Exploiting the Power of Equality-generating Dependencies in Ontological Reasoning

Luigi Bellomarini
Banca d'Italia
luigi.bellomarini@bancaditalia.it

Davide Benedetto
Università Roma Tre
davide.benedetto@uniroma3.it

Matteo Brandetti
TU Wien
matteo.brandetti@gmail.com

Emanuel Sallinger
TU Wien & University of Oxford
sallinger@dbai.tuwien.ac.at

ABSTRACT

Equality-generating dependencies (EGDs) allow to fully exploit the power of existential quantification in ontological reasoning settings modeled via Tuple-Generating Dependencies (TGDs), by enabling value-assignment or forcing the equivalence of fresh symbols. These capabilities are at the core of many common reasoning tasks, including graph traversals, clustering, data matching and data fusion, and many more related real-world scenarios.

However, the interplay of TGDs and EGDs is known to lead to undecidability or intractability of query answering in tractable Datalog+/- fragments, like Warded Datalog+/-, for which, in the sole presence of TGDs, query answering is PTIME in data complexity. Restrictions of equality constraints, like separable EGDs, have been studied, but all achieve decidability at the cost of limited expressive power, which makes them unsuitable for the mentioned tasks.

This paper introduces the class of "harmless" EGDs, that subsume separable EGDs and allow to model a very broad class of tasks. We contribute a sufficient syntactic condition for testing harmlessness, an undecidable task in general. We argue that in Warded Datalog+/- with harmless EGDs, ontological reasoning is decidable and PTIME. From such theoretical underpinnings, we develop novel chase-based techniques for reasoning with harmless EGDs and present an implementation within the Vadalog system, a state-of-the-art Datalog-based reasoner. We provide full-scale experimental evaluation and comparative analysis.

PVLDB Reference Format:
Luigi Bellomarini, Davide Benedetto, Matteo Brandetti, and Emanuel Sallinger. Exploiting the Power of Equality-generating Dependencies in Ontological Reasoning. PVLDB, 15(13): 3976 - 3988, 2022.
doi:10.14778/3565838.3565850

1 INTRODUCTION

Logic-based ontological reasoning is gaining renewed attention, as witnessed by the recent resurgence of the *Datalog* language in academia and in industry [4, 11, 24–26, 30, 59, 63]. Intuitively speaking, an ontological reasoning task consists in answering a conjunctive query Q over a database D, augmented with a set of logical rules Σ, as shown in the following example.

Example 1.1. A shock propagation scenario from our industrial partners, where a database D is augmented with a set of rules Σ describing the domain, as follows.

D=\{*Own*(*Bob*, *C*, *0.4*), *Own*(*Max*, *C*, *0.35*), *Own*(*Alice*, *D*, *0.5*), *Own*(*Markus*, *E*, *0.6*), *Company*(*C*), *Company*(*D*), *Company*(*E*), *NPL*(*C*), *Exposure*(*C*, *D*), *Exposure*(*D*, *E*)\}.

$$Company(c), Own(p,c,w), w > 0.3 \rightarrow KP(p,c) \quad (\sigma_1)$$
$$Company(c), NPL(c), \rightarrow \exists f\ Default(c,f,f) \quad (\sigma_2)$$
$$Default(c_1,f_x,f_1), Exposure(c_1,c_2), \rightarrow \exists f_2\ Default(c_2,f_1,f_2) \quad (\sigma_3)$$
$$Default(c,f_1,f_2), KP(p,c) \rightarrow \exists i\ Inv(p,c,i) \quad (\sigma_4)$$

An individual p is a key person (KP) of a company c if p owns more than 30% of the shares (w) of c (σ_1). If a company c is involved in non-performing loans (NPL), then it will default on its debts, initiating a failure event f (σ_2). If a company c_2 is financially exposed with another company c_1 which undergoes a failure event f_1, caused by another failure f_x, then c_2 will be in turn involved in a failure f_2 caused by f_1 (σ_3). Finally, a financial investigation i regards each key person of a defaulting company c (σ_4).

Rules in Σ are function-free Horn clauses, potentially including existential quantification, i.e., *Tuple-Generating Dependencies* (TGDs). They have the form $\forall \mathbf{x}\ \boldsymbol{\phi}(\mathbf{x}) \rightarrow \exists \mathbf{z}\ \boldsymbol{\psi}(\mathbf{y},\mathbf{z})$, where $\boldsymbol{\phi}(\mathbf{x})$ and $\boldsymbol{\psi}(\mathbf{y},\mathbf{z})$ are conjunctions of atoms over a relational schema S. Universal quantifiers are implied and we shall omit them. The semantics of TGDs is usually defined in an operational way with an algorithmic tool known as the CHASE procedure [53]. Intuitively, the chase expands D with facts entailed via the application of the TGDs in Σ, until all of them are satisfied, introducing fresh new symbols (i.e., *labelled nulls*) to satisfy existential quantification.

Now, consider the ontological reasoning task in which, given D and Σ, we want to understand whether two people are involved in the same investigation via the *Boolean Conjunctive Query* (BCQ):

$$q \leftarrow Inv(Bob, \_, z), Inv(Markus, \_, z)$$

By applying the chase on σ_1, we derive *KP*(*Bob*, *C*), *KP*(*Max*, *C*), *KP*(*Alice*, *D*) and *KP*(*Markus*, *E*). Then, we identify the defaulting company *Default*(C, v_1, v_1) by σ_2, where v_1 is a labelled null. The activation of σ_3 propagates the default to company D, obtaining *Default*(D, v_1, v_2), since company D is exposed with C. Then, σ_3 is activated again to produce *Default*(E, v_2, v_3). Finally, by σ_4 we

Proceedings of the VLDB Endowment, Vol. 15, No. 13 ISSN 2150-8097.
doi:10.14778/3565838.3565850

derive $Inv(Bob, C, v_4)$, $Inv(Max, C, v_5)$, $Inv(Alice, D, v_6)$ and $Inv(Markus, E, v_7)$, that is, the key people under investigation. Now, it can be observed that the answer to Q is negative, as $v_4 \neq v_7$, while in fact Markus and Bob should be part of the same investigation, as they are involved in the same default chain initiated by C.

Example 1.2. To capture such requirement, let us extend Σ by adding the following set Σ_E of rules.

$$KP(p_1, c), KP(p_2, c), Inv(p_1, c, i_1), Inv(p_2, c, i_2) \to i_1 = i_2 \quad (\eta_1)$$

$$Inv(p_1, c_1, i_1), Inv(p_2, c_2, i_2), Exposure(c_1, c_2) \to i_1 = i_2 \quad (\eta_2)$$

If p_1 and p_2 are key persons of the same defaulted company c, then they should be involved in the same investigation (η_1). Again, two key persons p_1 and p_2 of distinct defaulted companies c_1 and c_2 are under the same investigation if c_2 is exposed with c_1 (η_2).

The rules in Σ_E are *Equality-Generating Dependencies* (EGDs), i.e., first-order implications of the form $\forall \mathbf{x}\ \boldsymbol{\phi}(\mathbf{x}) \to x_i = x_j$, where $\boldsymbol{\phi}(\mathbf{x})$ is a conjunction of atoms over a relational schema and x_i, x_j are variables in $\mathbf{x}$. The semantics of EGDs is defined via a straightforward extension of the chase: as long as the rule premise applies, the equality in the conclusion is enforced, either assigning a labelled null to a constant value or to another labelled null, or comparing two constants, which may potentially lead to chase failure by hard violation of the rule.

In our case, from the application of EGD η_1 on $KP(Bob, C)$, $KP(Max, C)$, $Inv(Bob, C, v_4)$, $Inv(Max, C, v_5)$, we derive that $v_4 = v_5$, i.e., we assign *Bob* and *Max* to the same labelled null. Then, we activate η_2 on $Inv(Max, C, v_5)$, $Inv(Alice, D, v_6)$, $Exposure(C, D)$ and on $Inv(Alice, D, v_6)$, $Inv(Markus, E, v_7)$, $Debtor(D, E)$ to conclude that $v_5 = v_6$ and $v_6 = v_7$. As a result, thanks to the application of the EGDs, all the key persons are grouped together and subjects to the same investigation, with a positive answer to Q.

This example shows how the EGDs allow to exploit the expressive power of existential quantification and efficiently solve many relevant tasks through reasoning, which would be impossible to express or inefficient to evaluate by using only TGDs, such as graph navigation [3], feature-based clustering [22], data matching and integration [37], data fusion [20], also in many real-world scenarios.

Reasoning Languages with EGDs. The interplay of TGDs and EGDs leads to undecidability of ontological reasoning with Datalog$^\pm$ fragments (e.g., *Guarded*, *Warded* and *Shy*), while, in the sole presence of TGDs, such task is PTIME in data complexity [24, 25, 41, 49]. For instance, the task is undecidable even with inclusion and functional dependencies, or inclusion and key dependencies [31, 34].

There have been many attempts to identify Datalog decidable fragments by imposing restrictions on the interaction between TGDs and EGDs. *Weakly-acyclic* sets of TGDs guarantee decidable and tractable query answering when interacting with EGDs [35, 36, 54]. However, while their expressive power is suited for *data exchange* scenarios [36], they are not helpful for ontological reasoning because of their limitation in the joint use of existential quantification and recursion. For instance, it is impossible to express simple reasoning tasks like the one in Examples 1.1 and 1.2.

The class of *separable* EGDs enables a richer tractable interaction with the TGD fragments devised for ontological reasoning (e.g., *Guarded* TGDs) [25, 27, 32]. Yet, the expressive power of the EGDs is not exploited. In fact, separable EGDs can only enforce hard constraints on ground values and do not contribute facts to the reasoning query answer. For instance, the reasoning task captured by EGDs in Example 1.2 cannot be expressed with separable EGDs.

Harmless EGDs. In this paper, we propose *harmless EGDs*, a novel language for equality-generating dependencies that is well-suited for ontological reasoning and allows to exploit existential quantification to model a wide range of tasks. We formalize the language and implement it in Vadalog, a state-of-the-art Datalog-based reasoner. As a key feature, harmless EGDs non-trivially interact with *warded TGDs* [41], a fragment of the Datalog$^\pm$ family of languages [29] exhibiting very good balance between computational complexity—with ontological reasoning being PTIME in data complexity—and expressive power, in fact capturing SPARQL queries under the entailment regime for OWL 2 QL. At the same time, harmless EGDs do not affect decidability and tractability of the ontological reasoning task with warded TGDs. Intuitively, for every database and for every query, any assignment of labelled nulls obtained through the application of a harmless EGD, does not trigger the activation of other rules that would not be activated otherwise. As a consequence, no EGD can determine the derivation of a fact and, in this sense, does not "harm" the chase procedure. Indeed, unlike *separable* EGDs, harmless EGDs contribute facts to the ontological reasoning task. Consider again TGDs of the Example 1.1 (warded) and the EGDs of the Example 1.2 (harmless): in the absence of η_1 and η_2, a positive answer to Q would not be possible.

Warded Datalog$^\pm$ with harmless EGDs has broad practical applicability in ontological query answering. In fact, our use of EGDs is more general than separability and goes beyond pure value/id invention [47], thanks to the possibility to group facts by linking them via shared labelled nulls. In practice, this translates into the power to jointly use existentials and EGDs to model the broad set of tasks we have mentioned, including problems requiring the expressive power of transitive closure such as graph traversals and search, in a compact and efficient fashion.

Our **contribution** can be summarized as follows:

- We introduce the class of **harmless EGDs** showing that they non-trivially interact with TGDs and allow a broad application of labelled nulls in ontological reasoning. After arguing that establishing whether a set of EGDs is harmless is undecidable, we contribute a **sufficient syntactic condition**, namely *safe taintedness*, which witnesses harmless EGDs in many practical cases.
- We deal with **reasoning with harmless EGDs** in Warded Datalog$^\pm$ and we prove the problem is decidable and PTIME in data complexity. We provide the foundations for practical reasoning algorithms with harmless EGDs based on a **new variant of the chase** that considers limited portions of the reasoning graph.
- We describe the **implementation of harmless EGDs** in the Vadalog system, a state-of-the-art reasoner [13].
- We provide full-scale **experimental evaluation** of harmless EGDs in a variety of real-world and synthetic scenarios and validate the effectiveness of our techniques. We compare our reasoner with the top existing systems supporting EGDs, and show that it exhibits superior performance and expressive power.

Overview. The remainder of this paper is organized as follows. In Section 2 we provide the fundamental background notions. In

Section 3 we present the new class of harmless EGDs and the properties to handle recursion and termination. In Section 4 we illustrate the implementation of the harmless EGDs in the Vadalog system. Section 5 is dedicated to the practical aspects and the experimental evaluation. In Section 6 we discuss the related work. Our conclusions are drawn in Section 7. For space reasons, further use cases, examples and proofs are in the online Appendix [12].

2 PRELIMINARIES

Let us start by laying out the preliminary notions.

Relational Foundations. Let $\mathbf{C}$, $\mathbf{N}$, and $\mathbf{V}$ be disjoint countably infinite sets of *constants*, *(labelled) nulls* and *variables*, respectively. A *lexicographic ordering* is defined so that any constant in $\mathbf{N}$ follows all the constants in $\mathbf{C}$. A *term* is a constant, a variable or a labelled null. A *(relational) schema* $\mathbf{S}$ is a finite set of relation symbols with associated arities. Given a schema $\mathbf{S}$, an *atom* is an expression $R(\bar{v})$, where $R \in \mathbf{S}$ is of arity $n \geq 0$ and $\bar{v}$ is an n-tuple of terms. A *database (instance)* D over $\mathbf{S}$ associates to each relation symbol in $\mathbf{S}$ a relation of the respective arity over the domain of constants and nulls. We denote as $\text{dom}(D)$ the set of constants in D. Relation members are called *tuples* or *facts*. Sometimes we will use the terms tuple or fact interchangeably. Given two sets of atoms A_1 and A_2, we define a *homomorphism* from A_1 to A_2, a mapping $h : \mathbf{C} \cup \mathbf{N} \cup \mathbf{V} \to \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ such that $h(t) = t$, if $t \in \mathbf{C}$ (i.e., constants are preserved), and for each atom $a(t_1, \ldots, t_n) \in A_1$, we have that $h(a(t_1, \ldots, t_n)) = a(h(t_1), \ldots, h(t_n))$ is in A_2.

We define a *partial* homomorphism h from A_1 to A_2 a mapping of a subset $B \subseteq A_1$ to A_2, i.e., $h(B) \subseteq A_2$. We say that a homomorphism h maps A_1 *onto* A_2, whenever $h(A_1)$ coincides with A_2. Two atoms $a_1(t_1, \ldots, t_n)$ and $a_2(t_1, \ldots, t_n)$ are isomorphic if a_1 and a_2 refer to the same predicate and there exists a mapping $h : \mathbf{C} \cup \mathbf{N} \cup \mathbf{V} \to \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ s.t. $h(t) = t$, if $t \in \mathbf{C}$ (i.e., constants are preserved), and we have that $h(a_1(t_1, \ldots, t_n)) = a_1(h(t_1), \ldots, h(t_n)) = a_2(t_1, \ldots, t_n)$ and $h(a_2(t_1, \ldots, t_n)) = a_2(h(t_1), \ldots, h(t_n)) = a_1(t_1, \ldots, t_n)$, i.e., h is a bijection. This notion can be extended to sets. Two sets of atoms A_1 and A_2 are isomorphic if every atom in A_1 (resp. A_2) has an isomorphic atom in A_2 (resp. A_1).

Conjunctive Queries. A *conjunctive query* (CQ) Q over a schema $\mathbf{S}$ is an implication $q(\mathbf{x}) \leftarrow \boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$, where $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms over $\mathbf{S}$, $q(\mathbf{x})$ is an n-ary predicate that does not occur in $\mathbf{S}$, and $\mathbf{x}$ and $\mathbf{y}$ are vectors of terms. A *Boolean conjunctive query* (BCQ) is a CQ of arity zero.

Dependencies. A set of Datalog$^{\pm}$ rules is a set of tuple-generating dependencies (TGDs). A TGD is a first-order implication $\forall \mathbf{x}\, \boldsymbol{\phi}(\mathbf{x}) \to \exists \mathbf{z}\, \boldsymbol{\psi}(\mathbf{y}, \mathbf{z})$, where $\boldsymbol{\phi}(\mathbf{x})$ (the *body*) and $\boldsymbol{\psi}(\mathbf{y}, \mathbf{z})$ (the *head*) are conjunctions of atoms over a relational schema and boldface variables denote vectors of variables, with $\mathbf{y} \subseteq \mathbf{x}$. A TGD σ is satisfied by a database D (and we write $D \models \sigma$) if whenever there is a homomorphism θ such that $\theta(\boldsymbol{\phi}(\mathbf{x})) \subseteq D$, there exists an *extension* θ' of θ (i.e., $\theta \subseteq \theta'$) such that $\theta'(\boldsymbol{\psi}(\mathbf{y}, \mathbf{z})) \subseteq D$. An *equality-generating dependency* (EGD) is a first-order implication $\forall \mathbf{x}\, \boldsymbol{\phi}(\mathbf{x}) \to x_i = x_j$, where $\boldsymbol{\phi}(\mathbf{x})$ is a conjunction of atoms and $x_i, x_j \in \mathbf{x}$. A database D over $\mathbf{S}$ satisfies an EGD η if whenever there is a homomorphism θ such that $\theta(\boldsymbol{\phi}(\mathbf{x})) \subseteq D$, then we have that $\theta(x_i) = \theta(x_j)$.

Ontological Reasoning. Given a database D over $\mathbf{S}$ and a set $\Sigma = \Sigma_T \cup \Sigma_E$ of TGDs (Σ_T) and EGDs (Σ_E), we name the *models* of D and Σ as the set of all databases B (and we write $B \models D \cup \Sigma$) such that $B \supseteq D$, and $B \models \Sigma$. The answer to a CQ Q over D under Σ is the set of facts t such that $t \in Q(B)$, where $B \models D \cup \Sigma$. A positive answer to a BCQ ($D \cup \Sigma \models Q$) corresponds to a non-empty set of tuples, such that $t \in Q(B)$. Query answering under general TGDs is undecidable even when Q and Σ are fixed [24]. Warded Datalog$^{\pm}$ is a member of the Datalog$^{\pm}$ family with a good trade-off between expressive power and computational complexity, with CQ answering in PTIME. In the presence of EGDs, query answering under TGDs is undecidable, even for simple classes of EGDs and TGDs, such as functional and inclusion dependencies, or key and inclusion dependencies [31, 34]. Existing fragments, such as *separable EGDs* [23], allow very limited forms of interaction between TGDs and EGDs, which do not hamper tractability and decidability. CQ answering and BCQ answering under TGDs and EGDs are LOGSPACE-equivalent as the decision version of CQ answering and BCQ answering are mutually AC_0-reducible [24]. Hence, we will consider BCQs without loss of generality.

The Chase. Chase-based procedures [53] repair a database D by adding facts to it, until it satisfies a set of constraints Σ. Intuitively, the chase expands D with facts inferred by applying Σ to D into a database *chase*(D, Σ), possibly containing labelled nulls. A chase execution *chase*(D, Σ) builds a *universal model* for D and Σ, i.e., for every database B that is a model for D and Σ, there is a homomorphism mapping *chase*(D, Σ) to B. Let us recall two working rules: the *TGD chase step* and the *EGD chase step*. In the following, we will refer to the oblivious chase [24]. Given a database D, a TGD $\sigma : \boldsymbol{\phi}(\mathbf{x}) \to \exists \mathbf{z}\, \boldsymbol{\psi}(\mathbf{y}, \mathbf{z})$ is applicable to D if there exists a homomorphism θ such that $\theta(\boldsymbol{\phi}(\mathbf{x})) \subseteq D$. Then, the TGD chase step adds the fact $\theta'(\boldsymbol{\psi}(\mathbf{y}, \mathbf{z}))$ to D, if not already in or already added to D, where $\theta' \supseteq \theta$ is a homomorphism that extends θ by mapping the variables of $\mathbf{z}$ (if non-empty) to newly created labelled nulls that follow lexicographically the ones previously introduced. An EGD $\eta : \boldsymbol{\phi}(\mathbf{x}) \to x_i = x_j$ is applicable to D if there exists a homomorphism θ such that $\theta(\boldsymbol{\phi}(\mathbf{x})) \subseteq D$ and $\theta(x_i) \neq \theta(x_j)$. Given $\boldsymbol{\varphi}(\mathbf{x}) = \theta(\boldsymbol{\phi}(\mathbf{x})) \subseteq D$, an EGD chase step $\boldsymbol{\varphi}(\mathbf{x}) \xrightarrow{\eta_\theta} x_i = x_j$ proceeds as follows: (i) if both x_i and x_j are constants, it fails; (ii) replaces each occurrence of $\theta(x_j)$ with $\theta(x_i)$, if $\theta(x_i)$ precedes $\theta(x_j)$ in the lexicographic order, or vice versa otherwise. The chase iteratively applies (i) a TGD step once, (ii) the EGD step as long as applicable, i.e., until a fixpoint is reached, which may lead to an infinite sequence of chase step applications. The chase graph $\mathcal{G}(D, \Sigma)$ is a directed graph having as nodes the facts from *chase*(D, Σ) and having an edge from a node $\mathbf{a}$ to $\mathbf{b}$ if $\mathbf{b}$ is obtained from $\mathbf{a}$ by the application of one TGD chase step.

Separable EGDs. Given a set of TGDs and EGDs $\Sigma = \Sigma_T \cup \Sigma_E$, Σ_E is separable from Σ_T if for every database D: (i) if the chase of Σ over D fails, then D does not satisfy Σ_E; (ii) if the chase does not fail, then we have that $\text{chase}(D, \Sigma) \models Q$ iff $\text{chase}(D, \Sigma_T) \models Q$ for every BCQ Q. The main deficiency of such dependencies is their *limited unification power*. While, syntactically, separable EGDs can equate variables of predicates not appearing in $\mathbf{S}$, they never cause distinct labelled nulls to be unified, hence, they never contribute to the query answer and can be verified against D.

3 HARMLESS EGDS

Towards the definition of a framework able to support ontological reasoning with expressive interaction between TGDs and EGDs, in this section we provide the theoretical foundations that underpin a practical reasoning algorithm, as we shall see.

3.1 Harmlessness

The key idea of our reasoning framework is to consider a set of EGDs Σ_E to be *harmless* with respect to a set of rules $\Sigma = \Sigma_T \cup \Sigma_E$ (where Σ_T are the TGDs), if $chase(D, \Sigma_T)$ produces a more general result than $chase(D, \Sigma)$ for every D. Intuitively, harmlessness requires that for every D, $chase(D, \Sigma_T)$ can be mapped onto $chase(D, \Sigma)$ with an assignment of the labelled nulls to either null values or constant values. This corresponds to guaranteeing that every fact of $chase(D, \Sigma)$ can be "reached" by specializing a fact in $chase(D, \Sigma_T)$, via an assignment of the labelled nulls. Operationally, this condition prevents the EGDs in Σ from interfering with the activation of the TGDs: actually, harmless EGDs may not produce facts that, in turn, fire TGDs that would not be activated otherwise (e.g., in the sole presence of the TGDs): if it were the case, the activated TGDs would produce facts potentially "unreachable" via a mapping of $chase(D, \Sigma_T)$. As TGDs can determine the activation of EGDs and not vice versa, harmless EGDs can be enforced after all TGD chase steps have been performed, by applying the equalities of Σ_E on the facts of $chase(D, \Sigma_T)$ to fixpoint.

Definition 3.1 (Harmless EGDs). Given a set of TGDs and EGDs $\Sigma = \Sigma_T \cup \Sigma_E$ over a schema S*, we define Σ_E as* harmless *with respect to Σ if for every database D over* S*, if $chase(D, \Sigma)$ does not fail, then there exists a homomorphism h mapping $chase(D, \Sigma_T)$ onto $chase(D, \Sigma)$.*

Let us consolidate the intuition with the following example.

Example 3.2. Consider a formulation of the undirected graph connectivity problem (UST-CONN) [62] based on connected components: two nodes x and y are connected if they lie in the same connected component, i.e., $Q \leftarrow CC(x, c), CC(y, c)$. We augment a database D holding the graph, with a set of harmless EGDs as follows.

D={$Node(a)$, $Node(b)$, $Node(c)$, $Node(d)$, $Edge(a, b)$, $Edge(b, c)$, $Edge(c, d)$, $Edge(a, d)$}.

$$Node(x) \rightarrow \exists z\, CC(x, z) \qquad (\sigma_1)$$

$$Edge(x, y) \rightarrow Edge(y, x) \qquad (\sigma_2)$$

$$CC(x, z_1), Edge(x, y), CC(y, z_2) \rightarrow z_1 = z_2 \qquad (\eta)$$

Every node is assigned to a new connected component (σ_1). The graph is undirected (σ_2). Components linked by an edge are the same (η). The TGD applications to construct $chase(D, \Sigma_T)$ are as follows:

$$Node(a) \rightarrow CC(a, v_0);\ Node(b) \rightarrow CC(b, v_1); \qquad (\sigma_1)$$

$$Node(c) \rightarrow CC(c, v_2);\ Node(d) \rightarrow CC(d, v_3); \qquad (\sigma_1)$$

$$Edge(a, b) \rightarrow Edge(b, a);\ Edge(b, c) \rightarrow Edge(c, b); \qquad (\sigma_2)$$

$$Edge(c, d) \rightarrow Edge(d, c);\ Edge(a, d) \rightarrow Edge(d, a); \qquad (\sigma_2)$$

We then apply the EGD η to fixpoint to derive $chase(D, \Sigma)$:

$$CC(a, v_0), Edge(a, b), CC(b, v_1) \rightarrow v_0 = v_1; \qquad (\eta)$$

$$CC(b, v_0), Edge(b, c), CC(c, v_2) \rightarrow v_0 = v_2; \qquad (\eta)$$

$$CC(c, v_0), Edge(c, d), CC(d, v_3) \rightarrow v_0 = v_3; \qquad (\eta)$$

As a final result, we obtain the facts $CC(a, v_0)$, $CC(b, v_0)$, $CC(c, v_0)$ and $CC(d, v_0)$, where all the labelled nulls created by the TGD applications are mapped into the same value v_0 by η.

Observe that in Example 3.2, after the application of the EGDs, no further TGDs can be triggered, independently of the underlying database instance D. This is ensured by the requirement posed by harmlessness, i.e., to be able to map $chase(D, \Sigma_T)$ onto $chase(D, \Sigma)$ via a homomorphism h. Also notice that the specific homomorphism is produced as a side effect of the EGD application. In this case we have h={$v_0 \rightarrow v_0$, $v_1 \rightarrow v_0$, $v_2 \rightarrow v_0$, $v_3 \rightarrow v_0$}.

3.2 Safe Taintedness: A Syntactic Condition

We now study the problem of identifying a set of EGDs as harmless with respect to Σ.

THEOREM 3.3. *Given a set of TGDs and EGDs $\Sigma = \Sigma_T \cup \Sigma_E$, deciding whether Σ_E is harmless with respect to Σ is undecidable.*

In spite of such undecidability result, we introduce a sufficient syntactic condition to check harmlessness. In particular, our condition applies when harmless EGDs interact with warded TGDs.

First, we recall some working definitions of the warded fragment [41]. Let $p[i]$ be the attribute in the i-th position of a predicate p and refer to it as *position*. Let $exist(\sigma)$ be the set of existentially quantified variables of σ. Given a set of rules Σ, a position $p[i]$ is defined as *affected* if: (i) for some TGD $\sigma \in \Sigma$ and some variable $v \in exist(\sigma)$, v appears in position $p[i]$, for some atom over p in Σ; (ii) for some TGD σ and some variable $v \in body(\sigma) \cap head(\sigma)$, v appears only in affected positions in $body(\sigma)$ (i.e., it is *harmful*) and in position $p[i]$ in $head(\sigma)$ (i.e., it is *dangerous*).

We are ready to introduce the notions of *tainted position* and *tainted variable*, used in our syntactic condition.

Definition 3.4 (Taintedness). Given a set of rules Σ, a position $p[i]$ is inductively defined as tainted if: (i) *$p[i]$ is affected and Σ contains an EGD $\eta = \boldsymbol{\phi}(\mathbf{x}) \rightarrow x_i = x_j$ such that $p[i]$ is the position of x_i (resp. x_j) in $body(\eta)$ and x_i (resp. x_j) appears only in affected positions of $body(\eta)$ (it is harmful);* (ii) *for some TGD $\sigma \in \Sigma$ and some variable $v \in body(\sigma) \cap head(\sigma)$, v appears in a tainted position in $body(\sigma)$ (resp. $head(\sigma)$) and in position $p[i]$ in $head(\sigma)$ (resp. $body(\sigma)$). A variable appearing in a tainted position of a rule σ is named tainted variable (w.r.t. σ). Let $tainted(\sigma)$ be the set of all the tainted variables in the body of σ.*

We now can exploit such definition to provide a syntactic condition, *safe taintedness*, that witnesses harmlessness in practice.

THEOREM 3.5 (SAFE TAINTEDNESS). *Let $\Sigma = \Sigma_T \cup \Sigma_E$ be a set of TGDs Σ_T and EGDs Σ_E over schema* S*. The EGDs Σ_E are harmless with respect to Σ if for every dependency $\sigma \in \Sigma$: (i) every variable $v \in tainted(\sigma)$ appears only once in $body(\sigma)$, and, (ii) there are no constants appearing in tainted positions.*

Consider the set of TGDs of Example 1.1 and EGDs of Example 1.2. Position $Inv[3]$ is tainted, since: it is affected (because of the existential quantification in σ_4) and, in η_1 (and η_2), $Inv[3]$ is the position of i_1 (resp. i_2), which appears only in affected positions of $body(\eta_1)$ (and $body(\eta_2)$). However, the tainted variables i_1 and i_2 appear only once in the body of η_1 (and η_2) and no constants appear in $Inv[3]$.

Therefore Σ is harmless. Similarly, the EGD of Example 3.2 is harmless by safe taintedness. Position $CC[2]$ is tainted since: it is affected (existential quantification σ_1) and, in η, $CC[2]$ is the position of z_1 (resp. z_2), which appears only in affected positions of $body(\eta)$. Also in this case, safe taintedness holds as z_1 (resp. z_2) appears only once in the body of η and no constants appear in $CC[2]$.

3.3 Decidability and Complexity Results

We now contribute the core theoretical properties that enable an efficient practical implementation.

Warded Semantics. We need to recall more properties of the warded fragment [14, 41]. Given a set of rules Σ, a rule $\sigma \in \Sigma$ is *warded* if all the dangerous variables $v \in body(\sigma)$ appear in a single body atom, the *ward*, which shares only non-harmful variables with other body atoms. A set Σ is warded if the body variables of all the rules in Σ are warded. We say that a CQ Q is warded w.r.t. a set of TGDs Σ_T—and we name it *Warded Conjunctive Query* (WCQ)—if $\Sigma_T \cup \{\hat{Q}\}$ is warded, where $\hat{Q}$ is a TGD expressing Q. In our experience, the class of WCQs is sufficiently wide to cover all the real-world scenarios of our interest such as those dealt with in this paper, and we could verify that well-known and complex ontologies of most comprehensive chase benchmarks [15] only use WCQs. One favourable property of WCQs is that they can be equivalently rewritten into atomic queries, i.e., having a single body atom (proof in the Appendix). Also note that every BCQ Q is a WCQ, since no variables propagate to the head of $\hat{Q}$. In the rest of the paper, we will keep referring to BCQs to discuss the theoretical results without loss of generality (see Section 2), and, after presenting the mentioned rewriting approach, consider WCQs in the implementation and experimental evaluation.

A set of warded TGDs Σ_T has the property that after normalization steps (i.e., harmful joins elimination [9]), query answering on D under Σ_T can be performed over a variant of the oblivious chase (which we name $chase^W$). In such variant, an instance is repaired by activating a TGD step only if the TGD produces a fact that, up to renaming of labelled nulls, is not already in or has been already added to D. The execution of the chase for CQ answering in the warded fragment always leads to a finite sequence of chase steps.

We capture this consideration in the following theorem that directly derives from recent work [14, Th. 2].

Theorem 3.6. *Given a set Σ_T of warded TGDs and a database D, for every BCQ Q, it holds $chase^W(D, \Sigma_T) \models Q$ iff $chase(D, \Sigma_T) \models Q$.*

We will refer to the equivalence assumption of isomorphic facts and to the specific chase $chase^W$ as *warded semantics.*

Decidability. While warded semantics applies to warded TGDs, no conclusions can be drawn on whether it is applicable with warded TGDs and harmless EGDs (and, as we shall see, it is not). Hence, to argue for the decidability of BCQ answering in our context, we consider the *universal semantics* of D and Σ and we represent it as a purely theoretical $chase^B(D, \Sigma_T) \subseteq chase(D, \Sigma_T)$, finite and BCQ-equivalent to $chase(D, \Sigma_T)$, i.e., a universal model for $D \cup \Sigma_T$ (the finiteness of $chase^B$ is proved in the Appendix). To decide BCQs with harmless EGDs, after checking whether $D \cup \Sigma$ is satisfiable, we apply the TGDs Σ_T over D to construct $chase^B(D, \Sigma_T)$, then we apply Σ_E to fixpoint, i.e., we compute $chase^H(D, \Sigma) =$ $chase(chase^B(D, \Sigma_T), \Sigma_E)$. Finally, we answer Q on $chase^H(D, \Sigma)$, as $chase(D, \Sigma) \models Q$ iff $chase^H(D, \Sigma)) \models Q$. The next result argues for the correctness of our technique for BCQ decidability.

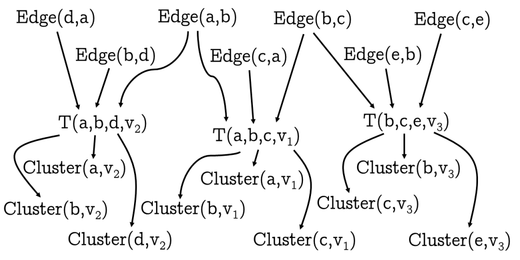

Figure 1: Chase application for Example 3.9.

Theorem 3.7. *Given a set $\Sigma = \Sigma_T \cup \Sigma_E$ of warded TGDs and harmless EGDs with respect to Σ, and a database D, if $D \cup \Sigma$ is satisfiable, for every BCQ Q, it holds $chase^H(D, \Sigma) \models Q$ iff $chase(D, \Sigma) \models Q$.*

Data Complexity. Our next result shows that the presence of harmless EGDs does not increase the data complexity of the warded fragment, which is still in PTIME.

Theorem 3.8. *BCQ answering for Warded Datalog$^{\pm}$ and harmless EGDs is PTIME-complete in data complexity.*

3.4 Relaxed Warded Semantics

We now concentrate on the correctness and the termination of the chase in our framework. We start by investigating the relationship between warded semantics and harmless EGDs.

Example 3.9. Consider the problem of detecting triangles in a graph. In the following case, we have a graph with a single connected component and three distinct triangles with non-overlapping edges.

D={$Edge(a, b)$, $Edge(b, c)$, $Edge(c, a)$, $Edge(b, d)$, $Edge(d, a)$, $Edge(c, e)$, $Edge(e, b)$}

$$Edge(x, y), Edge(y, z), Edge(z, x), x < z \to \exists w\ T(x, y, z, w) \quad (\sigma_1)$$
$$T(x, y, z, w) \to Cluster(x, w) \quad (\sigma_2)$$
$$T(x, y, z, w) \to Cluster(y, w) \quad (\sigma_3)$$
$$T(x, y, z, w) \to Cluster(z, w) \quad (\sigma_4)$$

The chase output is shown in Figure 1. Nodes x, y and z are part of the same triangle denoted by the existentially quantified variable w. With the condition $x < z$, we select only a triple of nodes for each triangle (σ_1). Then, Cluster stores the group of each node (σ_2, σ_3, σ_4).

Suppose we want to group the triangles that form a connected component. We add the harmless EGD $\eta = Cluster(x, z), Cluster(x, z')$ $\to z = z'$ and the query $Q \leftarrow Cluster(x, c), Cluster(y, c)$.

Yet, it turns out that harmless EGDs are incompatible with the warded semantics: it considers facts equivalent up to renaming of labelled nulls (i.e., isomorphic facts), whereas EGDs assign specific values to labelled nulls modifying their identity. In the example, we cannot rely on warded semantics and we need to generate multiple copies of equivalent facts, e.g., for $Cluster(a, v_1)$ and $Cluster(a, v_2)$. Actually, it is only from the comparison of different copies that η

can enforce $v_1 = v_2 = v_3$. However, not all the copies are needed: for example, once from *Cluster*(c, v_1) and *Cluster*(c, v_2) we establish $v_1 = v_2$ and from *Cluster*(c, v_1) and *Cluster*(c, v_3) we derive $v_1 = v_3$, we do not need both *Cluster*(b, v_2) and *Cluster*(b, v_3), as $v_2 = v_3$ already holds by transitivity.

Revisiting the Chase. In Theorem 3.7, we leveraged the universal model of $D \cup \Sigma_T$ to build $chase^H(D, \Sigma)$. As we cannot choose $chase^W$ to provide an efficient implementation of $chase^B$, our goal is to define a *relaxed warded semantics* $chase^{B_W}(D, \Sigma)$, a finite restriction of $chase(D, \Sigma)$ such that for every query Q, $chase^{B_W}(D, \Sigma) \models Q$ iff $chase(D, \Sigma) \models Q$ even when Σ contains a set of harmless EGDs.

To define $chase^{B_W}$, we need to introduce the notion of *warded forest* $\mathcal{W}(\mathcal{G})$ of a chase graph $\mathcal{G}(D, \Sigma)$. We refer to linear TGDs as TGDs that have a single body atom, warded join TGDs as TGDs that propagate dangerous variables and the remaining as non-linear join TGDs. The warded forest is the subgraph consisting of all nodes of the chase graph plus all the edges that correspond to the application of linear TGDs, and one edge for each warded join TGD, the one from ward. As a result, the facts are located in disconnected trees. The fact **b** is the *track* of a fact **a**, denoted as **b** = **track**(**a**), if **b** is the root of the tree in the warded forest where **a** belongs.

*Definition 3.10 (T-isomorphism). Let Σ be a set of warded TGDs, D a database, and T a fact of the chase graph $\mathcal{G}(\Sigma, D)$. Two facts **a** and **b** are* T-isomorphic, *if they are isomorphic and have the same track T = **track**(**a**) = **track**(**b**).*

T-isomorphism is a key notion to define our relaxed warded semantics $chase^{B_W}$. Operationally, it is an oblivious chase variant where an instance is repaired by activating a TGD step only if the TGD produces a fact that is not T-isomorphic to any fact that is already in or has been already added to D. We define it as follows.

Definition 3.11 (Relaxed Warded Semantics). Given a database D, a set of warded rules Σ, let $\mathbf{Q}$ be the quotient set $chase(D, \Sigma)/\mathcal{T}$, induced by the T-isomorphism $\mathcal{T}$. We define the relaxed warded semantics $chase^{B_W}(D, \Sigma)$ as the set of all the class representatives of $\mathbf{Q}$, one for each equivalence class of $\mathbf{Q}$.

The next results argue for the boundedness of $chase^{B_W}(D, \Sigma)$ and the correctness of our approach to CQ answering.

Theorem 3.12. *Let $\mathbf{S}$ be a database schema and w the maximal arity of its predicates. Given a database D and a set of warded TGDs Σ, both defined for $\mathbf{S}$, let P be the set of pairs $\langle T, \boldsymbol{a} \rangle$ where T = **track**(**a**). There is a constant δ depending on $\mathbf{S}$, $dom(D)$ and w, such that if $|P| > \delta$, then P contains at least two T-isomorphic facts.*

Towards an implementation of $chase^H$ that uses our relaxed warded semantics, we redefine $chase^H(D, \Sigma)$ as $chase(chase^{B_W}(D, \Sigma_T), \Sigma_E)$.

Theorem 3.13. *Given a set $\Sigma = \Sigma_T \cup \Sigma_E$ of warded TGDs and harmless EGDs with respect to Σ, and a database D, if $D \cup \Sigma$ is satisfiable, for every BCQ Q, it holds $chase(D, \Sigma) \models Q$ iff $chase(chase^{B_W}(D, \Sigma_T), \Sigma_E) \models Q$.*

Observe that the number of steps needed to build $chase^H$ is polynomial in data complexity.

Theorem 3.14. *Given a set $\Sigma = \Sigma_T \cup \Sigma_E$ of warded TGDs and harmless EGDs with respect to Σ, and a database D, if $D \cup \Sigma$ is satisfiable, then building $chase^H(D, \Sigma)$ is in PTIME in data complexity.*

3.5 Harmless, Separable EGDs, and Datalog$^\pm$

Before introducing our practical reasoning procedure, let us briefly discuss the relationship between harmless and separable EGDs.

Theorem 3.15. *If a set of TGDs and EGDs $\Sigma = \Sigma_T \cup \Sigma_E$ is separable then Σ_E is harmless w.r.t. Σ (and not vice versa).*

Unlike separable, harmless EGDs do not limit the possible causes of chase failure to inherent violations of Σ_E by D. On the contrary, a violation of Σ_E may depend on facts generated by the TGDs. The following example highlights this aspect.

Example 3.16. Consider again the formulation in Example 3.2, and let D={Node(a), Node(b), Node(c), Node(d), Node(e), Edge(a, b), Edge(b, c), Edge(c, d), Edge(d, e), CC(a, k_1), CC(d, k_2)}.

Nodes a and d are assigned to identifiers $CC(a, k_1)$ and $CC(d, k_2)$. The application of η propagates k_1 to all the nodes belonging to the same connected component generating the facts $CC(b, k_1)$,$CC(c, k_1)$, $CC(d, k_1)$. Applying η to $CC(d, k_1)$, $Edge(d, e)$, $CC(e, k_2)$ causes a hard violation (i.e., chase failure) since η tries to enforce $k_1 = k_2$. In this case, the rule set is not separable, yet η is harmless w.r.t. Σ. In fact, while separability assumes EGDs are pre-validated against facts in D, a hard violation of harmless EGDs may also depend on the facts generated in the chase. As we witness in this case, the violation of η depends on the facts generated by σ_1 and by the replacement of the labelled nulls derived by the application of η.

Having shown that harmlessness generalizes separability, we take up our comparison with two further reflections. First, as we have seen for harmless EGDs, checking whether a set of EGDs is separable from a set of TGDs is also an undecidable task and only a sufficient syntactic condition (*non-conflicting sets*) has been provided in that context as well [28]. Second, although harmlessness and separability are both defined independently of the TGD fragment, let us analyze how their properties in fact depend on it. Thanks to the particularly restrictive notion of non-conflicting sets, in separable EGDs, the syntactic condition, decidability, and complexity results hold in combination with multiple fragments. In fact, when non-conflicting sets of EGDs interact with Linear or Guarded TGDs (which do not allow or limit joins, respectively), query answering is decidable and in PTIME [25]. In spite of the less restrictive constraints posed by harmless EGDs, the syntactic condition (safe taintedness) is also independent of the fragment. Moreover, for the entire class of harmless EGDs, we have proven decidability and data complexity results for warded TGDs, hence with a high level of generality and broad applicability.

Beyond this, although the notion of harmlessness is less restrictive than separability, we believe that decidability and data complexity results can be extended to other Datalog data-tractable fragments that do not have a containment relationship with warded TGDs, such as Shy, Guarded, and Weakly Sticky [57]. Intuitively, as by Definition 3.1, independently of the TGD fragment, the application of harmless EGDs in the chase procedure does not trigger any rules that would not be triggered by the sole TGDs, harmless EGDs should neither affect termination nor increase data complexity.

3.6 T-Isomorphism Termination Strategy

The results of Section 3.4 suggest a practical procedure to reason under warded TGDs and harmless EGDs.

Along the lines of Theorem 3.13, in order to answer a query Q in the presence of TGDs and EGDs, we can construct the instance $chase^{H}(D, \Sigma)$ in two steps: (i) we build $chase^{B_W}(D, \Sigma)$ by activating a TGD chase step only if it produces facts that are not T-isomorphic to facts already generated (i.e., relaxed warded semantics); (ii) we then activate EGD chase steps to fixpoint over $chase^{B_W}(D, \Sigma)$.

We provide an efficient algorithm for the first step of such procedure, which we name *T-isomorphism termination strategy.*

Algorithm Description. Algorithm 1 determines whether a TGD chase step must be activated by invoking Algorithm 2, which checks whether a fact that is going to be generated is T-isomorphic to an existing one. The latter relies on the following two data structures.

(1) The **fact structure** is a structured representation of a fact **a**, with three distinct fields: (i) *generating_rule*, the kind of rule of Σ_T (linear / warded join / non-linear join) that generated **a**; (ii) *w_parent*, the parent fact in the warded forest from which **a** is generated; (iii) *w_track*, the ultimate root of the connected component containing **a** in the warded forest.
(2) The **track structure** T stores the nodes of the warded forest incrementally built during the TGD chase, grouping them by the track (root) of the connected component of the warded forest to which each node (i.e., fact) belongs. T is a dictionary of sets of facts. More precisely, each element T[**a**.w_track] represents the set of facts of the tree (i.e., connected component) rooted in **a**.w_track in the warded forest.

Algorithm 1 The Algorithm for $chase^{B_w}(D, \Sigma)$.

```
1:  function CHASE($D, \Sigma$)
2:      for all a ∈ D do
3:          a.w_track = a                                   ▷ input facts are tracks
4:      for all $\sigma \in \Sigma_T$ and x to which $\sigma$ applies do       ▷ for all TGDs
5:          if CHECK_TERMINATION($\sigma(\mathbf{x})$) then
6:              $D = D \cup \{\sigma(\mathbf{x})\}$
```

Algorithm 2 T-isomorphism Termination Strategy.

```
1:  function CHECK_TERMINATION(a)
2:      if a.generating_rule == {LINEAR or WARDED} then
3:          a.w_track = a.w_parent.w_track
4:          if ∃g ∈ T[(a.w_track)] s.t. a isomorphic to g then
5:              return false                           ▷ T-isomorphism found
6:          else
7:              T[a.w_track].append(a)
8:              return true
9:      else if a ∉ T then              ▷ other non-linear generating rules
10:         a.w_track = a
11:         T[a.w_track].append(a)                       ▷ new root addition
12:         return true
13:     else                                       ▷ the new tree is redundant
14:         return false
```

The algorithm assumes, without loss of generality, that: the TGDs at hand are warded, the EGDs are harmless and the existential quantification appears only in linear TGDs, as a rewriting to this form is always possible [14]. Linear and warded join TGDs produce facts **a** (line 2), for which, in the base case, T-isomorphism checks must be performed. In this case, the track of **a** is inherited from its direct parent (line 3). The track structure restricts the isomorphic check to the local connected component of the warded forest (line 8-10), featuring a form of *local detection.* If a T-isomorphic fact is found (line 4), the algorithm blocks the generation of **a**. Facts derived from non-linear join TGDs are the roots of new trees (connected components) of the warded forest (line 11). New trees are generated unless their root is already present in the track structure T (line 9). As we assume that non-linear join TGDs do not have existential quantification and do not propagate labelled nulls, their possible generation is efficiently checked as set containment of ground facts.

4 SYSTEM ARCHITECTURE

We describe the implementation of harmless EGDs in a dedicated processing module in the Vadalog system [13], which exploits the T-isomorphism termination strategy algorithm to enable the termination of the reasoning process while upholding correctness.

The Vadalog system is a state-of-the-art reasoner whose core language is based on Warded Datalog$^{\pm}$. Among the many features, it natively supports ontological query answering, probabilistic reasoning, numeric computations (i.e., standard and monotonic aggregations). Let us briefly describe the Vadalog system architecture.

System Architecture. Given a set of rules Σ and a CQ $Q : q(\mathbf{x}) \leftarrow \boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$, an active pipeline is compiled out of them, where atoms correspond to filters, connected by pipes that denote the input-output transformations applied by the rules. Data flow from sources filters, the extensional atoms, to the target, the query, undergoing the transformations (e.g., selections, projections, joins, value inventions) performed by the rules. Finally, the facts returned by the output filter are those composing the CQ answer. This process is implemented by four dedicated architectural components.

(1) The *CQ processor* rewrites Q (which is a WCQ) into an atomic query as follows: the set of rules Σ is updated as $\Sigma = \Sigma \cup \{\rho_Q\}$, where $\rho_Q = \nu(\mathbf{x}) \leftarrow \boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$ and ν is an invented atom. Then Q is rewritten into the atomic query $\hat{Q} : q(\mathbf{x}) \leftarrow \nu(\mathbf{x})$.
(2) The *logic optimizer* performs rewritings and syntactic condition checks. It checks the safe taintedness property; if satisfied, it rewrites the EGDs into (i) a TGD with the same body and the head composed of a new artificial atom; (ii) an EGD whose body is the artificial atom and whose head is the original EGD head. This simplifies the management of the operations required by the body of the EGDs (e.g. joins and selection).
(3) The *logic compiler* transforms TGDs and EGDs into a reasoning access plan: it produces a pipeline in the form of a predicate graph where each node (filter) represents an atom and there is an edge (pipe) from nodes m to n if there is a rule with m in the body and n in the head. Extensional atoms are mapped into the pipeline sources and $q(\mathbf{x})$ corresponds to the output filter.
(4) The *query compiler* converts the logic pipeline into a *reasoning query plan*, where the nodes are translated into active *data scans*, connected by intermediate buffers. In our architecture, we have four different types of scan: a linear scan for linear TGDs, a join scan for join TGDs, an EGD scan for harmless EGDs, and an output scan for the query. Note that, with the rewriting

technique in step (1), a CQ is implemented by just annotating the $q(\mathbf{x})$ filter as "output" and injecting a join rule into Σ.

Execution Model. The reasoning process follows a *pull-based* (query-driven) approach, where each filter (i.e., scan) reads facts from the respective parent. Such implementation is a generalization of the *volcano iterator model* [42] and allows to activate only the chase steps required to answer a query without generating the whole output of the chase procedure. The pull action starts from the output scan (i.e., corresponding to the CQ atom $q(\mathbf{x})$), which asks for output facts and propagates the request down to its predecessors, triggering the invocations along the pipeline searching for available facts. Scans interact with each other using primitives `open()`, `next()`, `get()`, `close()`, which respectively open the parent stream, ask for the presence of a fact to fetch, obtain it, and close the communication.

The pipeline in Figure 2 is associated with an alternative formulation of the UST-CONN problem, as follows.

Example 4.1. The UST-CONN problem is here modeled by the following set of TGDs and harmless EGDs.

$$Edge(x, y) \rightarrow \exists z\, Conn(x, y, z) \qquad (\sigma_1)$$

$$Edge(x, y) \rightarrow \exists z\, Conn(y, x, z) \qquad (\sigma_2)$$

$$Conn(x, y, z_1), Conn(y, w, z_2) \rightarrow z_1 = z_2 \qquad (\eta)$$

We assign each edge to a connected component (σ_1 and σ_2). We reverse the edges to support undirected graphs (σ_2). Edges sharing a node are assigned to the same connected component (η).

$$q(x, y) \leftarrow Conn(x, \_, z), Conn(y, \_, z) \qquad (Q)$$

By the query Q we ask whether the nodes x and y belong to the same connected component, i.e., are mutually reachable.

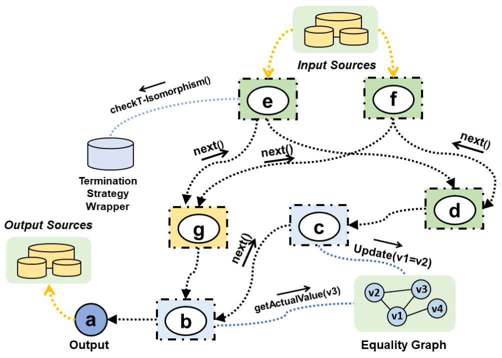

Figure 2: The execution pipeline of Example 4.1.

4.1 Harmless EGDs Module

Let us consider Figure 2, where scans e and f refer to the linear TGDs σ_1 and σ_2, c and d to the EGD η. The query Q is represented by the output scan g, whereas b is the *postprocessing egd scan*.

The application of an EGD updates the labelled nulls identity appearing in the facts produced by TGDs. The EGD module limits such updates to the facts that are candidates to answer a specific query. The correctness of such approach is guaranteed as: (i) since the EGDs are harmless, an EGD chase step never triggers other TGDs or EGDs, and (ii) the identity of labelled nulls in Warded Datalog$^{\pm}$ is not relevant to activate TGD chase steps (due to harmful join elimination [9]), i.e., does not trigger any further scans.

Our implementation relies on the postprocessing egd scan (b), the sink node of the pipeline. It retrieves all the facts involved in the query (g), replacing the labelled nulls in the facts that are candidate to answer the query. It is also connected to the EGD scans (c), that apply EGD chase steps. The egd scans use an ad-hoc data structure, named *equality structure*, that memorizes the assignments.

The postprocessing scan triggers all EGD scans until saturation. At every invocation, the EGD scans enforce a new equality between values and update the equality structure. When the EGD scans are saturated, the postprocessing scan triggers the output scan, which propagates the request to the underlying TGD scan and assigns the labelled nulls in the retrieved facts. Whenever a TGD scan is triggered, a T-isomorphism wrapper executes Algorithm 2 to control termination. We implemented two types of equality structures.

Equality Graph. The nodes store labelled nulls and constants. When an equality between values is enforced by the EGDs, an undirected edge is created between them. A connected component contains all the labelled nulls that are assigned to the same value.

Equality Hash Table. Each row of the hash table is of the type $v \rightarrow e$, where v is a labelled null and e is the assigned value. To save memory, our implementation shares value objects among nulls and just stores references [38].

scenarios	L/⋈ TGDs	L/⋈ recur	L/⋈ EGDs	∃ TGDs	CQs
SynthA	40/50	10/20	10/20	50	51
SynthB	70/30	20/10	20/10	40	41
SynthC	70/50	15/15	10/10	30	31
SynthD	95/25	0/5	0/25	50	51
SynthE	115/30	5/0	25/0	25	26
SynthF	80/45	0/0	15/15	45	46

Figure 3: iWarded **parameters of the synthetic scenarios.**

5 EXPERIMENTAL EVALUATION

In Section 5.1 we investigate the impact of specific properties of warded TGDs. In Section 5.2 we compare against other systems supporting EGDs. In Section 5.3 we demonstrate the effectiveness of harmless EGDs for graph traversal problems. In Section 5.4 we validate our system with real-world scenarios that can be handled with harmless EGDs. In Section 5.5 we compare our equality structures.

Test Setup. The Vadalog system was invoked via its REST interface. All external input and output sources have been organized in CSV files. We ran each experiment 10 times, averaging the elapsed times.

Hardware Configuration. We used a cloud instance of the Vadalog system, running Ubuntu v18 in a Linux machine with six physical 1.9 GHz Xeon v3 cores, 16 GB of RAM, and a 512 GB SSD.

5.1 iWarded: Synthetic Scenarios

To study the impact of the properties of warded TGDs and harmless EGDs on the performance of Vadalog, we used iWarded [10], a generator of Warded benchmarks. It controls the fragment internals such as the number of linear, non-linear join and warded join TGDs, the recursion length of the TGDs, the presence of existential quantification, the number of tainted positions, and the number of harmless EGDs. It is worth mentioning that the recursion length of

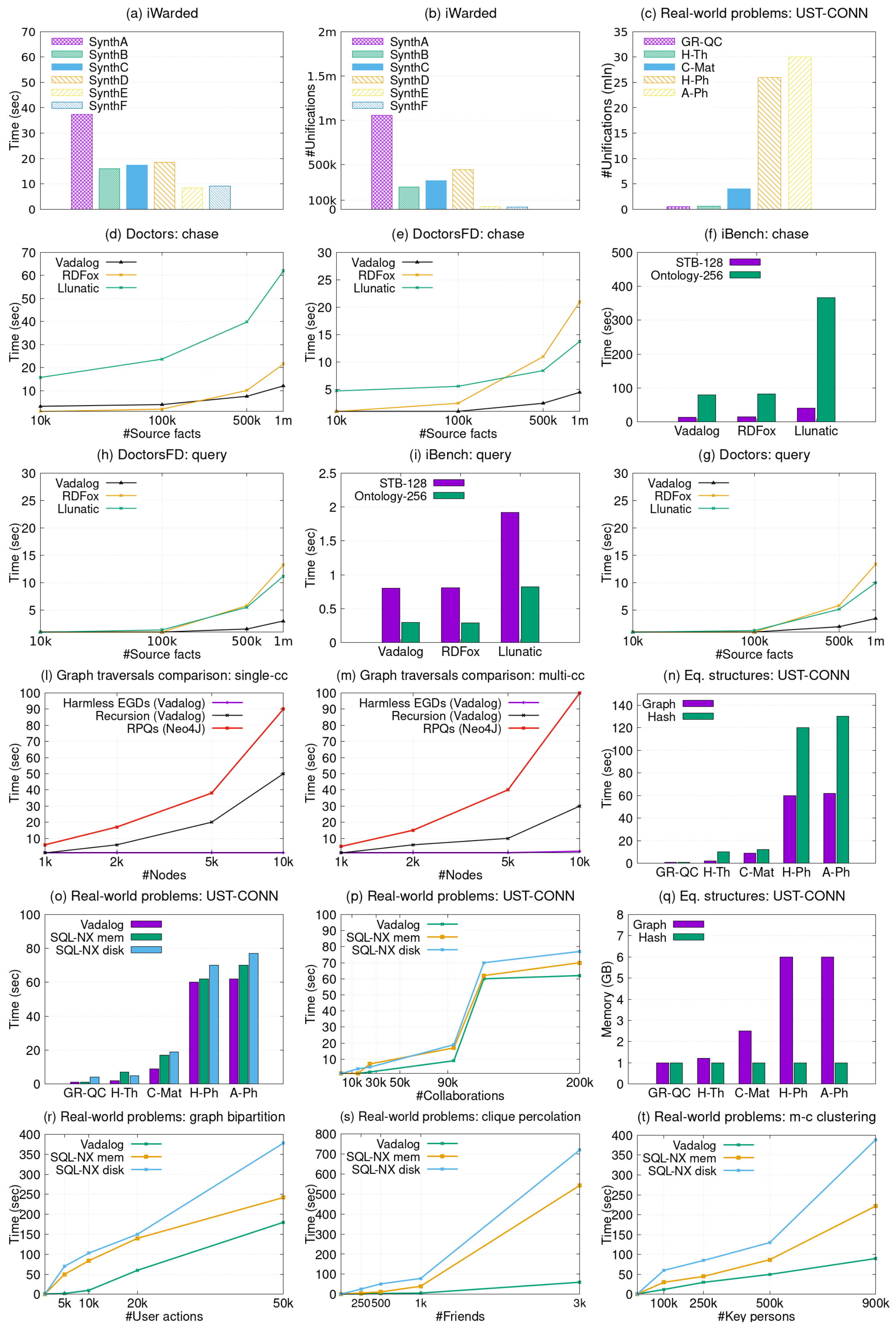

Figure 4: Reasoning statistics for the experimental evaluation.

```
conn(X,Y,Z) :- edge(X,Y).
Z1 = Z2 :- conn(X,Y,Z1), conn(Y,Z,Z2).
Q: q(X,Y) :- conn(X,_,Z), conn(Y,_,Z).
```

(i)

```
conn(X,Y) :- edge(X,Y).
conn(X,Y) :- conn(X,Z), edge(Z,Y).
Q: q(X,Y) :- conn(X,Y).
```

(ii)

```
CALL gds.alpha.allShortestPaths
YIELD sourceNodeId,targetNodeId
MATCH (source:NODE)
WHERE id(source)=sourceNodeId
MATCH (target:NODE)
WHERE id(target)=targetNodeId
RETURN source,target
```

(iii)

Figure 5: Three alternative formulations of a query for the UST-CONN problem: (i) Harmless EGDs in Vadalog; (ii) transitive closure in Vadalog; (ii) Cypher query expressed with Neo4j 4.0 Community Edition, graph data science (GDS) library [1, 64].

the TGDs is one of the main indicators for measuring the complexity of a program. Intuitively, the complexity grows with the number of interacting predicates in each recursive round, that determine the number of chase steps triggered by each set of mutually recursive TGDs in Σ_T. The input data have also been generated with iWarded, which guarantees uniform distributions.

Description of the Scenarios. We generated six different scenarios (Figure 3), each composed of 200 rules, distributed among TGDs and EGDs. Each scenario comprises about 20k input facts and uses a different configuration of the number of linear/join (L/$\bowtie$) TGDs, recursive TGDs, EGDs, and TGDs with existentials. We evaluated each scenario with different conjunctive queries, for a total of 246 CQs. Our queries have been generated by iWarded and then manually curated to incorporate most of the possible joins between the intensional atoms of the target schema, which also motivates the different number of queries tested for each scenario. *SynthA* and *SynthD* have the highest number of join EGDs; *SynthA*, *SynthB* and *SynthC* present many recursive TGDs; *SynthA* has a greater number of L EGDs w.r.t. the number of join EGDs; *SynthD* and *SynthE* are symmetric in the number of EGDs and recursive TGDs; *SynthF* has a balanced number of join EGDs and L EGDs and no recursion.

Discussion of the Results. For each scenario, Figure 4(a) reports the overall execution time, which is computed as the sum of average chase time and average query time. The marginal impact of the CQs on times is not evaluated here but specifically addressed in the chase-related scenarios in Section 5.2. The high number of join TGDs and join EGDs and the extensive use of recursion impact on the execution times. The worst observed performance (*SynthA*) depends on two facts: the production of many labelled nulls in recursive steps activates many EGD steps and overloads the equality structure; the presence of many harmful joins fragments the warded forest and produces many isolated trees. As T-isomorphism prunes isomorphic facts in the same connected components, many coexist in different trees. Scenarios *SynthB*, *SynthC* and *SynthD* confirmed that the presence of many join EGDs, harmful joins and recursive TGDs highly affects the performance. However, such scenarios have fewer recursive TGDs than *SynthA*, for whose execution required more than 20 seconds. In *SynthE* and *SynthF*, Vadalog shows the best performance thanks to the absence of harmful joins and recursive TGDs. In Figure 4(b) we show that the number of equalities enforced by the EGDs is proportional to the execution times.

5.2 Related Chase-based Tools

All existing chase-based tools supporting EGDs allow a limited form of interaction between TGDs and EGDs and many of them rely on weak acyclicity [35, 36, 54], with limited use of existential quantification with recursive TGDs. Other chase-based tools adopt broader TGD fragments (e.g., *Shy* [50]), yet either do not support EGDs or rely on separable EGDs in the form of functional dependencies (FDs). To the best of our knowledge, Vadalog is the only system that supports a richer interaction between TGDs and EGDs.

Systems Tested. We then limit the expressive power of Vadalog and consider only the scenarios that can be modeled via weakly acyclic sets of TGDs and EGDs, mostly data exchange tasks [36]. Data exchange consists in transforming an instance of a source schema into an instance of a target schema via source-to-target (s-t) TGDs and target TGDs and EGDs. We look at the systems and scenarios of ChaseBench [15], a comprehensive chase benchmark. Among the systems analyzed, we select LLunatic [39, 40] and RDFox [58], which support EGDs with weak acyclicity [19, 39]. We excluded other systems as they either do not support EGDs (Graal [8] and DLV [50, 51]), or support only FDs (ChaseFUN [21]), or did not terminate in three hours (PDQ [17, 18] and DEMo [61]).

LLunatic is an open-source data exchange tool that can handle s-t TGDs, target TGDs, and EGDs and compute certain query answering on the target schema. It runs on top of PostgreSQL [15] and transforms the dependencies into a SQL script that materializes all the target relations, before executing the query.

RDFox is a high-performance RAM-based Datalog engine which implements a parallel variant of the seminaive algorithm [15, 40].

Description of the Scenarios. We pick the ChaseBench scenarios with a significant presence of EGDs. Specifically, we consider two scenarios from IBench, a tool for generating sets of TGDs and EGDs [5]. In particular, STB-128 is a famous data mapping scenario composed of 128 s-t TGDs, 39 target TGDs and 193 FDs with 150k source instances; ONT-256 is a scenario composed of 256 s-t TGDs, 273 target TGDs and 923 FDs with 1m source instances. For STB-128 and ONT-256 we considered 20 CQs. Our benchmark also comprises *Doctors* and *DoctorsFD*, two famous data integration tasks from the schema mapping literature [56]. *Doctors* is composed of 5 s-t TGDs, two EGDs involving more than one relation in the body and 8 FDs; *DoctorsFD* is composed of 5 s-t TGDs and 8 FDs. In both scenarios we used source instances of 10k, 100k, 500k and 1m and we considered 9 CQs. We remark that all the scenarios involve harmless EGDs, in fact, as the reader can check [16], they all satisfy the safe taintedness condition (Section 3.2). The execution times include: (1) Load the dependencies from TXT file and the source instance from CSV files; (2) Run the chase to generate the target instance, evaluating chase time; (3) Evaluate query time. Step (3) is straightforward in LLunatic and RDFox because these systems adopt a materialization approach in which first the target instance is computed and then the CQs are executed. Conversely, Vadalog adopts a comprehensive pipeline which is compiled from both Σ and the rule ρ_Q, representing the CQ Q, and then executed. We measured times in step (2) by removing ρ_Q from Σ before compiling the pipeline; then, we stored into main memory the chase result and, for step (3), we evaluated a simple pipeline compiled only from ρ_Q. We repeated this process for every CQ of the setting and averaged the elapsed times.

Scenarios	Nodes	Edges	Nodes in the largest WCC	Edges in the largest WCC	Edge density in the largest SCC
HEP-PH	12008	118521	11204	117649	10.5
HEP-TH	9877	25998	8638	24827	2.87
GR-QC	5242	14496	4158	13428	3.22
COND-MAT	23133	93497	21363	91342	4.27
ASTRO-PH	18772	198110	17903	197031	11

Figure 6: Statistics of the five collaboration graphs.

Discussion of the Results. LLUNATIC and RDFox implement the chase under weakly acyclic sets of TGDs and apply the EGDs by retrieving and deleting all the affected facts from the materialized relations, apply the EGDs over such facts and insert back the updated facts. These operations require massive updates and are more expensive in an RDBMS-based than a RAM-based system [15]. These systems typically interleave the application of TGDs and EGDs, according to the standard chase procedure. As a result, multiple massive updates caused by EGDs tend to hamper performance. Instead, Vadalog applies the EGDs only once, enforcing the equalities between labelled nulls in a streaming fashion and storing them into the equality structure. Then, it updates only the facts that potentially contribute to the query answer. Results are reported as 2 groups of 3 charts each: Figures 4(d,e,f) show the average chase time for the *Doctors*, *DoctorsFD*, and the iBENCH scenarios (STB-125 and ONT-256). In Vadalog, the execution of the CQs is just the final part of the reasoning pipeline evaluation, and so the overall elapsed time mostly benefits from an efficient implementation of Σ. Figures 4(g,h,i) singles out the respective average query times, which nevertheless confirm very competitive performance for CQs.

Regarding chase time, in the *Doctors* scenario, LLUNATIC is outperformed by Vadalog and RDFox. RDFox is slightly faster than Vadalog on smaller input, whereas Vadalog scales better on large input (e.g., 500k and 1M facts). In *DoctorsFD*, Vadalog and RDFox have a similar trend until 100k facts. For larger instances, Vadalog outperforms both LLUNATIC and RDFox, while LLUNATIC behaves better than RDFox. On STB-125 and ONT-256 Vadalog and RDFox have a similar trend, with Vadalog being 2 and 3 seconds faster on STB-125 and ONT-256, respectively. In general, LLUNATIC performance is hampered by the cost of frequent I/O. As for query times, Vadalog outperforms RDFox and LLUNATIC by 6 seconds in *Doctors* and *DoctorsFD*. For the iBench scenarios, the times are similar to RDFox and slightly better than LLUNATIC.

5.3 Graph Traversals Comparison

We show how inherently recursive cases, like graph traversals, can be modeled with non-recursive TGDs thanks to harmless EGDs. We select again the UST-CONN problem and experiment three approaches to assess connectivity (Figure 5): (i) labelled null unification power of the EGDs in Vadalog, (ii) Datalog transitive closure in Vadalog, and (iii) *Regular Path Queries (RPQs)* in Neo4J.

Description of the Scenarios. We generated two graph topologies, the first with a single connected component (Figure 4(l)), the second with multi-connected components (Figure 4(m)). We test the approaches over four sparse graphs of the same topology with increasing sizes (1k, 2k, 5k, 10k nodes).

Discussion of the Results. Figure 4(l) and Figure 4(m) show that the harmless EGDs trend is sublinear. The recursive approach is outperformed by the EGDs technique. In fact, EGDs do not need all the previously derived facts, but can rely on a single join operation between the *Conn* relation and on the unification power of labelled nulls. Neo4j is outperformed by both recursion and EGD approaches, due to the inability of that system to deal with high number of source and target nodes in traversals based on adjacency lists [64].

5.4 Validation on Real-World Scenarios

The goal of this section is twofold: on one hand, we focus on real-world scenarios, with special attention to the recursive ones, that can be modeled with harmless EGDs, while being impossible or laborious to capture and inefficient to evaluate with the mere use of TGDs; on the other, we validate our EGD architecture by comparing the Vadalog system against ad-hoc implementations.

For the ad-hoc implementations, we adopt a materialization approach based on *SQLite* [46], and the graph libraries *networkX* [45]. We name such implementations *SQLite-networkX* (SQL-NX) [1]. Essentially, we used a SQL script that materializes the predicates created by the TGD applications and incrementally updates them to perform the EGD chase steps. Unification is executed with the equality graph approach (see Section 4.1), implemented with NetworkX algorithms. We use an in-memory and on-disk SQLite instance.

We now introduce four reasoning scenarios for real-world settings where harmless EGDs are effectively applicable.

Scenario 1: UST-CONN. We consider the UST-CONN problem (Example 4.1). We select as input five real-world public graphs of collaboration networks in the scientific domain [52], all having a huge connected component. Figure 6 details the five datasets and Figures 4(c,o,p) show the results of the experiments.

Scenario 2: Graph Bipartition. In the context of online platforms, we want to study the interactions between the involved domain entities, to check that specific integrity constraints are satisfied. Specifically, in the dataset about the popular MOOC online platform [48], the interactions between users and courses are modeled as a connected graph, where the nodes represent users or courses and the edges are the actions (for example "enrolment").

Example 5.1. The aim of the setting is checking the integrity constraints that enforce that actions are always taken by users and regard courses. This is the same as checking that the graph is bipartite.

$$\mathit{UserOrCourse}(x) \rightarrow \exists y\, \mathit{Part}(x,y) \quad (\sigma_1)$$

$$\mathit{Part}(x,z_1), \mathit{Action}(x,w), \mathit{Action}(w,y), \mathit{Part}(y,z_2) \rightarrow z_1 = z_2 \quad (\eta)$$

The EGD η merges the partitions, initially one for each node (σ), that are linked by a sequence of two actions.

$$q \leftarrow \mathit{Part}(x,z), \mathit{Action}(x,y), \mathit{Part}(y,z) \qquad (Q)$$

Finally, Q checks whether the graph is bipartite (q evaluates to false).

[1]The SQL-NX tool is available at https://github.com/Davben93/egd-experiments

For the tests, we considered connected subgraphs of growing size (Figure 4(r)) and measured the elapsed reasoning time.

Scenario 3: Clique Percolation. A popular technique to detect communities in social network graphs is the clique percolation method [60]. It builds communities from k-cliques (for simplicity, we will adopt k=3). Two k-cliques are considered adjacent if they share k-1 nodes. A community is defined as the maximal union of k-cliques that are mutually reachable via adjacent k-cliques.

Example 5.2. The goal of the setting is finding all the pairs of friends in the same community, considering a portion of the Facebook friendship network [55].

$$Edge(x,y), Edge(y,z), Edge(z,x) \rightarrow \exists c\, Clique(c,x,y,z) \quad (\sigma_1)$$

$$Clique(c,x,y,z) \rightarrow Community(c,x), Community(c,y), Community(c,z) \quad (\sigma_2)$$

$$Clique(c_1,x,y,\_), Clique(c_2,x,y,\_) \rightarrow c_1 = c_2 \quad (\eta_1)$$

$$Clique(c_1,\_,x,y), Clique(c_2,\_,x,y) \rightarrow c_1 = c_2 \quad (\eta_2)$$

$$Clique(c_1,x,\_,y), Clique(c_2,x,\_,y) \rightarrow c_1 = c_2 \quad (\eta_3)$$

Each clique of friends is identified and assigned to a different label representing a community (σ_1). We extract people and their corresponding communities (σ_2). Two communities are merged whenever two cliques of friends are adjacent (η_1, η_2, η_3).

$$q(x,y) \leftarrow Community(z,x), Community(z,y) \quad (Q)$$

Finally, Q extracts the pairs of friends in the same community.

We considered subnetworks of increasing size (Figure 4(s)).

Scenario 4: Multi-criteria Clustering. We consider the graph of the Italian companies used by the Bank of Italy to solve many financial tasks such as analysis of company control, prevention of company takeovers, and anti-money laundering [44]. The graph contains data about person-company and company-company shareholding relationships, counting 6.977M nodes and 6.252M edges.

Example 5.3. We want to cluster individuals according to multiple alternative criteria such as: being Persons of Significant Control (PSC) of the same company; being PSCs of a company related by a control relationship; being PSCs of companies of the same corporate group.

$$Person(p), Own(p,c,w), w > 0.25 \rightarrow PSC(p,c) \quad (\sigma_1)$$

$$PSC(p,c) \rightarrow \exists g\, KP(p,c,g) \quad (\sigma_2)$$

$$KP(\_,c,g_1), KP(\_,c,g_2) \rightarrow g_1 = g_2 \quad (\eta_1)$$

$$KP(\_,c_1,g_1), Control(c_1,c_2), KP(\_,c_2,g_2) \rightarrow g_1 = g_2 \quad (\eta_2)$$

$$KP(\_,c_1,g_1), CG(c_1,c_2), KP(\_,c_2,g_2) \rightarrow g_1 = g_2 \quad (\eta_3)$$

A shareholder p is a PSC of a company c if she owns more than 25% of the shares of c (σ_1). Every PSC is a key person for a group of companies g (σ_2). Key persons of the same company are assigned to the same group (η_1). If a company c1 controls a company c2, then their key persons are in the same group (η_2). If c1 and c2 are companies of the same corporate group (CG), they share the same key persons (η_3).

$$q(c_1,c_2) \leftarrow KP(p_1,c_1,g), KP(p_1,c_2,g), p_1 \neq p_2 \quad (Q_1)$$

$$q(p_1,p_2) \leftarrow KP(p_1,\_,g), KP(p_2,\_,g), p_1 \neq p_2 \quad (Q_2)$$

The CQ Q_1 computes the pair of companies which share the same group of key persons. Conversely, the CQ Q_2 identifies the linked key persons, those who are part of the same group of companies.

Our experiments consider an increasing size for *KP* (Figure 4(t)).

Discussion of the Results. As illustrated in Figure 4, Vadalog outperforms SQL-NX in all the scenarios. This is motivated by the ability of Vadalog to perform the basic operations (selection, join, handling existentials, enforcing equalities) in a streaming pipeline.

5.5 Comparison between Equality Structures

As shown by Figure 4(n), in the same settings of Scenario 1 in Section 5.4, the graph-based equality structure (Section 4.1) outperforms the hash-based one in terms of execution time as the input size grows. In particular, for *HEP-PH* and *ASTRO-PH*, the graph-based structure wins by two orders of magnitude. In fact, the hash equality structure requires frequent rearrangements to preserve the actual value for each new labelled null. On the other hand, it is efficient in terms of memory footprint (Figure 4(q)).

6 RELATED WORK

Most of the work on decidable variants of Datalog [6, 7, 24, 26, 30] that have been investigated only considers existential rules without equality constraints, also in the case of general notions like *model-faithful acyclicity* (MFA) [43]. General results have been achieved by *axiomatization approaches* such as *singulatisation* [54], not efficient in practice, or with *equality*-MFA [33], however, without simple sufficient conditions to detect non-termination. In the Datalog$^\pm$ context, there have been attempts to introduce restrictions to EGDs [25, 28, 32]. An initial intuition is that of *innocuous EGDs* [24], which enjoy the property that query answering is insensitive to them, provided that the chase does not fail. Given a set $\Sigma = \Sigma_T \cup \Sigma_E$, a chase application can simply ignore Σ_E, with the guarantee that all the facts needed for query answering will be entailed. This property, which is semantic and cannot be syntactically checked, is of scarce practical utility, as the adopted EGDs do not add expressive power to the TGDs. A more interesting notion is that of *separable EGDs* [23], which we have discussed at length. This concept was originally introduced in the context of inclusion dependencies and key dependencies [2], and has been reformulated under the notion of *EGD-stability* [23]: a set of TGDs and EGDs Σ is EGD-stable if, for every instance D, if D satisfies Σ_E, then the chase of D under Σ does not fail.

7 CONCLUSION

We introduced and implemented a new class of EGDs that can non-trivially interact with warded TGDs, without affecting decidability or tractability of the Warded fragment. The interplay of Warded Datalog$^\pm$ with harmless EGDs captures many practical tasks. Looking at extensions, in principle it seems possible to overcome the syntactic limitation of our fragment, namely, the inhibition to join variables affected by EGDs when they possibly bind to labelled nulls. However, for such an expanded fragment to be meaningful, it should be used in conjunction with TGDs where harmful joins are incorporated, beyond being mere syntactic sugar, like in the case of warded TGDs. That would likely require to give up at least PTIME data complexity, if at all possible, thus with limited practical applicability. Our studies will pursue such an investigation.

ACKNOWLEDGMENTS

The work on this paper was partially supported by the Vienna Science and Technology Fund (WWTF) grant VRG18-013.

REFERENCES

[1] 2022. The Neo4j Graph Data Science Library Manual v1.8. http://shorturl.at/gops [Online; 21-Sep-2022].

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases.* Addison-Wesley.

[3] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovasz, and Charles Rackoff. 1979. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symp. on Foundations of Computer Science.* 218–223.

[4] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. 2015. Design and Implementation of the LogicBlox System. In *SIGMOD.* 1371–1382.

[5] Patricia C. Arocena, Boris Glavic, Radu Ciucanu, and Renée J. Miller. 2015. The IBench Integration Metadata Generator. *VLDB* 9, 3 (2015), 108–119.

[6] Jean-François Baget, Michel Leclère, and Marie-Laure Mugnier. 2010. Walking the Decidability Line for Rules with Existential Variables. In *KR.*

[7] Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. 2011. Walking the Complexity Lines for Generalized Guarded Existential Rules. In *IJCAI.*

[8] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Swan Rocher, and Clément Sipieter. 2015. Graal: A Toolkit for Query Answering with Existential Rules. In *RuleML.*

[9] Teodoro Baldazzi, Luigi Bellomarini, Emanuel Sallinger, and Paolo Atzeni. 2021. Eliminating Harmful Joins in Warded Datalog+/-. In *International Joint Conference on Rules and Reasoning.* Springer, 267–275.

[10] Teodoro Baldazzi, Luigi Bellomarini, Emanuel Sallinger, and Paolo Atzeni. 2022. A Versatile Generator to Benchmark Warded Datalog+/- Reasoning (to appear). In *RuleML+RR (Lecture Notes in Computer Science).* Springer.

[11] Pablo Barceló and Reinhard Pichler (Eds.). 2012. *Datalog in Academia and Industry.* LNCS, Vol. 7494. Springer.

[12] Luigi Bellomarini, Davide Benedetto, Matteo Brandetti, and Emanuel Sallinger. 2022. Tech. Appendix. http://shorturl.at/agij2 [Online; 21-Sep-2022].

[13] Luigi Bellomarini, Davide Benedetto, Georg Gottlob, and Emanuel Sallinger. 2020. Vadalog: A modern architecture for automated reasoning with large knowledge graphs. *Information Systems* (2020), 101528.

[14] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. 2018. The Vadalog System: Datalog-based Reasoning for Knowledge Graphs. *PVLDB* 11, 9 (2018), 975–987.

[15] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the Chase. In *PODS.* 37–52.

[16] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. ChaseBench. https://github.com/dbunibas/chasebench. [Online; 21-Sep-2022].

[17] Michael Benedikt, Julien Leblay, and Efthymia Tsamoura. 2014. PDQ: Proof-Driven Query Answering over Web-Based Data. *VLDB* 7, 13 (2014), 1553–1556.

[18] Michael Benedikt, Julien Leblay, and Efthymia Tsamoura. 2015. Querying with Access Patterns and Integrity Constraints. *VLDB* 8, 6 (feb 2015), 690–701.

[19] Michael Benedikt, Boris Motik, and Efthymia Tsamoura. 2018. Goal-Driven Query Answering for Existential Rules with Equality. In *AAAI.* Article 215, 10 pages.

[20] Jens Bleiholder and Felix Naumann. 2009. Data fusion. *ACM computing surveys (CSUR)* 41, 1 (2009), 1–41.

[21] Angela Bonifati, Ioana Ileana, and Michele Linardi. 2016. Functional Dependencies Unleashed for Scalable Data Exchange. In *SSDBM.* Article 2, 12 pages.

[22] Max Bramer. 2007. *Clustering.* Springer.

[23] Andrea Calì, Marco Console, and Riccardo Frosini. 2013. Deep Separability of Ontological Constraints. *CoRR* abs/1312.5914 (2013).

[24] Andrea Calì, Georg Gottlob, and Michael Kifer. 2013. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.* 48 (2013), 115–174.

[25] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. 2009. A general datalog-based framework for tractable query answering over ontologies. In *PODS.* 77–86.

[26] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. 2010. Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. In *LICS.* 228–242.

[27] Andrea Calì, Georg Gottlob, Giorgio Orsi, and Andreas Pieris. 2012. *On the Interaction of Existential Rules and Equality Constraints in Ontology Querying.* Springer-Verlag, Berlin, Heidelberg, 117–133.

[28] Andrea Calì, Georg Gottlob, and Andreas Pieris. 2010. Advanced Processing for Ontological Queries. *Proc. VLDB Endow.* 3, 1 (2010), 554–565.

[29] Andrea Calì, Georg Gottlob, and Andreas Pieris. 2012. Ontological query answering under expressive Entity–Relationship schemata. *Information Systems* 37, 4 (2012), 320–335.

[30] Andrea Calì, Georg Gottlob, and Andreas Pieris. 2012. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence* 193 (2012), 87–128.

[31] Andrea Calì, Domenico Lembo, and Riccardo Rosati. 2003. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *SIGMOD.* 260–271.

[32] Andrea Calì and Andreas Pieris. 2011. On Equality-Generating Dependencies in Ontology Querying - Preliminary Report. In *AMW*, Vol. 749. CEUR-WS.org.

[33] David Carral and Jacopo Urbani. 2020. Checking Chase Termination over Ontologies of Existential Rules with Equality. In *AAAI.* AAAI Press, 2758–2765.

[34] Ashok Chandra and Moshe Vardi. 1985. The Implication Problem for Functional and Inclusion Dependencies is Undecidable. *SIAM J. Comput.* 14 (1985), 671–677.

[35] Alin Deutsch, Alan Nash, and Jeff Remmel. 2008. The Chase Revisited. In *SIGMOD.* 149–158.

[36] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336, 1 (2005), 89–124.

[37] Ivan P Fellegi and Alan B Sunter. 1969. A theory for record linkage. *J. Amer. Statist. Assoc.* 64, 328 (1969), 1183–1210.

[38] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education.

[39] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2014. Mapping and cleaning. In *ICDE.* 232–243.

[40] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2014. That's All Folks! LLUNATIC Goes Open Source. *PVLDB* 7, 13 (2014), 1565–1568.

[41] Georg Gottlob and Andreas Pieris. 2015. Beyond SPARQL under OWL 2 QL Entailment Regime: Rules to the Rescue. In *IJCAI.* 2999–3007.

[42] Goetz Graefe and William J McKenna. 1993. The volcano optimizer generator: Extensibility and efficient search. In *ICDE.* 209–218.

[43] Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. 2013. Acyclicity Notions for Existential Rules and Their Application to Query Answering in Ontologies. *J. Artif. Intell. Res.* 47 (2013), 741–808.

[44] Andrea Gulino, Stefano Ceri, Georg Gottlob, Emanuel Sallinger, and Luigi Bellomarini. 2021. Distributed Company Control in Company Shareholding Graphs. In *ICDE.* IEEE, 2637–2648.

[45] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.

[46] Richard D Hipp. 2020. SQLite. http://shorturl.at/cemJV [Online; 21-Sep-2022].

[47] Richard Hull and Masatoshi Yoshikawa. 1990. ILOG: Declarative Creation and Manipulation of Object Identifiers. In *VLDB.* 455–468.

[48] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *SIGKKD.* ACM.

[49] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. 2012. Efficiently Computable $Datalog^{\exists}$ Programs. In *KR* (Rome, Italy). 13–23.

[50] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. 2019. Fast Query Answering over Existential Rules. *ACM Trans. Comput. Logic* 20, 2, Article 12 (2019).

[51] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM Trans. Comput. Logic* 7, 3 (jul 2006), 499–562.

[52] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph Evolution: Densification and Shrinking Diameters. 1, 1 (2007).

[53] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. 1979. Testing Implications of Data Dependencies. *ACM Tran. on DB Systems* 4, 4 (1979), 455–468.

[54] Bruno Marnette. 2009. Generalized schema-mappings: from termination to tractability. In *PODS.* ACM, 13–22.

[55] Julian J. McAuley and Jure Leskovec. 2012. Learning to Discover Social Circles in Ego Networks. In *NIPS.* 548–556.

[56] Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2014. IQ-METER-an evaluation tool for data-transformation systems. In *ICDE.* 1218–1221.

[57] Mostafa Milani and Leopoldo E. Bertossi. 2016. Extending Weakly-Sticky Datalog$^{\pm}$: Query-Answering Tractability and Optimizations. In *RR (Lecture Notes in Computer Science)*, Vol. 9898. Springer, 128–143.

[58] Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. 2014. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In *AAAI.* 129–137.

[59] Walaa Eldin Moustafa, Vicky Papavasileiou, Ken Yocum, and Alin Deutsch. 2016. Datalography: Scaling datalog graph analytics on graph processing systems. In *BigData.* IEEE, 56–65.

[60] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (2005), 814–818.

[61] Reinhard Pichler and Vadim Savenkov. 2009. DEMo: Data Exchange Modeling Tool. *VLDB* 2, 2 (2009), 1606–1609.

[62] Omer Reingold. 2005. Undirected ST-connectivity in log-space. *Electronic Colloquium on Computational Complexity - ECCC*, 376–385.

[63] Victor Vianu. 2021. Datalog Unchained. In *PODS.* ACM, 57–69.

[64] Jim Webber, Emil Eifrem, and Ian Robinson. 2013. *Graph databases.* O'Reilly Media, Incorporated.