Query Rewriting for Existential Rules with Compiled Preorder
Mélanie König, Michel Leclère, Marie-Laure Mugnier

To cite this version:

HAL Id: lirmm-01164851
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01164851
Submitted on 18 Jun 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Query Rewriting for Existential Rules with Compiled Preorder

Mélanie König, Michel Leclère, Marie-Laure Mugnier
University of Montpellier, Inria, CNRS
Montpellier, France

Abstract
We address the issue of Ontology-Based Query Answering (OBQA), which seeks to exploit knowledge expressed in ontologies when querying data. Ontologies are represented in the framework of existential rules (aka Datalog\textsuperscript{±}). A commonly used technique consists in rewriting queries into unions of conjunctive queries (UCQs). However, the obtained queries can be prohibitively large in practice. A well-known source of combinatorial explosion are very simple rules, typically expressing taxonomies and relation signatures. We propose a rewriting technique, which consists in compiling these rules into a preorder on atoms and embedding this preorder into the rewriting process. This allows to compute compact rewritings that can be considered as “pivotal” representations, in the sense that they can be evaluated by different kinds of database systems. The provided algorithm computes a sound, complete and minimal UCQ rewriting, if one exists. Experiments show that this technique leads to substantial gains, in terms of size and runtime, and scales on very large ontologies. We also compare to other tools for OBQA with existential rules and related lightweight description logics.

1 Introduction
We address the issue of Ontology-Based Query Answering (OBQA), which seeks to exploit knowledge expressed in ontologies when querying data. We consider the novel framework of existential rules, also called Datalog\textsuperscript{±} [Baget et al., 2011; Calì et al., 2012; Krötzsch and Rudolph, 2011]. Existential rules are an extension of function-free positive conjunctive rules that allows for existentially quantified variables in rule heads. These rules are able to assert the existence of unknown entities, a fundamental feature in an open-domain perspective, where it cannot be assumed that the description of data is complete. Interestingly, existential rules generalize Horn description logics (DLs), in particular lightweight DLs used in the context of OBQA, such as DL-Lite [Calvanese et al., 2005] and $\mathcal{EL}$ [Baader, 2003], which form the core of so-called tractable profiles of the Semantic Web ontological language OWL 2. They overcome some limitations of these DLs by allowing for non-tree structures as well as unbounded predicate arity. We consider here the basic database queries, i.e., conjunctive queries (CQs).

There are two main approaches to query answering in presence of existential rules and Horn DLs. The first approach is related to forward chaining: it triggers the rules to build a finite representation of inferred data such that the answers can be computed by evaluating the query against this representation, e.g., [Calì et al., 2008; Thomazo et al., 2012]. The second approach, initiated by DL-Lite [Calvanese et al., 2005], is related to backward chaining: it rewrites the query using the rules such that the answers can be computed by evaluating the rewritten query against the data, e.g., [Baget et al., 2009; Gottlob et al., 2011] on existential rules. Some techniques combine both approaches [Kontchakov et al., 2011; Thomaz and Rudolph, 2014]. Each technique is applicable to a specific existential rule fragment.

In this paper, we focus on the query rewriting approach, which has the advantage of being independent from the data. This technique typically outputs a union of conjunctive queries (UCQ), with the aim of benefiting from optimized relational database management systems (RDBMS). However, despite the good theoretical data complexity, first experiments exhibited a serious problem related to the size of the rewritten query, e.g., [Rosati and Almatelli, 2010]. Indeed, the output query can be exponentially larger than the initial query, even with very simple ontologies, and even if the output is extended to arbitrary first-order queries [Kikot et al., 2011; 2012]. This led to a significant amount of work on rewriting into more compact queries, which may remain first-order queries (hence expressible in SQL) or go beyond them (like Datalog programs, e.g., [Gottlob and Schwentick, 2012; Stefanoni et al., 2012]). Nowadays, mature systems have emerged for OWL 2 QL [Rodriguez-Muro et al., 2013; Civili et al., 2013], as well as prototypes for the $\mathcal{EL}$ family and more expressive DLs, e.g., [Eiter et al., 2012]. Existential rules are more complex to process. Entailment with general existential rules is even undecidable (e.g., [Beeri and Vardi, 1981]). However, expressive subclasses ensuring the existence of a UCQ rewriting are known, such as linear rules, which generalize most DL-Lite dialects, the sticky family, classes satisfying conditions expressed on a graph of rule dependencies, and weakly recursive rules [Calì et al., 2009; 2012; Baget et al., 2011; Civili and Rosati, 2012].
A well-known source of combinatorial explosion are some very simple rules that typically express taxonomies or relation signatures. These rules are at the core of any ontology. We propose a new technique to tame this source of complexity. It relies on compiling these rules into a preorder on atoms and embedding this preorder into the rewriting process. Intuitively, each atom “represents” all its “specializations”. Hence, the query rewriting process avoids exploring many CQs and outputs a small UCQ. This UCQ can be seen as a “pivotal” representation, in the sense that it can be evaluated by different kinds of systems: it can be passed to a Datalog engine or unfolded into a positive existential query (a UCQ or a more compact form of query) to be processed by an RDBMS; it can also be directly processed if data is stored in main memory and the appropriate homomorphism operation is implemented.

First, we prove that the new rewriting operator is sound and complete, and provide an algorithm that, given any CQ and any set of existential rules, effectively outputs a minimal, sound and complete UCQ-rewriting, if one exists (which may not be the case due to the undecidability of the problem). Since the computation of the preorder is independent from any query, we can divide this algorithm into a compilation step performed offline, and a query rewriting step, which takes the preorder as input. Second, we report experiments, which show that this optimization leads to substantial gains, in terms of size of the output and query rewriting runtime, and is able to scale on very large ontologies. In the case when the desired output is a “regular” UCQ, it remains faster for our rewriting tool to compute a pivotal UCQ and to unfold it, than to directly compute a regular UCQ. Moreover, our prototype behaves well compared to other UCQ-rewriting tools tailored for DL-Lite ontologies, even if it does not exploit the specificities of these languages.

2 Preliminaries

An atom is of the form \( p(t_1, \ldots, t_k) \) where \( p \) is a predicate with arity \( k \), and the \( t_i \) are terms, i.e., variables or constants (we do not consider other function symbols). Given an atom or a set of atoms \( A \), \( \text{vars}(A) \), \( \text{consts}(A) \) and \( \text{terms}(A) \) denote its set of variables, of constants and of terms, respectively. In the following examples, all the terms are variables (denoted by \( x, y, z \), etc.) unless otherwise specified. \( \models \) denotes the classical logical consequence. Given sets of atoms \( A \) and \( B \), a homomorphism \( h \) from \( A \) to \( B \) is a substitution of \( \text{vars}(A) \) by \( \text{terms}(B) \) such that \( h(A) \subseteq B \). It is convenient to extend the domain of a substitution \( h \) to a set of terms, such that \( h(t) = t \) for all unchanged terms (in particular constants). If there is a homomorphism \( h \) from \( A \) to \( B \), we say that \( A \) maps to \( B \) (by \( h \)); then, \( A \) is said to be more general than \( B \), and \( B \) more specific than \( A \).

A fact is the existential closure of a conjunction of atoms (this generalization of the classical notion of a fact allows to take existential variables into account). A conjunctive query is an existentially quantified conjunction of atoms. W.l.o.g. we consider only Boolean conjunctive queries (i.e., closed formulas) in this paper and denote them by CQs. In the following, we will see facts and CQs as sets of atoms. The answer to a CQ \( Q \) in a fact \( F \) is yes if \( F \models Q \); it is well-known that \( F \models Q \) iff \( Q \) maps to \( F \).

Definition 1 (Existential rule) An existential rule (or simply rule hereafter) is a formula \( R = \forall x \forall y (\exists z \ H(x, y, z)) \) where \( B = \text{body}(R) \) and \( H = \text{head}(R) \) are conjunctions of atoms, resp. called the body and the head of \( R \). The frontier of \( R \) is the set of variables \( \text{vars}(B) \cap \text{vars}(H) = \{y\} \). The existential variables in \( R \) is the set of variables \( \text{vars}(H) \setminus \text{vars}(B) = \{z\} \).

In the following, we will omit quantifiers in rules as there is no ambiguity. E.g. \( p(x, y) \rightarrow p(y, z) \) denotes the formula \( \forall x \forall y (p(x, y) \rightarrow \exists z p(y, z)) \).

A knowledge base (KB) \( K = (F, R) \) is composed of a finite set of facts (which can be seen as a single fact) \( F \) and a finite set of existential rules \( R \). The CQ entailment problem is the following: given a KB \( K = (F, R) \) and a CQ \( Q \), does \( F, R \models Q \) hold? This problem is known to be undecidable in the general case [Beeri and Vardi, 1981]. It can be solved by computing a sound and complete rewritten query \( Q \) from \( Q \) and \( R \), provided that such a finite \( Q \) exists, then asking if \( F \models Q \). Here, the target query \( Q \) is a union of CQs (UCQ), that we see as a set of CQs, called a rewriting set of \( Q \).

Definition 2 (Sound and Complete (rewriting) set of CQs) Let \( R \) be a set of existential rules, \( Q \) be a CQ, and \( Q \) be a set of CQs. \( Q \) is said to be said to be sound w.r.t. \( Q \) and \( R \) if for any fact \( F \), for all \( Q_i \in Q \) if \( Q_i \) maps to \( F \) then \( R, F \models Q_i \). Reciprocally, \( Q \) is said to be complete w.r.t. \( Q \) and \( R \) if for any fact \( F \), if \( R, F \models Q \) then there is \( Q_i \in Q \) such that \( Q_i \) maps to \( F \).

A set of rules \( R \) for which any query \( Q \) has a finite sound and complete set of rewritings (in other words, \( Q \) is rewritable into a UCQ), is called a finite unification set (fus) [Baget et al., 2009]. Note that restricting a complete rewriting set \( Q \) to its most general elements preserves its completeness. A cover of \( Q \) is an inclusion-minimal subset \( Q' \) of \( Q \) such that each element \( Q \) of \( Q \) is more specific than an element \( Q' \) of \( Q' \) (i.e., in database terms: \( Q \) is contained in \( Q' \)). If a set has a finite cover, then all its covers have the same size, hence all minimal, sound and complete rewriting sets have the same cardinality [Köng et al., 2012]. Hereafter, we, respectively, denote by \( R, R' \) and \( Q, Q' \) the considered set of rules, a rule of \( R \), and the initial query. We assume that rules have disjoint sets of variables, as well as \( Q \) and \( R \).

The rewriting operation involves unifying part of \( Q \) and part of \( \text{head}(R) \). Existential variables in rule heads induce a possibly complex structure, therefore unification has to consider subsets of atoms at once (called “pieces”) instead of single atoms, hence the name piece-unifier. Indeed, if a variable \( x \) from \( Q \) is unified with an existential variable from \( \text{head}(R) \), then all atoms in which \( x \) occurs must be part of the unification, otherwise the obtained rewriting is unsound. Given \( Q' \subseteq Q \), we call separating variables of \( Q' \), the variables occurring in both \( Q' \) and \( Q \setminus Q' \); the other variables from \( Q' \) are called non-separating. Let \( Q' \) be the unified part of \( Q \); only non-separating variables from \( Q' \) can be unified with an existential variable of the rule.
Definition 3 (Piece-Unifier [Köng et al., 2012]) A piece-unifier of $Q$ with $R$ is a triple $\mu = (Q', H', u)$, where $Q' \neq \emptyset$, $Q' \subseteq Q$, $H' \subseteq \text{head}(R)$ and $u$ is a substitution of $T = \text{terms}(Q') \cup \text{terms}(H')$ by $T$, such that:

1. $u(H') = u(Q')$;
2. for all existential variable $x \in \text{vars}(H')$ and $t \in T$, with $t \neq x$, if $u(x) = u(t)$, then $t$ is a non-separating variable from $Q'$.

Example 1 (Piece-unification) Let $R = \text{twin}(x, y) \rightarrow \text{motherOf}(z, x) \land \text{motherOf}(z, y)$, where $z$ is an existential variable, expressing that twins have a common mother. Let $Q_{no} = \{\text{motherOf}(v, u), \text{painter}(v)\}$, asking if there is a mother who is a painter. The one-step rewriting of $R$ according to $\mu = (Q', H', u)$ is $Q_{yes} = \{\text{motherOf}(v, w), \text{motherOf}(v, t), \text{female}(w), \text{male}(t)\}$, asking if there is a mother of a female and a male. A piece-unifier of $Q_{yes}$ with $R$ is $\mu = (Q' = \{\text{motherOf}(v, w), \text{motherOf}(v, t), H' = \{\text{motherOf}(z, x), \text{motherOf}(z, y), u = (z \mapsto v, x \mapsto w, y \mapsto t)\})$ (in the general case $Q'$ and $H'$ may be non-isomorphic). The one-step rewriting of $Q_{yes}$ according to $\mu$ is $(\text{twin}(w, t), \text{female}(w), \text{male}(t))$, as formally defined below.

Definition 4 (One-step Rewriting, $\mathcal{R}$-rewriting, $\beta^*$) Given a piece-unifier $\mu = (Q', H', u)$ of $Q$ with $R$, the one-step rewriting of $Q$ according to $\mu$, denoted by $\beta(Q, R, \mu)$, is the $\mathcal{R}$-rewriting of $Q$ obtained by a finite sequence $(Q_0 = Q, Q_1, \ldots, Q_k)$ such that for all $0 \leq i < k$, there is $R_i \in \mathcal{R}$ and a piece-unifier $\mu_i$ of $Q_i$ with $R_i$ such that $Q_{i+1} = \beta(Q_i, R_i, \mu_i)$. We denote by $\beta^*(Q, \mathcal{R})$ the set of all $\mathcal{R}$-rewriting of $Q$.

It is known that $\beta^*(Q, \mathcal{R})$ is a sound and complete set (within the meaning of Def. 2) [Baget et al., 2011].

3 Embedding a Preorder on Atoms

We first explain the ideas underlying the rewriting technique before providing formal definitions. An essential component of any ontology is a hierarchy of concepts and, to a lesser extent, a hierarchy of relations (binary relations are also called roles or properties). In a logical framework, concepts and relations are represented by predicates. Simple rules of the form $p(x_1, \ldots, x_k) \rightarrow q(x_1, \ldots, x_k)$, where $k = 1$ for a concept, express that $p$ is more specific than $q$ (notation $p \leq q$). See e.g., the logical translation of atomic inclusions in DL, and the subClassOf and subPropertyOf assertions in RDFS / OWL. These rules, called hierarchical hereafter, are an obvious cause of combinatorial explosion in query rewriting, as illustrated by the next example.

Example 2 Let $R_1 \ldots R_n$ be rules of the form $R_i : b_i(x) \rightarrow b_{i-1}(x)$. Let $Q = \{b_0(x_1), \ldots, b_0(x_k)\}$. Each atom $b_0(x_i)$ in $Q$ is rewritten into $b_1(x_j)$, which in turn is rewritten into $b_2(x_j)$, and so on. Thus, there are $(n + 1)^k$ rewrites of $Q$.

Hierarchical rules can be compiled into a (partial) preorder (i.e., a reflexive and transitive relation) on predicates, say $\leq$. Then, the homomorphism notion can be extended to take this preorder into account. Let us call $\leq$-homomorphism $h$ from a set of atoms $A_1$ to a set of atoms $A_2$ a substitution of terms($A_1$) by terms($A_2$) such that for all $q(e_1, \ldots, e_p) \in A_1$, there is $p(h(e_1), \ldots, h(e_p)) \in A_2$ with $p \leq q$. It is easily checked that, given a set of hierarchical rules $\mathcal{R}_c$, it holds that $\mathcal{R}_c, F \models Q$ if and only if there is a $\leq$-homomorphism from $Q$ to $F$. Now, let $\mathcal{R}_c = \mathcal{R}_c \cup \mathcal{R}_c$ be a set of existential rules, where $\mathcal{R}_c$ is composed of the hierarchical rules. Given the preorder $\leq$ associated with $\mathcal{R}_c$, we would like to have $F, \mathcal{R}_c \models Q$ if and only if there is an $\mathcal{R}_c$-rewriting of $Q$ that maps to $F$ by a $\leq$-homomorphism. In Ex. 2, we would have $b_j \leq b_i$ for any $i \leq j$; then, since all rules are hierarchical, we would get a single rewriting, i.e., $Q$ itself, instead of exponentially many. However, to achieve completeness, we cannot simply rewrite $Q$ with $\mathcal{R}_c$, and forget about $\mathcal{R}_c$, we have to embed the preorder into the rewriting process. This idea can be further extended to compile all rules with an atomic body, as long as they do not introduce existential variables. This allows to compile other common axioms, as illustrated by Ex. 3. However, since the atoms in a rule may have predicates of different arity and arguments in different positions, we cannot rely on a simple preorder on predicates anymore. We have to embed a preorder on atoms.

Example 3 Let $\mathcal{R}_ex$ be the following set of rules: $R_1 = r(x, y) \rightarrow t(x, y)$ ($r$ is a specialization of $t$) $R_2 = s(x, y) \rightarrow t(y, x)$ $R_3 = t(x, y) \rightarrow s(y, x)$ ($s$ and $t$ are inverse relations) $R_4 = t(x, y) \rightarrow q(x)$ ($q$ is the domain of $t$) $R_5 = t(x, y) \rightarrow q(y)$ ($q$ is the range of $t$) $R_6 = p(x, y, z) \rightarrow r(x, z)$ ($r$ is a projection of $p$) $R_7 = p(x, y, z) \rightarrow s(x, y)$ (introduction of a “self loop”)

A rule is said to be compilable if it has a single body atom, no existential variable and no constant. W.l.o.g. we also assume that a compilable rule has a single head.

Definition 5 (Inferred Rule, Saturation) Let $R_1$ and $R_2$ be compilable rules such that head($R_1$) and body($R_2$) are unifiable by a (classical) most general unifier $u$. The rule inferred from $(R_1, R_2)$ is $R_1 \cdot R_2 = u(\text{body}(R_1)) \cup u(\text{head}(R_2))$. Given a set $\mathcal{R}_c$ of compilable rules, the saturation of $\mathcal{R}_c$, denoted by $\mathcal{R}_c^*$, is the closure of $\mathcal{R}_c$ by the $\cdot$ operation.

Example 4 The rules inferred from $\mathcal{R}_ex$ (Ex. 3) are the following: we recall that rules have disjoint sets of variables, even if we use the same variables for simplicity: $R_1 \cdot R_2 = r(x, y) \rightarrow s(y, x)$; $R_1 \cdot R_3 = r(x, y) \rightarrow q(x)$ $R_1 \cdot R_4 = r(x, y) \rightarrow q(y)$; $R_2 \cdot R_2 = s(x, y) \rightarrow s(x, y)$ $R_2 \cdot R_3 = s(x, y) \rightarrow q(y)$; $R_3 \cdot R_3 = s(x, y) \rightarrow q(x)$ $R_4 \cdot R_2 = t(x, y) \rightarrow t(x, y)$; $R_5 \cdot R_1 = p(x, y, z) \rightarrow t(x, z)$ $R_6 \cdot R_2 = p(x, y, z) \rightarrow t(x, x)$ $R_6 \cdot R_1 \cdot R_2 = p(x, y, z) \rightarrow s(x, x)$ $R_5 \cdot R_1 \cdot R_3 = p(x, y, z) \rightarrow q(x)$ $R_5 \cdot R_1 \cdot R_4 = p(x, y, z) \rightarrow q(z)$ $R_6 \cdot R_1 \cdot R_3 = p(x, y, z) \rightarrow q(z)$ $R_6 \cdot R_2 \cdot R_3 = p(x, y, z) \rightarrow q(x)$

The size of $\mathcal{R}_c^*$ is polynomial in the size of $\mathcal{R}_c$ when the predicate arity is bounded (more specifically, when the number of occurrences of a variable in a predicate is bounded):

1 The condition on constants is to simplify definitions.
Definition 8 (piled rules in piece-unifiers.

Moving tautological rules (necessarily of the form $R_s(a)$ for a substitution $A$ strictly more general than $R$) and $(\text{Def. 3})$, with Condition 1 replaced by $u(H') \prec u(Q')$. The one-step $\Rightarrow$-rewriting of $Q$ according $\mu$, denoted by $\beta_\prec$, is the $\mu$-body($R$) $\cup u(Q \setminus Q')$.

Example 7 Consider the rules from Ex. 3, the rule $R = b(x) \rightarrow t(x,y)$ (where $y$ is existential) and the query $Q_1 = \{t(u,v), q(w)\}$. There is no piece-unifier of $Q_1$ with $R$ but there is a $\prec$-piece-unifier $\mu = (Q_1, \text{head}(R), \{x \rightarrow u, y \rightarrow v\})$, since $R_3$ subsumes $t(u,v) \rightarrow q(w)$. We obtain $\beta_\prec(Q_1, R, \mu) = \{b(u)\}$. Consider now $Q_2 = \{q(w), s(z), w, c(w)\}$. With the $\prec$-piece-unifier $\mu' = (\{q(w), s(z), w\}, \text{head}(R), \{x \rightarrow w, y \rightarrow z\})$, using $R_3$ and $R_e$, we obtain $\beta_\prec(Q_2, R, \mu') = \{b(w), c(w)\}$.

Logical entailment between queries and facts can be computed by a homomorphism check, which we now extend to embed $\Rightarrow$.

Definition 9 ($\ll$-homomorphism) Let $A$ and $B$ be sets of atoms. A $\ll$-homomorphism from $B$ to $A$ is a substitution $h$ from vars$(B)$ to terms$(A)$ such that for all $b \in B$, there is $a \in A$ with $a \ll h(b)$.

Example 8 Consider $Q_1$ and $Q_2$ from Ex. 7. The substitution $h = \{(u, w), (z, w)\}$ is a $\ll$-homomorphism from $Q_1$ to $Q_2$. Indeed, $t(u, v)$ and $q(w)$ are both mapped to $s(z, w)$, by using $R_2$ and $R_2 \cdot R_3$ respectively.

Property 2 Let $A$ and $B$ be sets of atoms, and $R_e$ be a set of compilable rules with associated preorder $\ll$. There is a $\ll$-homomorphism from $B$ to $A$ if $\Phi(A), R_e \models \Phi(B)$, where $\Phi$ assigns an existentially closed formula to a set of atoms.

The next theorem states that the new rewriting operator $\beta_\ll$, associated with $\ll$-homomorphism, is logically sound and complete.

Theorem 1 (Soundness and Completeness of $\beta_\ll$) Let $\mathcal{K} = (F, \mathcal{R})$ be a KB, where $\mathcal{R}$ is partitioned into $\mathcal{R}_e$ and $\mathcal{R}_c$, a set of compilable rules with associated preorder $\ll$. Let $Q$ be a $\mathcal{CQ}$. Then, $\mathcal{F} \models Q$ if there is $Q' \in \beta_\ll(Q, \mathcal{R}_e)$ with a $\ll$-homomorphism from $Q'$ to $F$.

Proof: (Sketch) We rely on the soundness and completeness of classical piece-based rewriting and prove that: (1) if $Q' \in \beta_\ll(Q, \mathcal{R}_e)$ then $Q'' \in \beta_\ll(Q, \mathcal{R}_e)$ and (2) if $Q'' \in \beta_\ll(Q, \mathcal{R}_e)$ then there is $Q'' \in \beta_\ll(Q, \mathcal{R}_c)$ such that $Q'' \ll Q''$.

4 A Correct $\ll$-Rewriting Algorithm

The compilation step consists in partitioning $\mathcal{R}$ into $\mathcal{R}_e$ and $\mathcal{R}_c$, and compiling $\mathcal{R}_e$ into the preorder $\ll$. Then, the rewriting step (Algorithm 1) follows the general schema of [König et al., 2012]. Given $\mathcal{R}_e, \ll$ and $Q$, the algorithm starts from the rewriting set $\mathcal{Q}_F = \{Q\}$ and proceeds in a breadth-first manner. At each step, queries from $\mathcal{Q}_F$ that have been generated at the preceding step (i.e., set $\mathcal{Q}_F$) are explored. "Exploring" a query consists of computing the set of one-step rewritings of this query with all rules in $\mathcal{R}_e$ (set $\mathcal{Q}_e$). At the end of the step, only a cover of $\mathcal{Q}_F \cup Q$ is kept (in case of equivalent queries, priority is given to $\mathcal{Q}_F$ for termination reasons).

3109
Algorithm 1: \(\equiv_e\)-REWITING ALGORITHM

Data: A set of existential rules \(R_e\), a preorder on atoms \(\equiv\) and a conjunctive query \(Q\)

Result: A cover of the sound \(R_e\)-\(\equiv\)-rewritings of \(Q\)

\[Q_F \leftarrow \{Q\}; \quad \text{// resulting set} \]
\[Q_E \leftarrow \{Q\}; \quad \text{// queries to explore} \]

\[\text{while } Q_E \neq \emptyset \text{ do} \]
\[Q_t \leftarrow \emptyset; \quad \text{// queries generated at this rewriting step} \]
\[\text{for } Q_t \in Q_E \text{ do} \]
\[\text{for } R \in R_e \text{ do} \]
\[\text{for } \mu \equiv \text{-piece-unifier of } Q_t \text{ with } R \text{ do} \]
\[Q_t \leftarrow Q_t \cup \{\beta_e(Q_t, R, \mu)\}; \]
\[Q^e \leftarrow \text{ComputeCover}(Q_F \cup Q_t); \quad \text{// update cover} \]
\[Q_E \leftarrow Q^e \setminus Q_F; \quad \text{// select unexplored queries} \]
\[Q_F \leftarrow Q^e; \quad \text{// update cover} \]

Algorithm 1 outputs a (minimal) sound and complete finite rewriting set (with respect to \(\equiv\)-homomorphism), if such exists, and it does not terminate otherwise.

5 Query Evaluation

The rewriting set \(Q\) produced by Algorithm 1 can be seen as a “pivotal” representation, in the sense that it can be transformed into different kinds of queries, depending on the type of data storage and the applicative context. Obviously, \(Q\) can be directly evaluated with an adequate implementation of \(\equiv\)-homomorphism in the case the data can be loaded in main memory. Otherwise, the set \(Q \cup R_e\) can be straightforwardly translated into a Datalog query. A mixed approach can be adopted with \(R_e\) being used to saturate the data, and \(Q\) being evaluated over the saturated data. One may even assume that all information that could be inferred by compilable rules is already present in the data, and delegate the effective computation of saturation [Rodriguez-Muro and Calvanese, 2012].

When partial saturation of the data is not feasible, \(Q\) may also be unfolded into a set of CQs (i.e., a UCQ) \(\mathcal{Q}'\). \(\mathcal{Q}'\) is obtained from \(Q\) by adding, for each \(Q \in \mathcal{Q}\), all \(Q'\) such that \(Q' \equiv Q\) (then eliminating redundant queries). Our experiments (see the last section) show that it is more efficient to unfold \(Q\) than to directly compute \(\mathcal{Q}'\). More compact forms of positive existential queries can be computed, for instance unions of semi-conjunctive queries (SCQs), which are conjunctions of disjunctions [Thomazo, 2013]: each CQ \(Q \in \mathcal{Q}\) is transformed into an SCQ by replacing each atom \(a \in Q\) by the disjunction of all atoms \(a'\) such that \(a' \equiv a\).

6 Related Work

Since the seminal paper on DL-Lite [Calvanese et al., 2005], a significant amount of work has been carried out on query rewriting algorithms, mainly for DL-Lite, but also for other Horn-DLs. The work closest to ours is certainly the tree-witness (tw) rewriting algorithm for DL-Lite [Kikot et al., 2012] because of the similarities between tw-rewritings and pieces. Another similarity is that tw-rewriting can make the assumption that the database is already saturated with in- ferrable knowledge that does not involve existential variables (H-completeness assumption). In the context of DL-Lite, this kind of knowledge corresponds exactly to compilable rules (up to the usual DL restrictions: predicate arity bounded by two, no multiple occurrences of variables in atoms). Hence, one could see our technique as an extension of tw-rewriting with the H-completeness assumption to existential rule KBs. However, the underlying techniques are quite different. Moreover, tw-rewriting heavily exploits the specificities of DL-Lite. First, “DL-Lite rules” without existential variables are necessarily compilable rules. Second, the “anonymous part” of the possibly infinite canonical model of a DL-Lite knowledge base is a set of trees (instead of a hypergraph with any structure for existential rules). This allows for a smart technique that rewrites the query in a single pass (instead of possibly exponentially many passes with fus rules).
Regarding general existential rules, two rewriting methods were proposed [Baget et al., 2009; Gottlob et al., 2011] and respectively implemented in Alaska/PURE [König et al., 2012] and Nyaya [Virgilio et al., 2012].

7 Experiments

Our algorithm\(^2\) was implemented in Java, as an extension of the query rewriting prototype PURE. All tests were performed on a DELL machine with a processor at 3.60 GHz and 16 GB of RAM. As benchmarks dedicated to existential rules are not available yet, and in order to compare with other tools producing UCQs, which are mostly restricted to DL-Lite, we considered rule bases obtained by translation of DL-Lite\(_R\) ontologies: first, the widely used benchmark introduced in [Pérez-Urbina et al., 2009] (i.e., ADOLENA (A), STOCKEXCHANGE (S), UNIVERSITY (U) and VICODI (V)); second, very large ontologies built from OpenGalen2 (G) and OBOProtein (O), and used in [Trivela et al., 2013], which respectively contain more than 53k and 34k rules, with 54% and 64% of compilable rules. Each ontology is provided with 5 handcrafted queries. Timeout was set to 10 minutes. Due to space limitation, we list only parts of the experiments.

We first evaluated the impact of rule compilation on the rewriting process, w.r.t. rewriting sizes and runtime respectively. We denote by PURE\(_C\) the extension of PURE and call pivotal UCQ its output. Table 1 shows the size of the UCQ (we recall that it is the same for all systems outputing a minimal, sound and complete UCQ), the size of the pivotal UCQ produced by PURE\(_C\), and the number of generated queries during the rewriting process for PURE and PURE\(_C\). Missing values are due to timeouts. We find a huge gap between the sizes of the output; the pivotal UCQ is often restricted to a single CQ even when the UCQ has thousands of CQs (which also explains the gap between the numbers of generated queries). Unsurprisingly, the results on the runtimes lead to similar observations (Table 2, Columns 1 and 2). We see that PURE\(_C\) remains faster than PURE when we include the time required to unfold the pivotal UCQ into a UCQ (Table 2, Column 3), except for Q2 on O, which comes from the fact that the pivotal UCQ is almost as large as the UCQ. Note that we implemented a brute-force unfolding method, which removes redundant CQs only at the end by a pairwise comparison of queries. Almost all the unfolding time is actually spent in checking redundancies. Nevertheless, the size of the UCQ obtained for some queries (up to more than 30000 CQs on O) clearly advocates for more compact forms of output.

We also compared to other query rewriting tools, namely: PURE and Nyaya, which are the only tools processing existential rules, as well as some well-known DL tools: Requiem [Pérez-Urbina et al., 2009] (optimized “full modality”), Iqaros [Imprialou et al., 2012], Rapid [Chortaras et al., 2011] and tw-rewriting (part of the Ontop OBDA system [Rodriguez-Muro et al., 2013]). We emphasize again that these DL tools exploit the specificities of DL-Lite, specially the most recent ones, Rapid and tw-rewriting, whereas the algorithms of Nyaya and PURE are designed for general existential rules. Despite this fact, PURE\(_C\) (without or with unfolding) scales well on DL-Lite large ontologies (except for extreme cases which are difficult for all tools, see Ontology O). Globally, PURE\(_C\) behaves similarly to tw-rewriting and Rapid. If we restrict the comparison to classical UCQ output, the fastest tools are undeniably tw-rewriting and Rapid, followed by PURE\(_C\) with unfolding. The difficulties of Nyaya on A can be explained by the fact that A contains some rules with two atoms in the head, whereas Nyaya only processes rules with a single head; hence, it had to take as input the ontology obtained by decomposing these rules into single-head rules, which introduces new predicates, whereas PURE processes rules with any head size. Nyaya could not process the very large ontologies G and O, which also needed to be decomposed.

We checked that all systems return exactly the same size of UCQ, hence the choice of a given query rewriting tool between those outputing a UCQ is irrelevant for the query evaluation step. We carried out additional experiments to compare

### Table 1: Impact of rule compilation on sizes

<table>
<thead>
<tr>
<th>A</th>
<th>Q1</th>
<th>27</th>
<th>2</th>
<th>460</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
<td>50</td>
<td>2</td>
<td>172</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>104</td>
<td>1</td>
<td>317</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>224</td>
<td>2</td>
<td>827</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>624</td>
<td>1</td>
<td>1417</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Q1</td>
<td>15</td>
<td>1</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>Q2</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>72</td>
<td>1</td>
<td>118</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>185</td>
<td>329</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>30</td>
<td>1</td>
<td>60</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Q1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Q2</td>
<td>1152</td>
<td>1</td>
<td>1276</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>488</td>
<td>5</td>
<td>1515</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>147</td>
<td>1</td>
<td>155</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>324</td>
<td>19</td>
<td>909</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Q1</td>
<td>27</td>
<td>20</td>
<td>29</td>
<td>2</td>
</tr>
<tr>
<td>Q2</td>
<td>1356</td>
<td>1264</td>
<td>1356</td>
<td>1264</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>33887</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>34733</td>
<td>682</td>
<td>-</td>
<td>794</td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>36612</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

### Table 2: Impact of rule compilation on rewriting time

<table>
<thead>
<tr>
<th>A</th>
<th>Q1</th>
<th>180</th>
<th>10</th>
<th>130</th>
<th>1120</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
<td>90</td>
<td>40</td>
<td>860</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>Q3</td>
<td>170</td>
<td>10</td>
<td>30</td>
<td>2360</td>
<td>130</td>
</tr>
<tr>
<td>Q4</td>
<td>280</td>
<td>130</td>
<td>5550</td>
<td>250</td>
<td>130</td>
</tr>
<tr>
<td>Q5</td>
<td>1500</td>
<td>0</td>
<td>440</td>
<td>33200</td>
<td>460</td>
</tr>
<tr>
<td>B</td>
<td>Q1</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Q2</td>
<td>10</td>
<td>0</td>
<td>50</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Q3</td>
<td>110</td>
<td>0</td>
<td>70</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>Q4</td>
<td>120</td>
<td>0</td>
<td>60</td>
<td>20</td>
<td>130</td>
</tr>
<tr>
<td>Q5</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>70</td>
</tr>
<tr>
<td>C</td>
<td>Q1</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Q2</td>
<td>1060</td>
<td>50</td>
<td>620</td>
<td>-</td>
<td>209040</td>
</tr>
<tr>
<td>Q3</td>
<td>1020</td>
<td>70</td>
<td>260</td>
<td>-</td>
<td>259100</td>
</tr>
<tr>
<td>Q4</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>-</td>
<td>190250</td>
</tr>
<tr>
<td>Q5</td>
<td>890</td>
<td>30</td>
<td>90</td>
<td>-</td>
<td>238450</td>
</tr>
<tr>
<td>D</td>
<td>Q1</td>
<td>440</td>
<td>130</td>
<td>140</td>
<td>-</td>
</tr>
<tr>
<td>Q2</td>
<td>1160</td>
<td>1110</td>
<td>1870</td>
<td>-</td>
<td>21780</td>
</tr>
<tr>
<td>Q3</td>
<td>TO</td>
<td>90</td>
<td>557990</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>Q4</td>
<td>TO</td>
<td>430</td>
<td>TO</td>
<td>TO</td>
<td>139980</td>
</tr>
<tr>
<td>Q5</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
<td>562220</td>
</tr>
</tbody>
</table>

*Time.s in ms, with increments of 10 ms (0 means < 10); TO stands for timeout.*

---

\(^2\)Available at [https://github.com/graphik-team/graal](https://github.com/graphik-team/graal)
the evaluation of the pivotal UCQ over the data saturated by the compilable rules and the evaluation of the corresponding classical UCQ over the initial data (data generated with the modified LUBM generator [Lutz et al., 2013] and stored in a MySQL database). As expected, in a number of cases the system did not accept the classical UCQ because of its size, and in the other cases the pivotal UCQ was evaluated much more efficiently than the classical UCQ.

Further work includes extending query rewriting techniques outside the fis fragment, by exploiting datalog rewritability or combining with other paradigms for rules.

Acknowledgments. This work was partially funded by the ANR project PAGODA (ANR-12-JS02-007-01).

References


