Reasoning over Financial Scenarios with the Vadalog System

Teodoro Baldazzi
Università Roma Tre

Luigi Bellomarini
Bank of Italy

Emanuel Sallinger
TU Wien and University of Oxford

ABSTRACT

The last decade has witnessed a surge of interest from the financial sector towards intelligent systems that model complex business domains via powerful languages and employ them to reason over enterprise data and infer new knowledge. Since many languages and systems for Knowledge Representation and Reasoning have recently appeared on the scene, selecting the most suitable ones to tackle specific economic and financial scenarios is by no means trivial and calls for a deep awareness of the different language features, their mutual interactions, and how they are implemented by the available systems. Working with the Central Bank of Italy, we have been embarking on this adventure since five years ago and, together with the University of Oxford and TU Wien, have driven the development of Vadalog, one of the state-of-the-art reasoners. In this contribution, we summarize the characteristics of existing reasoners offering a summary table which, to the best of our knowledge, provides an in-depth comparison of all of them for the first time. Leveraging five reference use cases of interest, we discuss the most relevant language features and choices that have driven the design of Vadalog. As a by-product, we offer a guide for other entities in the private and public sectors (e.g., central banks, statistical offices, supervision authorities, financial intelligence units, etc.) willing to enhance their business processes with automated reasoning.

1 INTRODUCTION

A core pillar of our Information Age is the idea of knowledge as an essential driving force that can be relied upon to generate economic value in various fields of society [4]. In the last decade, the willingness to capitalize on and exploit such knowledge reached the economic and financial realm as well, sparking growing interest of the financial communities in modern *intelligent systems with reasoning capabilities*, able to capture complex real-world scenarios with powerful *Knowledge Representation and Reasoning* (KRR) languages and enhance the respective business processes.

Consequently, KRR systems and languages started to be adopted by central banks, supervision and monetary authorities, national statistical offices, financial intelligence units, to solve a plethora of relevant problems: addressing *banking supervision* inquiries, fighting hostile *company takeovers*, performing *credit-worthiness evaluation*, countering *anti-money laundering*, sustaining *insurance fraud detection*, backing *statistical research*, and many more, as witnessed by the success of dedicated venues and events [3, 54].

In parallel, in the database and the AI communities, we are observing the surge of increasingly mature, efficient, and expressive logic-based KRR languages, which have proved to be a key ingredient in the rise of a flurry of reasoning solutions. While featuring distinct properties and users—e.g., from *Knowledge Graph management* to *Big Data analytics*—all such systems share a common challenge: striking a balance between the expressive power needed to address real-world scenarios, and the scalability requirements needed to address massive volumes and user bases. In this work, we will be focusing on Datalog-based reasoners.

In the last five years of work with the Central Bank of Italy, we have been contributing to this newly-formed bridge between the financial sector and the database and AI communities. In fact, in the context of a joint effort with the University of Oxford and TU Wien, we developed Vadalog [21], a state-of-the-art reasoner that supports multiple evaluation methodologies and features, to effectively address relevant and challenging tasks from the economic and financial realm. We had the chance to report about many successful cases and industrial applications of our system [7, 8, 18, 19, 25, 28].

Yet, such a diversified and very technical offer of KRR languages and systems unavoidably represents a barrier to entry for new business players who are beginning to interact with this possibly unfamiliar field of research. Being able to determine the most suitable solutions to effectively address a specific economic or financial task is by no means trivial. Indeed, it involves being aware of which language features may best work to model the scenario at hand, such as *recursion*, *aggregation*, *existential quantification*, *negation* etc. Moreover, it requires a thorough understanding of the interplay among such features, as well as their implementation in reasoning systems. For instance, what capabilities are needed to support graph navigation, like in the case of supervision analyses or anti-money laundering inquiries? And, similarly, what features should I be looking for to address clustering settings? Recursion and existential quantification are the answers in these cases, respectively. Still, what if the two settings need to be satisfied at the same time? We know that query answering in Datalog becomes undecidable in the joint presence of these characteristics [43]. Thus, what systems are able to convey the right trade-off?

Contribution. Motivated by questions of the like, in this short paper we retrace the main milestones in the development of Vadalog, illustrating the properties and features we integrated into the system by presenting relevant financial use cases and scenarios for the Central Bank of Italy which drove our design choices. We complete this analysis by investigating how Vadalog emerges when compared with the other state-of-the-art reasoners existing in the literature. Such investigation is carried out by comparing the systems with respect to their main KRR capabilities, to the best of our knowledge, with the goal of offering a guideline for the new players in this field. Performance-based comparisons have already been discussed in related works and benchmark papers [6, 20, 29, 30]. More in detail, in this paper we provide the following contributions.

- We present **five representative use cases from the financial and economic domain**, namely *Company Control*, *Financial Shock Propagation*, *Close Link*, *Collateral Eligibility under Uncertainty* and *Company Supervision through Time*.

- We describe the **main language features, extensions, and reasoning capabilities of the Vadalog system**, required to solve the above use cases.
- We investigate the **related work of our system**, providing a summary table that compares the top-performing reasoners in the current literature, to the best of our knowledge, with respect to the main features and properties required to address such financial scenarios.
- We offer a **guidance for private and public entities** willing to begin to enhance their business processes with the powerful arsenal of techniques for logic-based automated reasoning.

Overview. The remainder of this paper is organized as follows. In Section 2 we provide some relevant notions regarding Datalog-based reasoning. In Section 3 we present the features of the Vadalog system to address the financial use cases. In Section 4 we discuss the related work and we illustrate the summary table of the reasoners. We draw our conclusions in Section 5.

2 REASONING WITH DATALOG

To enable the full potential of ontological reasoning, existential quantification is an essential requirement. This drove the development of the *Datalog*$^{\exists}$ family of logic languages [1], a natural extension of Datalog [5, 17, 37, 64] that allows existentially-quantified variables in rule heads.

Relational Foundations. Let **C**, **N**, and **V** be disjoint countably infinite sets of *constants*, *nulls* and *variables*, respectively. A (*relational*) *schema* **S** is a finite set of relation symbols (or *predicates*) with associated arity. A *term* is either a constant or a variable. An *atom* over **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 **S** associates to each symbol in **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 $h : \mathbf{C} \cup \mathbf{N} \cup \mathbf{V} \to \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ s.t. $h(t) = t$ if $t \in \mathbf{C}$, $h(t) \in \mathbf{C} \cup \mathbf{N}$ if $t \in \mathbf{N}$ and for each atom $a(t_1, \ldots, t_n) \in \varsigma_1$, then $h(a(t_1, \ldots, t_n)) = a(h(t_1), \ldots, h(t_n)) \in \varsigma_2$.

Dependencies. A Datalog$^{\exists}$ program consists of a set of facts and *existential rules*, or *tuple-generating dependencies* (TGDs), function-free Horn clauses of the form $\forall\bar{x}\forall\bar{y}(\varphi(\bar{x},\bar{y})\to\exists\bar{z}\,\psi(\bar{x},\bar{z}))$, where $\varphi(\bar{x},\bar{y})$ (the *body*) and $\psi(\bar{x},\bar{z})$ (the *head*) are conjunctions of atoms over the respective predicates and the arguments are vectors of variables and constants. Moreover, it may feature *equality-generating dependencies* (EGDs), first-order implications of the form $\forall\bar{x}(\varphi(\bar{x})\to x_i = x_j)$, where $\varphi(\bar{x})$ is a conjunction of atoms and $x_i, x_j \in \bar{x}$. Quantifiers can be omitted.

Language Extensions. Real-world applications require reasoning systems to support multiple features that extend the declarative language. Among them, *aggregate functions*, namely *sum*, *prod*, *min*, *max* and *count*, as well as SQL-like grouping constructs, are particularly relevant. Important extensions also include *negations* and *negative constraints*, of the form $\forall\bar{x}(\varphi(\bar{x})\to\bot)$, where $\varphi(\bar{x})$ is a conjunction of atoms and $\bot$ denotes the truth constant false to model disjointness or non-membership. Other features of practical utility are *expressions* in rule bodies, modelled with *comparison* ($>, <, \geq, \leq, \neq$) and *algebraic* ($+, -, *, /$, etc.) operators.

Ontological Reasoning and Query Answering. Intuitively speaking, an ontological reasoning task consists in answering a *conjunctive query* (CQ) Q over a database D, augmented with a set Σ of logical rules. A conjunctive query Q is an implication $q(\bar{x}) \leftarrow \psi(\bar{x},\bar{z})$, where $\psi(\bar{x},\bar{z})$ is a conjunction of atoms over **S**, $q(\bar{x})$ is an n-ary predicate $\notin$ **S**, and $\bar{x}, \bar{z}$ are vectors of variables and constants. A *Boolean* CQ (BCQ) $Q \leftarrow \psi(\bar{x},\bar{z})$ over D under Σ is a particular kind of CQ whose answer is positive iff there exists a homomorphism h: $C \cup V \to C \cup N$ s.t. $h(\psi(\bar{x},\bar{z})) \subseteq D$.

Other Forms of Reasoning. Traditional KRR languages and approaches are often insufficient to properly address complex real-world scenarios. For instance, how should we model events that are not always definitive, but rather may only occur with a given probability or in specific time frames? For these reasons, ontological reasoning has been complemented with other forms of automated reasoning, such as *probabilistic*, *temporal*, etc.

Chase Procedure. The semantics of a Datalog$^{\exists}$ program is usually defined in an operational way with an algorithmic tool known as the *chase procedure* [52]. Intuitively, it enforces the satisfaction of a set of dependencies Σ over a database D, incrementally expanding D with facts entailed via the application of the rules over D, until all of them are *satisfied*. Such facts possibly contain fresh new symbols ν (technically, *labelled nulls*) to satisfy existential quantification. A TGD $\sigma : \varphi(\bar{x},\bar{y})\to\psi(\bar{x},\bar{z}))$ is satisfied by D if, whenever a homomorphism θ occurs such that $\theta(\varphi(\bar{x},\bar{y})) \subseteq D$, there exists an *extension* θ' of θ (i.e., $\theta \subseteq \theta'$) such that $\theta'(\psi(\bar{x},\bar{z})) \subseteq D$. In other words, in the standard (namely, *oblivious*) chase, σ is applicable to D if there exists a θ such that $\theta(\varphi(\bar{x},\bar{y})) \subseteq D$. When applied, σ generates a new fact $\theta'(\psi(\bar{x},\bar{z}))$ that enriches D, if not already present, where θ' extends θ by mapping the variables of $\bar{z}$ (if not empty) to new labelled nulls named in a lexicographical order. Similarly, an EGD $\eta : \varphi(\bar{x})\to x_i = x_j$ is satisfied if, for each $\theta(\varphi(\bar{x}))$, $\theta(x_i) = \theta(x_j)$. Thus, η is applicable to D if there exists a θ such that $\theta(\varphi(\bar{x}))$ $\subseteq D$ and $\theta(x_i) \neq \theta(x_j)$. When applied, η checks if x_i and x_j are constants, in which case it fails; otherwise, it performs a *unification*, replacing each occurrence of $\theta(x_j)$ with $\theta(x_i)$ if $\theta(x_i)$ precedes $\theta(x_j)$ in the lexicographical order, vice versa otherwise. The *chase graph* $\mathcal{G}(D,\Sigma)$ is the directed graph with the facts in D and the ones generated during the chase as nodes and an edge from a node n to a node m if m is obtained from n (and possibly other facts) via applicable homomorphisms [34].

Evaluation Methodologies. Modern reasoners encode chase procedures via two main approaches: (i) a *materialization* technique that consists of producing and storing all the facts for each predicate by adopting the so-called *semi-naive* evaluation [1]. In this case, rules are typically evaluated according to push-based strategies, starting from the initial database and repeatedly applying the rules until a fixpoint is reached; (ii) a *streaming* technique that adopts reasoning query graphs [44], where nodes correspond to relational algebra operators (select, project and join) and edges are dependency connections between the rules. Such a graph forms an active pipeline and the data flows through its nodes, each receiving input data from the previous nodes and performing the required transformations in a pull-based fashion.

3 VADALOG VIA FINANCIAL SCENARIOS

In the previous section we provided an overview of the main theoretical concepts at the foundation of Datalog-based reasoning. However, translating them into features of practical utility and employing them to reason over complex real-world settings is by no means trivial. In this section we outline our journey in the development of the state-of-the-art Vadalog system, presenting financial scenarios of interest and adapting such features to effectively perform reasoning over them.

Scenario 1. (Company Control) This set of rules models the existence of a link between companies with common shareholders. It is a smaller and not entirely realistic version of the financial scenario, here simplified for explanation purposes.

$$Company(x) \rightarrow \exists s\ SH(x,s) \quad (1)$$
$$Controls(x,y), SH(x,s) \rightarrow SH(y,s) \quad (2)$$
$$SH(x,s), SH(y,s) \rightarrow StrongLink(x,y) \quad (3)$$
$$StrongLink(x,y) \rightarrow \exists s\ SH(x,s), SH(y,s) \quad (4)$$

For each company x there exists a shareholder (SH) s (rule 1). If x controls a company y, then s is also a shareholder of y (rule 2). If x and y have a common shareholder, then they are in a strong link (rule 3). Vice versa, if there is a strong link between x and y, then they have a common shareholder (rule 4).

To address this first scenario, our reasoner required full support for existential quantification in TGDs and recursion. However, ontological reasoning under Datalog$^{\exists}$ proved to be undecidable, due to the interplay between existentials and recursion, which may cause infinite labelled nulls to be generated in the chase and non-termination as a consequence [33, 34]. This led to the proposal in the literature of many languages of the so-called *Datalog*$^{\pm}$ family [9, 12, 34, 35, 49]. They apply to Datalog$^{\exists}$ specific restrictions to achieve a good trade-off between expressive power and computational complexity of the reasoning. Figure 1 provides an overview of the main Datalog$^{\pm}$ languages, their syntactic containment, and reasoning data complexity.

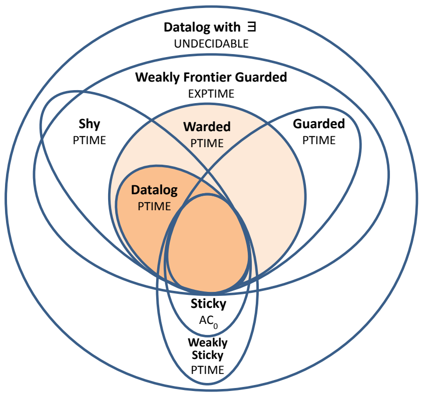


Figure 1: Datalog$^{\pm}$ languages and data complexity.

Specifically, we employed *Warded* Datalog$^{\pm}$ [43], a powerful fragment which we made usable in practice and implemented into Vadalog [24]. It encompasses both a high expressive power, capturing all SPARQL queries under OWL 2 QL entailment regime and set semantics, as well as a very good trade-off with data complexity, featuring PTIME for the reasoning and thus fulfilling the scalability requirements of our domain. This is achieved via the *wardedness* syntactic condition, which restricts the propagation of labelled nulls ν in the chase only to cases that are not *dangerous* for the decidability of the task. Wardedness also preserves termination of the chase procedure, enabling an *isomorphism*-based variant of the oblivious chase we developed that employs a *firing condition* to limit the applicability of the TGDs. Indeed, given an applicable TGD $\sigma : \varphi(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{z})$ and a homomorphism θ, σ is applied if, additionally, there is no isomorphism (i.e., same predicate name, same constants in the same positions and bijection between labelled nulls) between the newly-generated fact $\theta'(\psi(\bar{x}, \bar{z}))$ and a tuple already in the database.

Algorithm 1 provides the pseudocode of our *isomorphic chase*-based evaluation. The algorithm takes as input a database D, a set Σ of Warded TGDs and a query Q. The output of the algorithm is the answer to Q. First of all, the chase instance *chase* that stores the facts generated during the procedure is initialized to D and corresponds to the initial chase state (line 2). A data structure H is employed to store, at each step of the procedure, all the applicable homomorphisms (line 3). While H is not empty, the next applicable homomorphism h to activate is extracted from H (line 5), and a new fact p is generated (line 6). Now the isomorphism check occurs. If p is not isomorphic with a fact already present in *chase*, then it is added to the chase instance (lines 7-8). Finally, the new applicable homomorphisms, derived from p and Σ, are added to H (line 9). If H is empty, that is, all the applicable homomorphisms have already been triggered, then the answer to Q is evaluated and the procedure terminates (line 10).

Algorithm 1 Isomorphic Chase Evaluation Algorithm.

```
1:  function ISOMORPHIC_CHASE_EVAL($D$, $\Sigma$, $Q$)
2:      $chase = D$                          ▷ chase instance is initialized
3:      $H$ = init_structure($D, \Sigma$)    ▷ applicable homomorphisms
4:      while not $H$.is_empty() do
5:          $h = H$.poll_first()
6:          $p$ = apply($\Sigma, h$)          ▷ apply current homomorphism
7:          if check_isomorphism($p$,$chase$) then
8:              $chase = chase \cup \{p\}$
9:              $H$ = update_structure($\Sigma, p$)
10:     return answer($Q$, $chase$)
```

With reference to Scenario 1, let us consider the database instance $D = \{Company(Hsb), Company(Iba), Controls(Hsb, Iba)\}$ and the query Q: "*what are all the entailed StrongLinks?*" as ontological reasoning task. Employing the isomorphic chase, we first generate $SH(Hsb,\nu_0)$ by activating rule 1 from the fact $Company(Hsb)$. Next, we obtain $SH(Iba,\nu_0)$ from rule 2, and consequently $StrongLink(Hsb, Hsb)$, $StrongLink(Hsb, Iba)$, $StrongLink(Iba, Hsb)$ and $StrongLink(Iba, Iba)$ via the join in rule 3: these facts are the result for Q. Note that, by activating now rule 1 on $Company(Iba)$, we would generate $SH(Iba,\nu_1)$. Similarly, from $StrongLink(Iba, Iba)$ and rule 4 we would obtain $SH(Hsb,\nu_2)$ and $SH(Iba,\nu_2)$: this would in turn lead to new activations of rule 3 and the generation of an infinite set $\bigcup_{i=3,...}\{SH(Hsb, \nu_i), SH(Iba, \nu_i)\}$. Indeed, our chase variant prevents non-termination by pruning such facts, as they are isomorphic with $SH(Hsb,\nu_0)$ and $SH(Iba,\nu_0)$ already generated. The chase graph corresponding to the evaluation of Q via isomorphic chase is provided in Figure 2. Note that blue edges correspond to applied homomorphisms, red edges represent prunings of applicable homomorphisms, and green ones identify a positive isomorphism check between a pruned fact and one already generated. Edges are labelled by the triggered rules.

The correctness of such methodology is corroborated by the *reasoning boundedness* property of the Warded fragment, which states that facts derived from isomorphic origins are isomorphic, thus uninformative for the reasoning task. Additional techniques integrated into Vadalog enforce that the result of the isomorphic chase over a generic D and a given set of warded TGDs is always correct and unique for each possible CQ Q [13, 21, 29].

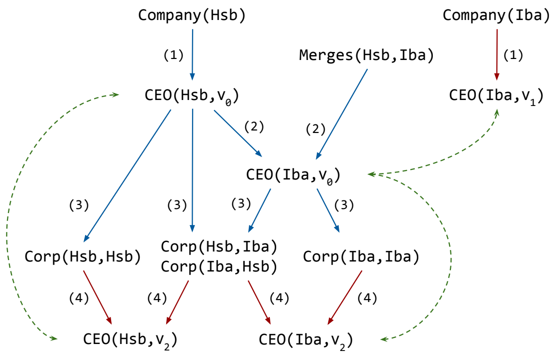


Figure 2: Chase graph for Scenario 1 over D.

Scenario 2. (**Financial Shock Propagation**) This set of rules models how the default of a financial intermediary affects other intermediaries that are financially exposed with it.

$$\mathit{FinInt}(x), \mathit{Own}(p,x,w), w > 0.3 \rightarrow \mathit{KP}(p,x) \quad (1)$$

$$\mathit{FinInt}(x), \mathit{NPL}(x) \rightarrow \exists f\ \mathit{Default}(x,f,f) \quad (2)$$

$$\mathit{Default}(x_1,f_x,f_1), \mathit{Exp}(x_1,x_2) \rightarrow \exists f_2\ \mathit{Default}(x_2,f_1,f_2) \quad (3)$$

$$\mathit{Default}(x,f_1,f_2), \mathit{KP}(p,x) \rightarrow \exists i\ \mathit{Inv}(p,x,i) \quad (4)$$

$$\mathit{KP}(p_1,x), \mathit{KP}(p_2,x), \mathit{Inv}(p_1,x,i_1), \mathit{Inv}(p_2,x,i_2) \rightarrow i_1 = i_2 \quad (5)$$

$$\mathit{Inv}(p_1,x_1,i_1), \mathit{Inv}(p_2,x_2,i_2), \mathit{Exp}(x_1,x_2) \rightarrow i_1 = i_2 \quad (6)$$

An individual p is a key person (KP) of a financial intermediary (FinInt) x if p owns more than 30% of the shares (w) of x (rule 1). If a FinInt x is involved in non-performing loans (NPL), then it will default on its debts, initiating a failure event f (rule 2). If a FinInt x_2 is financially exposed (Exp) with another FinInt x_1 which undergoes a failure event f_1, caused by another failure f_x, then x_2 will be in turn involved in a failure f_2 caused by f_1 (rule 3). A financial investigation i is launched for each KP of a defaulted FinInt x (rule 4). If p_1 and p_2 are KPs of the same defaulted FinInt x, then they are involved in the same investigation (rule 5). Finally, two KPs p_1 and p_2 of distinct defaulted FinInt x_1 and x_2 are under the same investigation if x_2 is exposed with x_1 (rule 6).

It often occurs that TGDs are not sufficient to exploit the full expressive power of existential quantification. To address settings such as the one above in Vadalog, we integrated EGDs in Warded Datalog$^{\pm}$. Yet, the presence of EGDs causes the undecidability of ontological reasoning even in our restricted settings. Among the solutions to enable their interplay with TGDs, the most adopted one in the literature is the fragment of *separable* EGDs [36], which however does not make use of the expressive capabilities of EGDs. Indeed, they are able to only enforce constraints on ground values from the domain and thus do not contribute to the answer of the reasoning task in practice.

To tackle this limitation in our settings, we developed and implemented in Vadalog the novel fragment of *harmless* EGDs [20], which enables more expressive interactions with warded TGDs, while preserving the reasoning tractability offered by Warded Datalog$^{\pm}$. 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, "harms" the chase procedure. Indeed, unlike separable ones, harmless EGDs contribute facts to the ontological reasoning task. With reference to Scenario 2, the EGDs in rules 5 and 6 are harmless and can be employed to group together all the key persons and to subject them to the same investigation. This use case also required us to integrate expressions and operators into our reasoner.

Scenario 3. (**Close Link**) This set of rules models the existence of a direct or indirect link between companies, based on a high overlap of shares, to assess whether a company can act as a guarantor for loans to another one.

$$\mathit{Own}(c_1,c_2,s) \rightarrow \mathit{MCl}(c_1,c_2,s) \quad (1)$$

$$\mathit{MCl}(c_1,c_2,s_1), \mathit{Own}(c_2,c_3,s_2) \rightarrow \mathit{MCl}(c_1,c_3,s_1 * s_2) \quad (2)$$

$$\mathit{MCl}(c_1,c_2,s), ts = \mathit{msum}(s), ts \geq 0.2 \rightarrow \mathit{Cl}_1(c_1,c_2) \quad (3)$$

$$\mathit{Cl}_1(c_3,c_1), \mathit{Cl}_1(c_3,c_2), \neg\, \mathit{Cl}_1(c_1,c_2), c_1 \neq c_2 \rightarrow \mathit{Cl}_2(c_1,c_2) \quad (4)$$

$$\mathit{Cl}_1(c_1,c_2) \rightarrow \mathit{Cl}(c_1,c_2) \quad (5)$$

$$\mathit{Cl}_2(c_1,c_2) \rightarrow \mathit{Cl}(c_1,c_2) \quad (6)$$

Two companies c_1 and c_2 s.t. c_1 owns a fraction s of the shares of c_2 are possible close links (MCl) (rule 1). If c_1 and c_2 are MCl with a share s_1 and c_2 owns in return a share s_2 of a company c_3, then also c_1 and c_3 are MCl with a share of $s_1 * s_2$ (rule 2). If the sum of all the partial shares s of c_2 owned directly or indirectly by c_1 is greater than or equal to 0.2, then c_1 and c_2 are close links (rule 3). If a third-party c_3 owns directly or indirectly, through one or more other companies, 0.2 or more of the share of c_1 and c_2 (close links with c_3), then c_1 and c_2 are also close links (rule 4).

This scenario introduces two other essential features a reasoner must include to address real-world settings: negations and aggregations. We say that a predicate is *extensional* if it is a ground relation in the database, thus it does not appear in a rule-head of the program, otherwise it is *intensional*. Regarding negations, Vadalog supports both *grounded* (i.e., in front of atoms whose predicates are extensional) and *stratified* negation (i.e., in front of atoms whose predicates are intensional and not recursive) [48].

Regarding aggregations, *standard* aggregate functions proved to be affected by multiple limitations and are disallowed in recursive settings. This issue has been solved by the recently proposed class of *monotonic* aggregations [60]. A rule with an aggregation is a first-order sentence $\forall \bar{x}(\varphi(\bar{x}), z = \mathbf{maggr}(x, \langle \bar{c} \rangle) \rightarrow \psi(\bar{g}, z))$, where **maggr** is the name of an aggregation function, $x \in \bar{x}$, and $\bar{g} \subseteq \bar{x}$ is a n-uple of group-by arguments, $\bar{c} \subseteq \bar{x}$ (with $\bar{c} \cap \bar{g} = \emptyset$) is a m-uple of variables contributing to the aggregation and z a *monotonic aggregate*, that is, an existentially quantified variable whose value is computed by the aggregation. Intuitively, for a monotonically decreasing (increasing) aggregation function, z memorizes at each step of the execution the most recently computed aggregate and returns an updated value at each invocation such that, for each value of $\bar{c}$, the minimum (maximum) value of x_i is considered in the current aggregate. In Vadalog, we extended such a powerful class of aggregations to be employed in the context of existential quantification [29].

Scenario 4. (**Collateral Eligibility under Uncertainty**) This set of rules models how lenders of distinct types are subject to restrictions enforced by financial supervision authorities that require loans to be covered by a collateral.

$$0.9 :: \mathit{LenderT}(x,y), \mathit{RR}(y,z) \rightarrow \exists v\ \mathit{Guarantee}(x,z,v) \quad (1)$$

$$0.8 :: \mathit{LenderT}(x,y), \mathit{LenderC}(y,z) \rightarrow \mathit{LenderT}(x,z) \quad (2)$$

$$0.7 :: \mathit{Contract}(x,y,z), \mathit{Exp}(y,w) \rightarrow \mathit{Contract}(z,w,x) \quad (3)$$

$$\mathit{Contract}(x,y,z), \mathit{RR}(w,y) \rightarrow \mathit{LenderT}(x,w) \quad (4)$$

Ignoring what precedes the :: symbols, if a lender x is of type y (e.g., a bank, a small company, etc.) and lenders of type y are subject to regulatory restrictions (RR) of type z (e.g., securities, real estate properties, cash, etc), then there exists a collateral of type z for x issued by a guarantor v (an individual, a bank, or a financial intermediary) (rule 1). If the type y is a subclass of the lender class z (e.g., credit unions is a subclass of retail lender), then the lender x is of type z (rule 2). If the loan from a lender x to a borrower z has been formalized by a contract, then there exists another contract for the financial exposure (Exp, i.e., the repayment obligation) of type w (corresponding to the type y of the loan) from z to x (rule 3). If a contract from x to z is in place to satisfy a RR that requires a guarantee with contract type y, based on a lender type w, then x is also of type w (rule 4).

This scenario features some notion of uncertainty related to our domain. Indeed, depending on how each financial intermediary implements the regulations, rules 1-3 may apply or not. Intuitively, probabilistic reasoning requires computing the probabilistic answer to a query as a set $\{\langle \bar{t}, P(\bar{t})\rangle\}$, where $\bar{t}$ is a fact and $P(\bar{t})$ is its *marginal probability*, i.e., the probability for $\bar{t}$ to be entailed. To address this task we developed SOFT VADALOG, the extension of VADALOG for probabilistic reasoning. Since computing exact marginal probabilities is intractable, we introduced the *MCMC-chase*, a chase variant that approximates them, performing logical and probabilistic inference at the same time while achieving PTIME [22, 26, 27].

SCENARIO 5. (**Company Supervision through Time**) This set of rules models how a governmental institution supervises the changes in the corporate structure of companies with strategic relevance, as well as the actions of those shareholders who are buying into the companies later in the game.

$$\diamondplus_{[0,1]} SignificantShare(x,y),$$
$$\neg \diamondminus_{[0,1]} SignificantShare(x,y) \rightarrow SignificantOwner(x,y) \quad (1)$$
$$WatchCompany(y), SignificantOwner(x,y),$$
$$Connected(x,z) \rightarrow WatchCompany(z) \quad (2)$$

If in an interval in the past (denoted by $\diamondminus_{[0,1]}$) a shareholder x does not own a significant amount of shares of a company y, while that is the case at some point in a future interval (denoted by $\diamondplus_{[0,1]}$), then x is a significant owner of y (rule 1). If x is a significant owner of a company y in a watchlist, then all the other companies z that are connected to x are also added to the watchlist (rule 2).

This last scenario consists in a temporal reasoning task. To support this form of reasoning in Datalog, the AI community introduced *DatalogMTL*, which extends the language with forms of *time awareness* and encodes operators that link standard atemporal assertions to references and streams of data valid within specific *time windows* [32]. Yet, the development of temporal reasoners based on DatalogMTL was still in its infancy. We extended VADALOG with temporal reasoning capabilities, providing for the first time, to the best of our knowledge, a production-ready temporal reasoner that fully supports recursion and allows addressing real-world temporal tasks with good results in performance and scalability [23].

As future directions, we are working on extending VADALOG with novel forms of reasoning that make use of powerful Machine Learning and AI techniques to enrich, support, and guide the evaluation of complex tasks in the financial domain. Some of our results regarding these novel topics are already present in the most recent literature [15, 16, 56, 65].

We close this section by briefly describing the specialized architecture and execution model that we devised for VADALOG.

Pipeline Architecture. Our system employs the *pipes and filters* architectural style. The set of logic rules Σ and the queries are compiled together into an active pipeline that reads the data from the input sources, performs the needed transformations, and produces the desired output as a result. This process is implemented by four dedicated architectural components [29]: (i). a *query processor* rewrites the CQ $Q : q(\bar{x}) \leftarrow \psi(\bar{x}, \bar{z})$ into an atomic query by first updating $\Sigma = \Sigma \cup \{\rho_Q\}$, where $\rho_Q : \chi(\bar{x}) \leftarrow \psi(\bar{x}, \bar{z})$ and χ is an invented predicate, and then creating the atomic query $\hat{Q} : q(\bar{x}) \leftarrow \chi(\bar{x})$; (ii). a *logic optimizer* performs logic transformations and rewritings to the original set of rules, such as multiple head elimination, removal of redundancies and harmful join elimination [11, 13, 14], as well as general logic optimizations inherited or adapted from typical RDBMS heuristics; (iii). a *logic compiler* takes as input the set of optimized rules and transforms it into a logic pipeline in the form of a *reasoning access plan*, which can be thought as a predicate graph where each node (filter) represents an atom and there is an edge (pipe) from a node *m* to a node *n* if there is a rule with *m* in the body and *n* in the head; (iv). a *query compiler* transforms the reasoning access plan into a *reasoning query plan*, a processing pipeline (an example of which is provided in Figure 3, as discussed in the following paragraph) where the nodes are translated into active *data scans* (linear scans for linear TGDs, join scans for join TGDs, EGD scans for harmless EGDs, and an output scan for the query), connected by intermediate buffers.

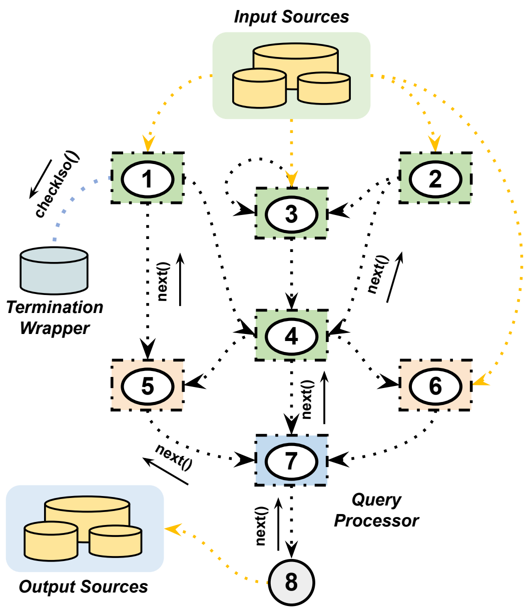


Figure 3: Processing pipeline of VADALOG for Scenario 2.

Execution Model. The reasoning process in VADALOG is performed as a data stream along the pipeline, implemented with a *pull-based* (query-driven) approach where each filter (i.e., scan) reads tuples from the respective parent, from the output scan down to the external data stores that inject ground facts into the pipeline. It is a generalization of the *volcano iterator model* [44] and allows to activate only the chase steps required to answer a query without generating the whole output of the chase procedure. Interactions between scans occur by means of primitives open(), next(), get(), close(), which open the parent stream, ask for the presence of a fact to fetch, obtain it, and close the communication, respectively [20]. Since, for each filter, multiple parent filters may be available, Vadalog selects which one to invoke (via next() call) by employing specific *routing strategies* (*round-robin*, *shortest path*, etc.) that manage a priority queue of the sources [21]. Whenever a TGD scan is triggered, a wrapper performs the isomorphism check to ensure termination, via the checkIso() primitive. Figure 3 shows the processing pipeline for Scenario 2 given, as ontological reasoning task, the query

$$SameInv(p_1, p_2) \leftarrow Inv(p_1, \_, i), Inv(p_2, \_, i) \qquad (Q)$$

of finding the pairs of key persons involved in the same investigation. Here the output filter 8 sends a next() message to 7, which propagates it to the TGDs (green) and EGDs (tan) scans. Note that each filter in the figure is labelled with the corresponding rule number from the above scenario.

4 FEATURE-BASED COMPARISON OF RELATED REASONING SYSTEMS

A reasoning system requires supporting a wide range of language features and techniques to address complex real-world tasks, such as the ones in the financial domain. As discussed in the previous section, these requirements drove the design and the development of Vadalog itself, as well as of other modern reasoners. In fact, a related question that we asked ourselves is how Vadalog ranks when compared with the other state-of-the-art systems with reasoning capabilities, especially with respect to the features required to tackle the scenarios presented in the previous section. Motivated by this, we investigated the related work of our reasoner, summarizing our findings in the table in Figure 4, which provides, to the best of our knowledge, the first feature-based comparison of the top-performing reasoners that are present in the current literature. Here our goal is not comparing systems on performance, which we consider out of the scope of the paper, and has however been carried out in related works and benchmark papers [6, 20, 29, 30].

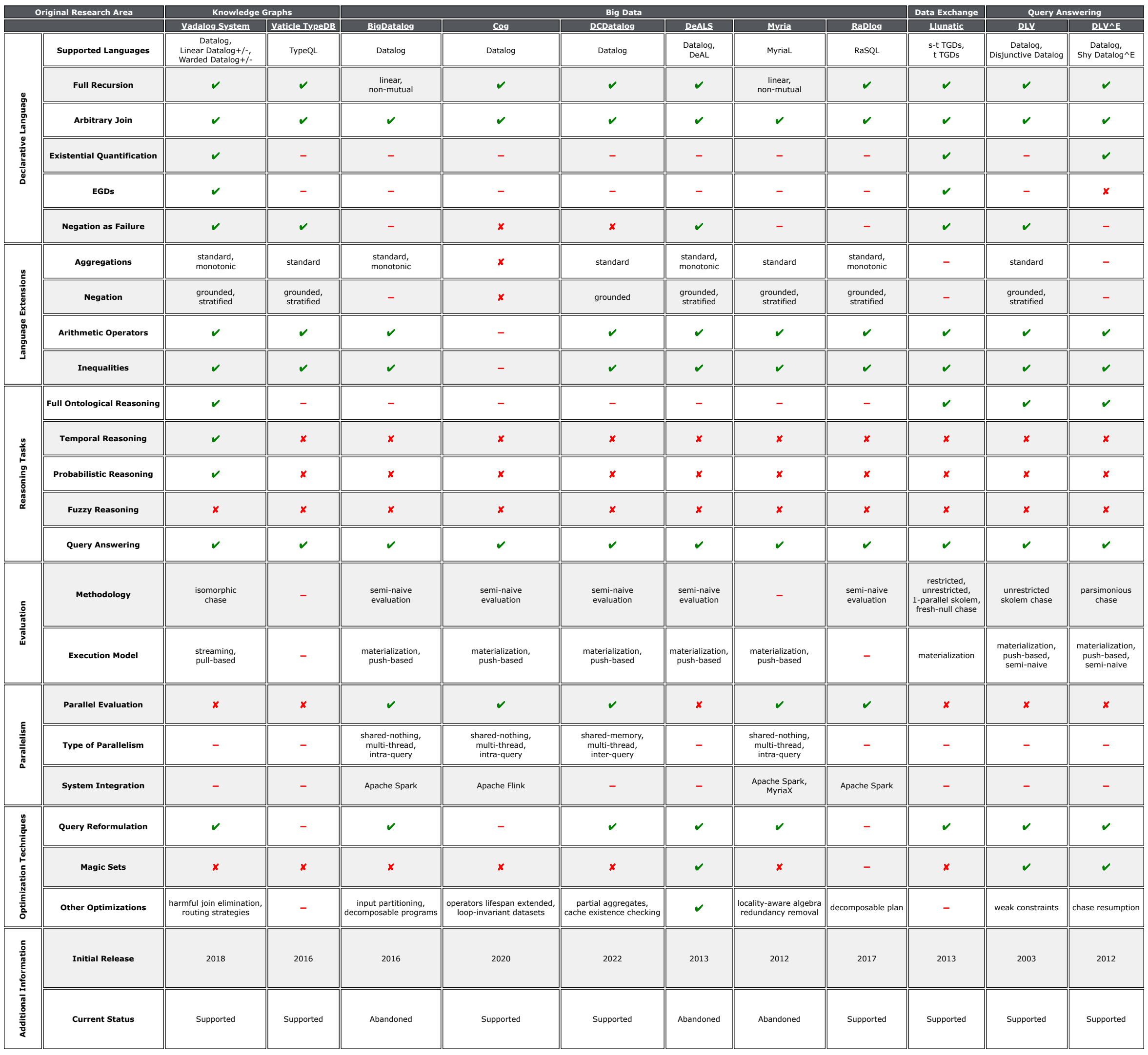
Overview of the Comparisons. Specifically, comparisons are performed with respect to the following categories of features:

- *declarative language*: it consists of the main properties of the formalism employed by a system, such as whether it allows expressing TGDs and EGDs, as well as if it supports relevant features such as recursion and existential quantification;
- *language extensions*: it includes advanced features that extend the declarative language, such as (standard or monotonic) aggregations, negations and arithmetic operators;
- *reasoning tasks*: it lists the main forms of reasoning that a system can support, such as ontological, temporal and probabilistic;
- *evaluation*: it shows the specifics of the evaluation employed by a system, such as the adopted chase variant and if it is based on a streaming or a materialization approach;
- *parallelism*: it encapsulates if a system supports *parallel evaluation*, as well as which form of parallelism it features (*shared-memory*, *shared-nothing*, etc.);
- *optimization techniques*: it consists of relevant optimizations available in a system, such as *query rewriting* and *magic sets* [1];
- *additional information*: it reports a system's initial release and whether it is currently supported.

The selection of such criteria for comparison derives from their relevance to properly address real-world tasks. Indeed, they range from the syntactic features, required to model industrial scenarios such as our financial ones discussed in the previous paragraph, to the overall reasoning capabilities and methodologies that enable the evaluation of such settings with each system in practice. We complete the comparison with additional parameters that impact, directly or indirectly, the reasoner, such as whether it features optimization techniques that support the evaluation and whether it is currently supported and aligned with the most recent advancements in the field.

Apart from Vadalog, we now list the other investigated systems, in order of appearance in the table in Figure 4.

- *Vaticle TypeDB* [63]: strongly-typed database that enables type and rule inference, as well as entity-relationship data modelling via its proprietary language *TypeQL*.
- *BigDatalog* [59]: full Datalog implementation, extended with monotonic aggregations and other features of practical utility, on Apache Spark for large-scale analytics.
- *Cog* [47]: evaluation system of positive Datalog programs that do not contain aggregates on the Apache Flink distributed dataflow system for Big Data analytics.
- *DCDatalog* [66]: parallel engine designed for shared-memory multicore machines to scale up and optimize Datalog evaluation in the presence of recursion.
- *DeALS* [61]: Datalog system extended with negation and monotonic aggregates for both traditional DBMS queries and graph analytics applications.
- *Myria* [53]: stack for Big Data data management and analytics with a shared-nothing and scalable query execution engine.
- *RaDlog* [46]: system implemented on top of Apache Spark that extends the SQL standard to support aggregates in-recursion (RaSQL) for Big Data analytics tasks.
- *Llunatic* [41]: initially developed for data cleaning, has since been redesigned as an open-source data exchange system running on top of PostgreSQL and supporting TGDs and EGDs.
- *DLV* [51]: disjunctive Datalog system for knowledge representation and reasoning supporting a range of features such non-monotonic negation, aggregates, and user-defined functions.
- $DLV^{\exists}$ [50]: system for query answering over programs in the *Shy* fragment of the Datalog$^{\pm}$ family, extending DLV with a bottom-up evaluation strategy and a chase variant that prevents the generation of facts homomorphic to previously created ones.
- *E* [57]: first-order theorem prover that can perform query answering tasks and that captures TGDs and EGDs.
- *Graal* [10]: open-source Java toolkit developed for ontological query answering in the framework of existential rules.
- *RDFox* [55]: high-performance RAM-based Datalog engine that supports TGDs and EGDs, as well as multiple features such as aggregations and stratified negation, to manage graph-structured data represented according to the RDF data model and to perform ontological reasoning via materialization.
- *OWL2DLV* [2]: Datalog system that extends DLV for large-scale ontological reasoning, evaluating SPARQL conjunctive queries over very large OWL 2 knowledge bases.
- *RecStep* [40]: general-purpose Datalog engine extended with stratified negation and aggregations, built on top of QuickStep, an in-memory parallel single-node relational system.
- *PDQ* [31]: developed to generate query plans over semantically-interconnected data sources, it reduces query reformulation to checking containment under TGDs and EGDs via the chase.
- *MASTRO* [38]: tool developed for ontology-based data access in which ontologies are specified in a family of tractable Description Logics, extended with features for intensional reasoning.



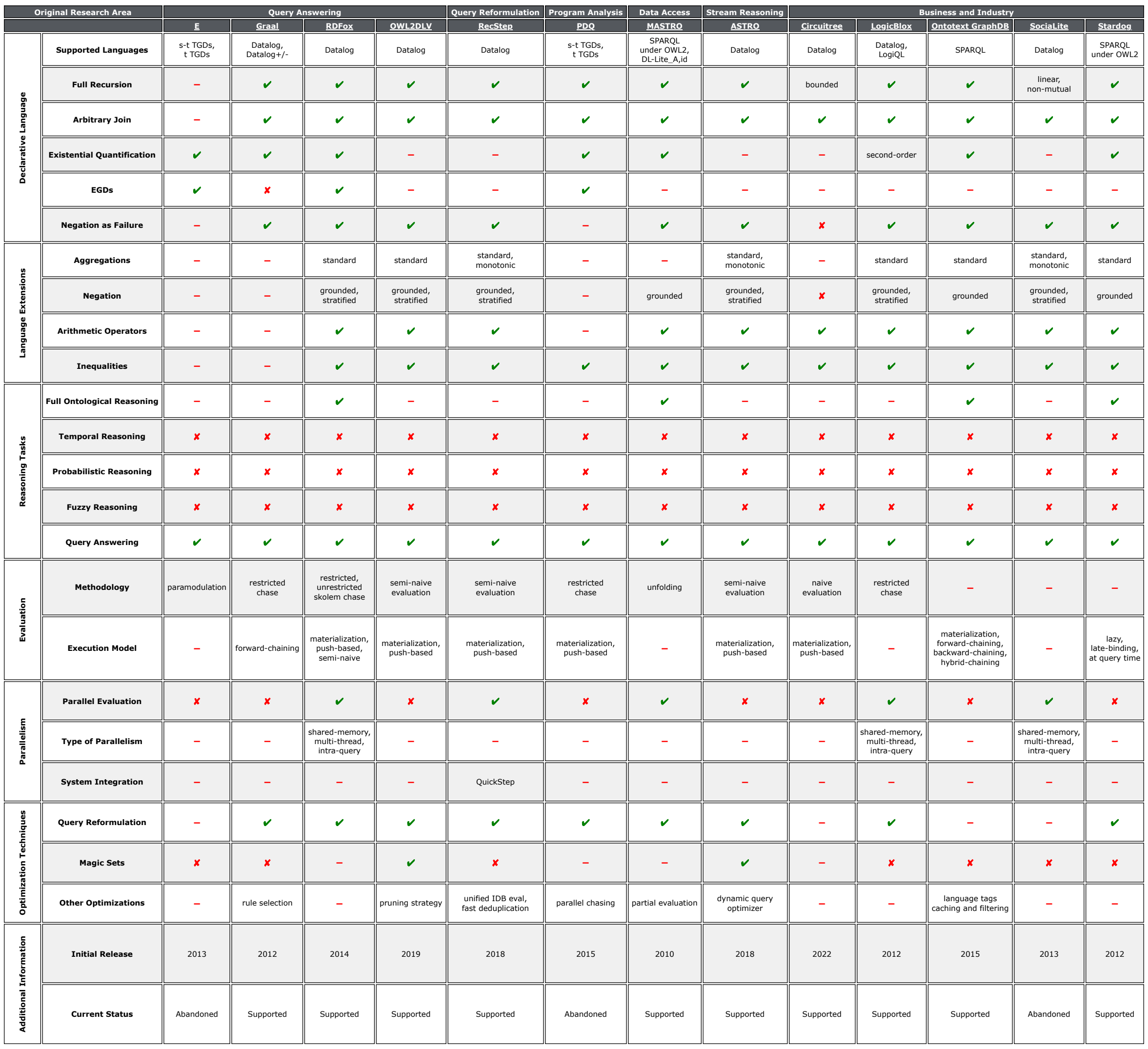
Original Research Area		Knowledge Graphs		Big Data						Data Exchange	Query Answering	
		Vadalog System	Vaticle TypeDB	BigDatalog	Cog	DCDatalog	DeALS	Myria	RaDlog	Llunatic	DLV	DLV^E
Declarative Language	Supported Languages	Datalog, Linear Datalog+/-, Warded Datalog+/-	TypeQL	Datalog	Datalog	Datalog	Datalog, DeAL	MyriaL	RaSQL	s-t TGDs, t TGDs	Datalog, Disjunctive Datalog	Datalog, Shy Datalog^E
	Full Recursion	✓	✓	linear, non-mutual	✓	✓	✓	linear, non-mutual	✓	✓	✓	✓
	Arbitrary Join	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Existential Quantification	✓	–	–	–	–	–	–	–	✓	–	✓
	EGDs	✓	–	–	–	–	–	–	–	✓	–	✘
	Negation as Failure	✓	✓	–	✘	✘	✓	–	–	✓	✓	–
Language Extensions	Aggregations	standard, monotonic	standard	standard, monotonic	✘	standard	standard, monotonic	standard	standard, monotonic	–	standard	–
	Negation	grounded, stratified	grounded, stratified	–	✘	grounded	grounded, stratified	grounded, stratified	grounded, stratified	–	grounded, stratified	–
	Arithmetic Operators	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓
	Inequalities	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓
Reasoning Tasks	Full Ontological Reasoning	✓	–	–	–	–	–	–	–	✓	✓	✓
	Temporal Reasoning	✓	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘
	Probabilistic Reasoning	✓	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘
	Fuzzy Reasoning	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘
	Query Answering	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Evaluation	Methodology	isomorphic chase	–	semi-naive evaluation	semi-naive evaluation	semi-naive evaluation	semi-naive evaluation	–	semi-naive evaluation	restricted, unrestricted, 1-parallel skolem, fresh-null chase	unrestricted skolem chase	parsimonious chase
	Execution Model	streaming, pull-based	–	materialization, push-based	materialization, push-based	materialization, push-based	materialization, push-based	materialization, push-based	–	materialization	materialization, push-based, semi-naive	materialization, push-based, semi-naive
Parallelism	Parallel Evaluation	✘	✘	✓	✓	✓	✘	✓	✓	✘	✘	✘
	Type of Parallelism	–	–	shared-nothing, multi-thread, intra-query	shared-nothing, multi-thread, intra-query	shared-memory, multi-thread, inter-query	–	shared-nothing, multi-thread, intra-query	–	–	–	–
	System Integration	–	–	Apache Spark	Apache Flink	–	–	Apache Spark, MyriaX	Apache Spark	–	–	–
Optimization Techniques	Query Reformulation	✓	–	✓	–	✓	✓	✓	–	✓	✓	✓
	Magic Sets	✘	✘	✘	✘	✘	✓	✘	–	✘	✓	✓
	Other Optimizations	harmful join elimination, routing strategies	–	input partitioning, decomposable programs	operators lifespan extended, loop-invariant datasets	partial aggregates, cache existence checking	✓	locality-aware algebra redundancy removal	decomposable plan	–	weak constraints	chase resumption
Additional Information	Initial Release	2018	2016	2016	2020	2022	2013	2012	2017	2013	2003	2012
	Current Status	Supported	Supported	Abandoned	Supported	Supported	Abandoned	Abandoned	Supported	Supported	Supported	Supported

- *ASTRO* [39]: Datalog-based declarative query engine for advanced stream reasoning that supports aggregates in the presence of recursive queries.
- *Circuitree* [42]: Datalog reasoner developed to generate *zero-knowledge proofs* to prove that a certain conclusion follows from a Datalog ruleset and encrypted input data.
- *LogicBlox* [5]: system built on *LogiQL*, a declarative language based on Datalog, that aims to reduce the complexity of software development for modern applications which enhance and automate decision-making.
- *Ontotext GraphDB* [45]: highly efficient and robust graph database with RDF and SPARQL support that can perform scalable reasoning and query evaluation.
- *SociaLite* [58]: system that employs a high-level graph query language based on Datalog for large-scale and efficient graph and social network analysis.
- *Stardog* [62]: enterprise knowledge graph platform with RDF and SPARQL support and an inference engine to efficiently perform ontological reasoning and query answering tasks.

The above systems were selected by taking into account the overall reasoning and query answering capabilities offered, as well as their relevance in the current literature (i.e., the existence of dedicated papers, their presence in third-party benchmarks and comparisons, etc.). Each system is categorized with respect to the research area it originally belonged to.

Finally, regarding the table itself, note that a cell with: (i) a green check mark indicates that a system supports that feature, (ii) a red X mark symbolizes that a feature that could be integrated is not supported, and (iii) a red hyphen represents the fact that a feature cannot be supported by a system due to external causes, such as another unsupported feature that is mandatory for the current one (e.g., ontological reasoning cannot be achieved without existential quantification), or that we were not able to determine whether such a feature is fully integrated into the system. For readability purposes, the table is split in two images.

Results of the Comparisons. Among the other compared reasoners, we were able to confirm only for seven of them that the interplay between recursion and existential quantification in TGDs is effectively managed. These systems, namely Llunatic, DLV, DLV$^{\exists}$, Graal, RDFox, GraphDB and Stardog, are thus able to perform full ontological reasoning and to address settings such as Scenario 1. Of the above ones, only Llunatic and RDFox also support EGDs to tackle use cases such as Scenario 2. Similarly, features like aggregations, negations and arithmetic operators, required to address settings like Scenario 3, are only supported by BigDatalog, DeALS, RaDlog, RDFox, RecStep and



Original Research Area		Query Answering				Query Reformulation	Program Analysis	Data Access	Stream Reasoning	Business and Industry				
		E	Graal	RDFox	OWL2DLV	RecStep	PDQ	MASTRO	ASTRO	Circuitree	LogicBlox	Ontotext GraphDB	SociaLite	Stardog
Declarative Language	Supported Languages	s-t TGDs, t TGDs	Datalog, Datalog+/-	Datalog	Datalog	Datalog	s-t TGDs, t TGDs	SPARQL under OWL2, DL-Lite_A,id	Datalog	Datalog	Datalog, LogiQL	SPARQL	Datalog	SPARQL under OWL2
	Full Recursion	–	✓	✓	✓	✓	✓	✓	✓	bounded	✓	✓	linear, non-mutual	✓
	Arbitrary Join	–	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Existential Quantification	✓	✓	✓	–	–	✓	✓	–	–	second-order	✓	–	✓
	EGDs	✓	✗	✓	–	–	✓	–	–	–	–	–	–	–
	Negation as Failure	–	✓	✓	✓	✓	–	✓	✓	✗	✓	✓	✓	✓
Language Extensions	Aggregations	–	–	standard	standard	standard, monotonic	–	–	standard, monotonic	–	standard	standard	standard, monotonic	standard
	Negation	–	–	grounded, stratified	grounded, stratified	grounded, stratified	–	grounded	grounded, stratified	✗	grounded, stratified	grounded	grounded, stratified	grounded
	Arithmetic Operators	–	–	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓
	Inequalities	–	–	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Reasoning Tasks	Full Ontological Reasoning	–	–	✓	–	–	–	✓	–	–	–	✓	–	✓
	Temporal Reasoning	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	Probabilistic Reasoning	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	Fuzzy Reasoning	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	Query Answering	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Evaluation	Methodology	paramodulation	restricted chase	restricted, unrestricted skolem chase	semi-naive evaluation	semi-naive evaluation	restricted chase	unfolding	semi-naive evaluation	naive evaluation	restricted chase	–	–	–
	Execution Model	–	forward-chaining	materialization, push-based, semi-naive	materialization, push-based	materialization, push-based	materialization, push-based	–	materialization, push-based	materialization, push-based	–	materialization, forward-chaining, backward-chaining, hybrid-chaining	–	lazy, late-binding, at query time
Parallelism	Parallel Evaluation	✗	✗	✓	✗	✓	✗	✓	✗	✗	✓	✗	✓	✗
	Type of Parallelism	–	–	shared-memory, multi-thread, intra-query	–	–	–	–	–	–	shared-memory, multi-thread, intra-query	–	shared-memory, multi-thread, intra-query	–
	System Integration	–	–	–	–	QuickStep	–	–	–	–	–	–	–	–
Optimization Techniques	Query Reformulation	–	✓	✓	✓	✓	✓	✓	✓	–	✓	–	–	✓
	Magic Sets	✗	✗	–	✓	✗	–	–	✓	–	✗	✗	✗	✗
	Other Optimizations	–	rule selection	–	pruning strategy	unified IDB eval, fast deduplication	parallel chasing	partial evaluation	dynamic query optimizer	–	–	language tags caching and filtering	–	–
Additional Information	Initial Release	2013	2012	2014	2019	2018	2015	2010	2018	2022	2012	2015	2013	2012
	Current Status	Abandoned	Supported	Supported	Supported	Supported	Abandoned	Supported	Supported	Supported	Supported	Supported	Abandoned	Supported

Figure 4: Table comparing systems with reasoning capabilities according to their features.

ASTRO. In fact, from our analysis RDFox appears to be the most versatile reasoner, being able to effectively perform real-world ontological reasoning and graph navigation tasks while supporting multiple features and extensions of practical utility required to tackle scenarios such as Scenario 1- 3. However, to the best of our knowledge, such a system does not exhibit probabilistic and temporal reasoning capabilities, and consequently it cannot address settings such as Scenario 4 and Scenario 5, respectively. From this investigation, Vadalog emerges among the state-of-the-art reasoners in the current literature, as it is the only one covering all the requirements, both in terms of feature support and overall reasoning capabilities, to tackle complex use cases in the economic and financial realm. At the same time, our system still requires further development to address some remaining limitations, such as the absence of a parallel evaluation for the reasoning task, as well as additional Datalog optimizations techniques like magic sets that are yet to be incorporated.

5 CONCLUSION

The newly-sparked interest of the economic and financial sector towards automated reasoning methodologies to address real-world scenarios is undermined by its unfamiliarity for the very diversified and technical offer of languages and systems in the database and AI field. Driven by this, we offered a guidance for new business players, describing the main features of the Vadalog reasoner we integrated to solve relevant financial problems. We also investigated how our system ranks when compared with the other top-performing reasoners in the literature, showing its high feature support and state-of-the-art reasoning capabilities.

ACKNOWLEDGMENTS

This work was supported by the Vienna Science and Technology Fund (WWTF) grant VRG18-013.

REFERENCES

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

[2] Carlo Allocca, Francesco Calimeri, Cristina Civili, Roberta Costabile, Bernardo Cuteri, Alessio Fiorentino, Davide Fuscà, Stefano Germano, Giovanni Laboccetta, Marco Manna, Simona Perri, Kristian Reale, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari. 2019. Large-Scale Reasoning on Expressive Horn Ontologies. In *Datalog (CEUR Workshop Proceedings)*, Vol. 2368. CEUR-WS.org, 10–21.

[3] Mario Alviano and Andreas Pieris (Eds.). 2022. *Proceedings of the 4th International Workshop on the Resurgence of Datalog in Academia and Industry (Datalog-2.0 2022) co-located with the 16th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2022), Genova-Nervi, Italy, September 5, 2022.* CEUR Workshop Proceedings, Vol. 3203. CEUR-WS.org.

[4] Debra Amidon, Piero Formica, and Eunka Mercier-Laurent. 2005. *Knowledge economics: emerging principles, practices and policies.* Tartu University Press.

[5] 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 Conference.* ACM, 1371–1382.

[6] Paolo Atzeni, Teodoro Baldazzi, Luigi Bellomarini, and Emanuel Sallinger. 2022. iWarded: A Versatile Generator to Benchmark Warded Datalog+/− Reasoning. In *Rules and Reasoning: 6th International Joint Conference on Rules and Reasoning, RuleML+ RR 2022, Berlin, Germany, September 26–28, 2022, Proceedings.* Springer, 113–129.

[7] Paolo Atzeni, Luigi Bellomarini, Michela Iezzi, Emanuel Sallinger, and Adriano Vlad. 2020. Augmenting Logic-based Knowledge Graphs: The Case of Company Graphs. In *KR4L@ECAI (CEUR Workshop Proceedings)*, Vol. 3020. CEUR-WS.org, 22–27.

[8] Paolo Atzeni, Luigi Bellomarini, Michela Iezzi, Emanuel Sallinger, and Adriano Vlad. 2020. Weaving Enterprise Knowledge Graphs: The Case of Company Ownership Graphs. In *EDBT.* OpenProceedings.org, 555–566.

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

[10] 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 (Lecture Notes in Computer Science)*, Vol. 9202. Springer, 328–344.

[11] Teodoro Baldazzi and Paolo Atzeni. 2022. Warded Datalog+/- Reasoning in Financial Settings with Harmful Joins. In *EDBT/ICDT Workshops (CEUR Workshop Proceedings)*, Vol. 3135. CEUR-WS.org.

[12] Teodoro Baldazzi, Luigi Bellomarini, Marco Favorito, and Emanuel Sallinger. 2022. On the Relationship between Shy and Warded Datalog+/-. *CoRR* abs/2202.06285 (2022).

[13] Teodoro Baldazzi, Luigi Bellomarini, Emanuel Sallinger, and Paolo Atzeni. 2021. Eliminating Harmful Joins in Warded Datalog+/-. In *RuleML+RR (Lecture Notes in Computer Science)*, Vol. 12851. Springer, 267–275.

[14] Teodoro Baldazzi, Luigi Bellomarini, Emanuel Sallinger, and Paolo Atzeni. 2022. Reasoning in Warded Datalog+/- with Harmful Joins. In *SEBD (CEUR Workshop Proceedings)*, Vol. 3194. CEUR-WS.org, 292–299.

[15] Teodoro Baldazzi, Davide Benedetto, Matteo Brandetti, Adriano Vlad, and Luigi Bellomarini. 2022. Heuristic-based Reasoning on Financial Knowledge Graphs. In *EDBT/ICDT Workshops (CEUR Workshop Proceedings)*, Vol. 3135. CEUR-WS.org.

[16] Teodoro Baldazzi, Davide Benedetto, Matteo Brandetti, Adriano Vlad, Luigi Bellomarini, and Emanuel Sallinger. 2022. Datalog-based Reasoning with Heuristics over Knowledge Graphs. In *Datalog (CEUR Workshop Proceedings)*, Vol. 3203. CEUR-WS.org, 114–126.

[17] Pablo Barceló and Reinhard Pichler (Eds.). 2012. *Datalog in Academia and Industry - Second International Workshop, Datalog 2.0, Vienna, Austria, September 11-13, 2012. Proceedings.* Lecture Notes in Computer Science, Vol. 7494. Springer.

[18] Luigi Bellomarini, Lorenzo Bencivelli, Claudia Biancotti, Livia Blasi, Francesco Paolo Conteduca, Andrea Gentili, Rosario Laurendi, Davide Magnanimi, Michele Savini Zangrandi, Flavia Tonelli, Stefano Ceri, Davide Benedetto, Markus Nissl, and Emanuel Sallinger. 2022. Reasoning on company takeovers: From tactic to strategy. *Data Knowl. Eng.* 141 (2022), 102073.

[19] Luigi Bellomarini, Marco Benedetti, Andrea Gentili, Rosario Laurendi, Davide Magnanimi, Antonio Muci, and Emanuel Sallinger. 2020. COVID-19 and Company Knowledge Graphs: Assessing Golden Powers and Economic Impact of Selective Lockdown via AI Reasoning. *CoRR* abs/2004.10119 (2020).

[20] Luigi Bellomarini, Davide Benedetto, Matteo Brandetti, and Emanuel Sallinger. 2022. Exploiting the Power of Equality-generating Dependencies in Ontological Reasoning. *Proc. VLDB Endow.* 15, 13 (2022), 3976 – 3988.

[21] Luigi Bellomarini, Davide Benedetto, Georg Gottlob, and Emanuel Sallinger. 2022. Vadalog: A modern architecture for automated reasoning with large knowledge graphs. *Inf. Syst.* 105 (2022), 101528.

[22] Luigi Bellomarini, Davide Benedetto, Eleonora Laurenza, and Emanuel Sallinger. 2023. A Framework for Probabilistic Reasoning on Knowledge Graphs. In *International Conference on Soft Methods in Probability and Statistics.* Springer, 48–56.

[23] Luigi Bellomarini, Livia Blasi, Markus Nissl, and Emanuel Sallinger. 2022. The Temporal Vadalog System. In *Rules and Reasoning: 6th International Joint Conference on Rules and Reasoning, RuleML+ RR 2022, Berlin, Germany, September 26–28, 2022, Proceedings.* Springer, 130–145.

[24] Luigi Bellomarini, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. 2018. Swift Logic for Big Data and Knowledge Graphs - Overview of Requirements, Language, and System. In *SOFSEM (Lecture Notes in Computer Science)*, Vol. 10706. Springer, 3–16.

[25] Luigi Bellomarini, Eleonora Laurenza, and Emanuel Sallinger. 2020. Rule-based Anti-Money Laundering in Financial Intelligence Units: Experience and Vision. In *RuleML+RR (Supplement) (CEUR Workshop Proceedings)*, Vol. 2644. CEUR-WS.org, 133–144.

[26] Luigi Bellomarini, Eleonora Laurenza, Emanuel Sallinger, and Evgeny Sherkhonov. 2020. Reasoning Under Uncertainty in Knowledge Graphs. In *RuleML+RR (Lecture Notes in Computer Science)*, Vol. 12173. Springer, 131–139.

[27] Luigi Bellomarini, Eleonora Laurenza, Emanuel Sallinger, and Evgeny Sherkhonov. 2022. Swift Markov Logic for Probabilistic Reasoning on Knowledge Graphs. *CoRR* abs/2210.00283 (2022).

[28] Luigi Bellomarini, Davide Magnanimi, Markus Nissl, and Emanuel Sallinger. 2020. Neither in the Programs Nor in the Data: Mining the Hidden Financial Knowledge with Knowledge Graphs and Reasoning. In *MIDAS@PKDD/ECML (Lecture Notes in Computer Science)*, Vol. 12591. Springer, 119–134.

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

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

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

[32] Sebastian Brandt, Elem Güzel Kalayci, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. 2018. Querying Log Data with Metric Temporal Logic. *J. Artif. Intell. Res.* 62 (2018), 829–877.

[33] 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.

[34] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. 2012. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Semant.* 14 (2012), 57–83.

[35] 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.* IEEE Computer Society, 228–242.

[36] Andrea Calì, Georg Gottlob, Giorgio Orsi, and Andreas Pieris. 2012. On the Interaction of Existential Rules and Equality Constraints in Ontology Querying. In *Correct Reasoning (Lecture Notes in Computer Science)*, Vol. 7265. Springer, 117–133.

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

[38] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. 2011. The MASTRO system for ontology-based data access. *Semantic Web* 2, 1 (2011), 43–53.

[39] Ariyam Das, Sahil M. Gandhi, and Carlo Zaniolo. 2018. ASTRO: A Datalog System for Advanced Stream Reasoning. In *CIKM.* ACM, 1863–1866.

[40] Zhiwei Fan, Jianqiao Zhu, Zuyu Zhang, Aws Albarghouthi, Paraschos Koutris, and Jignesh M. Patel. 2019. Scaling-Up In-Memory Datalog Processing: Observations and Techniques. *Proc. VLDB Endow.* 12, 6 (2019), 695–708.

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

[42] Tom Godden, Ruben de Smet, Christophe Debruyne, Thibaut Vandervelden, Kris Steenhaut, and An Braeken. 2022. Circuitree: A Datalog Reasoner in Zero-Knowledge. *IEEE Access* 10 (2022), 21384–21396.

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

[44] Goetz Graefe and William J. McKenna. 1993. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *ICDE.* IEEE Computer Society, 209–218.

[45] Ontotext GraphDB. 2015. GraphDB Documentation. https://graphdb.ontotext.com/documentation/10.0/index.html. [Online; accessed 02-Dec-2022].

[46] Jiaqi Gu, Yugo H. Watanabe, William A. Mazza, Alexander Shkapsky, Mohan Yang, Ling Ding, and Carlo Zaniolo. 2019. RaSQL: Greater Power and Performance for Big Data Analytics with Recursive-aggregate-SQL on Spark. In *SIGMOD Conference.* ACM, 467–484.

[47] Muhammad Imran, Gábor E. Gévay, and Volker Markl. 2020. Distributed Graph Analytics with Datalog Queries in Flink. In *SFDI/LSGDA@VLDB (Communications in Computer and Information Science)*, Vol. 1281. Springer, 70–83.

[48] Bas Ketsman and Christoph Koch. 2020. Datalog with Negation and Monotonicity. In *ICDT (LIPIcs)*, Vol. 155. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 19:1–19:18.

[49] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. 2012. Efficiently Computable Datalog∃ Programs. In *KR.* AAAI Press.

[50] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. 2019. Fast Query Answering over Existential Rules. *ACM Trans. Comput. Log.* 20, 2 (2019), 12:1–12:48.

[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. Log.* 7, 3 (2006), 499–562.

[52] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. 1979. Testing Implications of Data Dependencies. *ACM Trans. Database Syst.* 4, 4 (1979), 455–469.

[53] Myria. 2012. Myria Documentation. http://myria.cs.washington.edu/docs/myrial.html. [Online; accessed 02-Dec-2022].

[54] Maya Ramanath and Themis Palpanas (Eds.). 2022. *Proceedings of the Workshops of the EDBT/ICDT 2022 Joint Conference, Edinburgh, UK, March 29, 2022.* CEUR Workshop Proceedings, Vol. 3135. CEUR-WS.org.

[55] RDFox. 2014. RDFox Documentation. https://docs.oxfordsemantic.tech/index.html. [Online; accessed 02-Dec-2022].

[56] Mattia Scaccia, Ilaria Stocchi, and Luigi Bellomarini. 2022. Neurosymbolic Reasoning: Building Neural Networks Using Datalog. In *EDBT/ICDT Workshops (CEUR Workshop Proceedings)*, Vol. 3135. CEUR-WS.org.

[57] Stephan Schulz. 2013. System Description: E 1.8. In *LPAR (Lecture Notes in Computer Science)*, Vol. 8312. Springer, 735–743.

[58] Jiwon Seo, Stephen Guo, and Monica S. Lam. 2013. SociaLite: Datalog extensions for efficient social network analysis. In *ICDE*. IEEE Computer Society, 278–289.

[59] Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. 2016. Big Data Analytics with Datalog Queries on Spark. In *SIGMOD Conference*. ACM, 1135–1149.

[60] Alexander Shkapsky, Mohan Yang, and Carlo Zaniolo. 2015. Optimizing recursive queries with monotonic aggregates in DeALS. In *ICDE*. IEEE Computer Society, 867–878.

[61] Alexander Shkapsky, Kai Zeng, and Carlo Zaniolo. 2013. Graph Queries in a Next-Generation Datalog System. *Proc. VLDB Endow.* 6, 12 (2013), 1258–1261.

[62] Stardog. 2012. Stardog Documentation. https://docs.stardog.com/. [Online; accessed 02-Dec-2022].

[63] Vaticle TypeDB. 2016. Vaticle TypeDB Documentation. https://docs.vaticle.com/docs/general/introduction. [Online; accessed 02-Dec-2022].

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

[65] Adriano Vlad, Sahar Vahdati, Mojtaba Nayyeri, Luigi Bellomarini, and Emanuel Sallinger. 2022. Towards Hybrid Logic-based and Embedding-based Reasoning on Financial Knowledge Graphs. In *EDBT/ICDT Workshops (CEUR Workshop Proceedings)*, Vol. 3135. CEUR-WS.org.

[66] Jiacheng Wu, Jin Wang, and Carlo Zaniolo. 2022. Optimizing Parallel Recursive Datalog Evaluation on Multicore Machines. In *SIGMOD Conference*. ACM, 1433–1446.