





Efficient Dependency Analysis for Rule-Based Ontologies

Larry González , Alex Ivliev^(✉) , Markus Krötzsch ,
and Stephan Mennicke 

Knowledge-Based Systems Group, TU Dresden, Dresden, Germany
{larry.gonzalez,alex.ivliev,markus.kroetzsch,
stephan.mennicke}@tu-dresden.de

Abstract. Several types of *dependencies* have been proposed for the static analysis of existential rule ontologies, promising insights about computational properties and possible practical uses of a given set of rules, e.g., in ontology-based query answering. Unfortunately, these dependencies are rarely implemented, so their potential is hardly realised in practice. We focus on two kinds of rule dependencies – *positive reliances* and *restraints* – and design and implement optimised algorithms for their efficient computation. Experiments on real-world ontologies of up to more than 100,000 rules show the scalability of our approach, which lets us realise several previously proposed applications as practical case studies. In particular, we can analyse to what extent rule-based bottom-up approaches of reasoning can be guaranteed to yield redundancy-free “lean” knowledge graphs (so-called *cores*) on practical ontologies.

Keywords: Existential rules · Chase algorithm · Rule dependencies · Acyclicity · Core stratification · Ontology-based query answering · Ontology reasoning

1 Introduction

Existential rules are a versatile knowledge representation language with relevance in ontological reasoning [1, 5, 6, 10], databases [11, 13, 15], and declarative computing in general [3, 4, 9]. In various semantic web applications, existential rule engines have been used to process knowledge graphs and ontologies, often realising performance advantages on large data sets [2, 3, 7, 22].

Existential rules extend Datalog with the facility for *value invention*, expressed by existentially quantified variables in conclusions. This ability to refer to “unknown” values is an important similarity to description logics (DLs) and the DL-based ontology standard OWL, and many such ontologies can equivalently be expressed in existential rules. This can be a practical approach for ontology-based query answering [8, 10]. For reasoning, many rule engines rely on *materialisation*, where the input data is expanded iteratively until all rules are satisfied (this type of computation is called *chase*). With existentials, this can require adding new “anonymous” individuals – called *nulls* –, and the process

may not terminate. Several *acyclicity conditions* define cases where termination is ensured, and were shown to apply to many practical ontologies [10].

Nulls correspond to *blank nodes* in RDF, and – like *bnodes* in RDF [20] – are not always desirable. Avoiding nulls entirely is not an option in chase-based reasoning, but one can still avoid some “semantically redundant” nulls. For example, given a fact `person(alice)` and a rule `person(x) → ∃y. parent(x, y)`, the chase would derive `parent(alice, n)` for a fresh null n . However, if we already know that `parent(alice, bob)`, then this inference is redundant and can be omitted. In general, structures that are free of such redundancies are mathematically known as *cores*. An RDF-graph that is a core is called a *lean* graph [16]. Unfortunately, the computation of cores is expensive, and can in general not be afforded during the chase. Sometimes, however, when rules satisfy a condition known as *core stratification*, practical chase algorithms can also produce a core directly [17].

Interestingly, both of the previously mentioned types of conditions – acyclicity and core stratification – are detected by analysing *dependencies*¹ that indicate possible semantic interactions between rules. Early works focussed on cases where a rule ρ_2 *positively relies* on a rule ρ_1 in the sense that an application of rule ρ_1 might trigger an application of rule ρ_2 . They are used to detect several forms of acyclicity [1, 11, 21]. When adding negation, a rule might also inhibit another, and such *negative reliances* are used to define semantically well-behaved fragments of nonmonotonic existential rules [17, 19]. A third kind of dependency are *restraints*, which indicate that the application of one rule might render another one redundant: restraints were used to define *core stratified rule sets* [17], and recently also to define a semantics for queries with negation [12].

Surprisingly, given this breadth of applications, rule dependencies are hardly supported in practice. To our knowledge, positive reliances are only computed by the Graal toolkit [2], whereas negative reliances and restraints have no implementation at all. A possible reason is that such dependency checks are highly intractable, typically Σ_2^P -complete, and therefore not easy to implement efficiently. This is critical since their proposed uses are often related to the choice of a rule-processing strategy, so that their computation adds to overall reasoning time. Moreover, as opposed to many other static analyses, dependency computation is not mainly an application of algorithms that are already used in rule reasoning. Today’s use of dependencies in optimisation and analysis therefore falls short of expectations.

To address this problem, we design optimised algorithms for the computation of positive reliances and restraints. We propose *global* optimisations, reducing the number of relevant checks, and *local* optimisations, reducing the work needed to execute a specific check. The latter include an improved search strategy that often avoids the full exploration of exponentially many subsets of rule atoms, which may be necessary in the worst case. The underlying ideas can also be adapted to negative reliances and any of the modified definitions of positive reliances found in the literature.

¹ We use the term only informally, since (*tuple-generating*) *dependencies* are also a common name for rules in databases.

We implement our methods and conduct extensive experiments with over 200 real-world ontologies of varying sizes. Considering the effectiveness of our optimisations, we find that local and global techniques both make important contributions to overall performance, enabling various practical uses:

- We conduct the first analysis of the practical prevalence of *core stratification* [17] using our implementation of restraints. We find this desirable property in a significant share of ontologies from a curated repository and provide preliminary insights on why some rule sets are not core stratified.
- Comparing the computation of all positive reliances to Graal, we see speed-ups of more than two orders of magnitude. Our stronger definition yields an *acyclic graph of rule dependencies* [1] in more cases.
- The graph of positive reliances allows for showing how to speed up the expensive rule analysis algorithm *MFA* [10]. Compared to the MFA implementation of VLog [7], we observe speed-ups of up to four orders of magnitude.

2 Preliminaries

We build expressions from countably infinite, mutually disjoint sets \mathbf{V} of *variables*, \mathbf{C} of *constants*, \mathbf{N} of *labelled nulls*, and \mathbf{P} of *predicate names*. Each predicate name $p \in \mathbf{P}$ has an *arity* $\text{ar}(p) \geq 0$. *Terms* are elements of $\mathbf{V} \cup \mathbf{N} \cup \mathbf{C}$. We use \mathbf{t} to denote a list $t_1, \dots, t_{|\mathbf{t}|}$ of terms, and similar for special types of terms. An *atom* is an expression $p(\mathbf{t})$ with $p \in \mathbf{P}$, \mathbf{t} a list of terms, and $\text{ar}(p) = |\mathbf{t}|$. *Ground* terms or atoms contain neither variables nor nulls. An *interpretation* \mathcal{I} is a set of atoms without variables. A *database* \mathcal{D} is a finite set of ground atoms.

Syntax. An *existential rule* (or just *rule*) ρ is a formula

$$\rho = \forall \mathbf{x}, \mathbf{y}. \varphi[\mathbf{x}, \mathbf{y}] \rightarrow \exists \mathbf{z}. \psi[\mathbf{y}, \mathbf{z}], \quad (1)$$

where φ and ψ are conjunctions of atoms using only terms from \mathbf{C} or from the mutually disjoint lists of variables $\mathbf{x}, \mathbf{y}, \mathbf{z} \subseteq \mathbf{V}$. We call φ the *body* (denoted $\text{body}(\rho)$) and ψ the *head* (denoted $\text{head}(\rho)$). We may treat conjunctions of atoms as sets, and we omit universal quantifiers in rules. We require that all variables in \mathbf{y} do really occur in φ (*safety*). A rule is *Datalog* if it has no existential quantifiers.

Semantics. Given a set of atoms \mathcal{A} and an interpretation \mathcal{I} , a *homomorphism* $h: \mathcal{A} \rightarrow \mathcal{I}$ is a function that maps the terms occurring in \mathcal{A} to the (variable-free) terms occurring in \mathcal{I} , such that: (i) for all $c \in \mathbf{C}$, $h(c) = c$; (ii) for all $p \in \mathbf{P}$, $p(\mathbf{t}) \in \mathcal{A}$ implies $p(h(\mathbf{t})) \in \mathcal{I}$, where $h(\mathbf{t})$ is the list of h -images of the terms \mathbf{t} . If (ii) can be strengthened to an “if, and only if”, then h is *strong*. We apply homomorphisms to a formula by applying them individually to all of its terms.

A *match* of a rule ρ in an interpretation \mathcal{I} is a homomorphism $\text{body}(\rho) \rightarrow \mathcal{I}$. A match h of ρ in \mathcal{I} is *satisfied* if there is a homomorphism $h': \text{head}(\rho) \rightarrow \mathcal{I}$ that

agrees with h on all variables that occur in body and head (i.e., variables \mathbf{y} in (1)). Rule ρ is *satisfied* by \mathcal{I} , written $\mathcal{I} \models \rho$, if every match of ρ in \mathcal{I} is satisfied. A set of rules Σ is satisfied by \mathcal{I} , written $\mathcal{I} \models \Sigma$, if $\mathcal{I} \models \rho$ for all $\rho \in \Sigma$. We write $\mathcal{I} \models \mathcal{D}, \Sigma$ to express that $\mathcal{I} \models \Sigma$ and $\mathcal{D} \subseteq \mathcal{I}$. In this case, \mathcal{I} is a *model* of Σ and \mathcal{D} .

Applying Rules. A rule ρ of form (1) is *applicable* to an interpretation \mathcal{I} if there is an unsatisfied match h in \mathcal{I} (i.e., h cannot be extended to a homomorphism $\psi \rightarrow \mathcal{I}$). Applying ρ for h yields the interpretation $\mathcal{I} \cup \psi[h'(\mathbf{y}), h'(\mathbf{z})]$, where h' is a mapping such that $h'(y) = h(y)$ for all $y \in \mathbf{y}$, and for all $z \in \mathbf{z}$, $h'(z) \in \mathbf{N}$ is a distinct null not occurring in \mathcal{I} . The (*standard*) *chase* is a reasoning algorithm obtained by applying rules to a given initial database, such that all applicable rules are eventually applied (fairness).

Core Models. A model \mathcal{I} is a *core* if every homomorphism $h: \mathcal{I} \rightarrow \mathcal{I}$ is strong and injective. For finite models, this is equivalent to the requirement that every such homomorphism is an isomorphism, and this will be the only case we are interested in for this work. Intuitively, the condition states that the model does not contain a strictly smaller substructure that is semantically equivalent for conjunctive query answering.

Unification. For atom sets \mathcal{A} and \mathcal{B} , partial function $m: \mathcal{A} \rightarrow \mathcal{B}$ is an *atom mapping*, where $\text{dom}(m) \subseteq \mathcal{A}$ is the set of all atoms for which m is defined. A *substitution* is a function $\theta: \mathbf{C} \cup \mathbf{V} \cup \mathbf{N} \rightarrow \mathbf{C} \cup \mathbf{V} \cup \mathbf{N}$, such that $\theta(c) = c$ for all $c \in \mathbf{C} \cup \mathbf{N}$. Denote the application of θ to term t by $t\theta$, naturally extending to atoms and atom sets by term-wise application. The concatenation of substitutions σ and θ is $\sigma\theta$ where $t\sigma\theta = (t\sigma)\theta$. A substitution is a *unifier* for atom mapping m if for all $\alpha \in \text{dom}(m)$, $\alpha\theta = (m(\alpha))\theta$. A unifier μ for m is a *most general unifier* (mgu) for m if for all unifiers ν of m , there is a substitution σ , such that $\mu\sigma = \nu$.

3 Dependencies and Their Naive Computation

We first introduce the two kinds of rule dependencies that we consider: *positive reliances* and *restraints*. Our definitions largely agree with the literature, but there are some small differences that we comment on.

Definition 1. A rule ρ_2 positively relies on a rule ρ_1 , written $\rho_1 \prec^+ \rho_2$, if there are interpretations $\mathcal{I}_a \subseteq \mathcal{I}_b$ and a function h_2 such that

- (a) \mathcal{I}_b is obtained from \mathcal{I}_a by applying ρ_1 for the match h_1 extended to h'_1 ,
- (b) h_2 is an unsatisfied match for ρ_2 on \mathcal{I}_b , and
- (c) h_2 is not a match for ρ_2 on \mathcal{I}_a .

Definition 1 describes a situation where an application of ρ_1 immediately enables a new application of ρ_2 . Condition (b) takes into account that only unsatisfied matches can lead to rule applications in the standard chase. The same condition is used by Krötzsch [17], whereas Baget et al. [1, 2] – using what they call *piece-unifier* – only require h_2 to be a match. In general, weaker definitions are not incorrect, but may lead to unnecessary dependencies.

Example 1. Consider the following ontology. We provide three axioms in DL syntax (left-hand side) and their translation into existential rules (right-hand side).

$$\begin{array}{ll} A \sqsubseteq \exists R.B & a(x) \rightarrow \exists v. r(x, v) \wedge b(v) \quad (\rho_1) \\ R^- \circ R \sqsubseteq T & r(y, z_1) \wedge r(y, z_2) \rightarrow t(z_1, z_2) \quad (\rho_2) \\ \exists R^-.A \sqsubseteq B & a(t) \wedge r(t, u) \rightarrow b(u) \quad (\rho_3) \end{array}$$

For this rule set, we find $\rho_1 \prec^+ \rho_2$ by using $\mathcal{I}_a = \{a(c)\}$, $\mathcal{I}_b = \{a(c), r(c, n)\}$, and $h_2 = \{y \mapsto c, z_1 \mapsto n, z_2 \mapsto n\}$. Note that ρ_3 does not positively rely on ρ_1 although the application of ρ_1 may lead to a new match for ρ_3 . However, this match is always satisfied, so condition (b) of Definition 1 is not fulfilled.

The definition of restraints considers situations where the nulls introduced by applying rule ρ_2 are at least in part rendered obsolete by a later application of ρ_1 . This obsolescence is witnessed by an *alternative match* that specifies a different way of satisfying the rule match of ρ_2 .

Definition 2. Let $\mathcal{I}_a \subseteq \mathcal{I}_b$ be interpretations such that \mathcal{I}_a was obtained by applying the rule ρ for match h which is extended to h' . A homomorphism $h^A: h'(\text{head}(\rho)) \rightarrow \mathcal{I}_b$ is an *alternative match* of h' and ρ on \mathcal{I}_b if

- (1) $h^A(t) = t$ for all terms t in $h(\text{body}(\rho))$, and
- (2) there is a null n in $h'(\text{head}(\rho))$ that does not occur in $h^A(h'(\text{head}(\rho)))$.

Now ρ_1 restrains ρ_2 if it creates an alternative match for it:

Definition 3. A rule ρ_1 restrains a rule ρ_2 , written $\rho_1 \prec^\square \rho_2$, if there are interpretations $\mathcal{I}_a \subseteq \mathcal{I}_b$ such that

- (a) \mathcal{I}_b is obtained by applying ρ_1 for match h_1 extended to h'_1 ,
- (b) \mathcal{I}_a is obtained by applying ρ_2 for match h_2 extended to h'_2 ,
- (c) there is an alternative match h^A of h'_2 and ρ_2 on \mathcal{I}_b , and
- (d) h^A is no alternative match of h'_2 and ρ_2 on $\mathcal{I}_b \setminus h'_1(\text{head}(\rho_1))$.

Our definition slightly deviates from the literature [17], where (d) made a stronger requirement:

- (d') h_2 has no alternative match $h'_2(\text{head}(\rho_2)) \rightarrow \mathcal{I}_b \setminus h'_1(\text{head}(\rho_1))$.

As we will see, our modification allows for a much more efficient implementation, but it also leads to more restraints. Since restraints overestimate potential interactions during the chase anyway, all formal results of prior works are preserved.

Example 2. For the rules $\rho_1 = r(y, y) \rightarrow \exists w. r(y, w) \wedge b(w)$ and $\rho_2 = a(x) \rightarrow \exists v. r(x, v)$, we find $\rho_1 \prec^\square \rho_2$ by Definition 3, where we set $\mathcal{I}_a = \{a(c), r(c, n_1)\}$, $\mathcal{I}_b = \mathcal{I}_a \cup \{r(c, c), r(c, n_2), b(n_2)\}$, and $h^A = \{c \mapsto c, n_1 \mapsto n_2\}$. However, these \mathcal{I}_a and \mathcal{I}_b do not satisfy the stricter condition (d'), since $h^B = \{c \mapsto c, n_1 \mapsto c\}$ is an alternative match, too. Indeed, when ρ_2 is applicable in such a way as to produce an alternative match w.r.t. an application of ρ_1 , another one must have already existed.

Example 2 is representative of situations where (d) leads to different restraints than (d'): the body of the restraining rule ρ_1 must contain a pattern that enforces an additional alternative match (here: $r(y, y)$), while not being satisfiable by the conclusion of ρ_2 (here: $r(y, n_1)$). To satisfy the remaining conditions, $\text{head}(\rho_1)$ must further produce a (distinct) alternative match. Such situations are very rare in practice, so that the benefits of (d) outweigh the loss of generality.

Checking for positive reliances and restraints is Σ_2^P -complete. Indeed, we can assume \mathcal{I}_a and \mathcal{I}_b to contain at most as many elements as there are distinct terms in the rule, so that they can be polynomially guessed. The remaining conditions can be checked by an NP-oracle. Hardness follows from the Σ_2^P -hardness of deciding if a rule has an unsatisfied match [15].

The existence of alternative matches in a chase sequence indicates that the resulting model may contain redundant nulls. Ordering the application of rules during the chase in a way that obeys the restraint relationship (\prec^\square) ensures that the chase sequence does not contain any alternative matches and therefore results in a core model [17].

Example 3. Consider again the rule set from Example 1. For the interpretation $\mathcal{I}_0 = \{a(c), r(c, d)\}$ all three rules are applicable. Disregarding $\rho_3 \prec^\square \rho_1$ and applying ρ_1 first results in $\mathcal{I}_1 = \mathcal{I}_0 \cup \{r(c, n), b(n)\}$, which leads to the alternative match $h^A = \{c \mapsto c, n \mapsto d\}$ after applying ρ_3 . If we, on the other hand, start with ρ_3 , we obtain $\mathcal{I}'_1 = \mathcal{I}_0 \cup \{b(d)\}$. Rule ρ_1 is now satisfied and the computation finishes with a core model after applying ρ_2 .

The ontology from Example 1 is an example of a *core stratified* rule set. A set of rules is *core stratified* if the graph of all $\prec^+ \cup \prec^\square$ edges does not have a cycle that includes a \prec^\square edge. This property allows us to formulate a rule application strategy that respects the restraint relationship as follows: Given $\rho_1 \prec^\square \rho_2$, apply the restrained rule ρ_2 only if neither ρ_1 nor any of the rules ρ_1 directly or indirectly positively relies on is applicable.

4 Computing Positive Reliances

The observation that positive reliances can be decided in Σ_2^P is based on an algorithm that considers all possible sets \mathcal{I}_a and \mathcal{I}_b up to a certain size. This

is not practical, in particular for uses where dependencies need to be computed as part of the (performance-critical) reasoning, and we therefore develop a more goal-oriented approach.

In the following, we consider two rules ρ_1 and ρ_2 of form $\rho_i = \text{body}_i \rightarrow \exists z_i. \text{head}_i$, with variables renamed so that no variable occurs in both rules. Let \mathbf{V}_\forall and \mathbf{V}_\exists , respectively, denote the sets of universally and existentially quantified variables in ρ_1 and ρ_2 . A first insight is that the sets \mathcal{I}_a and \mathcal{I}_b of Definition 1 can be assumed to contain only atoms that correspond to atoms in ρ_1 and ρ_2 , with distinct universal or existential variables replaced by distinct constants or nulls, respectively. For this replacement, we fix a substitution ω that maps each variable in \mathbf{V}_\exists to a distinct null, and each variable in \mathbf{V}_\forall to a distinct constant that does not occur in ρ_1 or ρ_2 .

Algorithm 1: extend⁺

Input: rules ρ_1, ρ_2 , atom mapping m

Output: *true* iff the atom mapping can be extended successfully

```

1 for  $i \in \{\text{maxidx}(m) + 1, \dots, |\text{body}_2|\}$  do
2   for  $j \in \{1, \dots, |\text{head}_1|\}$  do
3      $m' \leftarrow m \cup \{\text{body}_2[i] \mapsto \text{head}_1[j]\omega_\exists\}$ 
4     if  $\eta \leftarrow \text{unify}(m')$  then
5       if  $\text{check}^+(\rho_1, \rho_2, m', \eta)$  then return true
6 return false
    
```

A second insight is that, by (c), ρ_1 must produce some atoms that are relevant for a match of ρ_2 , so that our algorithm can specifically search for a *mapped* subset $\text{body}_2^m \subseteq \text{body}_2$ and a substitution η such that $\text{body}_2^m \eta \subseteq \text{head}_1 \eta$. Note that η represents both matches h_1 and h_2 from Definition 1, which is possible since variables in ρ_1 and ρ_2 are disjoint. The corresponding set \mathcal{I}_a then is $(\text{body}_1 \cup (\text{body}_2 \setminus \text{body}_2^m))\eta\omega$. Unfortunately, it does not suffice to consider singleton sets for body_2^m , as shown by Example 4:

Example 4. Consider the rules from Example 1. Trying to map either one of the atoms of $\text{body}(\rho_2)$ to $\text{head}(\rho_1)$ yields an $\mathcal{I}_a = \{a(c), r(c, c')\}$, to which ρ_1 is not applicable. The correct $\mathcal{I}_a = \{a(c)\}$ as given in Example 1 is found by unifying both atoms of $\text{body}(\rho_2)$ with (an instance of) $\text{head}(\rho_1)$.

Therefore, we have to analyse all subsets $\text{body}_2^m \subseteq \text{body}_2$ for possible matches with head_1 . We start the search from singleton sets, which are successively extended by adding atoms. A final important insight is that this search can often be aborted early, since a candidate pair for \mathcal{I}_a and \mathcal{I}_b may fail Definition 1 for various reasons, and considering a larger body_2^m is not always promising. For example, if η is a satisfied match for ρ_2 over \mathcal{I}_b (b), then adding more atoms to body_2^m will never succeed.

These ideas are implemented in Algorithms 1 (**extend**⁺) and 2 (**check**⁺), explained next. For a substitution θ , we write θ_{\forall} (θ_{\exists} , resp.), to denote the substitution assigning existential variables (universal variables, resp.) to themselves, and otherwise agrees with θ .

Function **extend**⁺ iterates over extensions of a given candidate set. To specify how atoms of **body**₂ are mapped to **head**₁, we maintain an atom mapping $m: \text{body}_2 \rightarrow \text{head}_1$ whose domain $\text{dom}(m)$ corresponds to the chosen $\text{body}_2^m \subseteq \text{body}_2$. To check for the positive reliance, we initially call **extend**⁺($\rho_1, \rho_2, \emptyset$). Note that ρ_1 and ρ_2 can be based on the same rule (a rule can positively rely on itself); we still use two variants that ensure disjoint variable names.

Algorithm 2: check⁺

Input: rules ρ_1, ρ_2 , atom mapping m with mgu η

Output: *true* if a positive reliance is found for m

```

7 body2m ←  $\text{dom}(m)$ 
8 body2ℓ ←  $\{\text{body}_2[j] \in (\text{body}_2 \setminus \text{body}_2^m) \mid j < \text{maxidx}(m)\}$ 
9 body2r ←  $\{\text{body}_2[j] \in (\text{body}_2 \setminus \text{body}_2^m) \mid j > \text{maxidx}(m)\}$ 
10 if body1 $\eta$  contains a null then return false
11 if body2ℓ $\eta$  contains a null then return false
12 if body2r $\eta$  contains a null then return extend+( $\rho_1, \rho_2, m$ )
13  $\mathcal{I}_a \leftarrow (\text{body}_1 \cup \text{body}_2^\ell \cup \text{body}_2^r)\eta\omega$ 
14 if  $\mathcal{I}_a \models \exists z_1. \text{head}_1\eta\omega_{\forall}$  then return extend+( $\rho_1, \rho_2, m$ )
15 if  $\text{body}_2\eta\omega \subseteq \mathcal{I}_a$  then return extend+( $\rho_1, \rho_2, m$ )
16  $\mathcal{I}_b \leftarrow \mathcal{I}_a \cup \text{head}_1\eta\omega$ 
17 if  $\mathcal{I}_b \models \exists z_2. \text{head}_2\eta\omega_{\forall}$  then return false
18 return true

```

We treat rule bodies and heads as lists of atoms, and write $\varphi[i]$ for the i th atom in φ . The expression $\text{maxidx}(m)$ returns the largest index of an atom in $\text{dom}(m)$, or 0 if $\text{dom}(m) = \emptyset$. By extending m only with atoms of larger index (L1), we ensure that each $\text{dom}(m)$ is only considered once. We then construct each possible extension of m (L3), where we replace existential variables by fresh nulls in **head**₁. In Line 4, **unify**(m') is the most general unifier η of m' or undefined if m' cannot be unified. With variables, constants, and nulls as the only terms, unification is an easy polynomial algorithm.

Processing continues with **check**⁺, called in Line 5 of **extend**⁺. We first partition **body**₂ into the matched atoms **body**₂^m, and the remaining atoms to the left **body**₂^ℓ and right **body**₂^r of the maximal index of m . Only **body**₂^r can still be considered for extending m . Six if-blocks check all conditions of Definition 1, and *true* is returned if all checks succeed. When a check fails, the search is either stopped (L10, L11, and L17) or recursively continued with an extended mapping (L12, L14, and L15). The three checks in L10–L12 cover cases where \mathcal{I}_a (L13) would need to contain nulls that are freshly introduced by ρ_1 only later. L10 applies, e.g., when checking $\rho_2 \prec^+ \rho_1$ for ρ_1, ρ_2 as in Example 2, where we would get $a(n) \in \mathcal{I}_a$ (note the swap of rule names compared to our present algorithm).

Further extensions of m are useless for L10, since they could only lead to more specific unifiers, and also for L11, where nulls occur in “earlier” atoms that are not considered in extensions of m . For case L12, however, moving further atoms from body_2^r to body_2^m might be promising, so we call extend^+ there.

In L14, we check if the constructed match of ρ_1 on \mathcal{I}_a is already satisfied. This might again be fixed by extending the mapping, since doing so makes body_2^r and hence \mathcal{I}_a smaller. If we reach L15, we have established condition (a) of Definition 1. L15 then ensures condition (c), which might again be repaired by extending the atom mapping so as to make \mathcal{I}_a smaller. Finally, L17 checks condition (b). If this fails, we can abort the search: unifying more atoms of body_2 with head_1 will only lead to a more specific \mathcal{I}_b and η , for which the check would still fail.

Theorem 1. *For rules ρ_1 and ρ_2 that (w.l.o.g.) do not share variables, $\rho_1 \prec^+ \rho_2$ iff $\text{extend}^+(\rho_1, \rho_2, \emptyset) = \text{true}$.*

5 Computing Restraints

We now turn our attention to the efficient computation of restraints. In spite of the rather different definitions, many of the ideas from Sect. 4 can also be applied here. The main observation is that the search for an alternative match can be realised by unifying a part of head_2 with head_1 in a way that resembles our unification of body_2 with head_1 in Sect. 4.

To realise this, we define a function extend^\square as a small modification of Algorithm 1, where we simply replace body_2 in L1 and L3 by head_2 , and check^+ in L5 by check^\square , which is defined in Algorithm 3 and explained next.

Algorithm 3: check^\square

Input: rules ρ_1, ρ_2 , atom mapping m with mgu η

Output: *true* if a restraint is found for m

```

19  $\text{head}_2^m \leftarrow \text{dom}(m)$ 
20  $\text{head}_2^< \leftarrow \{\text{head}_2[j] \in (\text{head}_2 \setminus \text{head}_2^m) \mid j < \text{maxidx}(m)\}$ 
21  $\text{head}_2^> \leftarrow \{\text{head}_2[j] \in (\text{head}_2 \setminus \text{head}_2^m) \mid j > \text{maxidx}(m)\}$ 
22 if  $x\eta \in \mathbf{N}$  for some  $x \in \mathbf{V}_\forall$  then return false
23 if  $z\eta \in \mathbf{N}$  for some  $z \in \mathbf{V}_\exists$  in  $\text{head}_2^<$  then return false
24 if  $z\eta \in \mathbf{N}$  for some  $z \in \mathbf{V}_\exists$  in  $\text{head}_2^>$  then
25    $\_ \text{return } \text{extend}^\square(\rho_1, \rho_2, m)$ 
26 if  $\text{head}_2^m$  contains no existential variables then
27    $\_ \text{return } \text{extend}^\square(\rho_1, \rho_2, m)$ 
28  $\tilde{\mathcal{I}}_a \leftarrow \text{body}_2\eta_\forall\omega_\forall$ 
29 if  $\tilde{\mathcal{I}}_a \models \exists z_2. \text{head}_2\eta_\forall\omega_\forall$  then return false
30  $\mathcal{I}_a \leftarrow \tilde{\mathcal{I}}_a \cup \text{head}_2\eta_\forall\omega$ 
31  $\tilde{\mathcal{I}}_b \leftarrow \mathcal{I}_a \cup (\text{body}_1 \cup \text{head}_2^< \cup \text{head}_2^>)\eta\omega$ 
32 if  $\tilde{\mathcal{I}}_b \models \exists z_1. \text{head}_1\eta_\forall\omega_\forall$  then return } \text{extend}^\square(\rho_1, \rho_2, m)
33 if  $\text{head}_2\eta\omega \subseteq \tilde{\mathcal{I}}_b$  then return } \text{extend}^\square(\rho_1, \rho_2, m)
34 return true

```

We use the notation for $\rho_1, \rho_2, \omega, \mathbf{V}_\exists$, and \mathbf{V}_\forall as introduced in Sect. 4, and again use atom mapping m to represent our current hypothesis for a possible match. What is new now is that unified atoms in $\text{dom}(m)$ can contain existentially quantified variables, though existential variables in the range of m (from head_1) are still replaced by nulls as in Algorithm 1, L5. An existential variable in head_2 might therefore be unified with a constant, null, or universal variable of head_1 . In the last case, where we need a unifier η with $z\eta = x\eta$ for $z \in \mathbf{V}_\exists$ and $x \in \mathbf{V}_\forall$, we require that $x\eta = z\eta \in \mathbf{V}_\forall$ so that η only maps to variables in \mathbf{V}_\forall . η simultaneously represents the matches h_1, h_2 , and h^A from Definition 3.

Example 5. For rules $\rho_1 = r(x, y) \rightarrow s(x, x, y)$ and $\rho_2 = a(z) \rightarrow \exists v. s(z, v, v) \wedge b(v)$, and mapping $m = \{s(z, v, v) \mapsto s(x, x, y)\}$, we obtain a unifier η that maps all variables to x (we could also use y , but not the existential v). Let $x\omega = c$ be the constant that x is instantiated with. Then we can apply ρ_2 to $\tilde{\mathcal{I}}_a = \{a(z)\eta\omega\} = \{a(c)\}$ with match $h_2 = \{z \mapsto c, v \mapsto n\}$ to get $\mathcal{I}_a = \tilde{\mathcal{I}}_a \cup \{s(c, n, n), b(n)\}$, and ρ_1 to $\tilde{\mathcal{I}}_b = \mathcal{I}_a \cup \{r(c, c), b(c)\}$ with match $h_1 = \{x \mapsto c, y \mapsto c\}$ to get $\mathcal{I}_b = \tilde{\mathcal{I}}_b \cup \{s(c, c, c)\}$. Note that we had to add $b(c)$ to obtain the required alternative match h^A , which maps n to $v\eta\omega = c$ and c to itself.

As in the example, a most general unifier η yields a candidate h^A that maps every null of the form $v\omega_\exists$ to $v\eta_\exists\omega_\forall$. Likewise, for $i \in \{1, 2\}$, $h_i = \eta_\forall\omega$ are the (extended) matches, while $\eta_\forall\omega_\forall$ are the body matches. The image of the instantiated $\text{head}_2\eta_\forall\omega$ under the alternative match h^A is given by $\text{head}_2\eta\omega$. The corresponding interpretations are $\mathcal{I}_a = \text{body}_2\eta_\forall\omega_\forall \cup \text{head}_2\eta_\forall\omega$ and $\mathcal{I}_b = \mathcal{I}_a \cup \text{body}_1\eta_\forall\omega_\forall \cup \text{head}_1\eta_\forall\omega \cup (\text{head} \setminus \text{dom}(m))\eta\omega$, where $(\text{head}_2 \setminus \text{dom}(m))\eta\omega$ provides additional atoms required for the alternative match but not in the mapped atoms of head_2 . With these intuitions, Algorithm 3 can already be understood.

It remains to explain the conditions that are checked before returning *true*. As before, we partition $\text{dom}(m)$ into mapped atoms head_2^m and left and right remainder atoms. Checks in L22–L24 ensure that the only variables mapped by η to nulls (necessarily from $\text{head}_1\omega_\exists$) are existential variables in head_2^m : such mappings are possible by h^A . Extending m further is only promising if the nulls only stem from atoms in head_2^r .

Check L26 continues the search when no atoms with existentials have been selected yet. Selecting other atoms first might be necessary by our order, but no alternative matches can exist for such mappings (yet). Lines L29 and L32 check that the matches h_1 and h_2 are indeed unsatisfied. Extending m might fix L29 by making $\tilde{\mathcal{I}}_a$ smaller, whereas L32 cannot be fixed. Finally, L33 ensures condition (d) of Definition 3.

Example 6. Consider rules $\rho_1 = b(x, y) \rightarrow r(x, y, x, y) \wedge q(x, y)$, $\rho_2 = a(u, v) \rightarrow \exists w. r(u, v, w, w) \wedge r(v, u, w, w)$, and mapping $m = \{r(u, v, w, w) \mapsto r(x, y, x, y)\}$. We obtain unifier η mapping all variables to a single universally quantified variable, say x . We reach $\mathcal{I}_b = \{a(c, c), r(c, c, n, n), b(c, c), r(c, c, c, c)\}$, based on $\tilde{\mathcal{I}}_a = \{a(c, c)\}$ ($x\omega = c$), for which ρ_1 is applicable but $h^A = \{n \mapsto c, c \mapsto c\}$ is already an alternative match on $\tilde{\mathcal{I}}_b$, recognized by L33.

Theorem 2. For rules ρ_1 and ρ_2 that (w.l.o.g.) do not share variables, $\rho_1 \prec^\square \rho_2$ holds according to Definition 3 for some $\mathcal{I}_a \neq \mathcal{I}_b$ iff $\text{extend}^\square(\rho_1, \rho_2, \emptyset) = \text{true}$.

The case $\mathcal{I}_a = \mathcal{I}_b$, which Theorem 2 leaves out, is possible [17, Example 5], but requires a slightly different algorithm. We can adapt Algorithm 3 by restricting to one rule, for which we map from atoms in `head` to atoms in `head ω_\exists` . The checks (for `head $_2$`) of Algorithm 3 remain as before, but we only need to compute a single \mathcal{I} that plays the role of \mathcal{I}_a and \mathcal{I}_b . Check L33 is replaced by a new check

if `head η_\exists = head ω_\exists` **then return false** ;

ensuring that at least one null is mapped differently in the alternative match. With these modifications, we can show an analogous result to Theorem 2 for the case $\mathcal{I}_a = \mathcal{I}_b$.

6 Implementation and Global Optimisations

We provide a C++ implementation of our algorithms, which also includes some additional optimisations and methods as described next. Our prototype is build on top of the free rule engine VLog (Release 1.3.5) [23], so that we can use its facilities for loading rules and checking MFA (see Sect. 7). Reasoning algorithms of VLog are not used in our code.

The algorithms of Sects. 4 and 5 use optimisations that are *local* to the task of computing dependencies for a single pair of rules. The quadratic number of potential rule pairs is often so large, however, that even the most optimised checks lead to significant overhead. We therefore build index structures that map predicates p to rules that use p in their body or head, respectively. For each rule ρ_1 , we then check $\rho_1 \prec^+ \rho_2$ only for rules ρ_2 that mention some predicate from `head(ρ_1)` in their body, and analogously for $\rho_1 \prec^\square \rho_2$.

Specifically for large rule sets, we further observed that many rules share the exact same structure up to some renaming of predicates and variables. For every rule pair considered, we therefore create an abstraction that captures the co-occurrence of predicates but not the concrete predicate names. This abstraction is used as a key to cache results of prior computations that can be re-used when encountering rule pairs with the exact same pattern of predicate names.

Besides these optimisations, we also implemented unoptimised variants of the algorithms of Sects. 4 and 5 to be used as a base-line in experiments. Instead of our goal-directed check-and-extend strategy, we simply iterate over all possible mappings until a dependency is found or the search is completed.

7 Evaluation

We have evaluated our implementation regarding (1) efficiency of our optimisations and (2) utility for solving practical problems. The latter also led to the first study of so-called *core stratified* real-world rule sets. Our evaluation machine is a mid-end server (Debian Linux 9.13; Intel Xeon CPU E5-2637v4@3.50 GHz; 384 GB RAM DDR4; 960 GB SSD), but our implementation is single-threaded and did not use more than 2 GB of RAM per individual experiments.

Experimental Data. All experiments use the same corpus of rule sets, created from real-world OWL ontologies of the *Oxford Ontology Repository* (<http://www.cs.ox.ac.uk/isg/ontologies/>). OWL is based on a fragment of first-order logic that overlaps with existential rules. OWL axioms that involve datatypes were deleted; any other axiom was syntactically transformed to obtain a Horn clause that can be written as a rule. This may fail if axioms use unsupported features, especially those related to (positive) disjunctions and equality. We dropped ontologies that could not fully be translated or that required no existential quantifier in the translation. Thereby 201 of the overall 787 ontologies were converted to existential rules, corresponding largely to those ontologies in the logic Horn-*SRI* [18]. The corpus contains 63 small (18–1,000 rules), 90 medium (1,000–10,000 rules), and 48 large (10,000–167,351 rules) sets. Our translation avoided normalisation and auxiliary predicates, which would profoundly affect dependencies. This also led to larger rule bodies and heads, both ranging up to 31 atoms.

Table 1. Number of rule sets achieving a given order of magnitude of speed-up for computing \prec^+ (left) and \prec^\square (right) from one variant to another; t.o. gives the number of avoided timeouts

	≥ 1	<10	$<10^2$	$<10^3$	$>10^3$	t.o.	≥ 1	<10	$<10^2$	$<10^3$	$>10^3$	t.o.
N/L	48	104	14	1	2	32	53	92	17	2	2	35
G/A	103	67	9	1	0	21	90	81	9	1	0	20
N/G	24	1	27	33	60	56	35	11	53	30	20	52
L/A	5	33	30	41	47	45	17	72	48	10	17	37

Optimisation Impact. We compare four software variants to evaluate the utility of our proposed optimisations. Our baseline N is the unoptimised version described in Sect. 6, while L uses the locally optimised algorithms of Sects. 4 and 5. Version G is obtained from N by enabling the global optimisations of Sect. 6, and A combines all optimisations of L and G. For each of the four cases, we measured the total time of determining all positive reliances and all restraints for each rule set. A timeout of 60sec was used. The number of timeouts for each experiment was as follows:

\prec^+	N	L	G	A	\prec^\square	N	L	G	A
	80	48	24	3		87	52	35	15

To present the remaining results, we focus on *speed-up*, i.e., the ratio of runtime of a less optimised variant over runtime of a more optimised one. Table 1 classifies the observed speed-ups in several scenarios by their order of magnitude. For example, in the left table, the number 14 in line N/L and column “ $<10^2$ ” means that for 14 of the 201 rule sets, L was between 10 – 10^2 times faster than

N. Note that G/A shows the effect of adding *local* optimisations to G. Column “=1” shows cases where both variants agree, and column “t.o.” cases where the optimisation avoided a prior timeout (the speed-up cannot be computed since the timeout does not correspond to a time).

We conclude that both L and G can lead to significant performance gains across a range of ontologies. Strong effects are seen against the baseline (N/L and N/G), but also (to a slightly lesser extent) against variants with the other optimisations (G/A and L/A). Overall, \prec^{\square} turned out to be slower than \prec^+ , with the global optimisations being less effective.

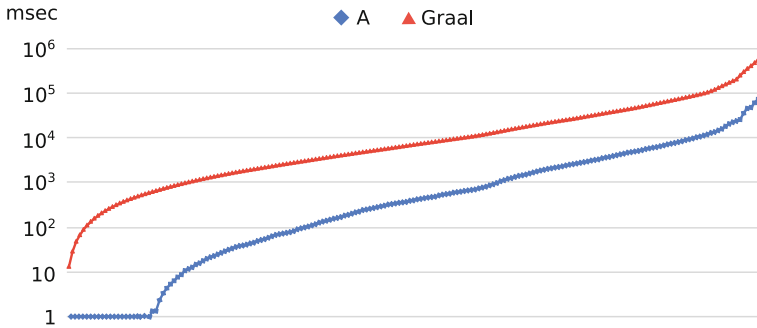


Fig. 1. Positive reliance computation in Graal (top) and our system (bottom)

Acyclic Positive Reliances. For rule sets where the graph of positive reliances is acyclic, query answering is possible with many existing rule engines [1]. To evaluate how our work compares to the state of the art in computing this graph, we measure the time taken by Graal to find all positive reliances and compare them to our prototype A from above. The results are shown in Fig. 1.

Our approach consistently outperformed Graal by about one order of magnitude. Overall, we can classify 178 ontologies in under 1 s, making this analysis feasible at reasoning time. The difference in execution time is explained by our optimisations: given two rules ρ_1 and ρ_2 , Graal computes all (exponentially many in the worst case) different ways to unify the $\text{head}(\rho_1)$ with $\text{body}(\rho_2)$ while our implementation (1) stops when a positive reliance is discovered, (2) discards atom mappings when a negative result is guaranteed, and (3) caches results of previous computations. Recall that Graal uses a slightly weaker notion of positive reliance (cf. Sect. 3), which leads to more cycles: we find 36 acyclic sets in Graal, but 70 such sets in our system.

Faster MFA. *Model-faithful acyclicity* (MFA) is an advanced analysis of rule sets that can discover decidability of query answering in many cases, but is 2EXPTIME-complete [10]. However, instead of performing this costly analysis on the whole rule set, an equivalent result can be obtained by analysing each strongly connected components of the \prec^+ -graph individually. We measure the

times for both approaches using the MFA implementation of VLog and our optimised variant A, with a timeout of 30min per rule set. The two variants are denoted V (VLog MFA) and C (component-wise MFA).

Using C, 163 ontologies are classified as MFA, 33 fail MFA, and 5 cases time out. V times out in 10 cases, but agrees on all other outcomes. C is slower in three cases that still run in under 50 ms. The numbers of speed-ups, grouped by order of magnitude, are as follows:

Speed-up	= 1	< 10	< 10 ²	< 10 ³	≥ 10 ³
V/C	0	85	54	41	11

We conclude that our optimised reliance computation is a feasible approach for speeding up MFA analysis.

Core Stratification. We can use our implementation to determine how common this favourable property (cf. Sect. 3) is among real-world ontologies. The analysis was feasible for 200 rule sets in our corpus, yielding 44 core stratified sets with up to 121,712 rules. One can improve this result by considering *pieces*, minimal subsets of rule heads where each two atoms refer to a common existentially quantified variable [1]. Each rule can then equivalently be replaced by several rules, each combining the original body with one of the pieces of the original head. Applying this transformation to our rule sets leads to more fine-grained dependencies that have fewer cycles over \prec^\square . With this modification, 75 rule sets are core stratified.

Our implementation fails in one case (ontology ID 00477), containing 167,351 rules like $A(x) \rightarrow \exists v. \text{located-in}(x, v) \wedge B(v)$, for various A and B . The required $> 28 \times 10^9$ checks, though mostly cached, take very long. In spite of many \prec^\square -relations, the set is core-stratified as it describes a proper meronymy.

The remaining 125 rule sets are not core stratified. To validate the outcome, we have analysed these sets manually, and found several common reasons why ontologies were indeed not core stratified (and therefore correctly classified in our implementation). The following two examples explain two typical situations.

Example 7. In some cases, core stratification fails even though there is a natural rule application order that always leads to a core. Consider the rules $\rho_1 = a(x) \rightarrow \exists v. r(x, v) \wedge b(v)$, $\rho_2 = r(x, y) \rightarrow s(y, x)$, and $\rho_3 = s(x, y) \rightarrow r(y, x)$. This set is not core stratified since we have $\rho_1 \prec^+ \rho_3$, $\rho_2 \prec^+ \rho_3$, $\rho_3 \prec^+ \rho_2$, and $\rho_3 \prec^\square \rho_1$. However, prioritising ρ_2 and ρ_3 over ρ_1 (i.e., using a *Datalog-first* strategy [9]) always leads to a core. Indeed, the positive reliance $\rho_1 \prec^+ \rho_3$ over-estimates relevant rule applications, since no new atom produced by ρ_1 can (indirectly) lead to an application of ρ_3 .

Example 8. In other cases, there is indeed no data-independent strategy for rule applications that would always lead to a core. Consider the rules $\rho_1 = a(x) \rightarrow \exists v. r(x, v) \wedge b(v)$ and $\rho_2 = r(x, y) \wedge r(y, z) \rightarrow r(x, z)$. Both are common in OWL

ontologies with existential axioms and transitive roles. The rule set is not core stratified since $\rho_1 \prec^+ \rho_2$ and $\rho_2 \prec^\square \rho_1$.

Consider $\mathcal{I}_a = \{a(1), a(2), r(1, 2)\}$. Applying ρ_1 over \mathcal{I}_a to all matches yields $\mathcal{I}_b = \mathcal{I}_a \cup \{r(1, n), b(n), r(2, m), b(m)\}$, which makes ρ_2 applicable to obtain $\mathcal{I}_c = \mathcal{I}_b \cup \{r(1, m)\}$. Here we have the alternative match $h^A = \{1 \mapsto 1, 2 \mapsto 2, n \mapsto m\}$.

In contrast, applying ρ_1 only for the match $\{x \mapsto 2\}$ produces $\mathcal{I}'_b = \mathcal{I}_a \cup \{r(2, n), b(n)\}$. A subsequent application of ρ_2 yields $\mathcal{I}'_c = \mathcal{I}'_b \cup \{r(1, n)\}$, which is a core model. Indeed, core models could often be achieved in such settings, but require fine-grained, data-dependent strategies that cannot be found by static analysis (concretely: we could consider r as a pre-order and apply ρ_1 to the r -greatest elements first, followed by an exhaustive application of ρ_2).

Overall, our manual inspection supported the correctness of our computation and led to interesting first insights about core stratification in practical cases. Regarding the contribution of this work, our main conclusion of this evaluation is that our proposed algorithms are able to solve real-world tasks that require the computation of positive reliances and restraints over large ontologies.

8 Conclusions

We have shown that even the complex forms of dependencies that arise with existential rules can be implemented efficiently, and that doing so enables a number of uses of practical and theoretical interest. In particular, several previously proposed approaches can be made significantly faster or implemented for the first time at all. Our methods can be adapted to cover further cases, especially the *negative reliances*.

Our work opens up a path towards further uses of reliance-based analyses in practice. Already our experiments on core stratification – though primarily intended to evaluate the practical feasibility of our restraint algorithm – also showed that (a) core stratification does occur in many non-trivial real-world ontologies, whereas (b) there are also relevant cases where this criterion fails although a rule-based core computation seems to be within reach. This could be a starting point for refining this notion. It is also interesting to ask whether good ontology design should, in principle, lead to specifications that naturally produce cores, i.e., that robustly avoid redundancies. A different research path is to ask how knowledge of dependencies can be used to speed up reasoning. Indeed, dependencies embody characteristics of existential rule reasoning that are not found in other rule languages, and that therefore deserve further attention.

Supplemental Material Statement. We provide full proofs in the technical report published on arXiv [14]. Our source code, experimental data, instructions for repeating all experiments, and our own raw measurements are available on [GitHub](#).

Acknowledgments. This work is partly supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in project 389792660 (TRR 248, [Center for Perspicuous Systems](#)), by the Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research) under European ITEA project 01IS21084 ([InnoSale, Innovating Sales and Planning of Complex Industrial Products Exploiting Artificial Intelligence](#)) and [Center for Scalable Data Analytics and Artificial Intelligence](#) (ScaDS.AI), by BMBF and DAAD (German Academic Exchange Service) in project 57616814 ([SECAI, School of Embedded and Composite AI](#)), and by the [Center for Advancing Electronics Dresden](#) (cfaed).

References

1. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: walking the decidability line. *Artif. Intell.* **175**(9–10), 1620–1654 (2011)
2. Baget, J.-F., Leclère, M., Mugnier, M.-L., Rocher, S., Sipieter, C.: Graal: a toolkit for query answering with existential rules. In: Bassiliades, N., Gottlob, G., Sadri, F., Paschke, A., Roman, D. (eds.) *RuleML 2015*. LNCS, vol. 9202, pp. 328–344. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21542-6_21
3. Bellomarini, L., Sallinger, E., Gottlob, G.: The vadalog system: datalog-based reasoning for knowledge graphs. *Proc. VLDB Endow.* **11**(9), 975–987 (2018)
4. Bourgaux, C., Carral, D., Krötzsch, M., Rudolph, S., Thomazo, M.: Capturing homomorphism-closed decidable queries with existential rules. In: Bienvenu, M., Lakemeyer, G., Erdem, E. (eds.) *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*, pp. 141–150. IJCAI (2021)
5. Cali, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. *J. Web Semant.* **14**, 57–83 (2012)
6. Cali, A., Gottlob, G., Pieris, A.: Towards more expressive ontology languages: the query answering problem. *J. Artif. Intell.* **193**, 87–128 (2012)
7. Carral, D., Dragoste, I., González, L., Jacobs, C., Krötzsch, M., Urbani, J.: VLog: a rule engine for knowledge graphs. In: Ghidini, C., et al. (eds.) *ISWC 2019*. LNCS, vol. 11779, pp. 19–35. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30796-7_2
8. Carral, D., Dragoste, I., Krötzsch, M.: The combined approach to query answering in Horn-*ALC^HOIQ*. In: Thielscher, M., Toni, F., Wolter, F. (eds.) *Proceedings of 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018)*, pp. 339–348. AAAI Press (2018)
9. Carral, D., Dragoste, I., Krötzsch, M., Lewe, C.: Chasing sets: how to use existential rules for expressive reasoning. In: Kraus, S. (ed.) *Proceedings of 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pp. 1624–1631. ijcai.org (2019)
10. Cuenca Grau, B., et al.: Acyclicity notions for existential rules and their application to query answering in ontologies. *J. Artif. Intell. Res.* **47**, 741–808 (2013)
11. Deutsch, A., Nash, A., Rummel, J.B.: The chase revisited. In: Lenzerini, M., Lembo, D. (eds.) *Proceedings of 27th Symposium on Principles of Database Systems (PODS 2008)*, pp. 149–158. ACM (2008)
12. Ellmauthaler, S., Krötzsch, M., Mennicke, S.: Answering queries with negation over existential rules. In: *Proceedings of AAAI Conference on Artificial Intelligence*, vol. 36, no. 5, pp. 5626–5633. AAAI Press (2022)

13. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. *Theoret. Comput. Sci.* **336**(1), 89–124 (2005)
14. González, L., Ivliev, A., Krötzsch, M., Mennicke, S.: Efficient dependency analysis for rule-based ontologies. *CoRR* abs/2207.09669 (2022). <https://arxiv.org/abs/2207.09669>
15. Grahne, G., Onet, A.: Anatomy of the chase. *Fundam. Inform.* **157**(3), 221–270 (2018)
16. Hogan, A.: Canonical forms for isomorphic and equivalent RDF graphs: algorithms for leaning and labelling blank nodes. *ACM Trans. Web* **11**(4), 1–62 (2017). <https://doi.org/10.1145/3068333>
17. Krötzsch, M.: Computing cores for existential rules with the standard chase and ASP. In: Calvanese, D., Erdem, E., Thielscher, M. (eds.) *Proceedings of 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)*, pp. 603–613. *IJCAI* (2020)
18. Krötzsch, M., Rudolph, S., Hitzler, P.: Complexities of Horn description logics. *ACM Trans. Comput. Logic* **14**(1), 2:1–2:36 (2013)
19. Magka, D., Krötzsch, M., Horrocks, I.: Computing stable models for nonmonotonic existential rules. In: Rossi, F. (ed.) *Proceedings of 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pp. 1031–1038. *AAAI Press/IJCAI* (2013)
20. Mallea, A., Arenas, M., Hogan, A., Polleres, A.: On blank nodes. In: Aroyo, L., et al. (eds.) *ISWC 2011. LNCS*, vol. 7031, pp. 421–437. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25073-6_27
21. Meier, M., Schmidt, M., Lausen, G.: On chase termination beyond stratification. *PVLDB* **2**(1), 970–981 (2009)
22. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: RDFox: a highly-scalable RDF store. In: Arenas, M., et al. (eds.) *ISWC 2015. LNCS*, vol. 9367, pp. 3–20. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25010-6_1
23. Urbani, J., Jacobs, C., Krötzsch, M.: Column-oriented datalog materialization for large knowledge graphs. In: Schuurmans, D., Wellman, M.P. (eds.) *Proceedings of 30th AAAI Conference on Artificial Intelligence (AAAI 2016)*, pp. 258–264. *AAAI Press* (2016)