Reformulating Ontological Queries Using Materialised Rewritings

Alexandros Chortaras
School of Electrical and Computer Engineering
National Technical University of Athens
achort@cs.ntua.gr

Bernardo Cuenca Grau
Department of Computer Science
University of Oxford
bernardo.cuenca.grau@cs.ox.ac.uk

Giorgos Stamou
School of Electrical and Computer Engineering
National Technical University of Athens
gstam@cs.ntua.gr

Giorgos Stoilos
School of Electrical and Computer Engineering
National Technical University of Athens
gstoil@image.ece.ntua.gr

ABSTRACT

Query rewriting is a prominent reasoning technique in ontology-based data access (OBDA) applications. Roughly speaking, a rewriting of a query $\mathcal{Q}$ w.r.t. an ontology is another query $\mathcal{Q}'$ that can be directly evaluated over the data without further reference to the input ontology. In this paper, we observe that many OBDA applications could significantly benefit from precomputing rewritings for certain queries. For example, in query optimisation, materialised rewritings of frequently asked queries can be used to speed up the query reformulation process. Moreover, in systems where users have different levels of access to information, materialised rewritings for the views assigned to each user can be exploited to obtain the set of answers to the input query derivable from the assigned views. Consequently, we investigate the problem of reformulating a query given a set of materialised rewritings and present a practical algorithm. Subsequently, we use our approach to design a fully fledged query rewriting algorithm which can exploit materialised rewritings to speed up the rewriting process. Our experimental results confirm the potential of our technique in practice.

CCS Concepts

•Computing methodologies → Knowledge representation and reasoning; Description logics;

Keywords

Ontologies; Semantic Web; OWL; Description Logics; Materialised Rewritings

1. INTRODUCTION

A recent application of ontologies that is continuously gaining momentum is *ontology-based data access* (OBDA) [22], where ontologies provide a formal semantically rich conceptualisation of the (possibly distributed) data sources, while answers to user queries reflect both the data in the sources as well as the knowledge in the ontology.

In this setting, a prominent technique for query answering is *query rewriting*. A rewriting for a given query $\mathcal{Q}$ and ontology $\mathcal{O}$ is another query $\mathcal{Q}'$ (typically a datalog query or a union of conjunctive queries) such that for any data set D the answers to $\mathcal{Q}$ with respect to D and $\mathcal{O}$ coincide with the answers to $\mathcal{Q}'$ with respect to D alone. Consequently, after computing a rewriting the problem of query answering can be delegated to efficient and scalable database and datalog evaluation systems. Ontology languages for which rewritings that are unions of conjunctive queries (called *UCQ rewritings*) always exist, are of particular interest since these can be translated into a union of SQL queries and evaluated over existing mature relational database systems. Numerous query rewriting systems have been recently developed in the literature. Prominent examples include Ontop [25], Requiem [20], Presto [26], Quest [24], Rapid [28], Nyaya [18], IQAROS [30], and Clipper [10].

An important limitation of state-of-the-art OBDA systems is that they cannot exploit rewritings that have been precomputed for certain queries; this implies, for example, that rewritings for each input query are computed from scratch. Unsurprisingly, many traditional data management applications rely on the availability of materialised information [14], which is relevant for a wide range of tasks such as query optimisation [11, 16], maintenance of physical data independence [29], data integration [9, 23], and data warehouse design [31]. In the context of OBDA, materialised rewritings of frequently asked queries could be used to speed up query answering; furthermore, OBDA systems in which users have different levels of access to information, materialised rewritings for the views assigned to each user can be exploited to obtain the answers to the input query derivable from the assigned views.

In this paper, we propose novel techniques for overcoming this limitation. In Section 3, we study the problem of reformulating a user query $\mathcal{U}$ when the only information available is a sound, but possibly incomplete, rewriting for a given set of queries. First, we study when an available rewriting is relevant for answering $\mathcal{U}$. Next, we propose an algorithm that rewrites $\mathcal{U}$ by relying only on the relevant rewritings. Our algorithm is guaranteed to be *sound*, in the sense that

WIMS '16, June 13-15, 2016, Nîmes, France

DOI: http://dx.doi.org/10.1145/2912845.2912867

when evaluating the computed rewriting for $\mathcal{U}$ over the system's data we obtain only correct answers. Our algorithm is thus applicable to many practical OBDA scenarios, such as query answering under restricted access to information.

In Section 4, we propose a sound and complete query rewriting algorithm that is able to exploit previously materialised rewritings, in the expectation that this additional information will help speed up the rewriting process. Our algorithm combines the reuse of information from materialised rewritings with reasoning steps, which are applied only whenever needed. In contrast to most optimisation techniques implemented in query rewriting systems, which are either system or language specific, our algorithm is *generic* in the sense that the required reasoning steps can be performed using any resolution-based rewriting calculus. In particular, our algorithm is compatible with rewriting calculi such as those underpinning Requiem, QuOnto, Nyaya, and Rapid, which can all be cast in the framework of resolution.

Finally, to test our claims in practice, we have optimised and implemented our rewriting algorithm using Requiem and conducted an experimental evaluation using both benchmark and realistic ontologies. Although the extent to which materialised rewritings help improving systems' performance depends on the overlap between the input query and the precomputed information, our results indicate that even a small set of precomputed rewritings for atomic queries can lead to a significant performance improvement.

2. PRELIMINARIES

We assume basic familiarity with first-order logic. Terms, atoms, literals, substitutions, formulas, sentences, clauses, satisfiability, and entailment are defined as usual. For a finite set of atoms $\{\alpha_1, \dots \alpha_n\}$, the expression $\bigwedge\{\alpha_1, \dots \alpha_n\}$ denotes the formula $\alpha_1 \wedge \ldots \wedge \alpha_n$.

A *fact* is a function-free ground atom and a *dataset* is a finite set of facts. A tuple of variables (resp. constants) is denoted as $\vec{x}$ (resp. as $\vec{a}$). For ϕ a formula, with $\phi(\vec{x})$ we stress that $\vec{x}$ are the free variables of ϕ. For σ a substitution, $\phi\sigma$ is the result of applying σ to ϕ.

Rules An *existential rule*, or simply *rule*, is a sentence of the following form [2, 4]:

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

where $\phi(\vec{x},\vec{z})$ and $\psi(\vec{x},\vec{y})$ are conjunctions of function-free atoms and $\vec{x},\vec{y}$ and $\vec{z}$ are pair-wise disjoint. Formula ϕ is called the *body* of r, formula ψ is called the *head*, and the universal quantifiers $\forall\vec{x}.\forall\vec{z}$ are typically omitted for brevity. If $\vec{y}$ is empty and the head consists of a single atom, then 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 variable that appears at least twice in the body of a rule and not in its head is called an *ej-variable*.

Many popular Horn ontology languages, such as DL-Lite$_R$ [5] and $\mathcal{ELHI}$ [20], as well as Datalog$^{\pm}$ [4] 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.

Queries A *datalog query* $\mathcal{Q}$ is a pair $\mathcal{Q} = \langle\mathcal{Q}_A, \mathcal{Q}_P\rangle$, where $\mathcal{Q}_A$ is the answer predicate and $\mathcal{Q}_P$ is a finite set of datalog rules such that the body of each rule in $\mathcal{Q}_P$ does not contain $\mathcal{Q}_A$. The set of *answer variables*—that is, variables occurring in an atom with predicate $\mathcal{Q}_A$—is denoted as $\mathsf{avar}(\mathcal{Q})$. Finally, by $\mathsf{var}(\mathcal{Q})$ we denote the set of all variables in $\mathcal{Q}_P$. A datalog query $\mathcal{Q} = \langle\mathcal{Q}_A, \mathcal{Q}_P\rangle$ is a *union of conjunctive queries* (UCQ) if $\mathcal{Q}_A$ is the only predicate occurring in the head of the rules in $\mathcal{Q}_P$; furthermore, $\mathcal{Q}$ is a *conjunctive query* (CQ) if it is a UCQ containing exactly one rule.

A tuple of constants $\vec{a}$ is a *certain answer* of a datalog query $\mathcal{Q} = \langle\mathcal{Q}_A, \mathcal{Q}_P\rangle$ over an ontology $\mathcal{O}$ and a dataset D if $\mathcal{O} \cup D \cup \mathcal{Q}_P \models \mathcal{Q}_A(\vec{a})$. We denote with $\mathsf{cert}(\mathcal{Q}, \mathcal{O} \cup D)$ the set of certain answers to $\mathcal{Q}$ with respect to (w.r.t.) $\mathcal{O} \cup D$.

To simplify the presentation, we often abuse notation and identify a CQ with the only rule it contains. Conversely, we often refer to a rule in a datalog query containing the answer predicate in the head as a CQ; e.g., the statement "for each CQ in $\mathcal{Q} = \langle\mathcal{Q}_A, \mathcal{Q}_P\rangle$" should be read as "for each rule $r \in \mathcal{Q}_P$ with $\mathcal{Q}_A$ occurring in the head". Finally, we often make no distinction between a datalog rule and its equivalent representation as a clause, e.g., the $A(x) \wedge B(x) \rightarrow C(x)$ corresponds to the clause $\neg A(x) \vee \neg B(x) \vee C(x)$. All notions introduced above, e.g., ej-variables, can also be defined for clauses in the obvious way.

Resolution Rules and Inferences We use standard notions of MGU, (hyper-)resolvent, and clause subsumption. We say that a clause C is redundant in a set of clauses N if it is either a tautology or is subsumed by a clause in N. For our purposes, a resolution inference rule is an $n+1$ relation on clauses annotated with a substitution. The elements of the relation and the substitution are written as follows:

$$\frac{C \quad C_1 \dots C_n}{C'}[\sigma]$$

Clause C is the main premise, and clauses $C_1, \dots, C_n$ are the side premises; we make the standard assumption that premises do not share variables. Clause C' is called the conclusion and the tuple $\langle C, C_1, \dots, C_n, C'\rangle$ is an inference. Given an inference $\langle C, C_1, \dots, C_n, C'\rangle$, we require C' to be a (hyper-)resolvent of C with $C_1, \dots, C_n$ 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. The annotation σ for an inference is defined as the subset of assignments in $\theta \circ \rho$ mapping variables to function-free terms. An inference system Γ is a collection of inference rules and for a set of clauses N, we denote with $\Gamma(N)$ the set of all inferences by Γ having all their premises in N. An inference system is called *converging* if all of its inference rules are such that the conclusion always has at-most the same number of ej-variables as the main premise.

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 to answering $\mathcal{Q}$ over $\mathcal{O}$ and an arbitrary dataset D. We make an explicit distinction between a rewriting and a *reformulation*, where the latter might only capture some (but not all) the information from $\mathcal{O}$ relevant for answering the query.

DEFINITION 1. *A* reformulation *of a CQ $\mathcal{Q} = \langle\mathcal{Q}_A, \mathcal{Q}_P\rangle$ with respect to an ontology $\mathcal{O}$ is a datalog query of the form $\mathcal{R} = \langle\mathcal{R}_A, \mathcal{R}_P\rangle$ such that $\mathcal{R}_A = \mathcal{Q}_A$ and $\mathcal{O} \cup \mathcal{Q}_P \models \mathcal{R}_P$.*

A reformulation $\mathcal{R}$ of $\mathcal{Q}$ with respect to $\mathcal{O}$ is a rewriting *if $\mathsf{cert}(\mathcal{Q}, \mathcal{O} \cup D) \subseteq \mathsf{cert}(\mathcal{R}, D)$ for each dataset D mentioning only predicates from $\mathcal{O}$.*

EXAMPLE 2. *Consider the ontology $\mathcal{O}$ and query $\mathcal{Q}$ given*

next:

$$\begin{aligned}\mathcal{O} &= \{\mathsf{Bolt}(x) \to \exists y.\mathsf{isPartOf}(x,y) \wedge \mathsf{Engine}(y),\\ &\quad\ \mathsf{Engine}(x) \to \exists y.\mathsf{hasPart}(x,y).\mathsf{Piston}(y),\\ &\quad\ \mathsf{isPartOf}(y,x) \to \mathsf{hasPart}(x,y)\}\\ \mathcal{Q} &= \mathsf{isPartOf}(x,y) \wedge \mathsf{hasPart}(y,z) \wedge \mathsf{Piston}(z) \to \mathcal{Q}_A(x)\end{aligned}$$

The pair $\langle Q, \{\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4\}\rangle$ *where* $\mathcal{Q}_1$–$\mathcal{Q}_4$ *are as defined below, is a rewriting* $\mathcal{Q}$ *with respect to* $\mathcal{O}$*:*

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

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

Many query rewriting algorithms for various ontology languages have been developed in recent years. Thus, to abstract from their specifics and be able to make general claims about them, we introduce an abstract notion that can capture many algorithms proposed in the literature.

Definition 3. *A* reformulation algorithm *for a class* $\mathcal{L}$ *of existential rules is an algorithm that for each* $\mathcal{L}$*-ontology* $\mathcal{O}$ *and CQ* $\mathcal{Q} = \langle \mathcal{Q}_A, \mathcal{Q}_P\rangle$ *proceeds in three phases:*

1. *It transforms* $\mathcal{O} \cup \mathcal{Q}_P$ *into a set of clauses* N_1*.*
2. *It relies on an inference system* Γ *to compute a sequence* $\langle N_1, \sigma_1\rangle \rhd \cdots \rhd \langle N_m, \sigma_m\rangle$ *of pairs of a set of clauses* N_i *and substitution* σ_i *such that* $\sigma_1 = \emptyset$ *and for each* $i \in \{1, \ldots, m-1\}$*,* N_{i+1} *extends* N_i *with a conclusion of an inference in* $\Gamma(N_i)$ *annotated with* σ_{i+1}*.*
3. *It returns a pair* $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}}\rangle$*, where* $\mathcal{R}_A = \mathcal{Q}_A$*,* $\mathcal{R}_P$ *is a function-free subset of* N_m*, and* $\rightsquigarrow_{\mathcal{R}}$ *is the relation defined as follows: for* $x \in \mathsf{var}(\mathcal{Q})$*,* $\mathcal{Q}' \in \mathcal{R}_P$*,* $y \in \mathsf{var}(\mathcal{Q}')$*, 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 reformulation algorithm is a rewriting algorithm *if for each input* $\mathcal{O}$ *and* $\mathcal{Q}$*, its output* $\mathcal{R}$ *is a rewriting of* $\mathcal{Q}$ *with respect to* $\mathcal{O}$*.*

Many state-of-the-art rewriting algorithms are either resolution based or can be cast in the framework of resolution, and hence they can be captured by our definition. More precisely, the inference rules in such algorithms can be formulated as variants of (hyper-)resolution and hence satisfy Properties 1 and 2 in Definition 3; furthermore, clauses with function symbols, which are used in some algorithms to derive necessary intermediate clauses, are never returned as part of the output rewriting and hence Property 3 also holds. The only non-standard component in Definition 3 is the use of annotations in inferences and the recording of the relation $\rightsquigarrow_{\mathcal{R}}$ between input and output variables. This information will be exploited in Section 3 to efficiently reuse precomputed reformulations. Most systems can be easily modified in order to track the substitutions and renamings being applied (and thus the annotations σ_i), and hence to return the relation $\rightsquigarrow_{\mathcal{R}}$.

3. EXPLOITING MATERIALISED REFORMULATIONS

In this section, we propose an algorithm that reformulates an input user query $\mathcal{U}$ by relying only on materialised reformulations for previous queries. The following example motivates the potential application of our algorithm to query answering in OBDA systems under restricted access to information.

Example 4. *Consider the following ontology* $\mathcal{O}$ *describing employees in a company:*

$$\begin{aligned}\mathcal{O} = \{&\mathsf{DeptMangr}(x) \to \mathsf{Mngr}(x), \mathsf{Mngr}(x) \to \mathsf{Empl}(x),\\ &\mathsf{OnPayroll}(x,y) \to \mathsf{Empl}(x)\}\end{aligned}$$

Assume also that users of an OBDA application based on $\mathcal{O}$ *have different levels of access to the system's information. For example, Level 1 users can access the list of all employees, except for department managers. To enforce this access policy, Level 1 users have been assigned a view defined by the following mapping from queries to a reformulation of them, where* $\mathcal{Q}_2 = \mathsf{Mngr}(x_2) \to \mathcal{Q}_A(x_2)$ *and* $\mathcal{Q}_3 = \mathsf{OnPayroll}(x_3, y_3) \to \mathcal{Q}_A(x_3)$*:*

$$\begin{aligned}\mathcal{Q}_1 = \mathsf{Empl}(x_1) \to \mathcal{Q}_A(x_1) \ &\mapsto\ \mathcal{R}_1 = \{\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3\}\\ \mathcal{Q}'_1 = \mathsf{DeptMngr}(x'_1) \to \mathcal{Q}_A(x'_1) \ &\mapsto\ \mathcal{R}'_1 = \emptyset\end{aligned}$$

Furthermore, assume also that the materialised reformulations have been computed by means of a reformulation algorithm and that the system also stores the output relation $\rightsquigarrow_{\mathcal{R}_1} = \{\langle x_1, x_2\rangle, \langle x_1, x_3\rangle\}$ *(see Definition 3). Assume now that a Level 1 user formulates the following query* $\mathcal{U}$*:*

$$\mathcal{U} = \mathsf{superviseBy}(x,y) \wedge \mathsf{Empl}(y) \to \mathcal{Q}_A(x) \qquad (2)$$

In the absence of access restrictions, a UCQ rewriting for $\mathcal{U}$ *would consist of* $\mathcal{U}$ *itself and the following CQs* (3)–(5)*:*

$$\mathsf{superviseBy}(u_1,v_1) \wedge \mathsf{DeptMngr}(v_1) \to \mathcal{Q}_A(u_1) \qquad (3)$$

$$\mathsf{superviseBy}(u_2,v_2) \wedge \mathsf{Mngr}(v_2) \to \mathcal{Q}_A(u_2) \qquad (4)$$

$$\mathsf{superviseBy}(u_3,v_3) \wedge \mathsf{OnPayroll}(v_3,w_3) \to \mathcal{Q}_A(u_3) \qquad (5)$$

Under the defined level of access, one would expect the OBDA system to compute a reformulation of $\mathcal{U}$ *consisting only of* (4) *and* (5)*. Such reformulation can be computed by means of simple matching, replacement, and renaming operations using* $\mathcal{R}_1$ *and without relying on reasoning. More precisely,* (5) *can be obtained as follows. First, we can identify the previous query* $\mathcal{Q}_1$ *as "relevant" to* $\mathcal{U}$*: atom* $\mathsf{Empl}(x_1)$ *in* $\mathcal{Q}_1$ *is isomorphic to* $\mathsf{Empl}(y)$ *in* $\mathcal{U}$ *and hence the reformulation materialised for* $\mathcal{Q}_1$ *can be used to also reformulate* $\mathcal{U}$*. Second, we can replace atom* $\mathsf{Empl}(y)$ *in* $\mathcal{U}$ *with* $\mathsf{OnPayroll}(x_3,y_3)$ *in* $\mathcal{Q}_3$*, where* $x_1 \rightsquigarrow_{\mathcal{R}_1} x_3$ *tells us that* x_3 *is the relevant variable; finally, we can apply the substitution* $\{y \mapsto x_3\}$ *and hence obtain the query* $\mathsf{superviseBy}(x,x_3) \wedge \mathsf{OnPayroll}(x_3,y_3) \to \mathcal{Q}_A(x)$*, which is equivalent to* (5) *up to renaming of variables.* ♢

Example 4 suggests that our algorithm needs to perform two key tasks when given an input query $\mathcal{U}$, namely to first identify which previous queries are "relevant" to $\mathcal{U}$ and to then reformulate $\mathcal{U}$ by reusing the materialised reformulations for the relevant queries.

Intuitively, a query $\mathcal{Q}$ is relevant to $\mathcal{U}$ if we can isomorphically map a subset of the body of $\mathcal{Q}$ to the body of $\mathcal{U}$.

For instance, $\mathcal{Q}_1$ in Example 4 is relevant to the input query $\mathcal{U}$.

DEFINITION 5. *Let $\mathcal{Q}$ and $\mathcal{U}$ be CQs and let σ be a one-to-one (partial) mapping from $\mathsf{var}(\mathcal{U})$ to $\mathsf{var}(\mathcal{Q})$. Query $\mathcal{Q}$ is* σ-relevant *to $\mathcal{U}$ if $\mathsf{bd}(\mathcal{Q}) \subseteq \mathsf{bd}(\mathcal{U}\sigma)$.*

Once a relevant query $\mathcal{Q}$ has been identified, our algorithm needs to reformulate the user query $\mathcal{U}$ by means of the reformulation $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$ available for $\mathcal{Q}$ in the OBDA system. As illustrated by Example 4, such reformulation of $\mathcal{U}$ is accomplished via an "unfolding" of the relevant body atoms in $\mathcal{U}$—that is, by replacing them with body atoms of CQs from $\mathcal{R}$. The unfolding process relies on the relation $\rightsquigarrow_{\mathcal{R}}$ associated to $\mathcal{R}$, which indicates how variables must be replaced. The following definition makes the notion of unfolding precise.

DEFINITION 6. *Let $\mathcal{O}$ be an ontology, let $\mathcal{Q}$ be a CQ, and let $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$ be the result of applying a reformulation algorithm to $\mathcal{Q}$ and $\mathcal{O}$. Furthermore, let $\mathcal{U}$ be a CQ, and let σ be a mapping such that $\mathcal{Q}$ is σ-relevant to $\mathcal{U}$. Finally, let $\mathcal{Q}' \in \mathcal{R}$ and let $\tau = \{x \mapsto y \mid x \in \mathsf{var}(\mathcal{Q}), y \in \mathsf{var}(\mathcal{Q}'), x \rightsquigarrow_{\mathcal{R}} y\}$. The* unfolding *of $\mathcal{U}$ w.r.t. $\mathcal{Q}'$ is the following CQ:*

$$\mathcal{U}' = [\textstyle\bigwedge(\mathsf{bd}(\mathcal{U}\sigma) \setminus \mathsf{bd}(\mathcal{Q})) \cup \mathsf{bd}(\mathcal{Q}') \rightarrow \mathsf{hd}(\mathcal{U}\sigma)]\tau \qquad (6)$$

The unfolding $\mathcal{U}'$ is sound *if $\mathcal{O} \cup \{\mathcal{U}\} \models \mathcal{U}'$.*

As illustrated by the following example, however, not all unfoldings of $\mathcal{U}$ are guaranteed to be sound. Clearly, unsound unfoldings cannot become a part of a reformulation of $\mathcal{U}$.

EXAMPLE 7. *Consider the following ontology and query:*

$$\begin{aligned} \mathcal{O} &= \{A(x) \rightarrow \exists y.S(x,y),\ S(x,y) \rightarrow R(x,y)\} \\ \mathcal{U} &= R(x,y) \wedge C(y) \rightarrow \mathcal{Q}_A(x) \end{aligned}$$

Assume a rewriting $\mathcal{R}_1$ for $\mathcal{Q}_1 = R(x_1,y_1) \rightarrow \mathcal{Q}_A(x_1)$ has been computed, which consists of $\mathcal{Q}_1$ and the following CQs:

$$\begin{aligned} \mathcal{Q}_2 = S(x_2,y_2) \rightarrow \mathcal{Q}_A(x_2); \quad x_1 \rightsquigarrow_{\mathcal{R}_1} x_2,\ y_1 \rightsquigarrow_{\mathcal{R}_1} y_2 \\ \mathcal{Q}_3 = A(x_3) \rightarrow \mathcal{Q}_A(x_3); \quad x_1 \rightsquigarrow_{\mathcal{R}_1} x_3 \end{aligned}$$

Query $\mathcal{Q}_1$ is relevant to $\mathcal{U}$ via $\sigma = \{x \mapsto x_1, y \mapsto y_1\}$. But then, if we unfold $\mathcal{U}$ w.r.t. $\mathcal{Q}_3 \in \mathcal{R}_1$ we get $\mathcal{U}' = A(x_3) \wedge C(y_1) \rightarrow Q_A(x_3)$, which is not entailed by $\mathcal{O} \cup \{\mathcal{U}\}$. ♢

Soundness of an unfolding can be checked by means of a reasoner that is sound and complete for deciding entailment of a fact by an ontology and an instance. For standard Horn DLs, example such reasoners include HermiT [17] and Pellet [27]. In practice, however, checking soundness of an unfolding using a fully-fledged DL reasoner can negatively impact the performance of reformulation. We thus define a sufficient condition that ensures soundness of an unfolding and which can be checked syntactically. We refer to the unfoldings satisfying this condition as *safe*.

DEFINITION 8. *Let $\mathcal{U}'$ be as given in Definition 6. The unfolding $\mathcal{U}'$ is* safe *if the following conditions hold:*

1. $\mathsf{var}((\mathsf{bd}(\mathcal{U}\sigma) \setminus \mathsf{bd}(\mathcal{Q})) \cap \mathsf{var}(\mathcal{Q}))\tau \subseteq \mathsf{var}(\mathcal{Q}')$, *and*
2. $(\mathsf{avar}(\mathcal{U}\sigma) \cap \mathsf{var}(\mathcal{Q}))\tau \subseteq \mathsf{var}(\mathcal{Q}')$.

Algorithm 1 UNFOLD$_{\mathcal{O}}(\mathcal{U}, \mathcal{Q}, \mathcal{R}, \rightsquigarrow_{\mathcal{R}})$

Input: $\mathcal{U}, \mathcal{Q}$: CQs; $\mathcal{R}$ and $\rightsquigarrow_{\mathcal{R}}$: output of a reformulation algorithm for $\mathcal{Q}$ and $\mathcal{O}$.

```
1: R_out := ∅
2: if Q not relevant to U then return R_out
3: σ := mapping such that Q is σ-relevant to U
4: for all CQs Q' ∈ R do
5:     U' := the unfolding of U w.r.t. Q' as in Definition 6
6:     if U' is safe then R_out := R ∪ {U'}
7: end for
8: R_out := R_out ∪ {r ∈ R | r is not a CQ}
9: return R_out
```

Intuitively, safety ensures that relevant information in $\mathcal{U}$ is not "lost" in the unfolding. Property 1 ensures that binary body atoms in $\mathcal{U}$ joining relevant variables are preserved in $\mathcal{U}'$, whereas Property 2 ensures preservation of answer variables. For instance, unfolding $\mathcal{U}'$ in Example 7 violates Property 1 and hence is not safe: $\mathsf{bd}(\mathcal{U}\sigma) \setminus \mathsf{bd}(\mathcal{Q}_1) = \{C(y_1)\}$, variable y_1 is mapped by substitution τ to itself, and thus the condition fails since $\mathsf{var}(\mathcal{Q}_3) = \{x_3\}$.

As shown in the following lemma, safety implies soundness because the inferences applied by a reformulation algorithm on $\mathcal{Q}$ to produce $\mathcal{Q}'$ can be "faithfully" applied over the subset of $\mathcal{U}$ that is isomorphic to $\mathcal{Q}$ in order to derive $\mathcal{U}'$.

LEMMA 9. *Every safe unfolding is sound.*

PROOF. Let $\mathcal{O}$, $\mathcal{Q}$, $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$, $\mathcal{U}$, σ, and τ be as in Definition 6, and let $\mathcal{U}'$ be an unfolding of $\mathcal{U}$ w.r.t. $\mathcal{Q}' \in \mathcal{R}$, as defined in (6). Furthermore, assume that $\mathcal{U}'$ is safe as specified in Definition 8. We show that $\mathcal{O} \cup \{\mathcal{U}\} \models \mathcal{U}'$. First, note that since σ is an one-to-one mapping (i.e., a variable renaming) it suffices to show that $\mathcal{O} \cup \{\mathcal{U}\sigma\} \models \mathcal{U}'$.

Since $\mathcal{Q}' \in \mathcal{R}$ and $\langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle$ is obtained by applying a reformulation algorithm to $\mathcal{Q}$ and $\mathcal{O}$, we have that $\mathcal{Q}' \in N_k$ for $\langle N_1, \sigma_1 \rangle \rhd \cdots \rhd \langle N_k, \sigma_k \rangle$ the sequence computed by the algorithm on input $\mathcal{Q}$ and $\mathcal{O}$. Clearly, all inferences that led to deriving $\mathcal{Q}'$ in N_k can be captured by a single inference $\Upsilon = \langle \mathcal{Q}, C_1, \ldots C_n, \mathcal{Q}' \rangle$ having $\mathcal{Q}$ as main premise and having clauses $C_1, \ldots, C_n$ derived from $\mathcal{O}$ as side premises; then, $\mathcal{Q}'$ is a (hyper-)resolvent of these premises via the composition of a MGU θ and a renaming ρ. As a result of Υ, a subset S of $\mathsf{bd}(\mathcal{Q})$ is replaced with a new set of atoms S'; so, if $\mathsf{bd}(\mathcal{Q}) = S_1 \cup S$, then, $\mathcal{Q}'$ is of the form $[\bigwedge(S_1 \cup S') \rightarrow \mathsf{hd}(\mathcal{Q})]\theta \circ \rho$. However, since $\mathcal{Q}'$ contains no functional terms all mappings from variables to function terms in θ can be discarded; hence, $\mathcal{Q}'$ is actually of the form $[\bigwedge(S_1 \cup S') \rightarrow \mathsf{hd}(\mathcal{Q})]\tau$ for τ as in Definition 6.

By Definition 5, $\mathsf{bd}(\mathcal{Q}) \subseteq \mathsf{bd}(\mathcal{U}\sigma)$; thus, $\mathcal{U}\sigma$ is of the form $S_2\sigma \cup S_1 \cup S$ with $S_2\sigma = \mathsf{bd}(\mathcal{U}\sigma) \setminus \mathsf{bd}(\mathcal{Q})$. Since $\mathcal{U}\sigma$ contains S, we can obtain $\mathcal{Q}'' = [\bigwedge(S_2\sigma \cup S_1 \cup S') \rightarrow \mathsf{hd}(\mathcal{U}\sigma)]\theta \circ \rho$ by applying the inference $\Upsilon' = \langle \mathcal{U}\sigma, C_1, \ldots C_n, \mathcal{Q}'' \rangle$ having the same side premises $C_1, \ldots C_n$ as Υ and annotation the function-free subset of $\theta \circ \rho$.

But then, Property 1 in Definition 8 can be used to show $(S_2\sigma)\theta \circ \rho = (S_1\sigma)\tau$ and Property 2 to show $\mathsf{hd}(\mathcal{U}\sigma)\theta \circ \rho = \mathsf{hd}(\mathcal{U}\sigma)\tau$, which together with the fact that $S_1\theta \circ \rho = S_1\tau$ and $S'\theta \circ \rho = S'\tau$ implies that $\mathcal{Q}'' = \mathcal{U}'$, as required. □

We are now ready to specify a simple reformulation algorithm UNFOLD$_{\mathcal{O}}$ (see Algorithm 1), which takes as input a

user query $\mathcal{U}$, a query $\mathcal{Q}$ and a reformulation for $\mathcal{Q}$ w.r.t. $\mathcal{O}$. The algorithm proceeds in the obvious way: it starts by checking relevance of $\mathcal{Q}$ to the user query and then proceeds by unfolding $\mathcal{U}$ w.r.t. each of the queries in the reformulation for $\mathcal{Q}$, while making sure that only safe unfoldings are included in the algorithm's output. Finally, the algorithm includes in the output unfolding all the datalog rules in the input reformulation that do not contain the answer predicate. Correctness is ensured by the following theorem, which follows directly from Lemma 9.

THEOREM 10. $\textsc{Unfold}_{\mathcal{O}}(\mathcal{U}, \mathcal{Q}, \langle \mathcal{R}, \rightsquigarrow_{\mathcal{R}} \rangle)$ *returns either the empty set, or a reformulation of* $\mathcal{U}$ *w.r.t.* $\mathcal{O}$.

4. QUERY REWRITING USING MATERIALISATIONS

In this section, we propose a sound and complete rewriting algorihtm that is well-suited for implementation and which is capable of exploiting previously materialised rewritings for optimisation purposes.

In order to ensure completeness, our algorithm combines the reuse of information from materialised rewritings with *reasoning steps*. As illustrated by the following example, reasoning is indeed required since a rewriting might not always be computable by relying only on unfolding operations.

EXAMPLE 11. *Consider the following ontology and user CQ:*

$$\begin{aligned} \mathcal{O} &= \{A(x) \rightarrow \exists y.R(x,y),\ R(z,y) \rightarrow C(y)\} \\ \mathcal{U} &= R(x,y) \wedge C(y) \rightarrow \mathcal{Q}_A(x) \end{aligned}$$

Assume that a rewriting $\mathcal{R}_1$ *for* $\mathcal{Q}_1 = C(x_1) \rightarrow \mathcal{Q}_A(x_1)$ *has been computed, which consists of* $\mathcal{Q}_1$ *itself and the additional CQ* $\mathcal{Q}_1' = R(z_1, y_1) \rightarrow \mathcal{Q}_A(y_1)$, *where* $x_1 \rightsquigarrow_{\mathcal{R}_1} y_1$. *Clearly,* $\mathcal{Q}_1$ *is relevant to* $\mathcal{U}$ *and the application of Algorithm 1 to* $\mathcal{U}$, $\mathcal{Q}_1$ *and* $\langle \mathcal{R}_1, \rightsquigarrow_{\mathcal{R}_1} \rangle$ *yields a reformulation* $\mathcal{R}_{\mathsf{out}} = \{\mathcal{Q}_1, \mathcal{Q}_2\}$ *where* $\mathcal{Q}_2 = R(x, y_1) \wedge R(z_1, y_1) \rightarrow \mathcal{Q}_A(x)$. *Such reformulation is, however, not a rewriting of* $\mathcal{U}$ *w.r.t.* $\mathcal{O}$: *given the dataset* $D = \{A(c)\}$, *we have that* $c \in \mathsf{cert}(\mathcal{U}, \mathcal{O} \cup D)$, *whereas* $c \notin \mathsf{cert}(\mathcal{R}_{\mathsf{out}}, D)$. *By exploting the inference rules in a rewriting algorithm, such as the one underpinning Requiem, we can derive the CQ* $\mathcal{Q}_3 = A(x_3) \rightarrow \mathcal{Q}_A(x_3)$ *and include it in* $\mathcal{R}_{\mathsf{out}}$ *to obtain a rewriting.* ♢

Algorithm 2 specifies a query rewriting procedure based on the intuitions outlined in Example 11. The algorithm accepts as input a user query $\mathcal{U}$, an ontology $\mathcal{O}$, and a set of triples $\langle \mathcal{Q}', \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$ which relate "previous" queries with a materialised rewriting. Algorithm 2 relies on the inference system Γ of some rewriting algorithm, the inference rules of which are used for performing the necessary reasoning steps; it is therefore implicitly assumed that the input ontology $\mathcal{O}$ belongs to a class of existential rules for which such rewriting algorithm is applicable.

The algorithm proceeds as follows. First, it initialises $\mathcal{R}_{\mathsf{out}}$ to the set of clauses obtained by applying to (the clausification of) $\mathcal{O}$ the inference rules in Γ exhaustively until no more inference rule is applicable; furthermore, it initialises $\mathcal{R}_{\mathsf{new}}$ to the input query $\mathcal{U}$. Then, it iterates through all triples in $\mathbf{Q}$ and unfolds the rewritings of queries that are relevant to $\mathcal{U}$ adding them first to $\mathcal{R}_{\mathsf{new}}$ and then to $\mathcal{R}_{\mathsf{out}}$. In this step, the algorithm tries to reuse the materialised information in as much as possible and in some cases encountered

Algorithm 2 $\textsc{UnfoldAndRewrite}(\mathcal{U}, \mathcal{O}, \mathbf{Q})$

Input: A CQ $\mathcal{U}$, an ontology $\mathcal{O}$ and a set $\mathbf{Q}$ of triples $\langle \mathcal{Q}', \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$ where $\mathcal{R}', \rightsquigarrow$ are outputs of a rewriting algorithm for $\mathcal{Q}'$ and $\mathcal{O}$.

1: $\mathcal{R}_{\mathsf{out}} :=$ saturation of $\mathcal{O}$ using the inference system Γ
2: $\mathcal{R}_{\mathsf{new}} := \{\mathcal{U}\}$
3: **for all** $\langle \mathcal{Q}', \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle \in \mathbf{Q}$ **do**
4: $\quad \mathcal{R}_{\mathsf{new}} := \mathcal{R}_{\mathsf{new}} \cup \textsc{Unfold}_{\mathcal{O}}(\mathcal{U}, \mathcal{Q}', \mathcal{R}', \rightsquigarrow_{\mathcal{R}'})$
5: **end for**
6: $\mathcal{R}_{\mathsf{out}} := \mathcal{R}_{\mathsf{out}} \cup \mathcal{R}_{\mathsf{new}}$
7: **while** $\mathcal{R}_{\mathsf{new}} \neq \emptyset$ **do**
8: $\quad$ Select and remove a CQ $\mathcal{Q}_h$ from $\mathcal{R}_{\mathsf{new}}$
9: $\quad \mathcal{R}_{\mathsf{out}} := \mathcal{R}_{\mathsf{out}} \cup \{\mathcal{Q}_h\}$
10: $\quad$ **for all** $\Upsilon \in \Gamma(\mathcal{R}_{\mathsf{out}})$ with main premise $\mathcal{Q}_h$ **do**
11: $\quad\quad$ **if** concl. C' nonredundant in $\mathcal{R}_{\mathsf{out}} \cup \mathcal{R}_{\mathsf{new}}$ **then**
12: $\quad\quad\quad \mathcal{R}_{\mathsf{new}} := \mathcal{R}_{\mathsf{new}} \cup \{C'\}$
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: **end while**
16: **return** the rules for all function-free clauses in $\mathcal{R}_{\mathsf{out}}$

in our evaluation succeeds in computing a complete rewriting. As illustrated in Example 11, however, in general we need to subsequently apply further inference steps to ensure completeness. Hence, the algorithm proceeds in checking all queries in $\mathcal{R}_{\mathsf{new}}$ to determine whether new queries can be obtained via inferences. If this is the case then these are added to $\mathcal{R}_{\mathsf{new}}$ for further processing. Redundancy checking is used as a mechanism to avoid recomputing unnecessary consequences that were already obtained via unfolding.

PROPOSITION 12. *Let* $\mathcal{U}$ *be a CQ, let* $\mathcal{O}$ *be an ontology, and let* $\mathbf{Q}$ *be a set of triples of the form* $\langle \mathcal{Q}', \mathcal{R}', \rightsquigarrow_{\mathcal{R}'} \rangle$ *where* $\mathcal{R}', \rightsquigarrow$ *are outputs of a rewriting algorithm for some CQ* $\mathcal{Q}'$ *over* $\mathcal{O}$. *If* Γ *used in Algorithm 2 is a converging inference system then given* $\mathcal{U}, \mathcal{Q}, \mathbf{Q}$ *algorithm* UNFOLDANDREWRITE *terminates and computes a rewriting for* $\mathcal{U}$ *with respect to* $\mathcal{O}$.

PROOF. *(Sketch:)* Soundness follows directly from the soundness of the unfolding and of the inference system Γ. Completeness follows from completeness of Γ and the structure of the algorithm that ensures that the set of clauses is saturated up to redundancy using Γ: all clauses in $\mathcal{O}$ are saturated initially and then all inferences possible having as a main premise a query are also performed (note that due to the special predicate $\mathcal{Q}_A$ in queries which does not appear anywhere in $\mathcal{O}$ or in the body of some other query, queries can never appear as side premises in an inference). Finally, termination follows by the convergence property of Γ. More precisely, there are finitely many symbols in $\mathcal{O} \cup \mathcal{U}$ and since Γ never increases ej-variables then, there is a bound on the largest clause that can be constructed using Γ. Hence, after a bounded number of steps only queries that are redundant would be infered in line 10 and the algorithm will stop adding elements to $\mathcal{R}_{\mathsf{new}}$. □

4.1 Further Optimisations and Refinements

Algorithm 2 exploits materialised rewritings only at a first stage, before any reasoning step has been applied. To maximise the use of materialised information, it is possible to interleave unfolding and reasoning (at least to a certain extent). This, however, requires utmost care, as successive

unfoldings could lead to non-termination; hence, a rather conservative strategy needs to be adopted.

EXAMPLE 13. *Consider* $\mathcal{U} = R(x,y) \wedge C(y) \rightarrow \mathcal{Q}_A(x)$ *and assume a rewriting for* $\mathcal{Q} = C(y) \rightarrow \mathcal{Q}_A(y)$ *is available, which consists of* $\mathcal{Q}$ *and* $\mathcal{Q}' = R(y',z') \wedge C(z') \rightarrow \mathcal{Q}_A(y')$. *Clearly,* $\mathcal{Q}$ *is relevant to* $\mathcal{U}$ *and unfolding* $\mathcal{U}$ *w.r.t.* $\mathcal{Q}'$ *yields* $\mathcal{U}' = R(u,v) \wedge R(v,w) \wedge C(w) \rightarrow \mathcal{Q}_A(u)$. *Subsequently,* $\mathcal{Q}$ *is relevant to* $\mathcal{U}'$ *and we can unfold it w.r.t.* $\mathcal{Q}'$ *to obtain a new query with an increased number of variables. Clearly, this process does not terminate.* ♢

A potential performance bottleneck is in line 10, where Algorithm 2 can perform unnecessary inferences. For example, if the selected query $\mathcal{Q}_h$ has been obtained already via unfolding of $\mathcal{U}$ then the corresponding conclusion might have also been generated via unfolding. In Example 11 applying an inference on query $\mathcal{Q}_1$ could regenerate query $\mathcal{Q}_2$; although such query is identified as redundant, resources have already been spent in order to generate it. When the set of queries generated by unfolding is large, the previous issue can occur quite often and hence adversely affect performance. One can optimise this process further by establishing a mechanism for detecting inferences that can be skipped. For instance, in Example 10 it suffices to apply only one inference on query $\mathcal{Q}_2$ to obtain $\mathcal{Q}_3$.

One such mechanism can be put in place by taking into account that only safe unfoldings are being computed. Our implementation in Requiem takes the safety conditions explicitly into account to restrict (always relying on a rather conservative strategy) the types of allowed resolution inferences while preserving completeness.

5. EVALUATION

We have implemented an algorithm that uses precomputed information in a prototype tool called QuDec, which relies on Requiem for performing reasoning. We have evaluated QuDec using both benchmark and realistic ontologies expressed in the logics DL-Lite$_R$ and $\mathcal{ELHI}$.

Regarding, DL-Lite$_R$, we used several benchmark ontologies and their respective test queries (denoted by $\mathcal{E}$) that have been proposed and used in [19], namely the Vicodi ontology (V), the South African National Accessibility Portal ontology (A), the European Union financial institutions ontology (S), the University ontology (U), and the artificial ontology P5; some of these ontologies contain a normalised version called AX, UX, and P5X (see [19] for details about these ontologies). Additionally, we used the queries (denoted by $\mathcal{B}$) computed automatically by the query generation tool SyGENiA [15, 7]. The largest set $\mathcal{B}$ had 1754 CQs while the smallest had 15 CQs. Moreover, we used a DL-Lite$_R$ fragment of the medical ontology GALEN (denoted by G), which contains nearly 60,000 axioms, together with 10 manually constructed queries (also denoted by $\mathcal{E}$). Regarding, $\mathcal{ELHI}$ we used a version of the LUBM ontology (L),[1] a photography ontology (F),[2] and NASA's SWEET 1.1 ontology (N).[3] Regarding test queries, we again used both synthetically generated queries (set $\mathcal{B}$) and manually generated queries (set $\mathcal{E}$). The largest synthetically generated query set was for ontology NS (with 883 CQs) and the smallest one was for ontology L (with 79 CQs). For L we also used the 14 test queries in the LUBM benchamrk [13], while for F and N we manually constructed 5 test queries.

To run QuDec we have used two different sets of precomputed rewritings $\mathbf{Q}$; the first, denoted by $\mathbf{Q}^{\mathsf{at}}$, consists of rewritings computed for all single-atom queries of the form $\alpha \rightarrow Q_A(x)$, where α is either $A(x)$ or $R(x,y)$ and A, R occur in the ontology; the second, denoted by $\mathbf{Q}^{\mathsf{tr}}$, contains all queries generated by SyGENiA. The respective configurations of QuDec are denoted by QuDec$^{\mathsf{at}}$ and QuDec$^{\mathsf{tr}}$, respectively. Finally, QuDec$^{\emptyset}$ denotes the configuration where $\mathbf{Q} = \emptyset$, which corresponds to a standard execution of Requiem.

Figure 1 presents the rewriting computation time (in milliseconds) for our tests. The three bars provided for each query set correspond to the configurations QuDec$^{\mathsf{at}}$, QuDec$^{\mathsf{tr}}$ and QuDec$^{\emptyset}$ in that order. The lower part of each bar (in darker colour) corresponds to the time required to search for relevant queries; the middle part (in gray) to the rewriting time; finally, the upper part (in lighter colour) to the time spent in redundancy elimination. All values are average computation times for each test query set.

First, we note that the time to search for relevant queries is negligible (in most figures the dark gray parts of the bar are barely visible). Second, it is clear from the figure that QuDec$^{\mathsf{tr}}$ significantly outperforms all other configurations. This is expected when ran over the test queries in $\mathcal{B}$ since all of them have an exact relevant query in $\mathbf{Q}^{\mathsf{tr}}$, hence QuDec$^{\mathsf{tr}}$ only has to retrieve the relevant precomputed rewriting. However, QuDec$^{\mathsf{tr}}$ also achieves significantly better performance on all other test queries. Third, even the simpler configuration QuDec$^{\mathsf{at}}$ is generally much more efficient than QuDec$^{\emptyset}$. It is, however, slower on ontology A. After inspecting the computed rewritings we concluded that this is because in this ontology QuDec$^{\mathsf{at}}$ produces significantly larger rewritings than QuDec$^{\emptyset}$. More precisely, the replacement approach generates a large number of redundant and overlapping queries. Finally, we note that QuDec$^{\mathsf{at}}$ was able to compute rewritings for the 10 queries over GALEN in just 3 seconds.

6. RELATED WORK AND CONCLUSIONS

Reformulating queries using database views (possibly in the presence of dependencies) is an important problem for several database applications, like query optimisation [11, 16], data integration [1, 8, 3, 12, 23] and privacy-aware data access [6] (see also [14]).

Our techniques bear a resemblance to those typically exploited for query answering using views. There are, however, two important differences. First, given a view $\mathcal{V}$ with head atom $V(\vec{x})$ and a query $\mathcal{Q}$, the goal in all previous approaches is to rewrite $\mathcal{Q}$ such that it uses the *head predicate* V (cf. [16, Example 2.3]). The expectation is that the materialised information for the (possibly virtual) table $V(\vec{x})$ will help reduce the size of subsequent joins in $\mathcal{Q}$. Second, database dependencies are interpreted as constraints which must be satisfied by all input databases. In the context of ontologies, however, data is typically considered to be *incomplete* and query answering over a set of rules is tantamount to a first-order logic entailment problem. Similar techniques to query rewriting using database views were used by the authors in [21] for replacing parts of a query with existing database view definition. Hence, again their respec-

[1] http://swat.cse.lehigh.edu/projects/lubm/
[2] http://www.co-ode.org/ontologies/photography/
[3] http://sweet.jpl.nasa.gov/ontology/

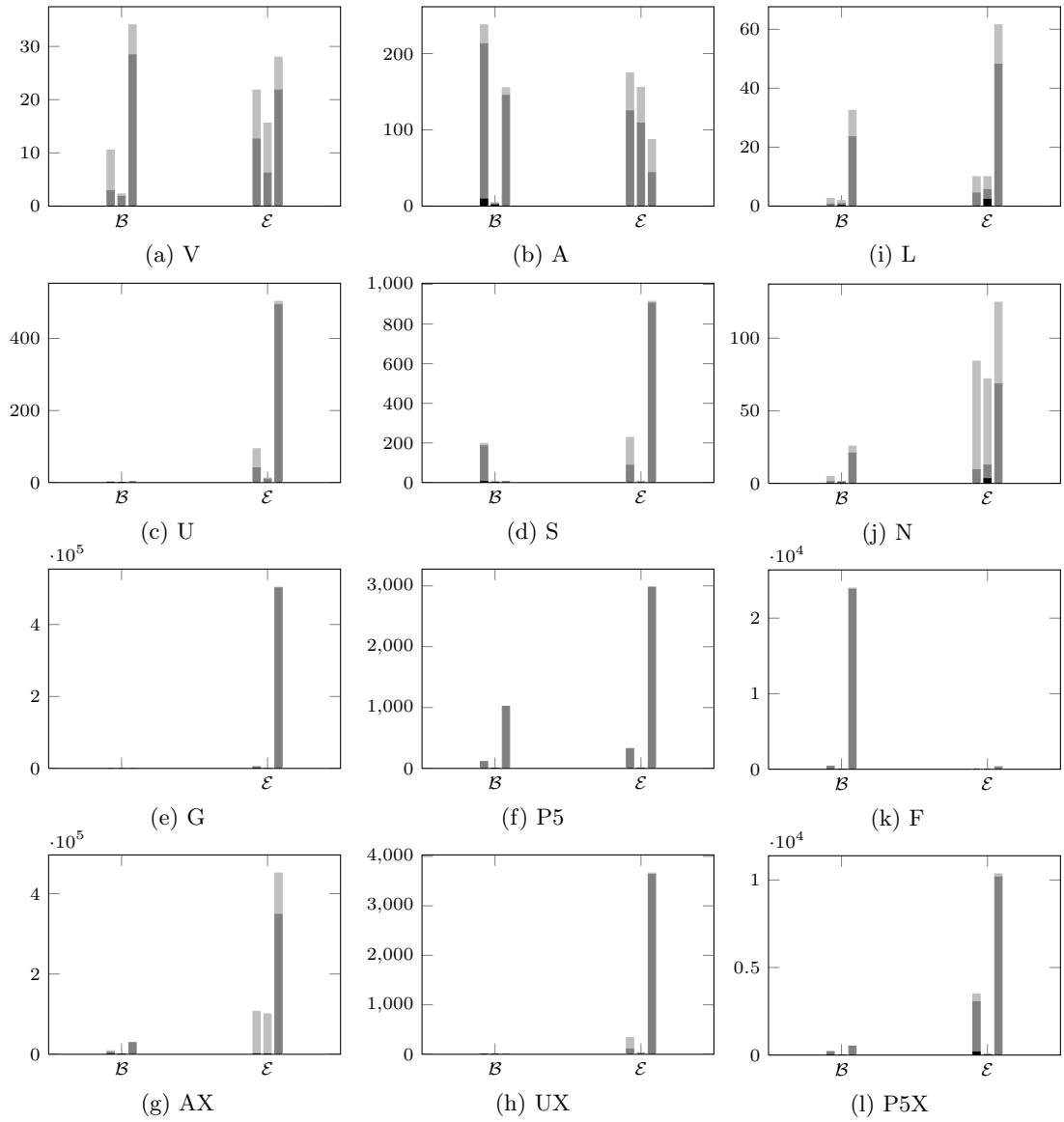

Figure 1: Computation times for each system and ontology.

tive definitions of relevance and safety are different than ours and are closer to those defined in the field of databases.

To the best of our knowledge, the problem of reformulating ontological queries using precomputed information has so far not been addressed in the literature. To bridge this gap, we have proposed a sound reformulation algorithm that can be used in OBDA applications where users have restricted access to information. We have then proposed a sound and complete algorithm which exploits materialised rewritings to speed up computation. Our experimental results are very encouraging and suggest that the use of even a small set of materialised rewritings can lead to significant performance improvement.

Concerning future work, we plan to investigate weaker relevance notions, in order to increase the amount of precomputed information that we can take advantage of. Moreover, we plan to develop further optimisation techniques and evaluate the usefulness of our approach for query answering under restricted access to information.

7. ACKNOWLEDGMENTS

The work of Giorgos Stoilos was partially funded by a Marie Curie FP7 Career Reintegration Grant within European Union's 7th Framework Programme under REA grant agreement 303914. The work of Bernardo Cuenca Grau was partially supported by the Royal Society under a University Research Fellowship.

8. REFERENCES

[1] F. N. Afrati and N. Kiourtis. Computing certain answers in the presence of dependencies. *Inf. Syst.*, 35(2):149–169, 2010.

[2] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Journal of Artificial Intelligence*, 175(9–10):1620–1654, 2011.

[3] Q. Bai, J. Hong, M. F. McTear, and H. Wang. A bucket-based approach to query rewriting using views in the presence of inclusion dependencies. *Journal of Research and Practice in Information Technology*, 38(3):251–266, 2006.

[4] A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *Proc. of LICS 2010*, pages 228–242, 2010.

[5] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

[6] D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. View-based query answering over description logic ontologies. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 242–251, 2008.

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

[8] A. Deutsch, L. Popa, and V. Tannen. Query reformulation with constraints. *SIGMOD Rec.*, 35(1):65–73, 2006.

[9] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997.

[10] T. Eiter, M. Ortiz, M. Simkus, T.-K. Tran, and G. Xiao. Query rewriting for Horn-$\mathcal{SHIQ}$ plus rules. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2012.

[11] J. Goldstein and P.-A. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *SIGMOD Management of data*, pages 331–342. ACM, 2001.

[12] J. Gryz. Query rewriting using views in the presence of functional and inclusion dependencies. *Information Systems*, 24(7):597–612, 1999.

[13] Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics*, 3(2):158–182, 2005.

[14] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

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

[16] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 95–104, 1995.

[17] B. Motik, R. Shearer, and I. Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.

[18] G. Orsi and A. Pieris. Optimizing query answering under ontological constraints. *VLDB Endowment*, 4(11):1004–1015, 2011.

[19] H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient Query Answering for OWL 2. In *8th International Semantic Web Conference (ISWC 2009)*, October 2009.

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

[21] F. D. Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT 2013)*, pages 561–572, 2013.

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

[23] R. Pottinger and A. Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB Journal*, 10(2-3):182–198, 2001.

[24] M. Rodriguez-Muro and D. Calvanese. High performance query answering over DL-Lite ontologies. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, 2012.

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

[26] R. Rosati and A. Almatelli. Improving query answering over DL-Lite ontologies. In *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR 2010)*, 2010.

[27] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

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

[29] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for physical data independence. *VLDB Journal*, 5(2):101–118, 1996.

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

[31] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of 23rd International Conference on Very Large Data Bases*, pages 136–145, 1997.