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 speaking, an ontological reasoning task consists in answering a

\[ \varphi(x, y) \rightarrow \varphi(x, z) \]

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

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\(^3\) 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 \( T \) 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 \( v \) (technically, labelled nulls) to satisfy existential quantification. A TGD \( \sigma : q(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{z}) \) is satisfied by \( D \) if, whenever a homomorphism \( \theta \) occurs such that \( \theta(\varphi(\bar{x}, \bar{y})) \subseteq D \), there exists an extension \( \theta' \) of \( \theta \) (i.e., \( \theta \subseteq \theta' \)) such that \( \theta'(\psi(\bar{x}, \bar{z})) \subseteq D \). In other words, in the standard (namely, oblivious) chase, \( \sigma \) is applicable to \( D \) if there exists a \( \theta \) such that \( \theta(\varphi(\bar{x}, \bar{y})) \subseteq D \). When applied, \( \sigma \) generates a new fact \( \theta'(\psi(\bar{x}, \bar{z})) \) that enriches \( D \), if not already present, where \( \theta' \) extends \( \theta \) by mapping the variables of \( \bar{z} \) (if not empty) to new labelled nulls named in a lexicographical order. Similarly, an EGD \( \eta : q(\bar{x}) \rightarrow x_i = x_j \) is satisfied if, for each \( \theta(\varphi(\bar{x})) \), \( \theta(x_i) = \theta(x_j) \). Thus, \( \eta \) is applicable to \( D \) if there exists a \( \theta \) such that \( \theta(\varphi(\bar{x})) \subseteq D \) and \( \theta(x_i) \neq \theta(x_j) \). When applied, \( \eta \) checks if \( x_i \) and \( x_j \) are constants, in which case it fails; otherwise, it performs a unification, replacing each occurrence of \( \theta(x_i) \) with \( \theta(x_j) \). When \( \theta(\psi(\bar{x}, \bar{z})) \subseteq D \), 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.

783
**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.

\[
\text{Controls}(x, y), \text{SH}(x, s) \rightarrow \text{SH}(y, s)
\]

\[
\text{SH}(x, s), \text{SH}(y, s) \rightarrow \text{StrongLink}(x, y)
\]

\[
\text{StrongLink}(x, y) \rightarrow 2 \times \text{SH}(x, s), \text{SH}(y, s)
\]

For each company \(x\) there exists a shareholder \((\text{SH}\ s)\) (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\(^2\) 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\(^2\) family [9, 12, 34, 35, 49]. They apply to Datalog\(^2\) 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\(^2\) languages, their syntactic containment, and reasoning data complexity.

![Datalog\(^2\) languages and data complexity](image)

Specifically, we employed Warded Datalog\(^2\) [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 \(v\) 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 \(\sigma : \phi(\bar{x}, \bar{z}) \rightarrow \psi(\bar{x}, \bar{z})\) and a homomorphism \(\theta\), \(\sigma\) 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 \(\Sigma\) 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 \(\Sigma\), 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. \(\text{function ISOMORPHIC\_CHASE\_EVAL}(D, \Sigma, Q)\)
2. \(\text{chase} = D\) \(\quad\rightarrow\) chase instance is initialized
3. \(H = \text{init}\_\text{structure}(D, \Sigma)\) \(\quad\rightarrow\) applicable homomorphisms
4. while not \(H\) is empty do
5. \(h = H.\text{poll\_first}()\)\(\quad\rightarrow\) apply current homomorphism
6. \(p = H.\text{apply}(\Sigma, h)\) \(\quad\rightarrow\) apply current homomorphism
7. if check_isomorphism\(p,\text{chase}\) then
8. \(\text{chase} = \text{chase} \cup \{p\}\)
9. \(H = H.\text{update}\_\text{structure}(\Sigma, p)\)
10. return answer\(Q, \text{chase}\)

With reference to Scenario 1, let us consider the database instance \(D = \{\text{Company}(H\v), \text{Company}(I\v), \text{Controls}(H\v, I\v)\}\) and the query \(Q: \{\text{what are all the entailed StrongLinks?}\}\) as ontological reasoning task. Employing the isomorphic chase, we first generate \(\text{SH}(H\v, I\v)\) by activating rule 1 from the fact \(\text{Company}(H\v)\). Next, we obtain \(\text{SH}(I\v, \bigcup\v)\) from rule 2, and consequently \(\text{StrongLink}(H\v, H\v), \text{StrongLink}(H\v, I\v), \text{StrongLink}(I\v, H\v)\) and \(\text{StrongLink}(I\v, I\v)\) via the join in rule 3: these facts are the result for \(Q\). Note that, by activating now rule 1 on \(\text{Company}(I\v)\), we would generate \(\text{SH}(I\v, \bigcup\v)\). Similarly, from \(\text{StrongLink}(I\v, I\v)\) and rule 4 we would obtain \(\text{SH}(H\v, \bigcup\v)\) and \(\text{SH}(I\v, \bigcup\v)\): this would in turn lead to new activations of rule 3 and the generation of an infinite set \(\bigcup\v\) of \(\text{SH}(H\v, \bigcup\v)\) and \(\text{SH}(I\v, \bigcup\v)\) 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 pruning 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.

\[
\begin{align*}
&\text{FinInt}(x), \text{Own}(p, x, w), w > 0.3 \rightarrow KP(p, x) \quad (1) \\
&\text{FinInt}(x), \text{NPL}(x) \rightarrow \exists f. \text{Default}(x, f, f) \quad (2) \\
&\text{Default}(x_1, f_1, f_1), \text{Exp}(x_1, x_2) \rightarrow \exists f_2. \text{Default}(x_2, f_1, f_2) \quad (3) \\
&\text{Default}(x, f_1, f_2), KP(p, x) \rightarrow \exists i. \text{Inv}(p, x, i) \quad (4) \\
&\text{Inv}(p_1, x_1, i_1), \text{Inv}(p_2, x_2, i_2), \text{Exp}(x_1, x_2) \rightarrow i_1 = i_2 \quad (5) \\
&\text{Inv}(p_1, x_1, i_1), \text{Inv}(p_2, x_2, i_2), \text{Exp}(x_1, x_2) \rightarrow i_1 = i_2 \quad (6)
\end{align*}
\]

An individual $p$ is a key person (KP) of a financial intermediary (FinInt) 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 $f_2$, 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 $x_1$, 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\textsuperscript{8}. 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\textsuperscript{8}. 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, "harm" 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.

\[
\begin{align*}
&\text{Own}(c_1, c_2, s) \rightarrow \text{MC}(c_1, c_2, s) \quad (1) \\
&\text{MC}(c_1, c_2, s_1), \text{Own}(c_2, c_3, s_2) \rightarrow \text{MC}(c_1, c_3, s_1 + s_2) \quad (2) \\
&\text{MC}(c_1, c_2, s), ts = \text{msum}(s), ts \geq 0.2 \rightarrow C_l(c_1, c_2) \quad (3) \\
&C_l(c_1, c_1), C_l(c_1, c_2), \neg C_l(c_1, c_2), c_1 \neq c_2 \rightarrow C_l(c_1, c_2) \quad (4) \\
&C_l(c_1, c_2) \rightarrow \text{Cl}(c_1, c_2) \quad (5) \\
&C_l(c_1, c_2) \rightarrow \text{Cl}(c_1, c_2) \quad (6)
\end{align*}
\]

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 (MC) (rule 1). If $c_1$ and $c_2$ are MC with a share $s_1$ and $c_2$ owns in return a share $s_2$ of a company $c_3$, then also $c_2$ and $c_3$ are MC with a share of $s_1 + s_2$ (rule 2). If the sum of all the partial shares 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 $∀x. (\varphi(x), z = \text{magg}(\varphi(x), \epsilon)) → ψ(\varphi(x), z)$, where $\text{magg}$ is the name of an aggregation function, $x ∈ \mathcal{X}$, and $\epsilon ≤ x$ is an $n$-uple of group-by arguments, $\epsilon ≤ x$ (with $\epsilon ≤ x$) 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 monotonic 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 $\epsilon$, the minimum (maximum) value of $x$ 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.

\[
\begin{align*}
&0.9 \equiv \text{LenderT}(x, y), \text{RR}(y, z) \rightarrow 3w. \text{Guarantee}(z, x, e) \quad (1) \\
&0.8 \equiv \text{LenderT}(x, y), \text{LenderC}(y, z) \rightarrow \text{LenderT}(x, z) \quad (2) \\
&0.7 \equiv \text{Contract}(x, y, z), \text{Exp}(y, w) \rightarrow \text{Contract}(z, w, x) \quad (3) \\
&\text{Contract}(x, y, z), \text{RR}(w, y) \rightarrow \text{LenderT}(x, w) \quad (4)
\end{align*}
\]
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 \( \{ (f, P(f)) \} \), where \( f \) is a fact and \( P(f) \) is its marginal probability, i.e., the probability for \( f \) 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.

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

If in an interval in the past (denoted by \( \Phi_{0,1} \)) 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 \( \Phi_{0,1} \)) 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 \( \Sigma \) 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 \( Q(x) \leftarrow \chi(x) \) into an atomic query by first updating \( \Sigma = \Sigma \cup \{ \rho_Q \} \), where \( \rho_Q : \chi(x) \leftarrow \psi(x, z) \) and \( \chi \) is an invented predicate, and then creating the atomic query \( Q(x) \leftarrow \chi(x) \); (ii) a logic optimizer performs logic transformations and rewrites 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.](image)

**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

$$\text{SameInv}(p_1, p_2) \leftarrow \text{Inv}(p_1, _, i), \text{Inv}(p_2, _, i) \quad (Q)$$

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.

- **Vatile TypeDB** [63]: strongly-typed database that enables type and rule inference, as well as entity-relationship data modelling via its proprietary language TypeQL.
- **BigDataSQL [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.
- **DCDatolog [46]**: 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.
- **DLV²** [50]: system for query answering over programs in the Shy fragment of the Datalog² 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** [46]: 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.
<table>
<thead>
<tr>
<th>Original Research Area</th>
<th>Knowledge Graphs</th>
<th>Execution Model</th>
<th>Big Data</th>
<th>Query Answering</th>
<th>Other Optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Languages</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLV</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLV^E</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Datalog</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Datalog^E</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graal</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LogicBlox</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyriaX</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Myria</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stardog</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SparK</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cog</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASTRO</td>
<td>[39]: Datalog-based declarative query engine for advanced stream reasoning that supports aggregates in the presence of recursive queries.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Circuitree</td>
<td>[42]: Datalog reasoner developed to generate zero-knowledge proofs to prove that a certain conclusion follows from a Datalog ruleset and encoded input data.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LogicBlox</td>
<td>[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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ontotext GraphDB</td>
<td>[45]: Highly efficient and robust graph database with RDF and SPARQL support that can perform scalable reasoning and query evaluation.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Socialite</td>
<td>[58]: System that employs a high-level graph query language based on Datalog for large-scale and efficient graph and social network analysis.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stardog</td>
<td>[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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 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^E, 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
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.