Reasoning on Web Data: Algorithms and Performance

Damian Bursztyn, François Goasdoué, Ioana Manolescu, Alexandra Roatis

Reasoning on Web Data: Algorithms and Performance

Proposed duration: 1.5 hours

Damian Bursztyn [⋆•1], François Goasdoué [◇⋆2], Ioana Manolescu [⋆•3], Alexandra Roatiş [⋆•4]

[◇] *Université Rennes 1, France* [⋆] *INRIA, France* [•] *Université Paris-Sud, France*

[1] damian.bursztyn@inria.fr [2] fg@irisa.fr [3] ioana.manolescu@inria.fr [4] alexandra.roatis@lri.fr

***Abstract*—Techniques for efficiently managing Semantic Web data have attracted significant interest from the data management and knowledge representation communities. A great deal of effort has been invested, especially in the database community, into algorithms and tools for efficient RDF query evaluation. However, the main interest of RDF lies in its blending of *heterogeneous data and semantics*. Simple RDF graphs can be seen as collections of facts, which may be further enriched with *ontological schemas*, or semantic constraints, based on which *reasoning* can be applied to infer new information. Taking into account this implicit information is crucial for answering queries.**

The literature provides two classes of techniques for implementing RDF reasoning, namely *query reformulation* and *saturation*. Both are based on the idea of decoupling RDF entailment – the reasoning mechanism based on which query answers are defined – from query evaluation; the performance of the respective algorithms depends on the expressive power of the ontological schema language, as well as on the subset of features from the RDF standard which is supported.

Our tutorial introduces the RDF ontological schema language for enhancing the RDF graphs' semantics, formalizes the query answering problem relying on reasoning, and provides a principled classification and analysis of the two techniques, with a particular focus on their performance trade-offs.

I. Motivation

Undoubtedly, there is currently strong interest within the Semantic Web, Databases and Knowledge Representation communities in techniques for *efficiently managing complex, semantic-rich Web data*. This is demonstrated by very active research, on one hand on indexing, query processing and dedicated systems for storing, querying, and updating such data; and on the other hand in expressive semantic constraint languages to use for describing its meaning. While early systems were developed for single-server settings, the large-scale distributed management of Web data graphs (for instance, in a cloud environment, based on MapReduce, on distributed memory etc.) is an extremely active topic now [1], [2].

However, in our experience, knowledge on Semantic Web data management is currently split between the specialists of "query evaluation", which consider large databases and complex queries, but tend to ignore data semantics, and the experts in "reasoning", whose main focus is on formal reasoning issues. In particular, this leads to reasoning aspects being rarely considered in database systems and prototypes handling RDF data, limiting their usability in actual RDF usage scenarios where semantic constraints are frequently used to compactly encode crucial information about the application domain [3].

One explanation for this relative lack of interest in reasoning within the database community may be the fact that one can "compile the knowledge into data", as follows. Infer all possible facts derived from the base data and by the semantic constraints *prior* to querying the database; add these facts to the database; then, process queries on the (enriched) database ignoring the constraints which have lead to it. This brings the problem of *query answering* (computing sound and complete answers based on the data and the semantics) to the relatively simpler one of *query evaluation*, which can be efficiently delegated to database engines. For instance, if the database only holds that "Tom is a cat" and the axiom that "any cat is a mammal", one can add to the database the fact that "Tom is a mammal" derived from the original fact and semantic constraint, and work on the resulting two-fact table, while discarding the constraint. This technique is termed *saturation* or *closure*.

While correct, such a technique raises performance issues when the data is dynamic. First, if the base data changes, one has to update the set of inferred facts, in order to reflect the changes in the base data. The same applies in the case of changes to the set of semantic constraints; in real life scenarios, one constraint is typically used to derive more than one new fact, thus database maintenance when the set of axiom (constraints) changes can be particularly expensive. In a classical centralized database setting, the semantic constraints are specified prior to populating the database and seldom change; in contrast, typical Semantic Web scenarios involve integrating data from *several RDF repositories*, also called "RDF endpoints". Since such repositories are often authored independently, they have their own sets of semantic constraints (or schemas, in short); computing prior to query answering all the consequences of facts from any endpoint and constraints from any (other) endpoint is not feasible.

The alternative technique is called *reformulation*. Here, the database is left unchanged, while queries are modified (reformulated) to take into account all the known semantic constraints. In the simple example above, a query asking for all mammals would be reformulated into "find all mammals *and* all cats as particular cases", and Tom would be returned even though it was not explicitly stated to be a mammal. While this technique does not require changing the database, it often leads to syntactically larger reformulated queries, whose efficient evaluation remains challenging.

Our tutorial introduces the existing techniques for semantic-aware query answering on Web data (in particular, RDF); its target audience comprises students, researchers and practitioners with an interest in Web data management. We analyze and classify existing algorithms, and outline their respective strengths and weaknesses from a performance perspective, before discussing currently open problems.

The tutorial has not been presented previously.

In the sequel, Section II details the tutorial organization, while Section III gives a short bibliography of the presenters.

II. Tutorial Organization

This tutorial is organized in four parts.

As background (Section II-A), we recall the basics of the RDF data model, most widely used to represent Web data graphs, and of its popular ontological schema language RDFS. We also presents the query dialect we consider, namely the widely used BGP (basic graph pattern) queries of SPARQL[1].

The technical core of our tutorial is outlined in Section II-B, where we describe the two main query answering techniques used to compute correct answers from semantics-rich RDF graphs, we classify existing algorithms, and point to the performance trade-offs involved. We outline the reasoning support available in today's most prominent tools and systems (Section II-C), finally, we highlight open issues (Section II-D).

A. RDF and RDFS

The Resource Description Framework [4] is a graph-based data model accepted as the W3C standard for Semantic Web applications. An *RDF graph* (or *graph*, in short) is a set of *triples* of the form $\mathtt{s}\ \mathtt{p}\ \mathtt{o}$. A triple states that its *subject* $\mathtt{s}$ has the *property* $\mathtt{p}$, and the value of that property is the *object* $\mathtt{o}$. We consider well-formed RDF triples [4], using uniform resource identifiers (URIs), typed or un-typed literals (constants) and blank nodes (unknown URIs or literals).

Figure 1 (top) shows how to use triples to describe resources, that is, to express class (unary relation) and property (binary relation) assertions. The RDF standard [4] provides a set of built-in classes and properties, as part of the rdf: and rdfs: pre-defined namespaces, e.g., rdf:type specifies the class(es) to which a resource belongs.

RDF Schema (RDFS) [5] is used to state *semantic constraints* between the classes and the properties used in those graphs. Figure 1 (bottom) shows the four main constraints and how to express them; *domain* and *range* denote respectively the first and second attribute of every property. The RDFS constraints (Figure 1) are interpreted under the open-world assumption (OWA) [6]. For instance, given two relations R_1, R_2, the OWA interpretation of the constraint $R_1 \subseteq R_2$ is: any tuple t in the relation R_1 *is considered as being also in the relation* R_2 (the inclusion constraint *propagates* t *to* R_2). Specifically, if the triples hasFriend rdfs:domain Person and Anne hasFriend Marie hold in the graph, then so does the triple Anne rdf:type Person. The latter is due to the *domain typing* constraint in Figure 1.

[1]The reader well-familiar with RDF, RDFS and SPARQL may safely skip Section II-A.

Assertion	Triple	Relational notation
Class	$\mathtt{s}$ rdf:type $\mathtt{o}$	$\mathtt{o}(\mathtt{s})$
Property	$\mathtt{s}\ \mathtt{p}\ \mathtt{o}$	$\mathtt{p}(\mathtt{s}, \mathtt{o})$

Constraint	Triple	OWA interpretation
Subclass	$\mathtt{s}$ rdfs:subClassOf $\mathtt{o}$	$\mathtt{s} \subseteq \mathtt{o}$
Subproperty	$\mathtt{s}$ rdfs:subPropertyOf $\mathtt{o}$	$\mathtt{s} \subseteq \mathtt{o}$
Domain typing	$\mathtt{s}$ rdfs:domain $\mathtt{o}$	$\Pi_{\text{domain}}(\mathtt{s}) \subseteq \mathtt{o}$
Range typing	$\mathtt{s}$ rdfs:range $\mathtt{o}$	$\Pi_{\text{range}}(\mathtt{s}) \subseteq \mathtt{o}$

Fig. 1. RDF (top) & RDFS (bottom) statements.

Rule [4]	*Instance entailment from combining schema and instance triples*
rdfs9	$\mathtt{c}_1$ rdfs:subClassOf $\mathtt{c}_2 \ \wedge\ \mathtt{s}$ rdf:type $\mathtt{c}_1$ $\vdash^{\mathbf{rdfs9}}_{\text{RDF}}$ $\mathtt{s}$ rdf:type $\mathtt{c}_2$
rdfs7	$\mathtt{p}_1$ rdfs:subPropertyOf $\mathtt{p}_2 \ \wedge\ \mathtt{s}\ \mathtt{p}_1\ \mathtt{o}$ $\vdash^{\mathbf{rdfs7}}_{\text{RDF}}$ $\mathtt{s}\ \mathtt{p}_2\ \mathtt{o}$
rdfs2	$\mathtt{p}$ rdfs:domain $\mathtt{c} \ \wedge\ \mathtt{s}\ \mathtt{p}\ \mathtt{o}$ $\vdash^{\mathbf{rdfs2}}_{\text{RDF}}$ $\mathtt{s}$ rdf:type $\mathtt{c}$
rdfs3	$\mathtt{p}$ rdfs:range $\mathtt{c} \ \wedge\ \mathtt{s}\ \mathtt{p}\ \mathtt{o}$ $\vdash^{\mathbf{rdfs3}}_{\text{RDF}}$ $\mathtt{o}$ rdf:type $\mathtt{c}$

Fig. 2. Sample immediate entailment rules.

RDF entailment. *Implicit triples*, are considered part of the RDF graph even though they are not explicitly present in it, e.g., the triple Anne rdf:type Person above. *RDF entailment* is mechanism through which, based on a set of explicit triples and some *entailment rules*, implicit RDF triples are derived. We denote by $\vdash^{i}_{\text{RDF}}$ *immediate entailment*, i.e., the process of deriving new triples through a single application of an entailment rule. More generally, a triple $\mathtt{s}\ \mathtt{p}\ \mathtt{o}$ is entailed by a graph $\mathtt{G}$, denoted $\mathtt{G} \vdash_{\text{RDF}} \mathtt{s}\ \mathtt{p}\ \mathtt{o}$, if and only if there is a sequence of applications of immediate entailment rules that leads from $\mathtt{G}$ to $\mathtt{s}\ \mathtt{p}\ \mathtt{o}$ (where at each step of the entailment sequence, the triples previously entailed are also taken into account). Figure 2 shows some immediate entailment rules used to derive facts from some RDFS constraints.

Graph saturation. The immediate entailment rules allow defining the finite *saturation* (a.k.a. closure) of an RDF graph $\mathtt{G}$, which is the RDF graph $\mathtt{G}^{\infty}$ defined as the fix-point obtained by repeatedly applying $\vdash^{i}_{\text{RDF}}$ on $\mathtt{G}$.

The saturation of an RDF graph is unique (up to blank node renaming), and does not contain implicit triples (they have all been made explicit by saturation). An obvious connection holds between the triples entailed by a graph $\mathtt{G}$ and its saturation: $\mathtt{G} \vdash_{\text{RDF}} \mathtt{s}\ \mathtt{p}\ \mathtt{o}$ if and only if $\mathtt{s}\ \mathtt{p}\ \mathtt{o} \in \mathtt{G}^{\infty}$.

RDF entailment is part of the RDF standard itself; in particular, *the answers of a query posed on* $\mathtt{G}$ *must take into account all triples in* $\mathtt{G}^{\infty}$, since *the semantics of an RDF graph is its saturation*. In Sesame [7], Jena [8], OWLIM [9] etc., RDF entailment is supported through *saturation*.

RDF querying through SPARQL. The W3C standard for querying RDF is SPARQL Protocol and RDF Query Language (SPARQL) [10]. We consider the well-known subset of SPARQL consisting of *basic graph pattern* (BGP) queries, also known as SPARQL conjunctive queries. A BGP is a set of *triple patterns*, or triples in short. In a query triple, subjects and properties can be URIs, blank nodes or variables; objects can also be literals.

Query answering. The evaluation of a query q against an RDF graph $\mathtt{G}$ only uses $\mathtt{G}$'s explicit triples, thus leads to an incomplete answer set in the general case.

The (complete) *answer set* of q against $\mathtt{G}$ is defined as the evaluation of q against $\mathtt{G}^\infty$, denoted by $q(\mathtt{G}^\infty)$.

B. Query answering techniques

The literature provides two main techniques reflecting in RDF query answers the entailed triples from the input graphs. The core of our tutorial is devoted to these techniques in the context of RDF databases, where the issues they raise are quite subtle due to interactions between the data model and query language expressive power.

Graph saturation. *Graph saturation* (or *closure*) consists of pre-computing (making explicit) and adding to an RDF graph all its implicit information, i.e., the entailed triples. Answering queries using saturation amounts to evaluating the queries against the saturated graph. While saturation leads to efficient query processing, it requires time to be computed, space to be stored, and must be recomputed upon updates. We present the main **RDF graph saturation** algorithms, some including implicit triple maintenance upon updates [9], [11], [12], [13].

Query reformulation. The alternative technique is *query reformulation*. This turns the query q into a *reformulated query* q^{ref}, which, evaluated against the original graph, yields the exact answers to the *original* query: $q^{ref}(\mathtt{G}) = q(\mathtt{G}^\infty)$. Since reformulation is made at query run-time, it is intrinsically robust to updates; reformulation is also typically very fast. However, reformulated queries are often syntactically more complex than the original, thus their evaluation may be costly.

RDF and SPARQL are complex languages with many features. For instance, the RDF specification supports a form of *incomplete information* through *blank nodes*, many entailment rules etc.; SPARQL 1.1 supports aggregates, negation etc. If saturation is used, one first chooses an RDF fragment and saturates the RDF graph accordingly. Then in a fully orthogonal way, one chooses the SPARQL dialect to evaluate on the saturated graph. In contrast, reformulation leads to a subtle interplay between the RDF and SPARQL dialects, since the query must be reformulated within the latter, so as to compute the query answers with respect to the former.

We detail the principles and classify the main algorithms for **RDF query reformulations** as follows.

Reformulation-based query answering has been investigated in (or can be transferred to) the Description Logic [14] fragment of RDF [15], [16], [17], and for a slight extension thereof blurring the distinction between constants and classes/properties (i.e., relations) [18], [19], [12], [20], [21]. These RDF fragments impose restrictions on triples (e.g., no blank nodes) and on entailment (e.g., only the RDFS entailment rules are considered). In the tutorial, we classify existing algorithms from the viewpoint of both the expressive power of the RDF fragment and of the query language.

Reformulation-based query answering in the DL fragment of RDF, for relational conjunctive queries, has been investigated in (can be transferred from) [15], [16], [17], while a slight extension thereof considered in [18], [19], [12], [20], [21] has been investigated for one-triple BGP queries [20], [21], BGP queries [19], [12], and SPARQL [18].

Performance perspective. If the RDF graph never changes, RDF saturation is clearly preferrable, since on the saturated graph $\mathtt{G}^\infty$, efficient query evaluation techniques can be directly used to compute query answers. At the other extreme, on a frequently changing graph, saturation maintenance costs may be prohibitive, and thus reformulation is the only choice.

From a practical perspective, the most appropriate technique to a given setting should be chosen with an eye on the performance. Figure 3 (borrowed from [12]) illustrates one way to *quantifying the relative interest of saturation and reformulation*; while the details can be found in [12], they are irrelevant here, where we just use it for illustration.

For a set of queries, Figure 3 shows several *thresholds*. Thus, the *saturation threshold* for a query q is: the minimum number of times n that q needs to be run, so that: the cost of saturating the graph (independent of q), plus the cost of evaluating n times $q(\mathtt{G}^\infty)$, is smaller than n times the cost of evaluating $q^{ref}(\mathtt{G})$. The larger the threshold, the "harder" it is to amortize saturation (which is a fixed, one-time cost). Similarly, the threshold of q for an instance (or schema) deletion (or insertion), is the minimum number of times one needs to run q so that the cost of *maintaining the saturation* $\mathtt{G}^\infty$ after an instance (or schema) insertion (resp. deletion) is smaller than the cost of running n times $q^{ref}(\mathtt{G})$.

As Figure 3 shows, the threshold vary very significantly, even on the same database (up to 7 orders of magnitude)! This demonstrates that (i) saturation is not always the best solution, e.g., in some cases it takes more than 10 million runs to amortize its cost; and (ii) a finer-grained analysis of the performance trade-offs involved is needed to make an informed choice between the two.

C. Entailment mechanisms supported in RDF systems

Many well-known RDF platforms, e.g., Jena [8], OWLIM [9], Sesame [7], Virtuoso [22], or research prototypes, e.g., RDF-3X [23], [24] either (i) ignore entailed triples or (ii) provide saturation-based query answering, based on some subsets of RDF entailment rules. We outline their reasoning capabilities and limitations, in particular:

AllegroGraph's [25] *RDFS++* performs run-time reasoning, sometimes incomplete, based on backward chaining. It supports all the RDFS predicates and some of OWL's. It is not complete, but it has predictable and fast performance.

Virtuoso [22] SPARQL uses a backward chaining implementation for inferring triples that are not physically stored, meaning queries return the complete answer set without having all the implied triples materialized. Its reasoning supports some of the RDFS and OWL predicates.

OWLIM [9] relies on a forward-chaining approach for materializing all implicit information before query processing. It then employs both inferencing techniques to compute only the relevant justifications w.r.t. an update, at maintenance time.

Oracle Spatial and Graph's [26] *RDF Semantic Graph* features provide persistent inferencing based on forward-chaining, that supports RDFS, OWL 2, and user-defined rules.

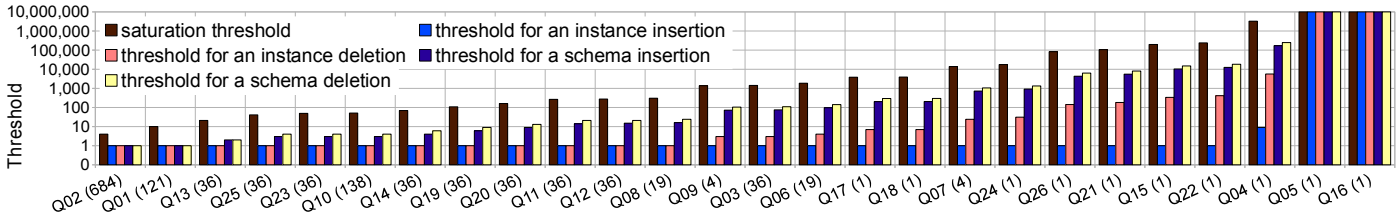

Fig. 3. Saturation thresholds: quantifying the amortization of saturation.

While the system does not currently support query reformulation, Oracle has considered the topic in [27].

D. Open issues

We find that the main currently open problems are related to performance: efficiently maintaining RDF graph saturation, especially in a distributed setting; efficiently evaluating large, complex reformulated RDF queries; and automatizing to the extent possible the choice between these two techniques, based on a quantitative evaluation of the application setting. As memory sizes grow larger, in-memory RDF reasoning is also attracting interest [28]. Finally, alternative methods for answering queries against an RDF graph can be devised, for instance based on translation to Datalog; given the presence of new-generation, very efficient Datalog engines [29], smart translations to Datalog and possibly RDF-specific Datalog optimization techniques are of interest.

III. Presenters

Damian Bursztyn started a PhD in January 2014 at Université Paris-Sud and INRIA. Damian's main research topic concerns models and algorithms for optimizing cloud-based Semantic Web data management. Personal webpage: http://pages.saclay.inria.fr/damian.bursztyn/

François Goasdoué received his PhD from Université Paris-Sud in 2001. He was a post-doc at Orange Labs Lannion, France, then he joined Université Paris-Sud in 2003 as an Associate Professor and is Full Professor at Université Rennes 1 since 2013. His main research interest is automated reasoning for data and knowledge base management, which includes answering queries under constraints, query rewriting using views etc. Personal webpage: http://people.irisa.fr/Francois.Goasdoue/

Ioana Manolescu received her PhD from INRIA and Université de Versailles Saint-Quentin in 2001. Ioana was a post-doc at Politecnico di Milano, Italy, then she joined INRIA where she is now senior researcher and the head of the OAK team, specialized in database optimization techniques for complex, large data. Her research interests include algebraic optimizations, adaptive storage and efficient management of semantically rich data. Personal webpage: http://pages.saclay.inria.fr/ioana.manolescu/

Alexandra Roatiş received her PhD from Université Paris-Sud and INRIA in September 2014. She obtained her Master's degree at the West University of Timişoara in Romania. Alexandra's main research topic concerns efficient and effective models and platforms for Semantic Web data management. Personal webpage: https://www.lri.fr/~roatis/

References

[1] Z. Kaoudi and I. Manolescu, "RDF in the Clouds: A Survey," The VLDB Journal, 2014.

[2] ——, "Cloud-based management of RDF data (tutorial)," SIGMOD, 2014.

[3] S. Abiteboul, I. Manolescu, P. Rigaux, M.-C. Rousset, and P. Senellart, *Web Data Management and Distribution*. Cambridge University Press, Dec 2011.

[4] "Resource Description Framework," http://www.w3.org/RDF.

[5] "RDF Schema," http://www.w3.org/TR/rdf-schema/.

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

[7] "Sesame," http://www.openrdf.org.

[8] "Jena," http://jena.sourceforge.net.

[9] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov, "OWLIM: A family of scalable semantic repositories," *Semantic Web*, vol. 2, no. 1, 2011.

[10] W3C, "SPARQL," http://www.w3.org/TR/rdf-sparql-query.

[11] J. Broekstra and A. Kampman, "Inferencing and Truth Maintenance in RDF Schema: Exploring a naive practical approach," in *PSSS Workshop*, 2003.

[12] F. Goasdoué, I. Manolescu, and A. Roatis, "Efficient Query Answering against Dynamic RDF Databases," in *EDBT*, 2013.

[13] C. Gutierrez, C. A. Hurtado, and A. A. Vaisman, "RDFS Update: From Theory to Practice," in *ESWC*, 2011.

[14] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[15] P. Adjiman, F. Goasdoué, and M.-C. Rousset, "SomeRDFS in the Semantic Web," *JODS*, 2007.

[16] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family," *J. of Automated Reasoning (JAR)*, 2007.

[17] G. Gottlob, G. Orsi, and A. Pieris, "Ontological Queries: Rewriting and Optimization," in *ICDE*, 2011, keynote.

[18] M. Arenas, C. Gutierrez, and J. Pérez, "Foundations of RDF Databases," in *Reasoning Web*, 2009.

[19] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu, "View Selection in Semantic Web Databases," *PVLDB*, 2011.

[20] Z. Kaoudi, I. Miliaraki, and M. Koubarakis, "RDFS Reasoning and Query Answering on DHTs," in *ISWC*, 2008.

[21] J. Urbani, F. van Harmelen, S. Schlobach, and H. Bal, "QueryPIE: Backward Reasoning for OWL Horst over Very Large Knowledge Bases," in *ISWC*, 2011.

[22] O. Erling, "Virtuoso, a Hybrid RDBMS/Graph Column Store," *IEEE Data Eng. Bull.*, 2012.

[23] T. Neumann and G. Weikum, "x-RDF-3X: Fast Querying, High Update Rates, and Consistency for RDF Databases," *PVLDB*, 2010.

[24] C. Weiss, P. Karras, and A. Bernstein, "Hexastore: Sextuple Indexing for Semantic Web Data Management," *PVLDB*, 2008.

[25] "AllegroGraph," http://franz.com/agraph/allegrograph/.

[26] "Oracle Spatial and Graph white papers," http://www.oracle.com/technetwork/database/ options/spatialandgraph/.

[27] Z. Wu, K. Rieb, G. Eadon, A. Khandelwal, and V. Kolovski, "Advancing the Enterprise-class OWL Inference Engine in Oracle Database," in *ORE*, 2012.

[28] J. D. Fernández, C. Gutierrez, M. A. Martínez-Prieto, and J. Pérez, "Towards In-Memory RDFS Entailment." in *AMW*, 2014.

[29] B. Motik, Y. Nenov, R. Piro, I. Horrocks, and D. Olteanu, "Parallel materialisation of datalog programs in centralised, main-memory RDF systems," in *AAAI*, 2014.