Swift Logic for Big Data and Knowledge Graphs

Overview of Requirements, Language, and System

Luigi Bellomarini[1], Georg Gottlob[1,2(✉)], Andreas Pieris[3], and Emanuel Sallinger[1]

[1] Department of Computer Science, University of Oxford, Oxford, UK
georg.gottlob@gmail.com
[2] Institute of Information Systems, TU Wien, Vienna, Austria
[3] School of Informatics, University of Edinburgh, Edinburgh, UK

Abstract. Many modern companies wish to maintain knowledge in the form of a corporate knowledge graph and to use and manage this knowledge via a knowledge graph management system (KGMS). We formulate various requirements for a fully-fledged KGMS. In particular, such a system must be capable of performing complex reasoning tasks but, at the same time, achieve efficient and scalable reasoning over Big Data with an acceptable computational complexity. Moreover, a KGMS needs interfaces to corporate databases, the web, and machine-learning and analytics packages. We present KRR formalisms and a system achieving these goals. To this aim, we use specific suitable fragments from the Datalog$^{\pm}$ family of languages, and we introduce the VADALOG system, which puts these swift logics into action. This system exploits the theoretical underpinning of relevant Datalog$^{\pm}$ languages and combines it with existing and novel techniques from database and AI practice.

1 Introduction

The so-called *knowledge economy*, characteristic for the current Information Age, is rapidly gaining ground. According to [1], as cited in [29], "The knowledge economy is the use of knowledge [...] to generate tangible and intangible values. Technology, and, in particular, knowledge technology, help to transform a part of human knowledge to machines. This knowledge can be used by decision support systems in various fields and generate economic value." The importance of knowledge as an essential economic driving force has been evident to most corporate decision makers since the late 1970s, and the idea of storing knowledge and processing it to derive valuable new knowledge existed in the context of *expert systems*. Alas, it seems that the technology of those 'early' times was not sufficiently mature: the available hardware was too slow and main memory too tight for more complex reasoning tasks; database management systems were too slow and too rigid; there was no web where an expert system could acquire data; machine learning, and, in particular, neural networks were ridiculed as

This paper is a significantly abbreviated and slightly updated version of [4].

© Springer International Publishing AG 2018
A M. Tjoa et al. (Eds.): SOFSEM 2018, LNCS 10706, pp. 3–16, 2018.
https://doi.org/10.1007/978-3-319-73117-9_1

largely unsuccessful; ontological reasoning was in its infancy and the available formalisms were much too complex for Big Data applications. Meanwhile, there has been huge technological progress, and also much research progress that has led to a better understanding of many aspects of knowledge processing and reasoning with large amounts of data. Hardware has evolved, database technology has significantly improved, there is a (semantic) web with linked open data, companies can participate in social networks, machine learning has made a dramatic breakthrough, and there is a better understanding of scalable reasoning mechanisms.

Because of this, and of some eye-opening showcase projects such as IBM Watson [18], thousands of large and medium-sized companies suddenly wish to manage their own *knowledge graphs*, and are looking for adequate *knowledge graph management systems (KGMS)*.

The term *knowledge graph* originally only referred to Google's Knowledge Graph, namely, "a knowledge base used by Google to enhance its search engine's search results with semantic-search information gathered from a wide variety of sources" [30]. Meanwhile, further Internet giants (e.g. Facebook, Amazon) as well as some other very large companies have constructed their own knowledge graphs, and many more companies would like to maintain a private corporate knowledge graph incorporating large amounts of data in form of facts, both from corporate and public sources, as well as rule-based knowledge. Such a corporate knowledge graph is expected to contain relevant business knowledge, for example, knowledge about customers, products, prices, and competitors rather than mainly world knowledge from Wikipedia and similar sources. It should be managed by a KGMS, i.e., a knowledge base management system (KBMS), which performs complex rule-based reasoning tasks over very large amounts of data and, in addition, provides methods and tools for data analytics and machine learning, whence the equation:

$$\boxed{\text{KGMS} \;=\; \text{KBMS} + \text{Big Data} + \text{Analytics}}$$

The word 'graph' in this context is often misunderstood to the extent that some IT managers think that acquiring a graph database system and feeding it with data is sufficient to achieve a corporate knowledge graph. Others erroneously think that knowledge graphs necessarily use RDF triple stores instead of plain relational data. Yet others think that knowledge graphs are limited to storing and analyzing social network data only. While knowledge graphs should indeed be able to manipulate graph data and reason over RDF and social networks, they should not be restricted to this. For example, restricting a knowledge graph to contain RDF data only would exclude the direct inclusion of standard relational data and the direct interaction with corporate databases.

Not much has been described in the literature about the architecture of a KGMS and the functions it should ideally fulfil. In Sect. 2 we briefly list what we believe are the main requirements for a fully fledged KGMS. As indicated in Fig. 1, which depicts our reference architecture, the central component of a KGMS is its core reasoning engine, which has access to a rule repository. Grouped around it are various modules that provide relevant data access and analytics

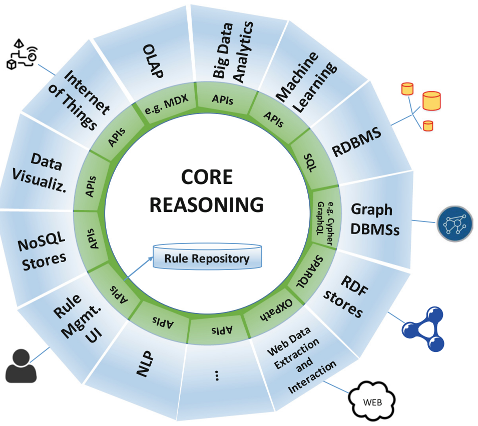

Fig. 1. KGMS reference architecture.

functionalities (see [4] for details). We expect a KGMS to fulfil many of these functions.

The reasoning core of a KGMS needs to provide a language for knowledge representation and reasoning (KRR). The data format for factual data should, as said, match the standard relational formalism so as to smoothly integrate corporate databases and data warehouses, and at the same time be suited for RDF and graph data. The rule language and reasoning mechanism should achieve a careful balance between expressive power and complexity. In Sect. 3 we present VADALOG, a Datalog-based language that matches this requirement. VADALOG belongs to the Datalog$^\pm$ family of languages that extend Datalog by existential quantifiers in rule heads, as well as by other features, and restricts at the same time its syntax so as to achieve decidability and data tractability; see, e.g., [5–8]. The logical core of the VADALOG language corresponds to *Warded Datalog*$^\pm$ [2,16], which captures plain Datalog as well as SPARQL queries using set semantics [22] under the entailment regime for OWL 2 QL [15], and is able to perform ontological reasoning tasks. Reasoning with the logical core of VADALOG is computationally efficient.

While the logical core of VADALOG has a number of beneificial properties, several features that have been added to it for achieving more powerful reasoning and data manipulation capabilities [4]. To give just one example here, the language is augmented by monotonic aggregations [26], which permits the use of aggregation (via summation, product, max, min, count) even in the presence of recursion. This enables us to swiftly solve problems such as the *company control* problem (studied e.g. in [10]) as explained in the following example.

Example 1 **(Example).** Assume the ownership relationship among a large number of companies is stored via facts (i.e., tuples of a database relation) of the following form Own($comp_1$, $comp_2$, w) meaning that company $comp_1$ directly owns a fraction w of company $comp_2$, with $0 \leq w \leq 1$. A company x controls a company y if x directly owns more than half of the shares of y or if x controls a set S of companies that jointly own more than half of y. Computing a predicate Control(x, y) expressing that company x controls company y, is then achieved in VADALOG by two rules:

$$\begin{aligned} \text{Own}(x, y, w), w > 0.5 &\rightarrow \text{Control}(x, y) \\ \text{Control}(x, y), \text{Own}(y, z, w), & \\ v = \texttt{msum}(w, \langle y \rangle), v > 0.5 &\rightarrow \text{Control}(x, z). \end{aligned}$$

Here, for fixed x, the aggregate construct $\texttt{msum}(w, \langle y \rangle)$ forms the sum over all values w such that for some company y, Control(x, y) is true, and Own(y, z, w) holds, i.e., company y directly owns fraction w of company z. ■

In [4] we introduce the VADALOG KGMS, which builds on the VADALOG language and combines it with existing and novel techniques from database and AI practice such as stream query processing, dynamic in-memory indexing and aggressive recursion control. The VADALOG system is Oxford's contribution to the VADA (*Value Added Data Systems*) research project [14,20,28], which is a joint effort of the universities of Edinburgh, Manchester, and Oxford.

2 Desiderata for a KGMS

In this section we briefly summarize what we think are the most important desiderata for a fully-fledged KGMS. We will list these requirements according to three categories, keeping in mind, however, that these categories are interrelated.

Language and System for Reasoning

There should be a logical formalism for expressing facts and rules, and a reasoning engine that uses this language, which should provide the following features.

Simple and Modular Syntax: It should be easy to add and delete facts and to add new rules. As in logic programming, facts should conceptually coincide with database tuples.

High Expressive Power: Datalog [10,19] is a good yardstick for the expressive power of rule languages. Over ordered structures (which we may assume here), Datalog with very mild negation captures PTIME; see, e.g., [11]. A rule language should thus ideally be at least as expressive as plain recursive Datalog, possibly with mild negation.

Numeric Computation and Aggregations: The basic logical formalism and inference engine should be enriched by features for dealing with numeric values, including appropriate aggregate functions.

Probabilistic Reasoning: The language should be suited for incorporating appropriate methods of probabilistic reasoning, and the system should propagate probabilities or certainty values along the reasoning process, that is, compute probabilities or certainty values for derived facts, and make adjustments wherever necessary. Probabilistic models may range from simple triangular norm operators (T-norm – cf [17]) over probabilistic database models [27] to Markov logic networks [23].

Ontological Reasoning: Ontological reasoning and query answering should be provided. We have two yardsticks here. First, ontological reasoning to the extent of tractable description logics such as DL-Lite$_R$ should be possible. Recall that DL-Lite$_R$ forms the logical underpinning of the OWL 2 QL profile of the Web Ontology Language as standardized by the W3C. Second, it should be expressive enough to cover all SPARQL queries under set semantics [22] over RDF datasets under the entailment regime for OWL 2 QL [15].

Low Complexity: Reasoning should be tractable in data complexity (i.e. when the rules are assumed to be fixed and the fact base is considered the input). Whenever possible, the system should recognize and take profit of rule sets that can be processed within low space complexity classes such as NLOGSPACE (e.g. for SPARQL) or even AC_0 (e.g. for traditional conjunctive database queries).

Rule Repository, Rule Management, and Ontology Editor: A library for storing recurring rules and definitions should be provided, as well as a user interface for rule management in the spirit of the ontology editor protégé [21].

Dynamic Orchestration: For larger applications, there must be a master module to allow the orchestration of complex data flows. For simple systems, the process must be easily specifiable. For complex systems, the process must be dynamically controllable through intelligent reasoning techniques or external control facilities and tools (e.g. BPM).

Accessing and Handling Big Data

Big Data Access: The system must be able to provide efficient access to Big Data sources and systems and fast reasoning algorithms over Big Data. In particular, the possibility of out-of-memory reasoning must be given in case the relevant data does not fit into main memory. Integration of Big Data processing techniques should be possible where the volume of data makes it necessary (see e.g. [25]).

Database and Data Warehouse Access: Seamless access to relational, graph databases, data warehouses, RDF stores, and major NoSQL stores should be granted. Data in such repositories should be directly usable as factual data for reasoning.

Ontology-Based Data Access (OBDA): OBDA [9] allows a system to compile a query that has been formulated on top of an ontology into one directly on the database. OBDA should be possible whenever appropriate.

Multi-query Support: Where possible and appropriate, partial results from repeated (sub-)queries should be evaluated once [24] and optimized in this regard.

Data Cleaning, Exchange and Integration: Integrating, exchanging and cleaning data should be supported both directly (through an appropriate KRR formalism that is made available through various applications in the knowledge repository), and by allowing integration of third-party software.

Web Data Extraction, Interaction, and IoT: A KGMS should be able to interact with the web by (i) extracting relevant web data (e.g. prices advertised by competitors) and integrating these data into the local fact base, and (ii) exchanging data with web forms and servers that are available through a web interface. One way to achieve this is given in [4]. Similar methods can be used for interacting with the IoT through appropriate network accessible APIs.

Embedding Procedural and Third-Party Code

Procedural Code: The system should have encapsulation methods for embedding procedural code (proprietary and third party) written in a variety of programming languages and offer a logical interface to it.

Third-Party Packages for Machine Learning, Text Mining, NLP, Data Analytics, and Data Visualization: The system should be equipped with direct access to powerful existing software packages for machine learning, text mining, data analytics, and data visualization. Given that excellent third-party software for these purposes exists, we believe that a KGMS should be able to use a multitude of such packages via appropriate logical interfaces.

3 Overview of the VADALOG Language and System

We here only give a brief overview of the VADALOG language and system. A more extensive overview of both language and system is given in [4] and the system is presented in detail in a forthcoming paper.

As said before, VADALOG is a KR language that achieves a careful balance between expressive power and complexity, and it can be used as the reasoning core of a KGMS. In Sect. 3.1 we discuss the logical core of VADALOG and some interesting fragments of it, while in Sect. 3.2 we discuss how this language can be extended with additional features that are much needed in real-world applications.

3.1 Core Language

The logical core of VADALOG is a member of the Datalog$^\pm$ family of knowledge representation languages, which we call Warded Datalog$^\pm$. The main goal of Datalog$^\pm$ languages is to extend the well-known language Datalog with useful modeling features such as existential quantifiers in rule heads (the '+' in the symbol '$\pm$'), and at the same time restrict the rule syntax in such a way that the decidability and data tractability of reasoning is guaranteed (the '$-$' in the symbol '$\pm$').

The core of Datalog$^\pm$ languages consists of rules known as *existential rules* or *tuple-generating dependencies*, which essentially generalize Datalog rules with existential quantifiers in rule heads; henceforth, we adopt the term existential rule. An example of such an existential rule is

$$\text{Person}(x) \;\rightarrow\; \exists y\, \text{HasFather}(x,y), \text{Person}(y)$$

which encodes that every person has a father who is also a person. In general, an existential rule is a first-order sentence

$$\forall \bar{x} \forall \bar{y} (\varphi(\bar{x}, \bar{y}) \;\rightarrow\; \exists \bar{z}\, \psi(\bar{x}, \bar{z}))$$

where φ (the *body*) and ψ (the *head*) are conjunctions of atoms with constants and variables.

The semantics of a set of existential rules Σ over a database D, denoted $\Sigma(D)$, is defined via the well-known chase procedure. Roughly, the chase adds new atoms to D (possibly involving null values used for satisfying the existentially quantified variables) until the final result $\Sigma(D)$ satisfies all the existential rules of Σ. Notice that, in general, $\Sigma(D)$ is infinite. Here is a simple example of the chase procedure.

Example 2. Consider the database $D = \{\text{Person}(\mathit{Bob})\}$, and the existential rule

$$\text{Person}(x) \;\rightarrow\; \exists y\, \text{HasFather}(x,y), \text{Person}(y).$$

The database atom triggers the above existential rule, and the chase adds in D the atoms

$$\text{HasFather}(\mathit{Bob}, \nu_1) \quad \text{and} \quad \text{Person}(\nu_1)$$

in order to satisfy it, where ν_1 is a (labeled) null representing some unknown value. The new atom $\text{Person}(\nu_1)$ triggers again the existential rule, and the chase adds the atoms

$$\text{HasFather}(\nu_1, \nu_2) \quad \text{and} \quad \text{Person}(\nu_2),$$

where ν_2 is a new null. The result of the chase is the instance

$$\{\text{Person}(\mathit{Bob}), \text{HasFather}(\mathit{Bob}, \nu_1)\} \;\cup\; \bigcup_{i>0} \{\text{Person}(\nu_i), \text{HasFather}(\nu_i, \nu_{i+1})\},$$

where $\nu_1, \nu_2, \ldots$ are (labeled) nulls. ■

Given a pair $Q = (\Sigma, \mathrm{Ans})$, where Σ is a set of existential rules and Ans an n-ary predicate, the evaluation of Q over a database D, denoted $Q(D)$, is defined as the set of tuples over the set C_D of constant values occurring in the database D that are entailed by D and Σ, i.e., the set

$$\{\langle t_1, \ldots, t_n\rangle \mid \mathrm{Ans}(t_1, \ldots, t_n) \in \Sigma(D) \text{ and each } t_i \in C_D\}.$$

The main reasoning task that we are interested in is *tuple inference*: given a database D, a pair $Q = (\Sigma, \mathrm{Ans})$, and a tuple of constants $\bar{t}$, decide whether $\bar{t} \in Q(D)$. This problem is very hard; in fact, it is undecidable, even when Q is fixed and only D is given as input [5]. This has led to a flurry of activity for identifying restrictions on existential rules that make the above problem decidable. Each such restriction gives rise to a new Datalog$^\pm$ language.

Warded Datalog$^\pm$: The Logical Core of VADALOG. The logical core of VADALOG relies on the notion of wardedness, which gives rise to Warded Datalog$^\pm$ [16]. In other words, VADALOG is obtained by extending Warded Datalog$^\pm$ with additional features of practical utility that are discussed in the next section.

Wardedness applies a restriction on how the "dangerous" variables of a set of existential rules are used. Intuitively, a "dangerous" variable is a body-variable that can be unified with a labeled null value when the chase algorithm is applied, and it is also propagated to the head of the rule. For example, given the set Σ consisting of the existential rules

$$P(x) \rightarrow \exists z\, R(x, z) \quad \text{and} \quad R(x, y) \rightarrow P(y),$$

the variable y in the body of the second rule is "dangerous" (w.r.t. Σ) since starting, e.g., from the database $D = \{P(a)\}$, the chase will apply the first rule and generate $R(a, \nu)$, where ν is a null that acts as a witness for the existentially quantified variable z, and then the second rule will be applied with the variable y being unified with ν that is propagated to the obtained atom $P(\nu)$. The goal of wardedness is to tame the way null values are propagated during the construction of the chase instance by posing the following conditions:

1. all the "dangerous" variables should coexist in a single body-atom α, called the ward, and
2. the ward can share only "harmless" variables with the rest of the body, i.e., variables that are unified only with database constants during the construction of the chase.

Warded Datalog$^\pm$ consists of all the (finite) sets of warded existential rules. The rule in Example 2 is clearly warded. Another example of a warded set of existential rules follows:

Example 3. Consider the following rules encoding part of the OWL 2 direct semantics entailment regime for OWL 2 QL (see [2,16]):

$$\begin{aligned}
\underline{\mathrm{Type}(x,y)}, \mathrm{Restriction}(y,z) &\rightarrow \exists w\, \mathrm{Triple}(x,z,w)\\
\underline{\mathrm{Type}(x,y)}, \mathrm{SubClass}(y,z) &\rightarrow \mathrm{Type}(x,z)\\
\underline{\mathrm{Triple}(x,y,z)}, \mathrm{Inverse}(y,w) &\rightarrow \mathrm{Triple}(z,w,x)\\
\underline{\mathrm{Triple}(x,y,z)}, \mathrm{Restriction}(w,y) &\rightarrow \mathrm{Type}(x,w).
\end{aligned}$$

It is easy to verify that the above set is warded, where the underlined atoms are the wards. Indeed, a variable that occurs in an atom of the form Restriction$(\cdot,\cdot)$, or the form SubClass$(\cdot,\cdot)$, or Inverse$(\cdot,\cdot)$, is trivially harmless. However, variables that appear in the first position of Type, or in the first/third position of Triple can be dangerous. Thus, the underlined atoms are indeed acting as the wards.

Let us now intuitively explain the meaning of the above set of existential rules: The first rule states that if a is of type b, encoded via the atom Type(a,b), while b represents the class that corresponds to the first attribute of some binary relation c, encoded via the atom Restriction(b,c), then there exists some value d such that the tuple (a,d) occurs in the binary relation c, encoded as the atom Triple(a,c,d). Analogously, the other rules encode the usual meaning of subclasses, inverses and the effect of restrictions on types. ■

Let us clarify that Warded Datalog$^\pm$ is a refinement of the language of *Weakly-Frontier-Guarded Datalog*$^\pm$, which is defined in the same way but without the condition (2) given above [3]. Weakly-Frontier-Guarded Datalog$^\pm$ is highly intractable in data complexity; in fact, it is EXPTIME-complete. This justifies Warded Datalog$^\pm$, which is a (nearly) maximal tractable fragment of Weakly-Frontier-Guarded Datalog$^\pm$.

Warded Datalog$^\pm$ enjoys several favourable properties that make it a robust core towards more practical languages:

- Tuple inference under Warded Datalog$^\pm$ is data tractable; in fact, it is PTIME-complete when the set of rules is fixed.
- Warded Datalog$^\pm$ contains full Datalog as sub-language without increasing the complexity. Indeed, a set Σ of Datalog rules is trivially warded since there are no dangerous variables (w.r.t. Σ).
- Warded Datalog$^\pm$ generalizes central ontology languages such as the OWL 2 QL profile of OWL, which in turn relies on the prominent description logic DL-Lite$_R$.
- Warded Datalog$^\pm$ is suitable for querying RDF graphs. Actually, by adding stratified and grounded negation to Warded Datalog$^\pm$, we obtain a language, called TriQ-Lite 1.0 [16], that can express every SPARQL query using set semantics [22] under the entailment regime for OWL 2 QL.

3.2 Extensions

In order to be effective for real-world applications, we extend the logical core of VADALOG described above with a set of additional features of practical utility.

Although the theoretical properties of the language are no longer guaranteed, our preliminary evaluation has shown that the practical overhead for many of these features remains reasonable in our streaming implementation. In the future, we plan to perform a more thorough complexity analysis and isolate sets of features for which beneficial complexity upper bounds are met and runtime guarantees are given.

Data Types: Variables and constants are typed. The language supports the most common simple data types: integer, float, string, Boolean, date. There is also support for composite data types, such as sets.

Expressions: Variables and constants can be combined into expressions, which are recursively defined as variables, constants or combinations thereof, for which we support many different operations for the various data types: algebraic sum, multiplication, division for integers and floats; containment, addition, deletion of set elements; string operations (contains, starts-with, ends-with, index-of, substring, etc.); Boolean operations (and, or, not, etc.). Expressions can be used in rule bodies (1) as the left-hand side (LHS) of a *condition*, i.e., the comparison (>, <, >=, <=, <>) of a body variable with the expression itself; (2) as the LHS of an *assignment*, i.e., the definition of a specifically calculated value, potentially used as an existentially quantified head variable. In our running example, variable v is calculated with the expression $\texttt{msum}(w, \langle y \rangle)$ and used in the condition $v > 0.5$.

Skolem Functions: Labeled null values can be suitably calculated with functions defined on-the-fly. They are assumed to be deterministic (returning unique labeled nulls for unique input bindings), and to have disjoint ranges.

Monotonic Aggregations: VADALOG supports aggregation (*min*, *max*, *sum*, *prod*, *count*), by means of an extension to the notion of monotonic aggregations [26], which allows adopting aggregation even in the presence of recursion while preserving monotonicity w.r.t. set containment. The company control example shows the use of `msum`, which calculates variable v, as the monotonically increasing sum of the quota w of company z owned by y, in turn controlled by x. The sum is accumulated so that above the threshold 0.5, we have that x controls z. Recent applications of VADALOG in challenging industrial use cases showed that such aggregations are very efficient in many real-world Big Data settings.

Data Binding Primitives: Data sources and targets can be declared by adopting *input/output annotations*, a.k.a. *binding patterns*. Annotations are special facts augmenting sets of existential rules with specific behaviours. The unnamed perspective used in VADALOG can be harmonized with the named perspective of many external systems by means of *bind* and *mapping* annotations, which also support *projection*. A special *query bind* annotation also supports binding predicates to queries against inputs/outputs (in the external language, e.g., SQL-queries for a data source or target that supports SQL). In our example, the extension of the Own predicate is our input, which we denote with an @`input`(“`Own`”) annotation. The actual facts then may be derived, e.g., from a relational or graph database, which we would respectively access with the two following annotations (the latter one using neo4j's cypher graph query language):

```
@bind("Own", "rdbms", "companies.ownerships").
   @qbind("Own", "graphDB",
      "MATCH (a)-[o:Owns]->(b)
         RETURN a,b,o.weight").
```

A similar approach is also used for bridging external machine learning and data extraction platforms into the system. This uses binding patterns as a form of *behaviour injection*: the atoms in rules are decorated with binding annotations, so that a step in the reasoning process triggers the external component. We give a simple example using the OXPath [13] large-scale web data extraction framework (developed as part of the DIADEM project [12]) – an extension of XPath that interacts with web applications to extract information obtained during web navigation. In our running example, assume that our local company ownership information is only partial, while more complete information can be retrieved from the web. In particular, assume that a company register acts as a web search engine, taking as input a company name and returning, as separate pages, the owned companies. This information can be obtained as follows:[1]

```
@qbind("Own", "oxpath",
   "doc('http://company_register.com/ownerships')
   /descendant::field()[1]/{$1}
   /following::a[.#='Search']/{click/}
      /(//a[.#='Next']/ {click/})*
         //div[@class='c']:<comp>
         [./span[1]:<name=string(.)>]
         [./span[3]:<percent=string(.)>]").
```

The above examples show a basic bridging between the technologies. Interesting interactions can be seen in more sophisticated scenarios, where the reasoning process and external component processing is more heavily interleaved.

Probabilistic Reasoning: VADALOG offers support for the basic cases in which scalable computation can be guaranteed. Facts are assumed to be probabilistically independent and a minimalistic form of probabilistic inference is offered as a side product of query answering. Facts can be adorned with probability measures according to the well-known possible world semantics [27]. Then, if the set of existential rules respects specific syntactic properties that guarantee probabilistic tractability (namely, a generalization of the notion of *hierarchical queries* [27]), the facts resulting from query answering are enriched with their marginal probability, safely calculated in a scalable way. In the following extension to our running example, we use probabilistic reasoning to account for uncertain ownerships (e.g., due to unreliable sources), prefixing the facts

[1] Concretely, the first position of the Own predicate is bound to the `$1` placeholder in the OXPath expression.

with their likelihood, so as to derive non-trivial conclusions on company control relationships:

0.8 :: Own(“`ACME`”, “`COIN`”, 0.7)
0.3 :: Own(“`COIN`”, “`SAVERS`”, 0.3)
0.4 :: Own(“`ACME`”, “`GYM`”, 0.55)
0.6 :: Own(“`GYM`”, “`SAVERS`”, 0.4).

In total, the language allows bridging logic-based reasoning and machine learning in three ways. First, the language supports scalable probabilistic inference in basic cases as seen above. Second, the extensions to the core language provide all the necessary features to abstract and embed advanced inference algorithms (e.g. belief propagation) so that they can be executed directly by the VADALOG system, and hence leverage its optimization strategies. Third, for the more sophisticated machine learning applications, data binding primitives allow a simple interaction with specialized libraries and systems as described before.

Post-processing Annotations: Since specific computations are often needed after the result has been produced, VADALOG supports many of them by means of annotations for the following features: *ordering* of the resulting values, as set semantics is assumed on the output, and yet a particular ordering of the facts may be desired by the consumer: for example, `@orderby`(“`Control`”, 1) sorts the obtained control facts by the controlling company; *deduplication*, in specific conditions (e.g. in presence of calculated values), the output may physically contain undesired duplicates; *non-monotonic aggregations* on the final result, without the limitations induced by recursion; and *certain answers*.

4 Conclusion

In this paper, we have formulated a number of requirements for a KGMS, which led us to postulate our reference architecture (see Fig. 1). Based on these requirements, we introduced the VADALOG language whose core corresponds to Warded Datalog$^{\pm}$. The basic VADALOG language is extended by features for numeric computations, monotonic aggregation, probabilistic reasoning, and, moreover, by data binding primitives used for interacting with the corporate and external environment. These binding primitives allow the reasoning engine to access and manipulate external data through the lens of a logical predicate. The external data may stem from a corporate database, may be extracted from web pages, or may be the output of a machine-learning program that has been evaluated over previously computed data relations. The VADALOG system, which is being implemented at the University of Oxford, puts these swift logics into action. This system exploits the theoretical underpinning of Warded Datalog$^{\pm}$ and combines it with existing and novel techniques from database and AI practice.

Many core features of the VADALOG system [4] are already integrated and show good performance. Our plan is to complete the system in the near future. A detailed report on the key technical features of the VADALOG reasoning system and on their implementation is already available on request from the authors. We believe that the VADALOG system is a well-suited platform for applications that integrate machine learning (ML) and data analytics with logical reasoning. We are currently implementing applications of this type and will report about them soon.

Acknowledgments. This work has been supported by the EPSRC Programme Grant EP/M025268/1 "VADA – Value Added Data Systems". The VADALOG system as presented here is the intellectual property of the University of Oxford.

References

1. Amidon, D.M., Formica, P., Mercier-Laurent, E.: Knowledge Economics: Emerging Principles. Tartu University Press Tartu, Pactices and Policies (2005)
2. Arenas, M., Gottlob, G., Pieris, A.: Expressive languages for querying the semantic web. In: PODS, pp. 14–26 (2014)
3. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: walking the decidability line. Artif. Intell. **175**(9–10), 1620–1654 (2011)
4. Bellomarini, L., Gottlob, G., Pieris, A., Sallinger, E.: Swift logic for big data and knowledge graphs. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, 19–25 August 2017, pp. 2–10. ijcai.org (2017). https://doi.org/10.24963/ijcai.2017/1
5. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: query answering under expressive relational constraints. J. Artif. Intell. Res. **48**, 115–174 (2013)
6. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. J. Web Sem. **14**, 57–83 (2012)
7. Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., Pieris, A.: Datalog+/−: a family of logical knowledge representation and query languages for new applications. In: LICS, pp. 228–242 (2010)
8. Calì, A., Gottlob, G., Pieris, A.: Towards more expressive ontology languages: the query answering problem. Artif. Intell. **193**, 87–128 (2012)
9. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The mastro system for ontology-based data access. Semant. Web **2**(1), 43–53 (2011)
10. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-83952-8
11. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. **33**(3), 374–425 (2001)
12. Furche, T., Gottlob, G., Grasso, G., Guo, X., Orsi, G., Schallhart, C., Wang, C.: DIADEM: thousands of websites to a single database. PVLDB **7**(14), 1845–1856 (2014). http://www.vldb.org/pvldb/vol7/p1845-furche.pdf
13. Furche, T., Gottlob, G., Grasso, G., Schallhart, C., Sellers, A.J.: Oxpath: a language for scalable data extraction, automation, and crawling on the deep web. VLDB J. **22**(1), 47–72 (2013)
14. Furche, T., Gottlob, G., Neumayr, B., Sallinger, E.: Data wrangling for big data: towards a lingua franca for data wrangling. In: AMW (2016)
15. Glimm, B., Ogbuji, C., Hawke, S., Herman, I., Parsia, B., Polleres, A., Seaborne, A.: SPARQL 1.1 entailment regimes, 2013. W3C Recommendation, 21 March 2013

16. Gottlob, G., Pieris, A.: Beyond SPARQL under OWL 2 QL entailment regime: rules to the rescue. In: IJCAI, pp. 2999–3007 (2015)
17. Hájek, P.: Metamathematics of Fuzzy Logic. Springer, Heidelberg (1998). https://doi.org/10.1007/978-94-011-5300-3
18. High, R.: The era of cognitive systems: an inside look at IBM Watson and how it works. IBM, Redbooks (2012)
19. Huang, S.S., Green, T.J., Loo, B.T.: Datalog and emerging applications: an interactive tutorial. In: SIGMOD, pp. 1213–1216. ACM (2011)
20. Konstantinou, N., Koehler, M., Abel, E., Civili, C., Neumayr, B., Sallinger, E., Fernandes, A.A.A., Gottlob, G., Keane, J.A., Libkin, L., Paton, N.W.: The VADA architecture for cost-effective data wrangling. In: SIGMOD, pp. 1599–1602 (2017)
21. Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W., Musen, M.A.: Creating semantic web contents with protege-2000. IEEE IS **16**(2), 60–71 (2001)
22. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 16:1–16:45 (2009). https://doi.org/10.1145/1567274.1567278
23. Richardson, M., Domingos, P.M.: Markov logic networks. Mach. Learn. **62**(1–2), 107–136 (2006)
24. Roy, P., Seshadri, S., Sudarshan, S., Bhobe, S.: Efficient and extensible algorithms for multi query optimization. In: SIGMOD, pp. 249–260 (2000)
25. Shkapsky, A., Yang, M., Interlandi, M., Chiu, H., Condie, T., Zaniolo, C.: Big data analytics with datalog queries on spark. In: SIGMOD, pp. 1135–1149 (2016). http://doi.acm.org/10.1145/2882903.2915229
26. Shkapsky, A., Yang, M., Zaniolo, C.: Optimizing recursive queries with monotonic aggregates in deals. In: ICDE, pp. 867–878 (2015)
27. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Morgan & Claypool, San Rafael (2011)
28. VADA: Project Website (2016). http://vada.org.uk/. Accessed 19 May 2017
29. Wikipedia: Knowledge economy (2017). https://en.wikipedia.org/wiki/Knowledge_economy. Accessed 19 May 2017
30. Wikipedia: Knowledge graph (2017). https://en.wikipedia.org/wiki/Knowledge_graph. Accessed 19 May 2017