Query Rewriting under $\mathcal{EL}$-TBoxes: Efficient Algorithms

Peter Hansen[1], Carsten Lutz[1], İnanç Seylan[1], and Frank Wolter[2]

[1] University of Bremen, Germany, {hansen,clu,seylan}@informatik.uni-bremen.de
[2] University of Liverpool, UK, frank@csc.liv.ac.uk

Abstract. We propose a new type of algorithm for computing first-order (FO) rewritings of concept queries under $\mathcal{EL}$-TBoxes that is tailored towards efficient implementation. The algorithm outputs a non-recursive datalog rewriting if the input is FO-rewritable and otherwise reports non-FO-rewritability. We carry out experiments with ontologies from practical applications which show that our algorithm performs very well in practice, and that $\mathcal{EL}$-TBoxes originating from real-world applications admit FO-rewritings (of reasonable size) in almost all cases, even when in theory such rewritings are not guaranteed to exist.

1 Introduction

Query rewriting is an important technique for implementing ontology-based data access (OBDA) based on a relational database system (RDBMS), thus taking advantage of those systems' efficiency and maturity [15, 9]. The general idea is to transform the original query q and the relevant TBox $\mathcal{T}$ into a first-order (FO) query $q_{\mathcal{T}}$ that is then handed over to the RDBMS for execution. One limitation of this approach is that, for the majority of description logics that are used as ontology languages, the query $q_{\mathcal{T}}$ is not guaranteed to exist. In fact, this is the case already for the members of the popular $\mathcal{EL}$ family of lightweight DLs [2, 11], which underly the OWL2 EL profile and are frequently used as ontology languages in health care and biology. This observation, however, does not rule out the possibility that FO-rewritings still exist in many practically relevant cases. In fact, TBoxes that emerge from practical applications tend to have a rather simple structure and one might thus speculate that, indeed, FO-rewritings under $\mathcal{EL}$-TBoxes tend to exist in practice.

In this paper, we consider the computation of FO-rewritings of concept queries under TBoxes that are formulated in the description logic $\mathcal{EL}$, where a concept query takes the form $C(x)$ with C an $\mathcal{EL}$-concept. It has recently been shown in [6] that deciding whether a concept query $C(x)$ is FO-rewritable under an $\mathcal{EL}$-TBox $\mathcal{T}$ is a PSpace-complete, and that the problem becomes ExpTime-complete when a signature is imposed on the set of admitted database instances (represented as an ABox). This shows that computing the desired rewritings is not an easy task. The existing approaches to query rewriting in $\mathcal{EL}$ and their relevance to the computation of FO-rewritings can be summarized as follows:

(i) approaches that target rewritings in the more expressive query language datalog which are incomplete in the sense that the generated datalog-rewritings are not guaranteed to be non-recursive even if there is an FO-rewriting [16, 14, 13]; (ii) backwards chaining approaches for existential rules (a strict generalization of $\mathcal{EL}$) which are complete in the sense that they find an FO-rewriting if there is one, but need not terminate otherwise [8]; (iii) the complete and terminating approach from [6] which aims at proving upper complexity bounds for FO-rewritings which cannot be expected to be feasible in practice because it relies on brute-force enumeration techniques.[3]

The aim of this paper is *to design algorithms for computing FO-rewritings of concept queries under $\mathcal{EL}$-TBoxes that are complete, terminating, and feasible in practice.* To this end, we start with a marriage of approaches (ii) and (iii) to get the best of both worlds; in particular, (ii) appears to be practically more feasible than (iii) while (iii) provides a way to achieve termination. The resulting algorithm is conceptually simple and constitutes a significant step towards our goal. However, it produces FO-rewritings that are unions of conjunctive queries (UCQs), which results in two significant drawbacks: first, recent experiments [10] have shown that executing UCQ-rewritings on RDBMSs is prohibitively expensive while executing equivalent rewritings that take the form of non-recursive datalog programs is much more feasible (even when the original query is a conjunctive query); and second, UCQ-rewritings can be of excessive size even in practically relevant cases [17].

To address these shortcomings, we refine our original algorithm to what we call a *decomposed algorithm*. While the original algorithm uses tree-shaped conjunctive queries (CQs) as an internal data structure, the new algorithm only represents single nodes of CQs together with information of how to reassemble these nodes into full tree-shaped CQs. This can be seen as a way to implement structure sharing and it also allows us to directly produce rewritings that are non-recursive datalog programs, avoiding UCQ-rewritings altogether. The algorithm runs in exponential time, is capable of deciding the existence of FO-rewritings in ExpTime, and produces monadic non-recursive datalog rewritings that are of at most exponential size (but much smaller in practice). Technically, the decomposed algorithm is much more subtle than the original one.

We then evaluate the decomposed algorithm by carrying out experiments with seven ontologies from practical applications. We ask for an FO-rewriting for every concept query $A(x)$, with A a concept name from the ontology under consideration. Out of 15970 requests in total, the decomposed algorithm times out only on 158 inputs, with a timeout of 15 seconds. We also analyze the size of the generated non-recursive datalog rewritings, with extremely encouraging results. Our experiments show that the decomposed algorithm performs very well on inputs from practical applications. They also confirm our initial belief that $\mathcal{EL}$-ontologies from practical applications often admit FO-rewritings. In particular, only 31 of 15970 queries turn out to not be FO-rewritable.

Proof details can be found in the long version available at:

http://www.informatik.uni-bremen.de/tdki/research/papers/2014/DL2014.pdf

[3] For approaches to FO-rewritability for non-Horn DLs we refer the reader to [7, 4].

2 Preliminaries

We use $\mathsf{N_C}$ and $\mathsf{N_R}$ to denote countably infinite sets of concept names and role names, respectively. An $\mathcal{EL}$-*concept* is formed according to the syntax rule $C ::= A \mid \top \mid C \sqcap C \mid \exists r.C$, a *concept inclusion (CI)* takes the form $C \sqsubseteq D$ with C and D $\mathcal{EL}$-concepts, and a *TBox* is a finite set of CIs. The semantics of concepts and TBoxes is defined as usual. We write $\mathcal{T} \models C \sqsubseteq D$ when C is subsumed by D under $\mathcal{T}$.

An ABox $\mathcal{A}$ is a set of *assertions* of the form $A(a)$ and $r(a, b)$ with A a concept name, r a role name, and a, b individual names from a countably infinite set $\mathsf{N_I}$. We use $\mathsf{ind}(\mathcal{A})$ to denote the set of individual names that occur in $\mathcal{A}$. A *concept query* is an expression $C(x)$ with C an $\mathcal{EL}$-concept and x a variable. We write $\mathcal{A}, \mathcal{T} \models C(a)$ if a is a certain answer to C given the ABox $\mathcal{A}$ and TBox $\mathcal{T}$. For an FO-query $q(x)$ with one free variable, we write $\mathcal{A} \models q(a)$ if $\mathcal{A}$ (viewed as a structure) satisfies q under the assignment that maps x to a.

A concept query $C(x)$ is *FO-rewritable under a TBox* $\mathcal{T}$ if there is an FO-formula $\varphi(x)$ such that for all ABoxes $\mathcal{A}$ and individuals a, we have $\mathcal{A}, \mathcal{T} \models C(a)$ iff $\mathcal{A} \models \varphi(a)$. In this case, we call $\varphi(x)$ an *FO-rewriting* of $C(x)$ under $\mathcal{T}$. An FO-formula $\varphi(x)$ is a *partial FO-rewriting* of A under $\mathcal{T}$ if for all ABoxes $\mathcal{A}$ and $a \in \mathsf{ind}(\mathcal{A})$, $\mathcal{A} \models \varphi(a)$ implies $\mathcal{A}, \mathcal{T} \models A(a)$. Thus a partial FO-rewriting is an FO-rewriting that is sound, but not necessarily complete. When studying FO-rewritability of concept queries $C(x)$, we can assume w.l.o.g. that C is a concept name since $C(x)$ is FO-rewritable under a TBox $\mathcal{T}$ iff $A(x)$ is FO-rewritable under $\mathcal{T} \cup \{C \sqsubseteq A\}$ where A is a fresh concept name [6].

We will also consider more specific forms of FO-rewritings, in particular *UCQ-rewritings* and *non-recursive datalog rewritings*. Although, strictly speaking, non-recursive datalog rewritings are not FO-rewritings, the existence of either kind of rewriting coincides with the existence of an FO-rewriting. In particular, non-recursive datalog rewritings can be viewed as a compact representation of a UCQ-rewriting that implements structure sharing. By a *monadic datalog rewriting*, we mean a datalog rewriting in which all intensional (IDB) predicates are unary.

For our technical constructions, it will be convenient to view $\mathcal{EL}$-concepts as CQs that take the form of a directed tree. We will represent such queries as sets of atoms of the form $A(x)$ and $r(x, y)$ with A a concept name, r a role name and x, y variables, not distinguishing between answer variables and quantified variables. Tree-shapedness of a conjunctive query q then means that the directed graph $(V, \{(x, y) \mid r(x, y) \in q\})$ is a tree, where V is the set of variables in q, and that $r(x, y), s(x, y) \in q$ implies $r = s$. In the following, we will not distinguish explicitly between an $\mathcal{EL}$-concept C and its query representation. We thus use $\mathsf{var}(C)$ to denote the set of variables that occur in C and x_ε to denote the root variable in C. For an $x \in \mathsf{var}(C)$, we use $C|_x$ to denote the $\mathcal{EL}$-concept represented by (the subtree rooted at) x. When we speak of a *top-level conjunct (tlc)* of an $\mathcal{EL}$-concept C, we mean a concept name A such that $A(x_\varepsilon) \in C$ or a concept $\exists r.D$ such that $r(x_\varepsilon, y) \in C$ and $D = C|_y$. We use $\mathsf{tlc}(C)$ to denote the

set of all top-level conjuncts of C. For any syntactic object (such as a concept or a TBox), we define its *size* to be the number of symbols used to write it.

3 A Backwards Chaining Algorithm

Let $\mathcal{T}$ be an $\mathcal{EL}$-TBox and A_0 a concept name for which an FO-rewriting is to be constructed. The algorithm presented in this section constructs a set of partial rewritings of A_0 under $\mathcal{T}$ by starting from $\{A_0\}$ and then exhaustively applying the concept inclusions in $\mathcal{T}$ as rules in a backwards chaining manner. Let C and D be $\mathcal{EL}$-concepts and $\varphi = E \sqsubseteq F$ an $\mathcal{EL}$-concept inclusion. Further, let $x \in \mathsf{var}(C)$ and let there be at least one tlc G of $C|_x$ with $\models F \sqsubseteq G$. Then D is *obtained from C by applying φ at x* if D can be obtained from C by

- removing $A(x)$ for all concept names A with $\models F \sqsubseteq A$;
- removing the subtree rooted at y whenever $r(x, y) \in C$ and $\models F \sqsubseteq \exists r.(C|_y)$;
- adding $A(x)$ for all concept name A that are a tlc of E;
- adding the subtree $\exists r.H$ to x for each $\exists r.H$ that is a tlc of E.

When the exact identity of x is not important, we say that D is *obtained from C by applying φ*. This corresponds to a backwards chaining step based on so-called piece unifiers in [3, 8]. The following is immediate.

Lemma 1. *If $\mathcal{T} \models C \sqsubseteq A_0$ and D can be obtained from C by applying some CI in $\mathcal{T}$, then $\mathcal{T} \models D \sqsubseteq A_0$.*

Apart from applying CIs from the TBox as rules, our algorithm will also minimize the generated partial rewritings to attain completeness and termination. To make this precise, we introduce some notation. For $\mathcal{EL}$-concepts C and D, we write $C \preceq D$ if there is $x \in \mathsf{var}(D)$ such that $C = D \setminus D|_x$, that is the concept C is obtained from D by dropping the subtree $D|_x$ from D. We use $\preceq^*$ to denote the transitive closure of $\preceq$ and say that C *is $\preceq$-minimal with $\mathcal{T} \models C \sqsubseteq A_0$* if $\mathcal{T} \models C \sqsubseteq A_0$ and there is no $C' \preceq C$ with $\mathcal{T} \models C' \sqsubseteq A_0$. Note that if $\mathcal{T} \models C \sqsubseteq A_0$, then it is possible to find in polynomial time a $C' \preceq^* C$ that is minimal with $\mathcal{T} \models C' \sqsubseteq A_0$ (since subsumption in $\mathcal{EL}$ can be decided in PTIME).

The constructions in [6] suggest that, to achieve termination, we can use a certain form of blocking, similar to the blocking used in DL tableau algorithms. Let $\mathsf{sub}(\mathcal{T})$ denote the set of subconcepts of (concepts that occur in) $\mathcal{T}$. For each $\mathcal{EL}$-concept C and $x \in \mathsf{var}(C)$, we set $\mathsf{con}^C_{\mathcal{T}}(x) := \{D \in \mathsf{sub}(\mathcal{T}) \mid \mathcal{T} \models C|_x \sqsubseteq D\}$. We say that C is *blocked* if there are $x_1, x_2, x_3 \in \mathsf{var}(C)$ such that

1. x_1 is an ancestor of x_2, which is an ancestor of x_3 and
2. $\mathsf{con}^C_{\mathcal{T}}(x_1) = \mathsf{con}^C_{\mathcal{T}}(x_2)$ and $\mathsf{con}^{C \setminus C|_{x_3}}_{\mathcal{T}}(x_1) = \mathsf{con}^{C \setminus C|_{x_3}}_{\mathcal{T}}(x_2)$.

The algorithm is formulated in Figure 1. Note that, by Lemma 1, the concept D considered in the condition of the while loop satisfies $\mathcal{T} \models D \sqsubseteq A_0$. We are thus guaranteed to find the desired D' inside the while loop. Also note that there are potentially many different D' that we could use, and each of them will work.

```
procedure find-rewriting(A_0(x), T)
    M := {A_0}
    while there is a C ∈ M and a concept D such that
            1. D can be obtained from C by applying some CI in T and
            2. there is no D' ⪯ D with D' ∈ M then
        find a D' ⪯* D that is minimal with T ⊨ D' ⊑ A_0
        if D' is blocked then
            return 'not FO-rewritable'
        add D' to M
    return the UCQ ⋁ M.
```

Fig. 1. The backwards chaining algorithm

Example 1. Let $\mathcal{T} = \{\exists r.(B_1 \sqcap B_2) \sqsubseteq A_0,\ \exists s.B_2 \sqsubseteq B_2,\ B_1 \sqsubseteq B_2\}$. Starting with $M = \{A_0\}$ and applying the first CI to $C = A_0$, we get $M = \{A_0, \exists r.(B_1 \sqcap B_2)\}$. Applying the second CI to $C = \exists r.(B_1 \sqcap B_2)$ then yields $D = \exists r.(B_1 \sqcap \exists s.B_2)$ which is not minimal with $\mathcal{T} \models D \sqsubseteq A_0$ as witnessed by $D' = \exists r.B_1$, which is added to M. At this point, rule application stops and the UCQ $\bigvee M$ is returned.

It is illustrative to try Example 1 without applying the minimization step. We then find a blocked concept in M, which shows that without minimization the algorithm is incomplete. It can also be seen that dropping minimization results in non-termination since the out-degree of concepts in M can grow unboundedly.

We now establish correctness and termination, showing first that, if the algorithm claims to have found an FO-rewriting, then this is indeed the case.

Proposition 1 (Soundness). *If the algorithm returns $\bigvee M$, then $\bigvee M$ is an FO-rewriting of A_0 under $\mathcal{T}$.*

Proof.(sketch) Let $\mathcal{A}$ be an ABox. We have to show: (1) if $\mathcal{A} \models \bigvee M(a_0)$, then $\mathcal{A}, \mathcal{T} \models A_0(a_0)$; (2) if $\mathcal{A}, \mathcal{T} \models A_0(a_0)$, then $\mathcal{A} \models \bigvee M(a_0)$. For Point 1, assume that $\mathcal{A} \models \bigvee M(a_0)$. Then there is a $C \in M$ with $\mathcal{A} \models C(a_0)$. Consequently $\mathcal{A}, \mathcal{T} \models C(a_0)$. By construction of M, all its elements C satisfy $\mathcal{T} \models C \sqsubseteq A_0$, thus $\mathcal{A}, \mathcal{T} \models A_0(a_0)$ as required.

For Point 2, we essentially follow the proof strategy from [3], based on the chase procedure. If $\mathcal{A}, \mathcal{T} \models A_0(a_0)$, then $A_0(a_0) \in \mathsf{chase}_{\mathcal{T}}(\mathcal{A})$ and consequently, there is a sequence of ABoxes $\mathcal{A} = \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_k$ that *demonstrates* $A_0(a) \in \mathsf{chase}_{\mathcal{T}}(\mathcal{A})$, that is, each $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by a single chase step and $A_0(a) \in \mathcal{A}_k$. It thus suffices to prove by induction on k that if $\mathcal{A} = \mathcal{A}_0, \ldots, \mathcal{A}_k$ is a chase sequence that demonstrates $A_0(a_0) \in \mathsf{chase}_{\mathcal{T}}(\mathcal{A})$, then $\mathcal{A} \models \bigvee M(a_0)$. This is slightly tedious, but straightforward. ❑

Note that the generated UCQ-rewritings are not necessarily of minimal size. It is possible to attain minimal-size rewritings by using a stronger form of minimality when constructing the concept D', namely by redefining the relation "$\preceq$" so that $C \preceq D$ if there is a root-preserving homomorphism from C to D (c.f. the notion of *most-general rewritings* in [8]). As a consequence, the $\preceq$-minimal concept D'

with $\mathcal{T} \models D' \sqsubseteq A$ can then be of size exponential in the size of D. However, D' can still be constructed in output-polynomial time.

Proposition 2 (Completeness). *If the algorithm returns 'not FO-rewritable', then A_0 has no FO-rewriting under $\mathcal{T}$.*

Proof. By Theorem 2 in [5], it suffices to show that if the algorithm returns 'not FO-rewritable', then

($*$) for every $k > 0$, there is a concept C whose depth exceeds k and such that $\mathcal{T} \models C \sqsubseteq A_0$ and $\mathcal{T} \not\models C|_k^- \sqsubseteq A_0$,

where $C|_k^-$ denotes the concept obtained from C be removing all variables whose depth exceeds k. Using a pumping argument, one can show the following sufficient condition for ($*$).

Fact. If there is a concept C that is blocked with variables $x_1, x_2, x_3 \in \mathsf{var}(C)$, $\mathcal{T} \models C \sqsubseteq A_0$, and $\mathcal{T} \not\models C \setminus C|_{x_3} \sqsubseteq A_0$, then ($*$) holds.

Now assume the algorithm returns 'not FO-rewritable'. Then there is a concept D that is minimal with $\mathcal{T} \models D \sqsubseteq A_0$ and that is blocked with variables x_1, x_2, x_3. By minimality of D, $\mathcal{T} \not\models (D \setminus D|_{x_3}) \sqsubseteq A_0$ and ($*$) follows. ❑

We prove in the appendix that the algorithm always terminates by showing that all concepts in M have outdegree at most n and depth at most 2^{2n}, n the size of $\mathcal{T}$. As remarked in [6], the size of UCQ-rewritings can be triple exponential in the size of $\mathcal{T}$, and thus the same is true for the runtime of the presented algorithm. While this worst case is probably not encountered in practice, the size of M can become prohibitively large for realistic inputs. For this reason, we propose an improved algorithm in the subsequent section, which produces non-recursive datalog rewritings instead of UCQ-rewritings and whose runtime is at most single exponential.

4 A Decomposed Algorithm

The algorithm presented in this section consists of three phases. In the first phase, a certain set Γ is computed that can be viewed as a decomposed representation of the set M from Section 3 in the sense that we store only single nodes of the tree-shaped concepts in M, rather than entire concepts. In the second phase, we compute a certain set Ω that enriches the node representation provided by Γ with sets of logical consequences as mentioned in Point 2 of the definition of *blocked* concepts. In the third phase, we first execute a certain cycle check on Ω, which corresponds to checking the existence of a blocked concept in M. If no cycle is found, we can read off a rewriting from Ω.

Assume that $\mathcal{T}$ is a TBox and A_0 a concept name for which we want to compute an FO-rewriting. To present the algorithm, it is convenient to decompose conjunctions on the right-hand side of CIs, that is, to assume that $\mathcal{T}$ consists of CIs of the form $C \sqsubseteq A$, A a concept name, or $C \sqsubseteq \exists r.D$. We start with describing the construction of Γ, whose elements we call node pairs. A *node pair* has

the form (C, S), where $C \in \mathsf{sub}(\mathcal{T})$ and $S \subseteq \mathsf{sub}(\mathcal{T})$ is a set of concept names and concepts of the form $\exists r.C$. Intuitively, a node pair (C, S) describes a set of concepts D (subtrees of concepts in Γ) such that $\mathcal{T} \models D \sqsubseteq C$ and the following conditions are satisfied:

(i) if $A \in S$ for a concept name A, then A is a tlc of D;
(ii) if $\exists r.E \in S$, then there is a tlc $\exists r.E'$ of D such that $\mathcal{T} \models E' \sqsubseteq E$.

The computation of Γ starts with $\{(A_0, \{A_0\})\}$ and proceeds by exhaustively applying the following two rules:

(r1) if $(C, S) \in \Gamma$, $D \sqsubseteq A \in \mathcal{T}$ and $A \in S$, then add pair $(C, (S \setminus \{A\}) \cup \mathsf{tlc}(D))$
(r2) if $(C, S) \in \Gamma$, $D \sqsubseteq \exists r.F \in \mathcal{T}$, and there is an $\exists r.G \in S$ with $\mathcal{T} \models F \sqsubseteq G$, then add the pair $(C, (S \setminus \{\exists r.G \mid \mathcal{T} \models F \sqsubseteq G\}) \cup \mathsf{tlc}(D))$

After applying either rule, we also have to add the pair $(G, \mathsf{tlc}(G))$ for every $\exists r.G \in \mathsf{sub}(D)$ to trigger further derivation.

The set Γ represents a (potentially infinitary) UCQ-rewriting of A_0 under $\mathcal{T}$ in a sense that we make precise now. Let $\widehat{\Gamma}$ be the set obtained as the limit of the sequence of sets $\widehat{\Gamma}_0, \widehat{\Gamma}_1, \ldots$ defined as follows:

- $\widehat{\Gamma}_0 := \{(C, \bigsqcap S) \mid (C, S) \in \Gamma \text{ and } S \subseteq \mathsf{N_C}\}$.
- $\widehat{\Gamma}_{i+1}$ is $\widehat{\Gamma}_i$ extended with all pairs (C, D) such that there are $(C, S) \in \Gamma$ and $(G, C_{r,G}) \in \widehat{\Gamma}_i$ for each $\exists r.G \in S$ and $D = \bigsqcap_{A \in S \cap \mathsf{N_C}} A \sqcap \bigsqcap_{\exists r.G \in S} \exists r.C_{r,G}$.

Proposition 3 (Soundness and Completeness of $\widehat{\Gamma}$). *For all ABoxes $\mathcal{A}$ and $a \in \mathsf{ind}(\mathcal{A})$, we have $\mathcal{A}, \mathcal{T} \models A_0(a)$ iff there is a $(A_0, D) \in \widehat{\Gamma}$ with $\mathcal{A} \models D(a)$.*

Note that Γ provides us with a sufficient condition for FO-rewritability and suggests a way to produce a non-recursive datalog rewriting. In fact, if Γ is *acyclic* in the sense that the directed graph

$$G_\Gamma = (\Gamma, \{((C, S), (C', S')) \mid S \text{ contains a concept } \exists r.C'\})$$

contains no cycle, then $\widehat{\Gamma}$ is finite and we obtain a non-recursive datalog program Π_Γ that is a rewriting of A_0 under $\mathcal{T}$ by taking the rule

$$P_C(x) \leftarrow \bigwedge_{A \in S} A(x) \wedge \bigwedge_{\exists r.D \in S} (\, r(x, y_{r,D}) \wedge P_D(y_{r,D}) \,)$$

for each $(C, S) \in \Gamma$ and using A_0 as the goal predicate. However, if Γ is not acyclic, then A_0 could still be FO-rewritable under $\mathcal{T}$, but the above program will be recursive. To deal with this problem, we need the next two phases.

We construct a set of node tuples Ω from Γ by further annotating and duplicating the pairs in Γ. A *node tuple* takes the form $t = (C_t, S_t, \mathsf{con}_t, s_t, \mathsf{xcon}_t)$ where C_t and S_t have the same form as the components of node pairs in Γ, $\mathsf{con}_t \subseteq \mathsf{sub}(\mathcal{T})$, s_t is an existential restriction in S_t or the special symbol "$-$", and xcon_t is either a subset of $\mathsf{sub}(\mathcal{T})$ or the special symbol "$-$". Intuitively, a node tuple $t \in \Omega$ describes a set of concepts D (subtrees of concepts in Γ) such that (C_t, S_t) describes D in the way described above and the following additional conditions are satisfied:

(iii) for each $E \in \mathsf{sub}(\mathcal{T})$, we have $\mathcal{T} \models D \sqsubseteq E$ iff $E \in \mathsf{con}_t$;
(iv) in the subtree of D rooted at s_t there is a leaf node such that for the concept D' obtained by dropped this node and each $E \in \mathsf{sub}(\mathcal{T})$, we have $\mathcal{T} \models D' \sqsubseteq E$ iff $E \in \mathsf{xcon}_t$.

When S_t contains no existential restrictions, we use "$-$" in the last two components. To understand s_t, it is useful to think of D as a tree and of s_t as a selected successor of the root of that tree. We start the construction of Ω with

$$\Omega_0 := \{(C, S, \mathsf{con}_\mathcal{T}(S), -, -) \mid (C, S) \in \Gamma \text{ with } S \subseteq \mathsf{N_C}\},$$

where for a set of concepts M, $\mathsf{con}_\mathcal{T}(M)$ denotes the set of concepts $D \in \mathsf{sub}(\mathcal{T})$ such that $\mathcal{T} \models \bigsqcap M \sqsubseteq D$. We call the tuples in Ω_0 *leaf tuples*. The final set Ω is constructed by exhaustively applying the following rule:

(r3) If t is a node tuple such that $\exists r_0.D_0, \dots, \exists r_n.D_n$ are the existential restrictions in S_t, $s_t = \exists r_\ell.D_\ell$, and $t_0, \dots, t_n \in \Omega$ with $C_{t_i} = D_i$ for $0 \leq i \leq n$, then add t to Ω if the following conditions are satisfied:
- there is a node pair $(C_t, S) \in \Gamma$ with $S_t \subseteq S$ and $S \cap \mathsf{N_C} = S_t \cap \mathsf{N_C}$;
- $\mathsf{con}_t = \mathsf{con}_\mathcal{T}(M)$, where $M = \bigcup(S_t \cap \mathsf{N_C}) \cup \{\exists r_i.G \mid i \leq n \text{ and } G \in \mathsf{con}_{t_i}\}$;
- $\mathsf{xcon}_t = \mathsf{con}_\mathcal{T}(M')$, where M' is

$$\bigcup(S_t \cap \mathsf{N_C}) \cup \{\exists r_\ell.G \mid G \in \mathsf{xcon}_{t_\ell}\} \cup \{\exists r_i.G \mid \ell \neq i \leq n \text{ and } G \in \mathsf{con}_{t_i}\}.$$

In Points 3, the concept $\exists r.-$ (which might occur in the set M') is identified with $\top$. For $t, t' \in \Omega$, we write $t \rightsquigarrow_\Omega t'$ if there are $t_0, \dots, t_n \in \Omega$ that satisfy the conditions listed in (r3) and such that $t' = t_\ell$, that is, t' is the tuple that was chosen for the selected successor. Note that the computation of the sets con_t and xcon_t is complete, relying on the following observation [12].

Lemma 2. *For any concept $C = A_1 \sqcap \dots \sqcap A_n \sqcap \exists r_1.G_1 \sqcap \dots \sqcap \exists r_m.G_m$,*

$$\mathsf{con}_\mathcal{T}(C) = \mathsf{con}_\mathcal{T}(\{A_1, \dots, A_n\} \cup \bigcup_{1 \leq i \leq m} \{\exists r_i.D \mid D \in \mathsf{con}_\mathcal{T}(G_i)\})$$

We now describe the third and last phase of the algorithm, which first checks whether an FO-rewriting exists at all and, if so, produces a rewriting that takes the form of a non-recursive monadic datalog program.

We start with introducing the relevant notion of a cycle. A tuple $t \in \Omega$ is a *root tuple* if $A_0 \in \mathsf{con}_t$ and $A_0 \notin \mathsf{xcon}_t$. A *path through* Ω is a finite sequence of node tuples $t_1, \dots, t_k$ from Ω such that $t_i \rightsquigarrow_\Omega t_{i+1}$ for $1 \leq i < k$. A tuple $t \in \Omega$ is *looping* if there is a path $t_1, \dots, t_k$ through Ω of length at least one such that $t = t_1$, $\mathsf{con}_t = \mathsf{con}_{t_k}$, and $\mathsf{xcon}_t = \mathsf{xcon}_{t_k}$. We say that Ω *contains a root cycle* if there are tuples $t, t' \in \Omega$ such that t is a root tuple, t' is a looping tuple, and t' is reachable from t along $\rightsquigarrow_\Omega$.

Proposition 4. *A_0 is not FO-rewritable under $\mathcal{T}$ if and only if Ω contains a root cycle.*

Proof.(sketch) "if". Assume that Ω contains a root cycle. Using this cycle as a guide, we show how to find a concept C that is blocked in the sense of Section 3 and satisfies $\mathcal{T} \models C \sqsubseteq A_0$ as well as $\mathcal{T} \not\models (C \setminus C|_{x_3}) \sqsubseteq A_0$, where x_3 is as in the definition of 'blocked'. Once we have constructed such a concept C, we can again rely on the results of [5], as in the proof of Proposition 2.

"only if". Assuming that Ω contains no root cycle, we show below how to constuct a non-recursive datalog-rewriting of A_0 under $\mathcal{T}$, thus A_0 is FO-rewritable under $\mathcal{T}$. ❑

As suggested by Proposition 4, our algorithm first checks whether Ω contains a root cycle and, if so, returns 'not FO-rewritable'. Otherwise, it constructs a non-recursive datalog program $\Pi_{\mathcal{T},A_0}$ as follows. First, we drop from Ω all tuples that are not reachable from a root tuple along an $\rightsquigarrow_\Omega$-path. For $t, t' \in \Omega$ and $\exists r.D \in S_t$, we write $t \rightsquigarrow_{\exists r.D} t'$ if there is a tuple $\widehat{t} = (C_t, S_t, \mathsf{con}_t, \exists r.D, \mathsf{xcon}_{\widehat{t}}) \in \Omega$ such that $\widehat{t} \rightsquigarrow_\Omega t'$. Note that, by definition of "$\rightsquigarrow_\Omega$", $t \rightsquigarrow_{\exists r.D} t'$ implies $C_{t'} = D$. Now, $\Pi_{\mathcal{T},A_0}$ contains for every $t \in \Omega$, the rule

$$P_{C_t,\mathsf{con}_t}(x) \leftarrow \bigwedge_{A \in S_t \cap \mathsf{N_C}} A(x) \wedge \bigwedge_{\exists r.D \in S_t} (r(x, y_{r,D}) \wedge \bigvee_{t' \in \Omega | t \rightsquigarrow_{\exists r.D} t'} P_{D,\mathsf{con}_{t'}}(y_{r,D}))$$

Note that the disjunctions can be removed by introducing auxiliary IDB predicates, without causing a significant blowup. The goal predicates of $\Pi_{\mathcal{T},A_0}$ are all predicates of the form $P_{A_0,\mathsf{con}}(x)$ with $A_0 \in \mathsf{con}$.

Theorem 1.
1. *The program $\Pi_{\mathcal{T},A_0}$ is a rewriting of A_0 under $\mathcal{T}$.*
2. *If Ω contains no root cycle, then $\Pi_{\mathcal{T},A_0}$ is non-recursive.*

Even if Ω contains no root-cycles, the program $\Pi_{\mathcal{T},A_0}$ may have up to (single) exponentially many IDB predicates. We observe that this cannot be significantly improved without giving up monadicity.

Theorem 2. *There is a family of TBoxes $\mathcal{T}_1, \mathcal{T}_2, \ldots$ such that for all $n \geq 1$, $\mathcal{T}_n$ is of size $\mathcal{O}(n^2)$, the concept name A_0 is FO-rewritable under $\mathcal{T}_n$, and the smallest non-recursive monadic datalog rewriting has size at least 2^n.*

Let us briefly analyze the complexity of the decomposed algorithm. It is easy to verify that the number of Γ-pairs and Ω-tuples is singly exponential in the size of $\mathcal{T}$ and that all required operations for building Γ and Ω and for determining the existence of a root cycle require only polynomial time. By Proposition 4, we have thus found an ExpTime algorithm for deciding FO-rewritability of $\mathcal{EL}$-concept queries. This is almost optimal as the problem we are dealing with is PSpace-complete and becomes ExpTime-hard if slightly varied, see [6].

5 Experiments

We have implemented the decomposed algorithm in Java and conducted a number of experiments. The implementation is not highly optimized, but some aspects of handling the set Γ are worth to point out. In particular, we use numbers

TBox	concept inclusions	concept names	role names	timeouts	not FO-rewritable
XP	1046	906	27	0	1
NBO	1468	962	8	0	6
ENVO	1848	1538	7	0	7
FBbi	567	517	1	0	0
MOHSE	3665	2203	71	2	1
not-galen	4636	2748	159	149	0
SO	2740	1894	13	7	16

Table 1. TBoxes used in the experiments.

to represent subconcepts in $\mathcal{T}$, store the S-component of each pair $(C, S) \in \Gamma$, which is a set of subconcepts of $\mathcal{T}$, as an ordered set, and use so-called *tries* as a data structure to store Γ. We remove pairs $(C, S) \in \Gamma$ where S is not minimal, that is, for which there is a $(C, S') \in \Gamma$ with $S' \subsetneq S$. It is easy to see that this optimization does not compromise correctness.

The experiments were carried out on a Linux (3.2.0) machine with a 3.5Ghz quad-core processor and 8GB of RAM. Although a large number of $\mathcal{EL}$-TBoxes is available on the web and from various repositories, most of them are *acyclic TBoxes* in the traditional DL sense, that is, the left-hand sides of all CIs are concept names, there are no two CIs with the same left-hand side, and there are no syntactic cycles. Since concept queries are always FO-rewritable under acyclic $\mathcal{EL}$-TBoxes [5], such TBoxes are not useful for our experiments. We have identified seven TBoxes that do not fall into this class, listed in Table 1 together with the number of concept inclusions, concept names, and role names that they contain. All TBoxes together with information about their origin are available at http://tinyurl.com/q96q34z.

For each of these TBoxes, we have applied the decomposed algorithm to every concept name in the TBox. In some rare cases, the set Γ has reached excessive size, resulting in non-termination. We have thus established a 15 second timeout for the Γ-phase of the algorithm. With that timeout, our algorithm was able to decide FO-rewritability in almost all of the cases, see Table 1. The overall runtime for all our experiments as a batch job (15970 invocations of the algorithm) took only 80 minutes. The generated non-recursive datalog-rewritings are typically of very reasonable size. The number of rules in the rewriting is displayed in the upper part of Figure 2; for example, for NBO, about 55% of all rewritings consist of a single rule, about 18% have two or three rules, about 10% have 4–7 rules, and so on. Note that the x-axis has logarithmic scale. The size of the rule bodies is typically very small, between one and two atoms in the vast majority of cases, and we have never encountered a rule with more than ten body atoms. It is also interesting to consider the number of IDB predicates in a rewriting, as intuitively these correspond to views that have to be generated by a database system that executes the query. As shown in the lower part of Figure 2, the number of IDB predicates is rather small, and considerably lower than the number of rules in the produced programs (we again use logarithmic scale on the x-axis).

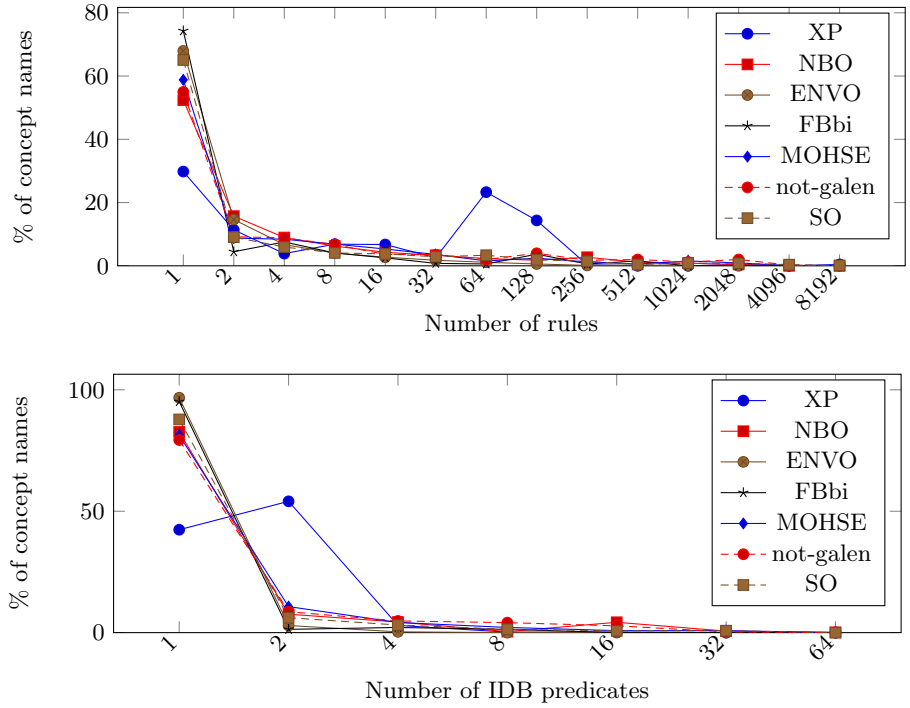

Fig. 2. Number of rules and IDB predicates in the rewriting

The experiments also confirm our initial belief that ontologies which are used in practical applications have a simple structure. As shown in Table 1, the number of concept names that are not FO-rewritable is extremely small. Moreover, if a concept name was FO-rewritable, then we were always able to determine this already in the Γ-phase of our algorithm, without ever entering the Ω-phase. Note, though, that for those cases that turned out to be not FO-rewritable, we had to go through the full Ω-construction.

6 Outlook

We plan to optimize the implementation of the decomposed approach further to eliminate the timeouts we encountered in the experiments. Moreover, we plan to extend the algorithm and implementation in several ways. First, we plan to generalize the approach from concept queries to conjunctive queries. Second, in many applications the ABox signature (i.e., the concept and role names occurring in the ABoxes) is a small subset of the signature of the TBox [1, 6]. Queries that are not FO-rewritable without any restriction on the ABox signature can become FO-rewritable under smaller ABox signatures. We therefore plan to extend the decomposed approach to arbitrary ABox signatures. Finally, we plan to extend our approach to more expressive Horn-DLs such as $\mathcal{ELI}$ with role inclusions and thereby unify DL-Lite and $\mathcal{EL}$ query rewriting approaches in one framework.

References

1. Baader, F., Bienvenu, M., Lutz, C., Wolter, F.: Query and predicate emptiness in description logics. In: KR. pp. 192–202 (2010)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ Envelope. In: IJCAI. pp. 364–369 (2005)
3. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artif. Intell. 175(9-10), 1620–1654 (2011)
4. Bienvenu, M., ten Cate, B., Lutz, C., Wolter, F.: Ontology-based data access: a study through Disjunctive Datalog, CSP, and MMSNP. In: PODS. pp. 213–224 (2013)
5. Bienvenu, M., Lutz, C., Wolter, F.: Deciding FO-rewritability in $\mathcal{EL}$. In: Description Logics. pp. 70–80 (2012)
6. Bienvenu, M., Lutz, C., Wolter, F.: First order-rewritability of atomic queries in horn description logics. In: IJCAI. pp. 754–760 (2013)
7. Kaminski, M., Grau, B.C.: Sufficient conditions for first-order and datalog rewritability in $\mathcal{ELU}$. In: Description Logics. pp. 271–293 (2013)
8. König, M., Leclère, M., Mugnier, M.L., Thomazo, M.: A sound and complete backward chaining algorithm for existential rules. In: RR. pp. 122–138 (2012)
9. Kontchakov, R., Rodriguez-Muro, M., Zakharyaschev, M.: Ontology-based data access with databases: A short course. In: Reasoning Web. pp. 194–229 (2013)
10. Lutz, C., İnanç Seylan, Toman, D., Wolter, F.: The combined approach to OBDA: Taming role hierarchies using filters. In: ISWC. pp. 314–330 (2013)
11. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in the description logic el using a relational database system. In: IJCAI. pp. 2070–2075 (2009)
12. Lutz, C., Wolter, F.: Deciding inseparability and conservative extensions in the description logic $\mathcal{EL}$. J. Symb. Comput. pp. 194–228 (2010)
13. Mora, J., Corcho, Ó.: Engineering optimisations in query rewriting for OBDA. In: I-SEMANTICS. pp. 41–48 (2013)
14. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. J. Applied Logic 8(2), 186–209 (2010)
15. Poggi, A., Lembo, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. Data Semantics 10, 133–173 (2008)
16. Rosati, R.: On conjunctive query answering in $\mathcal{EL}$. In: Description Logics. pp. 451–458 (2007)
17. Rosati, R., Almatelli, A.: Improving query answering over DL-Lite ontologies. In: KR. pp. 290–300 (2010)