Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning
Main Track

On the Relationship between Shy and Warded Datalog+/-

Teodoro Baldazzi[1] , **Luigi Bellomarini**[2] , **Marco Favorito**[2,3] , **Emanuel Sallinger**[4,5]
[1]Università Roma Tre, Italy
[2]Bank of Italy
[3]Università Sapienza, Italy
[4]TU Wien, Austria
[5]University of Oxford, UK
teodoro.baldazzi@uniroma3.it, {luigi.bellomarini,marco.favorito}@bancaditalia.it, sallinger@dbai.tuwien.ac.at

Abstract

Datalog$^\exists$ is the extension of Datalog with existential quantification. While its high expressive power, underpinned by a simple syntax and the support for full recursion, renders it particularly suitable for modern applications on knowledge graphs, query answering (QA) over such language is known to be undecidable in general. For this reason, different fragments have emerged, introducing syntactic limitations to Datalog$^\exists$ that strike a balance between its expressive power and the computational complexity of QA, to achieve decidability. In this short paper, we focus on two promising tractable candidates, namely Shy and Warded Datalog$^\pm$. Reacting to an explicit interest from the community, we shed light on the relationship between these fragments. Moreover, we carry out an experimental analysis of the systems implementing Shy and Warded, respectively DLV$^\exists$ and Vadalog.

1 Introduction

The last decade has witnessed a rising interest, both in academia and industry, towards querying and exploiting data in the form of *knowledge graphs* (KGs), modeled by combining extensional knowledge with ontological theories to infer intensional information. This led to the adoption of novel intelligent systems that perform *ontological reasoning* and *ontology-based query answering* (QA) tasks over KGs, employing powerful logic languages for knowledge representation (Krötzsch and Thost 2016).

Among the main requirements such languages must exhibit, a high expressive power is essential in modern applications on KGs, so as to model and reason on complex domains with full recursion and existential quantification (Bellomarini et al. 2017). At the same time, decidability and tractability of QA must be sustained, limiting the data complexity to a polynomial degree (Gottlob and Pieris 2015).

In this context, Datalog$^\exists$, the natural extension of Datalog with existential quantification in rule heads, became particularly relevant. Its semantics is specified in an operational way via the CHASE (Maier, Mendelzon, and Sagiv 1979), an algorithmic tool that takes as input a database D and a set Σ of rules, and modifies D by adding new tuples until Σ is satisfied. While this language encompasses both a high expressiveness and a simple syntax that enable powerful knowledge-modeling, QA over it is known to be undecidable in general (Calì, Gottlob, and Kifer 2013). For this reason, recent years have witnessed a rise of proposals for decidable classes of Datalog$^\exists$ in the literature (Calì, Gottlob, and Lukasiewicz 2012; Calì, Gottlob, and Pieris 2012; Baget, Leclère, and Mugnier 2010; Baget et al. 2011), defined by imposing proper syntactic limitations to strike a good balance between the expressive power of the language and the computational complexity of QA.

In this paper, we focus on two particularly promising languages, namely, *Shy Datalog*$^\exists$ (Leone et al. 2019) and *Warded Datalog*$^\pm$ (Gottlob and Pieris 2015), which have been independently introduced. Indeed, they both cover the requirements for knowledge representation, restricting Datalog$^\exists$, though with notably different constraints, and featuring PTIME data complexity for Boolean conjunctive QA (BCQA). Shy and Warded are employed in state-of-the-art reasoning systems. Specifically, Shy was introduced as part of the *parsimonious* class (Leone et al. 2012) and is adopted in the system *DLV*$^\exists$ (Leone et al. 2017). Likewise, Warded was recently introduced as fragment of the Datalog$^\pm$ family (Calì et al. 2010) and is implemented as logic core of the reasoner *Vadalog system* (Bellomarini et al. 2020). Both find many industrial applications in the financial, media intelligence, security, logistics, pricing domains, and more (Berger et al. 2019; Adrian et al. 2018).

Previous works have thoroughly discussed how these two languages individually compare to the other main decidable classes (Leone et al. 2019; Gottlob, Lukasiewicz, and Pieris 2014). However, to our surprise, determining the relationship between Shy and Warded is still an unexplored research topic, despite known to be recurring in the Datalog$^\exists$ academic and practitioner communities.

This paper aims at providing an answer to such question, by contributing the results summarized below. After a brief overview of Shy and Warded, we present a novel **theoretical analysis** of the relationship between the languages. From a syntactical point of view, we conclude that they intersect in a newly defined fragment, named **Protected Datalog**$^\pm$. Regarding semantics, we show that the CHASE procedures adopted by Shy and Warded are equivalent over Protected settings with respect to BCQA. To enrich the analysis with a more empirical perspective, we then illustrate an **experimental comparison** between DLV$^\exists$ and the Vadalog system on QA tasks over Protected settings.

Overview. This paper is organized as follows. In Section 2, we provide an overview of Shy and Warded. In Section 3, we discuss the relationship between them. In Section 4, we provide the experimental evaluation of DLV$^{\exists}$ and the Vadalog system. We draw our conclusions in Section 5.

2 Shy Datalog$^{\exists}$ and Warded Datalog$^{\pm}$

To guide our discussion, we briefly recall some relevant notions and provide an overview of the two fragments at issue. 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 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*. Given two conjunctions of atoms ς_1 and ς_2, we define a *homomorphism* from ς_1 to ς_2 as a mapping $h : C \cup N \cup V \to C \cup N \cup V$ such that $h(t) = t$ if $t \in C$, $h(t) \in C \cup N$ if $t \in N$ and if $a(t_1, \ldots, t_n)$ is an atom $\in \varsigma_1$, then $a(h(t_1), \ldots, h(t_n)) \in \varsigma_2$: ς_1 and ς_2 are *isomorphic* if h^{-1} is a homomorphism from ς_2 to ς_1.

A Datalog$^{\exists}$ program consists of a set of facts and *existential rules* $\forall\bar{x}\forall\bar{y}(\varphi(\bar{x},\bar{y})\to\exists\bar{z}\ \psi(\bar{x},\bar{z}))$, where φ (the *body*) and ψ (the *head*) are conjunctions of atoms. We omit $\forall$ and denote conjunction by comma. Let Σ be a set of Datalog$^{\exists}$ 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 an existentially quantified variable ($\exists$-*variable*) in i-th term or, (ii) there is a rule in Σ such that a universally quantified variable ($\forall$-*variable*) is only in affected body positions and in $p[i]$ in the head.

Shy Datalog$^{\exists}$. Let y be an $\exists$-variable in Σ. We define the position $p[i]$ as *invaded* by y if there is a rule $\rho \in \Sigma$ such that *head*$(\rho) = p(t_1, \ldots, t_k)$ and either (i) $t_i = y$ or, (ii) t_i is a $\forall$-variable that occurs in *body*(ρ) only in positions that are invaded by y. By such definition, if $p[i]$ is invaded, then it is affected, but not vice versa. Let $x \in \mathbf{X}$ be a variable in a conjunction of atoms $\varsigma_{[\mathbf{X}]}$. We say that x is *attacked* in $\varsigma_{[\mathbf{X}]}$ by y if x occurs in $\varsigma_{[\mathbf{X}]}$ only in positions invaded by y. If x is not attacked by any variable, x is *protected* in $\varsigma_{[\mathbf{X}]}$.

We define a set Σ as *Shy Datalog*$^{\exists}$ (or shy) if, for each rule $\sigma \in \Sigma$: (S1) if a variable x occurs in more than one body atom, then x is protected in *body*(σ); and, (S2) if two distinct $\forall$-variables are not protected in *body*(σ) but occur both in *head*(σ) and in two different body atoms, then they are not attacked by the same variable (Leone et al. 2019).

Warded Datalog$^{\pm}$. A $\forall$-variable x is *harmful*, wrt a rule ρ in Σ, if x appears only in affected positions in ρ, otherwise it is *harmless*. A (join) rule that contains a harmful (join) variable is a *harmful* (*join*) *rule*. If the harmful variable is in *head*(ρ), it is *dangerous* (Gottlob and Pieris 2015).

We define a set Σ as *Warded Datalog*$^{\pm}$ (or warded) if, for each rule $\sigma \in \Sigma$: (W1) all the dangerous variables appear in a single body atom, called *ward*; and, (W2) the ward only shares harmless variables with other atoms in the body.

Chase and Semantics. Chase-based procedures enforce the satisfaction of Σ over a database D ($\langle D,\Sigma\rangle$), incrementally expanding D into new instances I with facts derived from the application of the rules in $\langle D,\Sigma\rangle$, until Σ is satisfied ($chase(D, \Sigma)$). Such facts may contain labelled nulls as placeholders for the $\exists$-variables (Calì et al. 2010). We refer to its *oblivious* variant with $ochase$. Consider an instance $I' \supseteq I$. Given a rule $\sigma : \varphi(\bar{x},\bar{y})\to\exists\bar{z}\ \psi(\bar{x},\bar{z})) \in \Sigma$, a *chase step* $\langle\sigma,h\rangle$ is *applicable* to I' if there exists a homomorphism h that maps the atoms of $\varphi(\bar{x},\bar{y})$ to facts of I (i.e., $h(\varphi(\bar{x},\bar{y})) \subseteq I$). When the chase step is applicable, the atom $h'(\psi(\bar{x},\bar{z}))$ is added to I', where h' is obtained by extending h so that $h'(z_i) \in N$ is a fresh labelled null, $\forall\ z_i \in \bar{z}$.

However, in the presence of recursion, especially jointly with existential quantification, infinite labelled nulls could be generated in $ochase$, causing the procedure not to terminate and inhibiting the decidability of the QA task (Calì, Gottlob, and Lukasiewicz 2009). To practically achieve termination and decidability, shy and warded both employ variants of the $ochase$, based on *firing conditions* that limit the applicability of the chase steps. Specifically, shy adopts the so-called *parsimonious* chase ($pchase$): the chase step $\langle\sigma,h\rangle$ is applicable wrt $I' \supseteq I$ if, additionally, there is no homomorphism from *h(head(σ))* to I'. To cover decidability of CQ cases and preserve correctness of the evaluation, $pchase$ is extended into its variant *with resumption* ($pchase_r$) (Leone et al. 2019), which iteratively "resumes" the chase in the same state it was after termination but considering previous nulls as constants. Similarly, warded employs an *isomorphism-based* chase ($ichase$): the chase step $\langle\sigma,h\rangle$ is applicable wrt $I' \supseteq I$ if, additionally, there is no isomorphic embedding of *h(head(σ))* to I'. Here, decidability of CQA derives from (Bellomarini, Sallinger, and Gottlob 2018, Theorem 1, Theorem 2), as we shall see.

A *Boolean Conjunctive Query* (BCQ) is a first-order expression $q : \exists\mathbf{Y}\ \varsigma_{[\mathbf{X}\cup\mathbf{Y}]}$, where $\mathbf{X} \in C$. The answer of q over an instance I (namely, BCQA) is *true*, denoted by $I \models q$, if and only if there is a homomorphism h: $\mathbf{Y} \to C \cup N$ s.t. $h(\varsigma_{[\mathbf{X}\cup\mathbf{Y}]}) \subseteq I$. It holds that q is true over $chase(D, \Sigma)$, denoted by $chase(D, \Sigma) \models q$, if and only if $\langle D,\Sigma\rangle \models q$. We recall that the query output tuple problem (decision version of CQ evaluation) and BCQ evaluation are AC_0-reducible to each other (Calì, Gottlob, and Lukasiewicz 2012). Therefore, we only consider BCQ evaluation.

3 Relationship between Shy and Warded

We now have the means to discuss the relationship between shy and warded. Let us start by showing that the two fragments are not characterized by any form of syntactical containment (represented with the symbol $\not\subseteq$).

Proposition 1. shy $\not\subseteq$ warded

Proof. We prove the claim by showing that there exists a program that is shy but not warded. Let Σ be the following:

$$E_1(\mathrm{x}) \to \exists \mathrm{y}\ I_1(\mathrm{x}, \mathrm{y}) \quad (\alpha)$$

$$E_2(\mathrm{x}) \to \exists \mathrm{z}\ I_2(\mathrm{z}, \mathrm{x}) \quad (\beta)$$

$$I_1(\mathrm{x}, \mathrm{y}), I_2(\mathrm{z}, \mathrm{x}) \to I_3(\mathrm{x}, \mathrm{y}, \mathrm{z}) \quad (\rho)$$

Here, rules α and β are existential rules, both shy and warded, and introduce affectedness in positions $I_1[2]$ and $I_2[1]$, respectively. Rule ρ has a harmless join on x and propagates the harmful variables y and z to the head. Indeed, ρ is not warded, as y and z are dangerous and there is no ward (condition (W1) is not satisfied). However, it is shy, as the join variable x is protected (condition (S1)), whereas y and z are attacked by distinct variables, respectively y_α and z_β (condition (S2)). Therefore Σ is shy but it is not warded. ☐

Proposition 2. warded $\not\subseteq$ shy

Proof. We prove the claim by showing that there exists a program that is warded but not shy. Let Σ be the following:

$$E_1(\mathrm{x}) \rightarrow \exists \mathrm{y}\, I_1(\mathrm{x}, \mathrm{y}) \qquad (\alpha)$$

$$I_1(\mathrm{x}, \mathrm{y}), I_1(\mathrm{z}, \mathrm{y}) \rightarrow I_2(\mathrm{x}, \mathrm{z}) \qquad (\rho)$$

Here, rule α is an existential rule, both warded and shy, and introduces affectedness in position $I_1[2]$. Rule ρ contains a harmful join on y and propagates the harmless variables x and z to the head. Indeed, ρ is warded, as no dangerous variables occur in the rule. However, it is not shy, as the join variable y is attacked by y_α (condition (S1) is not satisfied). Therefore Σ is warded but it is not shy. ☐

Protected Datalog$^\pm$. To further explore the relationship between shy and warded, we first introduce the notion of *protected harmful* variable. Given a Datalog$^\exists$ set Σ, a $\forall$-variable x is protected harmful, with respect to a rule $\rho \in \Sigma$, if it appears in affected positions in ρ that are not invaded by the same $\exists$-variable: if the invading variable is the same, x is an *attacked harmful* variable. Without loss of generality (as more complex joins can be broken into steps (Bellomarini et al. 2020)), we define *protected harmful join rule* as a rule:

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

where A, B and C are atoms, $A[3]$ and $B[3]$ are positions invaded (and thus affected) by distinct $\exists$-variables, $x_1, x_2 \subseteq \overline{x}$, $y_1, y_2 \subseteq \overline{y}$ are disjoint tuples of harmless variables or constants and h is a protected harmful variable. By definition of protected variables and labelled nulls, the join on h only activates on constant values in the CHASE. If h is attacked, τ is an *attacked harmful join rule*.

We define a set Σ as *Protected Datalog$^\pm$* (or protected) if, for each rule $\sigma \in \Sigma$: (P1) σ does not contain attacked harmful joins; and, (P2) σ is warded (it satisfies (W1) and (W2)). With reference to the relationship between shy and warded, we first show that protected corresponds to the syntactical intersection of the two fragments (represented with $\cap$).

Theorem 1. warded $\cap$ shy $=$ protected.

Proof. We prove the equivalence by showing the containment in both directions of implication.

warded $\cap$ shy $\subseteq$ protected. Let Σ be a generic set of rules $\in$ warded $\cap$ shy. Then, Σ satisfies condition (P2) by definition. Regarding condition (P1), we proceed by contrapositive and show that $\neg(P1) \implies \neg(S1)$. Indeed, if Σ does not satisfy condition (P1), then there exists a rule $\sigma \in \Sigma$ with an attacked harmful join. However, this means that σ contains a variable that occurs in more than one body atom and it is not protected in *body*(σ) (condition $(S1)$ is not satisfied). Therefore, by contrapositive, $\Sigma \in$ protected.

protected $\subseteq$ warded $\cap$ shy. Let Σ be a generic set of rules $\in$ protected. By condition (P2), $\Sigma \in$ warded. Also, by condition (P1), Σ may only contain harmless joins and protected harmful joins, thus an attacked variable cannot occur in both body atoms by definition. Therefore, condition (S1) is satisfied. Finally, we proceed by contrapositive and show that $\neg(S2) \implies \neg(P2)$. Indeed, if Σ does not satisfy condition (S2), then there exists a rule $\sigma \in \Sigma$ with two dangerous attacked variables in distinct body atoms. However, this means that σ does not contain a ward, thus it is not warded (condition $(P2)$ is not satisfied). Therefore, by contrapositive, $\Sigma \in$ warded $\cap$ shy. This concludes the proof. ☐

Figure 1 illustrates the syntactic containment of these fragments, as well as their data complexity.

With reference to the semantic perspective of this analysis, let us first develop the following consideration.

Observation 1. *$pchase(D, \Sigma) \subseteq ichase(D, \Sigma)$, $\forall \Sigma \in$ Datalog$^\exists$, $\forall D$. This derives from the definition of pchase and ichase, as the applicability of their chase steps depends on fact homomorphism and fact isomorphism, respectively. In particular, whenever a pchase step $\langle \sigma, h \rangle$ is applicable with respect to $I' \supseteq I$, the absence of homomorphisms from $h(head(\sigma))$ to I' implies the absence of isomorphic embeddings of $h(head(\sigma))$ to I' (and not vice versa). The $ichase$ step is therefore applicable as well.*

Since, by Theorem 1, protected is a syntactical subset of warded, its data complexity is also PTIME, as shown in Figure 1. Moreover, we observe that reasoning with protected can adopt both $pchase_r$ and $ichase$. Based on these notions, we make an additional step in the comparative analysis of the fragments, stating that $pchase_r$ and $ichase$ are equivalent over protected settings, with respect to BCQA (i.e., a generic BCQ has the same answer).

Lemma 1. *Let $\Sigma \in$ shy, D a database and q a BCQ. Then, $ochase(D, \Sigma) \models q$ if and only if $pchase_r(D, \Sigma) \models q$.*

Proof. The result directly follows from (Leone et al. 2019, Theorem 4.9) for BCQA decidability over shy. ☐

Lemma 2. *Let $\Sigma \in$ warded, D a database and q a BCQ. Then, $ochase(D, \Sigma) \models q$ if and only if $ichase(D, \Sigma) \models q$.*

Proof. Completeness, namely $ochase(D, \Sigma) \models q$ implies $ichase(D, \Sigma) \models q$, directly follows from (Bellomarini, Sallinger, and Gottlob 2018, Theorem 1, Theorem 2), as chase subgraphs derived from isomorphic facts are isomorphic, thus irrelevant for BCQA. Soundness, namely $ichase(D, \Sigma) \models q$ implies $ochase(D, \Sigma) \models q$, holds because $ichase(\Sigma) \subseteq ochase(\Sigma)$ by definition. ☐

Theorem 2. *Let $\Sigma \in$ protected, D a database and q a BCQ. Then, $pchase_r(D, \Sigma) \models q$ if and only if $ichase(D, \Sigma) \models q$.*

Proof. By Theorem 1 and by definition of protected fragment, we know that $\Sigma \in$ shy and $\Sigma \in$ warded. Therefore, the result directly follows from Lemma 1 and Lemma 2. ☐

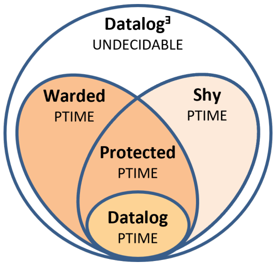

Figure 1: Syntactic containment of fragments and data complexity.

4 DLV$^{\exists}$ and Vadalog System

We integrate the analysis of shy and warded with an experimental comparison between their main state-of-the-art implementations, namely DLV$^{\exists}$ and the Vadalog system (VADALOG). Indeed, it follows from Section 3 that the two systems are comparable over protected settings.

Systems Overview. DLV$^{\exists}$ is an extension of the answer set programming system DLV (Leone et al. 2006), enriched with $pchase_r$ for CQA over shy programs. To answer CQs, it employs a *materialization* approach, producing and storing all the facts for each predicate via the so-called *semi-naive* evaluation (Abiteboul, Hull, and Vianu 1995), where rules are evaluated according to a bottom-up strategy from the initial database. It is available online (Leone et al. 2017). VADALOG is a well-known system for KG management, implementing warded and $ichase$ for reasoning and CQA (Bellomarini, Sallinger, and Gottlob 2018). To answer CQs, it employs a *streaming* approach, building a *reasoning query graph* as a processing pipeline, where nodes correspond to algebra operators that perform transformations over the data pulled from their predecessors, and edges are dependency connections between the rules. It is available upon request. While DLV$^{\exists}$ integrates powerful optimization techniques that VADALOG has yet to incorporate, the latter is also extended with multiple features of practical utility, such as aggregations and equality-generating dependencies.

Experiments and Results. We compared DLV$^{\exists}$ and VADALOG over distinct reasoning scenarios and QA tasks. The experiments were run on a local installation of the two systems, using a machine equipped with an Intel Core i7-8665U CPU running at 1.90 GHz and 16 GB of RAM. The results of the experiments, as well as the steps to reproduce them on DLV$^{\exists}$, were made public (Baldazzi et al. 2022).

The first set of experiments is based on a financial recursive scenario about persons and companies (Bellomarini, Sallinger, and Gottlob 2018) and real datasets extracted from DBPedia (DBpedia 2018). A *person of significant control* (PSC) for a company is a person that directly or indirectly has some control over the company. The goal of this task is finding all the PSCs for the companies in DBPedia. We ran it for all the 67K available companies and for 1K, 10K, 100K, 500K and 1M of the available persons. Figure 2 illustrates similar execution times for the two systems, all under 5 seconds. Specifically, DLV$^{\exists}$ has better times in the first cases, partially due to VADALOG's longer pre-processing phase for the creation of the query graph. With larger datasets, VADALOG's performance progressively improves, thanks to its efficient recursion control to avoid the exploration of redundant areas of the reasoning space, and its *routing strategies* to traverse the query graph (Bellomarini et al. 2020).

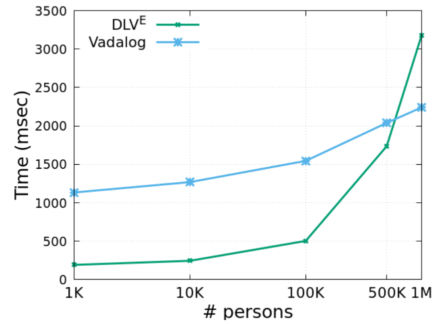

Figure 2: QA times for DBPedia PSC scenarios.

The second set of experiments is based on *Doctors*, a data integration task from the schema mapping literature (Mecca, Papotti, and Santoro 2014), included in the CHASEBENCH benchmark (Benedikt et al. 2017). It represents a plausible real-world case related to the healthcare domain and it features existential quantification. We ran it for 10K, 100K, 500K and 1M of all the datasets and over 7 distinct queries, of which we report the average times. Figure 3 illustrates that, while both systems show very good behaviour even in the most demanding cases, DLV$^{\exists}$ outperforms VADALOG. This is motivated by the powerful optimization techniques integrated in DLV$^{\exists}$ that limit the loading of redundant data for the query and reduce the space needed for materializing the output of $pchase_r$ (Leone et al. 2019).

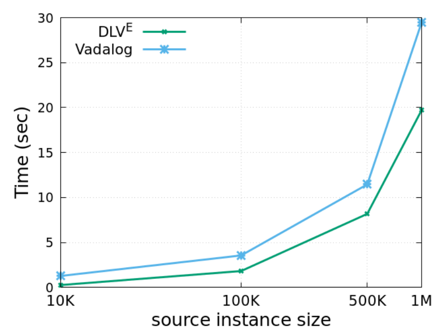

Figure 3: QA times for CHASEBENCH Doctors scenarios.

5 Conclusion

Shy Datalog$^{\exists}$ and Warded Datalog$^{\pm}$ are two relevant languages that extend Datalog with existential quantification while sustaining decidability of BCQA. Reacting to an explicit interest of the community, we provided an analysis of the fragments in terms of syntactical relationship and query evaluation, as well as an experimental comparison of their main implementations. Future work includes investigating their mutual reduction into the intersection fragment we defined and its impact in terms of their semantic relationship.

Acknowledgements

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

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of databases*, volume 8. Addison-Wesley Reading.

Adrian, W. T.; Alviano, M.; Calimeri, F.; Cuteri, B.; Dodaro, C.; Faber, W.; Fuscà, D.; Leone, N.; Manna, M.; Perri, S.; Ricca, F.; Veltri, P.; and Zangari, J. 2018. The ASP system DLV: advancements and applications. *Künstliche Intell.* 32(2-3):177–179.

Baget, J.-F.; Mugnier, M.-L.; Rudolph, S.; and Thomazo, M. 2011. Walking the complexity lines for generalized guarded existential rules. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Baget, J.-F.; Leclère, M.; and Mugnier, M.-L. 2010. Walking the decidability line for rules with existential variables. *KR* 10:466–476.

Baldazzi, T.; Bellomarini, L.; Favorito, M.; and Sallinger, E. 2022. Experimental evaluation. https://github.com/TeodoroBaldazzi/Shy-Warded-Datalog-KR2022. Accessed: 2022-02-01.

Bellomarini, L.; Gottlob, G.; Pieris, A.; and Sallinger, E. 2017. Swift logic for big data and knowledge graphs. In *IJCAI*, 2–10. Springer.

Bellomarini, L.; Benedetto, D.; Gottlob, G.; and Sallinger, E. 2020. Vadalog: A modern architecture for automated reasoning with large knowledge graphs. *IS*.

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

Benedikt, M.; Konstantinidis, G.; Mecca, G.; Motik, B.; Papotti, P.; Santoro, D.; and Tsamoura, E. 2017. Benchmarking the chase. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 37–52.

Berger, G.; Gottlob, G.; Pieris, A.; and Sallinger, E. 2019. The space-efficient core of vadalog. In *PODS*, 270–284. ACM.

Calì, A.; Gottlob, G.; Lukasiewicz, T.; Marnette, B.; and Pieris, A. 2010. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *2010 25th Annual IEEE LICS*, 228–242. IEEE.

Calì, A.; Gottlob, G.; and Kifer, M. 2013. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research* 48:115–174.

Calì, A.; Gottlob, G.; and Lukasiewicz, T. 2009. A general datalog-based framework for tractable query answering over ontologies. In *PODS*, 77–86.

Calì, A.; Gottlob, G.; and Lukasiewicz, T. 2012. A general datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics* 14:57–83.

Calì, A.; Gottlob, G.; and Pieris, A. 2012. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence* 193:87–128.

DBpedia. 2018. Global and unified access to knowledge graphs. http://wiki.dbpedia.org/services-resources/downloads/dbpedia-tables. Accessed: 2022-02-01.

Gottlob, G., and Pieris, A. 2015. Beyond sparql under owl 2 ql entailment regime: Rules to the rescue. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Gottlob, G.; Lukasiewicz, T.; and Pieris, A. 2014. Datalog+/-: Questions and answers. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Krötzsch, M., and Thost, V. 2016. Ontologies for knowledge graphs: Breaking the rules. In *International Semantic Web Conference (1)*, volume 9981 of *LNCS*, 376–392.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)* 7(3):499–562.

Leone, N.; Manna, M.; Terracina, G.; and Veltri, P. 2012. Efficiently computable datalog^E programs. In *Thirteenth international conference on the principles of knowledge representation and reasoning*.

Leone, N.; Manna, M.; Terracina, G.; and Veltri, P. 2017. Dlv^E system. https://www.mat.unical.it/dlve/. Accessed: 2022-02-01.

Leone, N.; Manna, M.; Terracina, G.; and Veltri, P. 2019. Fast query answering over existential rules. *ACM Transactions on Computational Logic (ToCL)* 20(2):1–48.

Maier, D.; Mendelzon, A. O.; and Sagiv, Y. 1979. Testing implications of data dependencies. *ACM Transactions on Database Systems* 4(4):455–468.

Mecca, G.; Papotti, P.; and Santoro, D. 2014. Iq-meter-an evaluation tool for data-transformation systems. In *2014 IEEE 30th International Conference on Data Engineering*, 1218–1221. IEEE.