Conjunctive Query Answering in $\mathcal{EL}$ using a Database System

Carsten Lutz[1], David Toman[2], and Frank Wolter[3]

[1] Fachbereich Informatik
Universität Bremen, Germany
clu@informatik.uni-bremen.de
[2] D.R. Cheriton School of Computer Science
University of Waterloo, Canada
david@uwaterloo.ca
[3] Department of Computer Science
University of Liverpool, UK
frank@csc.liv.ac.uk

Abstract. We study conjunctive query answering in the description logic $\mathcal{EL}$, the core of the designated OWL2-EL profile of OWL2. In particular, we present an approach that allows the use of conventional relational database technology for conjunctive query answering in $\mathcal{EL}$. This approach is inspired by the OWL support in *Oracle Database 11g, semantic technologies*.

1 Introduction

The OWL ontology language is currently being revisited by a W3C working group with the goal of designing a revised and extended version of OWL, called OWL2 [1]. The new recommendation aims to include a number of *profiles*, fragments of OWL2 that trade expressive power for various other desirable properties not enjoyed by the full language. At the time of writing, there were three profiles designated to become part of the OWL2 recommendation: OWL2-EL based on the description logic $\mathcal{EL}^{++}$ [4, 5], OWL2-QL based on the description logic DL-Lite$_A$ [7], and OWL2-RL inspired by the pD* semantics for OWL [17]. Each of these fragments aims at achieving a different benefit while simultaneously maximizing expressive power. In the case of OWL2-EL, the design goal is classification of ontologies in polynomial time. This profile has already been widely adopted by large-scale ontologies in which efficiency of reasoning plays a central role. In particular, many life-science ontologies such as SNOMED CT and NCI are formulated in OWL2-EL [16, 15].

In contrast to OWL2-EL, the design goal of both OWL2-QL and OWL2-RL is to enable the use of database technology for query answering over OWL instance data: many applications access considerable amounts of such data, often in the range of millions of objects. This makes it necessary to store the data in secondary storage and to develop highly efficient querying mechanisms. Relational database systems often satisfy these needs and benefit from decades of

research in academia and industry. Thus they are natural candidates for implementing a query processing solution for OWL. However, employing such an approach is rather challenging due to significant differences between relational database systems and OWL: first, relational systems adopt a closed-world semantics, i.e., all facts that are not explicitly stated to be true are assumed to be false. In contrast, OWL is based on an open world semantics which does not require one to fix the truth value of every fact and is more similar to an incomplete database [10]. Second, database systems are unaware of the intensional part of an OWL ontology (called henceforth the TBox).

In OWL2-QL and OWL2-RL these difficulties are addressed in different ways. In a nutshell, the OWL2-QL approach suggests employing a *backward chaining* approach while OWL2-RL suggests *forward chaining*. In both cases the instance data (called the ABox) is stored as a relational database instance. In OWL2-QL, a query q is *rewritten* into a new query q^* that takes into account the TBox. This rewriting is independent of the data. The resulting query q^* is passed to the database system to retrieve answers. In this way, OWL2-QL allows the use of an off-the-shelf relational database system with the help of an external query rewriting component. In contrast, query answering in OWL2-RL relies on a *data preprocessing* phase. This phase is initiated after each modification of the instance data and adds data that is entailed by the explicitly given data and the TBox. Data preprocessing is independent of the queries to be answered. To actually answer a query, it then suffices to pass it to the relational query engine without rewriting. The OWL2-RL approach thus suggests that the data management facilities of the database system should be modified to incorporate a data preprocessing phase, *but the query engine is left untouched*. Notably, OWL2-RL and the described approach to query answering is used in the *Oracle Database 11g semantic technologies* [18].

In this paper, we study conjunctive query answering in the description logic $\mathcal{EL}$ using relational database systems. Since $\mathcal{EL}$ can be viewed as the core of the OWL2-EL profile and is successfully used in the design of large-scale ontologies, adding efficient querying capabilities is of immediate practical use, e.g., in applications of OWL in medical informatics in which an ABox is used to describe medical records and where a medical $\mathcal{EL}$ ontology, such as SNOMED CT, assigns a meaning to the medical terms used in these records [12, 13]. Unsurprisingly, very large ABoxes are frequently encountered in such applications and efficient querying is a crucial requirement. Conjunctive query answering in $\mathcal{EL}$ has also been studied in [9, 14, 8], but not in the context of database systems.

The pure backward chaining approach of OWL2-QL cannot be applied to $\mathcal{EL}$ since that approach is limited to DLs for which the data complexity of conjunctive query answering is in LogSpace, while it has been shown that $\mathcal{EL}$ is PTime-hard in this respect [6]. Pure forward chaining is not applicable either because it may cause the introduction of infinitely many new database objects (due to the use of existential restrictions on the right-hand side of concept inclusions in a TBox). For these reasons, we present a novel approach to conjunctive query answering that has some similarity to both of the approaches above. It

can be summarized as follows similarly to *Oracle 11g semantic technologies*, we use a data preprocessing phase that adds new data to the database. This phase depends on the ABox and the TBox, but not on the queries to be answered. The data added during our preprocessing phase is auxiliary: it is only used internally by the ontology-aware database system for query answering and hidden from the user. To answer a query, we first rewrite it and then pass it on to the database query engine. Hence, our approach shares the main virtues of the two approaches for OWL2-QL and OWL2-RL above; in particular it leaves the query engine of the relational database system unchanged. Technically, our approach relies on the fact that, in $\mathcal{EL}$, conjunctive queries can be answered by considering a single canonical model. During the preprocessing phase, we complete the data in the database to such a model. The main challenge of pursuing this approach is to work with a finite canonical model while still giving correct answers to all conjunctive queries.

It is worth mentioning that the query rewriting in our approach is of a rather different nature than the rewriting done for OWL2-QL: it does not correspond to backward chaining and is independent of the data *and of the TBox*. Another significant difference from OWL2-QL is that data preprocessing needs only quadratic time and query rewriting needs only linear time. In contrast, the query rewriting step of OWL2-QL may induce an exponential blowup.

The remainder of this paper is organized as follows. In Section 2, we introduce the description logic $\mathcal{EL}$ and conjunctive query answering. Section 3 then provides an overview of our approach, introduces canonical models, and provides a first example. The data preprocessing phase is described in Section 4 and query rewriting is discussed in Section 5. In Section 6, we conclude with pointing out future research issues.

2 Preliminaries

We briefly introduce the description logic $\mathcal{EL}$ and conjunctive queries. In $\mathcal{EL}$, *concepts* are built according to the syntax rule

$$C ::= A \mid \top \mid C \sqcap D \mid \exists r.C$$

where, here and in the remaining paper, A ranges over *concept names* taken from a countably infinite set $\mathsf{N_C}$, r ranges over *role names* taken from a countably infinite set $\mathsf{N_R}$, and C, D range over concepts. A *TBox* is a finite set of *concept inclusions* $C \sqsubseteq D$, and an ABox is a finite set of *concept assertions* $A(a)$ and *role assertions* $r(a, b)$, where a, b range over a countably infinite set $\mathsf{N_I}$ of *individual names*. A *knowledge base* is a pair $(\mathcal{T}, \mathcal{A})$ with $\mathcal{T}$ a TBox and $\mathcal{A}$ an ABox.

As usual, an *interpretation* is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\Delta^{\mathcal{I}}$ a non-empty *domain* and $\cdot^{\mathcal{I}}$ an *interpretation function* that maps each concept name A to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each role name r to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ The interpretation function is extended

to composite concepts by setting

$$\begin{aligned} \top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\ (C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\ (\exists r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \exists e \in \Delta^{\mathcal{I}} : (d,e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}. \end{aligned}$$

An interpretation $\mathcal{I}$ *satisfies* a concept inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, a concept assertion $A(a)$ if $a^{\mathcal{I}} \in A^{\mathcal{I}}$, and a role assertion $r(a,b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ (ABox $\mathcal{A}$) if it satisfies all concept inclusions in $\mathcal{T}$ (assertions in $\mathcal{A}$). It is a model of a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ if it is a model of $\mathcal{T}$ and $\mathcal{A}$. For a concept inclusion or assertion α, we write $\mathcal{K} \models \alpha$ if α is satisfied in all models of $\mathcal{K}$. If empty, $\mathcal{A}$ is simply omitted.

Let $\mathsf{N_V}$ be a countably infinite set of *variables*. Together, the set of $\mathsf{N_V}$ of variables and the set $\mathsf{N_I}$ of individual names form the set $\mathsf{N_T}$ of *terms*. A *conjunctive query* is an expression of the form $\exists \boldsymbol{u}.\varphi(\boldsymbol{v}, \boldsymbol{u})$, where

- $\boldsymbol{u} = u_1, \ldots, u_n$ and $\boldsymbol{v} = v_1, \ldots, v_m$ are vectors of variables and
- φ is a conjunction of *concept atoms* $A(t)$ and *role atoms* $r(t, t')$, where A ranges over concept names, r over role names, and t, t' are terms from the set $\boldsymbol{v} \cup \boldsymbol{u} \cup \mathsf{N_I}$.

The variables in $\boldsymbol{u}$ are called *quantified variables*, and the variables in $\boldsymbol{v}$ are *answer variables*. We call the query *k-ary* if there are k answer variables, use $\mathsf{var}(q)$ to denote the set of all variables in q, $\mathsf{qvar}(q)$ for the set of quantified variables, $\mathsf{avar}(q)$ for the set of answer variables, and $\mathsf{term}(q)$ for the terms in q. Slightly abusing notation, we write $\alpha \in q$ if the concept or role atom α occurs in q.

Let $\mathcal{I}$ be an interpretation and $q = \exists \boldsymbol{u}.\varphi(\boldsymbol{v}, \boldsymbol{u})$ a conjunctive query. A *match* for $\mathcal{I}$ and q is a mapping $\pi : \mathsf{term}(q) \to \Delta^{\mathcal{I}}$ such that $\pi(a) = a^{\mathcal{I}}$ for all $a \in \mathsf{term}(q) \cap \mathsf{N_I}$ and all atoms in q are satisfied, i.e.,

- $\pi(t) \in A^{\mathcal{I}}$ for all $A(t) \in q$ and
- $(\pi(t), \pi(t')) \in r^{\mathcal{I}}$ for all $r(t, t') \in q$.

If $\boldsymbol{v} = v_1, \ldots, v_k$ with $\pi(v_i) = a_i^{\mathcal{I}}$ for $1 \leq i \leq k$, then π is called an $(a_1, \ldots, a_k)$-match for $\mathcal{I}$ and q. If such a match exists, we write $\mathcal{I} \models q[a_1, \ldots, a_k]$. A *certain answer* for a k-ary conjunctive query q and a knowledge base $\mathcal{K}$ is a tuple $(a_1, \ldots, a_k)$ such that $a_1, \ldots, a_k$ occur in $\mathcal{K}$ and, for each model $\mathcal{I}$ of $\mathcal{K}$, there is an $(a_1, \ldots, a_k)$-match for $\mathcal{I}$ and q. We use $\mathsf{cert}(q, \mathcal{K})$ to denote the set of all certain answers for q and $\mathcal{K}$. This defines the querying problem studied in this paper: given an $\mathcal{EL}$-knowledge base $\mathcal{K}$ and a conjunctive query q, we want to compute $\mathsf{cert}(q, \mathcal{K})$. Observe that the definition of certain answers reflects the open world semantics of ABoxes: we quantify over all possible models of $\mathcal{K}$ instead of *treating the ABox itself as a model*.

Without loss of generality, we admit only concept names in a conjunctive query, instead of composite concepts. Indeed, it is easily possible to *unfold* an

$\mathcal{EL}$-concept into the query. For example, the query $\exists u.r(u,v) \wedge (A \sqcap \exists s.B)(u)$ can be unfolded into $\exists u, u'.r(u,v) \wedge A(u) \wedge s(u,u') \wedge B(u')$.

In the remainder of this paper, we make the *unique name assumption*, i.e., we assume that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ for all interpretations $\mathcal{I}$ and all $a, b \in \mathsf{N_I}$ with $a \neq b$. The following (easy to prove) lemma shows that assuming UNA does not impact the certain answers.

Lemma 1. *Let $\mathcal{K}$ be a knowledge base, q a k-ary conjunctive query, and let $a_1, \ldots, a_k \in \mathsf{N_I}$. If there is a model $\mathcal{I}$ of $\mathcal{K}$ and an $(a_1, \ldots, a_k)$-match for $\mathcal{I}$ and q, then there is a model $\mathcal{I}'$ of $\mathcal{K}$ that respects the UNA and an $(a_1, \ldots, a_k)$-match for $\mathcal{I}'$ and q.*

3 Canonical Models

Every $\mathcal{EL}$ knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a *canonical model* $\mathcal{I}_\mathcal{K}$ that enjoys many pleasant properties [2, 11]. To define $\mathcal{I}_\mathcal{K}$, let $\mathsf{sub}(\mathcal{T})$ denote the set of all subconcepts of concepts used in $\mathcal{T}$, $\mathsf{Ind}(\mathcal{A})$ the set of individual names that occur in $\mathcal{A}$, and set $\mathsf{NI_{aux}} := \{x_C \mid C \in \mathsf{sub}(\mathcal{T})\}$. Then

$$
\begin{aligned}
\Delta^{\mathcal{I}_\mathcal{K}} &:= \mathsf{Ind}(\mathcal{A}) \cup \mathsf{NI_{aux}} \\
A^{\mathcal{I}_\mathcal{K}} &:= \{a \in \mathsf{ind}(\mathcal{A}) \mid \mathcal{K} \models A(a)\} \cup \{x_C \in \mathsf{NI_{aux}} \mid \mathcal{K} \models C \sqsubseteq A\} \\
r^{\mathcal{I}_\mathcal{K}} &:= \{(a,b) \in \mathsf{ind}(\mathcal{A}) \times \mathsf{ind}(\mathcal{A}) \mid r(a,b) \in \mathcal{A}\} \cup \\
&\quad\ \{(a, x_C) \in \mathsf{ind}(\mathcal{A}) \times \mathsf{NI_{aux}} \mid \mathcal{K} \models \exists r.C(a)\} \cup \\
&\quad\ \{(x_C, x_D) \in \mathsf{NI_{aux}} \times \mathsf{NI_{aux}} \mid \mathcal{K} \models C \sqsubseteq \exists r.D\} \\
a^{\mathcal{I}_\mathcal{K}} &:= a \text{ for all } a \in \mathsf{ind}(\mathcal{K})
\end{aligned}
$$

Among the pleasant properties of $\mathcal{I}_\mathcal{K}$ is that it can be used to answer *instance queries*. Such a query has the form $C(a)$ and is entailed by a knowledge base $\mathcal{K}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{K}$. It is well-known that $C(a)$ is entailed by $\mathcal{K}$ iff $a \in C^{\mathcal{I}_\mathcal{K}}$, and thus we can answer instance queries w.r.t. $\mathcal{K}$ by looking only at the single model $\mathcal{I}_\mathcal{K}$ [4]. Since $\mathcal{A}$ can be viewed as being a fragment of $\mathcal{I}_\mathcal{K}$, this observation suggests that we can use a database system to answer instance queries as follows: we store $\mathcal{A}$ as a database, complete $\mathcal{A}$ to $\mathcal{I}_\mathcal{K}$ in a preprocessing phase, and then just pass instance queries to the database system.

In principle, it is the same approach that we want to use for conjunctive queries. Unfortunately, for conjunctive queries q it is *not* true that $(a_1, \ldots, a_k) \in \mathsf{cert}(q, \mathcal{K})$ iff $\mathcal{I}_\mathcal{K} \models q[a_1, \ldots, q_k]$ for all $a_1, \ldots, a_k \in \mathsf{N_I}$ with k the arity of q. To see this, let us consider four examples. First, take the knowledge base $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{A}_1)$ with

$$
\begin{aligned}
\mathcal{T}_1 &= \{A \sqsubseteq \exists r.B\} \\
\mathcal{A}_1 &= \{A(a), A(b), r(a', c'), r(b', c')\} \\
q_1 &= \exists u.r(v,u) \wedge r(v', u)
\end{aligned}
$$

We have $(a, x_B) \in r^{\mathcal{I}_{\mathcal{K}_1}}$ and $(b, x_B) \in r^{\mathcal{I}_{\mathcal{K}_1}}$, and thus $\mathcal{I}_{\mathcal{K}_1} \models q[a,b]$. However, $(a,b) \notin \mathsf{cert}(q_1, \mathcal{K}_1)$ because it is easy to find a model $\mathcal{I}$ of $\mathcal{K}_1$ with $\mathcal{I} \not\models q_1[a,b]$.

Second, take

$$\begin{aligned}\mathcal{T}_2 &= \{A \sqsubseteq \exists r.B \sqcap \exists s.B\}\\ \mathcal{A}_2 &= \{A(a)\}\\ q_2 &= \exists u.r(v,u) \wedge s(v,u)\end{aligned}$$

Similarly to our first example, it is not hard to see that $\mathcal{I}_{\mathcal{K}_2} \models q_2[a]$, but $a \notin \mathsf{cert}(q_2, \mathcal{K}_2)$. Our third example is $\mathcal{K}_3 = (\mathcal{T}_3, \mathcal{A}_3)$ with

$$\begin{aligned}\mathcal{T}_3 &= \{A \sqsubseteq \exists r.B,\ B \sqsubseteq \exists s.B\}\\ \mathcal{A}_3 &= \{A(a)\}\\ q_3 &= \exists u.r(v,u) \wedge s(u,u)\end{aligned}$$

Again, $\mathcal{I}_{\mathcal{K}_3} \models q_3[a]$, but $a \notin \mathsf{cert}(q_3, \mathcal{K}_3)$. All the three examples above show situations in which matches can be found in the canonical model, matches that do not exist in the unraveling of this model. Our last example is $\mathcal{K}_4 = (\mathcal{T}_4, \mathcal{A}_4)$ with

$$\begin{aligned}\mathcal{T}_4 &= \{A \sqsubseteq A\}\\ \mathcal{A}_4 &= \{B(a)\}\\ q_4 &= \exists u.B(v) \wedge A(u)\end{aligned}$$

illustrates a different difficulty: while $x_A \in A^{\mathcal{I}_{\mathcal{K}_4}}$, in the unraveled model the interpretation of A is empty: hence, $\mathcal{I}_{\mathcal{K}_4} \models q_4[a]$, but $a \notin \mathsf{cert}(q_4, \mathcal{K}_4)$. This situation can be detected by observing that the value x_4 is not reachable from a by a role chain in $\mathcal{I}_{\mathcal{K}_4}$.

In principle, all of these problems can be overcome by replacing $\mathcal{I}_{\mathcal{K}}$ with its unraveling into a less constrained, tree-like model. In the following, we introduce unraveling as a general operation on models of a knowledge base. Let $\mathcal{K}$ be a knowledge base and $\mathcal{I}$ a model of $\mathcal{K}$. We use $\mathsf{Ind}(\mathcal{A})^{\mathcal{I}}$ to denote the set $\{a^{\mathcal{I}} \mid a \in \mathsf{Ind}(\mathcal{A})\}$. A *path* in $\mathcal{I}$ is a finite sequence $d_0 r_1 d_1 \cdots r_n d_n$, $n \geq 0$, where $d_0 \in \mathsf{Ind}(\mathcal{A})^{\mathcal{I}}$ and, for all $i < n$, $(d_i, d_{i+1}) \in r_{i+1}^{\mathcal{I}}$. We use $\mathsf{paths}(\mathcal{I})$ to denote the set of all paths in $\mathcal{I}$. If $p \in \mathsf{paths}(\mathcal{I})$, then $\mathsf{tail}(p)$ denotes the last element d_n in p. Now the *unraveling* $\mathcal{J}$ of $\mathcal{I}$ is defined as follows:

$$\begin{aligned}\Delta^{\mathcal{J}} &:= \mathsf{paths}(\mathcal{I})\\ a^{\mathcal{J}} &:= a^{\mathcal{I}}\\ A^{\mathcal{J}} &:= \{p \mid \mathsf{tail}(p) \in A^{\mathcal{I}}\}\\ r^{\mathcal{J}} &:= \{(d,e) \mid d,e \in \mathsf{Ind}(\mathcal{A})^{\mathcal{I}} \wedge (d,e) \in r^{\mathcal{I}}\} \ \cup \{(p, p \cdot re) \mid p, p \cdot re \in \Delta^{\mathcal{I}}\}\end{aligned}$$

where "$\cdot$" denotes concatenation. The following result is proved in [8] for the extension of $\mathcal{EL}$ with inverse and functional roles and for the case of 0-ary queries. An extension to k-ary queries is straightforward.

Lemma 2. *Let $\mathcal{K}$ be a knowledge base and $\mathcal{U}_{\mathcal{K}}$ the unraveling of $\mathcal{I}_{\mathcal{K}}$. For all k-ary conjunctive queries q and individual names $a_1, \ldots, a_k$, we have $(a_1, \ldots, a_k) \in \mathsf{cert}(q, \mathcal{K})$ iff $\mathcal{U}_{\mathcal{K}} \models q[a_1, \ldots, a_k]$.*

Alas, $\mathcal{U}_\mathcal{K}$ may be infinite and thus we cannot use it as a database. The solution is to continue working with $\mathcal{I}_\mathcal{K}$, but to rewrite q into q^* such that each match of q^* in $\mathcal{I}_\mathcal{K}$ can be reproduced as a match of q in $\mathcal{U}_\mathcal{K}$ and vice versa, thus $\mathcal{I}_\mathcal{K} \models q^*[a_1, \ldots, a_k]$ iff $\mathcal{U}_\mathcal{K} \models q[a_1, \ldots, a_k]$ for all $a_1, \ldots, a_k \in \mathsf{N_I}$. We defer a formal definition and concrete examples of this rewriting to Section 5.

4 Data Preprocessing

We describe the data preprocessing phase of our approach to conjunctive query answering in $\mathcal{EL}$, where we insert additional, auxiliary data. The purpose of this phase is to complete the ABox $\mathcal{A}$ initially stored in the database to (a fragment of) the canonical model $\mathcal{I}_\mathcal{K}$. In principle, the auxiliary data has to be updated each time that $\mathcal{A}$ and $\mathcal{T}$ are modified (for the purposes of query answering, however, it makes sense to assume that $\mathcal{T}$ is essentially immutable). To keep the presentation simple, we concentrate on the case where the auxiliary data is computed from scratch. In an actual implementation, it would be preferable to modify the auxiliary data in an incremental way when data is inserted to or deleted from $\mathcal{A}$.

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be the knowledge base over which queries are to be answered. We store the ABox $\mathcal{A}$ as data in a database system in the obvious way: individual names become database objects, concept names become unary tables, and role names become binary tables. When we understand $\mathcal{A}$ as a database, we denote it with $\mathcal{A}_{db}$ and use $\mathsf{ans}(q, \mathcal{A}_{db})$ to denote the set of answers that a relational database system returns for q over the database $\mathcal{A}_{db}$. As explained in the previous section, $\mathsf{cert}(q, \mathcal{K})$ and $\mathsf{ans}(q, \mathcal{A}_{db})$ are not identical in general. The precompletion phase extends $\mathcal{A}_{db}$ to a database $\mathcal{A}^*_{db}$ that represents (a fragment of) the canonical model $\mathcal{I}_\mathcal{K}$. In the following, we present details of how this completion can be achieved using the query mechanism of a relational system.

In $\mathcal{A}^*_{db}$, we use the elements of $\mathsf{NI_{aux}}$ as additional database objects, assuming that $\mathsf{NI_{aux}} \cap \mathsf{N_I} = \emptyset$. We also introduce an additional unary database table Aux that is used to identify elements of $\mathsf{NI_{aux}}$ and distinguish them from individual names in $\mathcal{A}$.[4] First, we use a reasoner such as CEL [3] to compute the canonical model of the TBox $\mathcal{T}$ defined as follows:

$$\begin{aligned} \Delta^{\mathcal{I}_\mathcal{T}} &:= \mathsf{NI_{aux}} \\ A^{\mathcal{I}_\mathcal{T}} &:= \{x_C \in \mathsf{NI_{aux}} \mid \mathcal{T} \models C \sqsubseteq A\} \\ r^{\mathcal{I}_\mathcal{T}} &:= \{(x_C, x_D) \in \mathsf{NI_{aux}} \times \mathsf{NI_{aux}} \mid \mathcal{T} \models C \sqsubseteq \exists r.D\} \end{aligned}$$

This model should not to be confused with the canonical model $\mathcal{I}_\mathcal{K}$ of the knowledge base $\mathcal{K}$; in fact, $\mathcal{I}_\mathcal{T}$ is obviously a fragment of $\mathcal{I}_\mathcal{K}$. It can be computed in polynomial time and its size is at most quadratic in the size of $\mathcal{T}$. To properly insert $\mathcal{I}_\mathcal{T}$ into the database, we need a notion of reachability between domain elements of $\mathcal{I}_\mathcal{T}$. For $x_C, x_D \in \Delta^{\mathcal{I}_\mathcal{T}}$, we say that x_D is *reachable* from x_C if there are $x_{C_0}, \ldots, x_{C_n} \in \Delta^{\mathcal{I}_\mathcal{T}}$, $n \geq 0$, such that $x_{C_0} = x_C$, $x_{C_n} = x_D$, and for all $i < n$,

[4] Thus, Aux denotes the complement of $\mathsf{Ind}(\mathcal{A})^{\mathcal{I}_\mathcal{K}}$, as introduced in Section 3.

for all $C, A \in \mathsf{sub}(\mathcal{T})$ such that for some $C \sqsubseteq D \in \mathcal{T}$, A is a conjunct of D **do**
  extend the table A with $\mathsf{ans}(q_C, \mathcal{A}^{(i)}_{db})$

for all $C, \exists r.D \in \mathsf{sub}(\mathcal{T})$ such that for some $C \sqsubseteq E \in \mathcal{T}$, $\exists r.D$ is a conjunct of E **do**
  if $\mathsf{ans}(q_C, \mathcal{A}^{(i)}_{db}) \neq \emptyset$ **then**
    extend the table r with $\{(a, x_D) \mid a \in \mathsf{ans}(q_C, \mathcal{A}^{(i)}_{db})\}$
    extend the table Aux with $\mathsf{Reach}(x_D)$
    extend the table A with $A^{\mathcal{I}_\mathcal{T}} \cap \mathsf{Reach}(x_D)$, for each concept name A
    extend the table s with $s^{\mathcal{I}_\mathcal{T}} \cap (\mathsf{Reach}(x_D) \times \mathsf{Reach}(x_D))$, for each role name s
  endif

Fig. 1. The central procedure for data preprocessing.

there is an $r \in \mathsf{N_R}$ with $(x_{C_i}, x_{C_{i+1}}) \in r^{\mathcal{I}_\mathcal{T}}$. We use $\mathsf{Reach}(x_C)$ to denote the set of those $x_D \in \Delta^{\mathcal{I}_\mathcal{T}}$ that are reachable from x_C (including x_C itself). Now, the data preprocessing phase constructs a sequence of ABoxes $\mathcal{A}^{(1)}_{db}, \ldots \mathcal{A}^{(k)}_{db}$ starting with $\mathcal{A}^{(0)}_{db} = \mathcal{A}_{db}$ and then repeatedly applying the procedure from Figure 1 to produce $\mathcal{A}^{(i+1)}_{db}$ from $\mathcal{A}^{(i)}_{db}$. In the figure, q_C denotes the result of converting the concept C into a tree-shaped conjunctive query with one answer variable that denotes the root (c.f. the remark at the end of Section 2) and "C is a conjunct of D" includes the case where $C = D$. The repeated application of this procedure stops when no more changes to the data occur.

It is not hard to see that $\mathcal{A}^*_{db}$ represents $\mathcal{I}_\mathcal{K}$ restricted to those domain elements that are reachable (along roles) from some individual name in $\mathcal{A}$. This restriction to reachable elements is not (only) an optimization: it is needed in order to eliminate the problem illustrated in the fourth example $(\mathcal{K}_4, q_4)$ in previous section. We now analyze the complexity of the preprocessing phase.

Lemma 3. *Let n be the number of individual names in $\mathcal{A}$ and m the number of subconcepts in $\mathcal{T}$. Then the procedure in Figure 1 is applied at most $n \cdot m$ times.*

We note that n and m are linear in the size of $\mathcal{A}$ and $\mathcal{T}$, and that the bound given in Lemma 3 is only an upper bound that is unlikely to occur in practice. Let us use $\mathcal{A}^*_{db}$ to denote the final database that is constructed in the preprocessing phase. It is not hard to see that the size of $\mathcal{A}^*_{db}$ is bounded by $\mathcal{O}(n \cdot m)$, where n and m are as in Lemma 3.

5 Query Rewriting

This section shows how to rewrite a given conjunctive query q into a (domain independent) first-order query q^* such that $\mathsf{cert}(q, \mathcal{K}) = \mathsf{ans}(q^*, \mathcal{A}^*_{db})$ for every knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, i.e., the answers that a relational system returns for q^* are precisely the certain answers for q and $\mathcal{K}$. We assume that $\mathcal{A}^*_{db}$ (defined in the previous section) is stored in a relational database system. We remind the reader that domain independent first-order queries are nothing else but SQL queries. In particular, every domain independent FO query can be rewritten into an SQL query in linear time.

To proceed, we use several auxiliary definitions. Let $\sim_q$ denote the smallest relation on $\mathsf{term}(q)$ that includes the identity relation, is transitive, and satisfies the following closure condition:

$(*)$ if there are $r(s,t), r(s',t') \in q$ with $t \sim_q t'$, then $s \sim_q s'$.

The relation $\sim_q$ is central to our rewriting procedure. Recall from Section 3 that the goal of query rewriting is to produce a query q^* such that matches of q^* in $\mathcal{I}_\mathcal{K}$ can be reproduced as matches of q in the unraveling $\mathcal{U}_\mathcal{K}$ of $\mathcal{I}_\mathcal{K}$ and vice versa. Intuitively, $\mathcal{U}_\mathcal{K}$ is produced from $\mathcal{I}_\mathcal{K}$ by keeping the $\mathsf{Ind}(\mathcal{A})$-part of $\mathcal{I}_\mathcal{K}$ intact and relaxing the $\mathsf{NI}_{\mathsf{aux}}$-part into a collection of trees. The importance of $(*)$ can be seen when assuming $t = t'$: then $(*)$ describes a non-tree situation in the query since $t = t'$ has two predecessors s and s'. Therefore, we should avoid matches π of q to the $\mathsf{NI}_{\mathsf{aux}}$-part of $\mathcal{I}_\mathcal{K}$ where $\pi(s) \neq \pi(s')$ as we will not be able to reproduce them in $\mathcal{U}_\mathcal{K}$. The case where $t \sim_q t'$ instead of $t = t'$ can be understood similarly.

It is not hard to verify that $\sim_q$ is an equivalence relation and can be computed in time polynomial in the size of q. For each equivalence class ζ of $\sim_q$, choose a representative $t_\zeta \in \zeta$. Let

- $\mathsf{Fork}_=$ be the set of pairs (T, ζ) with $T \subseteq \mathsf{term}(q)$ and ζ an equivalence class of $\sim_q$ such that for some $r \in \mathsf{N_R}$, $T = \{t \in \mathsf{term}(q) \mid \exists t' \in \zeta : r(t,t') \in q\}$, and T is of cardinality at least two;

- $\mathsf{Fork}_{\neq} \subseteq \mathsf{qvar}(q)$ be the set of quantified variables v such that for some $s, s', t \in \mathsf{term}(q)$ and $r, r' \in \mathsf{N_R}$ with $r \neq r'$, we have $r(s,v), r'(s',t) \in q$ and $v \sim_q t$;

- $\mathsf{Cyc} \subseteq \mathsf{qvar}(q)$ be the set of quantified variables v such that there are

$$r_0(t_0, t_0'), \ldots, r_{n-1}(t_n, t_n') \in q,\ n \geq 0,$$

  with $v = t_i$ for some $i \leq n$ and $t_i' \sim_q t_{i+1} \bmod n$ for all $i < n$.

It is not hard to see that $\mathsf{Fork}_=$, $\mathsf{Fork}_{\neq}$, and Cyc can be computed in time polynomial in the size of q. The rewritten query q^* is defined as $q \wedge q_1 \wedge q_2$, where q_1 and q_2 are as follows:

$$q_1 := \bigwedge_{v \in \mathsf{avar}(q) \cup \mathsf{Fork}_{\neq} \cup \mathsf{Cyc}} \neg \mathsf{Aux}(v)$$

$$q_2 := \bigwedge_{(\{t_1,\ldots,t_k\},\zeta) \in \mathsf{Fork}_=} (\,\mathsf{Aux}(t_\zeta) \rightarrow (t_1 = t_2 \wedge \cdots \wedge t_{k-1} = t_k)\,)$$

Note that the construction of q^* does not depend on $\mathcal{K}$ and can be carried out in polynomial time. Moreover, q^* is at most linear in the size of q. To see this, first note that the number of conjuncts in q_1 is bounded by the number of variables in q. Regarding q_2, let $\mathsf{Fork}_= = \{(T_1, \zeta_1), \ldots, (T_\ell, \zeta_\ell)\}$. It is not hard to see that

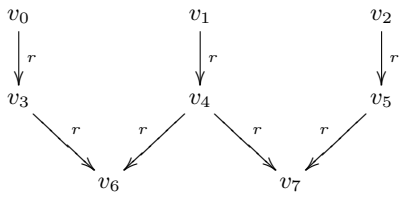

Fig. 2. Example Query.

$|T_1|+\cdots+|T_\ell|$ is bounded by the number of role atoms in q, and thus the number of conjuncts in q_2 is also bounded linearly in the size of q. Since q is a conjunct of q^*, it is readily checked that q^* is domain independent.

The following theorem states the correctness of our approach to conjunctive query answering in $\mathcal{EL}$ using a database system. Its proof is given in the full version of this paper available at http://lat.inf.tu-dresden.de/∼clu.

Theorem 1. $\mathsf{cert}(q, \mathcal{K}) = \mathsf{ans}(q^*, \mathcal{A}^*_{db})$.

We now give three examples for query rewriting:

- For tree-shaped queries and for queries without quantified variables almost no query rewriting is needed: in both cases $q^* = q \wedge \bigwedge_{v \in \mathsf{avar}(q)} \neg \mathsf{Aux}(v)$.

- The queries q_1 to q_3 from Section 3 are rewritten as follows. In the case of $q_1 = \exists u.r(v,u) \wedge r(v',u)$, $\sim_q$ consists of the equivalence classes $\{v, v'\}$ and $\{u\}$. Assume that the chosen representative for $\{v, v'\}$ is v. In this example, q_2 and $\mathsf{Fork}_=$ are the most important ingredients of the rewriting, and we have

  $$q_1^* = \exists u.\neg \mathsf{Aux}(v) \wedge \neg \mathsf{Aux}(v') \wedge r(v,u) \wedge r(v',u) \wedge (\mathsf{Aux}(u) \rightarrow v = v').$$

  In the case of $q_2 = \exists u.r(v,u) \wedge s(v,u)$, the $\mathsf{Fork}_{\neq}$ part of q_1 is the most important ingredient of the rewriting and we have

  $$q_2^* = \exists u.\neg \mathsf{Aux}(v) \wedge \neg \mathsf{Aux}(u) \wedge r(v,u) \wedge s(v,u).$$

  Finally, reconsider $q_3 = \exists u.r(v,u) \wedge s(u,u)$. Here, the Cyc part of the rewriting plays the crucial role, and we have

  $$q_3^* = \exists u.\neg \mathsf{Aux}(v) \wedge \neg \mathsf{Aux}(u) \wedge r(v,u) \wedge s(u,u)$$

- Let q be the query shown in Figure 2, where all variables are quantified. Then $\sim_q$ consists of the equivalence classes $\{v_0, v_1, v_2\}$, $\{v_3, v_4, v_5\}$, $\{v_6\}$, and $\{v_7\}$. Assume that the chosen representative for $\{v_3, v_4, v_5\}$ is v_3. Thus, we have

  $$\begin{aligned} q^* = q \wedge\ & \\ & \mathsf{Aux}(v_6) \rightarrow (v_3 = v_4) \wedge \\ & \mathsf{Aux}(v_7) \rightarrow (v_4 = v_5) \wedge \\ & \mathsf{Aux}(v_3) \rightarrow ((v_0 = v_1) \wedge (v_1 = v_2)) \end{aligned}$$

– Let q_n^c be the query that has no answer variables and whose body is an n-clique, i.e.,

$$q_n^c = \exists v_0, \ldots, v_{n-1}. \bigwedge_{i,j<n} r(v_i, v_j)$$

Then $\sim_q$ consists of a single equivalence class $\{v_0, \ldots, v_{n-1}\}$. Assume that the representative is v_0. Then we have

$$\begin{aligned}(q_n^c)^* = q_n^c \wedge \neg\mathsf{Aux}(v_0) \wedge \cdots \wedge \neg\mathsf{Aux}(v_{n-1}) \wedge \\ \mathsf{Aux}(v_0) \rightarrow ((v_0 = v_1) \wedge \cdots \wedge (v_{n-2} = v_{n-1}))\end{aligned}$$

which can be simplified to the equivalent $q_n^c \wedge \neg\mathsf{Aux}(v_0) \wedge \cdots \wedge \neg\mathsf{Aux}(v_{n-1})$.

As illustrated by the last example, we can drop a conjunct from q_2 whenever the variable occurring in its precondition occurs in a conjunct of q_1.

6 Conclusion

We find it very satisfactory that for the widely-used $\mathcal{EL}$ fragment of OWL2 there is a rather direct way to use relational database systems for sound and complete query answering. The work presented in this paper, however, is only a first step and leaves a lot of room for further investigation. In particular, it would be interesting to carry out experiments to evaluate the feasibility of our approach in practice. To reduce the (polynomial) blowup produced by the data preprocessing phase, one could study a more careful form of preprocessing that materializes only certain concept and role names in the database and takes care of the remaining ones using query rewriting. Another obvious direction is to extend the approach to more expressive variants of $\mathcal{EL}$. For example, we conjecture that adding the bottom concept and role hierarchies is simple. In contrast, transitive roles and left/right identity statements might pose more difficulties. Finally, one could also try to adopt the proposed technique for DL-Lite, thus establishing an alternative approach to query answering for that family of DLs. Since DL-Lite is lacking the finite model property, we cannot store a canonical model in a database (as in the case of $\mathcal{EL}$). However, it is possible that a finite representation of such a model can be used. In that case, the query rewriting step can be expected to depend both on the original query *and on the TBox*.

References

1. http://www.w3.org/2007/OWL/wiki/OWL_Working_Group.
2. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. LTCS-Report LTCS-05-01, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2005. See http://lat.inf.tu-dresden.de/research/reports.html.
3. F. Baader, C. Lutz, and B. Suntisrivaraporn. Tractable reasoning in the $\mathcal{EL}$ family of description logics. *Journal of Logic, Language, and Information*, 2007. To appear.

4. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 364–369. Professional Book Center, 2005.
5. F. Baader, S. Brandt, and C. Lutz. Pushing the el envelope further. In *In Proceedings of OWLED2008*, 2008.
6. D. Calvanese, G. de Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proceedings of KR2006*, pages 260–270. AAAI Press, 2006.
7. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati. Linking data to ontologies: The description logic DL-Lite$_A$. In *Proceedings of OWLED2006*, volume 216 of *CEUR-WS*, 2006.
8. A. Krisnadhi and C. Lutz. Data complexity in the $\mathcal{EL}$ family of DLs. In *Proceedings of DL2007*, volume 250 of *CEUR-WS*, 2007.
9. M. Krötzsch, S. Rudolph, and P. Hitzler. Conjunctive queries for a tractable fragment of OWL 1.1. In *Proceedings of ISWC2007*, volume 4825 of *LNCS*, pages 310–323. Springer, 2007.
10. A.Y. Levy. Obtaining complete answers from incomplete databases. In *In Proceedings of VLDB1996*, pages 402–412, 1996.
11. C. Lutz and F. Wolter. Conservative extensions in the lightweight description logic $\mathcal{EL}$. In *Proceedings of the 21th Conference on Automated Deduction (CADE-21)*, volume 4603 of *LNAI*, pages 84–99. Springer, 2007.
12. C. Patel, J.J. Cimino, J. Dolby, A. Fokoue, A. Kalyanpur, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. Matching patient records to clinical trials using ontologies. In *Proceedings of ISWC2007*, volume 4825 of *LNCS*, pages 816–829. Springer, 2007.
13. J. Patrick and P. Budd. Automatic conversion of clinical notes into snomed ct at point of care. In *Proceedings of HIC2006 and HINZ2006*, pages 209–213, 2006.
14. R. Rosati. On conjunctive query answering in $\mathcal{EL}$. In *Proceedings of DL2007*, volume 250 of *CEUR-WS*, 2007.
15. N. Sioutos, S. de Coronado, M.W. Haber, F.W. Hartel, W.L. Shaiu, and L.W. Wright. NCI thesaurus: a semantic model integrating cancer-related clinical and molecular information. *Journal of Biomedical Informatics*, 40(1):30–43, 2006.
16. K.A. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. *Journal of the American Medical Informatics Association*, 2000.
17. H.J. ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. *Journal of Web Semantics*, 3(2-3):79–115, 2005.
18. Z. Wu, G. Eadon, S. Das, E.I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan. Implementing an inference engine for RDFS/OWL constructs and user-defined rules in oracle. In *Proceedings of ICDE2008*, pages 1239–1248. IEEE, 2008.