# Query Answering for Existential Rules via Efficient Datalog Rewriting

**Zhe Wang**[1] , **Peng Xiao**[1*] , **Kewen Wang**[1] , **Zhiqiang Zhuang**[2] and **Hai Wan**[3]
[1]Griffith University, Australia
[2]Tianjin University, China
[3]Sun Yat-sen University, China
{zhe.wang, k.wang}@griffith.edu.au, peng.xiao@griffithuni.edu.au, zhuang@tju.edu.cn,
wanhai@mail.sysu.edu.cn

## Abstract

Existential rules are an expressive ontology formalism for ontology-mediated query answering and thus query answering is of high complexity, while several tractable fragments have been identified. Existing systems based on first-order rewriting methods can lead to queries too large for DBMS to handle. It is shown that datalog rewriting can result in more compact queries, yet previously proposed datalog rewriting methods are mostly inefficient for implementation. In this paper, we fill the gap by proposing an efficient datalog rewriting approach for answering conjunctive queries over existential rules, and identify and combine existing fragments of existential rules for which our rewriting method terminates. We implemented a prototype system Drewer, and experiments show that it is able to handle a wide range of benchmarks in the literature. Moreover, Drewer shows superior or comparable performance over state-of-the-art systems on both the compactness of rewriting and the efficiency of query answering.

## 1 Introduction

Existential rules (a.k.a. Datalog$\pm$ and tuple generating dependencies) [Baget *et al.*, 2011; Calì *et al.*, 2012b] 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 $\mathcal{EL}$ [Calì *et al.*, 2012a], which underlay the OWL 2 Profiles. This makes existential rules an appealing formalism for ontology-mediated query answering [Bienvenu *et al.*, 2014]. While query answering is undecidable over the full formalism, several interesting fragments have been proposed [Baget *et al.*, 2011; Calì *et al.*, 2012b; Calì *et al.*, 2013; Leone *et al.*, 2019] that support tractable query answering.

There are two major approaches for query answering over ontologies expressed in existential rules or Horn description logics, the chase-based approach and the query rewriting approach. The chase-based approach relies on the termination of the chase procedure (a.k.a. forward chaining), and it is shown that computing a full chase can be rather inefficient when query answers depend only on a small portion of it [Benedikt *et al.*, 2018]. On the other hand, the query rewriting approach does not require expanding the data. Given an ontology $\Sigma$ and a query $q$, a rewriting method transforms them into another query $q_\Sigma$, which is sometimes in a different query formalism, such that answering $q_\Sigma$ 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 [Pérez-Urbina *et al.*, 2010; Eiter *et al.*, 2012; Zhou *et al.*, 2015; Venetis *et al.*, 2016], particularly for DL-Lite and $\mathcal{EL}$ [Kontchakov *et al.*, 2010; Stefanoni *et al.*, 2013; Trivela *et al.*, 2015; Hansen *et al.*, 2015; Bienvenu *et al.*, 2017], it is challenging to extend them to more general existential rules, which allow predicates of arbitrary arities (instead of only unary and binary predicates) and variable permutations in the rules.

Existing query rewriting systems for existential rules are typically based on first-order rewritings [Gottlob *et al.*, 2014a; König *et al.*, 2015a; König *et al.*, 2015b], i.e., $q_\Sigma$ 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 [Calì *et al.*, 2012b]. 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 [Rosati and Almatelli, 2010; Bienvenu *et al.*, 2017].

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 [Hansen *et al.*, 2015]. 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 [Gottlob *et al.*, 2014b]. However, existing research on datalog rewrit-

[*]Corresponding Author

ing of existential rules are mostly theoretical [Gottlob and Schwentick, 2012; Bienvenu *et al.*, 2014] (refer to [Ahmetaj *et al.*, 2018] for a detailed discussion). While several algorithms and systems have been developed for datalog rewriting for various description logics [Eiter *et al.*, 2012; Trivela *et al.*, 2015; Hansen *et al.*, 2015], very few systems have been developed for datalog rewriting over more general existential rules. A notable exception is ChaseGoal [Benedikt *et al.*, 2018], which however, relies on the termination of the chase procedure.

In this paper, we fill the gap by presenting both a practical approach and a prototype system for datalog rewriting and query answering over a wide range of ontologies expressed in existential rules. Our algorithm is based on the notion of unfolding [Wang *et al.*, 2018] and to achieve compactness of rewriting, we separate the results of unfolding into short rules by introducing the so-called separating predicates and reusing such predicates when possible. While such a rewriting process may not terminate, we move on to identify classes of ontologies where the rewriting process terminates, introducing a class by combining existing well-accepted classes. And we introduce an efficient algorithm for computing the datalog rewritings. Finally, we implemented a prototype system, Drewer, and experiments show that it is able to handle a wide range of benchmarks in the literature. Moreover, Drewer shows superior or comparable performance over state-of-the-art systems on both the compactness of rewriting and the efficiency of query answering.

## 2 Preliminaries

We assume standard first-order logic notions, such as predicates, atoms, facts (i.e., ground atoms), formulas, entailment ($\models$) and equivalence ($\equiv$). For a set of formulas $\Phi$, $[\Phi]$ denotes the set of predicates in $\Phi$. For convenience, we sometimes identify a finite set of atoms with the conjunction of its atoms where all the variables are existentially quantified, and vice versa. A *dataset* is a finite set of facts (i.e., ground atoms).

For a set of atoms $A$, $\mathsf{var}(A)$ denotes the set of variables in $A$. By assuming a fixed order on all the variables, for a set of variables $V$, we may refer to the vector of $V$; also, for denotational simplicity, we may use a vector to represent the set of its elements. A substitution $\sigma$ is a functional mapping between two sets of terms such that $t\sigma = t$ if $t$ is a constant or is not in the domain of $\sigma$; and it naturally extends to (sets of) atoms and formulas. A *unifier* between two sets of atoms $A$ and $A'$ is a substitution $\tau$ such that $A\tau = A'\tau$; and $\tau$ is a *most general unifier* (MGU) if for each unifier $\tau'$ between $A$ and $A'$, there exists a substitution $\sigma$ such that $\tau' = \tau\sigma$.

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

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

where $\vec{x}$, $\vec{y}$ and $\vec{z}$ are pairwise disjoint vectors of variables, and $\varphi(\vec{x},\vec{z})$ and $\psi(\vec{x},\vec{y})$ are conjunctions of atoms containing only variables from respectively $\vec{x}\cup\vec{z}$ and $\vec{x}\cup\vec{y}$. Variables in $\vec{x}$ are *frontier variables* and those in $\vec{z}$ are *existential variables*, and we use $\vec{x}_r$ and $\vec{z}_r$ to emphasis they are for rule $r$. Formula $\varphi$ is the *head* of the rule $r$, denoted $\mathsf{head}(r)$, and formula $\psi$ is the *body* of $r$, denoted $\mathsf{body}(r)$ (again, they can be seen as sets of atoms). For brevity, universal quantifiers in a rule are often omitted, and we assume each rule employs a distinct set of variables. A rule $r$ is *implied* by another rule $r'$ if $r' \models r$. Two sets of rules $\Sigma_1$ and $\Sigma_2$ are *equivalent on* a set of predicates $P$, denoted $\Sigma_1 \equiv_P \Sigma_2$, if for each pair of dataset $D$ and fact $\alpha$ over $P$, $\Sigma_1 \cup D \models \alpha$ iff $\Sigma_2 \cup D \models \alpha$. It is known that a set of existential rules $\Sigma$ can be transformed into a set of rules $\Sigma'$ whose heads are singletons in a way that preserves query answering [Gottlob *et al.*, 2014a]. In what follows, we assume $\Sigma$ consists of such rules and use $\mathsf{head}(r)$ to denote the single atom in the rule head.

A *datalog* rule $r$ is an existential rule with a single head atom and $\vec{z}_r$ is empty. A *fact-preserving datalog rewriting* (FPDR) of a set of existential rules $\Sigma$ is a datalog program $\Pi$ that preserves fact derivation, that is, $\Pi \equiv_{[\Sigma]} \Sigma$; and it is a *strong* FPDR if additionally, $\Sigma \models \Pi$.

A *conjunctive query* (CQ) $q(\vec{x})$ can be conveniently represented as a datalog rule $\mathsf{Q}(\vec{x}) \leftarrow \varphi(\vec{x},\vec{y})$ where $\mathsf{Q}$ is a predicate with arity $|\vec{x}|$. When $\vec{x}$ is empty the query is Boolean, called *BCQ* and denoted $q$. A *UCQ* is a set of CQs with the same head. An *ontology-mediated query* (OMQ) is of the form $Q = \Sigma \cup \{q(\vec{x})\}$ with $\Sigma$ a finite set of existential rules and $q(\vec{x})$ a CQ. A tuple $\vec{a}$ with the same arity as $\vec{x}$ is an *answer* to $Q$ over a dataset $D$ if $Q \cup D \models \mathsf{Q}(\vec{a})$.

Datalog rewriting for an OMQ is relaxed to preserve only the query answers; in particular, a *query-preserving datalog rewriting* (QPDR) of an OMQ $Q = \Sigma \cup \{q(\vec{x})\}$ is a datalog program $\Pi_{\mathsf{Q}}$ such that $\Pi_{\mathsf{Q}} \equiv_{\{\mathsf{Q}\}} \Sigma$. Clearly, an FPDR of a OMQ is also a QPDR, but the converse does not necessarily hold. When $\Pi_{\mathsf{Q}}$ is a UCQ, it is a *UCQ rewriting*. Answering CQs can be reduced to that of BCQs and hence w.l.o.g. we consider only BCQs in this paper.

Existing works on UCQ rewriting of OMQs are (essentially) based on the notion of piece unification for two sets of atoms $B, H$ w.r.t. a set of variables $V$ from $H$ [Baget *et al.*, 2011; Leclère *et al.*, 2016]. A *piece unifier* of $B$ and $H$ w.r.t. $V$ is a tuple $\mu = (B', H', \tau)$, where $\emptyset \subset B' \subseteq B$, $H' \subseteq H$, and $\tau$ is a MGU between $B'$ and $H'$ such that for each $v \in V$, $v$ can only be unified with variables $u$ (i.e., $v\tau = u\tau$) occurring in $B'$ [1]. For a CQ $q$ to be rewritten by a rule $r$, one would take $B = \mathsf{body}(q)$, $H = \{\mathsf{head}(r)\}$, and $V$ consisting all the existential variables in $r$; thus, we can omit $V$ and talk about the piece unification of $\mathsf{body}(q)$ and $\mathsf{head}(r)$, and because $H$ is a singleton, the notion can be simplified as $\mu = (B', \tau)$.

## 3 Compact Datalog Rewriting

In this section, we introduce a compact datalog rewriting approach, based on the notion of unfolding for existential rules [Wang *et al.*, 2018]. A rule $r$ can be *unfolded* by a rule $r'$ if there exists a piece unifier $\mu = (B, \tau)$ of $\mathsf{body}(r)$ and $\mathsf{head}(r')$, and the result is $\mathsf{unf}^{\mu}(r, r')$:

$$\exists\vec{z}.[\mathsf{head}(r)\tau' \wedge \mathsf{head}(r')\tau] \leftarrow \bigwedge(\mathsf{body}(r) \setminus B)\tau \wedge \bigwedge \mathsf{body}(r')\tau'$$

[1]It excludes the cases where $v$ is unified with a constant, with a variable in $H$ other than $v$, or with a variable in $B$ shared between atoms inside and outside $B'$.

where $\vec{z}$ consists of all the variables in the head but not in the body, and $\tau'$ is a *safe extension* of $\tau$ by substituting variables $\vec{z}_r \cup \vec{y}_{r'}$ with fresh variables.

**Example 1.** *Let* $\Sigma_{ex1} = \{r_1 : \exists y.\mathsf{A}(x, y) \leftarrow \mathsf{B}(x, z),\ r_2 : \exists z.\mathsf{B}(y, z) \leftarrow \mathsf{A}(x, y)\}$ *and* $q_{ex} = \mathsf{Q} \leftarrow \mathsf{A}(u, v) \wedge \mathsf{A}(v, w)$.

*Then,* $q_{ex}$ *can be unfolded by* $r_1$ *with a piece unifier* $\mu = (\{\mathsf{A}(v, w)\}, \{x \mapsto v, y \mapsto w\})$, *and* $\mathsf{unf}^{\mu}(q_{ex}, r_1) = \exists w.[\mathsf{Q} \wedge \mathsf{A}(v, w)] \leftarrow \mathsf{A}(u, v) \wedge \mathsf{B}(v, z)$. *On the other hand,* $\mu = (\{\mathsf{A}(u, v)\}, \{x \mapsto u, y \mapsto v\})$ *is not a piece unifier, as it does not correctly unify the existential variable* $y$.

Note that the result of unfolding can be simplified when the unified rule heads of $r$ and $r'$ do not share existential variables, i.e., $\mathsf{var}(\mathsf{head}(r)\tau') \cap \mathsf{var}(\mathsf{head}(r')\tau) \cap \vec{z} = \emptyset$. In this case, the two heads can be separated and result in

$$\exists \vec{z_1}.\mathsf{head}(r)\tau' \leftarrow \bigwedge(\mathsf{body}(r) \setminus B)\tau \wedge \bigwedge \mathsf{body}(r')\tau' \quad (*)$$

where $\vec{z_1}$ consists of all the variables in the head but not in the body. Note that $\exists \vec{z_2}.\mathsf{head}(r')\tau \leftarrow \bigwedge(\mathsf{body}(r) \setminus B)\tau \wedge \bigwedge \mathsf{body}(r')\tau'$ is implied by $r'$ and thus is redundant.

For a rule set $\Sigma$, $\mathsf{unfold}(\Sigma)$ is the smallest rule set containing $\Sigma$ such that $\mathsf{unf}^{\mu}(r, r') \in \mathsf{unfold}(\Sigma)$ for each $r, r' \in \mathsf{unfold}(\Sigma)$ and each $\mu$ (disregarding variable renaming).

Towards a datalog rewriting method, we observe that when a strong datalog rewriting exists for a rule set, it can be obtained via unfolding.

**Proposition 1.** *For a set of rules* $\Sigma$, *a strong FPDR of* $\Sigma$ *exists iff a finite subset of* $\mathsf{unfold}(\Sigma)$ *is an FPDR of* $\Sigma$.

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 splitting long datalog rules generated via unfolding into compact ones. 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* $\mu = (B, \tau)$ *of* $\mathsf{body}(r)$ *and* $\mathsf{head}(r')$, *the result of rewriting* $r$ *by* $r'$ *with* $\mu$, *denoted* $\mathsf{rew}^{\mu}(r, r')$, *consists of rule (*) and the following two rules*

$$\mathsf{P}(\vec{x}) \leftarrow \bigwedge \mathsf{body}(r')\tau', \qquad (1)$$

$$\exists \vec{z_1}.\mathsf{head}(r)\tau' \leftarrow \bigwedge(\mathsf{body}(r) \setminus B)\tau \wedge \mathsf{P}(\vec{x}), \qquad (2)$$

*where* $\vec{x} = \vec{x}_r\tau \cup \mathsf{var}(\mathsf{body}(r) \setminus B)\tau \cap \vec{x}_{r'}\tau$, $\mathsf{P}$ *is a fresh predicate with arity* $|\vec{x}|$, *called a* separating predicate, *and* $\vec{z_1}$, $\tau$ *and* $\tau'$ *are as in (*).*

Intuitively, we split the body of (*) by introducing $\mathsf{P}$ to obtain compact rules. Note that rule (*) can be obtained by unfolding (2) by (1), yet it is generated for the correctness of rewriting as we show later. We call rules of the form (*) *auxiliary rules*; they will be deleted after the whole rewriting process is completed.

**Example 2.** *For* $\Sigma_{ex1}$ *and* $q_{ex}$ *in Example 1,* $\mathsf{rew}^{\mu}(q_{ex}, r_1)$ *consists of the following rules:*

$$r_3 : \mathsf{P}(v) \leftarrow \mathsf{B}(v, z), \qquad r_4 : \mathsf{Q} \leftarrow \mathsf{A}(u, v) \wedge \mathsf{P}(v),$$
$$r_5 : \mathsf{Q} \leftarrow \mathsf{A}(u, v) \wedge \mathsf{B}(v, z).$$

Replacing $\mathsf{unf}^{\mu}(r, r')$ with $\mathsf{rew}^{\mu}(r, r')$ for unfolding leads to a set of rules that are equivalent to $\mathsf{unfold}(\Sigma)$ w.r.t. fact derivation (over original predicates) and query answering. Yet allowing the unfolding of separating predicates (i.e., including them in piece unifiers) clearly forfeits their purpose, as they were introduced to split long rules and their unfolding simply reverses the split. Hence, the unfolding of separating predicates must not be allowed.

Furthermore, it is possible to reuse separating predicates. This is achieved through a labelling function $\lambda(\cdot)$ such that $\lambda(\mathsf{P}) = \mathsf{head}(r')\tau$. Intuitively, the label records how $\mathsf{P}$ is introduced (e.g., the head atom involved in the piece unification). When introducing a new separating predicate $\mathsf{P}'$ with the same arity and if $\lambda(\mathsf{P})$ is equivalent to $\lambda(\mathsf{P}')$ up to variable renaming, we *reuse* $\mathsf{P}$ to replace $\mathsf{P}'$.

We are ready to define our datalog rewriting.

**Definition 2.** *The* rewrite chaining *on a rule set* $\Sigma$ *is a sequence of rule sets* $\Sigma^i_{\mathsf{rew}}$ $(i \geq 0)$, *where* $\Sigma^0_{\mathsf{rew}} = \Sigma$, *and* $\Sigma^{i+1}_{\mathsf{rew}} = \Sigma^i_{\mathsf{rew}} \cup \{\, \mathsf{rew}^{\mu}(r, r') \mid r \in \Sigma^i_{\mathsf{rew}},\ r' \in \Sigma \,\}$ *for* $i \geq 0$ *satisfying the following two conditions: (i) separating predicates are reused whenever possible, and (ii) any rule that is implied by another rule is eliminated.*

*The* rewriting *of* $\Sigma$, $\mathsf{rewrite}(\Sigma)$, *is obtained from* $\Sigma^{\infty}_{\mathsf{rew}}$ *by deleting all auxiliary and non-datalog rules.*

**Example 3.** *For* $\Sigma_{ex1}$ *and* $q_{ex}$ *in Examples 1 and 2, the rewriting of* $q_{ex}$ *by* $r_1$ *and* $r_2$ *include additionally the following (non-exhaustive list of) rules:*

$$r_6 : \mathsf{P}'(v) \leftarrow \mathsf{A}(x, v), \qquad r_7 : \mathsf{Q} \leftarrow \mathsf{A}(u, v) \wedge \mathsf{P}'(v),$$
$$r_8 : \mathsf{Q} \leftarrow \mathsf{A}(u, v) \wedge \mathsf{A}(x, v), \qquad r_9 : \mathsf{P}'' \leftarrow \mathsf{B}(u, z),$$
$$r_{10} : \mathsf{Q} \leftarrow \mathsf{P}'', \qquad r_{11} : \mathsf{Q} \leftarrow \mathsf{B}(u, z).$$

*Note that rules* $r_9$ *and* $r_{10}$ *cannot be obtained without keeping auxiliary rules* $r_5$ *and* $r_8$ *during the rewriting.*

We establish the correctness of our rewriting approach. We use $\mathsf{rewrite}^{\mathsf{q}}()$ to denote the variant of $\mathsf{rewrite}()$ where only rules with query or separating predicates in their heads are rewritten (i.e., in Definition 1, $\mathsf{head}(r) = \mathsf{Q}$ or $\mathsf{head}(r) = \mathsf{P}'(\vec{x})$ for some separating predicate $\mathsf{P}'$).

**Proposition 2.** *For a rule set* $\Sigma$, *a BCQ* $q$, *and the OMQ* $Q = \Sigma \cup \{q\}$, $\mathsf{rewrite}(\Sigma)$ *(or* $\mathsf{rewrite}^{\mathsf{q}}(Q)$*) is an FPDR (resp., QPDR) of* $\Sigma$ *(resp.,* $Q$*) whenever* $\mathsf{rewrite}(\Sigma)$ *(resp.,* $\mathsf{rewrite}^{\mathsf{q}}(Q)$*) is equivalent to a finite set of rules.*

## 4 Datalog Rewritable Classes

The rewriting method in the previous section does not necessarily terminate; for example, it does not terminate on $\Sigma_{ex2} = \{\mathsf{C}(x, y) \leftarrow \mathsf{C}(x, z) \wedge \mathsf{C}(z, y)\}$. Yet it terminates on the class of *finite unification sets* (fus) [Baget *et al.*, 2011], for which several concrete classes have been identified, such as the *linear* (lin), *sticky* (stky) [Calì *et al.*, 2012b], and *acyclic graph of rule dependency* (agrd) [Baget *et al.*, 2011].

**Proposition 3.** *For a set of rules* $\Sigma$ *in* fus, $\mathsf{rewrite}(\Sigma)$ *is finite; and for any BCQ* $q$, $\mathsf{rewrite}(\Sigma \cup \{q\})$ *is finite.*

However, $\Sigma_{ex2}$ is not fus. It is not hard to see that the termination issue is caused by the generation of infinitely many auxiliary rules. We use $\mathsf{rewrite}_{\mathsf{a}}()$ to denote the variant of rewrite() where auxiliary rules (of the form (*)) are not generated during rewriting.

**Lemma 1.** *For a set of rules $\Sigma$,* $\mathsf{rewrite_a}(\Sigma)$ *is finite.*

A rule set $\Sigma$ is *separable* if $\mathsf{rewrite_a}(\Sigma) \equiv_{[\Sigma]} \Sigma$. Intuitively, the condition requires the rule bodies (in particular, the bodies of auxiliary rules) can be separated during rewriting. The class of separable rule sets is denoted sep. Clearly, a separable rule set always admits a (finite) datalog rewriting, yet the definition does not suggest how to effectively identify such a rule set. Thus, we first show that the existing shy class [Leone *et al.*, 2019] is a subclass of sep and then extend it to cover more practical rule sets.

A *position* is of the form $\mathsf{A}[i]$ with A being an $n$-ary predicate and $1 \leq i \leq n$, and a variable $v$ occurs at position $\mathsf{A}[i]$ if there is an atom $\mathsf{A}(t_1, \ldots, t_n)$ with $t_i = v$. For a set of rules $\Sigma$ and an existential variable $z$ in $\Sigma$, a position $\mathsf{A}[i]$ is *invaded* by $z$ if there is a rule $r \in \Sigma$ such that $\mathsf{head}(r) = \mathsf{A}(t_1, \ldots, t_n)$ and either $t_i = z$ or $t_i$ is a frontier variable that occurs in $\mathsf{body}(r)$ only at positions that are invaded by $z$. Recall that we assume each rule has a distinct set of variables. Then, a variable $x$ in $\Sigma$ is *attacked* by $z$ if $x$ only occurs in positions invaded by $z$. Two atoms in the same rule body are *chained* if (1) they share a variable that is attacked, or (2) they each contains a frontier variable and these two variables are both attacked by the same variable. Finally, $\Sigma$ is *shy* if it does not contain two chained atoms, and we denote the class of shy rule sets as shy. It can be seen that $\Sigma_{ex2}$ is shy, and every shy rule set is also separable.

**Theorem 1.** shy $\subset$ sep.

An example of separable but not shy rule set is $\Sigma_{ex3} = \{r_1 = \mathsf{A}(x, y) \leftarrow \mathsf{B}(x), r_2 = \mathsf{C}(x, y, z) \leftarrow \mathsf{A}(x, y) \wedge \mathsf{A}(x, z), r_3 = \mathsf{D}(x) \leftarrow \mathsf{C}(x, y, z)\}$. It is not shy because frontier variables $y$ and $z$ in $r_2$ are both attacked by the existential variable $y$ in $r_1$, which makes the two body atoms in $r_2$ chained. Yet it is separable, as any datalog rule in $\mathsf{rewrite}(\Sigma)$ over $[\Sigma]$, e.g., $\mathsf{D}(x) \leftarrow \mathsf{B}(x)$, is derivable from $\mathsf{rewrite_a}(\Sigma)$.

The class of separable rule sets can be expanded to cover more datalog rewritable cases. Note that OMQ $\Sigma_{ex1} \cup \{q_{ex}\}$ from Example 1 is not separable, yet a datalog rewriting does exist. We call a rule set $\Sigma$ *weakly-separable* if it can be transformed into a finite set of rules $\Sigma'$ such that $\mathsf{rewrite_a}(\Sigma') \equiv_{[\Sigma]} \Sigma$, and the extended class is denoted as wsep. From the definition, a weakly-separable rule set always admits a (finite) datalog rewriting. Indeed, the wsep class consists of all datalog rewritable rule sets $\Sigma$, as one can take the FPDR of $\Sigma$ as $\Sigma'$, which is finite and separable (as it contains no existential variable). Next, we want to extend the shy class (i) to allow blocks of, instead of individual, body atoms to be separable, and (ii) to combine it with concrete subclasses of fus.

A *block* $B \subseteq \mathsf{body}(r)$ for some rule $r$ is a smallest non-empty set such that if $\alpha \in B$ then for each atom $\beta$ chained to $\alpha$, $\beta \in B$. We say a block $B$ *depends* on a rule $r$ if $\mathsf{head}(r)$ share the same predicate with an atom in $B$. For a set of rules $\Sigma$, the *dependent rule set* of a block $B$, $\mathsf{dep}(B)$, is the smallest set of rules $r \in \Sigma$ such that $B$ depends on $r$ or some rule in $\mathsf{dep}(B)$ contains a block that depends on $r$.

**Definition 3.** *Let* fus-shy *consist of rule sets $\Sigma$ satisfying for each block $B$ in $\Sigma$ with $|B| \geq 2$,* $\mathsf{dep}(B) \in$ lin $\cup$ stky $\cup$ agrd.

The OMQ $\Sigma_{ex1} \cup \Sigma_{ex2} \cup \{q_{ex}\}$ belongs to fus-shy, as the only block with size greater than 1 is $B = \{\mathsf{A}(u, v), \mathsf{A}(v, w)\}$ in $q_{ex}$ and $\mathsf{dep}(B) = \Sigma_{ex1}$ which is fus; yet it is neither shy nor fus. We can show that each rule set that belongs to fus-shy is weakly separable, and thus is datalog rewritable. Intuitively, we can transform the rule set by fully unfolding the blocks $B$ of sizes greater than 1 by $\mathsf{dep}(B)$, resulting in an equivalent and separable rule set.

**Theorem 2.** lin $\cup$ stky $\cup$ agrd $\cup$ shy $\subset$ fus-shy $\subset$ wsep.

Moreover, a rule set in fus-shy coupled with any BCQ is also datalog rewritable.

**Proposition 4.** *For a rule set $\Sigma$ that belongs to* fus-shy *and any BCQ $q$, the OMQ $\Sigma \cup \{q\}$ belongs to* wsep.

## 5 An Efficient Rewriting Algorithm

In this section, we introduce an efficient method for computing datalog rewritings of the form $\mathsf{rewrite_a}(\Sigma)$ whenever they exist, and discuss how it can be adapted to compute $\mathsf{rewrite}(\Sigma)$. Inspired by [Hansen *et al.*, 2015], we compute a decomposed representation of the heads and bodies of the resulting rules generated during the rewriting, such that the representation is compact due to structure sharing and the datalog rewriting can be conveniently extracted from such a representation. For the ease of presentation, we first present a variant of the representation and then further simplify it.

For a set of rules $\Sigma$, its *rewriting forest* $F_\Sigma$ has nodes of the form $(\mathsf{P}(\vec{x}), H, B_1, B_2)$, where P is a fresh predicate not occurring in $\Sigma$, $\vec{x}$ is a vector of variables, $H$ is an atom, and $B_1, B_2$ are sets of atoms; and edges are labelled with piece unifiers. The roots of $F_\Sigma$ correspond to the rules in $\Sigma$, i.e., are of the form $(\top, \mathsf{head}(r), \emptyset, \mathsf{body}(r))$ for all rules $r \in \Sigma$, where $\top$ represents true. Intuitively, each node $((\mathsf{P}(\vec{x}), H, B_1, B_2)$ represents two rules as follows:

$$\mathsf{P}(\vec{x}) \leftarrow \bigwedge B_1, \qquad (1^*)$$

$$\exists \vec{z}.H \leftarrow \bigwedge B_2, \qquad (2^*)$$

where $\vec{z}$ consists of all the variables in the head but not in the body. Rules (1*) and (2*) correspond to the rules (1) and (2) generated during rewriting. Hence, P can be reused in the same way as separating predicates through the labelling function $\lambda(\cdot)$. Moreover, a node $n$ is *blocked* by another node $n'$ if its corresponds rules are both implied by those of $n'$.

Formally, $F_\Sigma$ has the smallest number of nodes and edges satisfying the following conditions: For each node $n = (\mathsf{P}(\vec{x}), H, B_1, B_2)$ and each root node $n' = (\top, H', \emptyset, B'_2)$, let $\vec{x}' = \mathsf{var}(H') \cap \mathsf{var}(B'_2)$; and

1. for each piece unifier $\mu = (B, \tau)$ of $B_2$ and $H'$, $n$ has a child $n'' = (\mathsf{P}''(\vec{x}''), H'', B''_1, B''_2)$ whenever $n''$ is not blocked s.t.
   - $\vec{x}'' = \mathsf{var}(H)\tau \cup \mathsf{var}(B_2 \setminus B)\tau \cap \vec{x}'\tau$, $\lambda(\mathsf{P}'') = H'\tau$,
   - $H'' = H\tau'$, where $\tau'$ is as in (*), $B''_1 = B'_2\tau'$ and $B''_2 = (B_2 \setminus B)\tau \cup \{\mathsf{P}''(\vec{x}'')\}$;
2. if $n$ is not a root, for each piece unifier $\mu$ of $B_1$ and $H'$, $n$ has a child $n''$ whenever $n''$ is not blocked s.t.
   - $\vec{x}'' = \vec{x}\tau \cup \mathsf{var}(B_1 \setminus B)\tau \cap \vec{x}'\tau$,

- $H'' = \mathsf{P}(\vec{x})\tau'$ and $B_2'' = (B_1 \setminus B)\tau \cup \{\mathsf{P}''(\vec{x}'')\}$,
- $B_1''$ and $\lambda(\mathsf{P}'')$ are as in Condition 1;

Our algorithm starts with the nodes corresponding to the rules in $\Sigma$ and expands the rewriting forest based on the above conditions. The expansion terminates due to the blocking condition, and the number of nodes are bounded based on the same argument as for Lemma 1. Let $\mathsf{datalog}(F_\Sigma)$ be the set of datalog rules obtained as above from the nodes of $F_\Sigma$.

**Theorem 3.** *For a set of rules $\Sigma$, $F_\Sigma$ is always finite and* $\mathsf{datalog}(F_\Sigma) \equiv_{[\Sigma]} \mathsf{rewrite_a}(\Sigma)$.

To further simplify the representation, note that $H$ and $B_2$ in each non-root node can be computed on the fly. In particular, for the computation of $H''$ and $B_2''$ under Conditions 2, it only needs the current node $n''$ and its parent node $n$; whereas under Condition 1, $B_2''$ refers to $B_2$ in the parent node $n$, which can be computed through back-tracking till a root.

To adapt rewriting forests for the computation of $\mathsf{rewrite}(\Sigma)$, we can expand each node with a fourth component $B_3$, which is a set of atoms. Intuitively, it is used to capture rules $\exists\vec{z}.H \leftarrow \bigwedge B_3$, that corresponds to the rules of the form (*) generated during the rewriting. In particular, $B_3'' = (B_2 \setminus B)\tau \cup B_2'\tau'$ in Conditions 1, and $B_3'' = (B_1 \setminus B)\tau \cup B_2'\tau'$ in Condition 2. Furthermore, there is a third condition: if $n$ is not a root, for each piece unifier $\mu$ of $B_3$ and $H'$, $n$ has a child $n''$ whenever $n''$ is not blocked such that $\vec{x}'' = \mathsf{var}(H)\tau \cup \mathsf{var}(B_3 \setminus B)\tau \cap \vec{x}'\tau$, $B_3'' = (B_3 \setminus B)\tau \cup B_2'\tau'$, and everything else is defined as in Condition 1. Again, $B_3$ in each node can be computed on the fly via back-tracking. The difference from $B_1$ and $B_2$ is that the sizes of $B_3$ are not necessarily bounded, unless $\mathsf{rewrite}(\Sigma)$ is finite.

## 6 Experiment

We have implemented a prototype system, Drewer (Datalog REWriting for Existential Rules), with our piece unification module adapted from the first-order rewriting system Graal[2] [König *et al.*, 2015a], and we deployed VLog[3] as our datalog engine. All experiments were performed on a laptop with a processor at 2.2 GHz and 8GB of RAM. The system and experiment benchmarks can be found at https://www.ict.griffith.edu.au/aist/Drewer.

We evaluated ontologies including the DL-Lite versions of LUBM, OpenGALEN2, OBOprotein and RS. RS is from [Bienvenu *et al.*, 2017] with a simple ontology but specially crafted long queries (with up to 15 atoms), which is a known challenge to existing rewriting-based systems. Reactome and Uniprot are in OWL2, and we used the existential rule fragments of them which are more expressive than DL-Lite. The ontologies were converted into existential rules using a transformation tool provided by Graal. DEEP200/300, STB-128, and ONT-256 are from ChaseBench [Benedikt *et al.*, 2017], a benchmark for chase-based reasoning systems. They contain existential rules with predicates of arities more than two. All the tested ontologies are found to be in $\mathsf{fus}$-$\mathsf{shy}$ and thus can be handled by Drewer, whereas none of the compared rewriting system could successfully handle all of them.

[2]http://graphik-team.github.io/graal/

[3]https://github.com/knowsys/vlog4j

We conducted two sets of experiments to evaluate the performance of our system. 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, Rapid [Trivela *et al.*, 2015] and Iqaros [Venetis *et al.*, 2016] are datalog and UCQ rewriting systems for description logic ontologies. We used original queries that come with the datasets and evaluated 5 queries per ontology, except for RS which has only 3 long 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. We set a 5 minutes time limit per query, and TO denotes a timeout whereas a "-" means the system could not handle the OMQ (or reported errors).

It can be seen from Table 1, the sizes of datalog rewritings are often comparable to or much smaller than those of UCQ rewritings. Note that to achieve efficiency as well as compactness in rewriting, Graal makes use of the so-called compiled pre-orders of the form $\mathsf{A} \preceq \mathsf{B}$ for predicates $\mathsf{A}, \mathsf{B}$, which can be seen as datalog rules $\mathsf{B}(\vec{x}) \leftarrow \mathsf{A}(\vec{x})$. Hence, the UCQ rewritings produced by Graal need to be coupled with the compiled pre-orders for query answering. For a fair comparison, we report the rewriting sizes for Graal in the form of $x + y$, where $x$ is the size of the UCQ rewriting and $y$ is that of the datalog rules corresponding to the compiled pre-orders. Since not all pre-orders are used for specific queries, we tracked for each query those pre-orders actually used for query answering (following Graal's native query answering process) and $y$ is the number of only those used pre-orders.

Regarding the time efficiency of rewriting, Drewer is again superior or comparable to other systems in almost all cases. In particular, all other systems reached a timeout on q5 for OBOprotein, whereas Drewer took only less than a second to complete the rewriting. And all other systems except for Graal failed to complete their rewriting on the 3 queries for RS, due to the large sizes of UCQ rewritings.

To evaluate the overall performance of Drewer in query answering, we conducted a second set of experiments comparing it with other in-memory (as Drewer is in-memory) query answering systems. To separate the contribution of our rewriting method from that of an efficient datalog engine, besides Graal and Iqaros, we also compare Drewer with state-of-the-art chased-based systems VLog [Carral *et al.*, 2019] and DLV$^\exists$ [Leone *et al.*, 2019]. All ontologies except for Uniprot are weak acyclic [Grau *et al.*, 2013], so the chase-based systems can be used for query answering. To compare with Graal on the quality of different rewritings, we used a datalog translation for both of its UCQ rewritings and the used pre-orders, and deployed the same datalog engine VLog.

Table 2 presents the average times (in milliseconds) for answering the 5 queries over each ontology, where we ran each query separately (which may involve running the chase procedure separately for each query). If a system reached a timeout on a particular query, we counted 5 minutes. Hence, a TO means a failure for all the 5 queries. We separate the query answering times (QA, including possibly rewriting, chase computation, and query evaluation) out of the total times, except for DLV$^\exists$ where it was difficult to make such a separation.

| Ontology | Query | Datalog Rewriting | | | | UCQ Rewriting | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Drewer | | Rapid | | Graal | | Iqaros | |
| | | Size | Time | Size | Time | Size* | Time | Size | Time |
| OpenGALEN2 | q1 | 3 | 1 | 3 | 5 | 1+2 | 117 | 2 | 4 |
| | q2 | 1296 | 34 | 1276 | 82 | 1+1275 | 175 | 1152 | 56 |
| | q3 | 93 | 1055 | 92 | 39 | 5+87 | 208 | 488 | 44 |
| | q4 | 162 | 7 | 155 | 15 | 1+154 | 108 | 147 | 11 |
| | q5 | 75 | 2194 | 81 | 25 | 19+62 | 211 | 324 | 40 |
| OBOprotein | q1 | 30 | 63 | 29 | 19 | 20+7 | 259 | 27 | 16 |
| | q2 | 1357 | 521 | 1356 | 720 | 1264+92 | 6866 | 1356 | 963 |
| | q3 | 34580 | 152 | 33919 | 128341 | 1+33918 | 338 | 33887 | 699 |
| | q4 | 34625 | 2496 | 34879 | 127786 | 682+34085 | 3239 | 34733 | 12832 |
| | q5 | 1386 | 516 | 27907 | TO | TO | TO | 36612 | TO |
| RS | q1 | 14 | 153 | TO | TO | 14+4 | 904 | TO | TO |
| | q2 | 22 | 849 | TO | TO | 100+4 | 13554 | TO | TO |
| | q3 | 25 | 8976 | TO | TO | 143+4 | 40875 | TO | TO |
| DEEP300 | q1 | 25 | 286 | - | - | 117 + 2 | 927 | - | - |
| | q2 | 30 | 280 | - | - | 528 + 3 | 12417 | - | - |
| | q3 | 30 | 282 | - | - | 624 + 2 | 17196 | - | - |
| | q4 | 234 | 352 | - | - | TO | TO | - | - |
| | q5 | 55 | 294 | - | - | TO | TO | - | - |
| ONT-256 | q1 | 11 | 35 | - | - | 8 + 2 | 94 | - | - |
| | q2 | 16 | 40 | - | - | 4 + 6 | 127 | - | - |
| | q3 | 14 | 33 | - | - | 8 + 6 | 139 | - | - |
| | q4 | 16 | 32 | - | - | 12 +5 | 176 | - | - |
| | q5 | 16 | 35 | - | - | 48 + 1 | 2378 | - | - |

Table 1: Comparison on query rewriting

| Ontology | Drewer | | VLog | | $\text{DLV}^{\exists}$ | Graal (VLog) | | Iqaros | |
|---|---|---|---|---|---|---|---|---|---|
| | Total | QA | Total | QA | Total | Total | QA | Total | QA |
| LUBM-100 | 21339 | 1114 | 21696 | 1952 | 162975 | 21224 | 301 | 22651 | 2604 |
| Reactome | 2876 | 61 | 3300 | 375 | 20092 | 240589 | 240004 | 6629 | 1455 |
| Uniprot | 1791 | 46 | TO | TO | 10448 | 1796 | 88 | 2991 | 755 |
| DEEP200 | 701 | 189 | 3700 | 2456 | 452 | 122295 | 121913 | - | - |
| DEEP300 | 840 | 303 | TO | TO | 562 | 125447 | 125042 | - | - |
| STB-128 | 8287 | 116 | 8822 | 603 | 22239 | 8207 | 148 | - | - |
| ONT-256 | 25892 | 56 | 28275 | 2070 | 206021 | 26305 | 589 | - | - |

Table 2: Comparison on query answering

Overall, our (datalog) rewritings demonstrate significantly better efficiency in query answering compared to UCQ rewritings (by Graal and Iqaros); while Drewer could successfully handle all queries, Graal reached a timeout on 4 queries for Reactome and 2 queries for DEEP200 and DEEP300. In the comparison with chase-based systems, our system demonstrates superior time efficiency (sometimes by magnitudes) than VLog on all tested cases, which shows our performance gain is significantly contributed by the rewriting. Drewer also outperforms or is comparable to $\text{DLV}^{\exists}$, which compute the parsimonious chase rather than the full chase, on most of the cases. A chase-based system still has the advantage of being able to answering multiple queries over a single computation of the chase.Yet when the chase is difficult or impossible to compute, e.g., on Uniprot, our rewriting can be a determining factor between success and failure.

## 7 Conclusion

In this paper, we have presented a novel approach to datalog rewriting for existential rules and introduced a new concrete datalog rewritable class by combining several existing classes. We also implemented and evaluated our prototype system, Drewer, which is capable to handle a wide range of practical ontologies with superior or comparable performance compared to state-of-the-art rewriting and query answering systems. For future work, we are working on identifying more general classes of datalog rewritable existential rules, and optimising our rewriting algorithm and implementation.

## Acknowledgements

This work was partially supported by NSFC under grant 61976153.

## References

[Ahmetaj *et al.*, 2018] S. Ahmetaj, M. Ortiz, and M. Simkus. Rewriting guarded existential rules into small datalog programs. In *Proc. of ICDT*, pages 4:1–4:24, 2018.

[Baget *et al.*, 2011] 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.

[Benedikt *et al.*, 2017] M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, P. Papotti, D. Santoro, and E. Tsamoura. Benchmarking the chase. In *Proc. of PODS*, pages 37–52, 2017.

[Benedikt *et al.*, 2018] M. Benedikt, B. Motik, and E. Tsamoura. Goal-driven query answering for existential rules with equality. In *Proc. of AAAI*, pages 1761–1770, 2018.

[Bienvenu *et al.*, 2014] M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.

[Bienvenu *et al.*, 2017] M. Bienvenu, S. Kikot, R. Kontchakov, V. V. Podolskii, V. Ryzhikov, and M. Zakharyaschev. The complexity of ontology-based data access with OWL 2 QL and bounded treewidth queries. In *Proc. of PODS*, pages 201–216, 2017.

[Calì *et al.*, 2012a] 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.

[Calì *et al.*, 2012b] A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.

[Calì *et al.*, 2013] 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.

[Carral *et al.*, 2019] D. Carral, I. Dragoste, L. González, C. J. H. Jacobs, M. Krötzsch, and J. Urbani. Vlog: A rule engine for knowledge graphs. In *Proc. of ISWC*, pages 19–35, 2019.

[Eiter *et al.*, 2012] T. Eiter, M. Ortiz, M. Simkus, T.-K. Tran, and G. Xiao. Query rewriting for Horn-SHIQ plus rules. In *Proc. of AAAI*, 2012.

[Gottlob and Schwentick, 2012] G. Gottlob and T. Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In *Proc. of KR*, 2012.

[Gottlob *et al.*, 2014a] 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.

[Gottlob *et al.*, 2014b] G. Gottlob, S. Rudolph, and M. Simkus. Expressiveness of guarded existential rule languages. In *Proc. of PODS*, pages 27–38, 2014.

[Grau *et al.*, 2013] 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.

[Hansen *et al.*, 2015] P. Hansen, C. Lutz, I. Seylan, and F. Wolter. Efficient query rewriting in the description logic EL and beyond. In *Proc. of IJCAI*, pages 3034–3040, 2015.

[König *et al.*, 2015a] M. König, M. Leclère, and M.-L. Mugnier. Query rewriting for existential rules with compiled preorder. In *Proc. of IJCAI*, pages 3106–3112, 2015.

[König *et al.*, 2015b] 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.

[Kontchakov *et al.*, 2010] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in DL-Lite. In *Proc. of KR*, 2010.

[Leclère *et al.*, 2016] M. Leclère, M.-L. Mugnier, and F. Ulliana. On bounded positive existential rules. In *Proc. of DL*, 2016.

[Leone *et al.*, 2019] N. Leone, M. Manna, G. Terracina, and P. Veltri. Fast query answering over existential rules. *ACM Trans. Comput. Log.*, 20(2):12:1–12:48, 2019.

[Pérez-Urbina *et al.*, 2010] 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.

[Rosati and Almatelli, 2010] R. Rosati and A. Almatelli. Improving query answering over DL-Lite ontologies. In *Proc. of KR*, 2010.

[Stefanoni *et al.*, 2013] G. Stefanoni, B. Motik, and I. Horrocks. Introducing nominals to the combined query answering approaches for EL. In *Proc. of AAAI*, 2013.

[Trivela *et al.*, 2015] 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.

[Venetis *et al.*, 2016] T. Venetis, G. Stoilos, and V. Vassalos. Rewriting minimisations for efficient ontology-based query answering. In *Proc. of ICTAI*, pages 1095–1102, 2016.

[Wang *et al.*, 2018] Z. Wang, K. Wang, and X. Zhang. Forgetting and unfolding for existential rules. In *Proc. of AAAI*, pages 2013–2020, 2018.

[Zhou *et al.*, 2015] 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.