Reasoning in Financial Settings with Harmful Joins

Teodoro Baldazzi[1], Paolo Atzeni[1]

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

Abstract

Warded Datalog+/- has recently emerged as a powerful logic language for ontological reasoning on large knowledge graphs, offering a very good trade-off between expressive power and data complexity. Achieving decidability and data tractability in practice, over complex recursive settings with existential quantification, requires reasoners to adopt specialized strategies that control the effects of recursion and ensure reasoning termination with small memory footprint. However, to enable these strategies, the Warded Datalog+/- settings must be in a "harmless" form, i.e., without joins on variables affected by existential quantification. We provide an algorithm to remove such "harmful" joins, supporting reasoning decidability and the full expressive power of the language while preserving the correctness of the task. The algorithm is integrated into the Vadalog system, a state-of-the-art Warded Datalog+/- -based reasoner. We employ it to solve the Strong Link problem, a relevant financial task to find possible links between pairs of companies, based on the existence of a person who owns a significant share of both their stocks. Solving this problem allows to investigate possibly malevolent shareholdings and ownerships.

Keywords

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

1. Introduction

Recent years have witnessed a growing interest, among companies of distinct size and scope, towards building and exploiting private corporate knowledge in the form of financial *Knowledge Graphs* (KG). This led to the rising adoption of intelligent systems that allow to manage such extensional knowledge and enrich it with new information, inferred via efficient *ontological reasoning* mechanisms and modeled by logical rules and ontologies in specific formalisms [1, 2]. Employing modern languages for *Knowledge Representation and Reasoning* (KRR) [3] enabled companies to solve relevant problems and tasks in distinct business domains, such as *investment analysis*, *company ownership*, *shock propagation*, *fraud detection*, *anti-money laundering*, etc.

A Formalism for Ontological Reasoning. As main requirements, KRR languages must exhibit full support for recursion and joins as well as existential quantification, all aspects essential to guarantee the expressive power needed for ontological reasoning and KG traversal. At the same time, decidability and tractability of the reasoning must be sustained, basically limiting the data complexity to a polynomial degree [4]. Warded Datalog$^{\pm}$ [2] is a member (technically, a *fragment*) of the Datalog$^{\pm}$ family [5] that recently rose among the logic languages for ontological reasoning on KGs. It covers these requirements, offering a very good trade-off between expressive power and computational complexity and capturing PTIME data complexity for the reasoning. The semantics of a set Σ of Warded rules over a database instance D is defined via the CHASE procedure [6]. Intuitively, it adds new facts to D, possibly containing freshly generated symbols ν (technically, *labelled nulls*) [7] that act as placeholders for the existentially quantified variables, until Σ is satisfied. Warded Datalog$^{\pm}$ is implemented in the *Vadalog system* [3], a state-of-the-art reasoner that allows to perform ontological reasoning in complex scenarios.

Critical Aspects in Reasoning. When reasoning on Datalog$^{\pm}$ settings, in the presence of recursion and existential quantification, infinite labelled nulls could be generated in the CHASE, causing the procedure not to terminate and inhibiting the decidability of the task [8]. In this work, we tackle such issue in the context of Warded Datalog$^{\pm}$ scenarios. Indeed, while the computational properties of the Warded fragment bode well for efficient implementations, it is still required to apply specific techniques (namely, *termination strategies*) that properly control the interactions between recursion and existentials in the CHASE. Consider the following example.

Example 1. *Scenario modeled with a Warded set* Σ.

$$Bank(x) \rightarrow \exists d\ Manager(x, d) \quad (\alpha)$$
$$Acquires(x, y), Manager(x, d) \rightarrow Manager(y, d) \quad (\beta)$$
$$BankGroup(x, y) \rightarrow \exists d\ Manager(x, d) \quad (\gamma)$$
$$Manager(x, d), Manager(y, d) \rightarrow BankGroup(x, y) \quad (\rho)$$

Σ represents a bank acquisition scenario. For each bank x there exists a manager d (rule α). If x acquires a bank y, d also becomes manager of y (rule β). If x and y have a common manager, they are in the same bank group (rule ρ) and vice versa (rule γ).

Published in the Workshop Proceedings of the EDBT/ICDT 2022 Joint Conference (March 29-April 1, 2022), Edinburgh, UK
teodoro.baldazzi@uniroma3.it (T. Baldazzi); paolo.atzeni@uniroma3.it (P. Atzeni)

Consider the database instance D = {*Bank*(*Unicredit*), *Bank*(*MPS*), *Acquires*(*Unicredit*, *MPS*)} and the query Q: "*what are all the BankGroups?*" as ontological reasoning task. It can be observed that the set of such bank groups is finite, whereas the CHASE does not terminate. First, we generate *Manager*(*Unicredit*,ν_0) and *Manager*(*MPS*,ν_1) by activating α from the facts *Bank*(*Unicredit*) and *Bank*(*MPS*), respectively. Then, we obtain *Manager*(*MPS*,ν_0) via β, *BankGroup*(*Unicredit*,*MPS*) via the join on ν_0 in ρ, *Manager*(*Unicredit*,ν_2), *Manager*(*MPS*,ν_2) by activating γ and so on. Indeed, the recursion involving β, γ and ρ causes the generation of an infinite set $\bigcup_{i=3,\ldots}\{Manager(Unicredit, \nu_i), Manager(MPS, \nu_i)\}$.

Enabling Reasoning Termination in Vadalog. To achieve reasoning termination in practice, while upholding the correctness of the task, the Vadalog system applies the *isomorphism termination strategy* in the CHASE: isomorphic copies of previously generated facts (i.e., same name, same constants in same position and bijection between labelled nulls) are not explored, i.e., the chase steps starting from them are not performed and the derived facts are not generated. This strategy exploits the theoretical underpinning of Warded Datalog$^{\pm}$ known as *reasoning boundedness*, which states that facts derived from isomorphic origins would be in turn isomorphic, thus uninformative for query answering [9].

However, as a necessary condition for such exploitation, the set of rules is required to be in a "harmless" form, i.e., without a "harmful" type of joins between variables affected by existential quantification (namely, in *Harmless* Warded Datalog$^{\pm}$). This is due to the fact that rules with these *harmful joins* could activate on labelled nulls, propagated from the existentials: therefore, the suppression of isomorphic facts carrying such nulls could hamper their activation and, consequently, undermine the correctness of the reasoning task. For instance, in Example 1 *Manager*(*MPS*,ν_0) is isomorphic to *Manager*(*MPS*,ν_1), yet its suppression would prevent the join in ρ with *Manager*(*Unicredit*,ν_0) from generating *BankGroup*(*Unicredit*,*MPS*) and the query from being answered correctly. On the other hand, it is intuitive to observe that precluding the use of harmful joins when modeling reasoning settings would affect the expressive power of the adopted KRR language.

In this work, we investigate the role of harmful joins in Warded Datalog$^{\pm}$ settings and we enable reasoning termination in their presence by applying the *Harmful Join Elimination* (HJE) algorithm [10], a technique to rewrite a set of Warded rules with harmful joins into an equivalent Harmless Warded form.

A Financial Use Case. As a powerful Warded Datalog$^{\pm}$-based reasoner, the Vadalog system is employed to efficiently solve relevant tasks in real-world scenarios, mainly related to the financial and business realm [11]. The applicability of the isomorphism termination strategy, to achieve reasoning decidability while preserving correctness of query answering, is essential in such complex domains. Therefore, reasoning settings that feature harmful joins and recursion require to be treated properly. Such is the case of the *Strong Link* problem [9], a high-interest scenario in the context of company ownership. It consists in determining possible links between pairs of companies, based on the existence of a person who owns a significant share of both their stocks. Solving this task allows to investigate and monitor (possibly malevolent) company shareholdings and ownerships.

Motivated by this, we apply our HJE algorithm to the Warded Datalog$^{\pm}$ scenario of Strong Link, rewriting it into its Harmless Warded equivalent and enabling the Vadalog system to solve the task in an efficient fashion, while preserving correctness.

In detail, the main **contributions** of this paper are:

- An integration of the **theoretical bases** for the Warded fragment. We introduce Harmless Warded Datalog$^{\pm}$ and we discuss the role of harmful joins in Warded settings, with reference to expressive power and reasoning termination. We define and solve the *disarmament problem*, which consists in rewriting a set of Warded rules with harmful joins into a Harmless Warded version, equivalent with respect to the CHASE.
- The Harmful Join Elimination algorithm, our **rewriting technique** that exploits such bases to solve the disarmament problem and enable Vadalog's termination strategy in practice on recursive reasoning settings with harmful joins. We illustrate the procedure and we apply it to the program in Example 1.
- A **real-world application** of the HJE algorithm on the Strong Link problem. We discuss the financial task and rewrite the corresponding setting with harmful joins into its Harmless Warded equivalent. We provide an experimental evaluation of the now enabled reasoning via the Vadalog system, extracting input data from the open KG provided by DBpedia [12].

Related Work. Harmful Join Elimination belongs to the class of methodologies for Datalog rewriting. Among them, we mention its conversion into specific fragments, such as *Guarded* [13], *Linear* [14], and *Disjunctive* [15]. Similarly, by interpreting the rules as queries, several methods have been devised to rewrite Datalog$^{\pm}$ programs with existential rules [16, 17] and from distinct formalisms, such as *Regular Path Queries* [18] and *Description Logics* [19, 20]. The importance of achieving decidability of Datalog$^{\pm}$ reasoning settings, in the presence of recursion and existential quantification, determined the research and the development of novel approaches to sustain termination of reasoning tasks [21, 5, 7]. Yet, the HJE algorithm is, to the best of our knowledge, the first technique to rewrite Warded Datalog$^\pm$ rules into an equivalent Harmless Warded version, that allows to achieve reasoning termination and decidability.

Overview. This paper is organized as follows. In Section 2, we recall relevant background notions. In Section 3, we discuss the disarmament problem and we illustrate the HJE algorithm in action. In Section 4, we provide the application of HJE on the Strong Link problem and we present the experimental evaluation. We draw our conclusions in Section 5.

2. Reasoning with Vadalog

In this section, we briefly recall some relevant notions to guide our discussion. Let C, N, and V be disjoint countably infinite sets of *constants*, (*labelled*) *nulls* and (*regular*) *variables*, respectively. A (*relational*) *schema* $\mathbf{S}$ is a finite set of relation symbols (or *predicates*) with associated arity. A *term* is a either a constant or variable. An *atom* over $\mathbf{S}$ is an expression of the form $R(\bar{v})$, where $R \in \mathbf{S}$ is of arity $n > 0$ and $\bar{v}$ is an n-tuple of terms. A *database* (*instance*) over $\mathbf{S}$ associates to each relation symbol in $\mathbf{S}$ a relation of the respective arity over the domain of constants and nulls. The members of the relations are called *tuples* or *facts* [9].

Existentials and Affectedness. The Vadalog system employs VADALOG, a KRR language that implements Warded Datalog$^\pm$ as its logical core. A Warded Datalog$^\pm$ 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 $i = 1, \ldots, k$). We define $p[i]$ as *affected* if (i) p appears in a rule in Σ with the i-th term that contains an existentially quantified variable or, (ii) there is a rule ρ of Σ such that a universally quantified variable is only in affected body positions and in position $p[i]$ in the head of ρ. A variable x is *harmful*, with respect to a rule, if x appears only in affected positions, otherwise it is *harmless*. A rule that contains a harmful variable is a *harmful rule*. In Example 1, α and γ are existential rules, β propagates the affected position *Manager*[2] and 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 appears in the head of the rule, it is *dangerous*. A rule with a dangerous variable is a *dangerous rule*.

Chase and Reasoning. Let D be a database and Σ a set of rules. The CHASE procedure is a fundamental algorithmic tool that enforces the satisfaction of Σ by expanding D into a new instance *chase*(D, Σ) with facts generated from the application of the rules in Σ over D [6]. We denote *chase graph* $\mathcal{G}(D, \Sigma)$ as the directed graph with the facts from *chase*(D, Σ) 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*, i.e., a rule in Σ [5]. Given a pair $Q = (\Sigma, \mathit{Ans})$, where *Ans* is an n-ary predicate, we define the evaluation of Q over D as the set of tuples $Q(D, \Sigma) = \{\bar{t} \in \mathit{dom}(D)^n \mid \mathit{Ans}(\bar{t}) \in \mathit{chase}(D, \Sigma)\}$, where $\bar{t}$ is a tuple of constants. We denote *reasoning task* as the task of finding a database instance J such that: (i) $\bar{t} \in J$ iff $\mathit{Ans}(\bar{t}) \in Q(D, \Sigma)$ and (ii) for every other instance J' such that $\bar{t} \in J'$ iff $\bar{t} \in Q(D, \Sigma)$, there is a homomorphism from J to J' [3].

3. Harmful Join Elimination

In this section, we present our rewriting technique and we apply it to Example 1. As the goal of this paper is to enable reasoning with the Vadalog system on a real-world scenario with harmful joins, we refer to the extended versions of the work [10, 22] for an in-depth discussion of the algorithm and the theory behind.

Without loss of generality (as more complex joins can be broken into multiple steps [3]), we define a *harmful join rule* as a rule of the form:

$$A(x_1, y_1, h), B(x_2, y_2, h) \rightarrow \exists z\; C(\bar{x}, z) \qquad (\rho)$$

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, h is a harmful variable.

Harmless Warded Datalog$^\pm$. As we have introduced, in order to exploit the reasoning boundedness property, Warded programs with recursion and existentials must not contain harmful joins, that is, they belong to the Harmless Warded fragment. A set of rules $\in$ *Harmless* Warded Datalog$^\pm$ if the following conditions hold: *(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.* Indeed, while the CHASE procedure remains potentially infinite in Harmless Warded settings due to the generation of infinite labelled nulls, the absence of joins activating on such nulls renders their identity irrelevant in the evaluation [9].

The Disarmament Problem. In presence of Harmless Warded programs, the isomorphism termination strategy can thus be applied without affecting the correctness of the reasoning task: given two isomorphic facts t and t', only t is explored in the CHASE, whereas the descending portions of the chase graph rooted in t' are pruned. However, restricting VADALOG to such fragment, while it preserves the computational qualities of Warded Datalog$^\pm$, limits the expressiveness of the KRR language to joins between harmless variables. To avoid such restrictions, we define the *disarmament problem* for a set Σ of Warded Datalog$^{\pm}$ rules as the task of finding a set Σ' of Harmless Warded rules that is equivalent to Σ. In this context, two sets of rules are equivalent if they have the same *meaning* with respect to the CHASE [14], i.e., *chase*$(D, \Sigma) =$ *chase*(D, Σ') modulo fact isomorphism for each database D. It can be proved that the disarmament problem is always solvable, that is, for each Warded set Σ there is an equivalent Harmless Warded set Σ' [22].

Disarming Warded Datalog$^{\pm}$. Let Σ be a Warded set with one or more harmful join rules. With the goal of enabling the isomorphism termination strategy, while preserving VADALOG's expressive power, we propose a technique to rewrite Σ into an equivalent Harmless form, i.e., to solve the disarmament problem.

We first provide some theoretical bases. By definition of harmful variables, we observe that Σ always contains one or more rules that propagate the *affectedness* of such variables (i.e., the nulls in the CHASE), from the existentials to the harmful join. We define them as follows.

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 $\Gamma_{Hi} = [\sigma_s, \ldots, \sigma_1]$ ($s < |\Sigma|$, $i \geq 1$) $\in \Sigma$, where: (i) σ_1: $H_1(x, y), R_1 \to \exists h\, H_2(x, y, h)$ is a* direct *cause, i.e., an existential rule that causes a position to be affected; and (ii) σ_k: $H_k(x, y, h), R_k \to H_{k+1}(x, y, h)$, $1 < k \leq s$, are* indirect *causes, i.e., rules that propagate the affectedness from σ_1 to ρ, such that $H_{s+1} = H$. $H_1, \ldots, H_s$ are atoms, $R_1, \ldots, R_s$ are (conjunctions of) atoms not containing h (as the rules are Warded).*

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 $Manager$ Man_1 (resp., Man_2 in order of appearance in ρ) are: $\Gamma_{Man_1 1} = [\alpha]$, $\Gamma_{Man_1 2} = [\beta, \alpha]$, $\Gamma_{Man_1 3} = [\gamma]$, $\Gamma_{Man_1 4} = [\beta, \gamma]$. Let $X_{ij} = \Gamma_{Ai} {}^\frown \Gamma_{Bj}$ be the concatenation of the sequences Γ_{Ai} and Γ_{Bj} for the atoms A, B in ρ: the causes in X_{ij} are labelled after the sequence they belong to. In our example $X_{21} = [\beta_{\Gamma_{Man_1 2}}, \alpha_{\Gamma_{Man_1 2}}, \alpha_{\Gamma_{Man_2 1}}]$. Intuitively, our goal is to replace each ρ with harmless rules that cover the generation of all the facts derived in the CHASE from the activation of ρ. To learn how the propagated nulls that activate the harmful join affect the meaning, and to consequently build proper harmless rules, we compose ρ along all X_{ij}. Such composition is performed via the *unfolding* operation [14], defined as follows.

Definition 2 (Unfolding). *Let ρ be a rule $A, B \to C$, where A and C are atoms and B is an atom or a conjunction of atoms, and let σ be a rule $R \to A'$, 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 τ: $(B, R \to C)\theta$. If the head of σ contains an existentially quantified 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 existentially quantified variables, to control the identity of labelled nulls.*

Moreover, to cover the activation of the harmful join in ρ on nulls that are propagated from causes involved in a recursion, the following *folding* operation [14] is applied.

Definition 3 (Folding). *Let ρ be a rule $A, B \to C$, where C is an atom and A and B are (conjunctions of) atoms, and let σ be a rule $A' \to R$, where R is an atom and A' is an atom or a conjunction of atoms. Let A' be unifiable with A by substitution θ. The result of* folding *ρ into σ is the rule τ: $(B, R \to C)\theta$.*

To keep track of the composition for each harmful join rule, we employ the *harmful unfolding tree* (hu-tree). Apart from the more technical side [10], 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 node labelled by the result of folding its parent node into σ_k.* By definition of unfolding and folding, it can be proved that the leaves of T are harmless rules that cover the generation of all the facts derived in the CHASE from the activation of ρ on labelled nulls [22]. With reference to Example 1, Figure 1 shows the path T_{21}, built by unfolding ρ with the causes in X_{21} (atoms are renamed for space reasons).

$$\begin{tikzcd}
\texttt{Man(x,d),Man(y,d)}\to\texttt{BGroup(x,y)} \arrow[d, "\beta_{\Gamma_{Man_1 2}}"] \\
\texttt{Acq}(v_1,v_2)\texttt{,Man}(v_1\texttt{,d}), \texttt{Man(y,d)}\to\texttt{BGroup}(v_2\texttt{,y}) \arrow[d, "\alpha_{\Gamma_{Man_1 2}}"] \\
\texttt{Acq}(v_1,v_2)\texttt{,Bank}(v_1)\texttt{,Man(y,d)}, f_{d\alpha}(v_1)\to\texttt{BGroup}(v_2\texttt{,y}) \arrow[d, "\alpha_{\Gamma_{Man_2 1}}"] \\
\texttt{Acq}(v_1,v_2)\texttt{,Bank}(v_1)\texttt{,Bank}(v_3), f_{d\alpha}(v_1), f_{d\alpha}(v_3)\to\texttt{BGroup}(v_2,v_3)
\end{tikzcd}$$

Figure 1: Path T_{21} of hu-tree T for $\langle \Sigma, \rho \rangle$ of Example 1.

Harmful Join Elimination. By exploiting these theoretical bases, we now present our rewriting algorithm, which we named *Harmful Join Elimination.* Given a Warded set Σ with one or more harmful join rules ρ, $\text{HJE}(\Sigma) = \Sigma' \in$ Harmless Warded Datalog$^{\pm}$ such that *chase*(D, Σ) = *chase*(D, Σ') modulo fact isomorphism for each D. HJE can be divided into two main phases, which we describe by applying them to Example 1.

In the *back-composition* phase, first HJE identifies the sets X_{ij} of causes of affectedness for ρ. Then, it builds the hu-tree T by composing back, via unfolding and folding, along each X_{ij}. With reference to Example 1, Figure 2 shows some of the sequences of unfolded causes (here not labelled) and the resulting leaves.

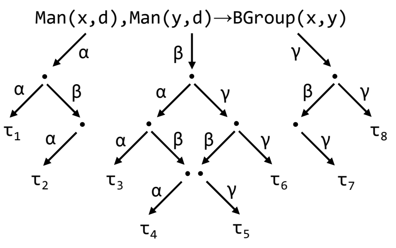


Figure 2: Unfolded causes in T for $\langle \Sigma, \rho \rangle$ of Example 1.

The rules labelling the resulting leaves are subject to an overall *cleanup* and *deduplication.* Rules that never activate are dropped. 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 unified and removed, otherwise the rule is dropped. With reference to our running example, the following rules are added to Σ'. Specifically, τs derive from unfolding, whereas υs derive from folding: τ_3 labels the leaf of T_{21} from Figure 1, after skolem unification during cleanup.

$$\mathit{Bank}(\mathrm{x}) \rightarrow \mathit{BGroup}(\mathrm{x}, \mathrm{x}) \quad (\tau_1)$$
$$\mathit{Bank}(\mathrm{x}), \mathit{Acq}(\mathrm{x}, \mathrm{y}) \rightarrow \mathit{BGroup}(\mathrm{x}, \mathrm{y}) \quad (\tau_2)$$
$$\mathit{Bank}(\mathrm{x}), \mathit{Acq}(\mathrm{x}, \mathrm{y}) \rightarrow \mathit{BGroup}(\mathrm{y}, \mathrm{x}) \quad (\tau_3)$$
$$\mathit{Bank}(\mathrm{x}), \mathit{Acq}(\mathrm{x}, \mathrm{y}), \mathit{Acq}(\mathrm{x}, \mathrm{z}) \rightarrow \mathit{BGroup}(\mathrm{y}, \mathrm{z}) \quad (\tau_4)$$
$$\mathit{BGroup}(\mathrm{x}, \mathrm{y}), \mathit{Acq}(\mathrm{x}, \mathrm{w}), \mathit{Acq}(\mathrm{x}, \mathrm{z}) \rightarrow \mathit{BGroup}(\mathrm{w}, \mathrm{z}) \quad (\tau_5)$$
$$\mathit{BGroup}(\mathrm{x}, \mathrm{y}), \mathit{Acq}(\mathrm{x}, \mathrm{z}) \rightarrow \mathit{BGroup}(\mathrm{x}, \mathrm{z}) \quad (\tau_6)$$
$$\mathit{BGroup}(\mathrm{x}, \mathrm{y}), \mathit{Acq}(\mathrm{x}, \mathrm{z}) \rightarrow \mathit{BGroup}(\mathrm{z}, \mathrm{x}) \quad (\tau_7)$$
$$\mathit{BGroup}(\mathrm{x}, \mathrm{y}) \rightarrow \mathit{BGroup}(\mathrm{x}, \mathrm{x}) \quad (\tau_8)$$
$$\mathit{BGroup}(\mathrm{x}, \mathrm{y}), \mathit{Acq}(\mathrm{x}, \mathrm{z}) \rightarrow \mathit{BGroup}(\mathrm{y}, \mathrm{z}) \quad (\upsilon_1)$$
$$\mathit{BGroup}(\mathrm{x}, \mathrm{y}), \mathit{Acq}(\mathrm{x}, \mathrm{z}) \rightarrow \mathit{BGroup}(\mathrm{z}, \mathrm{y}) \quad (\upsilon_2)$$
$$\mathit{BGroup}(\mathrm{x}, \mathrm{y}), \mathit{Acq}(\mathrm{y}, \mathrm{z}) \rightarrow \mathit{BGroup}(\mathrm{x}, \mathrm{z}) \quad (\upsilon_3)$$
$$\mathit{BGroup}(\mathrm{x}, \mathrm{y}), \mathit{Acq}(\mathrm{y}, \mathrm{z}) \rightarrow \mathit{BGroup}(\mathrm{z}, \mathrm{x}) \quad (\upsilon_4)$$

In the *grounding* phase, HJE adds new harmless rules to cover the activation of ρ on ground values, propagated from the database. It employs the Dom(h) [9] atom, to ensure that the harmful join variables bind only to constants in the domain, and an artificial atom H', where $H \in \{A, B\}$ of ρ. With reference to Example 1, the following rules (namely, *dom rules*) are added to Σ'.

$$\mathit{Dom}(\mathrm{d}), \mathit{Man}(\mathrm{x}, \mathrm{d}) \rightarrow \mathit{Man}'(\mathrm{x}, \mathrm{d}) \quad (\delta_1)$$
$$\mathit{Man}'(\mathrm{x}, \mathrm{d}) \rightarrow \mathit{Man}(\mathrm{x}, \mathrm{d}) \quad (\delta_2)$$
$$\mathit{Man}'(\mathrm{x}, \mathrm{d}), \mathit{Man}'(\mathrm{y}, \mathrm{d}) \rightarrow \mathit{BGroup}(\mathrm{x}, \mathrm{y}) \quad (\delta_3)$$

Additionally, to preserve the propagation of ground facts from rules $\pi \in \Sigma$ that are not causes of affectedness, if the head of π unifies with: (i) the atom H in ρ, a rule π' is added, which results from renaming H with H'; (ii) the atom H_k in the cause σ_k, a rule π' is added, which results from unfolding σ_k with π. Finally, ρ is removed from Σ' and the procedure terminates.

The resulting Σ' is a set of Harmless Warded rules. Its equivalence to the original set Σ can be easily derived as a generalization of proofs in the Datalog context [23]. The output facts of the reasoning task on Example 1, via isomorphism termination strategy, are: *BankGroup*(*Unicredit*,*Unicredit*), *BankGroup*(*MPS*,*MPS*), *BankGroup*(*Unicredit*,*MPS*) and *BankGroup*(*MPS*,*Unicredit*). HJE shows an exponential behaviour with respect to the number of causes, due to the worst-case generation of a distinct path in the hu-tree for each X_{ij}. Yet, such blowup is data independent and it does not affect reasoning performance [22].

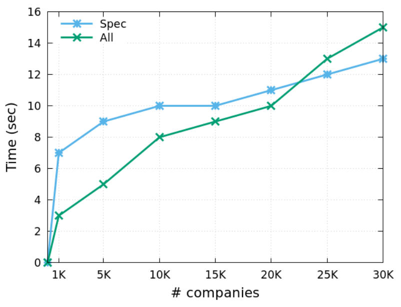
4. Financial Use Case

In this section, we provide a real-world application of the HJE algorithm. We employ it to enable reasoning with the Vadalog system on the Strong Link problem, which we then solve and empirically evaluate.

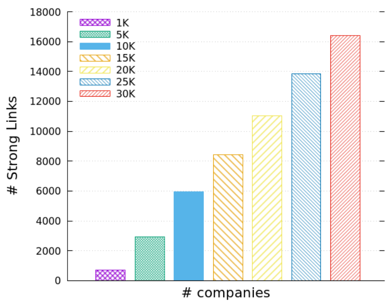
The Strong Link Problem. Being able to determine and monitor the connections between companies and shareholders, as well as to investigate possible felonious activities in company ownership scenarios, is of high-interest in the financial and corporate realm. With such goal in mind, we consider the Strong Link problem over "significantly controlled companies", that is, companies for which there exist "significant shareholders" who own more than 20% of their stocks [9]. Two distinct companies x and y are involved in a strong link if they share (at least) one significant shareholder. The Strong Link scenario can be modeled in Warded Datalog$^{\pm}$ as follows.

Example 2. *Strong Link modeled with Warded rules.*

$$\mathit{Company}(x) \rightarrow \exists p \exists s\ \mathit{Owns}(p, s, x) \quad (1)$$
$$\mathit{Owns}(p, s, x) \rightarrow \mathit{Stock}(x, s) \quad (2)$$
$$\mathit{Owns}(p, s, x) \rightarrow \mathit{PSC}(x, p) \quad (3)$$
$$\mathit{PSC}(x, p), \mathit{Controls}(x, y) \rightarrow \exists s\ \mathit{Owns}(p, s, y) \quad (4)$$
$$\mathit{PSC}(x, p), \mathit{PSC}(y, p), x \neq y \rightarrow \mathit{StrongLink}(x, y) \quad (5)$$
$$\mathit{StrongLink}(x, y) \rightarrow \exists p \exists s\ \mathit{Owns}(p, s, x) \quad (6)$$
$$\mathit{StrongLink}(x, y) \rightarrow \exists p \exists s\ \mathit{Owns}(p, s, y) \quad (7)$$
$$\mathit{Stock}(x, s) \rightarrow \mathit{Company}(x) \quad (8)$$



(a) Reasoning times of Spec and All scenarios.



(b) Strong Links discovered in All scenario.

Figure 3: Results of the experiments performed for Spec and All scenarios, to vary the number of companies.

	1K	*5K*	*10K*	*15K*	*20K*	*25K*	*30K*
Spec Times	7993	9187	10156	10795	11473	12313	13186
All Times	3158	5519	8751	9334	10437	13501	15749
Spec Links	3	3	7	8	9	9	10
All Links	716	2917	5935	8434	11050	13857	16390

Figure 4: Times in msecs and # strong links collected in Spec and All scenarios, to vary the number of companies.

For each company x there exists a person p who owns a certain share s (rule 1), which is part of the company stock (rule 2). If p owns a share s (rule 3), p is a "person with significant control" (psc) for x. For each company y controlled by x, p owns a certain share s of y (rule 4). If two distinct companies x and y share a common psc, they are involved in a strong link (rule 5). Vice versa, if there is a strong link between x and y, there exists a person p who owns a share of x (rule 6) and y (rule 7). If s is a share of a stock for x, then x is a company (rule 8).

Indeed, reasoning on the Strong Link program does not terminate, as the existential quantification in rule 4, rule 6 and rule 7, and their recursion with rule 3 and rule 5, cause the generation of an infinite set of PSC facts in the CHASE. Moreover, the isomorphism termination strategy cannot be applied, due to the presence of the harmful join rule 5 that hampers the reasoning boundedness of the program. Therefore, we apply HJE to enable reasoning with the Vadalog system. Note that rule 1, rule 6 and rule 7 are direct causes of affectedness for the PSC atoms in rule 5, whereas rule 3 and rule 4 are indirect ones. Due to the recursion on such causes, the back-composition phase builds a hu-tree which consists of both unfolding and folding leaves. The harmless rules that replace rule 5 in Example 2 are provided below (atoms are renamed and some rules are merged for space reasons). As in Example 1, δs are added via grounding, whereas τs and υs label hu-tree leaves deriving from unfolding and folding, respectively. Note that some rules were dropped during cleanup, as they would have never activated.

$$Dom(p), PSC(x,p) \rightarrow PSC'(x,p) \quad (\delta_1)$$
$$PSC'(x,p) \rightarrow PSC(x,p) \quad (\delta_2)$$
$$PSC'(x,p), PSC'(y,p), x \neq y \rightarrow SLink(x,y) \quad (\delta_3)$$
$$Comp(x), Contr(x,y) \rightarrow SLink(x,y), SLink(y,x) \quad (\tau_{1,2})$$
$$Comp(x), Contr(x,y), Contr(x,z) \rightarrow SLink(y,z) \quad (\tau_3)$$
$$SLink(x,y), Contr(x,w), Contr(x,z) \rightarrow SLink(w,z) \quad (\tau_4)$$
$$SLink(x,y), Contr(x,w), Contr(x,z) \rightarrow SLink(z,w) \quad (\tau_5)$$
$$SLink(x,y), Contr(x,z) \rightarrow SLink(x,z), SLink(z,x) \quad (\tau_{6,7})$$
$$SLink(x,y), Contr(x,z) \rightarrow SLink(y,z) \quad (\upsilon_1)$$
$$SLink(x,y), Contr(x,z) \rightarrow SLink(z,y) \quad (\upsilon_2)$$

Experiments and Results. We extracted input data for companies and ownership relations from the open KG provided by DBpedia [12], which publishes information for around 67K companies. We adopted datasets of increasing complexity, with 1K, 5K, 10K, 15K, 20K, 25K and 30K companies, respectively. The Vadalog system, with HJE integrated as part of its *logic optimizer* [3], was used as a "library" and invoked from specific Java test classes for end-to-end executions of the reasoning. We run the experiments on a local installation of the Vadalog system, with a MacBook Pro i7 with 2.5 GHz and 8 GB of RAM. We first applied HJE to the program, with average time of under 1 second. Then, we built two reasoning scenarios:

- *SpecStrongLinks* (Spec), to obtain all the strong links of the company *BBC*, to vary the number of companies;
- *AllStrongLinks* (All), to obtain all the possible pairs of strong links, to vary the number of companies.

The tasks were analyzed both in terms of time required for the reasoning to terminate and of number of strong links detected. Figure 3(a) illustrates the former perspective. The results prove the very good performance of the Vadalog system, even if tested on a local installation. Indeed, the Spec scenario requires more time than the All one for smaller datasets; when the number of companies increases, All tends to a steeper curve. On the other hand, Figure 3(b) shows the strong links founds between pairs of distinct companies in the All scenario. Finally, Figure 4 provides all the numerical results of the experiments.

5. Conclusion

Employing powerful reasoning engines such as Warded Datalog$^{\pm}$-based Vadalog system allows companies to solve relevant tasks in the financial and corporate realm. Among them, the Strong Link problem proved to be problematic to tackle, due to the presence of harmful joins and recursion that hampered termination and decidability of the reasoning in VADALOG. In this work, we discussed the disarmament problem of rewriting Warded settings with such problematic joins into an equivalent Harmless Warded form, while upholding the correctness of the task. We contributed the Harmful Join Elimination, a disarmament algorithm integrated into the Vadalog system, and we applied it to enable reasoning on Strong Link.

References

[1] M. Krötzsch, V. Thost, Ontologies for knowledge graphs: Breaking the rules, in: International Semantic Web Conference (1), volume 9981 of *Lecture Notes in Computer Science*, 2016, pp. 376–392.

[2] G. Gottlob, A. Pieris, Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue, in: IJCAI, 2015.

[3] L. Bellomarini, D. Benedetto, G. Gottlob, E. Sallinger, Vadalog: A modern architecture for automated reasoning with large knowledge graphs, Information Systems (2020) 101528.

[4] L. Bellomarini, G. Gottlob, A. Pieris, E. Sallinger, Swift logic for big data and knowledge graphs, in: IJCAI, Springer, 2017, pp. 2–10.

[5] A. Calì, G. Gottlob, T. Lukasiewicz, A general datalog-based framework for tractable query answering over ontologies, in: PODS, 2009, pp. 77–86.

[6] D. Maier, A. O. Mendelzon, Y. Sagiv, Testing implications of data dependencies, ACM Transactions on Database Systems 4 (1979) 455–468.

[7] A. Calì, 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.

[8] A. Calì, G. Gottlob, M. Kifer, Taming the infinite chase: Query answering under expressive relational constraints, Journal of Artificial Intelligence Research 48 (2013) 115–174.

[9] L. Bellomarini, E. Sallinger, G. Gottlob, The Vadalog System: Datalog-based reasoning for knowledge graphs, VLDB 11 (2018).

[10] 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.

[11] P. Atzeni, L. Bellomarini, M. Iezzi, E. Sallinger, A. Vlad, Weaving enterprise knowledge graphs: The case of company ownership graphs., in: EDBT, 2020, pp. 555–566.

[12] DBpedia, 2018. URL: http://wiki.dbpedia.org/services-resources/downloads/dbpedia-tables.

[13] G. Gottlob, S. Rudolph, M. Simkus, Expressiveness of guarded existential rule languages, in: PODS, 2014, pp. 27–38.

[14] F. Afrati, M. Gergatsoulis, F. Toni, Linearisability on datalog programs, Theoretical Computer Science 308 (2003) 199–226.

[15] M. Kaminski, Y. Nenov, B. C. Grau, Datalog rewritability of disjunctive datalog programs and non-Horn ontologies, Artificial Intelligence 236 (2016) 90–118.

[16] M. Kónig, M. Leclere, M.-L. Mugnier, Query rewriting for existential rules with compiled preorder, in: IJCAI, 2015, pp. 3006–3112.

[17] 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.

[18] N. Francis, L. Segoufin, C. Sirangelo, Datalog rewritings of regular path queries using views, arXiv preprint arXiv:1511.00938 (2015).

[19] G. Stefanoni, B. Motik, I. Horrocks, Small datalog query rewritings for el*, in: Proc. 25th Int'l Workshop on Description Logics, Citeseer, 2012.

[20] S. Ahmetaj, M. Ortiz, M. Simkus, Polynomial datalog rewritings for expressive description logics with closed predicates., in: IJCAI, 2016, pp. 878–885.

[21] G. Berger, G. Gottlob, A. Pieris, E. Sallinger, The space-efficient core of Vadalog, in: PODS, 2019, pp. 270–284.

[22] T. Baldazzi, L. Bellomarini, E. Sallinger, P. Atzeni, iwarded: A system for benchmarking datalog+/- reasoning (technical report), arXiv preprint arXiv:2103.08588 (2021).

[23] H. Tamaki, T. Sato, Unfold/fold transformation of logic programs, in: ICLP, Uppsala University, 1984, pp. 127–138.