The VADALOG System: Swift Logic for Big Data and Enterprise Knowledge Graphs*

Luigi Bellomarini[1,2], Georg Gottlob[1,3], Andreas Pieris[4], Emanuel Sallinger[1]

[1] University of Oxford, UK
[2] Banca d'Italia, Italy
[3] TU Wien, Austria
[4] University of Edinburgh, UK

1 Introduction

Many modern companies wish to maintain knowledge in the form of an enterprise knowledge graph [21] and to use and manage this knowledge via *knowledge graph management systems* (KGMS). We view a KGMS as 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:

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

In this paper, we summarize various requirements for a fully-fledged KGMS. Such a system must be capable of performing complex reasoning tasks, guaranteeing, at the same time, efficiency and scalability over Big Data with an acceptable computational complexity. Moreover, a KGMS needs interfaces to many heterogeneous data sources, including: corporate RDBMS, NoSQL stores, the web, machine-learning and analytics packages. We present *knowledge representation and reasoning* formalisms and a system achieving these goals. To this aim, we adopt specific suitable fragments from the Datalog$^{\pm}$ family of languages [3, 5, 6, 8, 9], and we introduce the VADALOG system (following the reference architecture shown in Fig. 1), which puts these swift logics into action. Our system exploits the theoretical underpinnings of relevant Datalog$^{\pm}$ languages, combining them with existing and novel techniques from database engineering and AI practice.

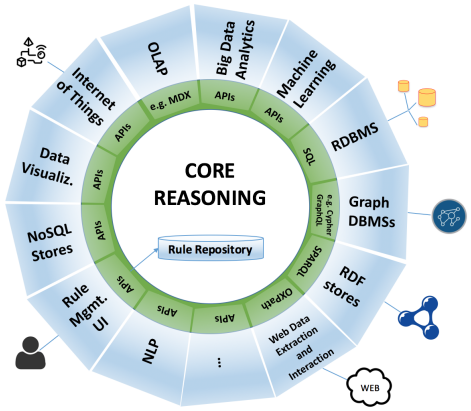

Fig. 1. KGMS Reference Architecture.

This paper is a short version of a fully detailed paper published in [2]. The VADALOG system is Oxford's contribution to VADA [20], a joint project of the universities of Edinburgh, Manchester, and Oxford. We reported first work on the overall VADA approach to data wrangling in [14]. In this paper, we focus on the VADALOG system at its core. Our system currently fully implements the core language and is already in use for a number of industrial applications. Many extensions, especially those important for our partners, are already realized, but others are still under development and will be integrated in the future. We conducted extensive benchmarks on the system, and the results are very promising in that they show good performance and scalability for large knowledge graphs.

* This paper is a short version of [2].

2 Desiderata for a KGMS

We now proceed to briefly summarize what we think are the most important desiderata for a fully-fledged KGMS. We will list these requirements according to two 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 [12, 17] is a good yardstick for the expressive power of rule languages. Over ordered structures, Datalog with very mild negation captures PTIME; see, e.g., [13]. 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.

Ontological Reasoning and Query Answering: First, ontological reasoning to the extent of tractable description logics such as DL-Lite$_R$ should be possible. Second, it should be expressive enough to cover all SPARQL queries over RDF datasets under the entailment regime for OWL 2 QL [15].

Low Complexity: Reasoning should be tractable in data complexity 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 [18].

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 (see e.g. [19]).

Database 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 [11] 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.

Data Cleaning, Exchange and Integration: Integrating, exchanging and cleaning data should be supported directly and via 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 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.

Procedural Code and Machine Learning: A KGMS should have encapsulation methods for embedding procedural code and offer a logical interface to it. The system should be equipped with direct access to existing software packages for *machine learning*, *text mining*, *data analytics*, and *data visualization*.

3 The VADALOG Language

VADALOG is a Datalog-based language that matches the requirements presented in Section 2. It 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 in order to achieve decidability and data tractability; see, e.g., [4, 5, 7, 10]. The logical core of the VADALOG language corresponds to *Warded Datalog*$^{\pm}$ [1, 16], which captures plain Datalog as well as SPARQL queries 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. VADALOG is obtained by extending Warded Datalog$^{\pm}$ with additional features of practical utility. We now illustrate the logical core of VADALOG, while a discussion of the additional features is in the full paper [2].

The logical core of VADALOG relies on the notion of wardedness, which 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 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; 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. As an example of a warded set of rules, the following rules encode part of the OWL 2 direct semantics entailment regime for OWL 2 QL (see [1, 16]):

$$\underline{\text{Type}(x, y)}, \text{Restriction}(y, z) \rightarrow \exists w\, \text{Triple}(x, z, w)$$
$$\underline{\text{Type}(x, y)}, \text{SubClass}(y, z) \rightarrow \text{Type}(x, z)$$
$$\underline{\text{Triple}(x, y, z)}, \text{Inverse}(y, w) \rightarrow \text{Triple}(z, w, x)$$
$$\underline{\text{Triple}(x, y, z)}, \text{Restriction}(w, y) \rightarrow \text{Type}(x, w).$$

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 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.

4 The VADALOG System

The functional architecture of the VADALOG system, our KGMS, is depicted in Figure 1. The knowledge graph is organized as a repository, a collection of VADALOG rules. The external sources are supported by means of *transducers*, intelligent adapters that integrate the sources into the reasoning process.

The VADALOG system fulfils the requirements presented in Section 2. The Big Data characteristics of the sources and the complex functional requirements of reasoning are tackled by leveraging the underpinnings of the core language, which are turned into practical execution strategies. In particular, in the reasoning algorithms devised for Warded Datalog$^{\pm}$, after a certain number of chase steps (which, in general, depends on the input database), the chase graph [5] (a directed acyclic graph where facts are represented as nodes and the applied rules as edges) exhibits specific periodicities and no new information, relevant to query answering, is generated. The VADALOG system adopts an *aggressive recursion and termination control* strategy, which detects such redundancy as early as possible by combining compile-time and runtime techniques. In combination with a highly engineered architecture, the VADALOG system achieves high performance and an efficient memory footprint.

At compile time, thanks to wardedness, which limits the interaction between the labeled nulls, the engine rewrites the program in such a way that joins on specific values of labeled nulls will never occur. At runtime, we do an *eager optimal pruning* of redundant chase branches: we exploit fact *provenance* to preempt the application of rules which will generate redundant facts. Due to wardedness, the provenance information needed is bounded.

The VADALOG system uses a pull stream-based approach (or pipeline approach), where the facts are actively requested from the output nodes to their predecessors and so on down to the input nodes, which eventually fetch the facts from the data sources. The stream approach is essential to limit the memory consumption or, at least make it predictable, so that the system is effective for large volumes of data. Our setting is made more challenging by the presence of multiple interacting rules in a single rule set and the wide presence of recursion. We address this by means of a specialized buffer management technique. We adopt pervasive local caches in the form of wrappers to the nodes of the access plan, where the facts produced by each node are stored. The local caches work particularly well in combination with the pull stream-based approach, since facts requested by a node successor can be immediately reused by all the other successors, without triggering further backward requests. Also, this combination realizes an extreme form of multi-query optimization, where each rule exploits the facts produced by the others, whenever applicable. To limit memory occupation, the local caches are flushed with an eager eviction strategy that detects when a fact has been consumed by all the possible requestors and thus drops it from the memory. Cases of actual cache overflow are managed by resorting to standard disk swap heuristics (e.g. LRU, LFU, etc.)

5 Conclusion

The VADALOG system is already in use for a number of industrial applications. We believe that the VADALOG system is a well-suited platform for knowledge graph 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 (`http://vada.org.uk/`). The VADALOG system as presented here is the intellectual property of the University of Oxford.

References

1. M. Arenas, G. Gottlob, and A. Pieris. Expressive languages for querying the semantic web. In *PODS*, pages 14–26, 2014.
2. L. Bellomarini, G. Gottlob, A. Pieris, and E. Sallinger. Swift logic for big data and knowledge graphs. In *IJCAI*, pages 2–10, 2017.
3. A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res*, 48:115–174, 2013.
4. A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.
5. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.
6. A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *Logic in Computer Science (LICS), 2010 25th Annual IEEE Symposium on*, pages 228–242. IEEE, 2010.
7. A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS*, pages 228–242, 2010.
8. A. Calì, G. Gottlob, and A. Pieris. New expressive languages for ontological query answering. In *Proc. of AAAI*, volume 2011, 2011.
9. A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence*, 193:87–128, 2012.
10. A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.
11. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The mastro system for ontology-based data access. *Semantic Web*, 2(1):43–53, 2011.
12. S. Ceri, G. Gottlob, and L. Tanca. *Logic programming and databases*. Springer, 2012.
13. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
14. T. Furche, G. Gottlob, B. Neumayr, and E. Sallinger. Data wrangling for big data: Towards a lingua franca for data wrangling. In *AMW*, volume 1644 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.
15. B. Glimm, C. Ogbuji, S. Hawke, I. Herman, B. Parsia, A. Polleres, and A. Seaborne. SPARQL 1.1 entailment regimes, 2013. W3C Recommendation 21 March 2013, 2013.
16. G. Gottlob and A. Pieris. Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue. In *IJCAI*, pages 2999–3007, 2015.
17. S. S. Huang, T. J. Green, and B. T. Loo. Datalog and emerging applications: an interactive tutorial. In *SIGMOD*, pages 1213–1216. ACM, 2011.
18. N. F. Noy, M. Sintek, S. Decker, M. Crubézy, R. W. Fergerson, and M. A. Musen. Creating semantic web contents with protege-2000. *IEEE IS*, 16(2):60–71, 2001.
19. A. Shkapsky, M. Yang, M. Interlandi, H. Chiu, T. Condie, and C. Zaniolo. Big Data Analytics with Datalog Queries on Spark. In *SIGMOD*, pages 1135–1149, 2016.
20. VADA. Project Website. `http://vada.org.uk/`, 2016. [Online; accessed 3-Mar-2018].
21. Wikipedia. Knowledge graph. `https://en.wikipedia.org/wiki/Knowledge_Graph`, 2017. [Online; accessed 3-Mar-2018].