

Reasoning in Warded Datalog \pm with Harmful Joins

Teodoro Baldazzi¹, Luigi Bellomarini², Emanuel Sallinger^{3,4} and Paolo Atzeni¹

¹Università Roma Tre, Department of Computer Science and Engineering, Rome, Italy

²Banca d'Italia, Italy

³TU Wien, Faculty of Informatics, Vienna, Austria

⁴University of Oxford, Department of Computer Science, Oxford, UK

Abstract

Warded Datalog \pm is a powerful member of the Datalog \pm family of logic languages, featuring full support for recursion and existential quantification. As a result of the promising trade-off between expressive power and data complexity offered, it recently rose as a relevant candidate for ontological reasoning on large knowledge graphs and is employed as logic core of the Vadalog system, a well-known state-of-the-art reasoner. To achieve decidability and data tractability in practice over recursive settings, reasoners such as Vadalog adopt specialized strategies that control the effects of recursion and ensure reasoning termination with small memory footprint. However, exploiting the theoretical underpinnings of the Warded fragment, to enable these strategies, requires the settings not to contain “harmful” joins, i.e., on variables affected by existential quantification. To support reasoning decidability and the full expressive power of the language, we developed the Harmful Join Elimination, an algorithm that removes such joins while preserving the correctness of the task, and we integrated it into the Vadalog system.

Keywords

Datalog, Vadalog, ontological reasoning, existential quantification, harmful joins

1. Introduction

Recent years’ widespread interest towards querying and exploiting large amounts of data in the form of *knowledge graphs* (KGs) has led to the rising development and adoption of intelligent systems that manage such extensional knowledge and infer new intensional one via efficient *ontological reasoning* mechanisms [1]. To achieve this, modern reasoning and KG navigation applications require to employ powerful formalisms and logic languages for knowledge representation [2], capable of providing full support for recursion and existential quantification as well as sustaining decidability and polynomial data complexity of the task [3].

Datalog \pm [4, 5, 6, 7] is one of the most commonly adopted families of logic languages (technically, *fragments*) for reasoning on KGs [6]. Among them, Warded Datalog \pm [8] effectively covers the above requirements. Indeed, it encompasses both a high expressiveness, capturing plain Datalog as well as SPARQL queries under OWL 2 QL entailment regime and set semantics, and a simple syntax that enable powerful knowledge-modeling. It also offers a very good trade-off with computational complexity and captures PTIME for the reasoning [9]. Its semantics is defined via the CHASE [10], an algorithmic tool that takes as input a database D and a set Σ of


SEBD 2022: The 30th Italian Symposium on Advanced Database Systems, June 19-22, 2022, Tirrenia (PI), Italy

✉ teodoro.baldazzi@uniroma3.it (T. Baldazzi); luigi.bellomarini@bancaditalia.it (L. Bellomarini);

sallinger@dbai.tuwien.ac.at (E. Sallinger); paolo.atzeni@uniroma3.it (P. Atzeni)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

rules, and adds new tuples to D until Σ is satisfied: such tuples may contain freshly generated symbols ν_i (technically, *labelled nulls*) that act as placeholders for the existentially quantified variables [6]. The Warded fragment is employed in the *Vadalog system* [11], a state-of-the-art reasoner that allows to perform ontological reasoning tasks in complex scenarios.

While such favourable characteristics of Warded Datalog[±] bode well for efficient implementations, the interplay between recursion and existential quantification may lead to the generation of infinite labelled nulls in the CHASE, causing the procedure not to terminate and thus inhibiting decidability of the reasoning task in practice [5]. To properly control such interactions, reasoners resort to specific techniques known as *termination strategies*. Specifically, the Vadalog system employs the *isomorphism* termination strategy, which consists in suppressing isomorphic copies of previously generated facts (i.e., same name, same constants in same positions and bijection between labelled nulls), hence the chase steps starting from them are not performed and the descending facts are not generated. The correctness of such strategy is corroborated by the *reasoning boundedness* property of the fragment, which states that facts derived from isomorphic origins are isomorphic, thus uninformative for the reasoning task [3]. However, as a necessary condition to exploit this property, the set of Warded rules in the scenario is required to be in a *harmless* form, i.e., without joins between variables affected by existential quantification. Indeed, during the CHASE rules with such *harmful* joins could activate on labelled nulls. Therefore, suppressing isomorphic facts that carry these nulls could inhibit rule activation and affect reasoning correctness. On the other hand, avoiding the use of these joins affects the expressive power of the fragment. Consider the following example.

Example 1. *Company merger scenario modeled with a set Σ of Warded Datalog[±] rules.*

$$\text{Company}(x) \rightarrow \exists c \text{CEO}(x, c) \quad (\alpha)$$

$$\text{Merges}(x, y), \text{CEO}(x, c) \rightarrow \text{CEO}(y, c) \quad (\beta)$$

$$\text{Corp}(x, y) \rightarrow \exists c \text{CEO}(x, c), \text{CEO}(y, c) \quad (\gamma)$$

$$\text{CEO}(x, c), \text{CEO}(y, c) \rightarrow \text{Corp}(x, y) \quad (\rho)$$

For each company x there exists a CEO c (rule α). If x merges with a company y , c also becomes CEO of y (rule β). If x and y have a common CEO, they are in the same corporation (rule ρ) and vice versa (rule γ). Consider the database instance $D = \{\text{Company}(\text{Hsb}), \text{Company}(\text{Iba}), \text{Merges}(\text{Hsb}, \text{Iba})\}$ and the query Q : “what are all the entailed Corporations?” as ontological reasoning task.

It can be observed that the set of corporations is finite, whereas the CHASE does not terminate. By following the procedure without termination strategy, we first generate $\text{CEO}(\text{Hsb}, \nu_0)$ and $\text{CEO}(\text{Iba}, \nu_1)$ by activating α from the facts $\text{Company}(\text{Hsb})$ and $\text{Company}(\text{Iba})$, respectively. Next, we obtain $\text{CEO}(\text{Iba}, \nu_0)$ from β , $\text{Corp}(\text{Hsb}, \text{Iba})$ via the join on ν_0 in ρ , then $\text{CEO}(\text{Hsb}, \nu_2)$ and $\text{CEO}(\text{Iba}, \nu_2)$ by activating γ , β and so on. Due to the existential quantification in rule γ and its interplay with the recursions in rules β and ρ , an infinite set $\bigcup_{i=3, \dots} \{\text{CEO}(\text{Hsb}, \nu_i), \text{CEO}(\text{Iba}, \nu_i)\}$ is generated. On the other hand, employing the isomorphism termination strategy is not feasible either, due to the harmful join in rule ρ . Indeed, $\text{CEO}(\text{Iba}, \nu_0)$ is isomorphic with $\text{CEO}(\text{Iba}, \nu_1)$, yet its pruning would prevent the join in ρ with $\text{CEO}(\text{Hsb}, \nu_0)$ from generating $\text{Corp}(\text{Hsb}, \text{Iba})$ and the query in Example 1 from being answered correctly.

In this discussion paper, we illustrate how we enabled reasoning termination and decidability in practice over Warded Datalog[±] programs with harmful joins while preserving the expressive

power of the fragment. First, we introduce the **disarmament problem**, which consists in rewriting such programs into a version without harmful joins (namely, in the newly defined fragment *Harmless Warded Datalog[±]*) that is equivalent with respect to the CHASE. Then, we present **Harmful Join Elimination** (HJE) [12, 13], the first, to the best of our knowledge, rewriting technique to solve the disarmament problem. Finally, we discuss HJE integration into the Vadalog system and its **reasoning application** over the scenario in Example 1.

Related Work. The importance of achieving reasoning decidability in Datalog[±] fragments and handling the interplay between recursion and existential quantification acted as catalysts for the development of novel approaches for reasoning termination [14, 7, 6]. More in general, HJE belongs to the class of methodologies for Datalog rewriting. Among them, we mention its conversion into fragments such as *Guarded* [15], *Linear* [16], and *Disjunctive* [17]. Furthermore, by interpreting the rules as queries, distinct techniques were proposed to rewrite Datalog[±] programs with existentials [18] from *Regular Path Queries* [19] and *Description Logics* [20].

Overview. The remainder of this paper is organized as follows. In Section 2, we provide an overview of Warded Datalog[±]. In Section 3, we illustrate the HJE algorithm as well as its application over Example 1 in the Vadalog system. We draw our conclusions in Section 4.

2. Syntax and Semantics of Warded Datalog[±]

A Warded Datalog[±] program consists of a set of facts and rules. An *existential rule* is a first-order sentence $\forall \bar{x} \forall \bar{y} (\varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \psi(\bar{x}, \bar{z}))$, where φ (the *body*) and ψ (the *head*) are conjunctions of atoms, over the respective predicates, with constants and variables. For brevity, we may omit quantifiers and denote conjunction by comma. Let Σ be a set of rules and $p[i]$ a position (i.e., the i -th term of a predicate p with arity k , where $i = 1, \dots, k$). We define $p[i]$ as *affected* if (i) p appears in a rule in Σ with an existentially quantified variable (\exists -variable) in the i -th term or, (ii) there is a rule in Σ such that a universally quantified variable (\forall -variable) is only in affected body positions and in $p[i]$ in the head. A \forall -variable x is *harmful*, wrt a rule ρ in Σ , if x appears only in affected positions in ρ , otherwise it is *harmless*. In Example 1, α and γ are existential rules, while ρ is a *harmful join rule*, i.e., a rule such that a harmful variable is involved in a join (namely, *harmful join*). If the harmful variable is in *head*(ρ), it is *dangerous* [1].

Chase-based procedures enforce the satisfaction of Σ over a database D , incrementally expanding D into new instances I with facts derived from the application of the rules, until Σ is satisfied ($I = \text{chase}(D, \Sigma)$). Consider an instance $I' \supseteq I$ and a rule $\sigma : \varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \psi(\bar{x}, \bar{z}) \in \Sigma$. In the version of the CHASE we refer to (known as *oblivious* [5]), a *chase step* $\langle \sigma, h \rangle$ is *applicable* to I' if there exists a homomorphism h that maps the atoms of $\varphi(\bar{x}, \bar{y})$ to facts of I (i.e., $h(\varphi(\bar{x}, \bar{y})) \subseteq I$). When the chase step is applicable, the atom $h'(\psi(\bar{x}, \bar{z}))$ is added to I' , where h' is obtained by extending h so that $h'(z_i)$ is a fresh labelled null, $\forall z_i \in \bar{z}$. The *chase graph* $\mathcal{G}(D, \Sigma)$ is the directed graph with the facts from $\text{chase}(D, \Sigma)$ as nodes and an edge from a node n to a node m if m is obtained from n (and possibly other facts) via a chase step [7].

Given a pair $Q = (\Sigma, \text{Ans})$, where Ans is an n -ary predicate, the evaluation of Q over D is the set of tuples $Q(D, \Sigma) = \{\bar{t} \in \text{dom}(D)^n \mid \text{Ans}(\bar{t}) \in \text{chase}(D, \Sigma)\}$, where \bar{t} is a tuple of constants. A *reasoning task* consists in finding an instance J st: (i) $\bar{t} \in J$ iff $\text{Ans}(\bar{t}) \in Q(D, \Sigma)$; and (ii) for every other J' st $\bar{t} \in J'$ iff $\bar{t} \in Q(D, \Sigma)$, there is a homomorphism from J to J' [11].

3. Enabling Reasoning in Warded Datalog[±]

To achieve reasoning termination and decidability over recursive Warded Datalog[±] settings, especially in the presence of existential quantification, the oblivious chase is enriched with the isomorphism termination strategy, which acts as a *firing condition* that limits the activation of applicable chase steps [11]. With reference to the definitions in Section 2, such *isomorphic* chase variant causes an applicable step $\langle \sigma, h \rangle$ to activate, wrt $I' \supseteq I$, if there is no isomorphism of $h(\text{head}(\sigma))$ with I' . Therefore, given two isomorphic facts f and f' , only f is explored in the isomorphic chase, whereas the descending portions of the chase graph rooted in f' are pruned because isomorphic to the ones rooted in f , thus uninformative for the reasoning task.

Harmless Warded Datalog[±]. However, as we have introduced, in order to exploit such reasoning boundedness property and sustain decidability of the task while preserving correctness, Warded programs with recursion and existentials must not contain harmful join rules [3, Theorem 2], that is, they belong to *Harmless Warded Datalog[±]* [12]. Without loss of generality (as more complex joins can be broken into multiple steps [11]), a harmful join rule is a warded rule $\rho : A(x_1, y_1, h), B(x_2, y_2, h) \rightarrow \exists z C(\bar{x}, z)$, where A, B and C are atoms, $A[3]$ and $B[3]$ are affected positions, $x_1, x_2 \subseteq \bar{x}$, $y_1, y_2 \subseteq \bar{y}$ are disjoint tuples of harmless variables or constants and h is a harmful variable. A set of rules \in Harmless Warded Datalog[±] if: (1) *it is warded, i.e., all the dangerous variables in its rules appear in a single body atom (the ward), which only shares harmless variables with the rest of the body;* and (2) *it does not contain harmful join rules.*

Disarming the Warded Fragment. While in presence of programs without harmful joins the isomorphic chase can be employed without affecting the correctness nor the computational properties of the reasoning, restricting the Warded fragment to its Harmless Warded counterpart affects the expressive power available to reasoners such as the Vadalog system implementing it. With the goal of preventing such syntactic limitations, we developed a technique to solve the *disarmament problem*, that is, to rewrite a set Σ of Warded Datalog[±] rules in input to the reasoners into an equivalent set Σ' of Harmless Warded rules. In this context, two sets of rules are considered equivalent if they have the same *meaning* with respect to the CHASE [16], i.e., $\text{chase}(D, \Sigma) = \text{chase}(D, \Sigma')$ modulo fact isomorphism for each database D . Such technique, which we named *Harmful Join Elimination* (HJE), also allowed us to operationally prove that for each Warded set there exists an equivalent Harmless Warded one [13].

Intuitively, the HJE algorithm replaces each harmful join rule ρ with a set of harmless rules that cover the generation of all the facts derived from activating ρ , thus preserving correctness. To achieve this, it operates by considering the rules involved in the propagation of the *affectedness* (i.e., the labelled nulls in the CHASE) from the existentials to the harmful join variables in ρ .

Definition 1 (Causes of Affectedness). *Let $\rho \in \Sigma$ be a harmful join rule and let $H \in \{A, B\}$ be an atom in the body of ρ . We define causes of affectedness as the sequences of rules $\Gamma_{H_i} = [\sigma_s, \dots, \sigma_1]$ ($s < |\Sigma|, i \geq 1$) $\in \Sigma$, where: (i) $\sigma_1: H_1(x, y_1), R_1 \rightarrow \exists h H_2(x, y_2, h)$ is a direct cause, i.e., an existential rule that causes a position to be affected; and (ii) $\sigma_k: H_k(x, y_k, h), R_k \rightarrow H_{k+1}(x, y_{k+1}, h)$, $1 < k \leq s$, are indirect causes, i.e., rules that propagate the affectedness from σ_1 to ρ , such that $H_{s+1} = H$. H_1, \dots, H_s are atoms, R_1, \dots, R_s are atoms or conjunctions of atoms not containing h (as the rules are Warded). Let $X_{ij} = \Gamma_{A_i} \cap \Gamma_{B_j}$ be the concatenation of the sequences Γ_{A_i} and Γ_{B_j} with the same direct cause: the causes in X_{ij} are labelled after the sequence they belong to.*

With reference to Example 1, the rules α and γ are direct causes of affectedness, whereas β is an indirect one. The sequences of causes for the atom CEO_k ($k \in \{1,2\}$, in order of appearance in ρ) are: $\Gamma_{CEO_k 1} = [\alpha]$, $\Gamma_{CEO_k 2} = [\beta, \alpha]$, $\Gamma_{CEO_k 3} = [\gamma]$, $\Gamma_{CEO_k 4} = [\beta, \gamma]$. For instance, $X_{21} = [\beta_{\Gamma_{CEO_1 2}}, \alpha_{\Gamma_{CEO_1 2}}, \alpha_{\Gamma_{CEO_2 1}}]$ is a concatenation, as $\Gamma_{CEO_1 2}, \Gamma_{CEO_2 1}$ contain the same direct cause α .

To determine how the propagated nulls activating the harmful join in ρ affect the meaning in the CHASE, and to consequently build proper harmless rules, we compose ρ along all its concatenations X_{ij} of causes of affectedness via the *unfolding* and *folding* operations [16].

Definition 2 (Unfolding and Folding). Let ρ be a rule $A, B \rightarrow C$, where A and C are atoms and B is an atom or a conjunction of atoms, and let σ be a rule $R \rightarrow A'$, where A' is an atom and R an atom or a conjunction of atoms. Let A' be unifiable with A by substitution θ . The result of unfolding ρ at A with σ is the rule $\tau: (B, R \rightarrow C)\theta$. If the head of σ contains an \exists -variable h , it replaces h with a Skolem atom $f_{h\sigma}$ in τ , where f is an injective, deterministic and range disjoint function that calculates the values for \exists -variables, to control the identity of labelled nulls. Now, let σ' be a rule $B' \rightarrow R$, where R is an atom and B' is an atom or a conjunction of atoms. Let B' be unifiable with B by substitution θ' . The result of folding ρ into σ' is the rule $\nu: (A, R \rightarrow C)\theta'$.

The results of the compositions for each harmful join rule are represented via the *harmful unfolding tree* (hu-tree). Apart from the more technical side [12], the hu-tree T for $\langle \Sigma, \rho \rangle$ can be defined as a rule-labelled tree-like structure where: (i) the root is labelled by ρ ; (ii) for each X_{ij} , there exists a root-to-leaf path T_{ij} whose nodes are labelled by the result of unfolding their parent nodes with the causes in X_{ij} (in order of appearance); (iii) for each $\sigma_k \in X_{ij}$ involved in a recursion, there exists a leaf labelled by result of unfolding its parent node μ with σ_k and then folding it into μ . By definition of unfolding and folding, the leaves in T are harmless rules that cover the generation of all the facts derived in the CHASE from activating ρ on labelled nulls [13].

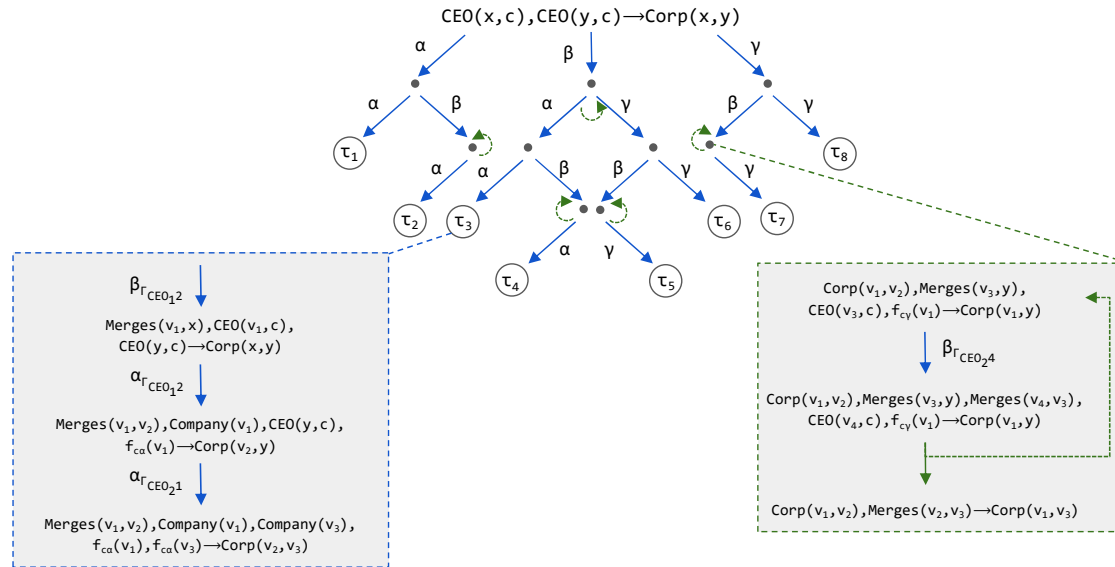


Figure 1: Hu-tree T for $\langle \Sigma, \rho \rangle$ of Example 1, with unfolding (blue) and folding (green) compositions.

Figure 1 shows the hu-tree T of $\langle \Sigma, \rho \rangle$ in Example 1. Specifically, the figure in the middle depicts the sequences of unfoldings (blue arrows) and foldings (green arrows) with the causes (here not labelled), from ρ to the leaves with the resulting harmless rules; the left figure illustrates more in detail the root-to-leaf path T_{21} obtained by unfolding the causes in X_{21} ; the right figure provides an application of folding over the recursive cause $\beta_{\Gamma_{CEO_2^4}}$. Note that T does not contain cycles, that is, only the plain arrows (not the dotted ones) in the figure are edges of the hu-tree.

HJE Algorithm. Given a Warded Datalog[±] set Σ with harmful join rules ρ , the HJE algorithm produces a new equivalent set $\Sigma' = \text{HJE}(\Sigma)$. It can be divided into two main phases. We refer to the extended versions of the work [12, 13] for an in-depth discussion of the algorithm.

In the *back-composition* phase (Algorithm 1), HJE first derives for each ρ all the causes of affectedness and the concatenations X_{ij} (line 5). Then, it iteratively builds the hu-tree T via unfolding (line 13) and, in case of recursive causes, folding (line 17) along the causes in each X_{ij} , in order of appearance, until all the root-to-leaf paths have been created: it employs the *maximum distance from harmless* (mdh), a value that corresponds to the highest cardinality among the X_{ij} , i.e., the height of T . The rules labelling the resulting leaves are then subject to an overall *cleanup* and *deduplication* (line 19). Rules that never activate are dropped. Also, if the functions of the Skolem atoms in a rule (derived from unfolding a direct cause in T) respect injectivity and range disjointness, they are simplified, otherwise the rule is dropped.

The *grounding* phase occurs if Σ contains rules π that are not causes of affectedness and that propagate ground values from the database to the harmful join in ρ . In this case, to cover the activation of ρ on such values, the $\text{Dom}(h)$ [3] rules are employed (below for A , corresponding ones for B), which force the harmful join variables to bind only to constants in the domain.

$$\text{Dom}(h), A(x, y, h) \rightarrow A'(x, y, h) \quad (\delta_1)$$

$$A'(x, y, h) \rightarrow A(x, y, h) \quad (\delta_2)$$

$$A'(x_1, y_1, h), B(x_2, y_2, h) \rightarrow \exists z C(\bar{x}, z) \quad (\delta_3)$$

Algorithm 1 Back-Composition in Harmful Join Elimination.

```

1: function BACK-COMPOSITION( $\Sigma$ )
2:    $P \leftarrow$  all harmful join rules in  $\Sigma$ ;  $\Sigma' = \Sigma$ 
3:   for  $\rho$  in  $P$  do
4:      $T \leftarrow$  empty hu-tree with  $\rho$  as root
5:      $\mathcal{X}_\rho \leftarrow$  all  $X_{ij}$  from  $\Gamma_{A_i}, \Gamma_{B_j}$  for  $\rho$  in  $\Sigma$  ▷ all sets of causes of affectedness for  $\rho$ 
6:      $Q \leftarrow$  queue with  $\rho$  enqueued
7:      $\text{mdh} = \max(|X_{ij}|)$  with  $X_{ij}$  in  $\mathcal{X}_\rho$ 
8:     for  $\text{dh} \leftarrow 1$  to  $\text{mdh}$  do ▷ build hu-tree  $T$  for  $\rho$ 
9:        $\mu = Q.\text{dequeue}()$  ▷ next node to compose back
10:      for  $X_{ij}$  in  $\mathcal{X}_\rho$  do
11:        if  $\text{dh} \leq |X_{ij}|$  then
12:           $\sigma = X_{ij}.\text{next}()$  ▷ next cause, from indirect to direct, in order of appearance in  $X_{ij}$ 
13:           $\nu = \text{unfold}(\mu, \sigma)$  ▷ unfold current node with current cause
14:           $Q.\text{enqueue}(\nu)$ 
15:           $T_{ij} \leftarrow T_{ij} \cup \{\nu\}$  ▷ update path  $T_{ij}$  at depth  $\text{dh}$  with unfolding node  $\nu$ 
16:          if  $\text{isRecursive}(\sigma, X_{ij})$  then
17:             $T \leftarrow T \cup \{\text{fold}(\text{unfold}(\nu, \sigma), \mu)\}$  ▷ update  $T$  with folding node for recursive cause
18:       $\Sigma'.\text{add}(T.\text{leaves}())$ 
19:    $\text{skolem-cleanup}(\Sigma')$  ▷ simplify Skolem atoms and deduplicate rules in  $\Sigma'$ 
return  $\Sigma'$ 

```

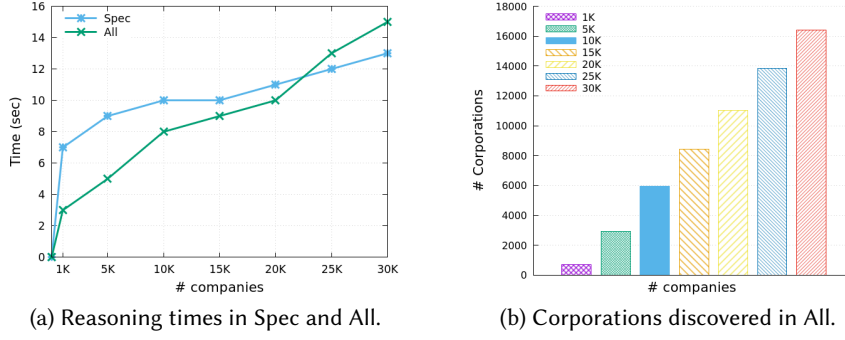


Figure 2: Results of the experiments for Spec and All tasks, to vary the number of companies.

HJE in Vadalog System. HJE is integrated into the *logic optimizer* of the Vadalog system, the component responsible for applying the required rewriting steps to the program before the reasoner processes it [11]. This enables reasoning termination and decidability over recursive Warded settings with harmful joins without affecting the expressive power of the Warded fragment. Thus, we are now able to reason over the scenario in Example 1. We run the experiments on a local installation of the Vadalog system, with a MacBook Pro i7 with 2.5 GHz and 16 GB of RAM. The rules listed below are added via HJE to Σ' (some are merged for space reasons): τ_3 labels the leaf of T_{21} and v_3 labels the folded node over $\beta_{\Gamma_{CEO_{24}}}$ from Figure 1, after skolem cleanup. We adjusted input data from the DBpedia [3] company KG, adopting datasets of increasing complexity, with 1K, 5K, 10K, 15K, 20K, 25K and 30K companies. We built two reasoning tasks: (i) *Spec*, to obtain all the corporations of the company *CBS*, and (ii) *All*, to obtain all the corporations, to vary the number of companies. Figure 2(a) illustrates the reasoning times, all under 16 seconds, which proves the very good performance of the Vadalog system. *Spec* requires more time than *All* for smaller datasets; when the number of companies increases, *All* tends to a steeper curve. Figure 2(b) shows the number of corporations discovered in *All*.

$$\begin{aligned}
 &Company(x) \rightarrow Corp(x, x) & (\tau_1) \\
 &Company(x), Merges(x, y) \rightarrow Corp(x, y), Corp(y, x) & (\tau_{2,3}) \\
 &Company(x), Merges(x, y), Merges(x, z) \rightarrow Corp(y, z) & (\tau_4) \\
 &Corp(x, y), Merges(x, w), Merges(x, z) \rightarrow Corp(w, z) & (\tau_5) \\
 &Corp(x, y), Merges(x, z) \rightarrow Corp(x, z), Corp(z, x) & (\tau_{6,7}) \\
 &Corp(x, y) \rightarrow Corp(x, x) & (\tau_8) \\
 &Corp(x, y), Merges(x, z) \rightarrow Corp(y, z), Corp(z, y) & (v_{1,2}) \\
 &Corp(x, y), Merges(y, z) \rightarrow Corp(x, z), Corp(z, x) & (v_{3,4})
 \end{aligned}$$

4. Conclusion

To achieve reasoning termination and decidability over Warded Datalog[±] in practice, the settings must not contain harmful joins. In this work, we discussed the disarmament problem of rewriting such joins into an equivalent Harmless Warded form that upholds the expressiveness of the fragment and the correctness of the task. We then presented the Harmful Join Elimination, the disarmament algorithm integrated into the Warded Datalog[±]-based reasoner Vadalog system.

References

- [1] G. Gottlob, A. Pieris, Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue, in: IJCAI, 2015.
- [2] M. Krötzsch, V. Thost, Ontologies for knowledge graphs: Breaking the rules, in: International Semantic Web Conference (1), volume 9981 of *LNCS*, 2016, pp. 376–392.
- [3] L. Bellomarini, E. Sallinger, G. Gottlob, The Vadalog System: Datalog-based reasoning for knowledge graphs, *VLDB 11* (2018).
- [4] P. Barceló, R. Pichler, Datalog in Academia and Industry: Second International Workshop, Datalog 2.0, Vienna, Austria, September 11-13, Proceedings, volume 7494, Springer, 2012.
- [5] A. Cali, G. Gottlob, M. Kifer, Taming the infinite chase: Query answering under expressive relational constraints, *Journal of Artificial Intelligence Research* 48 (2013) 115–174.
- [6] A. Cali, G. Gottlob, T. Lukasiewicz, B. Marnette, A. Pieris, Datalog+/-: A family of logical knowledge representation and query languages for new applications, in: 2010 25th Annual IEEE LICS, IEEE, 2010, pp. 228–242.
- [7] A. Cali, G. Gottlob, T. Lukasiewicz, A general datalog-based framework for tractable query answering over ontologies, *Journal of Web Semantics* 14 (2012) 57–83.
- [8] G. Gottlob, A. Pieris, E. Sallinger, Vadalog: recent advances and applications, in: JELIA, Springer, 2019, pp. 21–37.
- [9] L. Bellomarini, G. Gottlob, A. Pieris, E. Sallinger, Swift logic for big data and knowledge graphs, in: IJCAI, Springer, 2017, pp. 2–10.
- [10] D. Maier, A. O. Mendelzon, Y. Sagiv, Testing implications of data dependencies, *ACM Transactions on Database Systems* 4 (1979) 455–468.
- [11] L. Bellomarini, D. Benedetto, G. Gottlob, E. Sallinger, Vadalog: A modern architecture for automated reasoning with large knowledge graphs, *Information Systems* (2020) 101528.
- [12] T. Baldazzi, L. Bellomarini, E. Sallinger, P. Atzeni, Eliminating harmful joins in warded datalog+/-, in: International Joint Conference on Rules and Reasoning, Springer, 2021, pp. 267–275.
- [13] T. Baldazzi, L. Bellomarini, E. Sallinger, P. Atzeni, iwarded: A system for benchmarking datalog+/-reasoning (technical report), arXiv preprint arXiv:2103.08588 (2021).
- [14] G. Berger, G. Gottlob, A. Pieris, E. Sallinger, The space-efficient core of Vadalog, in: PODS, 2019, pp. 270–284.
- [15] G. Gottlob, S. Rudolph, M. Simkus, Expressiveness of guarded existential rule languages, in: PODS, 2014, pp. 27–38.
- [16] F. Afrati, M. Gergatsoulis, F. Toni, Linearisability on datalog programs, *Theoretical Computer Science* 308 (2003) 199–226.
- [17] M. Kaminski, Y. Nenov, B. C. Grau, Datalog rewritability of disjunctive datalog programs and non-Horn ontologies, *Artificial Intelligence* 236 (2016) 90–118.
- [18] Z. Wang, P. Xiao, K. Wang, Z. Zhuang, H. Wan, Query answering for existential rules via efficient datalog rewriting, in: IJCAI, 2020, pp. 1933–1939.
- [19] N. Francis, L. Segoufin, C. Sirangelo, Datalog rewritings of regular path queries using views, arXiv preprint arXiv:1511.00938 (2015).
- [20] S. Ahmetaj, M. Ortiz, M. Simkus, Polynomial datalog rewritings for expressive description logics with closed predicates, in: IJCAI, 2016, pp. 878–885.