Optimizing FOL reducible query answering

Damian Bursztyn, François Goasdoué, Ioana Manolescu

Optimizing FOL reducible query answering

Damian Bursztyn
INRIA & U. Paris-Sud, France
damian.burstzyn@inria.fr

François Goasdoué
U. Rennes 1 & INRIA, France
fg@irisa.fr

Ioana Manolescu
INRIA & U. Paris-Sud, France
ioana.manolescu@inria.fr

ABSTRACT

We devise a query optimization framework for formalisms enjoying FOL reducibility of query answering, for which it reduces to the evaluation of a FOL query against facts. This framework allows searching within a set of alternative equivalent FOL queries, i.e., FOL reformulations, one with minimal evaluation cost when evaluated through a relational database system. We provide two algorithms, an exhaustive and a greedy, for exploring this space of alternatives. We apply this framework to the DL-Lite$_\mathcal{R}$ description logic underpinning the W3C's OWL2 QL profile; an experimental evaluation validates the interest and applicability of our technique.

1. INTRODUCTION

Answering queries under deductive constraints is a longstanding database research topic [2]. It recently gained the interest of the database community at large, with the advent of *ontology-based data access* (OBDA) [18]. The latter consists in querying *facts* described by an application domain conceptualization, in the spirit of an ER models or UML class diagram, expressed by *deductive constraints*. Several languages have been investigated for expressing such constraints, e.g., Datalog± [9], Description Logics [3] and Existential Rules [4] for logical facts, or RDF Schema for RDF graphs[1]. In this context, *query answering* denotes the task of computing the answer to the given query, by taking into account both the facts and the constraints holding on them, in contrast with the more common *query evaluation* task, which just needs to examine the stored facts.

A prominent trend in the OBDA literature is to identify dialects of the above mentioned languages, enjoying *first-order logic (*FOL*) reducibility (or rewritability)* of query answering, e.g., [10, 9, 20]. For such languages, query answering reduces to the evaluation of a FOL query against the *facts*, obtained by compiling the domain contraints into the query. The so-called FOL *reformulations* thus obtained, correspond to SQL statements, which can be handed over for evaluation to an efficient relational database management system (RDBMS) storing the facts. However, FOL reformulations encoding domain constraints tend to be (significantly) more complex than the typical queries that RDBMS optimizers were tuned for, in particular due to repeated sub-queries, thus RDBMSs perform overall poorly at evaluating them. Therefore, OBDA optimizations investigated so far mainly aim to produce *fast*, for specific logical query, constraint and reformulation dialects, FOL reformulations as *minimal* as possible to limit redundancy, e.g., [24, 14, 23, 11, 29, 17, 30, 28, 15].

In this paper, our approach is broader, and performance-oriented. We devise a query optimization framework for *any* logical OBDA setting enjoying FOL reducibility of query answering [7]. We reduce query answering to the evaluation of *alternative* FOL queries belonging to richer languages than those considered so far in the literature; in particular, this may allow *several* (equivalent) FOL reformulations for the input query. This contrasts with related works (cf. Section 7), which aim at a *single* FOL reformulation (modulo minimization). Allowing a variety of reformulations is crucial for efficiency, as such alternatives, while computing the same answers, may have very different evaluation costs when evaluated through an RDBMS. Therefore, instead of having a single default choice which may perform poorly, we select one with lowest estimated evaluation cost among the possible alternatives. Figure 1 illustrates our query optimization framework.

We also apply this framework to the lightweight DL-Lite$_\mathcal{R}$ description logic [10] underpinning the popular W3C's OWL2 QL standard for rich Semantic Web applications[2]. Query answering in DL-Lite$_\mathcal{R}$ has received significant attention in the literature, notably techniques based on FOL reducibility, e.g., [10, 1, 24, 26, 11, 29].

Contributions. The contributions we bring to the problem of optimizing FOL reducible query answering can be outlined as follows:

1. For logical formalisms enjoying FOL reducibility of query answering, we provide a general *optimization framework* that reduces query answering to searching among a set of alternative equivalent FOL queries, i.e., FOL reformulations, one with minimal evaluation cost in an RDBMS (Section 3).

2. We characterize interesting spaces of such alternative equivalent FOL queries for DL-Lite$_\mathcal{R}$ (Section 4), called JUCQ or JUSCQ reformulations.

3. We then optimize query answering in the setting of DL-Lite$_\mathcal{R}$ by searching within one of these spaces and picking a reformulation with lowest (estimated) evaluation cost w.r.t. an RDBMS cost model. We provide two algorithms, an exhaustive and a greedy, for selecting a low-cost reformulation (Section 5).

[1] http://www.w3.org/2001/sw/Specs.html

[2] http://www.w3.org/TR/owl2-overview

BDA 29 septembre 2015, Île de Porquerolles, France.

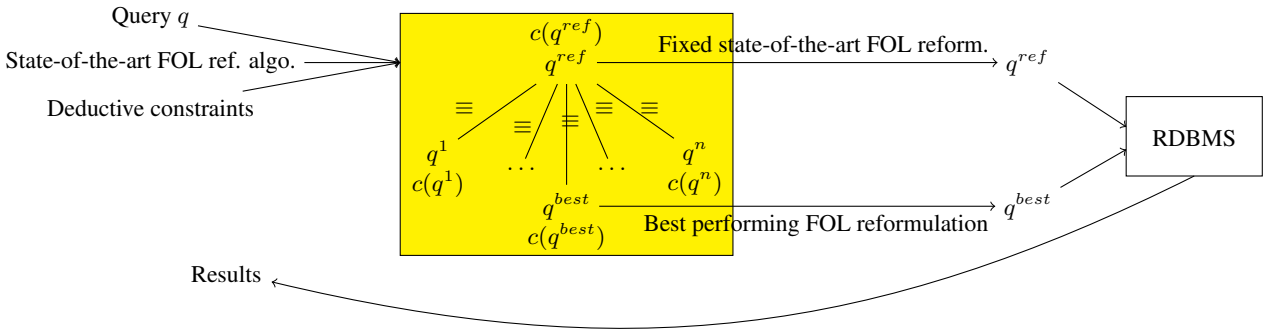

Figure 1: Outline of our optimized FOL reformulation approach.

4. Finally, we demonstrate experimentally the effectiveness and the efficiency of our query answering technique for DL-Lite$_\mathcal{R}$, by deploying it on top of the Postgres RDBMS (Section 6).

In the sequel, Section 2 recall preliminary notions on DL-Lite$_\mathcal{R}$. Then, we detail the above contributions, and finally we discuss related work and conclude in Section 7.

2. PRELIMINARIES

We recall the basics of the DL-Lite$_\mathcal{R}$ description logic [10], on which the OWL2 QL W3C standard for managing semantic-rich Web data is built. DL-Lite$_\mathcal{R}$ is a fragment of Datalog$\pm$ and of Existential Rules; it largely extends the subset of RDF (comprising RDF Schema constraints) which translates into description logics.

2.1 DL-Lite knowledge bases

A *knowledge base (KB)* $\mathcal{K}$ consists of a TBox $\mathcal{T}$ (ontology, or axiom set) and an ABox $\mathcal{A}$ (fact set, or dataset), denoted $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, with $\mathcal{T}$ expressing constraints on $\mathcal{A}$.

$\mathcal{T}$ and $\mathcal{A}$ are built from a set N_C of *concept names* (unary predicates), a set N_R of *role names* (binary predicates), and a set N_I of *individuals* (constants). The *ABox* consists of a finite number of *concept assertions* of the form $A(a)$ with $A \in N_C$ and $a \in N_I$, and of *role assertions* of the form $R(a, b)$ with $R \in N_R$ and $a, b \in N_I$. The *TBox* is a set of axioms that can be *concept inclusions* of the form $B \sqsubseteq C$, or *role inclusions* of the form $Q \sqsubseteq S$, formed using the following grammar (where $A \in N_C$ and $R \in N_R$):

$$B := A \mid \exists Q,\ C := B \mid \neg B,\ Q := R \mid R^-,\ S := Q \mid \neg Q$$

Remark that negation can *only* be used in the right-hand side of inclusion constraints. Constraints featuring such negation are called *disjointness* (or *exclusion*) constraints.

The semantics of inclusion constraints is defined, as customary, in terms of their FOL interpretations. Tables 1 and 2 provide the FOL and relational notations expressing these constraints equivalently.

An *interpretation* has the form $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty set and $\cdot^I$ is a function mapping each $a \in N_I$ to $a^I \in \Delta^I$, each $A \in N_C$ to $A^I \subseteq \Delta^I$, and each $R \in N_R$ to $R^I \subseteq \Delta^I \times \Delta^I$. The function $\cdot^I$ is straightforwardly extended to concepts and roles:

- projection: $(\exists Q)^I = \{o_1 \mid \exists o_2\ (o_1, o_2) \in Q^I\}$
- inverse: $(R^-)^I = \{(o_1, o_2) \mid (o_2, o_1) \in R^I\}$
- negation: $(\neg B)^I = \Delta^I \setminus B^I$ and $(\neg Q)^I = (\Delta^I \times \Delta^I) \setminus Q^I$

An interpretation I satisfies an inclusion $B \sqsubseteq C$ (resp. $Q \sqsubseteq S$) if $B^I \subseteq C^I$ (resp. if $Q^I \subseteq S^I$); it satisfies $A(a)$ (resp. $R(a, b)$) if $a^I \in A^I$ (resp. $(a^I, b^I) \in R^I$).

An interpretation I is a *model* of $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ if I satisfies all inclusions in $\mathcal{T}$ and assertions in $\mathcal{A}$. A KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is *consistent* if it has a model. In this case, we say that $\mathcal{A}$ is $\mathcal{T}$*-consistent*.

Finally, an inclusion or assertion α is *entailed* by a KB $\mathcal{K}$, written $\mathcal{K} \models \alpha$, if α is satisfied in all the models of $\mathcal{K}$.

Example 1 (DL-Lite$_\mathcal{R}$ data model)**.** *Consider the DL-Lite$_\mathcal{R}$ TBox $\mathcal{T}$ below expressing contraints on the atomic concepts* `Researcher` *and* `PhDStudent`*, and atomic roles* `worksWith` *and* `supervisedBy`.

$$\begin{array}{lrcl}
(T1) & \texttt{PhDStudent} & \sqsubseteq & \texttt{Researcher} \\
(T2) & \exists\texttt{worksWith} & \sqsubseteq & \texttt{Researcher} \\
(T3) & \exists\texttt{worksWith}^- & \sqsubseteq & \texttt{Researcher} \\
(T4) & \texttt{worksWith} & \sqsubseteq & \texttt{worksWith}^- \\
(T5) & \texttt{supervisedBy} & \sqsubseteq & \texttt{worksWith} \\
(T6) & \exists\texttt{supervisedBy} & \sqsubseteq & \texttt{PhDStudent} \\
(T7) & \texttt{PhDStudent} & \sqsubseteq & \neg\exists\texttt{supervisedBy}^-
\end{array}$$

It states that PhD students are researchers $(T1)$*, researchers work with researchers* $(T2)(T3)$*, working with someone is a symmetric relation* $(T4)$*, being supervised by someone implies working with her/him* $(T5)$*, only PhD students are supervised* $(T6)$ *and they cannot supervise* $(T7)$.

Consider now the following ABox $\mathcal{A}$ for the same atomic concepts and roles:

$$\begin{array}{ll}
(A1) & \texttt{worksWith}(Ioana, Francois) \\
(A2) & \texttt{supervisedBy}(Damian, Ioana) \\
(A3) & \texttt{supervisedBy}(Damian, Francois)
\end{array}$$

It states that Ioana works with François $(A1)$*, Damian is supervised by both Ioana* $(A2)$ *and François* $(A3)$.

It is worth noting that the knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ entails many constraints and assertions. For instance:

- $\mathcal{K} \models \exists\texttt{supervisedBy} \sqsubseteq \neg\exists\texttt{supervisedBy}^-$*, i.e., the two attributes of* `supervisedBy` *are disjoints, due to* $(T6) + (T7)$*;*
- $\mathcal{K} \models \texttt{worksWith}(Francois, Ioana)$*, i.e., François works with Ioana, due to* $(T4) + (A1)$*;*
- $\mathcal{K} \models \texttt{PhDStudent}(Damian)$*, i.e., Damian is a PhD student, due to* $(A2) + (T6)$*;*
- $\mathcal{K} \models \texttt{worksWith}(Francois, Damian)$*, i.e., François works with Damian, due to* $(A3) + (T5) + (T4)$.

Finally remark that $\mathcal{A}$ is $\mathcal{T}$-consistent, i.e., there is no violation of $(T7)$*, since the KB $\mathcal{K}$ does not entail that some PhD student supervises another.*

DL constraint	FOL constraint	Relational constraint (under Open World Assumption)
$A \sqsubseteq A'$	$\forall x[A(x) \Rightarrow A'(x)]$	$A \subseteq A'$
$A \sqsubseteq \exists R$	$\forall x[A(x) \Rightarrow \exists y R(x,y)]$	$A \subseteq \Pi_1(R)$
$A \sqsubseteq \exists R^-$	$\forall x[A(x) \Rightarrow \exists y R(y,x)]$	$A \subseteq \Pi_2(R)$
$\exists R \sqsubseteq A$	$\forall x[\exists y R(x,y) \Rightarrow A(x)]$	$\Pi_1(R) \subseteq A$
$\exists R^- \sqsubseteq A$	$\forall x[\exists y R(y,x) \Rightarrow A(x)]$	$\Pi_2(R) \subseteq A$
$\exists R' \sqsubseteq \exists R$	$\forall x[\exists y R'(x,y) \Rightarrow \exists z R(x,z)]$	$\Pi_1(R') \subseteq \Pi_1(R)$
$\exists R' \sqsubseteq \exists R^-$	$\forall x[\exists y R'(x,y) \Rightarrow \exists z R(z,x)]$	$\Pi_1(R') \subseteq \Pi_2(R)$
$\exists R'^- \sqsubseteq \exists R$	$\forall x[\exists y R'(y,x) \Rightarrow \exists z R(x,z)]$	$\Pi_2(R') \subseteq \Pi_1(R)$
$\exists R'^- \sqsubseteq \exists R^-$	$\forall x[\exists y R'(y,x) \Rightarrow \exists z R(z,x)]$	$\Pi_2(R') \subseteq \Pi_2(R)$
$R \sqsubseteq R'^-$ or $R^- \sqsubseteq R'$	$\forall x,y[R(x,y) \Rightarrow R'(y,x)]$	$R \subseteq \Pi_{2,1}(R')$ or $\Pi_{2,1}(R) \subseteq R'$
$R \sqsubseteq R'$ or $R^- \sqsubseteq R'^-$	$\forall x,y[R(x,y) \Rightarrow R'(x,y)]$	$R \subseteq R'$ or $\Pi_{2,1}(R) \subseteq \Pi_{2,1}(R')$

Table 1: DL-Lite$_\mathcal{R}$ inclusion constraints *without* negation in FOL, and in relational notation; A, A' are concept names while R, R' are role names. For the relational notation, we use 1 to designate the first attribute of any atomic role, and 2 for the second.

DL constraint	FOL constraint	Relational constraint (under Open World Assumption)
$A \sqsubseteq \neg A'$	$\forall x[A(x) \Rightarrow \neg A'(x)]$	$A \cap A' \subseteq \bot$
$A \sqsubseteq \neg\exists R$	$\forall x[A(x) \Rightarrow \neg\exists y R(x,y)]$	$A \cap \Pi_1(R) \subseteq \bot$
$A \sqsubseteq \neg\exists R^-$	$\forall x[A(x) \Rightarrow \neg\exists y R(y,x)]$	$A \cap \Pi_2(R) \subseteq \bot$
$\exists R \sqsubseteq \neg A$	$\forall x[\exists y R(x,y) \Rightarrow \neg A(x)]$	$A \cap \Pi_1(R) \subseteq \bot$
$\exists R^- \sqsubseteq \neg A$	$\forall x[\exists y R(y,x) \Rightarrow \neg A(x)]$	$A \cap \Pi_2(R) \subseteq \bot$
$\exists R' \sqsubseteq \neg\exists R$	$\forall x[\exists y R'(x,y) \Rightarrow \neg\exists z R(x,z)]$	$\Pi_1(R') \cap \Pi_1(R) \subseteq \bot$
$\exists R' \sqsubseteq \neg\exists R^-$	$\forall x[\exists y R'(x,y) \Rightarrow \neg\exists z R(z,x)]$	$\Pi_1(R') \cap \Pi_2(R) \subseteq \bot$
$\exists R'^- \sqsubseteq \neg\exists R$	$\forall x[\exists y R'(y,x) \Rightarrow \neg\exists z R(x,z)]$	$\Pi_2(R') \cap \Pi_1(R) \subseteq \bot$
$\exists R'^- \sqsubseteq \neg\exists R^-$	$\forall x[R'(y,x) \Rightarrow \neg\exists z R(z,x)]$	$\Pi_2(R') \cap \Pi_2(R) \subseteq \bot$
$R \sqsubseteq \neg R'^-$ or $R^- \sqsubseteq \neg R'$	$\forall x,y[R(x,y) \Rightarrow \neg R'(y,x)]$	$R \cap \Pi_{2,1}(R') \subseteq \bot$ or $\Pi_{2,1}(R) \cap R' \subseteq \bot$
$R \sqsubseteq \neg R'$ or $R^- \sqsubseteq \neg R'^-$	$\forall x,y[R(x,y) \Rightarrow \neg R'(x,y)]$	$R \cap R' \subseteq \bot$ or $\Pi_{2,1}(R) \cap \Pi_{2,1}(R') \subseteq \bot$

Table 2: DL-Lite$_\mathcal{R}$ inclusion constraints *with* negation, i.e., disjointness constraints, in FOL and relational notation. $\bot$ denotes the empty relation.

2.2 Queries

A FOL query is a FOL formula $\phi(\bar{x})$ whose free variables are $\bar{x}$. The *arity* of a query is the number of its free variables, e.g., 0 for a *Boolean* query. Given a FOL interpretation $I = (\Delta^I, \cdot^I)$, the semantics of a Boolean query is defined as true if $\phi()^I = \text{true}$ and false otherwise, while the semantics of a non-Boolean query of arity $n \geq 1$ is the relation of arity n defined on Δ^I as follows: $\{\bar{\mathbf{o}} \in (\Delta^I)^n \mid \phi(\bar{\mathbf{o}})^I = \text{true}\}$. An interpretation that makes true a Boolean FOL query, or non-empty a non-Boolean FOL query, is a *model* of that query.

Given a query q asked against a KB $\mathcal{K}$:

- If q is Boolean, the answer set of q over $\mathcal{K}$ is defined as: $ans(q,\mathcal{K}) = \{\langle\rangle\}$ if $\mathcal{K} \models q()$, with $\langle\rangle$ the empty tuple, and $ans(q,\mathcal{K}) = \emptyset$ otherwise.
- If q is a non-Boolean query of arity n, the answer set of q against $\mathcal{K}$ is: $ans(q,\mathcal{K}) = \{\bar{t} \in (N_I)^n \mid \mathcal{K} \models q(\bar{t})\}$ where $\mathcal{K} \models q(\bar{t})$ means as usual that every model of $\mathcal{K}$ is a model of $q(\bar{t})$.

Example 2 (Query answering)**.** *Consider the* FOL *query asking for the PhD students with whom someone works:*

$$\exists y\ \texttt{PhDStudent}(x) \wedge \texttt{worksWith}(y,x)$$

Given the KB $\mathcal{K}$ of the preceding example, the answer set of this query is $\{Damian\}$, since $\mathcal{K} \models \texttt{PhDStudent}(Damian)$ and $\mathcal{K} \models \texttt{worksWith}(Francois, Damian)$ hold.

FOL query dialects. In this paper, we focus on the following FOL query sublanguages. *Conjunctive Queries* (CQs), the core database queries a.k.a. Select-Project-Join queries, are conjunctions of atoms whose bound variables are existentially quantified, and whose atoms are of the forms $A(t)$ or $R(t,t')$ with t, t' variables or individuals. *Semi-Conjunctive Queries* (SCQs) are conjunctions of disjunctions of single-atom CQs with the same arity, where the atom is either of the form $A(t)$ or of the form $R(t,t')$ as above; the bound variables of SCQs are also existentially quantified. *Unions of* CQs (UCQs) are *disjunctions* of CQs with same arity; similarly *Unions of* SCQs (USCQs) are *disjunctions* of SCQs with same arity. Finally, *Joins of* UCQs (JUCQs) are *conjunctions* of UCQs; similarly *Joins of* USCQs (JUSCQs) are *conjunctions* of USCQs. All the abovementioned query languages are easily translated into SQL and thus can be evaluated by an RDBMS.

Notations. We write a FOL query $q(\bar{x}) \leftarrow \phi(\bar{x})$ and we refer to it by its *name* q or its *head* $q(\bar{x})$ when we also need to refer to its free variables; $\phi(\bar{x})$ is the *body* of q. In the queries we consider, existential variables in $\phi(\bar{x})$ are not explicitly quantified; they correspond to those which are not in $\bar{x}$.

Unless otherwise specified, we systematically use q to refer to the input CQ, $a_1, \ldots, a_n$ to designate the atoms in the body of q, $\mathcal{T}$ to designate a DL-Lite$_\mathcal{R}$ TBox, and $\mathcal{A}$ for the ABox.

FOL-reducibility of data management. DL-Lite$_\mathcal{R}$ has been designed so that the core data management tasks of deciding KB consistency and of query answering are FOL-*reducible* [10]. This property allows reducing a data management task over a KB $\mathcal{K} = \langle\mathcal{T},\mathcal{A}\rangle$ to the evaluation against $\mathcal{A}$ *only* of a FOL query computed using $\mathcal{T}$ *only*.

The interest of FOL-reducibility is to perform a data management task in two separate steps: a first reasoning step that begets the FOL query, and a second step which evaluates that query in a purely relational fashion, i.e., within an RDBMS after translation to SQL. These systems benefit from decades of research and development invested in optimizing data stores, and query evaluation engines; thus, they hold great potential for efficient FOL query evaluation.

The literature provides techniques for computing FOL reformulations of a CQ in settings related to DL-Lite$_\mathcal{R}$. They produce (i) a UCQ w.r.t. a DL-Lite$_\mathcal{R}$ TBox, e.g., [10, 1, 24, 11, 29], or extensions thereof using existential rules [17] or Datalog$\pm$ [30, 15], (ii) a USCQ [28] w.r.t. a set of existential rules generalizing a DL-Lite$_\mathcal{R}$ TBox, and (iii) a set of alternative equivalent JUCQs [8] w.r.t. an RDF database [13], whose RDF Schema constraints is the following subset of the DL-Lite$_\mathcal{R}$ TBox ones: $A_1 \sqsubseteq A_2$, $R_1 \sqsubseteq R_2$, $\exists R \sqsubseteq A$ and $\exists R^- \sqsubseteq A$.

Finally, we present the pioneering CQ-to-UCQ reformulation technique for DL-Lite$_\mathcal{R}$ [10], on which we rely in order to establish our results. These results extend to any other FOL reformulation techniques for DL-Lite$_\mathcal{R}$, e.g., optimized CQ-to-UCQ or CQ-to-USCQ reformulation techniques, since they produce equivalent FOL queries.

The technique of [10] relies on two operations: *the specialization of an atom into another* through backward chaining on a TBox constraint (recall that such constraints are particular FOL rules, see Tables 1 and 2), and *the specialization of two atoms into one* through their most general unifier. The technique consists in exhaustively applying these two operations to the input CQ, each operation generating a CQ subsumed by the input CQ. The same is then recursively applied on the subsumed CQs thus obtained, until reaching a fixpoint for the set of generated CQs. The finite union of the input CQ and generated ones forms the UCQ reformulation of the input CQ w.r.t. the TBox. The next example illustrates this.

Example 3 (CQ-to-UCQ reformulation technique)**.** *Consider the query* $q(x) \leftarrow \texttt{PhDStudent}(x) \wedge \texttt{worksWith}(y, x)$ *and KB* $\mathcal{K}$ *of the preceding examples. The* UCQ *reformulation of* q *is*

$$
\begin{aligned}
q^{\texttt{UCQ}}(x) \leftarrow q^1(x) \leftarrow\ & \texttt{PhDStudent}(x) \wedge \texttt{worksWith}(y, x)\\
\vee\ & q^2(x) \leftarrow \texttt{PhDStudent}(x) \wedge \texttt{worksWith}(x, y)\\
\vee\ & q^3(x) \leftarrow \texttt{PhDStudent}(x) \wedge \texttt{supervisedBy}(y, x)\\
\vee\ & q^4(x) \leftarrow \texttt{PhDStudent}(x) \wedge \texttt{supervisedBy}(x, y)\\
\vee\ & q^5(x) \leftarrow \texttt{supervisedBy}(x, z) \wedge \texttt{worksWith}(y, x)\\
\vee\ & q^6(x) \leftarrow \texttt{supervisedBy}(x, z) \wedge \texttt{worksWith}(x, y)\\
\vee\ & q^7(x) \leftarrow \texttt{supervisedBy}(x, z) \wedge \texttt{supervisedBy}(y, x)\\
\vee\ & q^8(x) \leftarrow \texttt{supervisedBy}(x, z) \wedge \texttt{supervisedBy}(x, y)\\
\vee\ & q^9(x) \leftarrow \texttt{supervisedBy}(x, x)\\
\vee\ & q^{10}(x) \leftarrow \texttt{supervisedBy}(x, y)
\end{aligned}
$$

where the first disjunct is q's body, the second is obtained from the first through a backward chaining application of $(T4)$ *on the atom* $\texttt{worksWith}(y, x)$*, the third (resp. forth) is obtained from the first (resp. second) through a backward chaining application of* $(T5)$ *on the atom* $\texttt{worksWith}(y, x)$ *(resp. on* $\texttt{worksWith}(x, y)$*). The next four disjuncts,* q^5 *to* q^8*, are obtained from the above four through a backward chaining application of* $(T6)$ *on the atom* $\texttt{PhDStudent}(x)$*. Finally, the ninth disjuncts is obtained from the seventh through the most general unifier of* $\texttt{supervisedBy}(x, z)$ *and* $\texttt{supervisedBy}(y, x)$*, while the tenth is obtained similarly from the eighth.*

It is worth noting that the (naive) exhaustive application of specialization steps leads, in general, to highly redundant reformulations w.r.t. the containment of their disjuncts. For instance, minimizing $q^{\texttt{UCQ}}$ *by eliminating disjuncts contained in another leads to*

$$
\begin{aligned}
q^{\texttt{UCQ}}_{min}(x) \leftarrow q^1(x) \leftarrow\ & \texttt{PhDStudent}(x) \wedge \texttt{worksWith}(y, x)\\
\vee\ & q^2(x) \leftarrow \texttt{PhDStudent}(x) \wedge \texttt{worksWith}(x, y)\\
\vee\ & q^3(x) \leftarrow \texttt{PhDStudent}(x) \wedge \texttt{supervisedBy}(y, x)\\
\vee\ & q^9(x) \leftarrow \texttt{supervisedBy}(x, y)
\end{aligned}
$$

in which all the disjuncts removed from $q^{\texttt{UCQ}}$ *are contained in* q^9*.*

Still, while minimal UCQ *reformulations can be obviously processed more efficiently than non-minimal ones, query engines overall perform poorly at evaluating them. This is mainly due to the repeated data accesses that are hardly optimized across union operators, e.g., above for* PhDStudent*,* worksWith *and* supervisedBy*:* CQ*s within a* UCQ *are basically processed independently one from another.*

3. OPTIMIZATION FRAMEWORK

In this section, we recall the notion of FOL reformulation which underpins FOL reducibility of query answering (Section 3.1). Then, for formalisms enjoying FOL reducibility of query answering, e.g., some DLs or some existential rules and Datalog$\pm$ fragments, we lay the principles of query answering based on *query covers*, which define a space of alternative equivalent FOL reformulations for a given query (Section 3.2), wherein we can search for a reformulation leading to the minimal evaluation cost in an RDBMS (Section 3.3).

3.1 FOL reducibility of query answering

FOL reducibility of query answering relies on FOL *reformulations* of queries.

Definition 1 (FOL reformulation)**.** *A* FOL reformulation $q^{\texttt{FOL}}$ *of* q *w.r.t.* $\mathcal{T}$ *is a* FOL *query such that* $ans(q, \langle \mathcal{T}, \mathcal{A} \rangle) = ans(q^{\texttt{FOL}}, \langle \emptyset, \mathcal{A} \rangle)$ *for any* $\mathcal{T}$*-consistent ABox* $\mathcal{A}$*.*

Observe that FOL reformulation reduces query answering against a KB to FOL query answering against a database storing the KB's ABox. The latter can hence be delegated to the query evaluation engine of an RDBMS.

3.2 Cover-based query answering

RDBMS query optimizers consider a set of *evaluation alternatives* (a.k.a. logical and physical plans), and select the one minimizing a *cost estimation function*. Since the number of alternatives is in $\mathcal{O}(2^n \times n!)$ for a CQ of n atoms [22], modern optimizers rely on heuristics to explore only a few alternatives; this works (very) well for *small-to-moderate size* CQs. However, FOL reformulations go *beyond* CQ*s* in general, and may be *extremely large* (see Section 6), leading the RDBMS to perform poorly.

To work around this limitation, we introduce the cover-based query answering technique to define a space of equivalent FOL reformulations of a CQ. A *cover* defines how the query is split into subqueries, that may overlap, called *fragment queries*, such that substituting each subquery with its FOL reformulation (obtained from any state-of-the-art technique) and joining the corresponding (reformulated) subqueries, *may* yield a FOL reformulation for the query to answer. Indeed, as we shall see soon, not every cover of a query leads to a FOL reformulation. However, every cover which does, yields an alternative *cover-based* FOL *reformulation* of the original query. Crucially for our problem, *a smart cover choice may lead to a cover-based reformulation whose evaluation is more efficient*. Thus, the cover-based technique amounts to *circumventing* the difficulty of modern RDBMSs to efficiently evaluate FOL reformulations in general.

Definition 2 (CQ cover)**.** *A* cover *of a query q, whose atoms are $\{a_1, \ldots, a_n\}$, is a set $C = \{f_1, \ldots, f_m\}$ of non-empty subsets of atoms of q, called* fragments*, such that (i) $\bigcup_{i=1}^{m} f_i = \{a_1, \ldots, a_n\}$ and (ii) no fragment is included into another.*

Definition 3 (Fragment queries of a CQ)**.** *Let $C = \{f_1, \ldots, f_m\}$ be a cover of q. A* fragment query *$q_{|f_i, 1 \leq i \leq m}$ of q w.r.t. C is the subquery whose body consists of the atoms in f_i and whose free variables are the free variables $\bar{x}$ of q appearing in the atoms of f_i,* plus *the existential variables in f_i that are shared with another fragment $f_{j, 1 \leq j \leq m, j \neq i}$, i.e., on which the two fragments join.*

Definition 4 (Cover-based FOL reformulation)**.** *Let $C = \{f_1, \ldots, f_m\}$ be a cover of q, and $q^{\texttt{FOL}}(\bar{x}) \leftarrow \bigwedge_{i=1}^{m} q^{\texttt{FOL}}_{|f_i}$ a FOL query, where $q^{\texttt{FOL}}_{|f_i}$, for $1 \leq i \leq m$, is a FOL reformulation w.r.t. $\mathcal{T}$ of the fragment query $q_{|f_i}$ of q.*

$q^{\texttt{FOL}}$ is a cover-based FOL reformulation of q w.r.t. $\mathcal{T}$ and C *if it is a FOL reformulation of q w.r.t. $\mathcal{T}$.*

3.3 Cover-based query optimization

We assume given a *query cost estimation function* ϵ which, for any FOL query q, returns the cost of evaluating it through an RDBMS storing the ABox. Thus, ϵ reflects the operations (selections, projections, joins, unions, scans etc) applied on the ABox to compute the results of a $q^{\texttt{FOL}}$ reformulation. The cost estimation ϵ also accounts for the effort needed to join the reformulated fragment query results. Note that any RDBMS storing the ABox includes such a cost estimation function, accessible for instance using the SQL `explain` directive. While the RDBMS optimizer explores several alternatives for evaluating a given FOL query, ϵ reflects the most efficient alternative found. One can also estimate costs outside the engine using well-known textbook formulas [25], as in e.g., [8].

Problem 1 (Optimization problem)**.** *Given a CQ q and a KB $\mathcal{K}$, the* cost-driven cover-based query answering problem *consists of finding a cover-based reformulation of q based on $\mathcal{K}$ with lowest (estimated) evaluation cost.*

4. COVER-BASED QUERY ANSWERING IN DL-Lite$_\mathcal{R}$

We now apply our general optimization framework to the particular case of the DL-Lite$_\mathcal{R}$ description logic. As stated in Definition 4, a query cover may not lead to a cover-based query reformulation. This is true even in the lightweight DL-Lite$_\mathcal{R}$ setting, as we shall see in the running example below, while it is not in the simpler RDF setting of [8]. We therefore provide a condition ensuring that a cover is *safe for query answering*, i.e., it leads to a FOL query reformulation. This results in a cover-based technique for computing a *set of* JUCQ *or of* JUSCQ *reformulations* of a CQ, when querying DL-Lite$_\mathcal{R}$ KBs. This extends [8] from RDF databases to DL-Lite$_\mathcal{R}$, and builds on the aforementioned state-of-the-art on UCQ and USCQ reformulations.

As already mentioned in Section 2, we use the simple CQ-to-UCQ reformulation technique of [10] for establishing our results, as well as for the running example for this section and the next one. Our results trivially extend to *any* other FOL reformulation techniques for DL-Lite$_\mathcal{R}$, e.g., optimized CQ-to-UCQ or CQ-to-USCQ reformulation techniques, since they produce *equivalent* (though possibly syntactically different) FOL queries.

Example 4 (Running example)**.** *Consider the KB $\mathcal{K} = \langle \mathcal{T} = \{B \sqsubseteq \exists R', R' \sqsubseteq R\}, \mathcal{A} = \{A(a), B(a)\}\rangle$ and the query $q(x) \leftarrow A(x) \wedge R(x, y) \wedge R'(z, y)$, whose only answer is a. The* UCQ *reformulation of q is*

$$
\begin{array}{rl}
q^{\texttt{UCQ}}(x) \leftarrow & q^1(x) \leftarrow (A(x) \wedge R(x, y) \wedge R'(z, y)) \\
& \vee\, q^2(x) \leftarrow (A(x) \wedge R'(x, y) \wedge R'(z, y)) \\
& \vee\, q^3(x) \leftarrow (A(x) \wedge R'(x, y)) \\
& \vee\, q^4(x) \leftarrow (A(x) \wedge B(x))
\end{array}
$$

where the first disjunct is q's body, the second is obtained from the first by specializing the atom $R(x, y)$ through a backward-chaining application of $R' \sqsubseteq R \in \mathcal{T}$, the third is obtained from the second through the most general unifier of $R'(x, y)$ and $R'(z, y)$, and the fourth results from the third by specializing $R'(x, y)$ through the backward-chaining application of $B \sqsubseteq \exists R' \in \mathcal{T}$.

Following Definition 4, the FOL *query obtained from q, its cover $C_1 = \{\{A(x), R(x, y)\}, \{R'(z, y)\}\}$, $\mathcal{T}$ and the* CQ-to-UCQ *reformulation technique is the* JUCQ*:*

$$
\begin{array}{rll}
q^{\texttt{JUCQ}}(x) \leftarrow & q_1^{\texttt{UCQ}}(x, y) \leftarrow & (A(x) \wedge R(x, y)) \\
& & \vee (A(x) \wedge R'(x, y)) \wedge \\
& q_2^{\texttt{UCQ}}(y) \leftarrow & R'(z, y)
\end{array}
$$

$q^{\texttt{JUCQ}}(x)$ is not *a reformulation of q w.r.t. $\mathcal{T}$: $ans(q^{\texttt{JUCQ}}, \langle \emptyset, \mathcal{A}\rangle) = \emptyset \neq \{a\}$*

The example above shows that the chosen cover C_1 prevents the unification of $R'(x, y)$ and $R'(z, y)$ performed by the plain CQ-to-UCQ reformulation, as these two atoms are split across two subqueries that are reformulated separately, while this unification is required for the correctness (actually the completeness) of the technique: it is the only way to trigger the use of the constraint $B \sqsubseteq \exists R'$.

More generally, given an input CQ and a TBox, each unification necessary for the correctness of the CQ-to-UCQ technique defines a set of input CQ atoms that cannot be separated within the cover: those which beget this unification.

We therefore provide a sufficient condition for a cover to be *safe for query answering*, i.e., to lead to a cover-based FOL reformulation. The main idea for this condition is to have a cautious approximation of the *query atoms* which are interdependent w.r.t. reformulation, i.e., which (directly or after specialization) unify through the CQ-to-UCQ reformulation technique, and keep them in the same cover fragment.

Only atoms with the same predicate may unify. Thus, we identify for each query atom *predicate* (i.e., concept or role name), the set of all TBox predicates in which the atom may turn through some sequence of atom specializations, i.e., backward chaining and/or unifications, the two operations applied by the technique of [10]. We say the query atom predicate *depends* on such TBox predicates, and we capture them in a conservative fashion, based on the syntax of the TBox constraints, in the following recursive definition.

Definition 5 (Concept and role dependencies w.r.t. a TBox)**.** *Given a TBox $\mathcal{T}$, a concept or role name N* depends w.r.t. $\mathcal{T}$ on *the set of concept and role names denoted $dep(N)$ and defined as the fixpoint of:*

$$
\begin{array}{l}
dep^0(N) = \{N\} \\
dep^n(N) = dep^{n-1}(N) \\
\qquad \cup \{cr(Y) \mid Y \sqsubseteq X \in \mathcal{T} \text{ and } cr(X) \in dep^{n-1}(N)\}
\end{array}
$$

where the cr function returns the concept or role name involved in any DL-Lite$_\mathcal{R}$ concept or role provided as input.

Example 5 (Dependencies (cont.))**.**

$$
\begin{array}{rl}
dep(A) = & \{A\} \\
dep(B) = & \{B\} \\
dep(R) = & \{R, R', B\} \\
dep(R') = & \{R', B\}
\end{array}
$$

Definition 6 (Safe cover for query answering)**.** *A cover C of q is* safe for query answering w.r.t. $\mathcal{T}$ *(or* safe *in short) iff it is a partition of q's atoms such that two atoms whose predicates depend on a common concept or role name w.r.t. $\mathcal{T}$ are in a same fragment.*

Note that while Definition 6 requires covers to be partitions, we will relax this restriction in Section 5.2.

Example 6 (Covers for JUCQ reformulation (cont.))**.** *The* FOL *query obtained with the safe cover $C_2 = \{\{A(x)\}, \{R(x,y), R'(z,y)\}\}$ is the* JUCQ reformulation*:*

$$q^{\text{JUCQ}}(x) \leftarrow \begin{array}{ll} q_1(x) \leftarrow & A(x) \\ q_2(x) \leftarrow & (R(x,y) \wedge R'(z,y)) \\ & \vee\ (R'(x,y) \wedge R'(z,y)) \\ & \vee\ (R'(x,y)) \vee (B(x)) \end{array} \wedge$$

Observe that $ans(q^{\text{JUCQ}}, \langle \emptyset, \mathcal{A} \rangle) = \{a\} = ans(q, \langle \mathcal{T}, \mathcal{A} \rangle)$.

Theorem 1 (Cover-based query answering)**.** *Applying Definition 4 on a safe cover C of q w.r.t. $\mathcal{T}$, and any* CQ*-to-*UCQ *(resp.* CQ*-to-*USCQ*) reformulation technique, yields a cover-based* JUCQ *(resp.* JUSCQ*) reformulation q^{FOL} of q w.r.t. $\mathcal{T}$.*

Proof. The proof follows from that of correctness of the CQ-to-UCQ reformulation technique in [10] for query answering. It directly *extends* to the use of any CQ-to-UCQ or CQ-to-USCQ reformulation technique for DL-Lite$_\mathcal{R}$, as, for any CQ and TBox, the FOL queries they compute are equivalent to that of [10].
Soundness: for any $\mathcal{T}$-consistent Abox $\mathcal{A}$, $ans(q^{\text{FOL}}, \langle \emptyset, \mathcal{A} \rangle) \subseteq ans(q, \langle \mathcal{T}, \mathcal{A} \rangle)$ holds.

Let t be a tuple in $ans(q^{\text{FOL}}, \langle \emptyset, \mathcal{A} \rangle)$. From Definition 4, q^{FOL} is $q^{\text{FOL}}(\bar{x}) \leftarrow \bigwedge_{i=1}^{m} q^{\text{FOL}}_{|f_i}$, thus t results from $t_i \in ans(q^{\text{FOL}}_{|f_i}, \langle \emptyset, \mathcal{A} \rangle)$, for $1 \leq i \leq m$. Therefore, for $1 \leq i \leq m$, $t_i \in ans(q_{|f_i}, \langle \mathcal{T}, \mathcal{A} \rangle)$ holds, because of the soundness of the CQ-to-UCQ reformulation technique. Hence, from Definition 4, $t \in ans(q, \langle \mathcal{T}, \mathcal{A} \rangle)$ holds.
Completeness: for any $\mathcal{T}$-consistent Abox $\mathcal{A}$, $ans(q, \langle \mathcal{T}, \mathcal{A} \rangle) \subseteq ans(q^{\text{FOL}}, \langle \emptyset, \mathcal{A} \rangle)$ holds.

Let t be a tuple in $ans(q, \langle \mathcal{T}, \mathcal{A} \rangle)$. Let q^{UCQ} be its reformulation using the CQ-to-UCQ technique. From the completeness of this technique, $t \in ans(q^{\text{UCQ}}, \langle \emptyset, \mathcal{A} \rangle)$ holds. Let q^{UCQ} be $\bigvee_{l=1}^{\alpha} cq_l$, then necessarily for some l: $t \in ans(cq_l, \langle \emptyset, \mathcal{A} \rangle)$ holds [10].

Let q^{FOL} be $\bigwedge_{i=1}^{m} q^{\text{FOL}}_{|f_i} = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{\beta_i} cq_{i,j}$. Since Definition 6 makes the reformulation of each fragment independent from another w.r.t. the CQ-to-UCQ technique, for any cq_l in q_{UCQ}: $cq_l = \bigwedge_{i=1}^{m} cq_{i,k \in [1,\beta_i]}$ holds. Hence, $t \in ans(q^{\text{FOL}}, \langle \emptyset, \mathcal{A} \rangle)$ holds. □

Remark that the trivial one-fragment cover (comprising all query atoms) is *always safe*; in this case, our CQ-to-JUCQ (resp. CQ-to-JUSCQ) reformulation technique reduces to the CQ-to-UCQ (resp. CQ-to-USCQ) one.

5. COVER-BASED QUERY OPTIMIZATION IN DL-Lite$_\mathcal{R}$

We study now the query answering optimization problem of Section 3.3 for DL-Lite$_\mathcal{R}$. We analyze a first optimization space in Section 5.1, before extending our discussion to a larger space in Section 5.2. Finally, we provide search algorithms in Section 5.3.

5.1 Safe cover optimization space

We show next that the set of all safe covers for a query q form a *lattice* $\mathcal{L}_q$ whose precedence relationship is denoted $\prec$, where $C_1 \prec C_2$ if each fragment of C_2 is a union of some fragments of C_1. The lattice has as lower bound (minimum) the single-fragment cover, and as upper bound (supremum) a specific cover C_{root}, termed the *root* cover, and defined below:

Definition 7 (Root cover)**.** *We term* root cover *for a query q and TBox $\mathcal{T}$ the cover C_{root} obtained as follows. Start with a cover C_1 where each atom is alone in a fragment. Then, for any pair of fragments $f_1, f_2 \in C_1$ and atoms $a_1 \in f_1, a_2 \in f_2$ such that there exists a predicate on which those of a_1 and a_2 depend w.r.t. $\mathcal{T}$, create a fragment $f' = f_1 \cup f_2$ and a new cover $C_2 = (C_1 \setminus \{f_1, f_2\}) \cup \{f'\}$. Repeat the above until the cover is stationary; this is the root cover, denoted C_{root}.*

It is easy to see that C_{root} is safe, and does not depend on the order in which the fragments are considered.

Example 7 (Root cover (cont.))**.** *Recall the query and TBox from Example 4. The root cover C_{root} in this case is the cover C_2 from Example 6.*

Proposition 1 states that C_{root} has the maximal number of smallest fragments. Its proof is based on the following lemma:

Lemma 1 (C_{root} fragment structure)**.** *A fragment f in the root cover C_{root} is of one of the following two forms:*

1. *a singleton, i.e., $f = \{a_i\}$ for some query atom a_i;*
2. *$f = \{a_{i_1}, \ldots, a_{i_n}\}$, for $n \geq 2$, and for every atom $a_{i_1} \in f$, there exists one atom $a_{i_2} \in f$, and a predicate b_j in the TBox, such that both predicates of a_{i_1} and of a_{i_2} depend on b_j.*

Proof. The lemma follows directly from the definition of C_{root}. Those atoms that do not share a dependency with any other atom appear in singleton fragments (case 1 above, as they never get fused). Atoms which share some dependencies (i.e., atoms whose predicates depend on one another) get fused in fragments of the form 2 above. □

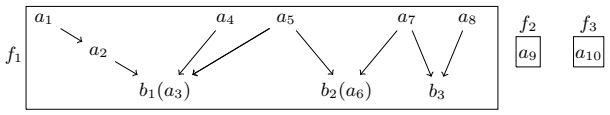

Figure 2: Sample C_{root} cover.

Figure 2 depicts a possible C_{root} cover of a 10-atoms query; the cover has 3 fragments, each shown in a rectangle. In the figure, every a_i denotes a query atom; an arrow from an atom a_i to the atom a_j denotes that the predicate of a_i depends on the predicate of a_j. Further, b_1, b_2 and b_3 are predicates: b_1 et b_2 appear in the query (as the predicates of a_2 and a_6, respectively) while b_3 does not appear in the query. In this example, a_1, a_2, a_4 and a_5 depend on b_1, which is also the predicate used in the query atom a_3. Similarly, $b_2(a_6)$ is a TBox predicate on which a_5 and a_7 depend, and which appears in atom a_6. The largest fragment corresponds to the second fragment shape from the above Lemma, while the two one-atom fragments correspond to the first shape.

Proposition 1 (Minimality of C_{root} fragments)**.** *Let C_{root} be the root cover for q and $\mathcal{T}$, and C be another safe cover. For any fragment $f \in C_{\text{root}}$, and atoms $a_i, a_j \in f$, there exists a fragment $f' \in C$ such that $a_i, a_j \in f'$, in other words: any pair of atoms together in C_{root} are also together in C.*

Proof. For ease of explanation, in the proof, we rely on the graphical directed graph representation used in Figure 2 for dependencies between the predicates appearing in the atoms of a cover and/or other predicates from the KB.

Because f holds at least a_i and a_j, it must be a fragment of form 2, as stated in Lemma 1. It follows, thus, that in f there exists

what we call an *extended path* e, going from a_i to a_j following the dependency edges either from source to target, or in the opposite direction; in other words, e alternately moves "up (or down) then down (or up)" a certain number of times in the fragment.

If e only contains edges in the same direction (either all are $\rightarrow$ or all are $\leftarrow$), it follows immediately that a_i and a_j are in the same fragment of C.

In the contrary case, there must exist some predicates in the TBox $b_1, \ldots, b_m$, $m \geq 1$, and some f atoms $a_1, \ldots, a_{m-1}$ defining an extended path e from a_i to a_j in f, as follows:

1. $a_i \rightarrow \ldots \rightarrow b_1$ (a $\rightarrow$ path segment), or b_1 is the predicate used in a_i;
2. b_k ($1 \leq k < m-1$), is the predicate used in a_l ($k \leq l < m-1$), or $b_k \leftarrow \ldots \leftarrow a_l$ (a $\leftarrow$ path segment);
3. $b_k \leftarrow \ldots \leftarrow a_l$ (a $\leftarrow$ path segment), with $1 \leq k < m-1$ and $k \leq l < m-1$, and $a_l \rightarrow \ldots \rightarrow b_{k+1}$. :
4. b_{m-1} is the predicate used in a_{m-1} or $b_{m-1} \leftarrow \ldots \leftarrow a_{m-1}$, then $a_{m-1} \rightarrow \ldots \rightarrow b_m$;
5. b_m is the predicate used in a_j or $b_m \leftarrow \ldots \leftarrow a_j$

Observe that items (2) and (3) can repeat (alternately) until b_{m-1} is reached.

Since C is safe, a_i and a_1 must appear in the same fragment in C (and only in that fragment), because they both depend on b_1.

For $1 \leq i \leq m-2$, a_i must appear in the same fragment as a_{i+1} (and only there), given that they both depend on b_i.

Since C is safe, a_j and a_{m-1} must appear in the same fragment of C (and only there).

From the above follows that $\{a_i, a_1, \ldots, a_{m-1}, a_j\}$ are all in the same fragment of C, which contradicts our hypothesis. □

Finally, from the above result, we derive easily that the space of safe covers of a query q is a lattice, denoted $\mathcal{L}_q$, with C_{root} as supremum and the single-fragment cover as infimum:

Theorem 2 (Safe cover space)**.** *Let C be a safe cover and f one of its fragment. Then, f is the union of some fragments from C_{root}.*

Proof. Suppose that f is not a union of some fragments from C_{root}, and let us show a contradiction. In this case, f necessarily contains a strict, non-empty subset of a fragment of C_{root}. It follows that there are two atoms whose predicates depend on a common concept or role name w.r.t. $\mathcal{T}$ (as they were together in the fragment of C_{root}) that are not in a same fragment of C. Therefore C is not a safe cover, a contradiction. □

5.2 Optimization Beyond Safe Covers

A dependency-rich TBox leads to few, large fragments in C_{root}, and thus to a relatively small lattice. In this section, we identify a set of covers which are not safe (in particular, they are not partitions), yet from which we still derive FOL reformulations of the query; this enlarges our space of alternatives and thus gives more optimization opportunities.

An *extended fragment* $f \| g$ of q is a pair of atom sets such that $f = \{a_1, \ldots, a_k\}$ is a set of atoms from q, and $g \subseteq f$. When $f = g$, extended fragments coincide with simple fragments, thus we generalize query covers (Definition 2) to consider that they only consist of extended fragments, while preserving that no fragment must be contained in another.

To an extended fragment we associate:

Definition 8 (Extended fragment query of a CQ)**.** *Let $f \| g \in C$ be an extended fragment of q. The* extended fragment query $q_{|f\|g}$ *of q w.r.t. C is the subquery whose body consists of the atoms in f, and whose free variables are the free variables of q appearing in the atoms of g, plus the variables appearing in an atom of g that are shared with some atom in g', for some fragment $f' \| g'$ of C.*

In an extended fragment query, atoms from $f \setminus g$ only *reduce (filter)* the answers, without adding variables to the head.

Given a cover of extended fragments, the *extended cover-based reformulation of a query q* is the FOL query:

$$q^e(\bar{x}) \leftarrow \bigwedge_{i=1}^{m} q^{\texttt{FOL}}_{|f_i\|g_i}$$

Example 8 (Extended cover-based reformulation)**.** *Recall the query and KB from Example 4. Let $f_0 = \{A(x)\}$ and $f_1 = \{R(x,y), R'(z,y)\}$ be the two fragments of the root cover C_{root}. Consider also the cover $C_3 = \{f_1 \| f_1, f_2 \| f_0\}$, with $f_2 = \{A(x), R(x,y)\}$.*

The extended fragment $f_1 \| f_1$ coincide with the simple fragment f_1, therefore the extended fragment query $q_{|f_1\|f_1}$ of q w.r.t. C_3 is the subquery $q_{|f_1\|f_1}(x) \leftarrow R(x,y) \wedge R'(z,y)$. Observe that y is not a free variable of $q_{|f_1\|f_1}$, as it is not a free variable of q, and moreover, it does it appear in f_0, whereas $f_2 \| f_0$ is the only other fragment in the cover C_3.

The extended fragment query $q_{|f_2\|f_0}$ of q w.r.t. C_3 is the subquery $q_{|f_2\|f_0}(x) \leftarrow A(x) \wedge R(x,y)$. Again, note that y is not a free variable of f_0, and therefore its not a free variable of $q_{|f_2\|f_0}$.

Then, the extended cover-based reformulation corresponding to C_3 is the FOL *query:*

$$q^e(x) \leftarrow q^{\texttt{FOL}}_{|f_1\|f_1}(x) \wedge q^{\texttt{FOL}}_{|f_2\|f_0}(x)$$

where:

$$q^{\texttt{FOL}}_{|f_1\|f_1}(x) \leftarrow \quad ((R(x,y) \wedge R'(z,y)) \vee R'(x,y) \vee B(x))$$

$$q^{\texttt{FOL}}_{|f_2\|f_0}(x) \leftarrow \quad ((A(x) \wedge R(x,y)) \vee (A(x) \wedge R'(x,y)) \vee (A(x) \wedge B(x)))$$

Observe that applying the TBox constraint $R' \sqsubseteq R$ to $q_{|f_1\|f_1}(x)$ leads to:

$$((R(x,y) \wedge R'(z,y)) \vee (R'(x,y) \wedge R'(x,y))) \equiv ((R(x,y) \wedge R'(z,y)) \vee R'(x,y))$$

Then, applying the TBox constraint $B \sqsubseteq \exists R'$ we obtain the reformulation of $q_{|f_1\|f_1}$ w.r.t. TBox $\mathcal{T}$, i.e., $q^{\texttt{FOL}}_{|f_1\|f_1}(x)$.

Similarly, applying to $q_{|f_2\|f_0}$ the constraint $R' \sqsubseteq R$ and subsequently $B \sqsubseteq \exists R'$ leads to $q^{\texttt{FOL}}_{|f_2\|f_0}(x)$.

Note that $ans(q^e, \langle \emptyset, \mathcal{A} \rangle) = \{a\} = ans(q, \langle \mathcal{T}, \mathcal{A} \rangle)$.

The introduction of extra atoms in extended fragments is reminiscent of the the classical *semijoin reducer* [5] optimization technique, which we briefly recall. Consider a CQ $q(x) \leftarrow R(x,y) \wedge S(y,z)$, to which corresponds the algebra expression $R(x,y) \bowtie_y S(y,z)$. The semijoin reducer technique consists, instead, of computing the result of q by the equivalent expression:

$$(R(x,y) \ltimes_y \pi_y(S(y,z))) \bowtie_y S(y,z)$$

where $\pi_y(S(y,z))$ is the first projection of S, and $\ltimes_y$ denotes the *semijoin* relational operator, returning every tuple from the left-hand side input that joins with the right-hand input. The semijoin filters ("reduces") the R relation to only those tuples having a match in S. If there are few distinct values of y in S, $\pi_y(S(y,z))$ is small, thus it can be loaded in memory and the $\ltimes_y$ operator can be evaluated very efficiently. Further, if only few R tuples survive

the $\ltimes_y$, the cost of the $\bowtie_y$ operator is reduced, given that there are less tuple comparisons to be made.

While the benefits of semijoins are well-known, there are many ways to introduce them in a given query, further increasing the space of alternative plans to be considered by an optimizer. While some heuristics have been proposed to explore only some carefully chosen semijoin plans [27], we noted that RDBMS optimizers do not explore semijoin options, in particular for the very large queries resulting from the FOL reformulations of CQs. *Extended fragments mitigate this problem by intelligently using semijoin reducers to fasten the evaluation of the* FOL *reformulation by the RDBMS.*

We now define an *extended search space* $\mathcal{E}_q$ of covers for a given query q, based on the safe cover lattice $\mathcal{L}_q$.

1. First, any $\mathcal{L}_q$ cover is also in $\mathcal{E}_q$.
2. Second, from any cover $C \in \mathcal{E}_q$ and fragment $f||g \in C$, a cover C' defined as follows is in $\mathcal{E}_q$:
   (a) pick a query atom $a \notin f$, which shares a variable with an atom in f;
   (b) let the new extended fragment $f'||g$ be $f \cup \{a\}||g$; let C' be $(C \setminus \{f||g\}) \cup \{f'||g\}$.

   Since the newly added atom was already in a fragment of C_{root}, C' is not a partition.

In the above, $\mathcal{E}_q$ covers are "grown" starting from the $\mathcal{L}_q$ ones. To any fragment f of a cover $C \in \mathcal{L}_q$, one can add an atom as described above, leading to an extended fragment $f'||f$; replacing f with f' in C leads to C'. Note that any subsequent addition of an atom to $f' \in C'$ is not allowed to change the head of the extended fragment query thus obtained, i.e., the latter will be of the form $q_{f''||f}$ etc.

The core result allowing us to benefit of the performance savings of extended covers in order to efficiently answer queries is:

Theorem 3 ($\mathcal{E}_q$ cover-based query answering)**.** *Applying Definition 4 together with Definition 8, on a* CQ *q, a TBox $\mathcal{T}$, a cover C from $\mathcal{E}_q$, and any* CQ*-to-*UCQ *(resp.* CQ*-to-USCQ) reformulation technique, yields a* JUCQ *(resp.* JUSCQ*) reformulation of q w.r.t. $\mathcal{T}$.*

Proof. The proof follows from that of Theorem 1. It relies on the fact that, given a safe cover $C = \{g_1, \ldots, g_m\}$ of q and an extended cover $C' = \{f_1||g_1, \ldots, f_m||g_m\}$ of q, the queries $q(\bar{x}) \leftarrow \bigwedge_{i=1}^{m} q_{|g_i}$ and $q'(\bar{x}) \leftarrow \bigwedge_{i=1}^{m} q_{|f_i||g_i}$ are equivalent, though each $q_{|g_i}$ subsumes $q_{|f_i||g_i}$. Indeed, q' is obtained from q by duplicating atoms already present in q, thus q^e only adds redundancy w.r.t. q, hence remains equivalent to it. □

5.3 Cost-based cover search algorithms

Our first algorithm, **EC-DL (Exhaustive Covers)**, starts from C_{root} and builds all $\mathcal{L}_q$ covers by fusing fragments, and all $\mathcal{E}_q$ covers by adding atoms, as explained previously. The second one, **GC-DL (Greedy Covers)**, explores $\mathcal{E}_q$ partially, in greedy fashion. It uses an *explored cover set* initialized with $\{C_{\text{root}}\}$. GC-DL picks from this set a cover C inducing a q^{FOL} reformulation with minimum evaluation cost, and attempts to build from it a cover C', through fragment fusion or atom addition. GC-DL only adds C' to the explored set if $\epsilon(C') < \epsilon(C)$, thus it only explores a small part of the search space. Both algorithms return a cover-based reformulation with the minimum estimated evaluation cost w.r.t. the explored space.

When fusing two fragments, ϵ decreases if the resulting fragment is more selective than the two fragments it replaces. Therefore, the RDBMS may find a more efficient way to evaluate the query of this fragment, and/or its result may be smaller, making the evaluation of q^{FOL} based on the new cover C' faster. When adding an atom to an extended fragment, ϵ decreases if the conditions are met for the semijoin reducer to be effective (Section 5.2). Such performance benefits can be very significant, as we show below.

6. EXPERIMENTAL EVALUATION

We implemented our reformulation-based query answering cover-based approach in Java 7, on top of PostgreSQL v9.3.2, running on an 8-core Intel Xeon (E5506) 2.13 GHz machine with 16GB RAM, using Mandriva Linux r2010.0. We used the LUBM$^{\exists}_{20}$ DL-Lite$_{\mathcal{R}}$ TBox and associated EUDG data generator [19]: LUBM$^{\exists}_{20}$ consists of 34 roles, 128 concepts and 212 constraints; the generated ABox comprises 15 million facts.

We store the ABox in a binary table for each role and a unary one for each concept; as customary in efficient Semantic Web data management systems, e.g., [21], each identifier or literal occurring in a fact is encoded into an integer prior to storing them in the RDBMS. Also in keeping with the best practices in efficient Semantic Web stores [31, 21], each concept table is indexed, and similarly each role table is indexed by each of its two attributes. The TBox and predicates dependencies are stored in memory. For our experiments, we relied on a CQ-to-UCQ reformulation tool, namely RAPID [11] (recall that any CQ-to-USCQ reformulation technique could have been used instead). For the cost estimation function ϵ, we relied on Postgres' own estimation, obtained using the `explain` directive[3].

We devised a set of 13 CQs (cf. Appendix) against this KB; the queries have between 2 and 10 atoms, with an average of 5.77 atoms. Their UCQ reformulations have 35 to 667 CQs, 290.2 on average. We depict below Q_9, one of the most complex, whose UCQ reformulation is a union of 368 CQs:

$Q_9(n, e) \leftarrow$ Student(x) $\wedge$ TakesCourse(x, c) $\wedge$ Advisor(x,a) $\wedge$
MemberOf(x,"http://dep0.univ0.edu") $\wedge$ Phone(x, "xxx") $\wedge$
MemberOf(a, "http://dep0.univ0.edu") $\wedge$ TeacherOf(p, c) $\wedge$
Email(p, e) $\wedge$ ResearchInterest(a, "res7") $\wedge$ Name(c, n)

where quoted strings stand for constants produced by the data generator (shortened for the presentation).

Figure 3 depicts the evaluation time through Postgres, for our 13 queries, of four FOL reformulations: (i) the UCQ produced by RAPID [11]; (ii) the JUCQ reformulation based on C_{root}; (iii) the JUCQ reformulation corresponding to the best-performing cover found by our algorithm EC-DL, and (iv) the JUCQ reformulation based on the best-performing cover found by GC-DL. First, the figure shows that fixed FOL reformulations are not efficiently evaluated, e.g., the UCQs for Q_1, Q_5 and Q_9-Q_{11}, and the JUCQs based on C_{root} for Q_6-Q_8 and Q_{13}. This poor performance correlates with the large size of the reformulations: they feature very large *unions* of CQs, very poorly handled by current RDBMS optimizers, which are designed and tuned for small CQs. This finding is confirmed in [8] for two other leading RDBMSs, thus it is not specific to the Postgres system used here. Second, the reformulation based on the cover returned by EC-DL is always more efficient than UCQ reformulation (more than one order of magnitude for Q_5), respectively, C_{root}-based JUCQ reformulation (up to a factor of 230 for Q_6). Third, in our experiments, the GC-DL-chosen cover leads to a JUCQ reformulation as efficient as the EC-DL one, demonstrating that even a partial, greedy cover search leads to good performance. (In general, though, the optimality of GC-DL is not guaranteed.)

[3] See http://www.postgresql.org/docs/9.1/static/sql-explain.html.

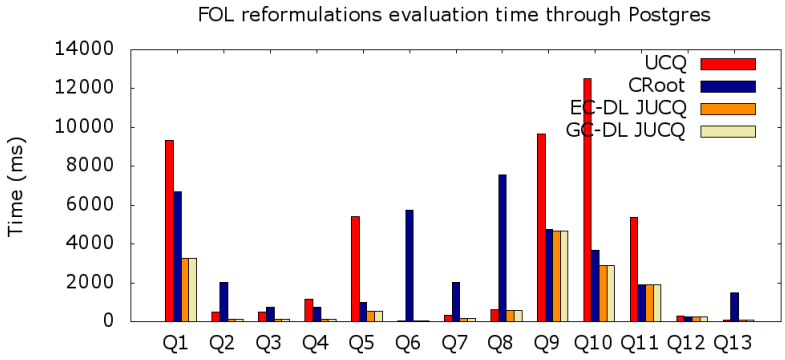

Figure 3: Evaluation time for JUCQ reformulations.

For Q_9, C_{root} is $\{\{a_5\}, \{a_8\}, \{a_9\}, \{a_{10}\}, \{a_1, a_2, a_3, a_4, a_6, a_7\}\}$ (where the atoms are ordered as shown above), while the cover chosen by EC-DL and GC-DL fuses the largest fragment of C_{root} with $\{a_{10}\}$. For Q_7 and Q_9-Q_{13}, the best cover we found is safe; for all the others, this is not the case, confirming the interest of extended fragments within covers.

Figure 4 depicts the running time of the EC-DL and GC-DL algorithms, which can be seen as the overhead of our cover-based technique. The time is very small, between 2 ms (Q_{11}-Q_{13}, with just 2 atoms) and 221 ms (EC-DL on Q_{10}, of 10 atoms). The time is higher for more complex queries, but these are precisely the cases where our techniques are most beneficial, e.g., for Q_{10}, EC-DL runs in less than 2% of the time to evaluate the UCQ reformulation, while the JUCQ we recommend is more than 4 times faster than the UCQ. As expected, GC-DL is faster than EC-DL due to the exploration of less covers. Together, Figure 3 and 4 confirm the benefits and practical interest of our cost-based cover search.

Discussion: implementation alternatives. Recall that all the above experiments were made with a simple data layout. Adopting another layout (as in [6, 8]) is orthogonal to our technique, though obviously the above-mentioned times would change. Indeed, our *cost-based* approach will get the best performance of any layout.

Finally, the choice of the cost model used in our approach deserves a discussion. We have used Postgres' own cost model; one could alternatively rely on an externally implemented cost estimation, as we did in our previous work [8]. There, we have shown that a relatively straightforward cost model estimator based on textbook material leads to estimations quite similar to those of Postgres, even though, clearly, Postgres' internal optimization strategy is much more elaborate than the simple assumptions our cost model made. This may be due to the fact that the biggest impact of the cost model is to allow the optimizer to avoid very bad plans – this is far more important than to pick the best one in a set of similar-performance efficient plans. Our findings in [8] (where, furthermore, we calibrated our own cost model to suit three different RDBMSs) make us confident that the benefits of our cost-based approach can carry over when other RDBMSs are used, too.

7. RELATED WORK AND CONCLUSION

We proposed a novel framework for any OBDA setting enjoying FOL reducibility of query answering, for which we studied a *space of alternative* FOL *reformulations to evaluate through an RDBMS*. This space is crucial, as it allows choosing a FOL reformulation most efficiently evaluated by the RDBMS. We applied this framework to the DL-Lite$_\mathcal{R}$ description logic underpinning the W3C's OWL2 QL standard for the Semantic Web, and experimentally demonstrated its performance benefits.

Our approach departs from the literature focused on a *single* FOL query reformulation, where optimization mainly reduces to *producing fast a* UCQ *reformulation as minimized as possible*: [24, 11, 12, 29, 17, 30, 14, 23, 15] consider DL-Lite$_\mathcal{R}$, existential rules and Datalog±. [28] studies CQ-to-USCQ reformulation for existential rules encompassing DL-Lite$_\mathcal{R}$; USCQ reformulations are shown to perform overall better than UCQ ones in an RDBMS. We build on these works to devise CQ-to-JUCQ and CQ-to-JUSCQ reformulation techniques, further used for cost-based query answering optimization. Our semijoin-inspired technique goes *against* minimization, as it *adds* redundant atoms, but does so with the help of a cost model only when this improves performance. [26] proposes a CQ-to-Datalog reformulation technique; a produced *non-recursive* Datalog program corresponds to a JUCQ, which is not chosen according to a cost-model. Finally, [8] is an instance of our general framework for the strictly less expressive case of RDF databases, where any cover leads to a reformulation. In general, and in particular for DL-Lite$_\mathcal{R}$ as we have shown, this is not the case, therefore the core contribution of the present work is an analysis of safe covers, guaranteed to lead to reformulations, and of a larger carefully chosen space of covers which do not satisfy the safety criterium, yet still lead to equivalent FOL reformulations of the incoming query.

We plan to extend our framework to efficient query answering *using views*, i.e., through a set of predefined CQs, which is typical of the *database query optimization* and *information integration* settings [16].

Acknowledgements This work has been partially funded by the PIA Datalyse project and the ANR PAGODA project.

8. REFERENCES

[1] N. Abdallah, F. Goasdoué, and M. Rousset. DL-LITER in the light of propositional logic for decentralized data management. In *IJCAI*, 2009.

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

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

[4] J. Baget, M. Leclère, M. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9-10), 2011.

[5] P. A. Bernstein and D. W. Chiu. Using semi-joins to solve relational queries. *J. ACM*, 28(1), 1981.

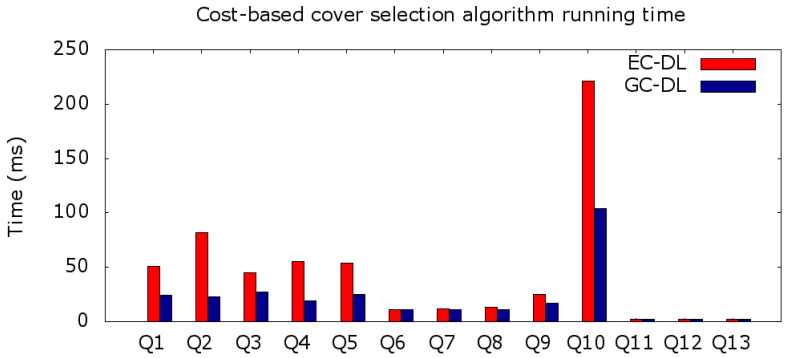

Figure 4: Running time of the cover-based technique.

[6] M. A. Bornea, J. Dolby, A. Kementsietsidis, K. Srinivas, P. Dantressangle, O. Udrea, and B. Bhattacharjee. Building an efficient RDF store over a relational database. In *SIGMOD*, 2013.

[7] D. Bursztyn, F. Goasdoué, and I. Manolescu. Efficient query answering in DL-Lite through FOL reformulation. In *DL*, 2015.

[8] D. Bursztyn, F. Goasdoué, and I. Manolescu. Optimizing reformulation-based query answering in RDF. In *EDBT*, 2015.

[9] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *PODS*, 2009.

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

[11] A. Chortaras, D. Trivela, and G. B. Stamou. Optimized query rewriting for OWL 2 QL. In *CADE*, 2011.

[12] G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. MASTRO: A reasoner for effective ontology-based data access. In *International Workshop on OWL Reasoner Evaluation (ORE-2012)*, 2012.

[13] F. Goasdoué, I. Manolescu, and A. Roatiş. Efficient query answering against dynamic RDF databases. In *EDBT*, 2013.

[14] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *ICDE*, 2011.

[15] G. Gottlob, G. Orsi, and A. Pieris. Query rewriting and optimization for ontological databases. *ACM Trans. Database Syst.*, 39(3):25, 2014.

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

[17] M. König, M. Leclère, M. Mugnier, and M. Thomazo. A sound and complete backward chaining algorithm for existential rules. In *RR*, 2012.

[18] M. Lenzerini. Ontology-based data management. In *CIKM*, 2011.

[19] C. Lutz, I. Seylan, D. Toman, and F. Wolter. The combined approach to OBDA: taming role hierarchies using filters. In *ISWC*, 2013.

[20] M. Mugnier. Ontology-based query answering with existential rules. In *RuleML*, 2012.

[21] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *VLDBJ*, 19(1):91–113, 2010.

[22] K. Ono and G. M. Lohman. Measuring the complexity of join enumeration in query optimization. In *VLDB*, 1990.

[23] G. Orsi and A. Pieris. Optimizing query answering under ontological constraints. *PVLDB*, 4(11), 2011.

[24] H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient query answering for OWL 2. In *ISWC*, 2009.

[25] R. Ramakrishnan and J. Gehrke. *Database management systems*. McGraw-Hill, 2003.

[26] R. Rosati and A. Almatelli. Improving query answering over DL-Lite ontologies. In *KR*, 2010.

[27] K. Stocker, D. Kossmann, R. Braumandl, and A. Kemper. Integrating semi-join-reducers into state of the art query processors. In *ICDE*, 2001.

[28] M. Thomazo. Compact rewriting for existential rules. *IJCAI*, 2013.

[29] T. Venetis, G. Stoilos, and G. B. Stamou. Incremental query rewriting for OWL 2 QL. In *Description Logics*, 2012.

[30] R. D. Virgilio, G. Orsi, L. Tanca, and R. Torlone. NYAYA: A system supporting the uniform management of large sets of semantic data. In *ICDE*, 2012.

[31] C. Weiss, P. Karras, and A. Bernstein. Hexastore: Sextuple indexing for Semantic Web data management. *PVLDB*, 2008.

APPENDIX

$q_1(u,i,n,e,t) \leftarrow ub{:}Professor(x) \wedge ub{:}degreeFrom(x,u) \wedge ub{:}researchInterest(x,i)$
$\wedge\, ub{:}name(x,n) \wedge ub{:}emailAddress(x,e) \wedge ub{:}telephone(x,t)$

$q_2(x,e,t) \leftarrow ub{:}Professor(x) \wedge ub{:}degreeFrom(x, \text{“}http://www.University870.edu'')$
$\wedge\, ub{:}researchInterest(x, \text{“}Research21'') \wedge ub{:}name(x, \text{“}AssociateProfessor2'')$
$\wedge\, ub{:}emailAddress(x,e) \wedge ub{:}telephone(x,t)$

$q_3(x) \leftarrow ub{:}Professor(x) \wedge ub{:}degreeFrom(x, \text{“}http://www.University870.edu'')$
$\wedge\, ub{:}researchInterest(x, \text{“}Research21'') \wedge ub{:}name(x, \text{“}AssociateProfessor2'')$
$\wedge\, ub{:}emailAddress(x, \text{“}AssociateProfessor2@Department1.University0.edu'')$
$\wedge\, ub{:}telephone(x, \text{“}xxx-xxx-xxxx'')$

$q_4(x,y) \leftarrow ub{:}Professor(x) \wedge ub{:}teacherOf(x,y)$
$\wedge\, ub{:}degreeFrom(x, \text{“}http://www.University870.edu'') \wedge ub{:}researchInterest(x, \text{“}Research21'')$
$\wedge\, ub{:}name(x, \text{“}AssociateProfessor2'') \wedge ub{:}telephone(x, \text{“}xxx-xxx-xxxx'')$
$\wedge\, ub{:}emailAddress(x, \text{“}AssociateProfessor2@Department1.University0.edu'')$

$q_5(x,y,z) \leftarrow ub{:}Professor(x) \wedge ub{:}teacherOf(x,y)$
$\wedge\, ub{:}worksFor(x,z) \wedge ub{:}degreeFrom(x, \text{“}http://www.University870.edu'')$
$\wedge\, ub{:}researchInterest(x, \text{“}Research21'') \wedge ub{:}name(x, \text{“}AssociateProfessor2'')$
$\wedge\, ub{:}emailAddress(x, \text{“}AssociateProfessor2@Department1.University0.edu'')$
$\wedge\, ub{:}telephone(x, \text{“}xxx-xxx-xxxx'')$

$q_6(x,n) \leftarrow ub{:}Faculty(x) \wedge ub{:}publicationAuthor(y,x) \wedge ub{:}researchInterest(x, \text{“}Research16'')$
$\wedge\, ub{:}name(y,n) \wedge ub{:}emailAddress(x, \text{“}AssociateProfessor0@Department0.University0.edu'')$

$q_7(n) \leftarrow ub{:}Professor(x) \wedge ub{:}teacherOf(x,c)$
$\wedge\, ub{:}memberOf(x, \text{“}http://www.Department0.University0.edu'') \wedge ub{:}name(x,n)$
$\wedge\, ub{:}emailAddress(x, \text{“}FullProfessor8@Department0.University0.edu'')$
$\wedge\, ub{:}telephone(x, \text{“}xxx-xxx-xxxx'')$

$q_8(x,n) \leftarrow ub{:}Faculty(x) \wedge ub{:}publicationAuthor(y,x) \wedge ub{:}researchInterest(x, \text{“}Research16'') \wedge ub{:}name(y,n)$

$q_9(n,e) \leftarrow ub{:}Student(x) \wedge ub{:}takesCourse(x,c) \wedge ub{:}advisor(x,a)$
$\wedge\, ub{:}memberOf(x, \text{“}http://www.Department0.University0.edu'') \wedge ub{:}telephone(x, \text{“}xxx-xxx-xxxx'')$
$\wedge\, ub{:}teacherOf(p,c) \wedge ub{:}emailAddress(p,e) \wedge ub{:}researchInterest(a, \text{“}Research7'')$
$\wedge\, ub{:}memberOf(a, \text{“}http://www.Department0.University0.edu'') \wedge ub{:}name(c,n)$

$q_{10}(n,e) \leftarrow ub{:}Professor(x) \wedge ub{:}takesCourse(x,c) \wedge ub{:}advisor(x,a)$
$\wedge\, ub{:}memberOf(x, \text{“}http://www.Department0.University0.edu'') \wedge ub{:}telephone(x, \text{“}xxx-xxx-xxxx'')$
$\wedge\, ub{:}teacherOf(p,c) \wedge ub{:}emailAddress(p,e) \wedge ub{:}researchInterest(a, \text{“}Research7'')$
$\wedge\, ub{:}memberOf(a, \text{“}http://www.Department0.University0.edu'') \wedge ub{:}name(c,n)$

$q_{11}(x) \leftarrow ub{:}Professor(x) \wedge ub{:}Student(x)$

$q_{12}(x) \leftarrow ub{:}Professor(x) \wedge ub{:}Department(x)$

$q_{13}(x) \leftarrow ub{:}Publication(x) \wedge ub{:}Department(x)$

Table 3: Queries used in our experiments.