ARTICLE IN PRESS

Future Generation Computer Systems xxx (xxxx) xxx

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Data science with Vadalog: Knowledge Graphs with machine learning and reasoning in practice

Luigi Bellomarini [b,a], Ruslan R. Fayzrakhmanov [a], Georg Gottlob [a,c,*], Andrey Kravchenko [a], Eleonora Laurenza [b], Yavor Nenov [a], Stéphane Reissfelder [a], Emanuel Sallinger [c,a], Evgeny Sherkhonov [a], Sahar Vahdati [a], Lianlong Wu [a]

[a] University of Oxford, United Kingdom
[b] Banca d'Italia, Italy
[c] TU Wien, Austria

ARTICLE INFO

Article history:
Received 24 May 2019
Received in revised form 15 July 2021
Accepted 18 October 2021
Available online xxxx

Keywords:
Knowledge Graphs
Data science
Machine learning
Reasoning
Probabilistic reasoning

ABSTRACT

Following the recent successful examples of large technology companies, many modern enterprises seek to build Knowledge Graphs to provide a unified view of corporate knowledge, and to draw deep insights using machine learning and logical reasoning. There is currently a perceived disconnect between the traditional approaches for data science, typically based on machine learning and statistical modeling, and systems for reasoning with domain knowledge. In this paper, we demonstrate how to perform a broad spectrum of data science tasks in a unified Knowledge Graph environment. This includes data wrangling, complex logical and probabilistic reasoning, and machine learning. We base our work on the state-of-the-art Knowledge Graph Management System Vadalog, which delivers highly expressive and efficient logical reasoning and provides seamless integration with modern data science toolkits such as the Jupyter platform. We argue that this is a significant step forward towards practical, holistic data science workflows that combine machine learning and reasoning in data science.

© 2021 Elsevier B.V. All rights reserved.

1. Introduction

Nowadays, organizations and enterprises increasingly depend on intelligent information systems that operationalize corporate data and knowledge in a unified way across system boundaries. Such systems crucially rely on insights produced by data scientists, who use advanced data and graph analytics together with statistical models. The general objective of such intelligence systems is to create predictive actionable knowledge from suitably preprocessed organizational and corporate data by means of data wrangling.

Heterogeneous Data Sources. One of the central challenges for stakeholders dealing with data is incorporating multiple heterogeneous sources of information. The past two decades witnessed an explosion of data resources that have been made publicly and proprietarily available (e.g., DBpedia [1], Wikidata [2]), or extracted from web pages via web data extraction [3]. Such resources originate from data streams collected from internal systems (e.g., Enterprise Resource Planning, Workflow Management, and Supply Chain Management) and external resources (e.g., news, social media feeds and the Common Crawl[1]).

The integration and management of such heterogeneous information is a non-trivial task that presents data scientists with a number of challenges including: handling schema differences as well as heterogeneous data types, managing updates in frequently changing content and structure, knowledge extraction from diverse resources, dealing with the uncertainty of the integrated data, handling data quality and consistency from independent providers, and finding ways of unifying them.

Knowledge Graphs. Following the increasing use by large technology companies such as Google, Amazon, Facebook, and LinkedIn, it is becoming common for enterprises to integrate their internal and external sources of information into a unified *Knowledge Graph*. A Knowledge Graph typically is representable as a set of triples (*subject*, *predicate*, *object*) where *subject*, *object* refer to entities and *predicate* refers to the relation between two entities. Such graph-structured data allow smooth accommodation of changes in their structure.. Examples of data formats include traditional graph data formats (e.g., RDF[2] used by triple stores

* Corresponding author at: University of Oxford, United Kingdom.
E-mail address: georg.gottlob@cs.ox.ac.uk (G. Gottlob).

[1] http://commoncrawl.org/
[2] https://www.w3.org/RDF/

https://doi.org/10.1016/j.future.2021.10.021
0167-739X/© 2021 Elsevier B.V. All rights reserved.

Please cite this article as: L. Bellomarini, R.R. Fayzrakhmanov, G. Gottlob et al., Data science with Vadalog: Knowledge Graphs with machine learning and reasoning in practice, Future Generation Computer Systems (2021), https://doi.org/10.1016/j.future.2021.10.021.

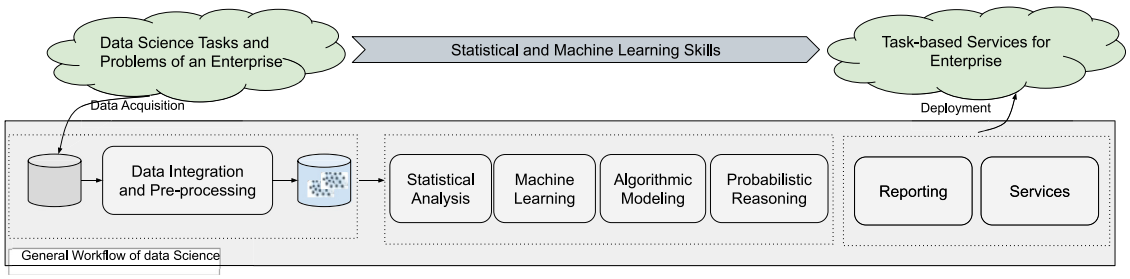


Fig. 1. Data Science Workflow integrated with Machine Learning.

such as GraphDB[3], Property Graphs used by graph databases like Neo4j[4], and JanusGraph[5]) as well as relational or other semi-structured data exposing graph structure.

In this paper, we show how to support the data science workflow using a state-of-the-art Knowledge Graph Management System (KGMS). In our case, this is the Vadalog system, which combines an expressive, scalable Knowledge Graph engine with good support for the data science workflow. The Vadalog system is Oxford's contribution to VADA [4,5], a joint project of the universities of Edinburgh, Manchester, and Oxford.

The Vadalog system is designed to be used in AI-driven applications. Currently, the system fully implements the core language and is in use for a number of industrial and commercial customer-based applications. In [6], we provide performance analysis of the Vadalog system in comparison to other systems namely RDFox [7], LLunatic [8], DLV [9,10], Graal [11], and PDQ [12]. The results show that the Vadalog system outperforms all systems both in synthetic and real-world use cases. Further discussions can be followed in the corresponding research work [6]. A use case where the Vadalog system is used in a financial domain is introduced in [13].

Context and Goal. We reported first work on the overall VADA approach to data wrangling in [14]. The Vadalog system has been covered in detail in the following of its principal aspects.

- The *vision* for a KGMS and the requirements of its components [15].
- The *technical* algorithms and system implementation of the system [6].
- The *use* of the system for building Enterprise AI *architectures* [13].

Further works have been presented on practical combinations of machine learning and logic-based reasoning [16], as well as giving a broad overview of the topic of reasoning in Knowledge Graphs in general [17–19].

What has been missing so far, and is a frequently requested work, is one showing how to *actually build a data science pipeline*, not from the perspective of a system architect (as in [13]), but on how to do this as a *data scientist*.

For many data scientists, building a data science workflow these days *is* synonymous to working with notebook environments such as Jupyter. Such a notebook environment will also be a connecting thread through this paper.

In this paper, we aim at providing a step-by-step guide to Vadalog's support of data science tasks. We introduce the main components with regard to the workflow of data science when combined with machine learning techniques. The services for each step of data science tasks and requirement for a system to support them are shown in Fig. 1. We will discuss these steps in detail in the following section, but want to highlight already here the "big picture" of data science and how a Knowledge Graph-based approach in general, and a Vadalog-based approach in particular, can support the data scientist.

Let us consider an example first. A basic classification model (e.g., a linear classifier) used in a data science workflow typically has no access to complex graph-based concepts (e.g., new parts of the dataset derived through highly recursive exploration of, potentially missing or incomplete, parts of the dataset). One such concrete example that will accompany us as a running example through this paper, is in the context of financial Knowledge Graphs: concepts such as *company control* or *close links* between companies or individuals are essential on the one hand, but also very hard to derive in general — often depending on complex legal regulations. We will use this and similar examples in the context of financial Knowledge Graphs as our running example throughout the paper. For further details on the financial setting, e.g., how to apply this during the COVID-19 induced crisis, we refer to the literature [20].

Going back to the big picture, in summary, Vadalog and KGMSs allow data science processes to leverage such complex concepts through the data science process, e.g., by seamlessly integrating them into statistical models. We will discuss this concrete aspect in more detail in Section 5.2, and show the general support throughout this work, and in particular in the following Section 2.

Organization. Each component of Fig. 1 is discussed in the following sections: Section 2 provides an overview of data science in the context of the Vadalog system. Section 3 provides an overview of the core Vadalog language and system. Section 4 describes the various features of the system within a typical data scientist's workflow in Jupyter. Section 5 demonstrates the system's integration with machine learning and algorithmic modeling on typical use cases. Section 6 describes the support of the system for probabilistic reasoning. Finally, Section 7 provides positioning of the Vadalog system in the database literature and a comparison with related work and systems. The conclusion is given in Section 8.

2. Data science and Vadalog system

The term "data science" has emerged as an interdisciplinary area that, among others, is rooted in the field of knowledge discovery. The field of knowledge discovery from databases (KDD) is sometimes characterized as the application of quantitative statistical methodologies to extract actionable knowledge from data [21]. Predictive modeling and machine learning are increasingly becoming central for any enterprise or ecosystems dealing with data. The predictive power of management systems heavily defines the role of such actionable knowledge in analytics and decision-making processes.

This emerging landscape requires integrated skills for data scientists including data management, statistics, machine learning, and optimization. Considering the characteristics of Big Data, a

[3] http://graphdb.ontotext.com/
[4] https://neo4j.com/
[5] http://janusgraph.org/

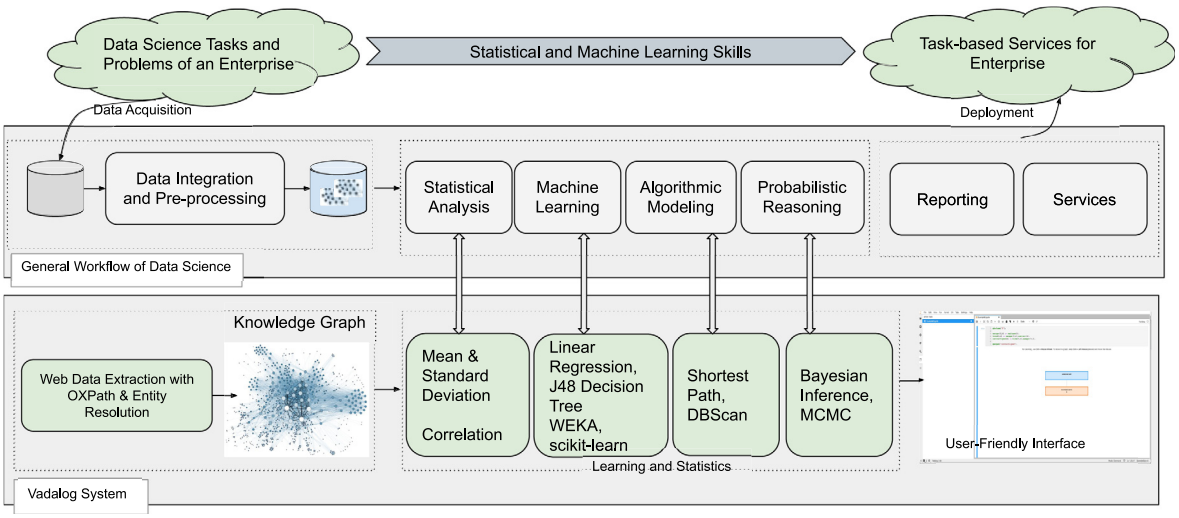


Fig. 2. Data Science Workflow integrated with machine learning in Vadalog System.

solution is to provide expressive and scalable knowledge management systems supporting each of such required aspects. In this section, we first take a close look into the workflow of data science in general. Further, we explain the support for each of the required processes empowered by Vadalog.

2.1. The data science spectrum

In Fig. 2, we align the data science process and required tasks with their support in the Vadalog system for each of them. Despite the different data science tasks in terms of domain and scale, the workflow of such projects can be generalized into five main steps which we list next.

1. **Data Integration and Pre-processing**: Data pre-processing is a crucial step in the data science workflow. Vadalog supports integration of data from heterogeneous data sources, which include relational and graph databases, as well as web data extraction. Once the data is integrated, it typically needs to be pre-processed so that statistics, algorithms, or machine learning models can be applied to it. Vadalog is a suitable tool for pre-processing tasks, which include data cleaning, such as getting rid of the null values, as well as data transformation. Once the data has been pre-processed, a number of approaches can be applied to get further insights.

2. **Statistical Analysis**: Statistics is crucial to help a data scientist get initial insights about the data. Vadalog supports all basic operations for statistical understanding of the data, such as computing the maximum and the minimum, the average, the median, the mode, the variance, and the standard deviation of a dataset.

3. **Machine Learning**: It is often the case that machine learning needs to be applied to the data in its supervised form (such as linear or logistic regression or a decision tree) or in unsupervised form (such as k-means clustering) to make predictions or classifications for previously unseen data, or to get better understanding of the underlying structure of existing data, like in the case of clustering. Vadalog supports the regime of machine learning integration where the features are created and/or imported from Vadalog into an external ML library such as WEKA [22] or Scikit-learn.[6] An ML model such as linear regression or a J48 decision tree [23] is learned and serialized on this feature set, and this model is employed to be applied on newly unseen data also imported from Vadalog. The output prediction or classification produced by the model is then imported back into Vadalog and can be treated as the final output or used in further steps of the data workflow, e.g., for post-classification reasoning. Note that this regime is agnostic to both the model and the ML library used, so we can seamlessly switch between models and libraries.

4. **Algorithmic Modeling**: Another very common case for data processing is when an algorithm needs to be applied to the data. Many popular algorithms, in particular graph algorithms and dynamic programming algorithms, such as Dijkstra's shortest path, DBScan, or knapsack, can be seamlessly integrated in Vadalog. The reason it is particularly convenient to implement graph algorithms in Vadalog is that nodes and arcs can be naturally represented as unary and binary predicates, and recursive steps can also be trivially represented in Vadalog. The same applies to dynamic programming algorithms, as recursive optimality equations underlying these algorithms can be naturally expressed.

5. **Probabilistic Reasoning**: In many cases, the outcome of a process applied to the data is non-deterministic and requires assigning probabilities. This gives the rise for probabilistic reasoning over the data, which is tightly integrated into Vadalog. One of the classic applications of probabilistic reasoning is Bayesian inference, which is based on the fundamental Bayes' theorem, which in its simplest form can be written as $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, where A and B are events and $P(B) \neq 0$. $P(A|B)$ is called posterior probability, and $P(A)$ is called prior probability. We use Bayesian inference to draw posterior probabilities and update probabilities for a collection of hypotheses based on the knowledge drawn from the priors. It is possible to represent priors and do Bayesian inference in Vadalog through probabilistic reasoning.

2.2. Connecting the dots

Knowledge Graphs are today's solution for the unification of heterogeneous data sources. The creation of a coherent Knowledge Graph from multiple sources of unstructured, semi-structured, and structured data is a challenging task that requires techniques from multiple disciplines. In particular, there are various approaches in the field of *information extraction* to automatically extract structured data from plain text [24] (e.g., news and social media feeds) and in the field of *web data extraction* for semi-structured heterogeneous sources [25,26] (e.g., websites with prices and product characteristics). Thus, for example, the

6 https://scikit-learn.org/

news feed item "PayPal buys Hyperwallet for $400M" could result into the structured statement "acquire(PayPal,Hyperwallet)". Information extraction from plain text and web data extraction from semi-structured documents are typically used along with entity resolution to incorporate the extracted information within an existing knowledge graph correctly. *Entity resolution* [27] combines multiple sources of (semi-)structured data that do not share common identifiers. The goal is to identify pairs of entities that refer to the same real-world object and merge them into a single entity. The matching is performed using noisy, semi-identifying information (e.g., names, addresses) and relationships, and employs specialized similarity functions for strings, numbers, and dates, to determine the overall similarity of two entities.

Publicly available datasets are often equipped with ontologies which describe relationships between entities. In such cases *ontological reasoning* needs to be applied to validate whether the results of entity resolution and information extraction violate any of the constraints imposed by the ontology as well as to enrich the data with new information stemming from the newly produced facts. Further note that, unsurprisingly, the use of *machine learning* is pervasive throughout the stages of the data scientist's workflow: from semantically annotating web page elements during web data extraction, through deciding whether entities should be matched during entity resolution, to predicting numerical trends during data analytics over the Knowledge Graph.

Finally, observe that although *uncertainty* is intrinsic to many of the tasks in the data scientist's workflow, it is typically resolved by means of a threshold. For example, during entity resolution, the similarity of the attributes of two entities is typically converted to a probability for the two entities to be the same, and they are matched if the probability exceeds a certain threshold. Similarly, the information extraction stage typically associates output facts with a level of uncertainty stemming from the extraction process, but likewise to the case of entity resolution, the uncertainty is converted into a probability for a fact to hold, and a hard decision is made on whether it should be included or not. One may want to impose levels of uncertainty using business rules to inform the decision of whether and how the Knowledge Graph should be updated. One such rule, for example, could be that public companies are much more likely to acquire private companies than vice-versa (*reverse takeover*). Such rules can be produced by a domain expert or learned from the data using rule learning [15]. Furthermore, instead of ignoring the uncertainty, after it is being used to determine whether to accept a fact or a match, for example, one could alternatively incorporate this uncertainty into the knowledge graph and propagate them into the further stages of data wrangling and data analytics.

To carry out the different stages of the described workflow data scientists need to use and coordinate a number of tools, languages, and technologies: for data access they require tools for web data extraction, various database management systems, triple stores and graph databases; during Knowledge Graph construction they require tools for entity resolution, information extraction, ontological reasoning, and uncertainty management; and during the analysis stage they require tools for graph analytics, machine learning and statistical modeling. The coordination of all these tools can be very challenging.

3. An introduction to Vadalog

This section is for making the paper self-contained. It is partially derived from [6,13,15]. If you are already familiar with the Vadalog system and language, you can safely skip it, the first novel part of this work is found in Section 4.

3.1. The Vadalog core system

We represent the functional architecture of the Vadalog KGMS in Fig. 3. The central point of the Vadalog core system consists of a rule-based reasoning engine. Rules are stored in a repository which is in direct interaction with the reasoning engine and a rule management user interface (e.g., Jupyter). The system is designed to be easily extendable with multiple protocols of communication between different parts via application programming interfaces (APIs). For example, it supports plug-ins from Relational or Graph databases, Resource Description Framework (RDF) stores as well as NoSQL. The detailed discussion about the components of the system can be found in in [15].

3.2. Vadalog core language

The Vadalog system heavily relies on the reasoning engine and its rule-based repository. Rules are written in Datalog$^{\pm}$ language which is an extension of a logic programming language named Datalog [28].

Datalog is a declarative rule-based language. A Datalog program consists of a set of rules and a set of facts (triples in knowledge graphs). Every rule is a *function-free* Horn clause, intuitively a first-order implication of the form `head :- body.`, where `head` is an atom and `body` is a conjunction of atoms `b1, . . . , bn`, with the comma denoting the logical *and*. The `:-` symbol denotes the logical implication $\leftarrow$. The body variables are considered as universally quantified and a program is said to be safe if all head variables also appear in the body.

The use of Datalog appeared in recent years as a common thread across general-purpose systems such as integration, extraction, networking, analysis, security, and computing as a higher level abstraction [29].

Datalog$^{\pm}$ is a family of languages that provide important extensions to Datalog in order to enable ontological reasoning. In particular, Datalog$^{\pm}$ extends Datalog by existential quantifiers in rule heads, as well as by other features, and at the same time restricts its syntax in order to achieve decidability and data tractability; see, e.g., [30,31]. Let us consider the following simple Datalog$^{\pm}$ program:

```
Rules: controls(Z,X) :- company(X).
    company(X) :- controls(X,Y).
Facts: company(ACME). controls(UBA,UBE).
```

The first rule establishes that, for each company `X`, there exists a company `Z`, such that `Z` controls `X`. Then, with the second rule, we make sure that controlling companies are in turn companies. Note that the two rules are mutually recursive, as the result of the former being an input to the latter and vice versa. The rules of a Datalog$^{\pm}$ program can be seen as a set of constraints on a database D, where the data, i.e., the tuples, are considered logical facts. The semantics of a rule is often defined operationally via the CHASE [32], a procedure that enforces database constraints and produces the reasoning results as consequence. Intuitively, for each binding of the body that does not correspond to a binding of the head (i.e., the rule as a logical implication is violated), the chase "triggers" the rule and adds the facts that correspond to the rule head to D. This process possibly involves the creation of new symbols, namely, fresh *labeled* or *marked nulls* for objects introduced via existential quantification.

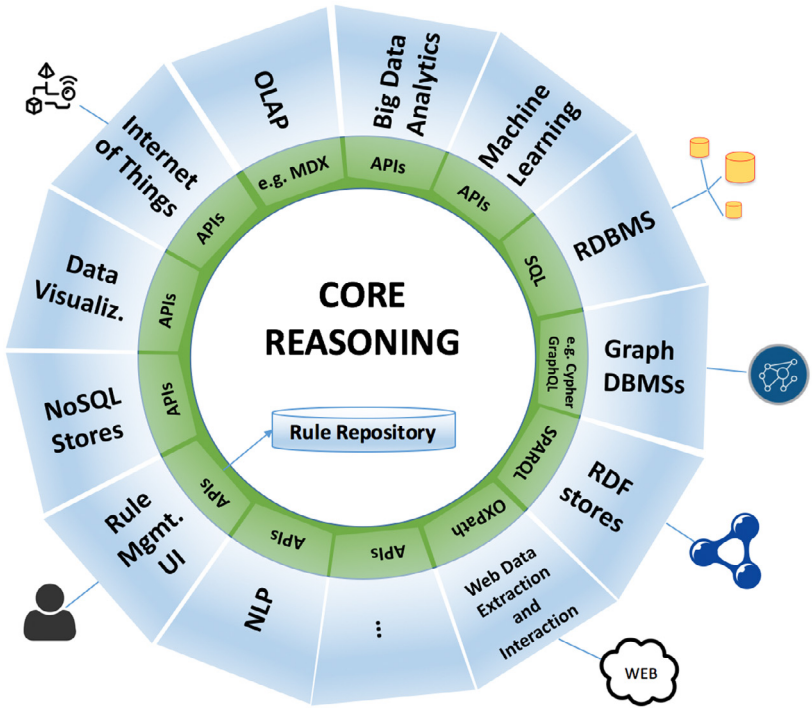


Fig. 3. KGMS Architecture [15].

Starting from the database D composed of the facts in our example, for `company(ACME)`, the chase generates `controls(`Z_1`,ACME)`, `controls(`Z_2, Z_1`)` with the first rule; `company(`Z_1`)`, `company(`Z_2`)`, and so on with the second rule. Observe that Z_1, Z_2, Z_3 are fresh symbols denoting the labeled nulls and stand for arbitrary companies, whose identity is unknown.

Warded Datalog$^{\pm}$ is the specific Datalog extension adopted in the core of the Vadalog system [33,34]. It has been shown that the computational complexity of logical reasoning tasks originate from the way in which the labeled nulls are allowed to propagate in the chase. We have seen, for instance, that the nulls produced by the first existential rule in our example, propagate to the second and finally back to the first one. If arbitrary propagation is allowed, reasoning is undecidable. Specific syntactic conditions can be adopted in order to constrain the positions where nulls can appear, achieving decidability or even tractability in this manner. In Warded Datalog$^{\pm}$, the syntactic restrictions are mild to the point they almost do not hamper the expressive power of the language, while achieving tractability at the same time.

Let us give an intuition of such "wardedness" property. Specific body variables are flagged as *dangerous* if they are bound to contain labeled nulls and also appear in the head of a rule. In other words, this "danger" encodes certain propagation of nulls throughout the reasoning process. Wardedness constrains that all the dangerous variables in a rule appear in one single body atom, namely the *ward*. Moreover, the ward must respect the property of being able to share variables with other body atoms only under the condition they are *harmless*, which means they cannot contain labeled nulls. Consider for example the following set of rules:

```
∃Z controls(Z,X) :- company(X).
company(X) :- controls(X,Y).
```

The variable `X` in the body of the second rule is *dangerous*. Considering for example the database $D = \{$`company(ACME)`$\}$, we have that the chase applies the first rule and generates `controls(`Z_1`,ACME)`, where Z_1 is a witness for the existentially quantified variable `Z`; then the second rule is applied, and the variable `X` is unified with Z_1 that is propagated to the head `company(X)`.

Observe that, throughout this paper, we will mix the *logical* notation adopted in formal contexts and the "code"-like Prolog-inspired notation, that is used in systems, such as the Vadalog system. Consider now the following set of rules:

```
∃Z holding(Z,X) :- company(X).
business(X,Z) :- holding(X,Y), business(Y,Z).
partners(Y,Z) :- holding(X,Y), holding(X,Z).
```

It is easy to verify that the rules are warded. In the second rule, the first position of `holding` contains a dangerous variable `X`; yet `X` does not appear in `business`, and `Y` cannot host nulls. Therefore, `holding` acts as the ward and the rule is warded. In the third rule, `X` can indeed host null values, but it does not appear in the head, making it in fact not dangerous. Therefore all the rules in our example are warded.

Beyond the direct application of such rules, it is important to know that rules of this kind have been used in data exchange and integration for many years. This includes advanced concept such as updatability [35], the role of function symbols [36], and the (mathematical) concept of limits being extended to them [37].

Vadalog is based on Warded Datalog$^{\pm}$ as its core and extends it with several features of practical utility such as an extension of the notion of *monotonic aggregation* [38] (*min*, *max*, *sum*, *prod*, *count*) and a rich *annotation* mechanism (syntactically denoted by the @symbol), to define interactions with data sources, external libraries, and so on. Reasoning in Warded Datalog$^{\pm}$ has been shown to be PTIME-complete in data complexity [33,34], which bodes well for an effective use of Vadalog in real-world large-scale settings. However, specific reasoning scenarios (e.g., in the area of computational social choice [39]) need to cope with extreme-scale data processing, for which polynomial complexity is not a viable choice. Warded Datalog$^{\pm}$ provides interesting subfragments, such as *Linear Piecewise* [40], for which reasoning is subpolynomial, thanks to a stricter limitation of the allowed recursion. In summary, Vadalog addresses also "Big Data" settings, by carefully reducing the expressive power of the rules.

Reasoning in Warded Datalog$^{\pm}$ is PTIME-complete in data complexity [33,34]. Although polynomial time data complexity is desirable for conventional applications, PTIME-hardness can be prohibitive for "Big Data" applications. One such example is towards building knowledge graphs that consider huge elections in the area of computational social choice [39]. Yet, in fact, this is true even for linear time data complexity. This is discussed in more detail in [15].

This core language has a number of extensions to make it practical, among them: data types, arithmetic, (monotonic) aggregation, bindings of predicates to external data sources, binding function symbols to external functions, and more. The Vadalog system is able to perform ontological reasoning tasks. Reasoning with the logical core of Vadalog is computationally efficient and captures OWL 2 QL[7] and SPARQL (through the use of existential quantifiers), and graph analytics (through non-trivial support for recursion and aggregation). The declarative nature of the language makes the code concise, manageable, and self-explanatory.

Vadalog supports aggregation (*min*, *max*, *sum*, *prod*, *count*), by means of an extension to the notion of monotonic aggregations [38]. The syntax of Vadalog for any aggregation function and operations is to use @function. For example, the @bind and @mapping annotations allow to customize the data sources for the @input function to print results.

3.3. A technical setup of a user interface for the Vadalog system

Vadalog conveys a universal system to integrate various approaches and tools into a unified framework. Such a system requires a user-friendly platform for the complex tasks running behind the data science and machine learning components. JupyterLab is a well-known platform for data analysts and scientists with a convenient interface for data processing and visualization. It was chosen as the setup of user interface for Vadalog. The Vadalog system has seamless integration with JupyterLab with the use of a Vadalog extension and kernel. A typical JupyterLab with a running Datalog program is shown in Fig. 4. Jupyter has a multi-user support, in which dedicated resources and the environment are associated with a concrete user. The Vadalog extension and kernel for JupyterLab give data scientists the possibility to evaluate the correctness of the program, run it, and analyze the derivation process of interesting output facts. All output is rendered in JupyterLab's output area.

Vadalog was developed with modular design. Each program snippet that contains a `@module("ModuleNameA")` annotation statement is treated as a module, and it can refer to other modules with a `@include("ModuleNameN")` statement. Within the JupyterLab environment each code cell represents a module and refers to other cells with respect to `@include` annotations. While running a Vadalog program, a selected code cell is considered as the main program, Vadalog takes a transitive closure over all dependencies and resolves them accordingly. This approach allows partitioning the logic of reasoning and making the code reusable.

The correctness of the program is assessed with the use of the code analysis functionality. This functionality gives the user the ability to catch common semantic errors when writing a Vadalog program. In particular, it gives a warning when a user introduces a cross-join in a rule, or when an undefined predicate is being used in the body of a rule, or when the given program is non-warded. Additionally, our JupyterLab extension has the code linting functionality that allows to catch syntactic errors in the process of Valadaog program development.

The analysis of derivations can be performed with the use of *explanations*.

For instance, in Fig. 5, a derivation tree is presented that explains how the fact `control("A", "C")` is derived given the data and the rules. Note that instead of provenance tracking, where one typically constructs a provenance Boolean formula or a semi-ring expression, in Vadalog the original program is augmented, given an atom to explain, with additional rules that together infer a set of atoms that participate in any proof of the atom that in turn form a proof tree. This construction is similar to [41]. The consolidated and enriched Knowledge Graph is then processed using the standard data science toolkit for graph analytics (including languages such as Cypher,[8] SPARQL,[9] and Gremlin[10]).

4. Data integration strategies in the Vadalog system

As the importance of data science constantly increases, the Vadalog system can support the entire spectrum of data science tasks and processes to a certain extent. For data integration tasks, two main strategies have been implemented in the Vadalog system. In this section, we introduce data bindings as Vadalog's support for data science in integration of heterogeneous information.

In terms of data binding, Vadalog has two principal extension points: (1) predicates and (2) functions. For *predicates*, Vadalog's bindings mark them as *input* or *output*, which can be bound to any particular *data source*. For *functions*, Vadalog's bindings are defined either as system built-in functions or bound to a particular library defined in a multitude of languages such as Java or Python. Data extensions provide access to relational data stored in the underlying database management system (e.g., Postgres or MySQL). As a second approach, Vadalog integrates storage of data as graph data using tools such as Neo4j. Further library extensions also allow the integration of state-of-the-art machine learning tools such as Weka, Scikit-learn (see Section 5.1), and a web data extraction tool, OXPath [3,42,43] (see Fig. 6). An additional integration with libraries for string similarities and regular expressions allows for defining complex entity resolution workflows.

Predicates. Bindings give the possibility to connect an automated reasoning workflow with external systems for data exchange.

External data sources and systems can be declared by adopting `@input` and `@output` annotations. Annotations are special facts augmenting the existential rules with specific characteristics. The `@input` annotation defines that the facts for a predicate should flow from a data source into the Vadalog program, while the `@output` annotation specifies what facts should be produced as a result of the program and, if needed, stored into a data source. Additional annotations such as `@bind` are used to interact with external systems. A query bind annotation `@qbind` is a special variant of `@bind` that supports binding predicates to queries against inputs and outputs in the external language (e.g., SQL-queries for a data source or a system that supports SQL). The first parameter of `@bind` and `@qbind` specifies the external resource a predicate is bound to; the second parameter defines the type of the target (e.g., "postgres"). In case the schema of an external resource cannot be derived automatically, or should be overridden, additional `@mapping` annotations can be used to define a mapping strategy for tuples between Vadalog and an external system.

In Fig. 4, we give a synthetic example of a Vadalog program to infer a company control indicator. It can be formulated

7 https://www.w3.org/TR/owl2-profiles/

8 https://neo4j.com/developer/cypher-query-language/

9 https://www.w3.org/TR/rdf-sparql-query/

10 https://tinkerpop.apache.org/gremlin.html

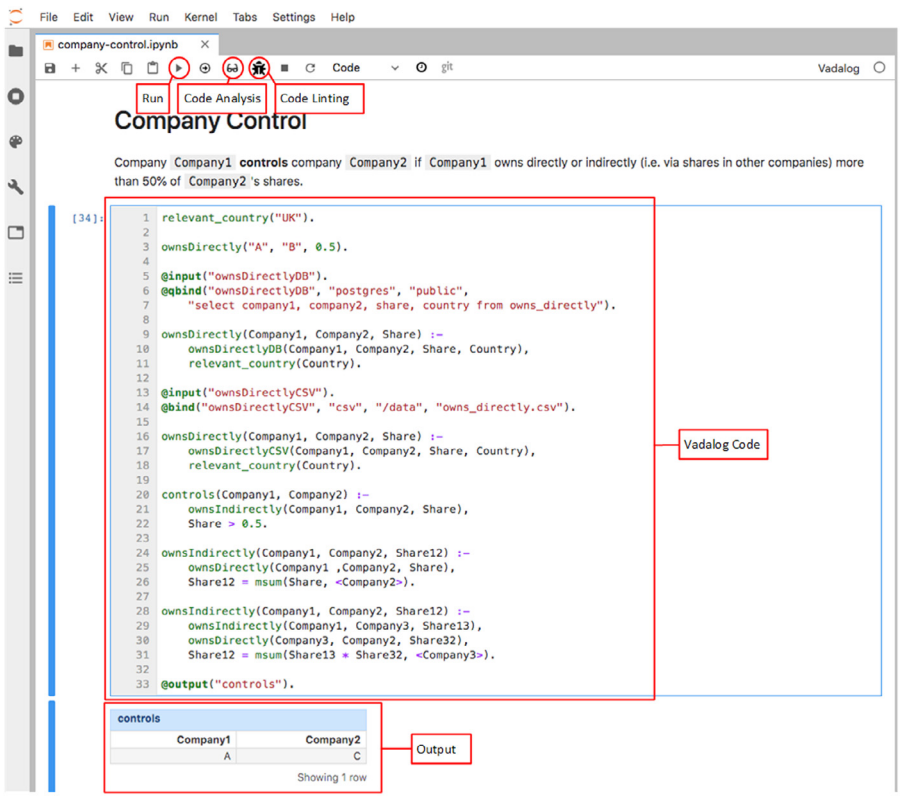


Fig. 4. An example of the Vadalog program for inferring a company control indicator.

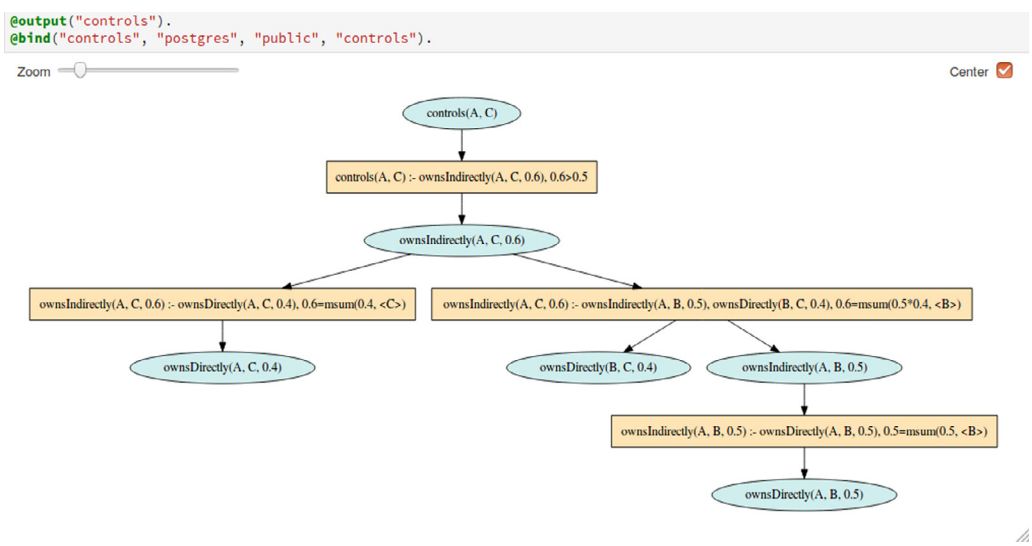


Fig. 5. A screenshot of the output depicting a "yes"-explanation for the fact `controls("A", "C")` in the company control example.

as follows: *A company A "controls" company B if A owns directly or indirectly (i.e., via shares in other companies) more than 50% of B's shares* (lines 20–22). As we can see, various strategies for binding external resources can be used in the Vadalog program. For example, the facts `ownsDirectly` can be propagated into the program from the query binding denoted by `@qbind` (lines 6–7 for Postgres, via facts for `ownsDirectlyDB`) or `@bind` (line 14 for CSV, via the facts for `ownsDirectlyCSV`). In `@qbind`, the SQL query is instantiated with the parameter from the predicate `relevant_country` (line 8). The query instantiation takes place in the context of the join, in which the parameter `C` from `relevant_country` is propagated into the fourth term of `ownsDirectlyDB`. In contrast, in case of `@bind`, the data is streamed into the Vadalog system and filtered on-the-fly by selecting only the information regarding the "relevant country" (line 14). The tuples for `ownsDirectly` can also be specified within the program in terms of facts (line 3). The result of the program can either be shown in the standard output, or saved into a database or any supported target system (via an appropriate `@bind` annotation).

In Fig. 6, we illustrate an example of binding with OXPath.

OXPath [3] is a web data extraction language with several extensions interacting with web applications and extracting data from them. Fig. 6 presents the OXPath binding for all the relevant

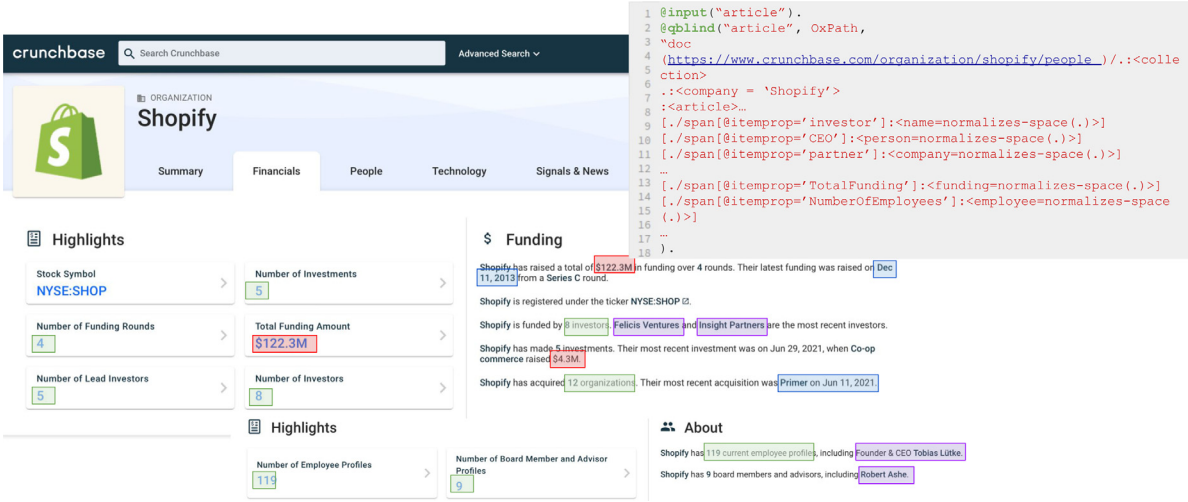


Fig. 6. Integration of OXPath, a web data extraction tool.

information about a company called *Shopify* from the CrunchBase website is shown. The extracted information can be represented as a relation `NumberOfEmployees(X,N)`. The example also highlights that several types of information including text, numeric values, and so on can be extracted by the integrated component of Vadalog.

Functions. Besides bindings, functions provide a data scientist with a rich set of value transformations and operations for different data types supported in Vadalog. A user can write expressions of different complexity with the use of operators and functions to perform arithmetic, manipulate strings, dates, and compare values. Examples of supported data types are `string`, `integer`, `double`, `date`, `boolean`, `set`, and a special identifier for unknown values, `marked null`. A data scientist can also extend the set of supported functions with custom Python functions, supported by the Vadalog framework. Functions can be combined into *libraries*. For example, `@library("sim:", "simmetrics")` enables the "simmetrics" library, whose methods can be invoked with the prefix `sim:`, as in `sim:removeDiacritics(Text)` to remove diacritics from `Text`. We also provide libraries for building regression or classification models on-the-fly and applying them on the data derived via reasoning.

Vadalog directly interacts with various data sources regardless of their nature, be they databases or the Web. Furthermore, thanks to its rich reasoning capabilities, Vadalog can lift the analysis up from basic values, tuples or relations within databases to semantically richer structures, e.g., from property graphs such as of Neo4j to concepts of a domain ontology.

5. Machine learning in the Vadalog system

The Vadalog system is a universal tool that can reconcile two opposite paradigms of data scientists and domain experts, so-called "inductive" (or bottom-up) and "deductive" (or top-down) approaches. An inductive paradigm goes along with a statement that "patterns emerge before reasons for them become apparent" [21]. It certainly refers to data mining and machine learning approaches which are used for deriving new knowledge and relations from data. As all data scientists face in practice, "all models are wrong and some are useful" [44, p. 208], which explains problems of finding the best model given a dataset. Furthermore, limitations related to labor-intensive labeling for some machine learning algorithms can also cause incorrect or incomplete results. Thus, the knowledge from a domain expert, obtained with a deductive approach, is also important to correct errors originating from the machine learning models, as we shall see.

Vadalog is particularly useful for processing Big Data, as the engine allows to continuously update the Knowledge Graph with new data coming in from different sources, such as multiple databases and the web, with a very high performance. Hence, also taking into account the reasoning power of Vadalog that can be used for statistical and algorithmic analysis of the data, Vadalog addresses the four main challenges of Big Data: acquisition, storage, processing, and analysis. The proposed framework can be scalable to hundreds of databases and terabytes of data. In particular the storage and processing of the data can be done in the cloud platform.

The veracity aspect of the proposed data analysis can be addressed through the post-processing verification rules that can also be easily encoded in Vadalog. For example, if a certain statistical or machine learning model is deployed to predict company shares in Europe, a set of post-processing rules can be introduced to check the veracity of this prediction. These rules can be about the range of the shares, or the active years of companies constrained by their profit and other companies they are controlling.

In this section, we will discuss how to integrate machine learning directly. In the first subsection, we will concretely talk about Weka and Scikit-learn integration. The system's TensorFlow integration is similar in style to the Scikit-learn integration. This will be followed in Section 5.2 by a case study on feature engineering. We will conclude in Section 5.3 on how to include custom ML algorithms directly into the system.

These integrations cover a range in terms of whether they are specific to a tool, and whether they require Datalog expertise. The direction integration discussed in Section 5.1 requires for its use no particular Datalog or Vadalog expertise, but is specific to each tool. We note though that while specific to each tool on the technical level (i.e., the person integrating a particular machine learning library for the first time to the system), to the user, the integration is accessed using a uniform syntax. That is, given that, e.g. TensorFlow and Weka are two machine learning libraries integrated into Vadalog already, to the user they are accessed in the same way, without requiring a high degree of Datalog or Vadalog expertise, and without any additional integration effort. Of course, for the overall Knowledge Graph using such machine learning integration, basics of Datalog and Vadalog will be required to benefit from the combination of KGs and ML. The second integration covered in Section 5.2 has a very different profile in terms of tool-specificity and Datalog expertise: It requires

```
j48("CreditRiskModel", X1, X2, X3, X4, Class) :-
    training_data(X1, X2, X3, X4, Class).
@bind("j48", "wekaj48", "risk_level", "file_path").
@mapping("j48", 0, "model", "string").
@mapping("j48", 1, "income", "double").
@mapping("j48", 2, "profit", "double").
@mapping("j48", 3, "balance", "double").
@mapping("j48", 4, "debt", "double").
@mapping("j48", 5, "risk_level", "string").
```

(a) Training a J48 decision tree Weka model

```
classified_data(X1, X2, X3, X4, Class) :-
    new_data(X1, X2, X3, X4),
    j48("CreditRiskModel", X1, X2, X3, X4, Class).

@input("j48").
@bind("j48", "weka",
"{'4' = 'low, medium, high'}",
"{model='${1}', tuple='(${2},${3},${4},${5})', class='${6}'}", "file_path").
@mapping("j48", 0, "model", "string").
@mapping("j48", 1, "income", "double").
@mapping("j48", 2, "profit", "double").
@mapping("j48", 3, "balance", "double").
@mapping("j48", 4, "debt", "double").
@mapping("j48", 5, "risk_level", "string").
```

(b) Classification with a trained J48 Weka model

Fig. 7. Weka decision tree training and classification.

good working knowledge of Datalog or Vadalog (or in general logic programming), but is then largely independent of the actual ML library utilized. The third integration discussed in Section 5.3 requires a higher level of knowledge of logical programming, but also provides a standalone solution not requiring any additional libraries.

5.1. Direct integration with a machine learning framework

The Vadalog system offers integration facilities with most common data scientists' tools, for instance Jupyter for both data processing and visualization. The integration with most used machine learning tools is also supported by the system thanks to an annotation mechanism. We will concentrate on Weka and Scikit-learn, showing a multi-value classification use case, in particular in the credit risk domain. We are going to show two examples of how these machine learning libraries are integrated into Vadalog and how they are used by data scientists. The approach is of course not limited to these two specific machine learning libraries.

Weka is a comprehensive open source machine learning toolkit. The integration of Weka in Vadalog is demonstrated in Fig. 7. One of the clustering and classification algorithms of Weka is the implementation of Iterative Dichotomiser 3 (ID3) decision tree algorithm, namely, the J48 model [45]. By applying a decision tree like J48 on a dataset with a list of predictors or independent variables and a list of targets or dependent variables, one can predict the target class of a new dataset record. Fig. 7(a) illustrates a J48 model generation example for a financial credit risk model of companies. The dataset containing four features and the corresponding risk class for each company is stored in the predicate `training_data`.

Here the training data is propagated to the bound decision tree classifier associated with the predicate `j48` and stored in a file. The mapping annotations in this program specify attributes and the class of tuples streamed into the underlying machine learning model. Fig. 7(b) depicts an example of the classification process given an id of a model stored in the `j48` model.

The attributes of the entries in the predicate `new_data` to be classified and the generated model are streamed into the underlying Weka framework via the predicate `j48`. Note now that the predicate `j48` has an additional attribute storing the id of the model. The results of the classification are instantiated in a relation `classified_data`. In the `@qbind` expression, the third parameter defines nominal attributes, a class in our case, which had index 4 in the training phase. The fourth parameter of `@qbind` defines the parameter propagation template from the predicate `j48` into the underlying model. As the result of the execution of the program, the predicate `classified_data` contains the class that is predicted by the model for the given input.

Scikit-learn. An external Python library such as Scikit-learn can be utilized for machine learning tasks over predicates, through the Vadalog Library framework. One basic linear regression example is shown below. The input consists of predicates in the form of `training_set(ID, X, Y)`. The `sk:fit` function feeds the input data one by one and returns the current training set size. Once a sufficient training set size is reached, the `sk:train` function is called with a Boolean return value. The returned Boolean value stands for whether the training is successful on the errors occurred or not. This true value is used as a guard in the result calculation rule to ensure the prediction part is called after the training is finished. The last rule takes `predict` inputs one by one and retrieves the output from a trained model. The value `#T` stands for the Boolean value "true".

```
@library("sk:", "sklearn").
training_set("ID1", [1, 1], 2).
training_set("ID1", [2, 2], 4).
training_set("ID1", [3, 3], 6).
predict("ID1", [17, 17]).

training_size(ID, C) :- training_set(ID, In, Out),
                        C=sk:fit(ID, In, Out).
classified(ID, R) :- training_size(ID, C), C>=3,
                        R = sk:train(ID).
result(ID, In, Out) :-
   predict(ID, In), classified(ID, #T),
                        Out = sk:predict(ID, In).
```

5.2. A data science workflow

We consider the case study of implementing a supervised machine learning framework and post-classification reasoning with Vadalog. Our implementation consists of three phases: **(1)** feature extraction with Vadalog, **(2)** interaction between Vadalog and a serialized classifier, **(3)** post-classification reasoning. We assume that the classifier has already been trained and serialized.

1. **Feature extraction with Vadalog** The task of selecting useful input attributes for machine learning models and leading their attention often relies on two major aspects, namely, feature selection and feature extraction. Let us assume we target the problem of classifying companies by sector of economic activities, i.e., automatically telling what the industry is they are active in, such as crop, manufacturing, building, transporting, and so on. A company has a set of basic features, which we capture via Vadalog EDB atom in simple join rules, as exemplified in the case of number of employees.

```
feature("NEmployees",N,X) :-
        numberOfEmployees(X,N), company(X).
@output("feature").
```

The idea is accessing, for each company `X`, the value `N` of the feature. All the features are then harmonized in a relational form in the `feature` atom, which will be fed to a pre-trained classifier,

as we shall see. The `@output` annotation is self-explanatory and simply defines the output of the reasoning process.

The power of reasoning in feature extraction becomes particularly visible when the features are not directly available as attributes of the entity of interest, like for instance our companies, but derive from a more general analysis of the data, driven by domain knowledge. Consider, for example, in our company domain, the knowledge of the number of direct and indirect shareholders of a company; it is indeed a good indicator of the sector of the company, as specific fields, e.g., professional investments, tend to be more subject to share dispersion, while other kinds of companies, e.g., food or manufacturing, are more often held by a smaller set of shareholders, or are even family businesses.

```
feature("NShares",C,X) :- nShareHolders(X,C).
ShareHolders(X,C) :-
        ownsDirectly(Y,X,W), C=mcount(Y).
nShareHolders(X,C) :-
         nShareHolders(Y,Z,W1), owns(Z,X,W2),
            C=mcount(Y).
```

Beyond the first basic examples, features could derive even from the application of very complex domain knowledge, for instance representing a financial regulation, a common case in the *FinTech* realm. For example, the following set of rules defines a feature counting the number of subjects in "close link" with a company `X`. According to the regulation of the European Central Bank [46], two companies are in a close link if one controls more than 20% of the other or if a third party owns more than 20% of both. Assuming that the `totalOwn(X,Y,W)` atom stores the total amount of shares `W` that `X` owns of `Y`, we can define the `nCloseLinks` feature with the following Vadalog rules:

```
feature("NCloseLinks",C,X) :- nCloseLinks(X,C).
nCloseLinks(X,C) :- totalOwn(Y,X,W),
            Y > 0.2, C=mcount(Y).
nCloseLinks(X,C) :- totalOwn(Z,X,W),
            totalOwn(Z,Y,V), W > 0.2,
            V > 0.2, C=mcount(Y).
```

In the above Vadalog program we omitted the `@output ("features")` annotation, for the sake of simplicity. This is indeed needed to specify the expected output of the reasoning process, the result of feature extraction in this case.

2. **Interaction with a Serialized Classifier** All of the extracted features are passed into a serialized classifier through the `@bind` operator. For the case presented above we use Weka as the machine learning library and J48 decision tree as the classifier. The implementation of the framework in Vadalog is both library and classifier agnostic, e.g., we can seamlessly integrate Vadalog with Scikit-learn, as demonstrated in Section 5.1. The J48 decision tree classifier can also be seamlessly replaced with any other classifier, e.g., Support Vector Machine (SVM). The classifications produced by the classifier are then passed back to Vadalog, also through the `@bind` operator.

3. **Post-classification Reasoning** We can now apply post-classification reasoning that cannot be easily represented by machine learning classifiers to the classifications computed in the previous phase.

In the following basic example, the rule is designed to reason over the classification information of companies and their sectors together with their control information. In particular, the classification for a company `X` is overridden by that of `Y` which controls it.

```
classification(X,A) :- controls(Y,X),
            classification(Y,A).
```

We further extend this example through a more complex rule where the reasoning of company classification depends on the family connections. For this we consider the information of companies that can be controlled by people or other companies. Here we present an example with three rules about companies controlled by people, and their family information. We assume that company `X` is controlled by person `P` who is from family `F`. Similarly, company `Y` is controlled by person `S` who also has roots in family `F`. Based on the rule below, it is possible to infer the sector of company `X` by knowing only the sector of company `Y`. In addition, the presence of information about families and last names, supports the possibility for reasoning about families. These rules in the post-reasoning phase show how we aim at allowing machine learning models to access complex features that facilitate the inference of hidden knowledge.

```
classification(X,A) :-
              controls(P,X), controls(S,Y),
             Family(S,F), classification(Y,A).
∃F Family(F,P) :- Person(P).
F1=F2 :- LastName(P1,L), LastName(P2,L),
     Family(F1,P1), Family(F2,P2).
```

Discussion. The benefits of employing Vadalog in data science workflows can be summarized in the gained access to hidden knowledge in different steps of the process, so that it can be leveraged in typical machine learning tasks. Such hidden knowledge is the one obtained via the reasoning process and seamlessly integrated into the statistical models. Let us discuss such advantages more in detail with respect to our exemplifying classification problem.

Basic classification models (e.g., linear classifiers) adopted in data science workflows have no access to network-based features, which derive in a complex way from the interaction of many entities. In our example, the number of shareholders and the number of close links are considered good "predictors" for the specific sector of economic activity, and thus represent an input for the machine learning model. On the other hand, such features are not immediately available and even depend on sophisticated regulations. A declarative encoding of the feature extraction builds a hybrid cohesive model, inheriting the advantages of declarative programming over a basic algorithmic approach. Furthermore, the kind of knowledge encoded in regulations, representing domain experts' opinions and modeling best practices, can hardly be encoded in the form of statistical models, and is best suited for a KRR representation.

The interaction with the classifier is straightforward and technically based on a relational interface. The application of Vadalog in post-classification is interesting and beneficial. It is our experience that the results of machine learning models must be verified and in certain cases complemented or even overridden by business knowledge. In our example, we assume that a shared company control implies the coincidence of economic sector; likewise, the same controlling family is an indicator of coinciding economic sectors. This relation is clearly probabilistic, though we are simplifying it as deterministic in this case and will discuss probabilistic reasoning in Section 6. Essentially, the post-classification reasoning contributes to the performance of the classification, by injecting ex post knowledge. While, for the sake of simplicity, in the example we show rules that operate in override mode with respect to machine learning classification, more sophisticated cases propose a mutual interaction between the decisions achieved by the classifiers and the conclusions of the rules. It is the case of link prediction scenarios in the financial realm [47], where we experienced the powerful combination of recursion and standard classification.

In recent work [47, Sec.6] we have provided experimental evidence of quantitative gains achieved by the presence of reasoning rules in our hybrid pipeline in terms of classification performance. In particular, our validation technique consisted in the setup of *ablation studies* for the case of predicting family links for company control purposes. We observed a shift of recall from 100% to 90%, with the progressive deactivation of logical inference steps (corresponding to the suppression of specific clustering decisions). It is interesting to observe that in this process, reasoning rules also affect the overall scalability with nontrivial dynamics. Rules allow to tune the trade-off between the accuracy of the results and the efficiency of the response. Stricter rules improve the performance, while providing more filtered test data to the machine learning model and thus lowering recall; on the other hand, more tolerant rules, produce a less substantial cut in the search space, with a more accurate final result and a longer elapsed time. Furthermore, in general, as far as scalability is concerned, the theoretical tractability of Warded Datalog$^{\pm}$ already bodes well for the feasibility of real-world settings. In practice, we could verify the system scalability of Vadalog reasoning for KGs in data science scenarios with tens of millions of nodes and hundreds of millions/billions of edges [48].

5.3. Direct implementation of algorithmic modeling

A core benefit of declarative approaches to data management is the independence of the specific algorithm. Our data science framework is fully declarative in the sense that both the reasoning components and the pure machine learning models do not require the user to provide algorithms, conveying high control on the overall application. As a consequential benefit, we obtain robustness to change, modularity, and explainability of the process.

Nevertheless, in specific situations, a real-world platform calls for ad-hoc implementations of data processing algorithms, for performance reasons or for peculiar functional requirements.

Our framework can be used at a lower level and, thanks to the high expressive power of Vadalog, the explicit encoding of machine learning algorithms is possible. Clearly, the actual implementation strategy is still delegated to the reasoning engine, which can take arbitrary optimization choices. However, unlike black-box machine learning algorithms, the user has a higher level of control.

So far, we presented how Vadalog can support relevant AI-tasks that could be used in data science workflows such as classification. We now show a lower level implementation of a popular density-based spatial clustering algorithm [49].

The algorithm takes a set of points in some metric space as input and clusters together points that are closely packed together, marking as outliers points that lie in low-density regions. The two main parameters of the algorithm are a real ϵ (distance threshold) and an integer *minPts* (the minimal number of points for a dense region). Intuitively, the algorithm performs the following steps.

1. Find the points in the ϵ neighborhood of every point.
2. Identify the **core points**, i.e., points with more than *minPts* neighbors.
3. Find the connected components of core points on the neighbor graph, ignoring all non-core points.
4. Assign each non-core point to a nearby cluster if the cluster is an ϵ-neighbor, otherwise assign it to noise.

We concentrate on Steps 2–4, while Step 1, which is a construction of the ϵ-neighbor graph, is usually the task of a downstream application where, e.g., points are keywords with a string similarity function as the distance, or points in multi-dimensional space with the Euclidean distance. We assume that the binary relation `graphEdge` storing ϵ-neighbors has been computed. Moreover, the relation `minPts` contains the input parameter. Step 2 is described by:

```
corePoint(Point) :-
  graphEdge(Point, Point1),
  minPts(CorePointThreshold),
  Count = count(Point1),
  Count > CorePointThreshold.
```

Step 3 can be formulated via the rules expressing the transitive closure:

```
connected(Point1, Point2) :-
  graphEdge(Point1, Point2),
  corePoint(Point1),
  corePoint(Point2).

connected(Point1, Point3) :-
  connected(Point1, Point2),
  graphEdge(Point2, Point3),
  corePoint(Point3).
```

For Step 4, we associate a cluster ID with its designated core point and assign a non-core point to a nearby cluster. We omit the rule computing the noise points, i.e., the points not associated with a cluster.

```
clusterMembers(Cluster, Point, "core") :-
  connected(Point, Point1),
  Cluster = min(Point1).

clusterMembers(Cluster, BorderPoint, "border") :-
  clusterMembers(Cluster, Point, "core"),
  graphEdge(Point, BorderPoint),
  not corePoint(BorderPoint).

@output("clusterMembers").
```

As the last step, we output the computed clusters.

Summary. In this section we have seen how the Vadalog system can be used in a wide variety of data science settings. We first showed the direct integration with machine learning frameworks. We then proceeded to showcase the overall data science workflow by going through our running example in great detail to enable data scientists to see how to design such workflows in Vadalog. We concluded the section by showing that, in special cases, the Vadalog system is flexible enough to directly encode and execute algorithmic models.

6. Probabilistic reasoning

In the design of data science solutions, it is more and more clear that completely neglecting domain knowledge and blindly relying only on inductive models (i.e., with parameters learnt from data) easily leads to sub-optimal results, subject to overfitting, when not to wrong conclusions. Thus, data scientists tend to integrate inductive reasoning with complementary deductive approaches that can in certain cases overrule machine learning models with domain knowledge.

In the Vadalog system, we introduce *Probabilistic Knowledge Graphs*, a valuable tool to craft a new kind of data science solutions where statistical models incorporate and are driven by the description of the domain knowledge.

The combination of uncertainty and logic describing rich uncertain relational structures is not new and has been the primary focus of Statistical Relational Learning (SRL) [50,51] and Probabilistic Logic Programming (PLP) [52]. A prominent representative

of an SRL language are Markov Logic Networks (MLN) [53], which allow one to describe relational structures in terms of first-order logic.

MLNs have been successfully applied in many areas such as natural language processing [54], ontology matching [55], and record linkage [56]. Yet, one common limitation of SRL models is their logical reasoning side: logic in SRL is not utilized for deducing new knowledge, but rather serves the role of a constraint language. Most probabilistic programming languages such as ProbLog adopt the Sato distribution semantics [57], where probabilistic choices are made on the underlying extensional data, and the rules are then applied to form the possible worlds. We find the languages from the PLP family to be very suitable for many use cases for combining uncertainty and rules. However, none of them support the creation of new values (either via existential variables or arithmetic operations), which is an essential feature for reasoning over knowledge graphs and for data science.

Therefore, systems that can be built on top of these models are of limited applicability in data science tasks. Consider the following example. Let G be a knowledge graph, which contains the following facts about the ownership and link relationships between companies, augmented with a Vadalog program composed of rules (1) and (2):

```
own("a", "b",0.4). own("b", "c", 0.5).
own("a", "d", 0.6). own("d", "c", 0.5).

linked(X, Y) :- own(X, Y, S), S > 0.2 .  (1)
@weight(0.8). own(X, Z, W) :-
  own(X, Y, S), own(Y, Z, T), W = sum(S * T). (2)
```

Rule (1) expresses that company x is linked to y if x owns directly or indirectly more than 20% of y's shares. Rule (2) is a recursive rule with an aggregate operator and expresses indirect shareholding: when x owns a number of companies y, each holding a different share t_y of z, then x owns $\sum_y (s \cdot t_y)$ of z. Moreover, the rule (2) is uncertain and has the weight 0.8, as specified by the `weight` annotation. An example of a "traditional" logical reasoning task is answering the following question over G: *"which companies are linked to a?"*. The result of the reasoning task is provided by the companies b and d, as directly specified by G, and, additionally, c, which is implied by the program. Indeed, by Rule (2) we first derive the fact `own("a", "c", 0.5)`, as $0.4 \times 0.5 + 0.6 \times 0.5 = 0.5$, and thus, by Rule (1), we deduce `linked("a", "c")`.

However, here we are in an uncertain setting: rule (2) is not definitive, but holds with a certain probability. We say that G is a Probabilistic Knowledge Graph. Probabilistic reasoning on G would then consist in answering queries over such uncertain logic programs, i.e., when we can only access the distribution of the entailed facts. The answer to the question – which companies are linked to a – would contain companies b and d with probability one and c with some probability p depending on the "ownership distance" between a and c.

In spite of its high relevance, surprisingly, none of the exiting KGMSs allow for uncertain reasoning, crucial in many contexts. The Vadalog system aims at filling this gap.

The Vadalog system provides a form of hybrid logic-probabilistic reasoning, where logical inference is driven and aided by statistical inference. We adopt the novel notion of *Probabilistic Knowledge Graph*, and propose Soft Vadalog, an extension to Vadalog with soft, weighted rules for representing and supporting uncertain reasoning in the Vadalog system [58]. A Soft Vadalog program is a template for a reason-tailored statistical model, namely the *chase tree*, the semantics of which is based on a probabilistic version of the *chase* procedure, a family of algorithms used in databases to enforce logic rules by generating the entailed facts.

In particular, the system adopts the *MCMC-chase* algorithm: a combination of a Markov chain Monte Carlo method with the chase. The application of the chase is guided by the MCMC, so that logical and statistical inference are performed in the same process.

7. Vadalog: Related work, systems and database literature

In this section we cover the extensive related work associated to the topic. We put a particular focus on the database part, as it likely is less familiar to a data scientist interested in this topic.

Ontological Reasoning. Vadalog is a state-of-the-art KGMS, based on expressive logic languages for *Knowledge Representation and Reasoning* (KRR). In the database literature, the main focus of logic-based KGMSs stems from the longstanding attention to the problem of *ontological reasoning* [31], which, in its essence, consists in answering a query Q over a database D, in the presence of a set Σ of schema constraints that model the domain of interest. In the query answering process, thanks to these constraints that act on D, new knowledge is generated in the form of new facts for the answer set. The reasoning research field inherits the extensive experience of the database community in *query answering* [59] and *data exchange* [60], where many fundamental results of applied logical inference have been laid out. For example, the CHASE procedure [32], a seminal technique to enforce constraint satisfaction, has become a central technique to perform reasoning inference steps, to the point that we witnessed the proliferation of chase variants, extensions, and practical applications [30,61], in an effort to make it suitable for ontological reasoning.

Key requirements. A crucial challenge that arose in the reasoning research area is the need for chase-based procedures to jointly handle *existential quantification*, which is the creation of new, fresh objects, and *recursion*, essential to implement Knowledge Graph traversals and, more generally, algorithms requiring the expressive power of the transitive closure. As the arbitrary interplay of these two features leads to undecidability of query answering even in very simple cases [34,62], a broad research stream started to dedicate attention to the design of KRR languages able to strike a good balance between expressive power and computational complexity, that is, they had to be sufficiently rich to model real-world problems and, at the same time, simple enough to be tractable and scalable [15]. Vadalog takes on most recent results in this wake, in particular by focusing on *Warded Datalog*$^{\pm}$ [34], which is at its core.

Knowledge Graphs. The link between general purpose reasoning, intended as ontological query answering, and Knowledge Graphs is based on a specific angle we consider for both the extensional (EDB) and intensional (IDB) data. In many domains of interest where KGs can be applied, the extensional data, i.e., the database D, can be effectively represented in the form of a usual graph, so by means of a set of nodes and relationships between them, or triples, as we have seen. Then, unlike usual graphs, in KGs such extensional knowledge is augmented by Σ, a set of rules intensionally defining new nodes, edges, or derived knowledge. In this sense, we adopt an operational definition of KGs in terms of their ground extensional, intensional, and derived intensional, i.e., the result of the reasoning process, components [13].

Data Science Workflows There is extensive related work that discusses the overall data science pipeline from data exploration, passing through data wrangling, and ending with ML models [63, 64]. A data science workflow is presented in [65] for managing big data analytics. The fundamental environment for the considered

workflow mainly is a data warehouse. However, the objective is to fit the scalability aspects of this environment to be able to deal with big data challenges in a data science workflow. In order to do so, a local approach is proposed to include preprocessing, computing models, data cleaning, aggregation, column transformation, and scoring datasets inside a relational database system. The challenges in each of these steps are discussed, and possible approaches are presented, mainly using SQL queries. Although some steps correlate with what we present here, their approach does not support rule-based reasoning, which is the main strength of Vadalog. In another work [66], such a workflow is discussed in the presence of Apache Spark and focuses on handling the memory and data locality appropriately. The given solution is a parallel scientific workflow engine called TARDIS. Similarly, in [67], an approach is represented for distributed caching of scientific workflows in a multi-site cloud. Although these and other similar works are mainly targeting the scientific workflow of data management, they are not mainly focusing on data science tasks to be solved with reasoning approaches.

Other Systems and Languages. Several reasoning systems can be considered as related work to Vadalog in the database literature. Some of them have been already cited and include DLV [10], Graal [11], LLunatic [8], RDFox [7], and PDQ [12], as well as DEMo [68], Pegasus [69], E [70], and ChaseFUN [71]. In this paper, we build on the mentioned body of knowledge from the whole database community and from our direct experience in developing the Vadalog system [40]. Our aim is pushing the boundaries of a joint use of machine learning and reasoning, paving the way to a hybrid data science process thanks to a fully engineered system and language. In particular, Vadalog extends the set of tools that are available to a data scientist and provides a cohesive framework to integrate all of them, as we have exemplified with specific reference to a classification problem. In this sense, the flurry of popular machine learning programming, scripting and statistical languages such as Python, R, STATA, eViews, as well as the most used libraries such as Keras, TensorFlow, Scikit-learn, Spark MLib, should all be considered fundamental components of our approach. Vadalog allows to inject a-priori business knowledge into the data science process, in the form of reasoning rules which provide the input to the mentioned tools – and models they offer – in the form of training sets and, in turn, receive output datasets from them. Vadalog bridges libraries and rules by means of a relational interface. We can say that our approach is *hybrid*, in the sense that the reasoning models we support consist of both reasoning rules and mathematical, quantitative models.

Injecting a-priori knowledge into statistical models is not new in general, and, for example, is of the essence in Bayesian models, where such knowledge is also updated in the process. Vadalog offers a compact, expressive, and elegant form to specify this knowledge in such a way that its contribution in the final data science decision is fully explainable and logically consequential. It is our experience that in domains where the business knowledge is available, sometimes even with some level of formalization or encoding, and is highly data-driven, reasoning rules are an extremely effective means to make the statistical models aware of the context. Beyond that, it goes without saying that our reasoning rules are not necessarily certain, but can well represent *probabilistic knowledge*, even encoding sophisticated graphical models like Markov Networks, as we have seen in Section 6.

We conclude with some more detailed motivation for the choice of a Datalog-based KRR language for a data science framework. One common option for reasoning with KGs represented as triples is SPARQL, typically adopted under the set semantics and the entailment regime of OWL 2 QL [15]. However, as shown by Krötzsch [72], reasoning with multi-attributed graphs (which represent n-ary relationships), very common in real-world settings like in financial KGs, is not possible with such language. On the other hand, support for full recursion is essential for graph traversal and is not natively supported by SQL or only in an extremely laborious or inefficient form. In fact, our experimental evidence and SQL code analysis show how Vadalog outperforms SQL even for *isomorphism check*, which is the standard operation to avoid duplication in the answer set as well as non-termination of the reasoning process [6, Sec.7]. More generally, while SQL is indeed a valuable choice for data analysis tasks, and thus plays the same role as R, Python, and other statistical tools that are complementary to our framework, SQL does not lend itself to effectively represent the reality of interest and encode business knowledge.

8. Conclusion

In this paper we showed how modern KGMSs such as Vadalog can be used to support a wide spectrum of data science tasks in a unified way. We described a five-stage template for practical data science workflows going through all of the major tasks in the spectrum: We started by describing (i) data wrangling, which is considered to be up to 80% of typical data science workloads. We also considered data extraction as another important way to ingest data into a data science workflow. We then considered how to integrate (ii) statistical reasoning and (iii) machine learning into such workflows. We then showed how to build (iv) algorithmic modeling tasks and concluded with (v) probabilistic reasoning, important in typical data science scenarios with uncertain data. Through this, we followed the workflow of a data scientist going through these steps in our running example based on financial Knowledge Graphs. While real-world data science workflows differ widely in scale and objective – the focus could be on either of our presented tasks, or even tasks only touched upon very briefly – we believe that our five-stage template is a good starting point for practical engineers to start using Knowledge Graphs for data science workflows. As ongoing and future work, we are deploying this process in industrial scenarios ranging from the financial space scenario described in this paper to aircraft engineering, consumer appliance manufacturing, supply chain management, and many more.

CRediT authorship contribution statement

Luigi Bellomarini: Writing – original draft, Writing – review & editing. **Ruslan R. Fayzrakhmanov:** Writing – original draft, Writing – review & editing. **Georg Gottlob:** Writing – original draft, Writing – review & editing. **Andrey Kravchenko:** Writing – original draft, Writing – review & editing. **Eleonora Laurenza:** Writing – original draft, Writing – review & editing. **Yavor Nenov:** Writing – original draft, Writing – review & editing. **Stéphane Reissfelder:** Writing – original draft. **Emanuel Sallinger:** Writing – original draft, Writing – review & editing. **Evgeny Sherkhonov:** Writing – original draft, Writing – review & editing. **Sahar Vahdati:** Writing – original draft, Writing – review & editing. **Lianlong Wu:** Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the EPSRC, United Kingdom programme grant EP/M025268/1 VADA, the Vienna Science and Technology Fund (WWTF), Austria grant VRG18-013, EPSRC, United Kingdom iCASE studentship grant EP/P510609/1 2370505, the EU Horizon 2020 grant 809965. Georg Gottlob was supported by his Royal Society Research Professorship, United Kingdom, Grant "RAISON DATA" RP/R1/201074.

References

[1] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, S. Hellmann, DBpedia - A crystallization point for the web of data, J. Web Sem. 7 (3) (2009) 154–165, http://dx.doi.org/10.1016/j.websem.2009.07.002.

[2] D. Vrandecic, M. Krötzsch, Wikidata: A free collaborative knowledgebase, Commun. ACM 57 (10) (2014) 78–85, http://dx.doi.org/10.1145/2629489.

[3] T. Furche, G. Gottlob, G. Grasso, C. Schallhart, A.J. Sellers, OXPath: A language for scalable data extraction, automation, and crawling on the deep web, VLDB J. 22 (1) (2013) 47–72.

[4] N. Konstantinou, M. Koehler, E. Abel, C. Civili, B. Neumayr, E. Sallinger, A.A.A. Fernandes, G. Gottlob, J.A. Keane, L. Libkin, N.W. Paton, The VADA architecture for cost-effective data wrangling, in: SIGMOD, ACM, 2017, pp. 1599–1602.

[5] N. Konstantinou, E. Abel, L. Bellomarini, A. Bogatu, C. Civili, E. Irfanie, M. Koehler, L. Mazilu, E. Sallinger, A.A.A. Fernandes, G. Gottlob, J.A. Keane, N.W. Paton, VADA: An architecture for end user informed data preparation, J. Big Data 6 (2019) 74.

[6] L. Bellomarini, E. Sallinger, G. Gottlob, The vadalog system: Datalog-based reasoning for knowledge graphs, PVLDB 11 (9) (2018) 975–987.

[7] B. Motik, Y. Nenov, R. Piro, I. Horrocks, D. Olteanu, Parallel materialisation of datalog programs in centralised, main-memory RDF systems, in: AAAI, AAAI Press, 2014, pp. 129–137.

[8] F. Geerts, G. Mecca, P. Papotti, D. Santoro, That's all folks! LLUNATIC goes open source, PVLDB 7 (13) (2014) 1565–1568.

[9] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV system for knowledge representation and reasoning, ACM Trans. Comput. Log. TOCL 7 (3) (2006) 499–562.

[10] M. Alviano, W. Faber, N. Leone, S. Perri, G. Pfeifer, G. Terracina, The disjunctive datalog system DLV, in: Datalog, in: Lecture Notes in Computer Science, vol. 6702, Springer, 2010, pp. 282–301.

[11] J. Baget, M. Leclère, M. Mugnier, S. Rocher, C. Sipieter, Graal: A toolkit for query answering with existential rules, in: RuleML, 2015, pp. 328–344.

[12] M. Benedikt, J. Leblay, E. Tsamoura, Querying with access patterns and integrity constraints, PVLDB 8 (6) (2015) 690–701.

[13] L. Bellomarini, D. Fakhoury, G. Gottlob, E. Sallinger, Knowledge graphs and enterprise AI: The promise of an enabling technology, in: 2019 IEEE 35th International Conference on Data Engineering, ICDE, IEEE, 2019, pp. 26–37.

[14] T. Furche, G. Gottlob, B. Neumayr, E. Sallinger, Data wrangling for big data: Towards a lingua franca for data wrangling, in: AMW, vol. 1644, 2016, pp. 1–5.

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

[16] L. Bellomarini, R.R. Fayzrakhmanov, G. Gottlob, A. Kravchenko, E. Laurenza, Y. Nenov, S. Reissfelder, E. Sallinger, E. Sherkhonov, L. Wu, Data science with vadalog: Bridging machine learning and reasoning, in: E.H. Abdelwahed, L. Bellatreche, M. Golfarelli, D. Méry, C. Ordonez (Eds.), Model and Data Engineering - 8th International Conference, MEDI 2018, Marrakesh, Morocco, October 24-26, 2018, Proceedings, in: Lecture Notes in Computer Science, vol. 11163, Springer, 2018, pp. 3–21.

[17] V. Janev, D. Graux, H. Jabeen, E. Sallinger (Eds.), Knowledge graphs and big data processing, in: Lecture Notes in Computer Science, vol. 12072, Springer, 2020.

[18] L. Bellomarini, E. Sallinger, S. Vahdati, Knowledge graphs: The layered perspective, in: V. Janev, D. Graux, H. Jabeen, E. Sallinger (Eds.), Knowledge Graphs and Big Data Processing, in: Lecture Notes in Computer Science, vol. 12072, Springer, 2020, pp. 20–34.

[19] L. Bellomarini, E. Sallinger, S. Vahdati, Reasoning in knowledge graphs: An embeddings spotlight, in: V. Janev, D. Graux, H. Jabeen, E. Sallinger (Eds.), Knowledge Graphs and Big Data Processing, in: Lecture Notes in Computer Science, vol. 12072, Springer, 2020, pp. 87–101.

[20] L. Bellomarini, M. Benedetti, A. Gentili, R. Laurendi, D. Magnanimi, A. Muci, E. Sallinger, COVID-19 and company knowledge graphs: Assessing golden powers and economic impact of selective lockdown via AI reasoning, CoRR, abs/2004.10119, 2020, arXiv:2004.10119.

[21] V. Dhar, Data science and prediction, Commun. ACM 56 (12) (2013) 64–73.

[22] I.H. Witten, E. Frank, Data mining: Practical machine learning tools and techniques with java implementations, Acm Sigmod Rec. 31 (1) (2002) 76–77.

[23] S. Drazin, M. Montag, Decision tree analysis using weka, in: Machine Learning-Project II, University of Miami, 2012, pp. 1–3.

[24] S. Sarawagi, Information extraction, Found. Trends Databases 1 (3) (2008) 261–377.

[25] E. Ferrara, P.D. Meo, G. Fiumara, R. Baumgartner, Web data extraction, applications and techniques: A survey, Knowl. Based Syst. 70 (2014) 301–323.

[26] R.R. Fayzrakhmanov, E. Sallinger, B. Spencer, T. Furche, G. Gottlob, Browserless Web Data Extraction: Challenges and Opportunities, in: Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018, pp. 1095–1104.

[27] P. Christen, Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection, in: Data-Centric Systems and Applications, Springer, 2012.

[28] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.

[29] S.S. Huang, T.J. Green, B.T. Loo, Datalog and emerging applications: An interactive tutorial, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, 2011, pp. 1213–1216.

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

[31] A. Calì, G. Gottlob, A. Pieris, Towards more expressive ontology languages: The query answering problem, Artificial Intelligence 193 (2012) 87–128.

[32] D. Maier, A.O. Mendelzon, Y. Sagiv, Testing implications of data dependencies, ACM Trans. Database Syst. 4 (4) (1979) 455–468.

[33] M. Arenas, G. Gottlob, A. Pieris, Expressive languages for querying the semantic web, in: PODS, 2014, pp. 14–26.

[34] G. Gottlob, A. Pieris, Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue, in: IJCAI, 2015, pp. 2999–3007.

[35] P. Guagliardo, R. Pichler, E. Sallinger, Enhancing the Updatability of Projective Views, in: Proceedings of the 7th Alberto Mendelzon International Workshop on Foundations of Data Management, 2013.

[36] G. Gottlob, R. Pichler, E. Sallinger, Function symbols in tuple-generating dependencies: Expressive power and computability, in: T. Milo, D. Calvanese (Eds.), Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015, ACM, 2015, pp. 65–77.

[37] P.G. Kolaitis, R. Pichler, E. Sallinger, V. Savenkov, Limits of schema mappings, Theory Comput. Syst. 62 (4) (2018) 899–940.

[38] A. Shkapsky, M. Yang, C. Zaniolo, Optimizing recursive queries with monotonic aggregates in DeALS, in: ICDE, 2015, pp. 867–878.

[39] T. Csar, M. Lackner, R. Pichler, E. Sallinger, Winner determination in huge elections with MapReduce, in: AAAI, AAAI Press, 2017, pp. 451–458.

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

[41] S. Köhler, B. Ludäscher, Y. Smaragdakis, Declarative datalog debugging for mere mortals, in: International Datalog 2.0 Workshop, Springer, 2012, pp. 111–122.

[42] R.R. Fayzrakhmanov, C. Michels, M. Neumann, Introduction to OXPath, CoRR, abs/1806.10899, 2018, URL http://arxiv.org/abs/1806.10899.

[43] C. Michels, R.R. Fayzrakhmanov, M. Ley, E. Sallinger, R. Schenkel, OXPath-BAsed data acquisition for dblp, in: 2017 ACM/IEEE Joint Conference on Digital Libraries, JCDL 2017, Toronto, on, Canada, June 19-23, 2017, IEEE Computer Society, 2017, pp. 319–320.

[44] G.E.P. Box, J.S. Hunter, W.G. Hunter, Statistics for Experimenters: Design, Innovation, and Discovery, second ed., Wiley-Interscience, 2005, p. 633.

[45] A.E. Khedr, A.M. Idrees, A.I. El Seddawy, Enhancing iterative dichotomiser 3 algorithm for classification decision tree, Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 6 (2) (2016) 70–79.

[46] GUIDELINE (EU) 2011/14 of the ECB Guideline of the European Central Bank, https://cutt.ly/DjJQcup.

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

[48] L. Bellomarini, M. Benedetti, A. Gentili, R. Laurendi, D. Magnanimi, A. Muci, E. Sallinger, COVID-19 and company knowledge graphs: Assessing golden powers and economic impact of selective lockdown via AI reasoning, CoRR, abs/2004.10119, 2020.

[49] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in: Kdd, vol. 96, 1996, pp. 226–231.

[50] L. Getoor, B. Taskar, Introduction To Statistical Relational Learning (Adaptive Computation and Machine Learning), The MIT Press, 2007.

[51] L.D. Raedt, Logical and Relational Learning: From ILP To MRDM (Cognitive Technologies), Springer-Verlag, Berlin, Heidelberg, 2008.

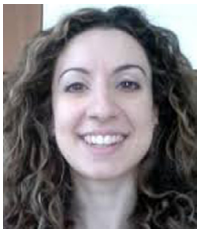
[52] L. De Raedt, A. Kimmig, Probabilistic (logic) programming concepts, Mach. Learn. 100 (1) (2015) 5–47.
[53] M. Richardson, P.M. Domingos, Markov logic networks, Mach. Learn. 62 (1–2) (2006) 107–136.
[54] H. Poon, P.M. Domingos, Unsupervised ontology induction from text, in: ACL, 2010, pp. 296–305.
[55] S. Albagli, R. Ben-Eliyahu-Zohary, S.E. Shimony, Markov network based ontology matching, J. Comput. System Sci. 78 (1) (2012) 105–118.
[56] P. Singla, P.M. Domingos, Entity resolution with Markov logic, in: ICDM, 2006, pp. 572–582.
[57] T. Sato, A statistical learning method for logic programs with distribution semantics, in: ICLP, MIT Press, 1995, pp. 715–729.
[58] L. Bellomarini, E. Laurenza, E. Sallinger, E. Sherkhonov, Reasoning under uncertainty in knowledge graphs, in: RuleML+RR, in: Lecture Notes in Computer Science, vol. 12173, Springer, 2020, pp. 131–139.
[59] R. Fagin, P.G. Kolaitis, R.J. Miller, L. Popa, Data exchange: Semantics and query answering, Theoret. Comput. Sci. 336 (1) (2005) 89–124.
[60] L. Libkin, Data exchange and incomplete information, in: SIGMOD, 2006, pp. 60–69.
[61] M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, P. Papotti, D. Santoro, E. Tsamoura, Benchmarking the chase, in: PODS, 2017, pp. 37–52.
[62] A.K. Chandra, M.Y. Vardi, The implication problem for functional and inclusion dependencies is undecidable, SIAM J. Comput. 14 (3) (1985) 671–677.
[63] L. Cao, Data science: A comprehensive overview, ACM Comput. Surv. 50 (3) (2017) 1–42.
[64] D.M. Blei, P. Smyth, Science and data science, Proc. Natl. Acad. Sci. 114 (33) (2017) 8689–8692.
[65] C. Ordonez, J. García-García, Managing big data analytics workflows with a database system, in: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid, IEEE, 2016, pp. 649–655.
[66] D. Gaspar, F. Porto, R. Akbarinia, E. Pacitti, Tardis: Optimal execution of scientific workflows in apache spark, in: International Conference on Big Data Analytics and Knowledge Discovery, Springer, 2017, pp. 74–87.
[67] G. Heidsieck, D. De Oliveira, E. Pacitti, C. Pradal, F. Tardieu, P. Valduriez, Distributed caching of scientific workflows in multisite cloud, in: International Conference on Database and Expert Systems Applications, Springer, 2020, pp. 51–65.
[68] R. Pichler, V. Savenkov, DEMo: Data exchange modeling tool, PVLDB 2 (2) (2009) 1606–1609.
[69] M. Meier, The backchase revisited, VLDB J. 23 (3) (2014) 495–516.
[70] S. Schulz, System description: E 1.8, in: LPAR, in: Lecture Notes in Computer Science, vol. 8312, Springer, 2013, pp. 735–743.
[71] A. Bonifati, I. Ileana, M. Linardi, Functional dependencies unleashed for scalable data exchange, in: SSDBM, ACM, 2016, pp. 2:1–2:12.
[72] M. Krötzsch, Efficient rule-based inferencing for OWL EL, in: IJCAI 2011, 2011, pp. 2668–2673.

Luigi Bellomarini

Ruslan R. Fayzrakhmanov

Georg Gottlob

Andrey Kravchenko



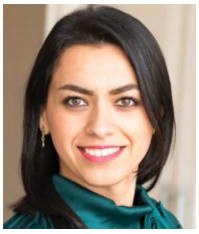
Eleonora Laurenza

Yavor Nenov

Stéphane Reissfelder

Emanuel Sallinger

Evgeny Sherkhonov



Sahar Vahdati

Lianlong Wu