A Compilation Technique for Interactive Ontology-mediated Data Exploration*

Medina Andresel, Magdalena Ortiz de la Fuente, and Mantas Šimkus

TU Wien, Austria

Abstract We make a step towards the interactive exploration of data in the ontology-mediated setting, presenting a technique to construct an offline compilation that allows us to answer efficiently different related queries in online phase, without the need to access again the original data. We also propose algorithms to construct relevant variations of a given query that, for example, reduce or increase the number of answers, while modifying the query in a minimal way, or make explicit the common properties of all objects that are in the answers to a given query.

1 Introduction

Ontology based data-access (OBDA) [9], understood broadly as the use of ontologies to mediate access to data sources and to leverage domain knowledge when querying possibly incomplete data, has become a key application of Description Logics (DLs) and a central topic of research in the last decade. However, most work focuses on providing algorithms for answering queries, and the lack of support to users when they formulate queries or process their query answers has been recognized as an obstacle to the wider adaptation of OBDA [1].

A major gap of existing systems is that they only support answering queries *one-at-a-time*. Given multiple variations of a query, answers are computed from scratch for each one of them, accessing the data every time. Moreover, it is left to the users to find and formulate the query that precisely captures their information needs. Inspired by existing powerful database management tools such as *Online Analytical Processing* (OLAP), where data may be pre-processed and stored for interactive data analysis, we are interested in compiling relevant information in an offline-phase, in order to support efficient online answering of related queries, as well as making modifications to queries to help users explore their answers.

We consider ontologies in *DL-Lite*$_{\mathcal{R}}$ [3], the DL that underlies the OWL 2 QL profile [7]. We focus on *conjunctive queries* (CQs) that are *tree-shaped* and have exactly one answer variable. We assume that two initial queries are given, which we call the *lower- and upper-bound* query. Intuitively, the lower bound query simply says which kind of objects the user is interested in; it may be very general, like a query that retrieves all persons, or all courses. The upper bound query (which need not have any answers) gathers all properties of interest that these objects may potentially have. We develop a technique for compiling a data

* Funded by FWF projects T515, W1255 and P25207.

structure such that, without accessing the data again, we can efficiently get the answers to any query that asks for objects of the desired type, with different combinations of properties. We also use this compilation to obtain *(minimal) modifications* of a query that retrieve *strictly more* or *strictly less* answers, as well as to obtain the *most specialized query* that retrieves exactly the same answers but makes explicit all their shared properties. A prototype implementation of our algorithms shows promising results, pointing to adequate functionality to support users in their attempts to understand datasets in the presence of ontologies.

2 Preliminaries

We briefly recall the syntax and semantics of *DL-Lite*$_\mathcal{R}$. As usual, N_C, N_R and N_I are countably infinite sets of concept names, role names, and individuals. The set of *roles* is $\overline{N_R} = N_R \cup \{P^- \mid P \in N_R\}$, and R^- denotes the inverse of the role R. Basic concepts are concept names A, and expressions of the form $\exists R$ with R a role. A *TBox* is a set of *axioms* of the forms $R_1 \sqsubseteq R_2$, $R_1 \sqsubseteq \neg R_2$, $C_1 \sqsubseteq C_2$, and $C_1 \sqsubseteq \neg C_2$, where R_1, R_2 are roles and the C_1, C_2 basic concepts. An *ABox* is a finite set of *assertions* of the form $A(b)$, $\neg A(b)$ and $P(b,c)$, $\neg P(b,c)$, where $A \in N_C$, $P \in N_R$ and $b, c \in N_I$. We denote by $\mathbf{Ind}(\mathcal{A})$ the *set of individuals* appearing in $\mathcal{A}$. A *knowledge base (KB)* has the form $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ for $\mathcal{A}$ an ABox and $\mathcal{T}$ a TBox. The semantics of DL KBs is defined in terms of *interpretations* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ with $\Delta^\mathcal{I}$ the *domain* and $\cdot^\mathcal{I}$ the usual *interpretation function*. We use the usual definitions and notations for satisfaction of axioms, assertions, TBoxes and ABoxes, models of a KB, and entailment.

We also recall the syntax and semantics of *conjunctive queries* (CQs) over DL KBs. A CQ has the form $q(\boldsymbol{x}) \coloneq \exists \boldsymbol{y}.\varphi$, where φ is a conjunction of atoms of the form $A(t)$ and $R(t,t')$, where each *term* t, t' is either an individual or a variable from $\boldsymbol{x} \cup \boldsymbol{y}$. The variables in $\boldsymbol{x}$ are called answer variables, and *term*(q) denotes the set of terms occurring in q. Given a CQ q and an interpretation $\mathcal{I}$, we call a *partial match* any function π mapping terms of q to objects in $\Delta^\mathcal{I}$, and we write $\mathcal{I} \models_\pi A(t)$ for a concept atom $A(t)$ if $\pi(t) \in A^\mathcal{I}$, and $\mathcal{I} \models_\pi R(t_1, t_2)$ for role atom $R(t_1, t_2) \in q$ if $(\pi(t_1), \pi(t_2)) \in R^\mathcal{I}$. A *match for q in $\mathcal{I}$* is a partial match with domain *term*(q) such that $\mathcal{I} \models_\pi \alpha$ for every atom α in q. A tuple of individuals $\boldsymbol{t}$ is a *certain answer* to $q(\boldsymbol{x})$ over KB $\mathcal{K}$ if for each $\mathcal{I}$ model of $\mathcal{K}$ there exists a match π for q in $\mathcal{I}$ s.t. $(\boldsymbol{t})^\mathcal{I} = \pi(\boldsymbol{x})$. We use *cert*$(q, \mathcal{K})$ to denote the set of certain answers of q over $\mathcal{K}$. It is well known that if a *DL-Lite*$_\mathcal{R}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is consistent, it possesses a *canonical model* $\mathcal{I}^{\mathcal{T},\mathcal{A}}$ such that for any CQ $q(\boldsymbol{x})$, *cert*$(q, \mathcal{K}) = \{\boldsymbol{t} \mid \mathcal{I}^{\mathcal{T},\mathcal{A}} \models_\pi q, \pi(\boldsymbol{x}) = \boldsymbol{t}\}$ [3]. Its domain $\Delta^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$ consists of all words $aR_1 \dots R_n$ $(n \geq 0)$, where $a \in \mathbf{Ind}(\mathcal{A})$ and each R_i is a role such that $\exists R_{i-1}^- \sqsubseteq_\mathcal{T} \exists R_i$. We let $\mathbf{AnonObj} = \Delta^{\mathcal{I}^{\mathcal{T},\mathcal{A}}} \setminus \mathbf{Ind}(\mathcal{A})$ be the *set of anonymous objects*. The interpretation function $\cdot^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$ is as follows:

$$a^{\mathcal{I}^{\mathcal{T},\mathcal{A}}} = a \qquad A^{\mathcal{I}^{\mathcal{T},\mathcal{A}}} = \{a \mid \mathcal{T}, \mathcal{A} \vDash A(a)\} \cup \{aR_1 \dots R_n \mid n \geq 1, \exists R_n^- \sqsubseteq_\mathcal{T} A\}$$

$$\begin{aligned} P^{\mathcal{I}^{\mathcal{T},\mathcal{A}}} = &\{(a,b) \mid R(a,b) \in \mathcal{A} \text{ and } R \sqsubseteq_\mathcal{T} P\} \cup \{(b,a) \mid R(a,b) \in \mathcal{A} \text{ and } R \sqsubseteq_\mathcal{T} P^-\} \\ &\cup \{(w_1, w_2) \mid w_2 = w_1 S, S \sqsubseteq_\mathcal{T} P\} \cup \{(w_2, w_1) \mid w_2 = w_1 S, S \sqsubseteq_\mathcal{T} P^-\} \end{aligned}$$

We can rely on $\mathcal{I}^{\mathcal{T},\mathcal{A}}$ for query answering, since we can assume that KB consistency has been tested in advance, and queries trivialize over inconsistent KBs.

3 Answering Related Queries

In this paper we only consider CQs of a restricted form, which we call *1treeCQs*. A 1treeCQs is *tree-shaped* (i.e., its primal graph is a tree), and has exactly one answer variable, which is its root. We denote this variable $\mathbf{x}$. We use a special subquery relation between 1treeCQs:

Definition 1 (Subquery). *Given a DL-Lite$_\mathcal{R}$ TBox $\mathcal{T}$ and two 1treeCQs q_1 and q_2, we call q_1* subquery *of q_2 (w.r.t. $\mathcal{T}$), written $q_1 \trianglelefteq_\mathcal{T} q_2$, iff* $\mathrm{term}(q_1) \subseteq \mathrm{term}(q_2)$ *and **(i)** for each atom $R_1(t_1, t_2) \in q_1$ there exists $R_2(t_1, t_2)$ or $R_2^-(t_2, t_1) \in q_2$ such that $R_2 \sqsubseteq_\mathcal{T} R_1$; and **(ii)** for each atom $C_1(t) \in q_1$ there exists $C_2(t) \in q_2$ such that $C_2 \sqsubseteq_\mathcal{T} C_1$. In this case, we also call q_2 a* superquery *of q_1 (w.r.t. $\mathcal{T}$).*

The following claim is straightforward:

Proposition 1. *Let $\mathcal{K}$ be a DL-Lite$_\mathcal{R}$ KB. For any two given 1treeCQs q_1, q_2 such that $q_1 \trianglelefteq_\mathcal{T} q_2$, we have that* $\mathrm{cert}(q_2, \mathcal{K}) \subseteq \mathrm{cert}(q_1, \mathcal{K})$.

3.1 Compiling the Answers of Related Queries

For a given *DL-Lite$_\mathcal{R}$* KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$, let $\mathrm{q_L}$, $\mathrm{q_U}$ be two 1treeCQs such that $\mathrm{q_L} \trianglelefteq_\mathcal{T} \mathrm{q_U}$ holds. We call them *the lower bound query* and the *the upper bound query*, respectively. Any 1treeCQ q such that $\mathrm{q_L} \trianglelefteq_\mathcal{T} q \trianglelefteq_\mathcal{T} \mathrm{q_U}$ is called *in-between query*. We use the term *1treeCQs family* to refer to the set of 1treeCQs containing a pair of $\mathrm{q_L}$ and $\mathrm{q_U}$, together with all the in-between queries. Intuitively, the lower bound is a general query that retrieves all objects the user could be interested in, while the upper bound is a query (which may have no answers) that gathers properties that those objects could potentially have; the family of 1treeCQs contains queries that ask for combinations of those objects.

Example 1. Our examples are based on the well-known LUBM ontology and describe the domain of universities [5]. If we are interested in exploring the properties of the employees in the database, we can use as lower bound $\mathrm{q_L}(\mathbf{x}) : -\mathrm{Employee}(\mathbf{x})$, and as upper bound a complex query with many potential properties of employees, i.e., $\mathrm{q_U}(\mathbf{x}) : -\mathrm{Dean}(\mathbf{x})$, $\mathrm{Prof}(\mathbf{x})$, $\mathrm{FullProf}(\mathbf{x})$, $\mathrm{teacherOf}(\mathbf{x}, y_1)$, $\mathrm{Course}(y_1)$, $\mathrm{headOf}(\mathbf{x}, y_2)$, $\mathrm{Dep}(y_2)$, $\mathrm{autPub}(\mathbf{x}, y_3)$, $\mathrm{Pub}(y_3)$. Some of the in-between queries in the induced family are discussed in Section 5, and depicted in Figures 5 and 6.

In what follows, we assume a given pair of $\mathrm{q_L}$ and $\mathrm{q_U}$, and a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. The goal of this section is to build a structure that stores all information needed for answering any query in the 1treeCQ family, without having to look up $\mathcal{A}$.

To this aim, we introduce the notion of *matching witness*. Intuitively, it stores all objects in the canonical model that may be relevant to a match of an

in-between query. We start from the answers to q_L (since any answer to an in-between query must also be an answer to q_L), and from there, we consider *partial matches* for pairs of terms t_1, t_2 that occur together in an atom $R(t_1, t_2) \in q_U$. For each object in the range of a partial match, we store some *labels* that store the concept and roles the object satisfies that may be relevant to the matches.

We introduce some notation. For $a \in \Delta^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$, $MS_{conc}(a, \mathcal{K})$ denotes the set of *most specialized concepts satisfied by a in $\mathcal{I}^{\mathcal{T},\mathcal{A}}$*, containing each concept name A such that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash A(a)$ and there is no $A' \neq A$ with $A' \sqsubseteq_{\mathcal{T}} A$ and $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash A'(a)$. Similarly, for a pair $a, b \in \Delta^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$, the set of all *most specialized roles satisfied by (a, b) in $\mathcal{I}^{\mathcal{T},\mathcal{A}}$* is denoted $MS_{role}(a, b, \mathcal{K})$ and contains each role R such that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash R(a, b)$ and there is no $R' \neq R$ with $R' \sqsubseteq_{\mathcal{T}} R$ and $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash R'(a, b)$. Given a partial match $\pi := [t_1 \mapsto o_1, t_2 \mapsto o_2]$, a role label R is *relevant* w.r.t. π iff there exists $R'(t_1, t_2) \in q_U$ with $R \sqsubseteq_{\mathcal{T}} R'$ or $R' \sqsubseteq_{\mathcal{T}} R$. Similarly, a concept label C is *relevant* (w.r.t π) iff there exists $C'(t_2) \in q_U$ with $C \sqsubseteq_{\mathcal{T}} C'$ or $C' \sqsubseteq_{\mathcal{T}} C$. For each pair t_1, t_2 of query terms that occur together in some atom $R(t_1, t_2) \in q_U$, and each $o_1 \in \Delta^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$, we define a set of *matching candidates* that contains each o_2 such that if o_1 is a match for t_1 in $\mathcal{I}^{\mathcal{T},\mathcal{A}}$, then o_2 may be a match for t_2.

Definition 2 (Matching candidates). *Let $\{t_1, t_2\} \subseteq \text{term}(q_U)$ be such that there is some $R(t_1, t_2)$ in q_U, and let $o_1 \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{AnonObj}$. The set of* matching candidates *for t_2 relative to $t_1 \mapsto o_1$, denoted $\mathbb{MC}^{t_1 \mapsto o_1}(t_2)$, is:*

I. if $t_2 \in \mathbf{Ind}(\mathcal{A})$, then
  1. if there is some $R \in MS_{role}(o_1, t_2, \mathcal{K})$ that is relevant w.r.t. $[t_1 \mapsto o_1, t_2 \mapsto t_2]$, then $\mathbb{MC}^{t_1 \mapsto o_1}(t_2) := \{t_2\}$,
  2. otherwise, $\mathbb{MC}^{t_1 \mapsto o_1}(t_2) := \emptyset$.

II. if $t_2 \in \text{vars}(q_U)$, then
  1. if $o_1 \in \mathbf{Ind}(\mathcal{A})$, then $\mathbb{MC}^{t_1 \mapsto o_1}(t_2) := cand_{\mathcal{A}} \cup cand_{o_1 R}$ where:

$$cand_{\mathcal{A}} = \{o_2 \mid \mathcal{T}, \mathcal{A} \vDash R(o_1, o_2), R' \sqsubseteq_{\mathcal{T}} R, R'(t_1, t_2) \in q_U\}$$
$$cand_{o_1 R} = \{o_1 R \mid \mathcal{T}, \mathcal{A} \vDash \exists R(o_1), R \text{ is relevant w.r.t } [t_1 \mapsto o_1, t_2 \mapsto o_1 R]\}$$

  2. if $o_1 := wR \in \mathbf{AnonObj}$, then $\mathbb{MC}^{t_1 \mapsto o_1}(t_2) := cand_{wRS} \cup cand_w$ where:

$$cand_{wRS} = \{wRS \mid \mathcal{T} \vDash \exists R^- \sqsubseteq \exists S, S \text{ relevant w.r.t } [t_1 \mapsto wR, t_2 \mapsto wRS]\}$$
$$cand_w = \{w \mid \mathcal{T} \vDash R^- \sqsubseteq S, R' \sqsubseteq_{\mathcal{T}} S, R'(t_1, t_2) \in q_U\}$$

We also define *label strings* that store the concepts and roles for partial matches:

Definition 3 (Label String). *For a partial match π, we define labels(π) as:*

I. if π is of the form $[t \mapsto o]$, where $t \in \text{term}(q_U)$ and $o \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{AnonObj}$, then labels($\pi$) $= \langle \{C \mid C \text{ relevant w.r.t. } \pi\} \rangle$

II. if π is of the form $[t_1 \mapsto o_1, t_2 \mapsto o_2]$ with $o_2 \in \mathbb{MC}^{t_1 \mapsto o_1}(t_2)$, then labels($\pi$) $= \langle \mathbb{R}_{lbl}, \mathbb{C}_{lbl} \rangle$, where $\mathbb{C}_{lbl} = \{C \mid C \in MS_{conc}(o_2, \mathcal{K}) \text{ relevant w.r.t. } \pi\}$, and $\mathbb{R}_{lbl}$ is defined as follows:
  A. if $o_1, o_2 \in \mathbf{Ind}(\mathcal{A})$ or o_1 is of the form $o_2 R$, then $\mathbb{R}_{lbl} = \{R \mid R \in MS_{role}(o_1, o_2, \mathcal{K})$ and R is relevant w.r.t $\pi\}$

B. if $o_2 = o_1 R$, *then* $\mathbb{R}_{lbl} = \{R\}$

Now we are ready to build matching witnesses by iteratively constructing relevant partial matches and storing them together with their label strings:

Definition 4 (Matching Witness). *Let* $a_1, \ldots, a_n$ *be an arbitrary enumeration of the individuals that are a certain answer to* $\mathrm{q_L}$. *We define the* root matching witness *as* $w_{root} = \langle [labels(\mathbf{x} \mapsto a_1)a_1, \ldots labels(\mathbf{x} \mapsto a_n)a_n] \rangle$.

For a given term $t \in \mathrm{term}(\mathrm{q_U})$ *and an object* $o \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{AnonObj}$, *the* matching witness *for* t *and* o *is defined as follows:* $w_o^t = \langle values_{t_1}, \ldots, values_{t_k} \rangle$, *where* $\{t_1, \ldots, t_k\} = \{t' \mid$ *there exists* $R(t, t') \in \mathrm{q_U}\}$, *and* $values_{t_i}$, $1 \leq i \leq k$, *is constructed as follows:*

- *if* $\mathbb{MC}^{t \mapsto o}(t_i) = \{o_1 \ldots o_m\}$, *then* $values_{t_i} = [\phi_1 o_1, \ldots, \phi_m o_m]$, *where* $\phi_j = labels(t \mapsto o, t_i \mapsto o_j)$, $1 \leq j \leq m$.
- *In particular, if* $\mathbb{MC}^{t \mapsto o}(t_i) = \emptyset$, *then* $values_{t_i} = [\varepsilon]$ *(*ε *is the empty string).*

We denote by $\mathbb{W}$ *the set obtained by starting from* $\mathbf{x}$ *and* w_{root}, *and then recursively, for each child* t' *of the current term* t *and each object* o *in* $\mathcal{I}^{\mathcal{T},\mathcal{A}}$ *that occurs in the range of a witness, adding the witness* $w_o^{t'}$ *to* $\mathbb{W}$.

3.2 Compilation-based Query Answering

$\mathbb{W}$ gathers all information that we need to answer any query in the 1treeCQ family. In fact, to decide whether an individual is an answer to an arbitrary in-between query, it suffices to recursively check the following satisfaction conditions. Below, for $t \in term(q)$, we denote by $q \downarrow_t$ the query obtained by restricting q to atoms that involve only terms that are descendants of t in q.

Definition 5 (Satisfaction). *Let* $t \in \mathrm{term}(\mathrm{q_U})$ *and* $q \trianglelefteq_{\mathcal{T}} \mathrm{q_U} \downarrow_t$. *An object* $o \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{AnonObj}$ satisfies q (w.r.t. $\mathbb{W}$) *iff:*

- *if* $t = \mathbf{x}$, *then* o *is an answer to* $\mathrm{q_L}$, *and for each* $C'(\mathbf{x}) \in q$ *there exists* $C \in labels(\mathbf{x} \mapsto o)$ *s. t.* $C \sqsubseteq_{\mathcal{T}} C'$
- *if* t *is a leaf in (the tree-structure of)* q, *then* $w_o^t = \langle\rangle \in \mathbb{W}$

Additionally, the following condition must hold: for each t' *child of* t *in* q *there exists* $o' \in \mathbb{MC}^{t \mapsto o}(t')$ *such that* $labels(t \mapsto o, t' \mapsto o') = \langle \mathbb{R}_{lbl}, \mathbb{C}_{lbl} \rangle$ *satisfies: for each* $R'(t, t') \in q$ *there exists* $R \in \mathbb{R}_{lbl}$ *such that* $R \sqsubseteq_{\mathcal{T}} R'$ *and for each* $C'(t') \in q$ *there exists* $C \in \mathbb{C}_{lbl}$ *such that* $C \sqsubseteq_{\mathcal{T}} C'$; *and* o' *satisfies* $q \downarrow_{t'}$ *(w.r.t.* $\mathbb{W}$*).*

This correctly captures the answers of any query in the 1treeCQ family:

Theorem 1 (Correctness of compilation-based QA). *For each in-between 1treeCQ* q, *we have:* $\mathrm{cert}(q, \mathcal{K}) = \{a \in \mathbf{Ind}(\mathcal{A}) \mid$ *a satisfies* q *(w.r.t.*$\mathbb{W}$*)*$\}$.

4 Query Modifications

Given a query, we identify interesting variations of it that could be relevant to a user. For example, we identify the minimal ways to modify the current query so that it provides more or less answers. We are also interested in identifying all the common properties that our set of answers has, that is, the possible additions to our query that, while making it syntactically more restrictive, would still retrieve the same set of answers. In the experiments in the next section, we will illustrate the usefulness of these query modifications on several examples.

Algorithm 1.1: neutralSpecialization

Input: $\mathbb{W}$ the set of matching witnesses, q in-between query
Output: Q the set of neutral specializations of q

$S := \{\}$;
foreach $a \in \mathbf{ans}(q, \mathbb{W})$ **do**
  $S := S \cup \{q \mid q$ **x**-maximal for $a\}$;
$Q := \emptyset$;
`/* compute intersection of each tuple in the n-fold Cartesian product over S */`
`intersect(`S,Q,q,$\emptyset$,0`)`;
return Q;

Algorithm 1.2: intersect

Input: $S = \{Q_1, \ldots, Q_n\}$ set of sets of queries, $Q_{neutral}$ collects the neutral specializations, q in-between query, q_{inter} intersection query, k the index of current set in S

if $k == n$ **then**
  $Q_{neutral} := Q_{neutral} \cup \{q_{inter}\}$;
else
  foreach *query* $q' \in Q_k$ **do**
    if $q \trianglelefteq_{\mathcal{T}} q'$ **then**
      if $q_{inter} == \emptyset$ **then**
        $q_{inter} := q'$;
      else
        $q_{inter} := q_{inter} \cap_{\mathcal{T}} q'$;
      `intersect(`S,$Q_{neutral}$,q,q_{inter},$k+1$`)`

Algorithm 1.3: strictSpecialization

Input: $\mathbb{W}$ the set of matching witnesses, q in-between query
Output: Q set of strict specializations of q

$Q := \emptyset$;
foreach $a \in \mathbf{ans}(q, \mathbb{W})$ **do**
  foreach q_1 **x**-*maximal for* a **do**
    foreach q_2 *max. neutral spec. of* q **do**
      if $q_2 \trianglelefteq_{\mathcal{T}} q_1$ **then**
        $q_{diff} := q_1 \setminus q_2$;
        foreach *query atom* $A \in q_{diff}$ **do**
          if $q_2 \cup \{A\}$ *is 1treeCQ* **then**
            $Q := Q \cup \{q_2 \cup \{A\}\}$;
return Q;

Algorithm 1.4: minGeneralization

Input: $\mathbb{W}$ the set of matching witnesses, q in-between query
Output: Q the set of minimal generalizations for q

$Q := \emptyset$;
foreach $a \in \mathbf{ans}(\mathrm{q_L}, \mathbb{W})$ *and* $a \notin \mathbf{ans}(q, \mathbb{W})$ **do**
  foreach q'**x**-*maximal for* a **do**
    $q_{inter} := \emptyset$;
    if $q' \sqsubseteq_{\mathcal{T}} q$ **then**
      $Q := Q \cup q'$;
    else
      $q_{inter} := q \sqcap_{\mathcal{T}} q'$;
      if $q_{inter} \neq \emptyset$ **then**
        $Q := Q \cup q_{inter}$;
return Q;

4.1 Construction of Maximal Queries

Intuitively, the set of constructed matching witnesses contains information regarding how much of the upper-bound query we have matched in the canonical model, for each $a \in \mathbf{Ind}(\mathcal{A})$ - possible answer. Hence we can read from the witnesses, for each possible answer a, the set of maximal queries in the family for which a is a match. We will see that these maximal queries provide the key information to suggest the user the desired query modifications. We remark that there can be multiple such maximal queries for the same individual a.

Definition 6 (t-Maximal Query). *Let $q \downarrow_t$ be a 1treeCQ rooted in t, where $t \in \mathrm{term}(\mathrm{q_U})$, s.t. $q \downarrow_t \sqsubseteq_{\mathcal{T}} \mathrm{q_U} \downarrow_t$. A query q is said to be* t-maximal *for $a \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{AnonObj}$ iff: **(a)** a satisfies $q \downarrow_t$ w.r.t. $\mathbb{W}$, and **(b)** for each $q' \downarrow_t$, $q \downarrow_t \neq q' \downarrow_t$, s.t. $q \downarrow_t \sqsubseteq_{\mathcal{T}} q' \downarrow_t \sqsubseteq_{\mathcal{T}} \mathrm{q_U} \downarrow_t$, a does not satisfy $q' \downarrow_t$ w.r.t. $\mathbb{W}$.*

In what follows, we use $ans(q, \mathbb{W})$ to denote the *answers for q over* $\mathbb{W}$, which are the individuals a such that a satisfies q w.r.t. $\mathbb{W}$ (see Theorem 1).

4.2 Specializations and Generalizations

We use the term *query specialization* to refer to any superquery of a given q in a 1treeCQ family. Indeed, superqueries 'specialize' the query by requiring additional or more specific properties to hold. We further distinguish between two kinds of specializations: the ones that retrieve the same answers as the given q, and the ones that retrieve strictly less answers.

Definition 7 (Neutral Specialization). *We call q_2 a* neutral specialization *of q_1 if $q_1 \sqsubseteq_{\mathcal{T}} q_2$ and* $\mathrm{ans}(q_2, \mathbb{W}) = \mathrm{ans}(q_1, \mathbb{W})$.

Trivially, any in-between 1treeCQ is a neutral specialization of itself.

Definition 8 (Strict Specialization). *We call q_2 a* strict specialization *of q_1 if $q_1 \sqsubseteq_{\mathcal{T}} q_2$ and* $\mathrm{ans}(q_2, \mathbb{W}) \subsetneq \mathrm{ans}(q_1, \mathbb{W})$*, with* $\mathrm{ans}(q_2, \mathbb{W}) \neq \emptyset$.

Conversely, any subquery of a given 1treeCQ q is less constrained than q, and retrieves at least the same answers. We are interested in those subqueries that retrieve strictly more answers, which we call *query generalizations*.

The first query modification problem we consider is to construct the maximal neutral specializations of a query. Intuitively, the maximal neutral specialization includes all most specialized additional constraints that we can add to the query without modifying its answers, and hence it makes explicit all the properties that are common to all the answer individuals.

Definition 9 (Maximal Neutral Specialization). *A* maximal neutral specialization *for q w.r.t.* $\text{ans}(q, \mathbb{W})$ *is a neutral specialization q' such that, for each q'' s.t. $q'' \neq q'$ and $q' \trianglelefteq_{\mathcal{T}} q''$, we have* $\text{ans}(q'', \mathbb{W}) \neq \text{ans}(q, \mathbb{W})$.

Algorithm 1.1 computes neutral specializations for a given in-between query q, by intersecting maximal queries for each $a \in ans(q, \mathbb{W})$. Binary operation $\cap_{\mathcal{T}}$ on 1treeCQs keeps only those query atoms that are subsumed in both 1treeCQs. One can show that by dropping all subqueries from the neutral specializations we obtain the maximal neutral specializations.

The second query modification problem we consider is to find the minimal strict specializations of a given query, that is, which are the minimal modifications to the query that will result in a smaller set of answers.

Definition 10 (Minimal Strict Specialization). *A* minimal strict specialization *for q is a strict specialization q' such that, for each q'' with $q \trianglelefteq_{\mathcal{T}} q'' \trianglelefteq_{\mathcal{T}} q'$, q'' is a neutral specialization of q.*

It can be the case that q cannot be strictly specialized without loosing all answers. In this case we say that q's only strict specialization is q_U. One can obtain the minimal strict specializations for a given in-between query by discarding all superqueries from the set of strict specializations, constructed by Algorithm 1.3.

Conversely a *query generalization* refers to any subquery of a given q in a 1treeCQ family that has strictly more answers. Finally, we consider the problem of finding the minimal generalizations of given query, that is, how to minimally relax the query to retrieve more answers.

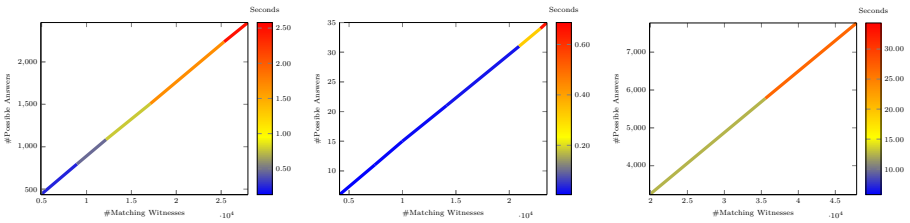
Definition 11 (Minimal Generalization). *Let q be an in-between 1treeCQ. A* minimal generalization or relaxation *for q is a generalization q' such that for each q'', that is $q' \trianglelefteq_{\mathcal{T}} q'' \trianglelefteq_{\mathcal{T}} q$, it holds that:* $\text{ans}(q'', \mathbb{W}) = \text{ans}(q, \mathbb{W})$

It might be the case that an in-between 1treeCQ q cannot be generalized if $ans(q, \mathbb{W}) = ans(\text{q}_\text{L}, \mathbb{W})$. Hence, whenever q is a neutral specialization of q_L it doesn't have any generalizations. Algorithm 1.4 constructs all the minimal generalizations for a given in-between query.

5 Implementation and Experiments

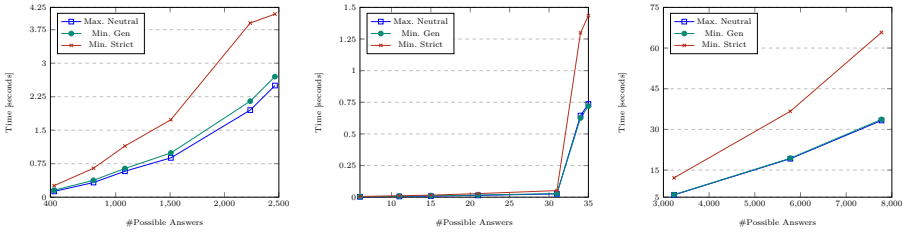
The matching witness solution is implemented in Java (jdk1.8), using additional libraries. For ontology loading we used the OWLAPI java library[1]. For ABox

[1] http://owlapi.sourceforge.net/



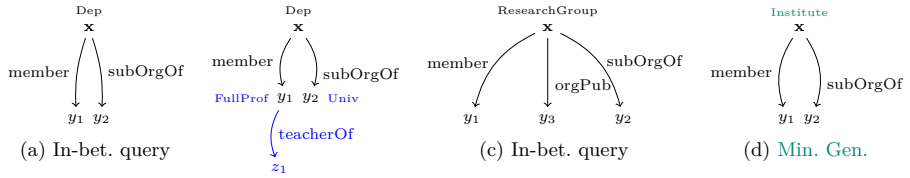
(a) Batch1 (b) Batch2 (c) Batch3

Figure 1: Performance of QA



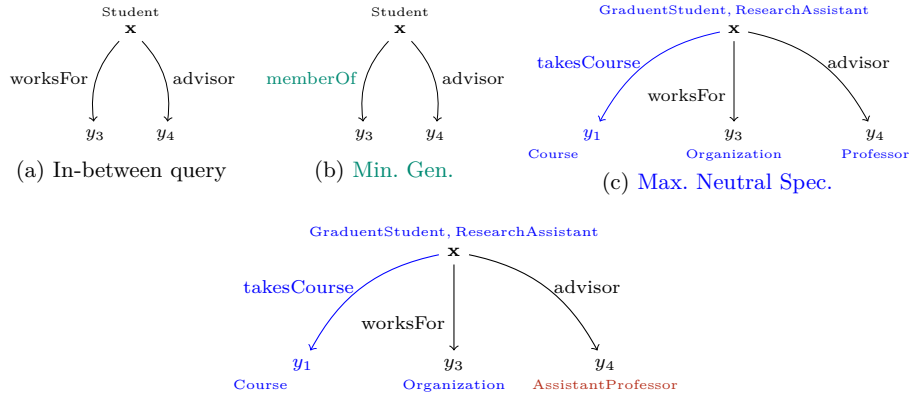
(a) Batch1 (b) Batch2 (c) Batch3

Figure 2: Performance for query modifications



(a) In-bet. query (b) Max. Neutr. Spec. (c) In-bet. query (d) Min. Gen.

Figure 3: Example of query modifications for in-between queries

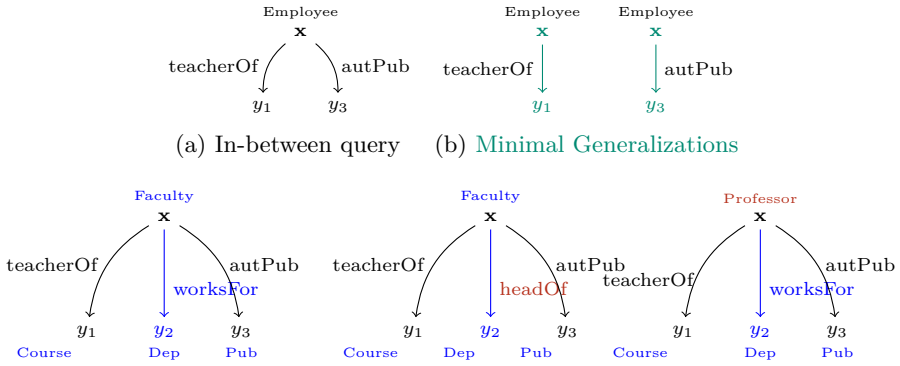


(a) In-between query (b) Min. Gen. (c) Max. Neutral Spec.

(d) Min. Strict Spec.

Figure 4: Example of obtained modifications for in-between query

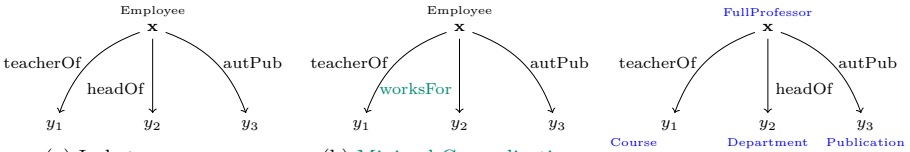
reasoning we used the Ontop [10] since it efficiently provides complete answers for *DL-Lite*$_{\mathcal{R}}$. However, Ontop was specially tailored for OBDA and does not provide full support when the data is not stored using relational databases. So



(a) In-between query (b) Minimal Generalizations

(c) Max. Neutral Spec. (d) Min. Strict Spec.

Figure 5: Example of obtained modifications for in-between query



(a) In-between query (b) Minimal Generalization (c) Max. Neutral Specialization

Figure 6: Example of obtained modifications for in-between query

we used HermiT[2] for TBox reasoning, which captures more expressive DLs but is not as efficient over *DL-Lite*$_{\mathcal{R}}$. To boost the performance we used multi-threading for computing the matching witnesses as well as for query answering.

Specifications. The entire evaluation was done on a server using 7 CPUs (each core running at 2.3GHz), 10 GB RAM, operating system Ubuntu Server amd64, java version Java SE Runtime Environment 7.

Ontology. As ontology we used LUBM [5] - a well known ontology suitable for testing, especially since it provides a data generator, and its domain is easy to comprehend and relatively simple. To match our DL fragment, we used a *DL-Lite*$_{\mathcal{R}}$ version previously used in [6].

Datasets. We used the LUBM generator[3] to compute the data sets. The datasets size vary between $\approx$ 3.5 MB and $\approx$ 19 MB.

Evaluation measurements. We evaluated 3 batches of 1treeCQs families defined by 3 pairs of bound queries that can be accessed online [4]. With this simple prototype implementation, the time for constructing the compilation varies from minutes (for small-sized data sets) up to few hours (for largest data set). We hope to decrease this time dramatically in an improved version that, among others optimizations, does not rely on expensive calls to external reasoners. We measured the number of possible answers for the 1treeCQs family, the number of matching witnesses, the average time for answering an in-between query, and for each query modification, the average time per in-between query.

[2] http://www.hermit-reasoner.com/
[3] http://swat.cse.lehigh.edu/projects/lubm/
[4] https://github.com/medinaAnd/MatchingWitnesses

Figure 1 reflects the results of query answering procedure for each batch. There is a clear dependence on the number of possible answers for the query answering performance. Figure 2 shows the time performance for each query modification on each batch. Given that the modifications call the query answering procedure, the efficiency depends on the number of possible answers.

To illustrate the usefulness of the obtained query modifications, we present in Figures 3, 4, 5, and 6 several variations of some in-between queries obtained with our algorithms. The relevant fragment of the ontology is given below:

$$\text{ResearchGroup} \sqsubseteq_{\mathcal{T}} \text{Institute} \qquad \text{headOf} \sqsubseteq_{\mathcal{T}} \text{worksFor} \qquad \text{AssProf} \sqsubseteq_{\mathcal{T}} \text{Prof}$$
$$\text{worksFor} \sqsubseteq_{\mathcal{T}} \text{memberOf} \qquad \text{Prof} \sqsubseteq_{\mathcal{T}} \text{Faculty} \qquad \text{FullProf} \sqsubseteq_{\mathcal{T}} \text{Prof}$$
$$\text{Faculty} \sqsubseteq_{\mathcal{T}} \text{Employee}$$

Note that query modifications depend also on the *data*, not only on the ontology. E.g., for the maximal neutral specialization 5c, no axiom in the ontology states that each employee that is a head of something is also a full professor.

6 Related Work

Works that aim at assisting users formulate queries include *Quelo* [4], the Faceted Search Interface *SemFacet* [2], and the Optique *virtual query formulation system* (VQS) [11]. Quelo and SemFacet both provide interfaces in which users can build tree-shaped queries like the ones considered here, which can be specialized by adding additional atoms, or generalized by dropping them. Moreover, the interfaces have an ontological reasoning layer that is used to find, for example, relevant specializations to suggest to the user, similarly to our work. However, unlike our work, those interfaces do not aim at compiling data to support online answering of different query variations. Instead, they use off-the-shelf reasoners, and query answering is then done from scratch for each query variation. Another closely related work is presented in [8], where the authors focus on *information overload* and study ways to specialize a query. They consider $\mathcal{EL}$ and arbitrary CQs. We focus only on tree-shaped CQs and *DL-Lite*, but consider also other reasoning problems.

7 Conclusions

In this paper we contributed to the theoretical foundations for compilation-based query answering and construction of useful query modifications in support of data exploration. Our prototype shows overall good performance, however implementation can be further improved. Many questions remain open for future work. A natural next step is to explore how one can extend the solution to other DLs. It seems that this can be done at least for Horn-DLs, since a canonical model can be constructed. Another interesting but challenging question is how to extend the solution in such way that it considers queries that are not tree-shaped. This would be indeed very valuable, and we believe it may be possible.

References

1. Ahmet, S., Evgeny, K., Dmitriy, Z., Ernesto, J.R., Martin, G., Ian, H.: Ontology-based visual query formulation: An industry experience. In: Proceedings of the 11th International Symposium on Visual Computing (ISVC 2015). Springer, Las Vegas, Nevada, USA (2015), http://www.ahmetsoylu.com/wp-content/uploads/2015/10/Soylu_et_al_ISVC2015.pdf
2. Arenas, M., Cuenca Grau, B., Kharlamov, E., Marciuska, S., Zheleznyakov, D.: Faceted search over ontology-enhanced rdf data. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. pp. 939–948. CIKM '14 (2014)
3. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The dl-lite family. J. OF AUTOMATED REASONING p. 2007 (2007)
4. Franconi, E., Guagliardo, P., Tessaris, S., Trevisan, M.: Quelo: an ontology-driven query interface (2011)
5. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. Web Semant. 3(2-3), 158–182 (Oct 2005), http://dx.doi.org/10.1016/j.websem.2005.06.005
6. Kontchakov, R., Rezk, M., Rodriguez-Muro, M., Xiao, G., Zakharyaschev, M.: Answering SPARQL queries over databases under OWL 2 QL entailment regime. In: Proc. of International Semantic Web Conference (ISWC 2014). Lecture Notes in Computer Science, Springer (2014)
7. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: Owl 2 web ontology language: Profiles. World Wide Web Consortium, Working Draft WD-owl2-profiles-20081202 (December 2008)
8. Nutakki, P.: Specializing conjunctive queries in the EL-family for better comprehension of result sets. Master's thesis, TU Dresden (2011)
9. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Spaccapietra, S. (ed.) Journal on Data Semantics X, Lecture Notes in Computer Science, vol. 4900, pp. 133–173. Springer Berlin Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-77688-8_5
10. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: International Semantic Web Conference (1). Lecture Notes in Computer Science, vol. 8218, pp. 558–573. Springer (2013)
11. Soylu, A., Kharlamov, E., Zheleznyakov, D., Jimenez-Ruiz, E., Giese, M., Horrocks, I.: Optiquevqs: Visual query formulation for obda. In: Proceedings of the 27th International Workshop on Description Logics (DL 2014). vol. 1193, pp. 725–728. CEUR-WS.org, Vienna, Austria (2014), http://ceur-ws.org/Vol-1193/paper_88.pdf