inference with query $\mathcal{Q}$ as a main premise and every $\langle \mathcal{A}, \mathcal{B} \rangle \in \Upsilon$ either $\mathcal{A} \neq \mathcal{A}_\pi$ or $\mathcal{B} \neq \mathcal{B}_\pi$; otherwise it returns false.

Our algorithm for computing a rewriting for projected queries using a precomputed rewriting is depicted in Algorithm 1. It accepts as input the ontology $\mathcal{O}$, the precomputed rewriting $\mathcal{R}$ for a query $\mathcal{Q}$ w.r.t. $\mathcal{O}$, the relation $\rightsquigarrow_\mathcal{R}$, and a projection operator over $\mathcal{Q}$. Internally, it also uses a standard rewriting algorithm rew to apply inferences whenever required. The algorithm proceeds as follows. First, it initialises a new rewriting $\mathcal{R}'$ with $\mathcal{R}_D$ and a relation $\rightsquigarrow_{\mathcal{R}'}$ with $\rightsquigarrow_\mathcal{R}$. Then, it iterates over all queries $\mathcal{Q}_i$ in $\mathcal{R}$ and computes their projection $\pi_{j_1,\ldots,j_n}(\mathcal{Q}_i)$ adding it to the result. Moreover, for each projected query it checks if further rewriting is required and if so it calls a rewriting algorithm rew with inputs the projected query and the ontology $\mathcal{O}$.

Theorem 5. *Let $\mathcal{Q}$ be a CQ, let $\mathcal{O}$ be an ontology, let $\pi_{j_1,\ldots,j_n}$ be a projection over $\mathcal{Q}$, and let $\mathcal{R}$ and $\rightsquigarrow_\mathcal{R}$ be the output of a rewriting algorithm when applied over $\mathcal{Q}$ and $\mathcal{O}$. When applied to $\mathcal{O}, \mathcal{R}$, $\rightsquigarrow_\mathcal{R}$, and $\pi_{j_1,\ldots,j_n}$ Algorithm 1 terminates. Let $\langle \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$ be the output of the algorithm; then $\mathcal{R}'$ is a rewriting of $\pi_{j_1,\ldots,j_n}(\mathcal{Q})$ w.r.t. $\mathcal{O}$.*

Algorithm 1 $\mathsf{RemoveVars}(\mathcal{O}, \mathcal{R}, \rightsquigarrow_\mathcal{R}, \pi_{j_1,\ldots,j_n})$

input: An ontology $\mathcal{O}$, a rewriting $\mathcal{R}$ with partition $\mathcal{R}_D \uplus \mathcal{R}_Q$ and a relation $\rightsquigarrow_\mathcal{R}$ both of which are the output of a rewriting algorithm for a CQ $\mathcal{Q}$ w.r.t. $\mathcal{O}$, and a projection operator $\pi_{j_1,\ldots,j_n}$.

1: Initialise $\mathcal{R}' := \mathcal{R}_D$ and $\rightsquigarrow_{\mathcal{R}'} := \rightsquigarrow_\mathcal{R}$
2: **for all** $\mathcal{Q}_i \in \mathcal{R}_Q$ **do**
3: Add $\mathcal{Q}'_i$ to $\mathcal{R}'$, where $\mathcal{Q}'_i := \pi_{j_1,\ldots,j_n}(\mathcal{Q}_i)$
4: **if** $\mathsf{needsRewriting}(\mathcal{Q}_i, \pi_{j_1,\ldots,j_n}, \mathcal{O})$ **then**
5: $\langle \mathcal{R}_n, \rightsquigarrow_{\mathcal{R}_n} \rangle := \mathsf{rew}(\mathcal{Q}'_i, \mathcal{O})$
6: $\mathcal{R}' := \mathcal{R}' \cup \mathcal{R}_n$ and $\rightsquigarrow_{\mathcal{R}'} := \rightsquigarrow_{\mathcal{R}'} \cup \rightsquigarrow_{\mathcal{R}_n}$
7: **end if**
8: **end for**
9: **return** $\langle \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$

Proof. Termination follows by the fact that Algorithm 1 iterates over each member of a finite set ($\mathcal{R}_Q$) only once and because the rewriting algorithm rew terminates.

Consider, $\mathcal{O}, \langle \mathcal{R}, \rightsquigarrow_\mathcal{R} \rangle, \mathcal{Q}$, and $\pi_{j_1,\ldots,j_n}$ to be the input of Algorithm 1, where $\langle \mathcal{R}, \rightsquigarrow_\mathcal{R} \rangle$ is computed by some rewriting algorithm rew. Assume now that the output of Algorithm 1 is the pair $\langle \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$ and assume also that $\langle \mathcal{R}_i, \rightsquigarrow_{\mathcal{R}_i} \rangle$ is the rewriting computed after i steps of the rewriting algorithm rew when applied to $\mathcal{Q}', \mathcal{O}$.

Correctness is strongly based on the following property, which we show using induction over i:

(VR): For all $i \geq 0$ and for all $\mathcal{Q}'_l \in \mathcal{R}_i$ there exists $\mathcal{Q}_l \in \mathcal{R}$ such that for $\mathcal{Q}^\pi_l = \pi_{j_1,\ldots,j_n}(\mathcal{Q}_l)$ one of the following conditions holds:

1. $\mathcal{Q}^\pi_l$ subsumes $\mathcal{Q}'_l$, or
2. $\mathsf{needsRewriting}(\mathcal{Q}_l, \pi_{j_1,\ldots,j_n}, \mathcal{O})$ returns true and some $\mathcal{Q}''_l \in \mathsf{rew}(\mathcal{Q}^\pi_l, \mathcal{O})$ subsumes $\mathcal{Q}'_l$.

Base Case ($i = 0$): At the beginning $\mathcal{R}_0 = \{\mathcal{Q}'\}$ where $\mathcal{Q}' = \pi_{j_1,\ldots,j_n}(\mathcal{Q})$, hence Property (VR) is trivially satisfied.

Induction Step: Assume that Property (VR) holds at step i for each $\mathcal{Q}'_l \in \mathcal{R}_i$—that is, either Item 1. or 2. of Property (VR) holds. Next, assume that at step $i + 1$ algorithm rew applies an inference step on some query $\mathcal{Q}'_l$ producing (function-free) query $\mathcal{Q}'_{l+1}$. We now examine separately the two cases of the induction hypothesis:

1. $\mathcal{Q}^\pi_l$ subsumes $\mathcal{Q}'_l$. By known properties of subsumption in First-Order theorem proving [9] this implies that either $\mathcal{Q}^\pi_l$ subsumes $\mathcal{Q}'_{l+1}$ or an inference is applicable to $\mathcal{Q}^\pi_l$ and the result, call it $\mathcal{Q}^\pi_0$, subsumes $\mathcal{Q}'_{l+1}$. If the former case is true then we are done (there exists $\mathcal{Q}_l \in \mathcal{R}$ such that $\mathcal{Q}^\pi_l$ subsumes $\mathcal{Q}'_{l+1}$). Assume that the latter is true. Recall that $\mathcal{Q}_l$ contains the same body atoms as $\mathcal{Q}^\pi_l$. Since rewriting algorithms perform inferences using only the body atoms of the query, we have that the same resolution inference is applicable to $\mathcal{Q}_l$ producing query $\mathcal{Q}_0$. However, since $\mathcal{Q}_l$ contains more distinguished variables than $\mathcal{Q}^\pi_l$, then $\mathcal{Q}_0$ might contain functional terms in those variables of the head that have been projected out in $\mathcal{Q}^\pi_l$. If it does not then $\mathcal{Q}_0 \in \mathcal{R}$ and clearly $\mathcal{Q}^\pi_0 = \pi_{j_1,\ldots,j_n}(\mathcal{Q}_0)$ as the only difference between $\mathcal{Q}_0$ and $\mathcal{Q}^\pi_0$ are the head variables that have been projected out. If it does contain functionals in the head this implies that $\mathcal{Q}_0 \notin \mathcal{R}$ since a Q-oriented algorithm produces only function-free clauses. But then, all the latter imply that function needsRewriting$(\mathcal{Q}_l, \pi_{j_1,\ldots,j_n}, \mathcal{O})$ returns true and then we clearly have $\mathcal{Q}^\pi_0 \in \mathsf{rew}(\mathcal{Q}^\pi_l, \mathcal{O})$. Hence, in either case Property (VR) holds.

2. needsRewriting$(\mathcal{Q}_l, \pi_{j_1,\ldots,j_n}, \mathcal{O})$ returns true and there exists query $\mathcal{Q}''_l \in \mathsf{rew}(\mathcal{Q}^\pi_l, \mathcal{O})$ that subsumes $\mathcal{Q}'_l$. Again, either $\mathcal{Q}''_l$ subsumes $\mathcal{Q}'_{l+1}$ or an inference is applicable to $\mathcal{Q}''_l$ and the result, call it $\mathcal{Q}''_0$ subsumes $\mathcal{Q}''_l$. In either case Item 2. would again hold as we will either have $\mathcal{Q}''_l \in \mathsf{rew}(\mathcal{Q}^\pi_l, \mathcal{O})$ or $\mathcal{Q}''_0 \in \mathsf{rew}(\mathcal{Q}^\pi_l, \mathcal{O})$.

This concludes the proof of Property (VR).

Assume now that when applied to $\mathcal{Q}', \mathcal{O}$, the rew algorithm terminates after n steps computing $\mathcal{R}_n$. Property (VR) implies that upon termination of Algorithm 1 for each query $\mathcal{Q}_l \in \mathcal{R}_n$ a query $\mathcal{Q}'$ in $\mathcal{R}'$ would exists that subsumes $\mathcal{Q}_l$. □

A potential performance bottleneck of Algorithm 1 is in line 5 where it executes the sub-routine rew. If the function is called a large number of times then performance can be adversely affected, hence any well-behaved implementation should attempt to minimise this number. One approach, which has been implemented in the prototype tool we will present in the evaluation section, is the following. Let $\mathcal{Q}'_i$ be the projection of a query $\mathcal{Q}_i$ and assume that $\mathcal{Q}_i$ satisfies the conditions in line 4. Instead of immediately rewriting $\mathcal{Q}'_i$ we add it to a set toRew. Then, when the outer for-loop terminates (lines 2–7), we remove from toRew all queries that are redundant in toRew and hence, subsequently, execute rew only over the non-redundant queries. Furthermore, a rewriting that has already been computed for some $\mathcal{Q}_i \in$ toRew can again be used to prune queries from toRew for which rew has not yet been executed. Finally, a good heuristic is to check if the projection of the query $\mathcal{Q}$ for which $\mathcal{R}$ has been computed is in toRew. In that case the algorithm can break and execute rew only over $\pi_{j_1,\ldots,j_n}(\mathcal{Q})$ as this obviously constructs the desired rewriting.

4. Rewriting Under Answer Variable Additions

In this section we study the problem of computing a rewriting for a query whose distinguished variables have been extended with new ones, by exploiting as much as possible a precomputed rewriting for the original query. As in the previous section we first give an illustrative example.

Example 6. Consider ontology $\mathcal{O}$, query $\mathcal{Q}'_1$, rewriting $\mathcal{R}' = \{\mathcal{Q}'_1, \mathcal{Q}'_2, \mathcal{Q}'_3\}$ and relation $\rightsquigarrow_{\mathcal{R}'}$ from Example 3. Assume now that we want to "extend" this query and also retrieve the respective courses, i.e., we want to compute a rewriting for the query $\mathcal{Q}_1$ of Example 3 which, as shown, consists of the set $\mathcal{R} = \{\mathcal{Q}_1, \mathcal{Q}_2\}$ that can be computed from $\mathcal{Q}_1$ and $\mathcal{O}$ using any rewriting algorithm.

However, we can again note that instead of applying a rewriting algorithm completely from scratch we can exploit the previously computed rewriting $\mathcal{R}'$ of $\mathcal{Q}'_1$ w.r.t. $\mathcal{O}$. More precisely, $\mathcal{Q}_1$ can be computed form $\mathcal{Q}'_1$ by replacing the head atom $Q_A(x_1)$ of $\mathcal{Q}'_1$ with the head atom $Q_A(x_1, y_1)$ (y_1 is the new distinguished variable that is used to retrieve the respective taught courses). Moreover, $\mathcal{Q}'_2$ can be computed by replacing the head atom $Q_A(x_2)$ of $\mathcal{Q}'_2$ with the head atom $Q_A(x_1, y_1)\tau$, where $\tau = \{x_1 \mapsto x_2, y_1 \mapsto y_2\}$ is computed from $\rightsquigarrow_{\mathcal{R}'}$. Finally, we note that the query $\mathsf{Professor}(x_3) \rightarrow Q_A(x_3, y_1)$ produced form $\mathcal{Q}_3$ using the above process is not entailed form $\mathcal{O} \cup \mathcal{Q}$, hence no query should be obtained from $\mathcal{Q}_3$. ◊

The above example suggests that given a query $\mathcal{Q}$ with head atom $Q_A(\vec{x})$, a rewriting $\mathcal{R}$ of $\mathcal{Q}$ w.r.t. some ontology $\mathcal{O}$ and a new vector $\vec{y}$ of distinguished variables such that $\vec{y} \subseteq \mathsf{var}(\mathcal{Q})$, to compute a rewriting for the query $\bigwedge \mathsf{bd}(\mathcal{Q}) \rightarrow Q_A(\vec{x}, \vec{y})$ our algorithm needs to proceed as follows: for each $\mathcal{Q}_i \in \mathcal{R}$ if $\vec{y}\tau \subseteq \mathsf{var}(\mathcal{Q}_i)$, where τ is a proper variable renaming constructed using $\rightsquigarrow_{\mathcal{R}}$, then extend the distinguished variables of $\mathcal{Q}_i$ by adding the new list $\vec{y}\tau$; otherwise discard $\mathcal{Q}_i$. This process is made precise in the following definition.

Definition 7. Let $\mathcal{O}$ be an ontology, let $\mathcal{Q}$ be a CQ with $\vec{x}$ as distinguished variables, and let $\langle \mathcal{R}_D \uplus \mathcal{R}_Q, \rightsquigarrow_{\mathcal{R}} \rangle$ be the output of a rewriting algorithm when applied over $\mathcal{Q}, \mathcal{O}$. Furthermore, let $\vec{z}$ be a vector of variables such that $\vec{z} \subseteq \mathsf{var}(\mathcal{Q})$, let $\mathcal{Q}' \in \mathcal{R}_Q$, and let $\tau := \{x \mapsto y \mid x \in \mathsf{var}(\mathcal{Q}), y \in \mathsf{var}(\mathcal{Q}')$ and $x \rightsquigarrow_{\mathcal{R}} y\}$. The *extension* of $\mathcal{Q}'$ by $\vec{z}$ is the formula $\mathcal{Q}''$ defined as follows:

$$\mathcal{Q}'' = \bigwedge \mathsf{bd}(\mathcal{Q}') \rightarrow Q_A(\vec{x}, \vec{z})\tau \tag{6}$$

The extension is called *safe* if $\vec{z}\tau \subseteq \mathsf{var}(\mathcal{Q}')$.

In Example 6 we were able to compute a rewriting for the extension $\mathcal{Q}_1$ of $\mathcal{Q}'_1$ by computing all safe extensions of the queries in $\mathcal{R}'$ without applying any inferences. This, however, is in general not possible and, as the following example shows, depends on the input rewriting.

Example 8. Consider the following ontology and query:

$$\mathcal{O} = \{A(x) \rightarrow \exists y.R(x, y)\}; \quad \mathcal{Q}_1 = A(x_1) \wedge R(x_1, y_1) \rightarrow Q_A(x_1).$$

The set $\mathcal{R} = \{\mathcal{Q}_1, \mathcal{Q}_2\}$, where $\mathcal{Q}_2 = A(x_2) \rightarrow Q_A(x_2)$, is a rewriting of $\mathcal{Q}_1$ w.r.t. $\mathcal{O}$. However, we can observe that $\mathcal{Q}_2$ subsumes $\mathcal{Q}_1$, hence $\mathcal{Q}_1$ can be removed and $\mathcal{R}' = \{\mathcal{Q}_2\}$ is also a rewriting of $\mathcal{Q}_1$ w.r.t. $\mathcal{O}$.

Now suppose that we want to extend $\mathcal{Q}_1$ by also adding variable y_1 to the list of distinguished variables and that we want to compute a rewriting for the extended query, i.e., $\mathcal{Q}'_1 = A(x_1) \wedge R(x_1, y_1) \rightarrow Q_A(x_1, y_1)$. Using $\mathcal{R}'$ the only query that can be extended by y_1 is $\mathcal{Q}_2$, however, its extension is not safe. Hence, using $\mathcal{R}'$ the empty set is obtained. In contrast using $\mathcal{R}$ the extension of $\mathcal{Q}_1$ by y_1 is the query $\mathcal{Q}'_1$ and the singleton set $\{\mathcal{Q}'_1\}$ the desired rewriting. ◊

Intuitively, the issue in the previous example is that although $\mathcal{Q}_2$ subsumes $\mathcal{Q}_1$ in $\mathcal{R}$, the extensions of query $\mathcal{Q}_2$ by some variable might not be safe and hence not part of the output. Consequently, the extensions of $\mathcal{Q}_1$ won't be redundant and should be added to the result. This is only possible if $\mathcal{Q}_1$ has not been removed from $\mathcal{R}$ due to $\mathcal{Q}_2$. If redundant queries are removed then our algorithm should additionally apply inferences over the extended queries as well as over the extension of the input query $\mathcal{Q}$. However, this in general implies that a rewriting procedure should be applied from scratch cancelling out any possible benefits by using the precomputed information. Next, we formalise a property of input rewritings that is sufficient for computing a rewriting using only safe extensions.

Definition 9. Let $\mathcal{Q}$ be a CQ, let $\mathcal{O}$ be an ontology, and let $\mathcal{R}$ be a rewriting of $\mathcal{Q}$ w.r.t. $\mathcal{O}$ with partition $\mathcal{R}_D \uplus \mathcal{R}_Q$ computed by a rewriting algorithm based on an inference system Γ. We say that $\mathcal{R}$ is *inference-closed* for $\mathcal{Q}, \mathcal{O}$ if $\mathcal{Q} \in \mathcal{R}$ and if, additionally, for every $\mathcal{Q}_1 \in \mathcal{R}_Q$ if a query $\mathcal{Q}_2$ is derivable from $\mathcal{O} \cup \mathcal{Q}_1$ by Γ, then also $\mathcal{Q}_2 \in \mathcal{R}_Q$.

It is easy to see that the rewriting $\mathcal{R}'$ from Example 8 is not inference-closed since $\mathcal{Q}_1 \notin \mathcal{R}'$, while $\mathcal{R}$ is inference-closed.

Intuitively, a rewriting $\mathcal{R}$ is inference-closed if all queries that are produced by the algorithm during the computation of $\mathcal{R}$ are also members of $\mathcal{R}$. This usually implies that $\mathcal{R}$ is computed without using optimisations like subsumption deletion which

Algorithm 2 ExtendRewritingForNewVars($\mathcal{Q}, \mathcal{R}, \rightsquigarrow_{\mathcal{R}}, \vec{y}$)

input: A query $\mathcal{Q}$, a rewriting $\mathcal{R}$ with partition $\mathcal{R}_D \uplus \mathcal{R}_Q$, and a relation $\rightsquigarrow_{\mathcal{R}}$ that are the output of a rewriting algorithm for $\mathcal{Q}$ w.r.t. an ontology $\mathcal{O}$ and a vector of variables $\vec{y}$ such that $\vec{y} \subseteq \mathsf{var}(\mathcal{Q})$.

1: Initialise $\mathcal{R}' := \mathcal{R}_D$ and $\rightsquigarrow_{\mathcal{R}'} := \rightsquigarrow_{\mathcal{R}}$
2: **for all** $\mathcal{Q}_i \in \mathcal{R}_Q$ **do**
3: $\quad \mathcal{Q}'_i :=$ the extension of $\mathcal{Q}_i$ by $\vec{y}$
4: $\quad$ **if** $\mathcal{Q}'_i$ is safe **then**
5: $\quad\quad$ Add $\mathcal{Q}'_i$ to $\mathcal{R}'$
6: $\quad$ **end if**
7: **end for**
8: **return** $\langle \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$

can remove clauses that are derived by Γ, or that such optimisations have been turned-off. Although this would typically imply that the process of computing a rewriting that is later on an input to our designed techniques is inefficient (as subsumption optimisation has been turned-off), there are systems that can compute such rewritings very efficiently [41]. Moreover, the property of inference-closure has also been turned out to be relevant and useful for other problems in query rewriting, like query rewriting under ontology revisions [39].

Algorithm 2 presents our approach for computing a rewriting for queries extended with new distinguished variables summarising the previous discussion. The algorithm accepts as an input a query $\mathcal{Q}$, a rewriting $\mathcal{R}$ of some $\mathcal{Q}$ w.r.t. $\mathcal{O}$, a relation $\rightsquigarrow_{\mathcal{R}}$, and a vector of variables $\vec{y}$. It then iterates over all CQs $\mathcal{Q}_i \in \mathcal{R}_Q$ and computes the extension $\mathcal{Q}'_i$ of $\mathcal{Q}_i$ by $\vec{y}$ as in Definition 7. If the extension is safe then $\mathcal{Q}'_i$ is added to $\mathcal{R}'$ (line 5), while if not $\mathcal{Q}'_i$ is discarded.

Theorem 10. *Let $\mathcal{O}$ be an ontology, let $\mathcal{Q}$ be a CQ let $\vec{y}$ be a tuple of variables such that $\vec{y} \subseteq \mathsf{var}(\mathcal{Q})$ and let $\mathcal{R}$ be an inference-closed rewriting for $\mathcal{Q}$ w.r.t. some ontology $\mathcal{O}$ together with $\rightsquigarrow_{\mathcal{R}}$. When applied to $\mathcal{R}, \rightsquigarrow_{\mathcal{R}}$, and $\vec{y}$ Algorithm 2 terminates. Let $\mathcal{R}'$ be the output of the algorithm; then $\mathcal{R}'$ is an inference-closed rewriting of the extension of $\mathcal{Q}$ by $\vec{y}$ w.r.t. $\mathcal{O}$.*

Proof. Termination follows by the fact that Algorithm 2 iterates over a finite set of queries from $\mathcal{R}$ only once.

Consider, $\mathcal{O}$, $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$, $\mathcal{Q}$, and $\vec{y}$, to be the input of Algorithm 2, where $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$ is computed by some rewriting algorithn rew and $\vec{y}$ is a tuple of variables such that $\vec{y} \subseteq \mathsf{var}(\mathcal{Q})$.

Assume now that the output of Algorithm 2 is the pair $\langle \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$ and assume also that $\langle \mathcal{R}_i, \rightsquigarrow_{\mathcal{R}_i} \rangle$ is the rewriting computed after i steps of the rewriting algorithm rew when applied to the extension of $\mathcal{Q}$ by $\vec{y}$, call it $\mathcal{Q}'$ in the following.

Correctness is an immediate consequence of the following property, which we show using induction over i:

> (VE): For all $i \geq 0$ and for all $\mathcal{Q}'_l \in \mathcal{R}_i$ there exists $\mathcal{Q}_l \in \mathcal{R}$ such that the extension $\mathcal{Q}_e$ of $\mathcal{Q}_l$ by $\vec{y}$ is safe and subsumes $\mathcal{Q}'_l$.

Base Case ($i = 0$): At the beginning $\mathcal{R}_0 = \{\mathcal{Q}'\}$ where $\mathcal{Q}'$ is the extension of $\mathcal{Q}$ by $\vec{y}$, hence property (VE) is clearly satisfied.

Induction Step: Assume that Property (VE) holds at step i for each $\mathcal{Q}'_l \in \mathcal{R}_i$. Moreover, assume that in step $i+1$ query $\mathcal{Q}'_{l+1}$ is produced from $\mathcal{Q}'_l$ by an application of an inference step of the Q-oriented rewriting algorithm rew. Since $\mathcal{Q}_e$ subsumes $\mathcal{Q}'_l$ by known properties of subsumption we have that either $\mathcal{Q}_e$ subsumes $\mathcal{Q}'_{l+1}$ or an inference is applicable on $\mathcal{Q}_e$ and the result $\mathcal{Q}'_e$ subsumes $\mathcal{Q}'_{l+1}$. If the former is the case then we are done. Hence, assume that the latter is the case. Since all variables in $\vec{y}$ appear in the head of $\mathcal{Q}_e$ the inference that produces $\mathcal{Q}'_e$ must be such that it does not eliminate or map variables from $\vec{y}$ to functional terms. Now recall that $\mathcal{Q}_l$ is like $\mathcal{Q}_e$ but the latter contains all variables $\vec{y}$ in the head. Hence, the same inference step is applicable to $\mathcal{Q}_l$ producing $\mathcal{Q}_0$. Moreover, as noted before, the variables $\vec{y} \subseteq \mathsf{var}(\mathcal{Q}_l)$ are not affected in this inference step and hence we must have $\vec{y} \subseteq \mathsf{var}(\mathcal{Q}_0)$. Thus, the extension of $\mathcal{Q}_0$ by $\vec{y}$ is safe and clearly the extension is the query $\mathcal{Q}'_e$. Moreover, since $\mathcal{R}$ is inference-closed we also have $\mathcal{Q}_0 \in \mathcal{R}$. Consequently, a query $\mathcal{Q}_0$ exists in $\mathcal{R}$ such that its extension by $\vec{y}$ is safe and it subsumes $\mathcal{Q}'_{l+1}$.

This concludes the proof of Property (VE). Assume that, when applied to $\mathcal{Q}', \mathcal{O}$, the rew algorithm terminates after n

steps computing $\mathcal{R}_n$. Property (VE) implies that upon termination of Algorithm 2 for each query $\mathcal{Q}_l \in \mathcal{R}_n$ a query $\mathcal{Q}'$ in $\mathcal{R}'$ would exists that subsumes $\mathcal{Q}_l$.

Finally, it is clear that the output produced by the algorithm is also inference-closed. □

5. Rewriting Under Atom Removals

In this section we study the problem of computing a rewriting for a query some of the body atoms of which have been removed, by again exploiting as much as possible a previously computed rewriting for the input query. To simplify the presentation we only consider the case where one atom is being removed; removing more atoms can be solved using this algorithm. The following example illustrates some of the key ideas behind the algorithm we will present next.

Example 11. Consider the ontology $\mathcal{O}$ given next:

$$\mathsf{GradStudent}(x) \rightarrow \mathsf{Student}(x), \tag{7}$$

$$\mathsf{TennisPlayer}(x) \rightarrow \mathsf{Athlete}(x) \tag{8}$$

and consider also the query $\mathcal{Q}_1 = \mathsf{Student}(x_1) \wedge \mathsf{Athlete}(x_1) \rightarrow Q_A(x_1)$. The set $\mathcal{R} = \{\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4\}$, where $\mathcal{Q}_i$ are as defined next, is a rewriting for $\mathcal{Q}_1$ w.r.t. $\mathcal{O}$ computed using systems like PerfectRef, Rapid, and Requiem:

$$\begin{aligned}
\mathcal{Q}_2 &= \mathsf{Student}(x_2) \wedge \mathsf{TennisPlayer}(x_2) \rightarrow Q_A(x_2),\\
\mathcal{Q}_3 &= \mathsf{GradStudent}(x_3) \wedge \mathsf{Athlete}(x_3) \rightarrow Q_A(x_3),\\
\mathcal{Q}_4 &= \mathsf{GradStudent}(x_4) \wedge \mathsf{TennisPlayer}(x_4) \rightarrow Q_A(x_4)
\end{aligned}$$

Moreover, $\rightsquigarrow_{\mathcal{R}} = \{\langle x_1, x_2\rangle, \langle x_1, x_3\rangle, \langle x_1, x_4\rangle\}$, which denotes how variable x_1 relates to variables x_2, x_3, and x_4.

More precisely, we have the following inferences and respective annotations $\Upsilon_i = \{\langle \mathcal{A}_i, \mathcal{B}_i\rangle\}$:

$$\begin{aligned}
\langle \mathcal{Q}_1, (8), \mathcal{Q}_2\rangle \quad &\text{with} \quad \mathcal{A}_2 = \{\mathsf{Athlete}(x_1)\} \text{ and}\\
&\qquad\ \ \mathcal{B}_2 = \{\mathsf{TennisPlayer}(x_2)\}\\
\langle \mathcal{Q}_1, (7), \mathcal{Q}_3\rangle \quad &\text{with} \quad \mathcal{A}_3 = \{\mathsf{Student}(x_1)\} \text{ and}\\
&\qquad\ \ \mathcal{B}_3 = \{\mathsf{GradStudent}(x_3)\}\\
\langle \mathcal{Q}_2, (7), \mathcal{Q}_4\rangle \quad &\text{with} \quad \mathcal{A}_4 = \{\mathsf{Student}(x_2)\} \text{ and}\\
&\qquad\ \ \mathcal{B}_4 = \{\mathsf{GradStudent}(x_4)\}\\
\langle \mathcal{Q}_3, (8), \mathcal{Q}_4\rangle \quad &\text{with} \quad \mathcal{A}_5 = \{\mathsf{Athlete}(x_3)\} \text{ and}\\
&\qquad\ \ \mathcal{B}_5 = \{\mathsf{TennisPlayer}(x_4)\}
\end{aligned}$$

Assume now that subsequently the user wants to retrieve all individuals that are students—that is, issue the query $\mathcal{Q}'_1 = \mathsf{Student}(x_5) \rightarrow Q_A(x_5)$. It can be verified that $\mathcal{R}' = \{\mathcal{Q}'_1, \mathcal{Q}'_2\}$, where $\mathcal{Q}'_2 = \mathsf{GraduateStudent}(x_6) \rightarrow Q_A(x_6)$ is a rewriting of $\mathcal{Q}'_1$ w.r.t. $\mathcal{O}$ which can be computed in a standard way.

However, we can observe that a rewriting for $\mathcal{Q}'_1$ w.r.t. $\mathcal{O}$ can be directly computed using the information that has been materialised previously for $\mathcal{R}$. More precisely, a query equivalent to $\mathcal{Q}'_1$ can be obtained simply by discarding $\mathsf{Athlete}(x_1)$ from query $\mathcal{Q}_1$, while a query equivalent to $\mathcal{Q}'_2$ can be obtained by removing from $\mathcal{Q}_3$ the atom $\mathsf{Athlete}(x_1)\tau_3$ where $\tau_3 = \{x_1 \mapsto x_3\}$ is defined using $\rightsquigarrow_{\mathcal{R}}$. Finally, we note that queries $\mathcal{Q}_2$ and $\mathcal{Q}_4$ need to be discarded since for any mapping σ we have $\mathsf{Athlete}(x_1)\sigma \notin \mathsf{bd}(\mathcal{Q}_2)$ and $\mathsf{Athlete}(x_1)\sigma \notin \mathsf{bd}(\mathcal{Q}_4)$. Intuitively, this is because the removed atom $\mathsf{Athlete}(x_1)$ "participated" in the inference $\langle \mathcal{Q}_1, (8), \mathcal{Q}_2\rangle$, i.e., $\mathsf{Athlete}(x_1) \in \mathcal{A}_2$, and in the inference $\langle \mathcal{Q}_3, (8), \mathcal{Q}_4\rangle$, i.e., $\mathsf{Athlete}(x_1)\tau_3 \in \mathcal{A}_5$, where $\tau_3 = \{x_1 \mapsto x_3\}$. Hence, in the reduced query these inferences are not possible. (Note that $\mathcal{Q}_4$ is also produced by the inference $\langle \mathcal{Q}_2, (7), \mathcal{Q}_4\rangle$; however, since $\mathcal{Q}_2$ is discarded this inference should also be discarded). ◊

The previous example suggests that given a rewriting $\mathcal{R}$ for a query $\mathcal{Q}$, a key operation in computing a rewriting for a query $\mathcal{Q}'$ such that $\mathsf{bd}(\mathcal{Q}') = \mathsf{bd}(\mathcal{Q}) \setminus \alpha$ is by "removing" atom $\alpha\tau$ from each CQ $\mathcal{Q}_i \in \mathcal{R}$ such that $\alpha\tau \in \mathsf{bd}(\mathcal{Q}')$, where τ is a proper mapping defined using $\rightsquigarrow_{\mathcal{R}}$; otherwise $\mathcal{Q}_i$ should be discarded. Moreover, it shows that the algorithm can also exploit

additional information produced during the computation of a rewriting $\mathcal{R}$, like the sets $\mathcal{A}_i$, in order to decide if an inference is still possible after the removal of some atom.

The following example illustrates a second technical case that can arise when computing the rewriting of a reduced query, and which again requires using additional input information.

Example 12. Consider the following ontology and query:

$$\begin{aligned}\mathcal{O} &= \{\mathsf{Student}(x) \rightarrow \exists y.\mathsf{takesCourse}(x,y),\\ &\quad\ \mathsf{GradStudent}(x) \rightarrow \mathsf{Student}(x)\}\\ \mathcal{Q}_1 &= \mathsf{Student}(x_1) \wedge \mathsf{takesCourse}(x_1,y_1) \rightarrow Q_A(x_1)\end{aligned}$$

A rewriting for $\mathcal{Q}_1$ w.r.t. $\mathcal{O}$ consists of the set $\mathcal{R} = \{\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3\}$, where $\mathcal{Q}_2 = \mathsf{Student}(x_2) \rightarrow Q_A(x_2)$ and $\mathcal{Q}_3 = \mathsf{GradStudent}(x_3) \rightarrow Q_A(x_2)$. More precisely, $\mathcal{Q}_2$ is produced by an inference with main premise $\mathcal{Q}_1$ and $\Upsilon_2 = \{\langle \mathcal{A}_2, \mathcal{B}_2\rangle\}$ where $\mathcal{A}_2 = \{\mathsf{takesCourse}(x_1,y_1)\}$ and $\mathcal{B}_2 = \{\mathsf{Student}(x_2)\}$, and $\mathcal{Q}_3$ is produced by an inference with $\mathcal{Q}_2$ as a main premise and $\Upsilon_3 = \{\langle \mathcal{A}_3, \mathcal{B}_3\rangle\}$, where $\mathcal{A}_3 = \{\mathsf{Student}(x_2)\}$ and $\mathcal{B}_3 = \{\mathsf{GradStudent}(x_3)\}$. Moreover, we have $\rightsquigarrow_{\mathcal{R}} = \{\langle x_1, x_2\rangle, \langle x_1, x_3\rangle\}$.

Consider now that we want to remove atom $\alpha = \mathsf{Student}(x_1)$ and thus compute a rewriting for $\mathcal{Q}'_1 = \mathsf{takesCourse}(x_1,y_1) \rightarrow Q_A(x_1)$ w.r.t. $\mathcal{O}$, like, e.g., the rewriting $\mathcal{R}' = \{\mathcal{Q}'_1, \mathcal{Q}_2, \mathcal{Q}_3\}$.

Suppose that we want to compute $\mathcal{R}'$ using our approach. First, we remove atom $\mathsf{Student}(x_1)$ from $\mathcal{Q}_1$ and thus obtain $\mathcal{Q}'_1$. Subsequently, for $\tau = \{x_1 \mapsto x_2\}$ we can remove $\mathsf{Student}(x_1)\tau$ from $\mathcal{Q}_2$; however, this would not produce a valid rule. Instead, we observe that atom $\mathsf{Student}(x_1)\tau$ appears in $\mathcal{Q}_2$ because it has been added by an inference with $\mathcal{Q}_1$ as the main premise and over an atom of $\mathcal{Q}_1$ that is "different" than the one that is being removed, i.e., $\mathcal{A}_2 = \{\mathsf{takesCourse}(x_1,y_1)\} \not\ni \mathsf{Student}(x_1)\tau$ and $\mathsf{Student}(x_1)\tau \in \mathcal{B}_2$. Intuitively, this implies that the construction of this part by the inference system is independent from the removal of atom α. Hence, our algorithm should "copy" the respective queries, i.e., $\mathcal{Q}_2$ and $\mathcal{Q}_3$ (the latter produced from $\mathcal{Q}_2$) to the result, constructing the desired rewriting. ◊

A similar situation arises when we have an inference of the form $\langle \mathcal{Q}_1, C, \mathcal{Q}_2\rangle$ where $\mathcal{Q}_1 = R(x,y) \wedge R(z,y) \rightarrow Q_A(x)$, $C = A(x) \rightarrow \exists y.R(x,y)$, $\mathcal{Q}_2 = A(x) \rightarrow Q_A(x)$ and $\mathcal{A}_1 = \{R(x,y), R(z,y)\}$, while the atom that is being removed is $R(z,y)$. Although $R(z,y) \notin \mathsf{bd}(\mathcal{Q}_2)$ and $R(z,y) \in \mathcal{A}_1$, i.e., the removed atom participates in the inference that produced $\mathcal{Q}_2$, the inference can still be performed even after removal, and hence the algorithm should again copy $A(x) \rightarrow Q_A(x)$ to the result. In both this and the case of Example 12 the intuition is that there are queries which are "independent" from the removal of an atom and hence these should be copied to the result. This is formalised next.

Definition 13. Let $\mathcal{Q}$ be a CQ, let $\mathcal{O}$ be an ontology, and let $\mathcal{R} = \mathcal{R}_D \uplus \mathcal{R}_Q$ be a rewriting for $\mathcal{Q}$ w.r.t. $\mathcal{O}$ returned by a rewriting algorithm using an inference system Γ. A query $\mathcal{Q}' \in \mathcal{R}_Q$ is called *independent* from α if either conditions hold:

- $\mathcal{Q}'$ is the conclusion of an inference from some $\mathcal{Q}'' \in \mathcal{R}_Q$ with annotation Υ such that for $\tau'' := \{x \mapsto y \mid x \in \mathsf{var}(\mathcal{Q}), y \in \mathsf{var}(\mathcal{Q}''), x \rightsquigarrow_{\mathcal{R}} y\}$ we have $\alpha\tau'' \in \mathsf{bd}(\mathcal{Q}'')$ and for some $\langle \mathcal{A}, \mathcal{B}\rangle \in \Upsilon$ and $\tau' := \{x \mapsto y \mid x \in \mathsf{var}(\mathcal{Q}), y \in \mathsf{var}(\mathcal{Q}'), x \rightsquigarrow_{\mathcal{R}} y\}$ we have $\mathcal{A} \supset \{\alpha\tau''\}$ or $\alpha\tau' \in \mathcal{B}$, or

- there exists $\mathcal{Q}_j \in \mathcal{R}_Q$ such that $\mathcal{Q}_j$ is independent from α and $\mathcal{Q}'$ is derivable from $\mathcal{O} \cup \{\mathcal{Q}_j\}$ by Γ.

In Example 12, query $\mathcal{Q}_2$ is independent from $\mathsf{Student}(x_1)$; $\mathcal{Q}_2$ is derived by an inference with main premise $\mathcal{Q}_1$, $\mathsf{Student}(x_1) \in \mathsf{bd}(\mathcal{Q}_1)$ and for $\tau = \{x_1 \mapsto x_2\}$ and $\Upsilon_2 = \{\langle \mathcal{A}_2, \mathcal{B}_2\rangle\}$ the annotation of the inference (cf. also Example 12) we have $\mathsf{Student}(x_1)\tau \in \mathcal{B}_2$. Moreover, $\mathcal{Q}_3$ is also independent from $\mathsf{Student}(x_1)$, since $\mathcal{Q}_3$ is derived from $\mathcal{Q}_2$ which as shown before is independent from α.

As the following example illustrates, the operations we have illustrated before might not be enough to compute a rewriting for the new (reduced) query. In addition, like in the case of query projection, there are cases that we need to apply additional inferences over the reduced queries.

Example 14. Consider the following ontology and query:

$$\mathcal{O} = \{A(x) \rightarrow \exists y.R(x,y)\}; \quad \mathcal{Q}_1 = R(x_1,y_1) \wedge B(y_1) \rightarrow Q_A(x_1)$$

The set $\mathcal{R} = \{\mathcal{Q}_1\}$ is a rewriting of $\mathcal{Q}_1$ w.r.t. $\mathcal{O}$.

Consider now that we want to compute a rewriting for $\mathcal{Q}'_1 = R(x'_1,y'_1) \rightarrow Q_A(x'_1)$ using the approach illustrated above. This would yield the set $\mathcal{R}' = \{\mathcal{Q}'_1\}$ which is clearly not a rewriting for $\mathcal{Q}'_1$ w.r.t. $\mathcal{O}$. More precisely, for the instance $I = \{A(a)\}$ we have $\mathsf{cert}(\mathcal{Q}'_1, \mathcal{O} \cup I) = \{a\}$, while $\mathsf{cert}(\mathcal{R}', I) = \emptyset$.

To compute a rewriting for $\mathcal{Q}'_1$ we need to further apply the inference rules of a rewriting algorithm on query $\mathcal{Q}'_1$. This would produce query $\mathcal{Q}'_2 = A(x'_2) \rightarrow Q_A(x'_2)$ from $\mathcal{Q}'_1$ and it can then be verified that $\mathcal{R}' = \{\mathcal{Q}'_1, \mathcal{Q}'_2\}$ is a rewriting for $\mathcal{Q}'_1, \mathcal{O}$. ◇

Again we rely on the following function to check the cases where additional inferences need to be applied.

Definition 15. Let $\mathcal{Q}$ be a CQ, let α be an atom of $\mathcal{Q}$ and let $\mathcal{O}$ be an ontology. Then, function $\mathsf{needsRewriting}(\mathcal{Q}, \alpha, \mathcal{O})$ returns true if there exists inference with query $\bigwedge \mathsf{bd}(\mathcal{Q}) \setminus \{\alpha\tau\} \rightarrow \mathsf{hd}(\mathcal{Q})$ as a main premise and annotations $\sigma_\alpha, \Upsilon_\alpha$ and there exists $\langle \mathcal{A}_\alpha, \mathcal{B}_\alpha \rangle \in \Upsilon_\alpha$ such that for every inference with $\mathcal{Q}$ as a main premise and annotations σ, Υ and every $\langle \mathcal{A}, \mathcal{B} \rangle \in \Upsilon$ either $\mathcal{A} \neq \mathcal{A}_\alpha$ or $\mathcal{B} \neq \mathcal{B}_\alpha$. Otherwise it returns false.

Finally, similar to the case of extending queries with new answer variables, the following example illustrates that the input rewriting needs to be inference-closed, otherwise one might need to exhaustively apply inferences over all reduced queries.

Example 16. Consider $\mathcal{O}$ and $\mathcal{Q}$ from Example 8 as well as the inference-closed rewriting $\mathcal{R} = \{\mathcal{Q}_1, \mathcal{Q}_2\}$, where $\mathcal{Q}_2 = A(x_2) \rightarrow Q_A(x_2)$ and the non-inference-closed one $\mathcal{R}' = \{\mathcal{Q}_2\}$.

Now suppose that we want to remove atom $\alpha = A(x_1)$, i.e., we want to compute a rewriting for $\mathcal{Q}'_1 = R(x_1,y_1) \rightarrow Q_A(x_1)$. One such rewriting computed using any state of the art system or algorithm is $\mathcal{R}'' = \{\mathcal{Q}'_1, \mathcal{Q}_2\}$. However, we can easily see that using the method illustrated above we cannot obtain $\mathcal{R}''$ from $\mathcal{R}'$; however, it can be computed from $\mathcal{R}$. ◇

Algorithm 3 presents our approach for computing a rewriting for a reduced ontology by exploiting previously materialised information and is based on the intuitions outlined before. The algorithm accepts as input an ontology $\mathcal{O}$, a query $\mathcal{Q}$ a rewriting $\mathcal{R}$ of $\mathcal{Q}$ w.r.t. $\mathcal{O}$ and an atom $\alpha \in \mathcal{Q}$. It also relies on a typical query rewriting algorithm rew in order to apply rewriting whenever required.

Algorithm 3 $\mathsf{RemoveAtom}(\mathcal{O}, \mathcal{Q}, \mathcal{R}, \rightsquigarrow_{\mathcal{R}}, \alpha)$

input: An ontology $\mathcal{O}$, a query $\mathcal{Q}$, a rewriting $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$ of $\mathcal{Q}$ w.r.t. $\mathcal{O}$ with partition $\mathcal{R}_D \uplus \mathcal{R}_Q$, and an atom $\alpha \in \mathsf{bd}(\mathcal{Q})$.

1: Initialise $\mathcal{R}' := \mathcal{R}_D$
2: $\rightsquigarrow_{\mathcal{R}'} := \{\langle x, y \rangle \mid x \rightsquigarrow_{\mathcal{R}} y \text{ and } x \in \mathsf{var}(\mathsf{bd}(\mathcal{Q}) \setminus \{\alpha\})\}$
3: Let a proj. operator π to include a position k for all $\mathsf{avar}(\mathcal{Q})$
4: **if** $y \in \mathsf{var}(\alpha)$ with $y \in \mathsf{avar}(\mathcal{Q}) \wedge y \notin \mathsf{var}(\mathsf{bd}(Q) \setminus \alpha)$ **then**
5: Remove from π each position j_k s.t. $x_{j_k} = y$
6: **end if**
7: **for** $\mathcal{Q}_i \in \mathcal{R}_Q$ **do**
8: $\tau := \{x \mapsto y \mid x \in \mathsf{var}(\mathcal{Q}), y \in \mathsf{var}(\mathcal{Q}_i) \text{ and } x \rightsquigarrow_{\mathcal{R}} y\}$
9: **if** $\mathcal{Q}_i$ is independent from α **then**
10: Add $\mathcal{Q}_i$ to $\mathcal{R}'_Q$
11: **else if** $\alpha\tau \in \mathsf{bd}(\mathcal{Q}_i)$ **then**
12: $\mathcal{Q}'_i := \pi(\bigwedge \mathsf{bd}(\mathcal{Q}_i) \setminus \{\alpha\tau\} \rightarrow \mathsf{hd}(\mathcal{Q}_i))$
13: Add $\mathcal{Q}'_i$ to $\mathcal{R}'_Q$
14: **if** $\mathsf{needsRewriting}(\mathcal{Q}_i, \alpha\tau, \mathcal{O})$ **then**
15: $\langle \mathcal{R}_n, \rightsquigarrow_{\mathcal{R}_n} \rangle := \mathsf{rew}(\mathcal{Q}'_i, \mathcal{O})$
16: $\mathcal{R}' := \mathcal{R}' \cup \mathcal{R}_n$ and $\rightsquigarrow_{\mathcal{R}'} := \rightsquigarrow_{\mathcal{R}'} \cup \rightsquigarrow_{\mathcal{R}_n}$
17: **end if**
18: **end if**
19: **end for**
20: **return** $\langle \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$

The algorithm first initialises a new rewritng $\mathcal{R}' := \mathcal{R}_D$ and the relation $\rightsquigarrow_{\mathcal{R}'}$ as $\rightsquigarrow_{\mathcal{R}}$ but restricted to the variables that appear in $\mathsf{bd}(\mathcal{Q}) \setminus \alpha$. Before proceeding as illustrated above, the algorithm first checks if the atom that is being removed contains a distinguished variable of $\mathcal{Q}$ that does not appear in any other atom of $\mathcal{Q}$ (line 4). In this case, besides removing the atom from each query, the algorithm also needs to delete this variable from the head of the reduced query. This is accomplished by constructing a proper projection operator π.

Subsequently, the algorithm enters the main loop. It iterates over each query $\mathcal{Q}_i \in \mathcal{R}_Q$ and creates an appropriate mapping τ of the variables of $\mathcal{Q}_i$ w.r.t. the variables of $\mathcal{Q}$ (line 8). Next, it checks if $\mathcal{Q}_i$ is independent from α (line 9) and if this is the case it adds $\mathcal{Q}_i$ to the output. Otherwise if α appears in the body of $\mathcal{Q}_i$ (line 11) it creates the new query $\mathcal{Q}'_i := \bigwedge \mathsf{bd}(\mathcal{Q}_i) \setminus \{\alpha\tau\} \rightarrow \mathsf{hd}(\mathcal{Q}_i)$ (line 12) and adds it to the output. Finally, it uses function needsRewriting to check if $\mathcal{Q}'_i$ needs to be further rewritten and if so it appends the produced rewriting to the output (lines 14–17).

Theorem 17. *Let $\mathcal{Q}$ be a CQ, let $\mathcal{O}$ be an ontology, let α be an atom such that $\alpha \in \mathsf{bd}(\mathcal{Q})$ and let $\mathcal{R}, \rightsquigarrow_{\mathcal{R}}$ be the output of a rewriting algorithm when applied to $\mathcal{Q}, \mathcal{O}$ such that $\mathcal{R}$ is inference-closed. Let $\mathcal{R}'$ be the rewriting returned by Algorithm 3 when applied to $\mathcal{O}, \mathcal{R}, \rightsquigarrow_{\mathcal{R}}$, and α; then $\mathcal{R}'$ is an inference-closed rewriting of $\bigwedge \mathsf{bd}(\mathcal{Q}) \setminus \{\alpha\} \rightarrow \mathsf{hd}(\mathcal{Q})$ w.r.t. $\mathcal{O}$.*

Proof. Termination follows by the fact that Algorithm 3 iterates over each member of a finite set ($\mathcal{R}_Q$) and because the rewriting algorithm rew terminates.

Consider, $\mathcal{O}$, $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$, $\mathcal{Q}$, and α to be the input of Algorithm 3, where $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$ is an inference-closed rewriting computed by an algorithm rew and $\alpha \in \mathsf{bd}(\mathcal{Q})$.

Assume now that the output of Algorithm 3 is the pair $\langle \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$ and assume also that $\langle \mathcal{R}_i, \rightsquigarrow_{\mathcal{R}_i} \rangle$ is the rewriting computed after i steps of the rewriting algorithm rew when applied to $\mathcal{Q}' = \bigwedge \mathsf{bd}(\mathcal{Q}) \setminus \{\alpha\} \rightarrow \mathsf{hd}(\mathcal{Q})$ and $\mathcal{O}$.

Completeness follows by the following property, which we show using induction over i. To simplify notation and without loss of generality we assume that no projection is needed.

(AR): For all $i \geq 0$ and $\mathcal{Q}'_l \in \mathcal{R}_i$ there exists $\mathcal{Q}_l \in \mathcal{R}$ such that one of the following conditions hold:

1. $\mathcal{Q}_l$ is independent from α and subsumes $\mathcal{Q}'_l$, or
2. for τ as in line 8 of Algorithm 3 we have $\alpha\tau \in \mathsf{bd}(\mathcal{Q}_l)$ and for $\mathcal{Q}^r_l = \bigwedge \mathsf{bd}(\mathcal{Q}_l) \setminus \{\alpha\tau\} \rightarrow \mathsf{hd}(\mathcal{Q}_l)$ either of the following hold:
   (a) $\mathcal{Q}^r_l$ subsumes $\mathcal{Q}'_l$, or
   (b) needsRewriting$(\mathcal{Q}_l, \alpha, \mathcal{O})$ returns true and some $\mathcal{Q}''_l \in \mathsf{rew}(\mathcal{Q}^r_l, \mathcal{O})$ subsumes $\mathcal{Q}'_l$.

Base Case ($i = 0$): At the begining $\mathcal{R}_0 = \{\mathcal{Q}'\}$ where $\mathcal{Q}' = \bigwedge \mathsf{bd}(\mathcal{Q}) \setminus \{\alpha\} \rightarrow \mathsf{hd}(\mathcal{Q})$, hence, Property (AR) is satisfied.

Induction Step: Assume that Property (AR) holds at step i for each $\mathcal{Q}'_l \in \mathcal{R}_i$—that is, either Item 1., Item 2(a), or Item 2(b) of Property (AR) hold. Next, assume that at step $i + 1$ algorithm rew applies an inference step on some query $\mathcal{Q}'_l$ producing $\mathcal{Q}'_{l+1}$. We now examine seperately all the cases of the induction hypothesis:

1. $\mathcal{Q}_l$ is independend from α and it subsumes $\mathcal{Q}'_l$. By known properties of subsumption this implies that either $\mathcal{Q}_l$ subsumes $\mathcal{Q}'_{l+1}$ or an inferece is applicable to $\mathcal{Q}_l$ and the result, call it $\mathcal{Q}_0$, subsumes $\mathcal{Q}'_{l+1}$. If the former case is true then we are done. If the second case is true then again we are done because we will have that $\mathcal{Q}_0 \in \mathcal{R}$ and, moreover, by Definition 13 $\mathcal{Q}_0$ is also independent from α (it is produced by a query that is independent from α).

2. $\alpha\tau \in \mathsf{bd}(\mathcal{Q}_l)$, where $\tau := \{x \mapsto y \mid x \in \mathsf{var}(\mathcal{Q}), y \in \mathsf{var}(\mathcal{Q}_l)$ and $x \rightsquigarrow_{\mathcal{R}} y\}$ and either $\mathcal{Q}^r_l$ subsumes $\mathcal{Q}'_l$ or needsRewriting$(\mathcal{Q}_l, \alpha, \mathcal{O})$ returns true and there exists query $\mathcal{Q}''_l \in \mathsf{rew}(\mathcal{Q}^r_l, \mathcal{O})$ that subsumes $\mathcal{Q}'_l$. We examine the two cases sepereatelly:

   (a) $\mathcal{Q}^r_l$ subsumes $\mathcal{Q}'_l$. This implies that either $\mathcal{Q}^r_l$ subsumes $\mathcal{Q}'_{l+1}$ or an inference is applicable to $\mathcal{Q}^r_l$ and the result, call it $\mathcal{Q}^r_0$ subsumes $\mathcal{Q}'_{l+1}$. If the former case is true then we are done. Assume that the latter is true. Recall that $\mathcal{Q}_l$ contains the same body atoms as $\mathcal{Q}^r_l$ except for $\alpha\tau$. Since rewriting algorithms perform inferences using only the body atoms, we have that all the inferences that are applied to $\mathcal{Q}^r_l$ can also be applied to $\mathcal{Q}_l$ producing $\mathcal{Q}_{l+1}$. For these inferences $\bigwedge \mathsf{bd}(\mathcal{Q}_{l+1}) \setminus \{\alpha\tau''\} \rightarrow \mathsf{hd}(\mathcal{Q}_{l+1})$ subsumes $\mathcal{Q}'_{l+1}$, where $\tau'' = \{x \mapsto y \mid x \in \mathsf{var}(\mathcal{Q}), y \in \mathsf{var}(\mathcal{Q}_{l+1})$ and $x \rightsquigarrow_{\mathcal{R}} y\}$. However, in $\mathcal{Q}_l$ an inference can also be applied to $\alpha\tau$. Finally in case an inference is applied on an atom

different from $\alpha\tau$ that cannot be applied to $\mathcal{Q}_l$ function needsRewriting$(\mathcal{Q}_l, \alpha, \mathcal{O})$ returns true and then we clearly have $\mathcal{Q}_0^r \in \mathsf{rew}(\mathcal{Q}_l^r, \mathcal{O})$. Hence, Property (AR) holds.

(b) Similar to case 2. from proof of Theorem 5.

This concludes the proof of Property (AR). Assume that, when applied to $\mathcal{Q}', \mathcal{O}$, the rew algorithm terminates after n steps computing $\mathcal{R}_n$. Property (AR) implies that upon termination of Algorithm 2 for each query $\mathcal{Q}_l \in \mathcal{R}_n$ a query $\mathcal{Q}'$ in $\mathcal{R}'$ would exists that subsumes $\mathcal{Q}_l$. □

A potential performance bottleneck of Algorithm 3 is in line 9 where it checks if a query is independent from α by checking the conditions of Definition 13. As shown in the definition this involves checking various derivation relations between members of $\mathcal{R}_\mathcal{Q}$. To efficiently implement this step we can assume that the rewriting algorithm used to compute the input rewriting $\mathcal{R}$ can also provide us with additional information that is produced during the computation of $\mathcal{R}$. More precisely, besides $\mathcal{R}$ and $\rightsquigarrow_\mathcal{R}$ we can assume that a rewriting algorithm also returns (or provides acess to) a relation $\mathcal{H}$ and a mapping $\mathcal{S}$ such that if $\mathcal{Q}'$ is the conclusion of an inference with main premise $\mathcal{Q}$ and annotation Υ, then $\langle \mathcal{Q}, \mathcal{Q}' \rangle \in \mathcal{H}$ and $\mathcal{S}(\langle \mathcal{Q}, \mathcal{Q}' \rangle) = \Upsilon$. Our algorithm can exploit such additional information to efficiently check if a query $\mathcal{Q}_i$ is derivable from $\mathcal{O} \cup \{\mathcal{Q}_j\}$ by, e.g., checking if $\mathcal{Q}_i$ is reachable from $\mathcal{Q}_j$ in the relation $\mathcal{H}$. Moreover, it can use $\mathcal{S}$ to have access to the various annotations without needing to execute inferences. Regarding line 15 and the sub-routine rew the same discussion as in the end of Section 3 applies.

6. Rewriting Minimisation

Besides computing a rewriting, an important issue in rewriting-based query answering is evaluating the computed set of sentences over a (deductive) database. As stated in Theorems 10 and 17, Algorithms 2 and 3 return an inference-closed rewriting $\mathcal{R}'$, which implies that $\mathcal{R}'$ might contain redundant queries. As explained in the previous sections these additional queries are important if we wish to subsequently call these algorithms again to add more distinguished variables or delete further atoms from the query. However, when we want to evaluate $\mathcal{R}'$ over a database, it is preferable to compute a mimimal set $\mathcal{R}''$ that is equivalent to $\mathcal{R}'$. One important, data-independent technique to mimimise $\mathcal{R}'$ is by applying the well-known redundancy elimination algorithm removeRedundant$(\mathcal{R}')$ which checks if a query $\mathcal{Q}_i \in \mathcal{R}'$ is subsumed by a query $\mathcal{Q}_j \in \mathcal{R}'$ (and vice versa) and then removes the proper query [31].[1]

Hence, before evaluating the returned rewriting we can use this procedure to eliminate the redundant queries. However, as it has been shown before [10, 41] this method might not perform well in practice because it consists of two nested for-loops over the (potentially large) computed rewriting $\mathcal{R}'$. Let $\mathcal{R}$ be the input to Algorithms 2 or 3 and $\mathcal{R}'$ their respective output. In [41], studying a different problem, we have shown how prior knowledge about the subsumption relations in $\mathcal{R}$ can speed up the discovery of subsumption relations in $\mathcal{R}'$ and hence build a non-redundant rewriting $\mathcal{R}''$ very efficiently. Following the ideas set in [41] we use the following three approaches:

1. Using information about redundant queries in $\mathcal{R}$ we try to identify queries that are also redundant in $\mathcal{R}'$.
2. Using information about non-redundancy of a query in $\mathcal{R}$ we try to decide the non-redundancy of certain queries in $\mathcal{R}'$.
3. Using information about non-subsumers in $\mathcal{R}$, we can try to decide if certain queries in $\mathcal{R}'$ are also non-subsumers.

The above three approaches can be used by algorithm removeRedundant as follows: given a set $\mathcal{R}'_{nr} \subseteq \mathcal{R}'$ of non-redundant queries and a set $\mathcal{R}'_{ns} \subseteq \mathcal{R}'$ of non-subsumer queries, removeRedundant$(\mathcal{R}')$ can check only if a query in $\mathcal{R}' \setminus \mathcal{R}'_{nr}$ is subsumed by some query in $\mathcal{R}' \setminus \mathcal{R}'_{ns}$ and dispense with other checks. As shown in [41] these techniques can significantly improve the efficiency of removeRedundant.

[1] We do not comment here on other techniques which are based on the structure and design of the database schema [35, 36]. These are independent of the rewriting algorithm and can be used here as well.

Next we provide properties that can be used to identify redundant, non-redundant, and non-subsumer queries during the execution of the main loop of Algorithms 2 and 3.

Proposition 18. *Let $\mathcal{O}$ be an ontology, $\mathcal{Q}$ be a CQ, let $\mathcal{R} = \mathcal{R}_D \uplus \mathcal{R}_Q$ be a rewriting of $\mathcal{Q}$ w.r.t. $\mathcal{O}$, and let $\mathcal{R}'$ be the output of Algorithm 2 when run for $\mathcal{R}$ and some arbitrary tuple of variables $\vec{y}$ s.t. $\vec{y} \subseteq \mathsf{var}(\mathcal{Q})$. Let $\mathcal{Q}_1$ be a CQ in $\mathcal{R}$. Then the following hold:*

1. *If there exists $\mathcal{Q}_2 \in \mathcal{R}$ that subsumes $\mathcal{Q}_1$ and if both extensions $\mathcal{Q}'_1$ and $\mathcal{Q}'_2$ of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ by $\vec{y}$, respectively, are safe, then $\mathcal{Q}'_1$ is redundant in $\mathcal{R}'$.*
2. *If $\mathcal{Q}_1$ is non-redundant in $\mathcal{R}$, then the safe extension of $\mathcal{Q}_1$ is also non-redundant in $\mathcal{R}'$.*
3. *If $\mathcal{Q}_1$ is a non-subsumer in $\mathcal{R}$, then the safe extension of $\mathcal{Q}_1$ is also non-subsumer in $\mathcal{R}'$.*

The proof is quite straightforward.

Next we deal with body atom removal and Algorithm 3.

Proposition 19. *Let $\mathcal{Q}$ be a CQ, let $\mathcal{O}$ be an ontology, let $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$ be the output of a rewriting algorithm when applied to $\mathcal{Q}$ and $\mathcal{O}$, and let α be an atom of $\mathcal{Q}$. Let also $\langle \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$ be the output of Algorithm 3 when applied to $\mathcal{O}, \mathcal{R}, \rightsquigarrow_{\mathcal{R}}$, and α, let $\mathcal{Q}_i \in \mathcal{R}$ be some CQ, and let $\tau = \{x \mapsto y \mid x \in \mathsf{var}(\mathcal{Q}), y \in \mathsf{var}(\mathcal{Q}_i)$ and $x \rightsquigarrow_{\mathcal{R}} y\}$, and assume that Algorithm 3 never executes line 15.*

If $\mathcal{Q}_i$ is non-redundant in $\mathcal{R}$ and for each $\mathcal{Q}_j \in \mathcal{R}$ the mapping τ_i defined as $\{x \mapsto y \mid x \in \mathsf{var}(\mathcal{Q}), y \in \mathsf{var}(\mathcal{Q}_j)$ and $x \rightsquigarrow_{\mathcal{R}} y\}$ is one-to-one, then the following properties hold:

(N1) *If $\alpha\tau \in \mathsf{bd}(\mathcal{Q}_i)$, then $\mathcal{Q}_i$ is non-redundant in $\mathcal{R}'$*

(N2) *$\bigwedge \mathsf{bd}(\mathcal{Q}_i) \setminus \{\alpha\tau\} \rightarrow \mathsf{hd}(\mathcal{Q}_i)$ is non-redundant in $\mathcal{R}'$*

If $\mathcal{Q}_i$ is non-subsumer in $\mathcal{R}$, then the following holds:

(S) *$\mathcal{Q}_i$ is also non-subsumer in $\mathcal{R}'$.*

Finally, if there exists $\mathcal{Q}_j \in \mathcal{R}$ that subsumes $\mathcal{Q}_i$, then the following properties hold, where $\tau' = \{x \mapsto y \mid x \in \mathsf{var}(\mathcal{Q}), y \in \mathsf{var}(\mathcal{Q}_j)$ and $x \rightsquigarrow_{\mathcal{R}} y\}$:

(R1) *If $\mathcal{Q}_j \in \mathcal{R}'$ or $\bigwedge \mathsf{bd}(\mathcal{Q}_j) \setminus \{\alpha\tau'\} \rightarrow \mathsf{hd}(\mathcal{Q}) \in \mathcal{R}'$, then $\mathcal{Q}_i$ is also redundant in $\mathcal{R}'$,*

(R2) *If $\mathcal{Q}_j \in \mathcal{R}'$ (resp. $\bigwedge \mathsf{bd}(\mathcal{Q}_j) \setminus \{\alpha\tau'\} \rightarrow \mathsf{hd}(\mathcal{Q}) \in \mathcal{R}'$) and for all $\alpha' \in \mathsf{bd}(\mathcal{Q}_j)$ (resp. $\alpha' \in \mathsf{bd}(\mathcal{Q}_j) \setminus \{\alpha\tau'\}$) atom α' has a different predicate name than α then $\bigwedge \mathsf{bd}(\mathcal{Q}_i) \setminus \{\alpha\tau\} \rightarrow \mathsf{hd}(\mathcal{Q})$ is redundant.*

Proof. First, we consider Properties **(N1)** and **(N2)**.

(N1): Let $\mathcal{Q}_j$ be an arbitrary query in $\mathcal{R}$ different from $\mathcal{Q}_i$. Upon termination, either $\mathcal{Q}_j$ or $\bigwedge \mathsf{bd}(\mathcal{Q}_j) \setminus \{\alpha\tau_j\} \rightarrow \mathsf{hd}(\mathcal{Q})$, for τ_j a substitution that maps the variables of $\mathcal{Q}$ to those of $\mathcal{Q}_j$, is added to $\mathcal{R}'$ by Algorithm 3. Since $\mathcal{Q}_i$ is non-redundant in $\mathcal{R}$, $\mathcal{Q}_j$ cannot subsume $\mathcal{Q}_i$. Hence, we need to show that $\bigwedge \mathsf{bd}(\mathcal{Q}_j) \setminus \{\alpha\tau_j\} \rightarrow \mathsf{hd}(\mathcal{Q})$ also does not subsume $\mathcal{Q}_i$.

Since $\mathcal{Q}_i$ is non-redundant this means that for every $\mathcal{Q}_j$ in $\mathcal{R}$ and every θ there exists $At \in \mathcal{Q}_j$ s.t. $At\theta \notin \mathcal{Q}_i$. Let θ' be such that $\tau = \tau_j\theta'$. This is possible since τ_j is one-to-one; hence, there exists an inverse τ_j^- and then we can set $\theta' = \tau_j^-\tau$. By condition of non-redundancy of $\mathcal{Q}_i$ we also have that for this θ' there exists $At \in \mathcal{Q}_j$ s.t. $At\theta' \notin \mathcal{Q}_i$. Consider now this specific At. It suffices to show that $At \in \mathcal{Q}_j \setminus \alpha\tau_j$. Assume to the contrary that $At \notin \mathcal{Q}_j \setminus \alpha\tau_j$. Since $At \in \mathcal{Q}_j$ this implies that $At = \alpha\tau_j$. By $At\theta' \notin \mathcal{Q}_i$ we also have $\alpha\tau_j\theta' \notin \mathcal{Q}_i$, however, by construction of θ' we have $\alpha\tau \notin \mathsf{bd}(\mathcal{Q}_i)$ which leads to a contradiction.

(N2): It follows from Property **N1** that since $\bigwedge \mathsf{bd}(\mathcal{Q}_j) \setminus \{\alpha\tau_j\} \rightarrow \mathsf{hd}(\mathcal{Q})$ does not subsume $\mathcal{Q}_i$ then clearly it also does not subsume $\bigwedge \mathsf{bd}(\mathcal{Q}_i) \setminus \{\alpha\tau\} \rightarrow \mathsf{hd}(\mathcal{Q})$.

Now we consider Property **(S)**. Since $\mathcal{Q}_i$ does not subsume any CQ in $\mathcal{R}$, we have $\mathcal{Q}_i\theta \not\subseteq \mathcal{Q}_j$ for all $\mathcal{Q}_j \in \mathcal{R}$ different from $\mathcal{Q}_i$ and for all substitutions θ. Hence, for each θ there exists at least one atom At in the body of $\mathcal{Q}_i$ such that $At \in \mathcal{Q}_i\theta$ and $At \notin \mathcal{Q}_j$. Upon termination, Algorithm 3 can add to $\mathcal{R}'$ either the CQ $\mathcal{Q}_j$ or the CQ $\bigwedge \mathsf{bd}(\mathcal{Q}_j) \setminus \{\alpha\tau'\}$, for a substitution τ' that maps the variables of $\mathcal{Q}$ to the variables of $\mathcal{Q}_j$. It is obvious that in both cases for any θ there would stil exist an atom $At \in \mathcal{Q}_i\theta$ such that $At \notin \mathcal{Q}_j$.

Finally, we consider Properties **(R1)** and **(R2)**. First, since $\mathcal{Q}_i$ is subsumed by $\mathcal{Q}_j$ there exists θ such that for each $At \in$

$\mathcal{Q}_j\theta$ we have $At \in \mathcal{Q}_i$. Now, Property **(R1)** is straightforward since for the same θ that we have that for each $At \in \mathcal{Q}_j\theta$ (or $At \in (\mathcal{Q}_j \setminus \alpha\tau')\theta$) we still have $At \in \mathcal{Q}_i$.

Finally, for Property **(R2)**, assume that $\mathcal{Q}_j \in \mathcal{R}'$; the case with $\bigwedge \mathsf{bd}(\mathcal{Q}_j) \setminus \{\alpha\tau'\} \in \mathcal{R}'$ is similar. By assumption we have that for each $At \in \mathcal{Q}_j\theta$ also $At \in \mathcal{Q}_i$. Consider an arbitrary atom At. We can only have $At \notin \mathsf{bd}(\mathcal{Q}_i) \setminus \{\alpha\tau\}$ if $At = \alpha\tau$, i.e., the atom that is removed is some atom of $\mathcal{Q}_j\theta$. However, by assumption At has a different predicate than α, hence we trivialy have that $At \neq \alpha\tau$ for any τ. □

7. Evaluation

We have implemented Algorithms 1–3 in prototype systems. In order to implement the function rew that is used, we have done one implementation using Rapid [10], a very fast rewriting system for DL-Lite ontologies, and one using Requiem, a rewriting system for the much more expressive language $\mathcal{ELHI}$ [31]. In the following we call the first implementation Ref_L and the second Ref_E.

We have compared Ref_L against Rapid and Ref_E against Requiem. For the first comparison we used the evaluation framework proposed in [31]. It consists of eight DL-Lite ontologies, namely V, P5, P5X, S, U, UX, A, and AX, and five test queries for each of them. For the second comparison we used the LUBM benchmark [17], a well-known benchmark that consists of one ontology (L) and 14 test queries, as well as three $\mathcal{ELHI}$ fragments extracted from the GALEN ontology (G), a well-known biomedical ontology with complex structure, called G_1, G_2, and G_3.[2] For GALEN we have manually created five test queries which are given in Table 1.

Since the problems we study involve the addition or removal of variables or removal of body atoms we have sometimes discarded those test queries that have only one distinguished variable; regarding our custom GALEN queries we have changed them accordingly to suit our purposes. Moreover, in case that a query has more than one variable (atom) or it can be extended in more than one way we have performed all possible refinements and hence the numbers we will present are averages over all runs. All experiments were conducted on a Mac Book Pro with a 2.66GHz processor and 4GB of RAM with a time-out of 600 seconds.

Table 1: Queries for G_1, G_2, and G_3

$Q(x) \leftarrow \mathsf{FunctionalAttribute}(x, y)$
$Q(x) \leftarrow \mathsf{isCountConcentrationOf}(x, y) \wedge \mathsf{Cell}(y) \wedge$
$\quad \mathsf{isMixedThroughout}(y, z) \wedge \mathsf{Tissue}(z)$
$Q(x) \leftarrow \mathsf{hasFeature}(x, y) \wedge \mathsf{ProcessFeature}(y)$
$Q(x) \leftarrow \mathsf{Displacement}(x) \wedge \mathsf{isOutcomeOf}(x, y) \wedge$
$\quad \mathsf{hasDuration}(y, z) \wedge \mathsf{ErythrocyteSedimentation}(y) \wedge$
$\quad \mathsf{Duration}(z) \wedge \mathsf{hasQuantity}(z, k) \wedge \mathsf{OneHour}(k)$
$Q(x) \leftarrow \mathsf{Behaviour}(x) \wedge \mathsf{hasGoal}(x, y)$

(a) G_1

$Q(x) \leftarrow \mathsf{BodyPart}(x), \mathsf{hasIntrinsicAbnormalityStatus}(x, y)$
$Q(x) \leftarrow \mathsf{Artery}(x), \mathsf{hasFeature}(x, y)$
$Q(x) \leftarrow \mathsf{GenericBodyStructure}(x), \mathsf{FeatureStateAttribute}(x, y)$
$Q(x) \leftarrow \mathsf{leftRightPaired}(x), \mathsf{hasIntrinsicAbnormalityStatus}(x, y)$
$Q(x) \leftarrow \mathsf{SolidBodxyStructure}(x), \mathsf{hasState}(x, y)$

(b) G_2

$Q(x) \leftarrow \mathsf{MicroOrganism}(x) \wedge \mathsf{playsPhysiologicalRole}(x, y)$
$Q(x) \leftarrow \mathsf{MicroscopicStructure}(x) \wedge \mathsf{actsOn}(x, y)$
$Q(x) \leftarrow \mathsf{Process}(x) \wedge \mathsf{hasGoal}(x, y)$
$Q(x) \leftarrow \mathsf{Feature}(x) \wedge \mathsf{hasFeature}(x, y)$
$Q(x) \leftarrow \mathsf{Process}(x) \wedge \mathsf{hasOutcome}(x, y)$

(c) G_3

All tables we present next use the following notation. Regarding our prototypes, with $\sharp\mathcal{R}$ we denote the size of the rewriting that is the input to our algorithms, with t_{alg} we denote the computation time that either of the Algorithms 1–3 needs (without the final redundancy elimination step). Additionally, for the tables reported for Algorithms 1 and 3, with t_{rew} we denote the time consumed by the sub-routine rew. Regarding the systems Rapid and Requiem, with t_{R} we denote the computation time again without redundancy elimination. For all systems, t_{sub} denotes the time for subsumption deletion. After this

[2] The fragments are available at `http://image.ece.ntua.gr/~gstoil/CQ_Refinement`

Table 2: Evaluation of Algorithm 1 using DL-Lite ontologies

$\mathcal{O}$	Q	Ref_L, Algorithm 1				Rapid	
		$\sharp\mathcal{R}$	t_{rew}	t_{alg}	t_{sub}	t_{R}	t_{sub}
V	2	10	5,0	7,0	2,0	17,5	0,5
	3	72	0,0	2,0	8,0	14,5	0,0
	4	185	0,0	4,5	27,5	25,5	0,5
S	2	7	1,5	2,0	0,5	19,0	1,0
	3	8	13,8	14,3	0,8	6,3	0,2
	4	11	8,3	8,7	0,2	19,2	0,3
	5	14	16,7	17,0	0,9	39,9	1,1
U	2	3	0,5	0,5	0,0	3,0	0,0
	3	8	3,3	3,5	0,0	4,2	0,0
	4	8	11,0	11,0	0,0	12,5	0,0
UX	2	3	0,0	0,0	0,0	4,5	0,0
	3	16	0,0	0,2	0,3	10,5	0,0
	4	11	0,5	0,5	0,5	26,0	0,0

step all systems return rewritings of the same size, hence the numbers are not presented. All times are in milliseconds.

7.1. Answer Variable Removal

Table 2 presents the comparison between Ref_L and Rapid over the DL-Lite ontologies. As we can observe Ref_L is generally faster than Rapid (with the exception of ontology V query 4 where t_{sub} dominates the total time). Notable cases are query 5 over ontology S and queries 3 and 4 over UX. The reason for this is that Algorithm 1 iterates over a small rewriting applying the projection operator. Moreover, very little time is spent in executing function rew as a sub-routine which was called at most three times in ontology S and on average less than 1.5 times.

Table 3 presents the comparison between Ref_E and Requiem over the $\mathcal{ELHI}$ ontologies. For query 9 Requiem could not compute a rewriting within the time-out of 600 seconds and thus we could also not run Ref_E (no input rewriting available). Again we can observe that our system is in most (if not all cases) much faster than Requiem. Notable cases are queries 1, 2, 5 and 13 over the LUBM ontology for which it is almost 25 times faster, queries 1 and 3 over G_1, all queries over G_2, and queries 1, 3, and 5 over G_3. Like above, the reason for this is that our algorithm iterates over a small input rewriting and performs simple operations. Only exceptions are queries 4, 7,

Table 3: Evaluation of Algorithm 1 using $\mathcal{ELHI}$ ontologies

$\mathcal{O}$	Q	Ref_E, Algorithm 1				Requiem	
		$\sharp\mathcal{R}$	t_{rew}	t_{alg}	t_{sub}	t_{R}	t_{sub}
L	1	3	42,5	42,5	0,0	159,0	110,0
	2	10	33,0	33,7	2,0	88,2	3,5
	3	1	0,0	0,0	0,0	1,5	1,0
	4	348	38,9	39,8	1,7	27,6	29,8
	5	334	31,0	33,0	2,5	46,0	56,0
	7	334	110,5	110,5	4,3	15,5	10,3
	8	347	19,7	20,3	0,6	15,9	11,4
	9	-	-	-	-	-	-
	10	331	10,5	10,5	0,5	16,5	10,0
	11	3	0,0	0,0	0,0	0,0	0,0
	12	333	13,3	13,3	0,2	14,8	11,5
	13	330	0,5	0,5	0,0	27,5	20,5
G_1	1	10	101,0	101,5	4,5	658,0	100,0
	2	62	185,0	186,0	21,0	266,0	106,5
	3	124	180,0	181,5	18,0	361,0	134,5
	4	4	84,5	84,5	2,0	126,5	30,5
	5	12	76,0	76,0	1,5	121,5	0,5
G_2	1	373	1 591,0	1 591,5	1,0	4 038,5	68,0
	2	395	2 818,0	2 818,5	1,5	3 835,5	24,0
	3	484	1 574,0	1 575,5	5,0	3 850,0	28,0
	4	685	1 688,0	1 692,5	43,0	4 149,0	63,0
	5	443	1 632,5	1 634,0	1,5	3 889,5	16,0
G_3	1	11	72,5	72,5	1,0	334,5	24,0
	2	128	112,0	114,8	4,7	163,3	18,7
	3	63	20,0	20,5	1,5	101,5	3,5
	4	79	96,2	96,5	1,4	82,2	3,3
	5	165	19,0	19,0	0,5	47,5	5,0

and 8 over L and query 4 over G_3. With careful analysis we concluded that in these tests Ref_E executed the sub-routine rew quite a lot of times. For example, in queries 7 and 8 over L rew was called on average 4 times in each case, as opposed to an average of less than one in all other queries over L. Similarly in query 4 over G_3 rew was called twice with an average of 0.5 in all other cases. Note finally that t_{rew} usually dominates the total computation time and that in most pathological cases it already exceeds Requiem's time.

7.2. Variable Extensions

Table 4 presents the comparison between Ref_L and Rapid over the DL-Lite ontologies. We can observe that in the ma-

Table 4: Evaluation of Algorithm 2 using DL-Lite ontologies.

$\mathcal{O}$	Q	Ref$_L$, Algorithm 2			Rapid	
		$\sharp\mathcal{R}$	t_{alg}	t_{sub}	t_{R}	t_{sub}
V	2	10	1,0	0,0	56,0	1,0
	5	30	3,1	1,6	27,4	0,0
P5	1	6	0,0	0,0	0,0	0,0
	2	10	0,7	0,0	1,7	0,0
	3	13	0,3	0,1	3,3	0,1
	4	15	0,3	0,0	11,5	0,3
	5	16	0,1	0,1	22,0	0,6
P5X	1	14	0,0	0,0	1,0	0,0
	2	81	0,3	0,0	11,0	0,0
	3	413	1,6	1,0	15,9	2,3
	4	2 070	10,1	3,7	62,9	64,5
	5	10 352	70,0	19,0	126,1	94,9
U	1	2	0,0	0,0	0,0	0,0
	5	3 375	22,0	10,0	2,0	0,0
UX	1	5	0,0	0,0	0,3	0,0
	5	8 955	27,0	26,0	2,0	0,0
A	1	378	1,0	0,0	2,0	0,0
	2	103	1,0	0,0	1,0	0,0
	3	104	1,0	1,0	3,0	0,3
	4	471	3,0	2,0	16,0	1,0
	5	624	5,7	2,3	20,0	1,7
AX	1	794	0,0	0,0	5,0	1,0
	2	1 812	3,0	8,0	40,0	3,0
	3	4 763	19,0	44,0	122,0	2,3
	4	7 229	19,0	26,0	77,0	2,0
	5	78 885	661,3	777,3	742,7	31,7

jority of cases our algorithm is much faster, something that can again be justified by the simple structure of Algorithm 2 and the fact that it does not require to call a rewriting algorithm. Moreover, we can observe that our techniques for optimizing the redundancy elimination step presented in Section 6 improve the performance of the redundancy elimination significantly. This is mostly evident in queries 4 and 5 over ontology P5X, where the subsumption deletion step takes a few milliseconds. Recall that this algorithm requires the input rewriting to be inference-closed. However, this is only reflected in the performance of query 5 over ontologies U and UX and query 5 over AX, where due to the size of the input rewriting our algorithm needs a significant amount of time.

Table 5: Evaluation of Algorithm 2 using $\mathcal{ELHI}$ ontologies

$\mathcal{O}$	Q	Ref$_E$, Algorithm 2			Requiem	
		$\sharp\mathcal{R}$	t_{alg}	t_{sub}	t_{R}	t_{sub}
L	1	5	1,0	0,0	192,0	1,0
	2	32	0,7	1,3	105,0	20,0
	3	17	0,0	0,0	1,0	2,0
	4	155	0,0	1,0	33,0	196,0
	5	158	1,0	0,0	24,0	17,0
	7	27	0,0	0,0	14,0	11,0
	8	23	0,0	0,0	18,0	12,0
	9	-	-	-	-	-
	10	11	0,0	0,0	15,0	13,0
	11	3	0,0	0,0	0,0	0,0
	12	8	0,0	0,0	24,0	11,0
	13	191	1,0	1,0	27,0	20,0
G_1	1	22	2,0	0,0	965,0	2,0
	2	439	2,0	0,0	360,0	10,0
	3	783	3,0	1,0	221,0	15,0
	4	7	0,0	1,0	144,0	1,0
	5	21	1,0	0,0	134,0	1,0
G_2	1	98	1,0	0,0	3 488,0	78,0
	2	60	0,0	0,0	2 802,0	51,0
	3	461	2,0	1,0	2 906,0	33,0
	4	622	3,0	3,0	3 151,0	64,0
	5	210	0,0	1,0	2 962,0	22,0
G_3	1	14	1,0	0,0	389,0	2,0
	2	323	3,0	2,0	279,5	67,0
	3	91	1,0	1,0	130,0	3,0
	4	139	1,0	0,2	105,3	4,3
	5	60	0,0	1,0	57,0	8,0

Table 5 presents the comparison between Ref$_E$ and Requiem over the $\mathcal{ELHI}$ ontologies. Since all variables in all LUBM queries are distinguished, we first removed an arbitrary variable, then computed a rewriting, and finally, run our algorithm adding the variable that was previously removed. As we can see our algorithm is in all queries much faster than Requiem with notable cases all queries over GALEN where it is always about three (in G_2 even four) orders of a magnitude faster. Note that, the better behaviour compared to DL-Lite can be justified by the smaller computed rewritings.

Table 6: Evaluation of Algorithm 3 using DL-Lite ontologies

$\mathcal{O}$	Q	Ref_L, Algorithm 3				Rapid	
		$\sharp\mathcal{R}$	t_{rew}	t_{alg}	t_{sub}	t_{R}	t_{sub}
V	2	10	0,0	2,0	1,0	20,0	1,0
	3	72	0,0	3,5	4,5	7,0	0,5
	4	185	0,0	2,0	0,5	7,0	0,0
	5	30	0,0	2,0	1,0	11,0	0,0
P5	2	10	1,0	1,0	0,0	1,0	0,0
	3	13	4,0	4,0	1,0	3,0	0,0
	4	15	6,0	6,0	1,0	5,0	1,0
	5	16	10,0	10,0	1,0	15,0	1,0
P5X	2	81	2,0	3,0	0,0	3,0	0,0
	3	413	8,0	8,0	1,0	16,0	1,0
	4	2 070	21,0	21,0	5,0	28,0	11,0
	5	10 352	211,0	211,0	81,0	159,0	120,0
S	2	204	0,0	2,5	1,0	2,5	0,0
	3	864	0,0	40,0	7,3	2,3	0,3
	4	1 428	0,0	15,0	5,7	2,0	0,0
	5	6 048	0,0	96,8	149,4	6,0	0,0
U	1	2	2,0	2,0	1,0	4,0	0,0
	2	190	0,0	1,0	0,5	1,5	0,0
	3	300	0,0	7,3	2,5	2,3	0,3
	4	1 688	0,0	16,5	0,0	1,5	0,5
	5	3 375	0,0	51,3	38,5	6,0	0,0
UX	1	5	1,0	1,0	0,0	11,0	0,0
	2	287	0,0	3,0	0,0	2,0	0,0
	3	1 260	0,0	22,5	17,8	4,8	0,0
	4	5 137	0,0	120,0	2,0	4,5	0,0
	5	8 955	0,0	152,0	88,8	14,8	0,0
A	1	378	0,0	0,0	0,0	2,5	0,5
	2	103	1,5	1,5	0,0	1,5	0,5
	3	104	2,0	3,7	4,3	10,7	0,0
	4	471	1,5	2,0	0,0	4,0	0,5
	5	624	8,7	11,7	11,7	25,3	7,0
AX	1	794	0,0	1,0	0,5	5,0	0,0
	2	1 812	1,5	2,0	1,0	6,5	0,5
	3	4 763	4,7	88,0	3 228,3	296,3	8,7
	4	7 229	2,0	5,0	9,5	27,0	5,5
	5	78 885	50,3	59,5	356,0	349,7	24,3

Table 7: Evaluation of Algorithm 3 using $\mathcal{ELHI}$ ontologies

$\mathcal{O}$	Q	Ref_E, Algorithm 3				Requiem	
		$\sharp\mathcal{R}$	t_{rew}	t_{alg}	t_{sub}	t_{R}	t_{sub}
L	1	3	4,5	5,0	0,0	87,0	0,5
	2	30	0,0	3,5	1,8	92,2	2,8
	3	17	0,0	1,0	0,0	0,0	0,0
	4	319	0,0	0,0	1,0	56,0	1,0
	5	154	0,0	1,0	0,0	40,0	0,0
	7	1 315	0,0	10,5	28,0	44,0	60,5
	8	1 055	0,0	39,0	41,0	166,5	176,5
	9	-	-	-	-	-	-
	10	264	0,0	0,0	0,0	17,0	0,0
	11	3	0,0	0,0	0,0	0,0	0,0
	12	156	0,0	4,0	1,0	22,5	6,0
	13	185	0,0	1,0	0,0	1,0	0,0
G_1	1	16	0,0	1,0	1,0	1 150,0	1,0
	2	4 117	0,0	114,0	68,0	673,0	3,0
	3	6 742	0,0	143,0	27,0	1 416,0	82,0
	4	21	0,0	1,0	0,0	376,0	0,0
	5	111	0,0	7,0	0,0	380,0	0,0
G_2	1	80	0,0	5,0	1,0	694,0	2,0
	2	54	0,0	1,0	0,0	245,0	0,0
	3	26 859	0,0	120,0	7,0	669,0	67,0
	4	380	0,0	2,0	1,0	251,0	2,0
	5	152	0,0	0,0	0,0	169,0	1,0
G_3	2	784	62,0	67,5	25,0	1 313,0	17,5
	3	91	47,0	47,0	21,0	1 231,0	2,0
	4	2 439	23,3	67,0	8,7	1 259,3	21,3

7.3. Body Atom Removal

Table 6 presents the comparison between Ref_L and Rapid over the DL-Lite ontologies. We can observe that in most cases our algorithm behaves as good as Rapid (recall that Rapid is a highly optimised DL-Lite tailored system). However there are some cases where Rapid is much faster, like in query 5 in ontologies S and U, and queries 4 and 5 in UX. This is because, as noted, the input inference-closed rewriting is quite large. Another performance bottleneck is when the output of Ref_L is large, and hence the input to the subsumption deletion algorithm is also large. This is for example the case in query 3 ontology AX, where the subsumption algorithm requires more than 3 seconds, while Rapid only needs 8,7 milliseconds. Apparently our optimisations for subsumption checking did not

have any effects on this case.

Finally, Table 7 presents the comparison between Ref_E and Requiem over the $\mathcal{ELHI}$ ontologies. We can observe that in all cases our implementation outperforms Requiem with most notable cases queries 1, 2, and 8 over the LUBM ontology and again all queries over GALEN's fragments where it can by several orders of a magnitude faster. Note that, despite some large input rewritings (queries 2 and 3 over G_1 and query 3 over G_2), the performance was not affected much in comparison to Requiem since the latter already requires several seconds due to the more complex structure of GALEN. Finally, rew was called at-most once, hence there was no overhead from repeated calls to this sub-routine.

8. Conclusions

In the current paper we have studied the problem of computing a rewriting for a refined query—that is, a query some of the answer variables of which have been removed (added) or some body atoms removed, by exploiting as much as possible a previously computed rewriting.

First, we studied the problem theoretically and have shown that in all cases it is possible to exploit pre-computed information giving detailed algorithms. However, in variable and atom removal this is generally not possible without (at least to some extent) making use of a standard rewriting algorithm rew. We implemented all proposed algorithms in two prototype systems, one that supports DL-Lite and one that supports $\mathcal{ELHI}$ and have conducted an extensive experimental evaluation. Our results reflect the structure and properties of the proposed algorithms. That is, variable additions that do not require the use of a rewriting algorithm are in the vast majority of cases much faster than existing systems. Variable removal is again generally faster than the compared system and atom removal was very competitive over the DL-Lite and much faster over the $\mathcal{ELHI}$ ontologies. We have found that the performance of the algorithms using rew can be influenced by the number of times this sub-routine is executed, but with some simple optimisations this number can be reduced significantly and adverse effects were observed only a few times in our experiments. Moreover, we have found that in some cases the requirement of inference-closed rewritings can also influence the overall performance, however, this was not an issue over the more complex GALEN fragments where Requiem already requires quite some effort.

As part of future work we will investigate and devise further optimisations for subsumption deletion, for reducing the number of times the sub-routine rew is called, as well as for the core of our algorithms. Moreover, the extension to more expressive ontology languages for which rewriting algorithms exist is also an open problem.

Acknowledgements

Giorgos Stoilos was funded by a Marie Curie Career Reintegration Grant within European Union's 7th Framework Programme (FP7/2007-2013) under REA grant agreement 303914.

References

[1] Andrea Acciarri, Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati. Quonto: Querying ontologies. In *Proceedings of the 20th International Conference on Artificial Intelligence (AAAI-05)*., 2005.

[2] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-Lite family and relations. *Journal of Artificial Intelligence Research*, 36:1–69, 2009.

[3] F. Baader, D. McGuinness, D. Nardi, and P.F. Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2002.

[4] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the 19th International Joint Conference on AI (IJCAI-05)*, pages 364–369. Morgan-Kaufmann Publishers, 2005.

[5] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[6] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS*, pages 228–242, 2010.

[7] Diego Calvanese and Giuseppe De Giacomo. Data integration: a logic-based perspective. *AI Magazine*, 26(1):59–70, 2005.

[8] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

[9] Chin-Liang Chang and Richard C. T. Lee. *Symbolic logic and mechanical theorem proving*. Computer science classics. Academic Press, 1973.

[10] A. Chortaras, D. Trivela, and G. Stamou. Optimized query rewriting in OWL 2 QL. In *Proceedings of the 23rd International Conference on Automated Deduction (CADE 23), Polland*, pages 192–206, 2011.

[11] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Journal of Web Semantics (JWS)*, 6(4):309–322, 2008.

[12] S. Derriere, A. Richard, and A. Preite-Martinez. An Ontology of Astronomical Object Types for the Virtual Observatory. In *Proceedings of the 26th meeting of the IAU: Virtual Observatory in Action: New Science, New Technology, and Next Generation Facilities*, pages 17–18, 2006.

[13] Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for horn-shiq plus rules. In *AAAI*, 2012.

[14] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. *Journal of Artificial Intelligence Research (JAIR)*, 31:157–204, 2008.

[15] C. Golbreich, S. Zhang, and O. Bodenreider. The Foundational Model of Anatomy in OWL: Experience and Perspectives. *Journal of Web Semantics*, 4(3):181–195, 2006.

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

[17] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semant.*, 3(2-3):158–182, October 2005.

[18] Ashish Gupta, Inderpal S. Mumick, and Kenneth A. Rossl. Adapting materialized views after redefinitions. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 211–222, 1995.

[19] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a web ontology language. *Journal Web Semantics*, 1(1):7–26, 2003.

[20] Bernard J. Jansen, Amanda Spink, Chris Blakely, and Sherry Koshman. Defining a session on web search engines: Research articles. *Journal of the American Society for Information Science and Technology*, 58:862–871, 2007.

[21] Atanas Kiryakov, Barry Bishoa, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. The Features of BigOWLIM that Enabled the BBCs World Cup Website. In *Workshop on Semantic Data Management (SemData)*, 2010.

[22] Christoph Koch. Query rewriting with symmetric constraints. In *Proceedings of the Second International Symposium on Foundations of Information and Knowledge Systems*, pages 130–147, 2002.

[23] Carsten Lutz. The complexity of conjunctive query answering in expressive description logics. In *Proceedings of the 4th International Joint Conference on Automated Reasoning, IJCAR 2008*, pages 179–193, 2008.

[24] Mukesh Mohania. Avoiding re-computation: View adaptation in data warehouses. In *In Proceedings of 8 th International Database Workshop*, pages 151–165, 1997.

[25] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles, 2009.

[26] Boris Motik, Ian Horrocks, and Su Myeon Kim. Delta-Reasoner: A Semantic Web Reasoner for an Intelligent Mobile Platform. In *Proceedings of the 21st International World Wide Web Conference (WWW 2012)*, pages 63–72, 2012.

[27] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *J. Web Sem.*, 3(1):41–60, 2005.

[28] Giorgio Orsi and Andreas Pieris. Optimizing query answering under ontological constraints. *PVLDB*, 4(11):1004–1015, 2011.

[29] Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Characterizing data complexity for conjunctive query answering in expressive description logics. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-06)*, 2006.

[30] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *Proceedings of the 1st international conference on Scalable information systems (InfoScale 06)*. ACM, 2006.

[31] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Efficient Query Answering for OWL 2. In *Proceedings of the International Semantic Web Conference (ISWC 09)*, pages 489–504, 2009.

[32] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, 8(2):186–209, 2010.

[33] Ian Horrocks Boris Motik und Laurent Pierre Pierre Chaussecourte, Birte Glimm. The energy management adviser at EDF. In *Proceedings of the 12th International Semantic Web Conference (ISWC 2013)*, 2013.

[34] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *Journal on Data Semantics*, X:133–173, 2008.

[35] Mariano Rodriguez-Muro and Diego Calvanese. High performance query answering over DL-Lite ontologies. In *Proceedings of the 13th International Conference Principles of Knowledge Representation and Reasoning, KR 2012,*, 2012.

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

[37] Riccardo Rosati and Alessandro Almatelli. Improving query answering over DL-Lite ontologies. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-10)*, 2010.

[38] Giorgos Stoilos, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks. Repairing ontologies for incomplete reasoners. In *Proceedings of the 10th International Semantic Web Conference (ISWC-11), Bonn, Germany*, pages 681–696, 2011.

[39] Eleni Tsalapati, Giorgos Stoilos, Giorgos Stamou, and George Koletsos. Query rewriting under ontology contraction. In *Proceedings of the 6th International Conference on Web Reasoning and Rule Systems (RR 2012),*

Vienna, Austria, pages 172–187, 2012.

[40] Tassos Venetis, Giorgos Stoilos, and Giorgos Stamou. Query rewriting under query extensions for OWL 2 QL ontologies. In *Proceedings of the 7th Workshop on Scalable Semantic Web Knowledge Base Systems*, 2011.

[41] Tassos Venetis, Giorgos Stoilos, and Giorgos Stamou. Query extensions and incremental query rewriting for OWL 2 QL ontologies. *Journal on Data Semantics, Accepted*, 2013.