Practical Datalog Rewriting for Existential Rules

Zhe Wang, Peng Xiao, and Kewen Wang

Griffith University, Australia

Abstract. Existential rules is an expressive ontology formalism for ontology-mediated query answering and as a trade-off, query answering is of high complexity, while several tractable fragments have been identified. Existing systems are based on first-order rewriting methods and the resulting queries can be too large for DBMS to handle. It is shown that datalog rewriting can result in more compact queries but the previous study is mostly theoretical. A practical datalog rewriting algorithm is still missing. In this paper, we fill the gap by proposing a practical datalog rewriting algorithm for conjunctive query answering over existential rules, and establish its correctness over well known fragments of existential rules. Experiments on our prototype system showed superior or comparable performance over state-of-the-art systems on both the compactness of rewritings and the efficiency of query answering.

1 Introduction

Existential rules (a.k.a. Datalog± and tuple generating dependencies) [2, 6] is a family of expressive ontology languages. It attracted intensive interest lately due to its expressive power covering datalog and many Horn description logics, including the core dialects of DL-Lite and $\mathcal{EL}$ [5], which underlay the OWL 2 Profiles. This makes existential rules an appealing formalism for ontology-mediated query answering. While the query answering problem is undecidable over the full formalism, several interesting fragments have been proposed [4, 6, 17] that support tractable query answering.

A major approach for query answering over ontologies expressed in description logics or existential rules is query rewriting. Given an ontology Σ and a query q, a rewriting method transforms them into another query q_Σ, which is sometimes in a different query formalism, such that answering q_Σ can be handled by conventional database management systems (DBMSs) and at the same time preserves the answers to the original ontology-mediated query. The rewriting approach is particularly promising as it allows ontology-mediated query answering to be implemented on top of existing highly-optimised database query engines. While many algorithms and systems have been developed for various description logics [18, 7, 25, 23], particularly for DL-Lite and $\mathcal{EL}$ [15, 20, 22, 12], it is challenging to extend them to more general existential rules, as existential rules allow predicates of arbitrary arities (instead of only unary and binary predicates) and variable permutations in the rules.

Existing systems for query answering over existential rules are typically based on first-order rewritings [8, 13, 14], i.e., q_Σ is a first-order query. A limitation of such an approach is that it can only handle ontologies and queries that are first-order rewritable. Well-accepted first-order rewritable classes are the linear and sticky existential rules [6]. Yet many practical ontologies do not necessarily fall into these classes, such as some ontologies formulated in $\mathcal{EL}$. Even for ontologies and queries that are first-order rewritable, the results of rewriting can suffer from a significant blow-up and become difficult for DBMSs to handle [19].

On the other hand, taking datalog as the target query language can lead to much more compact rewritings and it is shown for description logics that executing (non-recursive) datalog rewritings is much more feasible for DBMSs than equivalent first-order rewritings [12]. All ontologies and queries that are first-order rewritable are trivially datalog rewritable, and more datalog rewritable classes are known, such as the guarded existential rules [9]. However, existing research on datalog rewriting of existential rules are mostly theoretical [10, 3] (refer to [1] for a detailed discussion). While several algorithms and systems have been developed for datalog rewriting for various description logics [7, 12], to the best of our knowledge, a practical system is still missing for datalog rewriting over more general existential rules.

In this paper, we fill the gap by presenting both a practical algorithm and a prototype system for datalog rewriting and query answering over existential rules. Our algorithm is based on the notion of unfolding [24] and the observation that unfolding can generate a datalog rewriting whenever it exists. Yet instead of naive unfolding, to achieve compactness of rewriting, we separate the results of unfolding into short rules by introducing fresh predicates and reusing such predicates when possible. Such a separation technique not only reduces the length of generated rules but also facilitates structure sharing (via the reuse of predicates). While such a rewriting process may not terminate, we move on to identify classes of ontologies where the rewriting process terminates and produce correct datalog rewritings, including both a novel class and several existing well-accepted classes. After that, we introduce a practical algorithm for computing the datalog rewriting through constructing rewriting graphs, which again facilitates structure sharing and can be seen as a decomposed representation of rewritings. Finally, we implemented our algorithm as a prototype system and evaluate it against the state-of-the-art systems for both existential rules and description logics on commonly used benchmark ontologies and queries.

2 Preliminaries

Let N_c, N_n, and N_v be disjoint, countably infinite sets of constants, (labelled) nulls, and variables respectively. A term is either a constant, null, or variable. We assume standard first-order logic notions, such as predicates, (ground) atoms, formulas, entailment ($\models$) and equivalence ($\equiv$). An *instance* is a (possibly infinite) set of atoms that contains only constants and nulls. A *dataset* is a finite ground instance. For a formula or an instance φ, $\mathsf{var}(\varphi)$ denotes the variables in φ.

An *existential rule* (or a rule) r is a formula of the form

$$\forall \boldsymbol{x}.\forall \boldsymbol{y}.[\exists \boldsymbol{z}.\varphi(\boldsymbol{x},\boldsymbol{z}) \leftarrow \psi(\boldsymbol{x},\boldsymbol{y})]$$

where $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ are pairwise disjoint vectors of variables, and $\varphi(\boldsymbol{x},\boldsymbol{z})$ and $\psi(\boldsymbol{x},\boldsymbol{y})$ are conjunctions of atoms with variables from respectively $\boldsymbol{x} \cup \boldsymbol{z}$ and $\boldsymbol{x} \cup \boldsymbol{y}$. We assume rules do not contain constants. Variables in $\boldsymbol{x}$ are called frontier variables, denoted $\mathsf{fr}(r)$. and those in $\boldsymbol{z}$ are called existential variables, denoted $\mathsf{ext}(r)$. Formula φ is the *head* of the rule r, denoted $\mathsf{head}(r)$, and formula ψ is the *body* of r, denoted $\mathsf{body}(r)$;

again, they can be seen as (existentially quantified conjunctions of) sets of atoms. For brevity, universal quantifiers in a rule are often omitted, and we sometimes express rule r as $\mathsf{head}(r) \leftarrow \mathsf{body}(r)$. A *datalog* rule r is an existential rule whose head consists of a single atom and $\mathsf{ext}(r)$ is empty.

A *conjunctive query* (CQ) q is a first-order formula of the form $\exists \boldsymbol{y}.\varphi(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are tuples of variables and ϕ is a conjunction of atoms containing only constants and variables from $\boldsymbol{x} \cup \boldsymbol{y}$. Sometimes it is convenient to treat it as a datalog rule $\mathsf{Q}(\boldsymbol{x}) \leftarrow \varphi(\boldsymbol{x}, \boldsymbol{y})$ where Q is a fresh predicate with arity $|\boldsymbol{x}|$. If $\boldsymbol{x}$ is empty then q is a *Boolean CQ* (BCQ). Answering CQs can be reduced to that of BCQs and hence w.l.o.g. we consider only BCQs in this paper. For convenience, we assume a fixed renaming between variables and labelled nulls, which allows us to identify a BCQ with the set of its atoms where all the variables are renamed as nulls, and vice versa. A union of BCQs (UCQ) is a disjunction of BCQs, which is seen as a finite set of finite instances.

An *ontology-mediated query* (OMQ) is a pair $Q = (\Sigma, q)$ with Σ a finite set of rules and q a BCQ. The answer of Q over a dataset D is true if $\Sigma \cup D \models q$, and false otherwise. A *datalog rewriting* of an OMQ (Σ, q) is a of the form (Π, Q) where Π is a datalog program and Q is a nullary predicate, such that for any dataset D, $\Sigma \cup D \models q$ iff $\Pi \cup D \models \mathsf{Q}$. We call it a *strong datalog rewriting* if additionally $\Sigma \cup \{q\} \models \Pi$. Q is the query predicate of q and sometimes denoted as Q_q. When Π consists of only non-recursive datalog rule with Q in their heads, the rewriting is a UCQ rewriting.

The UCQ rewriting of OMQs can be obtained through the notion of piece unification for a given BCQ q and a rule r [16]. First, for a subset q' of q, a variable occurring both in q' and in $q \setminus q'$ is a separating variable for q' in q. A *piece unifier* of q and r is a triple $\mu = (B, H, \tau)$, where $\emptyset \subset B \subseteq q$, $H \subseteq \mathsf{head}(r)$, and τ is a most general unifier between B and H such that the following condition is satisfied for each $z \in \mathsf{ext}(r)$: If a term t ($t \neq z$) in $B \cup H$ is unified with z, i.e., $z\tau = t\tau$, then t is not a separating variable for B in q.

For a set X of variables and a substitution σ, $\sigma|_X$ denotes the restricted substitution to domain X. Another substitution σ' is a *safe extension* of $\sigma|_X$ on a set of variables Y, if σ' coincides with $\sigma|_X$ and for each $y \in Y$, $y\sigma'$ is a fresh variable.

3 Datalog Rewriting for OMQs

In this section, we will introduce a compact datalog rewriting approach, based on the notion of unfolding for existential rules [24]. A rule r can be unfolded by a rule r' if there exists a piece unifier $\mu = (B, H, \tau)$ of $\mathsf{body}(r)$ and r', and the result of unfolding r by r' with μ is the following rule denoted $\mathsf{unf}^{\mu}(r, r')$:

$$\exists \boldsymbol{z}.[\mathsf{head}(r)\tau'' \cup \mathsf{head}(r')\tau'] \leftarrow (\mathsf{body}(r) \setminus B)\tau \cup \mathsf{body}(r')\tau' \qquad (1)$$

where τ' is a safe extension of $\tau|_{\mathsf{var}(H)}$ on $\mathsf{var}(r') \setminus \mathsf{var}(H)$, τ'' is a safe extension of $\tau|_{\mathsf{fr}(r)}$ on $\mathsf{ext}(r)$, and $\boldsymbol{z}$ consists of all the variables in the head but not in the body. The result of exhaustive unfolding for Σ is denoted $\mathsf{unfold}(\Sigma)$.

Note that if a strong datalog rewriting exists for an OMQ, then there exists a subset of its unfolding that consists of datalog rules and is such a datalog rewriting.

Lemma 1. *For an OMQ (Σ, q), a strong datalog rewriting exists iff there exists a finite subset $\Pi \subseteq \mathsf{unfold}(\Sigma \cup \{q\})$ such that (Π, Q_q) is such a datalog rewriting.*

Clearly, a naive method to compute a datalog rewriting using the above unfolding is impractical, as the datalog rules obtained from unfolding can be very large (indeed, are often of unbounded sizes). In what follows, we introduce a practical approach for datalog rewriting by breaking up long datalog rules into smaller ones. It is known that every OMQ (Σ, q) can be transformed into an OMQ (Σ', q) whose rule heads have single atoms in a way that preserves query answering [8]. For convenience, in what follows, we assume Σ consists of rules of single-atom heads. As a first step, we present an alternative operator, which we simply call rewriting, between two existential rules.

Definition 1. *For two rules r, r' and a piece unifier $\mu = (B, H, \tau)$ of body(r) and r', the result of rewriting r by r' with μ, denoted $\mathsf{rew}^{\mu}(r, r')$, consists of the following rules*

$$\mathsf{P}(\boldsymbol{x}_{r'})\tau \leftarrow \mathsf{body}(r')\tau', \tag{2}$$

$$\exists \boldsymbol{z}.\mathsf{R}(\boldsymbol{v}) \leftarrow (\mathsf{body}(r) \setminus B)\tau \cup \{\mathsf{P}(\boldsymbol{x}_{r'})\tau\}, \tag{3}$$

$$\alpha \leftarrow \mathsf{R}(\boldsymbol{v}) \textit{ for each } \alpha \in \mathsf{head}(r)\tau'' \cup \mathsf{head}(r')\tau', \tag{4}$$

$$\exists \boldsymbol{z}.[\mathsf{head}(r)\tau'' \cup \mathsf{head}(r')\tau'] \leftarrow (\mathsf{body}(r) \setminus B)\tau \cup \{\mathsf{P}(\boldsymbol{x}_{r'})\tau\}, \tag{5}$$

where τ', τ'' are as in Equation (1), $\boldsymbol{v}$ is a tuple of variables in $\mathsf{head}(r)\tau'' \cup \mathsf{head}(r')\tau'$, $\boldsymbol{z}$ consists of all the variables in the head but not in the body, and P and R are fresh predicates with arities $|\boldsymbol{x}_{r'}|$ and $|\boldsymbol{v}|$.

Note that rule (5) can be captured by rules (3) and (4), yet it is included for the correctness of rewriting in some cases. We mark rule (5) as *temporary* which will be deleted after the rewriting is completed, and we will discuss in next section when the generation of such temporary rules are not necessary. Also, for rules of the form (3) that are not datalog rules, we also mark them for deletion.

We can use $\mathsf{rew}^{\mu}(r, r')$ to replace $\mathsf{unf}^{\mu}(r, r')$ in the unfolding of rules and it is not hard to see that the resulting set of rules are equivalent w.r.t. query answering. Indeed, if we allow fresh predicates P and R to be further unfolded, then the rewriting process generates strictly more rules than $\mathsf{unfold}(\Sigma)$. However, it is clear that allowing the unfolding of fresh predicates forfeits their purpose, as they were introduced to break up long rules and their unfolding simply reverses it. Hence, it is natural to disallow unfolding of such fresh predicates.

Furthermore, it also does not make sense to introduce a different fresh predicate each time when the "same" piece unifier, i.e., up to variable renaming, is applied. This is achieved through a label function $\lambda(\cdot)$ that maps each fresh predicate to a set of atoms. Formally, for predicates P and R in Definition 1, $\lambda(\mathsf{P}) = H\tau = \mathsf{head}(r')\tau'$ and $\lambda(\mathsf{R}) = \gamma(\mathsf{head}(r))\tau'' \cup \gamma(\mathsf{head}(r'))\tau'$, where $\gamma(\mathsf{A}(\boldsymbol{x})) = \mathsf{A}(\boldsymbol{x})$ if A is a (non-fresh) predicate occurring in the initial rules and otherwise $\gamma(\mathsf{A}(\boldsymbol{x})) = \lambda(\mathsf{A})$.

Finally, special handling for rewriting a query q by another rule r' is needed. In particular, the handling of head atoms is simpler, such that $\mathsf{rew}^{\mu}(q, r')$ replaces rules (3)–(5) with the following rules

$$\mathsf{Q} \leftarrow (\mathsf{body}(q) \setminus B)\tau \cup \{\mathsf{P}(\boldsymbol{x}_{r'})\tau\}, \tag{6}$$

$$\mathsf{Q} \leftarrow (\mathsf{body}(q) \setminus B)\tau \cup \mathsf{body}(r')\tau'. \tag{7}$$

Again, rule (7) can be captured by rules (2) and (6), yet it is included for the correctness of rewriting in some cases. It is also temporary rules to be deleted after the rewriting is completed.

We are ready to define the rewriting of a set of existential rules Σ.

Definition 2. *The* rewriting *of a set of existential rules Σ, denoted* $\mathsf{rewrite}(\Sigma)$ *is the (possibly infinite) set of rules obtained by exhaustively applying rewriting between each pair of rules r, r' as in Definition 1 under the following conditions and by deleting temporary and non-datalog rules at the end:*

- *H does not contain any fresh predicate;*
- *If r is a query (i.e., has Q in the head), its rewriting follows (2), (6) and (7);*
- *For each introduced fresh predicate $\mathsf{A} \in \{\mathsf{P}, \mathsf{R}\}$ in an atom of the form $\mathsf{A}(\boldsymbol{x})$, if there is an atom $\mathsf{A}'(\boldsymbol{x}')$ with A' another fresh predicate of the same type introduced in previous rewriting steps such that there is a most general unifier σ between $\lambda(\mathsf{A})$ and $\lambda(\mathsf{A}')$ with $\boldsymbol{x}\sigma = \boldsymbol{x}'\sigma$, then reuse A' to replace A.*
- *Eliminate rule r if there is another rule r' and a homomorphism σ from $\mathsf{body}(r')$ to $\mathsf{body}(r)$ such that a safe extension σ' of σ exists with $\mathsf{head}(r')\sigma' = \mathsf{head}(r)$.*

We use $\mathsf{rewrite}|_R(\Sigma)$ (resp., $\mathsf{rewrite}_R(\Sigma)$) with $R \subseteq \{(2), \ldots, (7)\}$ to denote the variant of $\mathsf{rewrite}(\Sigma)$ where temporary rules in R are not generated (resp., only rules in R are generated) during rewriting.

Example 1. Consider $\Sigma_1 = \{r_1 : \mathsf{B}(y) \leftarrow \mathsf{A}(x, y), r_2 : \exists y.\mathsf{A}(x, y) \leftarrow \mathsf{B}(x)\}$ and $q = \mathsf{Q} \leftarrow \mathsf{A}(x, y) \wedge \mathsf{A}(y, z)$. The (one step) rewriting of r_1 by r_2 results the following rules:

$$r_3 : \mathsf{P}_1(x) \leftarrow \mathsf{B}(x), \quad r_4 : \exists y.\mathsf{R}_1(x, y) \leftarrow \mathsf{P}_1(x), \quad r_5 : \mathsf{A}(x, y) \leftarrow \mathsf{R}_1(x, y),$$
$$r_6 : \mathsf{B}(y) \leftarrow \mathsf{R}_1(x, y), \quad r_7 : \exists y.[\mathsf{A}(x, y) \wedge \mathsf{B}(y)] \leftarrow \mathsf{P}_1(x),$$

where $\lambda(\mathsf{P}_1) = \{\mathsf{A}(x, y)\}$ and $\lambda(\mathsf{R}_1) = \{\mathsf{A}(x, y), \mathsf{B}(y)\}$.

Rewriting q by r_2, by r_1, and then by r_2 leads to (not a complete list of) rules:

$$r_8 : \mathsf{Q} \leftarrow \mathsf{A}(x, y) \wedge \mathsf{P}_1(y), \quad r_9 : \mathsf{Q} \leftarrow \mathsf{A}(x, y) \wedge \mathsf{B}(y),$$
$$r_{10} : \mathsf{P}_2(y) \leftarrow \mathsf{A}(z, y), \quad r_{11} : \mathsf{Q} \leftarrow \mathsf{A}(x, y) \wedge \mathsf{P}_2(y), \quad r_{12} : \mathsf{Q} \leftarrow \mathsf{A}(x, y) \wedge \mathsf{A}(z, y),$$
$$r_{13} : \mathsf{Q} \leftarrow \mathsf{P}_1(x), \quad r_{14} : \mathsf{Q} \leftarrow \mathsf{B}(x).$$

Note that P_1 is reused; also, rules r_{13} and r_{14} cannot be obtained without keeping temporary rules r_9 and r_{12} during the rewriting.

Now we establish the soundness and completeness of our datalog rewriting.

Proposition 1. *For an OMQ $Q = (\Sigma, q)$, let $\Pi = \mathsf{rewrite}(\Sigma \cup \{q\})$ (resp., $\Pi = \mathsf{rewrite}|_{(5)}(\Sigma \cup \{q\})$ or $\Pi = \mathsf{rewrite}|_{(7)}(\Sigma \cup \{q\})$). If Π is finite then (Π, Q_q) is a datalog rewriting of Q.*

Proof (Sketch). We want to show for each dataset D, $\Sigma \cup D \models q$ iff $\Pi \cup D \models \mathsf{Q}_q$. First, we only need to establish "if" direction for $\Pi = \mathsf{rewrite}(\Sigma \cup \{q\})$, which can be checked inductively through the unfolding of rules in Π. In particular, consider the unfolding of (reused) fresh predicates in two rules r_1 and r_2 of the form $\mathsf{Q} \leftarrow B_1 \cup \{\mathsf{A}(\boldsymbol{x})\}$ and

$\mathsf{A}(\boldsymbol{x}) \leftarrow B_2$, where A is a fresh predicate and B_1, B_2 do not contain fresh predicates. Suppose $\lambda(\mathsf{A}) = H(\boldsymbol{y}, \boldsymbol{z})$ with some substitution σ such that $\boldsymbol{y}\sigma = \boldsymbol{x}$, then by the definition of rewriting and unfolding, there are rules (up to variable renaming) $\mathsf{Q} \leftarrow B_1 \cup H'\sigma$ with $H' \subseteq H$ and $H\sigma \leftarrow B_2$ from $\mathsf{unfold}(\Sigma \cup \{q\})$. Note that the predicate reuse condition in Definition 2 ensures that $H'\sigma$ is a piece for unification. Thus, the result of unfolding r_1 by r_2 is in $\mathsf{unfold}(\Sigma \cup \{q\})$. This can be extended to the cases where B_1, B_2 contain fresh predicates.

Also, we only need to show the "only if" direction for $\Pi = \mathsf{rewrite}|_{(5)}(\Sigma \cup \{q\})$ and $\Pi = \mathsf{rewrite}|_{(7)}(\Sigma \cup \{q\})$, and we only need to show for each rule $r \in \mathsf{unfold}(\Sigma \cup \{q\})$ with head Q, r also occurs in $\mathsf{unfold}(\Pi)$ (up to variable renaming). We only demonstrate the proof for $\Pi = \mathsf{rewrite}|_{(7)}(\Sigma \cup \{q\})$. We show this by an induction on forward chaining, i.e., consider an unfolding chain: (i) $r_1, \ldots, r_n \in \Sigma \cup \{q\}$, (ii) for $i \geq 2$, $r_{i,\ldots,1}$ is the result of unfolding r_i by $r_{i-1,\ldots,1}$, and (iii) $r_n = q$. For the first two rules r_1, r_2 in the chain, $r_{2,1}$ is of the form (1), and it has corresponding results of rewriting, rules (5) and (2), denoted $r^{(5)}_{2,1}$ and $r^{(2)}_{2,1}$. The head of $r_{2,1}$ is preserved in $r^{(5)}_{2,1}$ for further unfolding and the body of $r_{2,1}$ can be reconstructed through (unfolding) $r^{(5)}_{2,1}$ and $r^{(2)}_{2,1}$. For each rule r_i ($i \geq 2$), there is a rule $r^{(5)}_{i,\ldots,1}$ in the process of rewriting that rewrites it to $r^{(5)}_{i,\ldots,1}$ and $r^{(2)}_{i,\ldots,1}$, such that the head of $r_{i,\ldots,1}$ is preserved in $r^{(5)}_{i,\ldots,1}$ for further unfolding and the body of $r_{i,\ldots,1}$ can be reconstructed through (unfolding) $r^{(5)}_{i,\ldots,1}$ and $r^{(2)}_{i,\ldots,1}$. Finally, the body of $r_{n,\ldots,1}$ can be reconstructed by unfolding $r^{(5)}_{n,\ldots,1}$ and $r^{(2)}_{i,\ldots,1}$ for all $1 \leq i \leq n$ in Π. That is, $r = r_{n,\ldots,1}$ is in $\mathsf{unfold}(\Pi)$.

4 Datalog Rewritable Classes

The process of rewriting introduced in the previous section does not necessarily terminate; for example, it does not terminate on $\Sigma_2 = \{\exists z.\mathsf{A}(z, x) \leftarrow \mathsf{A}(x, y)\}$. It is not hard to see that the termination issue is caused by the generation of temporary rules. If we disallow the generation of temporary rules, the rewriting always terminates.

Lemma 2. *For a set of rules Σ, the computation of $\mathsf{rewrite}|_{(5),(7)}(\Sigma)$ always terminates and the result is finite.*

This can be seen as follows. Suppose the maximum number of atoms in the initial rule bodies is m. For each rule of the forms (2)–(4) and (6), it has a single atom head and its body has at most m atoms. Also, for each fresh predicate P, $\lambda(\mathsf{P})$ consists of a single head atom of an initial rule (up to variable renaming). Similarly, for each fresh predicate R, $\lambda(\mathsf{R})$ consists of at most m head atoms of initial rules, as each rewriting step may add one atom to the head and rule (3) can be rewritten at most m times (noting that unfolding of fresh predicates is disallowed).

In what follows, we discuss several classes of rules on which (finite) datalog rewritings can be obtained by restricting the generation temporary rules. We start by defining a novel class called separable rule sets, which is through adapting a marking procedure from [17].

We use z_r to denote an existential variable z in rule r. For a set of rules Σ, an atom α and a variable x occurring at position i in α, we mark the position with a minimum set of variables $M(i, \alpha)$ satisfying the following conditions: (i) if $\mathsf{head}(r) = \alpha$ for some

$r \in \Sigma$ with $x \in \mathsf{ext}(r)$ then $M(i,\alpha) = \{x_r\}$; (ii) if $\mathsf{head}(r) = \alpha$ for some $r \in \Sigma$ with $x \in \mathsf{fr}(r)$ then $M(i,\alpha)$ is the intersection of all $M(j,\beta)$ s.t. $\beta \in \mathsf{body}(r)$ and x occurs at position j in atom β; (iii) if $\alpha \in \mathsf{body}(r)$ then $M(i,\alpha)$ is the union of $M(i,\mathsf{head}(r'))$ for all $r' \in \Sigma$ s.t. r can be unfolded by r' with α unified with $\mathsf{head}(r')$.

Definition 3. *A set of rules Σ is* separable *if for each rule $r \in \Sigma$, there do not exists a variable x shared by two body atoms such that the intersection of all $M(i,\alpha)$, for each atom $\alpha \in \mathsf{body}(r)$ and each position i that x occurs in α, is non-empty.*

It is not hard to see that Σ_2 is separable.

The class of shy rule sets is proposed in [17]. Intuitively, a set of rules is shy if no existential variable occurs in a position of a relation where a join variable occurs in a rule body, neither can two variables occurring in such (existential-variable) positions of the body also occur in the same head atom.

Proposition 2. *Each rule set that is shy is also separable.*

This can be seen from that any rule set violating separability also violates Condition 1 of shyness [17]. However, the converse of the proposition is not necessarily true and $\Sigma_2 \cup \{\mathsf{B}(x,y) \leftarrow \mathsf{A}(x,u) \wedge \mathsf{A}(y,v)\}$ is such a case.

Proposition 3. *For a OMQ (Σ,q) such that $\Sigma \cup \{q\}$ is separable,* $\mathsf{rewrite}_{(2),(6)}(\Sigma \cup \{q\})$ *is a datalog rewriting of (Σ,q).*

Intuitively, the proof is based on the fact that the unfolding of rules in $\Sigma \cup \{q\}$ only involves a single atom in the body each time, and thus the result of unfolding can be reconstructed from the result of rewriting even if the bodies are separated.

The class of sticky rule sets is introduced in [6], which are shown to be first-order rewritable. The key idea is that fresh variable generated during backward chaining cannot occur in two separate atoms [21]. The two classes of sticky and separable rules are incomparable; for instance, $\Sigma_3 = \{\mathsf{A}(x,y) \leftarrow \mathsf{A}(x,z) \wedge \mathsf{A}(z,y)\}$ is separable but not sticky, and $\Sigma_1 \cup \{\mathsf{C}(x,y,z) \leftarrow \mathsf{A}(x,y) \wedge \mathsf{A}(y,z)\}$ (where Σ_1 is from Example 1) is sticky but not separable.

Proposition 4. *For a OMQ (Σ,q) such that Σ is sticky,* $\mathsf{rewrite}_{(2),(6),(7)}(\Sigma \cup \{q\})$ *is a datalog rewriting of (Σ,q).*

Indeed, the proposition holds for all finite unification sets [2], as the backward chaining always terminates on such rule sets and hence $\mathsf{rewrite}_{(2),(6),(7)}(\Sigma \cup \{q\})$ (in particular, rules of the form (7)) is finite. Moreover, if we disallow the reuse of fresh predicates, the resulting datalog rewriting is a non-recursive one.

Finally, we are also interested in the class of acyclic rule sets, for which various notions of acyclicity have been proposed [11], which guarantees the termination of forward chaining (i.e., on certain chase procedures).

Proposition 5. *For a OMQ (Σ,q) such that Σ is acyclic,* $\mathsf{rewrite}|_{(7)}(\Sigma \cup \{q\})$ *is a datalog rewriting of (Σ,q).*

It can be seen from that the acyclicity conditions guarantee the Skolem chase terminates on critical instances [11], which avoids the generation of infinitely many Skolemised terms. In our case, it avoids the generation of infinitely many existential variables in the heads of rules (5), whereas the sizes of their bodies are always bounded. Hence, the number of such rules is finite, and $\mathsf{rewrite}|_{(7)}(\Sigma \cup \{q\})$ is finite.

5 A Rewriting Algorithm

In this section, we introduce a practical algorithm for computing datalog rewritings of the form $\mathsf{rewrite}|_{(5)}(\Sigma)$, which can be configured to compute $\mathsf{rewrite}|_{(5),(7)}(\Sigma)$ and $\mathsf{rewrite}_{(2),(6),(7)}(\Sigma)$. It can also be easily adapted to compute $\mathsf{rewrite}|_{(7)}(\Sigma)$. Inspired by [12], we compute a decomposed representation of the heads and bodies of the resulting rules from unfolding, such that (i) the representation is compact due to structure sharing, (ii) the datalog rewriting $\mathsf{rewrite}|_{(5)}(\Sigma)$ can be conveniently extracted from such a representation, and (iii) the complete rules from unfolding (e.g., rule (7)) can be reconstructed via backtracking.

For a set of rules Σ whose maximum number of body atoms is m, we define a *rewriting graph* to be a direct graph (N, E) whose nodes in N are of the form (R, I), where R and I each consists of at most m atoms, and whose edges in E are labelled with piece unifiers. Each node $n = (R, I)$ is associated with two fresh predicates P_n and R_n, and we reuse fresh predicates as in Section 3 through a label function $\lambda(\cdot)$. For each node $n = (R, I)$, let $\boldsymbol{u}_n = \mathsf{fr}(R \leftarrow I)$ and $\boldsymbol{v}_n = \mathsf{var}(R)$. Intuitively, an edge from node $n = (R, I)$ to node $n' = (R', I')$ labelled with $\mu = (B, H, \tau)$ represents a set of rules (i.e., rules (2)–(4) and (6)) obtained during rewriting as follows:

$$\begin{aligned} \mathsf{P}_{n'}(\boldsymbol{u}_{n'}) &\leftarrow I', && \text{corresponding to (2)} \\ \exists \boldsymbol{z}.\mathsf{R}_{n'}(\boldsymbol{v}_{n'}) &\leftarrow (I \setminus B)\tau \cup \{\mathsf{P}_{n'}(\boldsymbol{u}_{n'})\}, && \text{corresponding to (3) and (6)} \\ \alpha &\leftarrow \mathsf{R}_{n'}(\boldsymbol{v}_{n'}) \text{ for each } \alpha \in R', && \text{corresponding to (4)} \end{aligned}$$

where $\boldsymbol{z}$ consists of all the variables in the head but not in the body. For a rewriting graph G, $\mathsf{dlg}(G)$ is the set of datalog rules obtained as above from the edges of G.

Now, we present the algorithm for computing rewriting graphs. Note that similar as the rule elimination in Definition 2, for a new node $n = (R, I)$, if there is an existing node $n' = (R', I')$ and a homomorphism σ from I' to I such that a safe extension σ' of σ exists with $R \subseteq R'\sigma'$, then n should be eliminated. In Algorithm 1, actions with a $*$ are subject to the reuse of fresh predicates and node elimination. Also, τ' and τ'' denote safe extensions of τ as in Equation (1).

In Algorithm 1, we first initialise the nodes to be the pairs of heads and bodies of existing rules (line 2) and use a queue Γ to store the nodes representing rules to be rewritten (line 4 onwards). Since a piece unifier cannot contain a fresh predicate (by Definition 2 and guaranteed in line 7), only original rules or rules of the forms (2), (3), (6) and (7) need to be rewritten, and only original rules or rules of the form (4) with head α can rewrite such rules. Lines 9–21 are the rewriting process, with lines 10–12 corresponding to the rewriting of an original rule or a rule (2), and lines 13–15 corresponding to that of rules (3) and (6). The rewriting produces 2 new node (subject to predicate reuse and node elimination): node n'' represents the generated rules (2)–(4) and (6), and is added to N. It is added to the queue for the further rewriting of rule (2); node $n^\dagger$ is not added to N but is added to the queue for the further rewriting of rules (3) and (6). Note that temporary rules of the form (7) need to be generated to ensure the correctness. The generation of their representations is achieved through backtracking (lines 22–27). It is not hard to verify that rule (7) corresponds to $\exists \boldsymbol{z}.\mathsf{R}_n(\boldsymbol{v}_n) \leftarrow S$.

We establish the correctness of Algorithm 1 as follows.

Algorithm 1: Compute Rewriting Graphs

```
input  : A set of existential rules $\Sigma$
output: A rewriting graph $(N, E)$
1  begin
2      initialise $N := \{(\mathsf{head}(r), \mathsf{body}(r)) \mid r \in \Sigma\}$ and $E := \emptyset$;
3      assign* fresh predicates $\mathsf{P}_n, \mathsf{R}_n$ to each $n = (R, I) \in N$ with $\lambda(\mathsf{P}_n) = \lambda(\mathsf{R}_n) = R$;
4      let $\Gamma$ be a queue and initially $\Gamma := N$;
5      while $\Gamma \neq \emptyset$ do
6          take $n = (R, I)$ popped from $\Gamma$;
7          foreach $n' = (R', I') \in N$ and $\alpha \in R'$ containing no fresh predicate do
8              consider rule $r := \alpha \leftarrow I'$;
9              foreach piece unifier $\mu = (B, H, \tau)$ of $I$ and $r$ do
10                 if $n \in N$ then
11                     take $n'' := (R'', I'\tau')$ where $R'' = R\tau'' \cup \{\alpha\tau'\}$ if $R$ is a singleton;
                         otherwise $R'' = \{\mathsf{P}_n(\boldsymbol{u}_n)\tau'', \alpha\tau'\}$;
12                     assign* a fresh predicate $\mathsf{R}_{n''}$ and let $\lambda(\mathsf{R}_{n''}) := \lambda(\mathsf{P}_n)\tau'' \cup \{\alpha\tau'\}$;
13                 else
14                     take $n'' := (R'', I'\tau')$ where $R'' = R\tau'' \cup \{\alpha\tau'\}$ if $R$ is a singleton;
                         otherwise $R'' = \{\mathsf{R}_n(\boldsymbol{v}_n)\tau'', \alpha\tau'\}$;
15                     assign* a fresh predicate $\mathsf{R}_{n''}$ and let $\lambda(\mathsf{R}_{n''}) := \lambda(\mathsf{R}_n)\tau'' \cup \{\alpha\tau'\}$;
16                 assign* a fresh predicate $\mathsf{P}_{n''}$ and let $\lambda(\mathsf{P}_{n''}) := \alpha\tau'$;
17                 add* $n''$ to $N$ and $(n, n'')$ to $E$ labelled with $\mu$; push* $n''$ to $\Gamma$;
18                 take $n^\dagger := (R'', (I \setminus B)\tau \cup \{\mathsf{P}_{n''}(\boldsymbol{u}_{n''})\})$;
19                 assign* a fresh predicate $\mathsf{R}_{n^\dagger}$ and let $\lambda(\mathsf{R}_{n^\dagger}) = \lambda(\mathsf{R}_n)\tau'' \cup \{\alpha\tau'\}$;
                   push* $n^\dagger$ to $\Gamma$;
20         initialise $p := n$ and $S := I$;
21         while $p$ has a parent $n' = (R', I')$ and $n' \neq n$ do
22             suppose the path from $n'$ to $n$ has labels $(B_1, H_1, \tau_1), \ldots, (B_m, H_m, \tau_m)$;
23             let $p := n'$ and $S := S \cup (I' \setminus B_1)\tau_1 \cdots \tau_m$;
24             take $n^\ddagger := (R, S)$ and reuse the predicate $\mathsf{R}_{n^\ddagger} := \mathsf{R}_n$; push* $n^\ddagger$ to $\Gamma$;
25     return $(N, E)$;
```

Theorem 1. *For a set of rules Σ, if $\mathsf{rewrite}|_{(5)}(\Sigma)$ is finite then Algorithm 1 computes a finite rewriting graph G such that $\mathsf{dlg}(G) \equiv \mathsf{rewrite}|_{(5)}(\Sigma)$.*

6 Experiments

We have implemented a prototype system Drewer (Datalog REWriting for Existential Rules), where the piece unification implementation is adapted from the first-order rewriting system Graal [13], and used RDFox[1] as the datalog engine. We conducted two set of experiments to evaluate the performance of our system. All experiments were performed on a desktop machine with a processor at 3.30 GHz and 8GB of RAM.

[1] `https://www.cs.ox.ac.uk/isg/tools/RDFox/`

In the first set of experiments, we compared our system with state-of-the-art query rewriting systems regarding the compactness and efficiency of query rewriting. In particular, Graal is a first-order rewriting system for existential rules, Iqaros [23] is a first-order rewriting system for OWL 2 DL featured in its rewriting minimisation, and Rapid [22] is a datalog rewriting system for DL-Lite and $\mathcal{EL}$ based on optimised resolution. For benchmark ontologies and queries, we selected 3 commonly used ontologies [13], which are DL-Lite versions of Adolena (A), OpenGALEN2 (G), and StockExchange (S). Each of the ontologies comes with 5 conjunctive queries.

Table 1 records the sizes and times for query rewriting, where sizes are measured by the numbers of atoms and the times are in milliseconds.

Ontology	Query	Rewriting Size			Rewriting Time			
		UCQ	Drewer	Graal	Drewer	Graal	Rapid	Iqaros
A	q_1	27	54	2	26	15	9	3
	q_2	50	32	2	3	3	5	3
	q_3	104	32	1	3	8	6	18
	q_4	224	35	2	2	3	15	10
	q_5	624	37	1	3	2	20	312
G	q_1	2	3	1	62	6	5	5
	q_2	1152	1276	1	70	50	41	4040
	q_3	488	80	5	99	45	20	5779
	q_4	147	155	1	60	3	2	402
	q_5	324	59	19	68	34	9	7206
S	q_1	6	10	1	11	7	6	0
	q_2	2	30	1	1	0	2	3
	q_3	4	15	1	3	1	2	11
	q_4	4	30	1	0	1	2	8
	q_5	8	16	1	1	5	3	117

Table 1: Comparison on query rewriting.

Both Rapid and Iqaros output minimal rewritings of the same sizes recorded under "UCQ". To achieve efficiency as well as compactness in rewriting, Graal makes use of the so called rule compilation, which leads to its small rewriting output. Yet, the results of rewriting produced by Graal are not standard UCQs, which have to be evaluated through a special homomorphism mechanism, and thus cannot be directly handled by DBMSs for datalog engines. We can see that the benefit of datalog rewriting w.r.t. sizes compared to UCQ rewritings becomes obvious when large UCQ rewritings are generated ($\geq$50), with two exceptions q_2 and q_4 on G. This is because the ontology has a large number of simple axioms. For rewriting times, the performance of our system is comparable to Graal and Rapid, with an exception on G due to the reason discussed before.

In the second set of experiments, we compared our system with existing end-to-end in-memory (as our system is in-memory) query answering systems regarding time efficiency. We use RDF4j as the data store of Graal. Another system we compared is Pagoda [25], which is not a standard rewriting system, but a hybrid system combining a datalog engine and an OWL 2 reasoner. To compare with Graal and Iqaros, we used the DL-Lite versions of two widely used ontologies, LUBM and Reactome [25].

As Reactome comes with over 100 queries, we randomly selected 5 queries. Table 2 records the times (in seconds) on pre-processing (Pre) and query evaluation (Eval).

Ontology	Query	Drewer		Graal		Pagoda		Iqaros	
		Pre	Eval	Pre	Eval	Pre	Eval	Pre	Eval
LUBM	q_1	22.94	1.49	200.74	4.73	121.98	0.82	22.3	0.00
	q_2		0.46		13.87		0.00		0.00
	q_3		0.03		0.00		0.95		0.01
	q_4		1.48		0.00		0.39		0.02
	q_5		1.49		0.18		0.65		0.03
Reactome	q_1	7.83	0.04	59.80	5.15	13.73	0.03	8.46	0.01
	q_2		0.07		5.35		0.02		0.00
	q_3		0.08		5.03		0.01		0.00
	q_4		0.07		5.05		0.01		0.01
	q_5		0.07		5.12		0.02		0.00

Table 2: Comparison on query answering.

Our system showed superior performance in most cases compared to Graal, which is possibly due to the special homomorphism mechanism implemented in it. both of which took significantly more time in pre-processing. Considering only the query evaluation times, Drewer is comparable to Pagoda, whereas Pagoda took more pre-processing time for the materialization of upper-bound and lower-bound datalog programs. Iqaros shows best performance among all the systems, while the performance of Drewer is comparable on Reactome. Note that Iqaros is specialised for first-order rewritable OWL 2 DL ontologies, whereas Drewer is for more general existential rules.

7 Conclusion

In this paper, we have presented a novel approach to datalog rewriting for existential rules and introduced a new datalog rewritable class, the separable rule sets, which generalises the class of shy rule sets. Furthermore, we presented a practical algorithm for computing datalog rewritings and established its correctness. Finally, we implemented and evaluated our prototype system Drewer, which is to the best of our knowledge, the first datalog rewriting system for existential rules. Our system demonstrated superior or comparable performance compared to state-of-the-art rewriting and query answering systems.

For future work, we are working on identifying a more general class of datalog rewritable existential rules, optimising our rewriting algorithm and implementation, and conducting further evaluation on more complex ontologies and queries.

References

1. S. Ahmetaj, M. Ortiz, and M. Simkus. Rewriting guarded existential rules into small datalog programs. In *Proc. of ICDT-18*, pp. 4:1–4:24, 2018.
2. J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

3. M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. Ontology-based data access: a study through disjunctive datalog, csp, and MMSNP. In *Proc. of PODS-13*, pp. 213–224, 2013.
4. A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.
5. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.
6. A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.
7. T. Eiter, M. Ortiz, M. Simkus, T.-K. Tran, and G. Xiao. Query rewriting for horn-shiq plus rules. In *Proc. of AAAI-12*, 2012.
8. G. Gottlob, G. Orsi, and A. Pieris. Query rewriting and optimization for ontological databases. *ACM Trans. Database Syst.*, 39(3):25:1–25:46, 2014.
9. G. Gottlob, S. Rudolph, and M. Simkus. Expressiveness of guarded existential rule languages. In *Proc. of PODS-14*, pp. 27–38, 2014.
10. G. Gottlob and T. Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In *Proc. of KR-12*, 2012.
11. B. C. Grau, I. Horrocks, M. Krötzsch, C. Kupke, D. Magka, B. Motik, and Z. Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *J. Artif. Intell. Res.*, 47:741–808, 2013.
12. P. Hansen, C. Lutz, I. Seylan, and F. Wolter. Efficient query rewriting in the description logic EL and beyond. In *Proc. of IJCAI-15*, pp. 3034–3040, 2015.
13. M. König, M. Leclère, and M.-L. Mugnier. Query rewriting for existential rules with compiled preorder. In *Proc. of IJCAI-15*, pp. 3106–3112, 2015.
14. M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. Sound, complete and minimal ucq-rewriting for existential rules. *Semantic Web*, 6(5):451–475, 2015.
15. R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in DL-Lite. In *Proc. of KR-10*, 2010.
16. M. Leclère, M.-L. Mugnier, and F. Ulliana. On bounded positive existential rules. In *Proc. of DL-16*, 2016.
17. N. Leone, M. Manna, G. Terracina, and P. Veltri. Efficiently computable datalog∃ programs. In *Proc. of KR-12*, 2012.
18. H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable query answering and rewriting under description logic constraints. *J. Applied Logic*, 8(2):186–209, 2010.
19. R. Rosati and A. Almatelli. Improving query answering over dl-lite ontologies. In *Proc. of KR-10*, 2010.
20. G. Stefanoni, B. Motik, and I. Horrocks. Introducing nominals to the combined query answering approaches for EL. In *Proc. of AAAI-13*, 2013.
21. M. Thomazo. *Conjunctive Query Answering Under Existential Rules - Decidability, Complexity, and Algorithms.* PhD thesis, Montpellier 2 University, France, 2013.
22. D. Trivela, G. Stoilos, A. Chortaras, and G. B. Stamou. Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Semant.*, 33:30–49, 2015.
23. T. Venetis, G. Stoilos, and V. Vassalos. Rewriting minimisations for efficient ontology-based query answering. In *Proc. of ICTAI-16*, pp. 1095–1102, 2016.
24. Z. Wang, K. Wang, and X. Zhang. Forgetting and unfolding for existential rules. In *Proc. of AAAI-18*, pp. 2013–2020, 2018.
25. Y. Zhou, B. C. Grau, Y. Nenov, M. Kaminski, and I. Horrocks. Pagoda: Pay-as-you-go ontology query answering using a datalog reasoner. *J. Artif. Intell. Res.*, 54:309–367, 2015.