Summary: Efficient Query Rewriting in the Description Logic EL and Beyond

Topic 7: Query Rewriting on OBDA - No. 7

s1856331: Muthuri Mwenda - April 2019

Abstract

This paper describes a design for complete, terminating and feasible algorithms that compute FO-rewritings of concept queries under description logics EL and ELH^{dr}'s terminology boxes. It proposes an algorithm for computing first-order (FO) rewritings of concept queries which outputs non-recursive datalog rewritings if the input is FO-rewritable, otherwise returns a non-FO-rewritability flag. Experiments are conducted with real-world ontologies and the results demonstrate feasibility of the algorithm in practice.

1 Introduction

An ontology is a high-level global schema that provides a vocabulary for user queries thus isolating the user from details of the structure of data sources. Data sources can be relational databases, triple stores, datalog engines etc. [13]. Accordingly, Ontology Based Data Access (OBDA) describes the use of knowledge representation and reasoning techniques to interpret data from multiple sources through explicit definition of terms and relationships in the ontology. An OBDA system transforms user queries into the vocabulary of the data by providing a shared conceptualization of the domain of interest and passing on query evaluation to the actual data sources. Owing to this notion, Description Logics (DL) are logical representation systems for an ontology [3]. Baader [3] denotes that a DL knowledgebase comprises of a TBox and an ABox. A TBox contains intensional knowledge in the form of terminology and is built through declarations that describe the general properties of concepts. An ABox is a set of assertions and contains extensional (or assertional) knowledge which is specific to the individuals of the domain of discourse.

Several approaches have been considered to implement OBDA with description logics. This paper explores one such approach: query rewriting. Query rewriting involves transforming the original query q and the relevant TBox T into a first-order (FO) query q_T that is delegated on to a relational database management system (RDBMS) for execution. The reason of rewriting is to capitalize on RDBMS' efficiency and maturity because these logics allow for answering complex queries in LOGSPACE with respect to data complexity [12, 8]. The significant limitation of query rewriting is that for majority of description logics under use as ontology languages, the query q_T is not guaranteed to exist. This limitation affects the EL family of description logics as described by [2, 10]. In spite of this observation, FO-rewrittings exist in many practical and relevant cases. This paper shows that TBoxes that emerge from practical applications tend to assume a relatively simple structure and in most cases, EL-TBoxes tend to exist in practice.

We explore the construction of FO-rewrittings of concept queries under TBoxes that are formulated in the description logic ELH^{dr}. ELH^{dr} extends EL with role inclusions and domain/range restrictions on roles as denoted by Lutz et al. [10]. For ease of construction, the formulated algorithm uses TBoxes without domain and range restrictions since Hansen et al. [1] prove that we can construct an ELH TBox without range and domain restrictions whose FO-rewritability of a concept query guarantees its FO-rewritability under a TBox with the restrictions.
Bienvenu et al. [5] demonstrates that deciding FO-rewritability of a concept query under a given TBox T is PSPACE-complete both in EL and in ELH^{dr} and escalates to EXPTIME-complete with the additional input of ABoxes. Current approaches to rewrite queries are summarized as:

i Rewritings into a more expressive query language datalog. Since datalog rewritings are not guaranteed, these rewritings are not guaranteed to be found. Additionally, the generated rewritings are not always non-recursive even with the existence of a FO-rewriting [11].

ii Backwards chaining for existential rules which are complete; they find a FO-rewriting if there is one but need not terminate otherwise [7].

iii Complete and terminating approaches aiming to prove upper complexity bounds but are not practically feasible [5].

Since (ii) appears more feasible than (iii) and (iii) provides a way to terminate, their combination results in a backwards chaining algorithm that produces unions of conjunctive queries (UCQs) with two drawbacks: (*i*) UCQ rewritings on RDBMSs are excessively large while executing equivalent rewritings to take the form of non-recursive datalog programs [9] and (*ii*) UCQ rewritings can be of excessive size in practical cases [14].

The outline of this summary proceeds as follows; Section 2 defines the notation to be used in the subsequent sequel. The Background, Section 3, introduces the backward chaining concept, a backward chaining algorithm for FO-rewritings and highlights the targeted bottleneck for which the succeeding Section 4 proposes an improved algorithm [1]. Section 5 gives an illustration of and deductions from the experiments of the refined algorithm.

2 Preliminaries

N_C and NR denotes mutually disjoint countably infinite sets of concept and role names respectively. An EL-concept is formed according to the syntax rule $C ::= A \mid \top \mid C \sqcap C \mid \exists r.C$, where $A \in N_C$ and $r \in N_R$. Let C, D be EL-concepts and r, s be role names. $C \sqsubseteq D$ is a concept inclusion (CI), $r \sqsubseteq s$ is a role inclusion RI, $\mathrm{dom}(r) \sqsubseteq C$ is a domain restriction and $\mathrm{ran}(r) \sqsubseteq C$ is a range restriction. An $ELH-TBox$ is a finite set of CIs and RIs while an ELH^{dr}-TBox includes domain and range restrictions. For a CI or RI α, $T \models \alpha$ is written for every model of T that satisfies α. When T is empty, we write $\models \alpha$.

An ABox A is a set of assertions of the form $A(a)$ and $r(a, b)$ with A being a concept name, r a role name and a, b being individual names from countably infinite set N_I. $\mathrm{Ind}(A)$ denotes the set of individual names that occur in A. Accordingly, a *concept query* is an expression $C(x)$ with C being an EL-concept. Given an ABox A and a TBox T, we write $A, T \models C(a)$ to denote that a is a certain answer if $a \in \mathrm{ind}(A)$ and $a^I \in C^I$ for all models I of T and A. A *signature* is a finite set $\Sigma \subseteq N_C \cup N_R$. An ABox is a Σ-ABox if it only uses concept and role names from Σ.

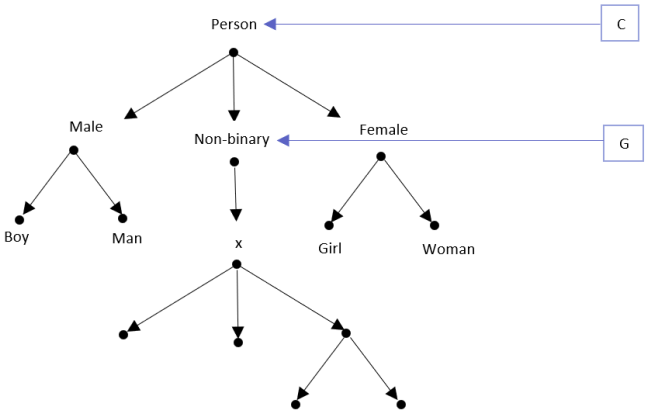


Figure 1: An illustration of EL-concepts as a directed tree

For a FO-formula $q(x)$ with one free variable x we write $A \models q(a)$ if A satisfies q under the mapping of x to a. A concept query $C(x)$ is FO-rewritable under a TBox T and a signature Σ if there us a FO-formula $\varphi(x)$ such that for all Σ-ABoxes A and individuals a, a is a certain answer iff $A \models \varphi(a)$. In this case, $\varphi(x)$ is a FO-rewriting of $C(x)$ under T and Σ.

Since the paper considers rewritings into unions of conjunctive queries (UCQs) and non-recursive datalog programs, we recall from [4] that non-recursive datalog rewritings can be viewed as a compact representation of UCQ-rewriting. Further, the paper considers monadic datalog where all intensional (IDB) predicates are unary. For all these rewritings, if Σ is the set of all concept and role names in T and C,

[1] The latter algorithm has the runtime at most single exponential, a substantial improvement from the former's triple exponential

then $C(x)$ is rewritable under T and Σ iff it is rewritable under T any signature. In this case, $C(x)$ is said to be rewritable under T and the *full signature*.

By viewing EL-concepts as conjunctive queries (CQs) that take the form of a directed tree, Figure 1, we present them 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 being variables. For a conjunctive query q, tree-shapedness means that the directed graph $V, \{(x, y) \mid r(x, y) \in q\}$; where V is the set of variables in q, and that $r(x, y),\ s(x, y) \in q$ implies $r = s$. To denote the set of variables that occur in a concept C we use $\text{var}(C)$ while $C|_x$ denotes the EL-concept represented by the subtree rooted at x for an $x \in \text{var}(C)$. The root variable in C is indicated as x_ε.
A top-level conjunct (tlc) of an EL-concept C is a concept name or a concept that is the root variable of C. Precisely, for a concept A, its tlc is denoted as $A(x_\varepsilon)$. To indicate the set of all top-level conjuncts of C, we use $\text{tlc}(C)$. Finally, for a syntactic object like a TBox, its size refers to the number of symbols used to write it.

3 Background: The Backward Chaining Algorithm

Backward chaining is a logical process of inferring unknown truths from known conclusions by traversing backwards from the solutions and establishing the preceding conditions and rules [6]. The algorithm presented in this section constructs a set of UCQ-rewritings of a concept query $A_0(x)$ under an ELH-TBox T by starting from the set $\{A_0\}$ and exhaustively applying axioms in T as rules in a backward chaining manner. It terminates by either constructing a UCQ-rewriting of the concept query or returning a 'not FO-rewritable' flag. the steps necessary for the algorithm; rewriting and minimization.

3.1 Rewriting

Let A_0 and T be an input to the algorithm. Central backwards chaining steps is defined: Let C and D be EL-concepts and let α be a concept or role inclusion from T. We can obtain D from C by applying α as follows:

3.1.1 For Concept Inclusions

Let $\alpha = E \sqsubseteq F$ be a CI, $x \in \text{var}(C)$ and let there be at least one tlc G of $C|_x$ with $\models F \sqsubseteq G$ as illustrated by the pointed arrows of Figure 1. D is obtained from C by applying α at x if D can be obtained from C by:

- Removing concept queries $A(x)$ for all concept names A not satisfied by every model of T;
- Removing the subtree C_y whenever a role $r(x, y) \in C$ exists and the concept inclusion $\models F \sqsubseteq \exists r.(C|_y)$ including the edge $r(x, y)$;
- Adding concept queries $A(x)$ for all concept names A that are tcl of E;
- Adding a subtree $\exists r.H$ to x for each role r that is a tlc of E.

3.1.2 For Role Inclusions

Let $\alpha = r \sqsubseteq s$ be a RI and let $s(x, y) \in C$. D can be obtained from C by applying α at $s(x, y)$ if D can be obtained from C by replacing $s(x, y)$ by $r(x, y)$.
Lemma 1. *If $T \models C \sqsubseteq A_0$ and D can be obtained from C by applying some inclusion in T, then $T \models D \sqsubseteq A_0$.*
An important implementation for the algorithm to attain completeness, it needs minimization.

3.2 Minimization

Let C and D be EL-concepts and $x \in \text{var}(C)$. To denote the concept obtained by removing from C the subtree rooted at x we write $C \backslash C|_x$. We write $C \prec D$ id there exists $x \in \text{var}D$ such that $C = D \backslash D|_x$. C is $\prec$-minimal with $T \models C \sqsubseteq A_0$ if $T \models C \sqsubseteq A_0$ and there is no $C' \prec C$ with $T \models C' \sqsubseteq A_0$. $\prec^*$ denotes the reflexive and transitive closure of $\prec$.
To achieve termination, [5] suggests the use of blocking analogous to that in description logics. Blocking is explained below.

3.3 Blocking

Let sub(T) denote the set of subconcepts of (concepts that occur in) T. For each EL-concept C and $x \in \text{var}(C)$, we set $\text{con}_T^C(x) := \{D \in \text{sub}(T) \mid T \models C|_x \sqsubseteq D\}$. C is said to be blocked if there are $x_1, x_2, x_3 \in \text{var}(C)$ such that:

1. x_1 is a proper ancestor of x_2 is an ancestor of x_3 and
2. $\text{con}_T^D(x_1) = con_T^D(x_2)$ for $D \in \{C, C \backslash C|_{x_3}\}$.

Figure 2 highlights the backward chaining algorithm.

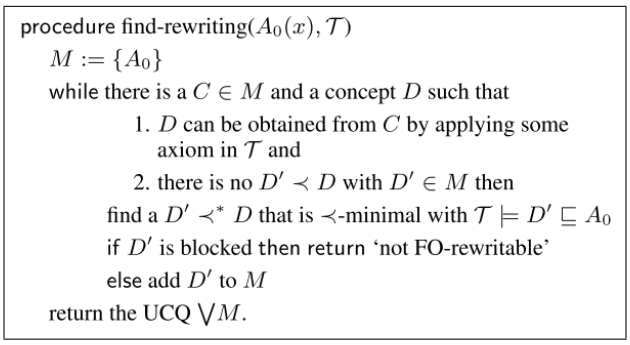


Figure 2: Pseudocode for the backward chaining algorithm

By using a more stringent form of minimality when constructing concept D' by redefining $\prec$ so that $C \preceq D$ if there is a root preserving homomorphism from C to D, it is possible to attain minimal-size rewritings [7]. This implies that in spite of this algorithm implementing the minimization step, the generated UCQ-rewritings are not guaranteed to be minimal. Consequentially, the $\prec$-minimal concept D' can be of exponential in the size of D.
Bienvenu et al. [5] remarks that the size of UCQ-rewritings can be triple exponential in the size of T. The same holds for the runtime presented in Figure 2. It follows that even for realistic inputs, the size of M can be too large. This prompts for the improved algorithm introduced in Section 4 whose runtime is at most single exponential.

4 A Decomposed Algorithm

The algorithm presented in this section generates non-recursive datalog rewritings instead of UCQ-rewritings as in the preamble. Although non-recursive datalog programs are not technically FO-rewritings, Hansen et al. [1] prove that every UCQ and non-recursive datalog rewriting of a concept query $A_0(x)$ under T' and Σ' can be converted in linear time into a UCQ and, respectively, non-recursive datalog rewritings of A_0 under T and Σ. Non-recursive datalog rewritings can be viewed as a compact representation of a UCQ rewriting.
This algorithm consists of three phases:

1. Phase 1: A set Γ is computed and can be viewed as a decomposed representation of the set M from Figure 2. Rather than storing the entire concepts in M, only a single node of the tree-shaped concepts is stored.
2. Phase 2: If construction of a non-recursive datalog rewriting is not possible in Phase 1, compute a set Ω that enriches the node representation provided by Γ with sets of logical consequences that are relevant for point 2 of the definition of blocked concepts.
3. Phase 3: Execute a cycle check on Ω which corresponds to checking the existence of blocked concepts in M. If no cycle is found, we construct a non-recursive datalog rewriting.

They are discussed in more detail.

Phase 1. Assume that T is a TBox, Σ an ABox-signature and A_0 a concept name for which we want to compute a FO-rewriting under T and Σ. The algorithm is designed for normalized existential rules, where only one atom appears in the right-hand side of the CI.
We start by describing the construction of a set Γ whose elements are called node pairs. A node pair has the form (C, S) where $C \in \text{sub}(T)$, and $S \subseteq \text{sub}(T)$ is a set of concept names and concepts of the form $\exists r.C$. A node pair describes the set of concepts D such that $T \models D \sqsubseteq C$ (S is the set of subconcepts in T required to obtain D from C) and the following conditions hold:

i the concept names that are tlcs of D are $S \cap N_C$;
ii the existential restrictions that are the tlcs of D are obtained from the existential restrictions in S by replacing each $\exists r.E \in S$ with some $\exists s.E'$ such that $T \models \exists s.E' \sqsubseteq \exists r.E$[2].

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

(Rule 1:) if $D \in \Gamma, A \in T$ and $A \in S$, then extend Γ with the node pair $(C, (S \backslash \{A\}) \cup \text{tlc}(D))$.

(Rule 2:) if $D \in T,\ D \sqsubseteq \exists r.F \in T$, and there is an $\exists s.G \in S$ with $T \models F \sqsubseteq G$ and $T \models r \sqsubseteq s$ then extend Γ with the node pair$(C, (S \backslash \{\exists s.G \mid T \models F \sqsubseteq G \text{ and } T \models r \sqsubseteq s\}) \cup \text{tlc}(D))$.

After applying either rule, the pair (G,tlc(G)) has to be added for every subconcept $\exists r.G$ of D to trigger further derivation.
Beginning with Γ it is possible to derive a potentially infinitary UCQ-rewriting of a concept A_0 under T and Σ. Take Γ_Σ to be the set of all node pairs $(C, S) \in T$ such that $S \cap N_C \subseteq \Sigma$. In some instances, Γ_Σ provides sufficient condition for FO-rewritability of A_0 under T and Σ and might suggests a way to produce a non-recursive datalog rewriting. If Γ_Σ is acyclic in that the directed graph contains no cycle, we can obtain a non-recursive datalog program that is a rewriting of A_0 under T and Σ. The generated rewriting is potentially significantly smaller than a UCQ-rewriting. If Γ_Σ is not acyclic, then A_0 could be rewritable under T and Σ but the generated datalog program will be recursive.

Phase 2. Construct a set of node tuples Ω_Σ by further annotating (and duplicating) the pairs of Γ_Σ. A node tuple is in the form $t = (C_t, S_t, \text{con}_t, \text{xcon}_t)$ where C_t and S_t have the form as the components of the node pair in Γ_Σ, $C_t \in \text{con}_t \subseteq \text{sub}(T)$, E_t is the special symbol "$-$" or of the form $\exists s.C$ such that $\exists r.C \in S_t$ for some r with $T \models s \sqsubseteq r$, and xcon_t is a subset of con_t or "$-$".
When S_t contains no existential restrictions, we use "$-$" for E_t and xcon_t. Intuitively, E_t can be thought of as a selected successor of the root of the tree D. The construction of Ω_Σ with the setting
$\Omega_\Sigma = \{(C, S \cap N_C,\ \text{con}_T(S \cap N_C), -, -) \mid (C, S) \in \Gamma_\Sigma\}$,
where the set of concepts M, $\text{con}_T(M)$ denotes the set of all concepts $D \in \text{sub}(T)$ such that $T \models \sqcap M \sqsubseteq D$. Tuples in the set of above tuples are called leaf tuples. The final set Ω_Σ is constructed by exhaustively applying the following rule:
(r_n) If $t = (C_t, S_t, \text{con}_t, E_t, \text{xcon}_t)$ is a node tuple with $\exists r_0.D_0, ..., \exists r_n.D_n$ the existential restrictions in $S_t (n \geq 0)$ and there are role names $s_0, ..., s_n \in \Sigma$ and node tuples $t_0, ...t_n \in \Omega_\Sigma$ and an $l \in \{0, ..., n\}$ such that the following conditions hold:

1. $T \models s_i \sqsubseteq r_i$ and $C_{ti} = D_i$ for $0 \leq i \leq n$;
2. $E_t = \exists s_l.D_l$;
3. there is a node pair $(C_t, S) \in \Gamma_\Sigma$ with $S_t \subseteq S$ and $S \cap N_C = S_t \cap N_C$;
4. $\text{con}_t = \text{con}_T(M)$, where $M = (S_t \cap N_C) \cup \{\exists s_i. \sqcap \text{con}_{t_i} \mid i \leq n\}$;
5. $\text{xcon}_t = \text{con}_T(M')$, where $M' = (S_t \cap N_C) \cup \{\exists s_l. \sqcap \text{xcon}_{t_l}\} \cup \{\exists s_i. \sqcap \text{con}_{t_i} \mid l \neq i \leq n\}$.

In point 5, $\exists r.-$ is identified with $\top$. For $t, t' \in \Omega_\Sigma$, we write $t \rightsquigarrow_{\Omega_\Sigma} t'$ if there are $t_0, ...t_n \in \Omega_\Sigma$ that satisfy the condition listen in (r_Ω) and such that $t' = t_l$ meaning that t' was the tuple that was chosen for the selected successor.

Phase 3. Check whether a FO-rewriting exists all. If so, produce the rewriting that takes the form of a non-recursive monadic datalog program.
A cycle is defined as follows: A tuple $t \in \Omega_\Sigma$ is a root tuple if $C_t = A_0,\ A_0 \in \text{con}_t$ and $A_0 \notin \text{xcon}_t$. A path through Ω_Σ is a finite sequence of node tuples $t_1, ..., t_k$ from Ω_Σ such that $t_i \rightsquigarrow_{\Omega_\Sigma} t_{i+1}$ for $1 \leq i < k$. A tuple $t \in \Omega_\Sigma$ is looping if there is a path $t_i, ..., t_k$ through Ω_Σ such that $k > 1,\ t = t_i,\ \text{con}_t = con_{t_k}$, and $\text{xcon}_t = \text{xcon}_{t_k}$. Ω_Σ contains a cycle if there are tuples $t, t' \in \Omega_\Sigma$ such that t is a root tuple, t' is a looping tuple, and t' is reachable from t along $\rightsquigarrow_{\Omega_\Sigma}$.
Theorem 1. Concept A_0 is not FO-rewritable under T and Σ if and only if Ω_Σ contains a root cycle.
The algorithm first checks whether Ω_Σ contains a root cycle and if so, returns 'not FO-rewritable'. Otherwise, it constructs a datalog program that is the rewriting of A_0 under T and non-recursive iff A_0 is FO-rewritable under T and Σ. To eliminate 'accidental' recursiveness from the generated datalog program, all rules that contain a predicate which is not reachable from a goal predicate are removed.
In the worst case, the datalog program generated is of double exponential size.

[2]Replace tlc(S) with new (minimal) tlcs which can be used to obtain the tlc(S) using inclusions in T

4.1 Complexity of Decomposed Algorithm

All required operations for building Γ and Σ and for determining the existence of a root cycle require polynomial time. By Theorem 1 an EXPTIME algorithm has been found for deciding FO-rewritability of concept queries under EL-TBoxes and ABox signatures which is optimal [5].

5 Experiments

The experiments described here were conducted on an implementation of the decomposed algorithm. They were carried out on a Linux(3.2.0) machine with a 3.5 GHz quad-core processor of 8 GB of RAM. Seven TBoxes that do not fall in the class of *acyclic TBoxes*[3] were selected because they are not always FO-rewritable [4]. Figure 3 lists the selected TBoxes together with the number of concept inclusions

TBox	CI	CN	RN	no	stop	time	RQ stop	RQ time
ENVO	1942	1558	7	7	100%	2s	92.6%	2m52s
FBbi	567	517	1	0	100%	3s	86.1%	19m25s
MOHSE	3665	2203	71	1	99.6%	6m35s	58.7%	7h17m
NBO	1468	962	8	6	100%	3s	61.5%	3h05m
not-galen	4636	2748	159	44	95.9%	1h15m	48.9%	11h43m
SO	3160	2095	12	15	99.8%	4m28s	77.9%	3h53m
XP	1046	906	27	1	100%	27s	0.0%	7h33m

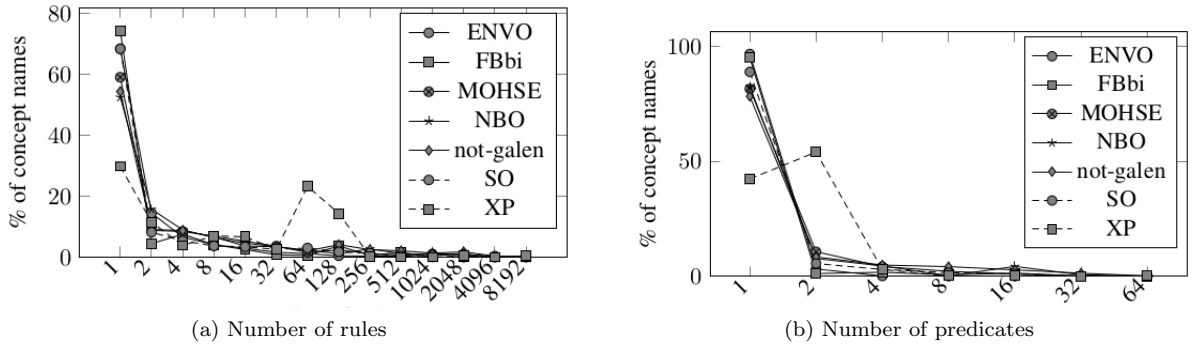
Figure 3: TBoxes used in the experiments

(CI), concept names (CN), and role names (RN) that they contain. Experiments were conducted using the full ABox signature.

To every TBox, the decomposed algorithm was applied to every concept name. In some rare cases, either the set Γ reached excessive size or the calculation of Ω took too long and resulted in non-termination. Termination is achieved in almost all cases heeding to the 30 second timeout that was set. The "stop" column of Figure 3 shows the fraction of inputs for which the algorithm terminated and the "time" column indicates the overall runtime (including timeouts) required to process all concept names from the ontology.

The generated non-recursive datalog rewritings are of reasonable size: the number of rules is indicated by Figure 4(a) whose x-axis has exponential scale. Notably, the size of rule bodies is between one and two atoms in most cases. During the experiment no rule was encountered with more than ten body atoms.

Figure 4(b) highlights the number of IDB predicates in a rewriting. From the experiment the number is quite small and considerably lower than the number of rules in the produced programs. The x-axis adopts an exponential scale.



(a) Number of rules

(b) Number of predicates

Figure 4: Statistics of the rewriting

On comparing the performance of the decomposed algorithm with that of REQUIEM[4]. The same 30

[3]TBoxes whose left-hand sides of all CIs are concept names, with no two CIs having the same left-hand side and have no syntactic cycles.

[4]REQUIEM implements an incomplete resolution based approach to computing FO-rewritings for ontology languages up to $ELHI$[11]

second timeout, which resulted in the termination rate and overall runtime (including timeouts) and is displayed in Figure 3 as "RQ". Since REQUIEM cannot determine whether an input is FO-rewritable, termination cases correspond to positive answers.

5.1 Deductions

The experiments confirm that ontologies that are used in practical applications have a simple structure. As Figure 3 indicates, the number of concept names ("no" column) that are not FO-rewritable is extremely small. Additionally, if a concept name was FO-rewritable, then a rewriting was always found in Phase 1 of the algorithm.

References

[1] Removing domain and range restrictions from elh^{dr} tboxes - efficient query rewriting in the description logic el and beyond. http://www.informatik.uni-bremen.de/tdki/research/papers.html. Accessed: 2019-04-04.

[2] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the el envelope. In *IJCAI*, volume 5, pages 364–369, 2005.

[3] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, and Daniele Nardi. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[4] Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. Deciding fo-rewritability in el. *Description Logics*, pages 70–80, 2012.

[5] Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. First-order rewritability of atomic queries in horn description logics. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[6] Michel Chein and Marie-Laure Mugnier. *Graph-based knowledge representation: computational foundations of conceptual graphs*. Springer Science & Business Media, 2008.

[7] Mélanie König, Michel Leclere, Marie-Laure Mugnier, and Michaël Thomazo. A sound and complete backward chaining algorithm for existential rules. In *International Conference on Web Reasoning and Rule Systems*, pages 122–138. Springer, 2012.

[8] Roman Kontchakov, Mariano Rodriguez-Muro, and Michael Zakharyaschev. Ontology-based data access with databases: A short course. In *Reasoning web. semantic technologies for intelligent data access*, pages 194–229. Springer, 2013.

[9] Carsten Lutz, Inanç Seylan, David Toman, and Frank Wolter. The combined approach to obda: Taming role hierarchies using filters. In *International semantic web conference*, pages 314–330. Springer, 2013.

[10] Carsten Lutz, David Toman, and Frank Wolter. Conjunctive query answering in the description logic el using a relational database system. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[11] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, 8(2):186–209, 2010.

[12] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. In *Journal on data semantics X*, pages 133–173. Springer, 2008.

[13] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. In *International Semantic Web Conference*, pages 558–573. Springer, 2013.

[14] Riccardo Rosati and Alessandro Almatelli. Improving query answering over dl-lite ontologies. In *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*, 2010.