Knowledge-Based Systems 56 (2014) 36–48

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Query rewriting under query refinements

Tassos Venetis, Giorgos Stoilos *, Giorgos Stamou

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

ARTICLE INFO

Article history:
Received 10 May 2013
Received in revised form 9 September 2013
Accepted 19 October 2013
Available online 5 November 2013

Keywords:
Ontologies
Description logics
Query rewriting
Query refinement
DL-Lite
EL

ABSTRACT

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

1. Introduction

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

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

An important approach for query answering in these languages is via a technique called *query rewriting* [8,2,32]. According to this technique a query $\mathcal{Q}$ and an ontology $\mathcal{O}$ are transformed into a program $\mathcal{R}$, typically a datalog program called a *rewriting*, such that the answers of $\mathcal{R}$ over any input data $\mathcal{D}$ (discarding the ontology) are precisely the answers of $\mathcal{Q}$ over $\mathcal{D}$ and $\mathcal{O}$. Such an approach to query answering is interesting from a practical perspective because after computing $\mathcal{Q}$ the problem of query answering can be delegated to efficient and scalable (deductive) database and datalog evaluation systems by either directly evaluating $\mathcal{R}$ using off-the-shelf systems [13], by implementing customised engines [27], or by integrating $\mathcal{R}$ in an optimal way into data saturation engines [38].

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

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

* Corresponding author. Tel.: +30 210 7722521.
E-mail addresses: avenet@image.ece.ntua.gr (T. Venetis), gstoil@image.ece.ntua.gr (G. Stoilos), gstam@cs.ntua.gr (G. Stamou).

0950-7051/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2013.10.020

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

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

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

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

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

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

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

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

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

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

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

The rest of the paper is organised as follows. In Section 2 we recapitulate all the necessary terminology as well as relevant definitions that will help us with the rest of the paper. Subsequently, in Sections 3–5, we study the aforementioned refinement problems. More precisely, in Section 3 we study query rewriting when some head variables have been removed; in Section 4 we study query rewriting when new variables have been added to the head of the current query; and finally, in Section 5 we study query rewriting when an atom from the query has been removed. For each studied problem we present motivating examples, detailed algorithms, and proofs. Subsequently, in Section 6 we study for each refinement problem the possiblity of some optimisations. Next, in Section 7 we present two implementations of all our algorithms, one using the system Rapid and one using Requiem. We also present results of a detailed experimental evaluation comparing them against the classical Rapid and Requiem implementation. Finally, in Section 8 we conclude the paper.

2. Preliminaries

2.1. Existential rules

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

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

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

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

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

2.2. Queries

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

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

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

Given CQs $\mathcal{Q}_1, \mathcal{Q}_2$ with distinguished variables $\vec{x}$ and $\vec{y}$, respectively, we say that $\mathcal{Q}_2$ *subsumes* $\mathcal{Q}_1$ (or that $\mathcal{Q}_2$ is a *subsumer* of $\mathcal{Q}_1$), if there exists a substitution σ from $\mathsf{var}(\mathcal{Q}_2)$ to $\mathsf{var}(\mathcal{Q}_1)$ such that every atom in $\mathcal{Q}_2\sigma$ also appears in $\mathcal{Q}_1$. For a rewriting $\mathcal{R}$ we say that $\mathcal{Q}_1$ is *redundant* in $\mathcal{R}$ if a different query $\mathcal{Q}_2 \in \mathcal{R}$ exists that subsumes it; otherwise it is called *non-redundant* in $\mathcal{R}$. Finally, $\mathcal{Q}_1$ is called *non-subsumer* if it does not subsume any other query in $\mathcal{R}$.

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