Efficient Dependency Analysis for Existential Rules

Larry González, Alex Ivliev, Markus Krötzsch and Stephan Mennicke

Knowledge-Based Systems Group
Faculty of Computer Science / cfaed / ScaDS.AI /
Centre for Tactile Internet with Human-in-the-Loop (CeTI)
TU Dresden

Abstract
This short paper reviews the main contributions of our recent work on static analysis of existential rules (a.k.a. tuple-generating dependencies). Between such rules, several kinds of logical relationships – also called *dependencies* in an unfortunate clash of terminology – are of interest, but their computation highly intractable (Σ_2^P-complete). We develop new, optimised procedures for this task, and present a prototype implementation that scales to rule sets with more than 100,000 rules. This allows us to perform much faster acyclicity checks and to identify rule sets that admit efficient core computation via the standard chase.

1. Introduction

Existential rules (or *tuple-generating dependencies*) are a versatile logical formalism with relevance in databases [1, 2, 3], ontological reasoning [4, 5, 6, 7], and declarative computing in general [8, 9, 10]. An existential rule (or just *rule*) ρ is a formula

$$\rho = \forall \boldsymbol{x}, \boldsymbol{y}.\, \varphi[\boldsymbol{x}, \boldsymbol{y}] \to \exists \boldsymbol{z}.\, \psi[\boldsymbol{y}, \boldsymbol{z}], \tag{1}$$

where φ and ψ are conjunctions of first-order atoms that may use variables from the (mutually disjoint) sets $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ as indicated. We call φ the *body* (denoted $\mathsf{body}(\rho)$) and ψ the *head* (denoted $\mathsf{head}(\rho)$). Universal quantifiers are usually omitted.

Computing with such rules can be challenging, especially since a "forward" application of rules (i.e., *materialisation* of consequences) requires new *nulls* to be introduced in order to satisfy existential quantifiers. Many variants of the *chase* procedure [11, 1] offer strategies for handling the two major complications that this brings:

1. Non-termination: If rules recursively introduce new nulls, the outcome of the chase might not be finite.
2. Redundancy: Nulls that seem necessary during the chase might actually be redundant if other domain elements suffice to satisfy the same conditions.

AMW'23: 15th Alberto Mendelzon International Workshop on Foundations of Data Management, May 22–26, 2023, Santiago, Chile
larry.gonzalez@tu-dresden.de (L. González); alex.ivliev@tu-dresden.de (A. Ivliev); markus.kroetzsch@tu-dresden.de (M. Krötzsch); stephan.mennicke@tu-dresden.de (S. Mennicke)
0000-0001-9412-9363 (L. González); 0000-0002-1604-6308 (A. Ivliev); 0000-0002-9172-2601 (M. Krötzsch); 0000-0002-3293-2940 (S. Mennicke)

The challenge of non-termination has inspired much research in *acyclicity* conditions, which suffice to ensure chase termination (Cuenca Grau et al. offer an overview [7], though further approaches have since been proposed [12]). Redundancy in turn can be eliminated by computing *cores* [1], and this is crucial especially when adding non-monotonic features [13, 14].

Approaches to both of the above challenges may leverage logical *dependencies* between rules (which have also been called *reliances* to avoid the name clash with database *dependencies* meaning *rules*). Early works focused 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 acyclity [4, 1, 15].[1] When adding negation, a rule might also inhibit another, and such *negative reliances* are used to define semantically well-behaved fragments of non-monotonic existential rules [14, 16]. A third kind of dependency are *restraints*, which indicate that the application of one rule might render another rule application redundant (which would be detected in the *standard chase*, a.k.a. *restricted chase*). Restraints are used to define *core-stratified rule sets* [14] and a well-behaved semantics for queries with negation [13].

Surprisingly, given this breadth of applications, rule dependencies are hardly supported in practice. Besides our work, we know just one tool for positive reliances (Graal [17]), and none for negative reliances or restraints. Indeed, the high complexity of the problem (typically Σ_2^P-complete) is a challenge for taking advantage of such analysis in time-critical tasks. Moreover, as opposed to many other static analyses, dependency computation is not mainly an application of available (chase-like) algorithms.

In our work [18], we have therefore developed new optimised algorithms for the computation of positive reliances and restraints, and evaluated their performance on real-world rule sets with large numbers of rules (mainly obtained by converting ontologies). Our key contributions were:

- optimised methods for the Σ_2^P-complete tasks of checking dependencies,
- optimisations for computing all dependencies in large rule sets,
- a prototype C++ implementation, and
- evaluations of the individual optimisations, and of the potential of these analyses for solving common tasks at realistic scales.

The use cases we evaluated are checking *acyclicity of the graph of rule dependencies* [4], speeding up the check for *model-faithful acyclicity* [7], and checking *core stratification* of rule sets [14]. The latter condition ensures that the outcome of the (expensive, impractical) *core chase* can be obtained by the (implemented, practical) standard chase.

2. Positive Reliances and Restraints

Many readers will be familiar with our notation – detailed preliminaries are found in the long paper [18]. We consider first-order interpretations $\mathcal{I}$ that may contain constants and named nulls, where databases can be considered finite interpretations. For a rule ρ, a homomorphism

[1]Deutsch et al. [1] were often overlooked in related works on that specific topic, maybe due to their choice, in the face of the obvious terminological dilemma, to not name the concept at all. Of course, the basic idea of positive dependency still predates their work and is already found in the definition of stratified and acyclic (non-recursive) logic programs.

$h : \mathsf{body}(\rho) \to \mathcal{I}$ is called a *match* of ρ in $\mathcal{I}$. A match is *satisfied* if it can be extended to a homomorphism $h' : \mathsf{head}(\rho) \to \mathcal{I}$. If a match is not satisfied, it can be made so by *applying* the rule, introducing new nulls for existential variables as required. A sequence of iterative, fair rule applications is called *standard chase*. We capture the idea that an application of ρ_1 immediately enables a new application of ρ_2:

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,
(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 $\mathcal{I}_a$.

For example, consider rules $\rho_1 = p(x) \to \exists v. r(x, v) \wedge q(v)$ and $\rho_2 = s(x) \wedge r(x, y) \to t(y)$. Then $\rho_1 \prec^+ \rho_2$ with $\mathcal{I}_a = \{p(a), s(a)\}$ and $\mathcal{I}_b = \mathcal{I}_a \cup \{r(a, n), q(n)\}$. On the other hand, the rule $\rho_3 = s(y) \wedge r(x, y) \to t(y)$ does not positively rely on ρ_1, since the required precondition $s(y)$ cannot hold for the null n newly introduced when applying ρ_1.

Detecting positive reliances is feasible in Σ^{P}_2 since the relevant interpretations $\mathcal{I}_a$ and $\mathcal{I}_b$ can be small (restricted to images of atoms found in the rules), but it inherits the Σ^{P}_2-hardness of deciding if a rule has an unsatisfied match [3]. If the relation $\prec^+$ has no cycles on a given set of rules Σ, then Σ is *agrd* ("acyclic graph of rule dependencies") and the chase is guaranteed to terminate on all databases [4].

The second kind of dependencies we consider are *restraints*. We say that ρ_1 *restrains* ρ_2 if it is possible that an application of ρ_1 make a null obsolete that was introduced by ρ_2. Being "obsolete" is captured by the concept of an *alternative match*, first introduced in the study of cores [14]:

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 the match h that is extended to h'. A homomorphism $h^A : h'(\mathsf{head}(\rho)) \to \mathcal{I}_b$ is an* alternative match *of h' and ρ on $\mathcal{I}_b$ if*

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

An alternative match therefore specifies a way in which (part) of a rule head might have been satisfied with fewer new nulls. A restraint is now defined as a dependency between rules where one rule might create a new alternative match for the other. As for positive reliances, deciding restraints is Σ^{P}_2-complete. Instead of reproducing the formal definition, which is similar to Definition 1 [18], we show an example:

Example 1. *The rule $\rho_1 = r(x, y) \to s(y, x)$ restrains $\rho_2 = r(x, y) \to \exists v.s(y, v)$, denoted $\rho_1 \prec^\square \rho_2$. Indeed, given $\mathcal{I} = \{r(a, b)\}$, we might apply ρ_2 to get $\mathcal{I}_a = \{r(a, b), s(b, n)\}$ with n a new null; applying ρ_1 now yields $\mathcal{I}_b = \mathcal{I}_a \cup \{s(b, a)\}$, which enables an alternative match for the previous application of ρ_2 (using b instead of n as a value for v). In many cases, such redundant nulls can be avoided by applying rules in an order that respects restraints, which is the intuition behind the notion of* core stratification *[14].*

3. Computing Positive Reliances and Restraints

In this section, we describe the optimisation techniques we implemented to compute positive reliances and restraints. We distinguish *local optimisations* that focus on determining the dependency between a pair of rules, and *global optimisation* that improve performance for computing all dependencies for a rule set. Complete algorithms and details on our optimisations are in the long version of this paper [18].

Local Optimisations We consider two rules ρ_1 and ρ_2 of the form $\rho_i = \mathsf{body}_i \to \exists \boldsymbol{z}_i.\, \mathsf{head}_i$, and we want to check if $\rho_1 \prec^+ \rho_2$. It turns out that the interpretations $\mathcal{I}_a$ and $\mathcal{I}_b$ in Definition 1 can be assumed to contain only atoms that occur in ρ_1 or ρ_2. Furthermore, the definition requires that ρ_1 must produce atoms that can be directly matched by ρ_2. Therefore, our algorithms search through potential *mapped* subsets $\mathsf{body}_2^m \subseteq \mathsf{body}_2$, trying to find a substitution η such that $\mathsf{body}_2^m\eta \subseteq \mathsf{head}_1\eta$. Such an η can represent the matches required in Definition 1.

Unfortunately, one can not restrict to single atoms here: in the worst case, we may need to analyse all subsets $\mathsf{body}_2^m \subseteq \mathsf{body}_2$, starting from singleton sets followed by extending those sets atom by atom. To do this, we execute a *depth-first* search over all body_2^m, but we also prune the search space in cases where we find that adding further atoms to body_2^m does not have a chance of success. We construct mapped sets in a lexicographic order that prevents sets from being considered more than once, and we stop as soon as a positive reliance is detected.

The optimised search for restraints uses similar ideas, but considers slightly different cases. In both algorithms, the key is the careful analysis of cases where the search can be aborted early.

Global Optimisations Even with very efficient algorithms to compare a pair of rules, the quadratic number of possible pairs can be prohibitive in large rule sets. To mitigate this, we create an index that allows us to restrict attention to pairs of rules that share predicate names (in head and body for positive reliances, and in head and head for restraints). Moreover, we developed a structure-based similarity detection that allows us to cache and reuse results for isomorphic pairs of rules. Indeed, large rule sets often contain rules of recurring, uniform shapes that are amenable to such a caching strategy.

4. Evaluation Results

We have implemented the algorithms on top of VLog (Release 1.3.5), a free existential rule engine [19]. Using our prototype, we have evaluated the algorithms regarding (1) effectiveness of the individual optimisations, and (2) utility for solving practical problems. All experiments were performed on 201 rule sets, generated from the *Oxford Ontology Repository*. We give a short overview of the main outcomes of our experiments here; further details and results can be found in the full publication [18].

In the first part of the evaluation, we found that local and global optimisations lead to significant performance gains, both individually (comparing only-local/only-global to a baseline) and in combination (comparing global+local to only-local/only-global). We conclude that both kinds of methods are justified and practically useful. We also observed that the computation of positive reliances was generally faster than the computation of restraints, and that for computing restraints, local optimisations were more effective than global optimisations.

For the second part of our evaluation, we used our prototype for checking rule sets for (a) *acyclicity of the graph of rule dependencies* (agrd), (b) *model-faithful acyclicity* (MFA), and (c) *core-stratification* of real-world rule sets.

For task (a), our prototype consistently outperformed Graal [17], the only other agrd implementation. Furthermore, we found more acyclic rule sets because our notion of positive reliances is stricter than the one used by Graal, while still guaranteeing termination of acyclic rule sets.

For task (b), we took advantage of the fact that MFA computation can equivalently be performed individually on strongly connected components of the dependency graph of task (a). Comparing to the native MFA implementation in VLog, this refined approach led to much faster checking times in spite of the additional effort of computing dependencies.

For task (c), we checked if rule sets are *core-stratified*, i.e., if the graph of positive reliances and restraints does not contain a cycle through a restraint edge. In this case, we obtain a strategy of applying rules in such a way that the standard chase produces a *core model*, without requiring the (infeasibly expensive) additional computations of the core chase [14]. Among the 201 ontologies we analysed, 44 were found to be core-stratified, and this number went up to 75 when decomposing large rule heads into *pieces* (a well-known, semantically equivalent transformation [4]). These results are the first to show that there are real-world rule sets with this favourable theoretical property.

5. Conclusions and Outlook

We have shown that even the complex forms of reliances arising for existential rules can be implemented efficiently, and that doing so enables applications of practical and theoretical interest. In particular, several previously proposed approaches can be made significantly faster or implemented for the first time at all.

As a further step, our methods can be readily adapted to cover *negative reliances*. A different research path is to ask how knowledge of dependencies can be used to speed up the chase. Indeed, dependencies embody characteristics of existential rule reasoning that are not found in other rule languages and, therefore, deserve further attention.

On the practical side, we are currently integrating our techniques into our recently released rule engine *Nemo* [20],[2] an all-new Rust implementation that focusses on fast and scalable in-memory data processing. In this context, rule dependencies will play a role for improving chase performance, for obtaining better results (preferably core models), and for providing additional services like acyclicity checking.

Acknowledgments This work was supported in DFG grant 389792660 (TRR 248, Center for Perspicuous Systems), by BMBF in grants ScaDS.AI, ITEA-01IS21084 (InnoSale), and 13GW0552B (KIMEDS), and in DAAD grant 57616814 (SECAI).

[2]https://github.com/knowsys/nemo/

References

[1] A. Deutsch, A. Nash, J. B. Remmel, The chase revisited, in: Proc. PODS'08, ACM, 2008, pp. 149–158.

[2] R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa, Data exchange: semantics and query answering, Theoretical Computer Science 336 (2005) 89–124.

[3] G. Grahne, A. Onet, Anatomy of the chase, Fundam. Inform. 157 (2018) 221–270.

[4] J.-F. Baget, M. Leclère, M.-L. Mugnier, E. Salvat, On rules with existential variables: Walking the decidability line, Artif. Intell. 175 (2011) 1620–1654.

[5] A. Calì, G. Gottlob, T. Lukasiewicz, A general Datalog-based framework for tractable query answering over ontologies, J. Web Semant. 14 (2012) 57–83.

[6] A. Calì, G. Gottlob, A. Pieris, Towards more expressive ontology languages: The query answering problem, Artif. Intell. 193 (2012) 87–128.

[7] B. Cuenca Grau, I. Horrocks, M. Krötzsch, C. Kupke, D. Magka, B. Motik, Z. Wang, Acyclicity notions for existential rules and their application to query answering in ontologies, J. of Artificial Intelligence Research 47 (2013) 741–808.

[8] L. Bellomarini, E. Sallinger, G. Gottlob, The Vadalog system: Datalog-based reasoning for knowledge graphs, Proc. VLDB Endowment 11 (2018) 975–987.

[9] C. Bourgaux, D. Carral, M. Krötzsch, S. Rudolph, M. Thomazo, Capturing homomorphism-closed decidable queries with existential rules, in: Proc. 18th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'21), IJCAI, 2021, pp. 141–150.

[10] D. Carral, I. Dragoste, M. Krötzsch, C. Lewe, Chasing sets: How to use existential rules for expressive reasoning, in: Proc. IJCAI'19, IJCAI, 2019, pp. 1624–1631.

[11] M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, P. Papotti, D. Santoro, E. Tsamoura, Benchmarking the chase, in: Proc. PODS'17, ACM, 2017, pp. 37–52.

[12] D. Carral, I. Dragoste, M. Krötzsch, Restricted chase (non)termination for existential rules with disjunctions, in: Proc. IJCAI'17, 2017, pp. 922–928.

[13] S. Ellmauthaler, M. Krötzsch, S. Mennicke, Answering queries with negation over existential rules, in: Proc. 36th AAAI Conf. on Artificial Intelligence (AAAI'22), 2022, pp. 5626–5633.

[14] M. Krötzsch, Computing cores for existential rules with the standard chase and ASP, in: Proc. 17th Int. Conf. on Princ. of Knowl. Repr. and Reasoning (KR'20), 2020, pp. 603–613.

[15] M. Meier, M. Schmidt, G. Lausen, On chase termination beyond stratification, PVLDB 2 (2009) 970–981.

[16] D. Magka, M. Krötzsch, I. Horrocks, Computing stable models for nonmonotonic existential rules, in: Proc. IJCAI'13, AAAI Press/IJCAI, 2013, pp. 1031–1038.

[17] J. Baget, M. Leclère, M. Mugnier, S. Rocher, C. Sipieter, Graal: A toolkit for query answering with existential rules, in: Proc. RuleML'15, Springer, 2015, pp. 328–344.

[18] L. González, A. Ivliev, M. Krötzsch, S. Mennicke, Efficient dependency analysis for rule-based ontologies, in: Proc. ISWC'22, Springer, 2022, pp. 267–283.

[19] J. Urbani, C. Jacobs, M. Krötzsch, Column-oriented Datalog materialization for large knowledge graphs, in: Proc. AAAI'16, AAAI Press, 2016, pp. 258–264.

[20] A. Ivliev, S. Ellmauthaler, L. Gerlach, M. Marx, M. Meißner, S. Meusel, M. Krötzsch, Nemo: First glimpse of a new rule engine, in: Proc. 39th Int. Conf. on Logic Programming, ICLP 2023 Technical Communications, EPTCS, to appear.