INFSYS
RESEARCH
REPORT

INSTITUT FÜR INFORMATIONSSYSTEME
ABTEILUNG WISSENSBASIERTE SYSTEME

EFFICIENTLY QUERYING RDF(S) ONTOLOGIES WITH ANSWER SET PROGRAMMING

Giovambattista Ianni Alessandra Martello
Claudio Panetta Giorgio Terracina

INFSYS RESEARCH REPORT RR-1843-08-06
AUGUST 2ND, 2008

Institut für Informationssysteme
Abtg. Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TU
TECHNISCHE UNIVERSITÄT WIEN

INFSYS Research Report
INFSYS Research Report RR-1843-08-06, August 2nd, 2008

Efficiently querying RDF(S) ontologies with Answer Set Programming

Giovambattista Ianni[1,2] Alessandra Martello[2]
Claudio Panetta[2] Giorgio Terracina[2]

Abstract. Ontologies are pervading many areas of knowledge representation and management. To date, most research efforts have been spent on the development of sufficiently expressive languages for the representation and querying of ontologies; however, querying efficiency has received attention only recently, especially for ontologies referring to large amounts of data. In fact, it is still uncertain how reasoning tasks will scale when applied on massive amounts of data. This work is a first step toward this setting: it first shows that RDF(S) ontologies can be expressed, without loss of semantics, into Answer Set Programming (ASP). Then, based on a previous result showing that the SPARQL query language (a candidate W3C recommendation for RDF(S) ontologies) can be mapped to a rule-based language, it shows that efficient querying of big ontologies can be accomplished with a database oriented extension of the well known ASP system DLV, which we recently developed. Results reported in the paper show that our proposed framework is promising for the improvement of both scalability and expressiveness of available RDF(S) storage and query systems.

---

[1]Institut für Informationssysteme 184/3, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria.
[2]Dipartimento di Matematica, Università della Calabria, I-87036 Rende (CS), Italy.

Acknowledgements: The work of the first author has been supported under FWF projects P20841 (Modular HEX-Programs), P17212 (Answer Set Programming for the Semantic Web), and MIUR (Italian Research Ministry) Interlink II04CG8AGG. We gratefully acknowledge Axel Polleres (DERI-Galway) for his insightful comments and for providing a prototypical version of the SPARQL-Datalog translator.

To appear on Journal of Logic and Computation, Oxford University Press, Oxford, UK. doi:10.1093/logcom/exn043

1 Introduction

The *Semantic Web* [6, 16] is an extension of the current Web by standards and technologies that helps machines understand the information on the Web. In this context, machines should be enabled to support richer discovery, data integration, navigation, and automation of tasks. Roughly speaking, the main ideas underlying the Semantic Web are oriented to *(i)* add a machine-readable meaning to Web pages and Web resources (annotations), *(ii)* use ontologies for a precise definition of shared terms in Web resources, *(iii)* make use of Knowledge Representation and Reasoning technology for automated reasoning on Web resources, and *(iv)* apply cooperative agent technology for processing the information on the Web. The development of the Semantic Web proceeds in layers of Web technologies and standards, where every layer is built on top of lower layers.

Research work is continuously ongoing on the three consecutive RDF(S) (Resource Description Framework (Schema)), Ontology and Rule layers (listed from bottom to top). The RDF(S) layer was initially conceived as a basic framework for defining resources available on the Web and their connections. RDF (Rich Description Framework), refers to a logical format of information, which is based on an encoding of data as a labeled graph (or equivalently, a ternary relation, commonly called RDF *graph*). RDF data can be interpreted under plain RDF semantics or under RDFS semantics[1]. In the aforementioned layered vision, RDF(S) should have little or no semantics, focusing only on the logical format of information.

The Ontology layer should be built on top of RDF(S) and should provide the necessary infrastructure for describing knowledge about resources. An ontology can be written using one of the three official variants of the Web Ontology Language (hereafter OWL) [25], currently accepted as a W3C Standard Recommendation. An OWL knowledge base is written in RDF format, where some of the keywords of the language are enriched with additional meaning.

Having machine readable annotations coupled with Web resources enables a variety of applications. As a traditional example, consider flight, train, coach and metro carriers, and hotels offering their services on the Web. They might, for instance, export timetables and room availabilities in standard RDF/OWL format. An automated Web service could thus easily compose data coming from different carriers and hotels, and after performing necessary reasoning tasks (for instance, aligning arrival/departure times and room availabilities, combining "train trips" with "flight trips" once it is inferred they are specialized subclasses of "trips"), it can propose overall itinerary solutions to end users. While services offering partial solutions to this task (and similar ones) exist, it is currently hard to achieve this goal in its entirety, due to the lack of structure in Web information.

OWL is based on description logics [5]. Description logics has a long tradition as a family of formalisms for describing concept terminologies and can feature rich expressiveness, like some prominent description logics, such as $\mathcal{SHOIN}(\mathbf{D})$ (which is the theoretical basis of the variant OWL-DL of OWL). The payload of this expressiveness is, unfortunately, the high computational cost associated with many of the reasoning tasks commonly performed over

[1] Throughout the paper we refer to RDF(S) for denoting the whole framework, no matter of the chosen semantics. Whenever necessary, we explicitly refer to RDF and/or RDFS.

an ontology. Nonetheless, a variety of Web applications require highly scalable processing of data, more than expressiveness. This puts the focus back to the lower RDF(S) data layer. In this context, RDF(S) should play the role of a lightweight ontology language. In fact, RDF(S) has few and simple descriptive capabilities (mainly, the possibility of describing and reasoning over monotonic taxonomies of objects and properties). One can thus expect from RDF(S) query systems the ability of querying very large datasets with excellent performance, yet allowing limited reasoning capabilities on the same data. In fact, as soon as the RDF(S) format for data was settled, and much earlier than when RDF(S) ontologies became growingly available[2], research has focused on how RDF(S) can be fruitfully stored, exchanged and queried[3].

As a candidate W3C recommendation [31], the SPARQL language is reaching consensus as query language of election for RDF(S) data. In this scenario, an RDF(S) storage facility (commonly called *triplestore*) plays the role of a database. However, an important difference with respect to traditional relational databases, is that a triplestore (also) represents information not explicitly stored, and which can be obtained by logical inference. Allowed logical inference is specified in terms of *entailment rules*. Different kinds of entailment rules can be exploited, namely *normative* (i.e. coming from the RDF(S) semantics specifications [29]), some subset of the normative ones (such as the so-called ρDF fragment of RDF(S) introduced in [28]) or user defined entailment rules.

Figure 1(a) shows the generic architecture of most current triplestore querying systems. In particular, the triplestore acts as a database and the query engine (possibly a SPARQL-enabled one[4]) manipulates this data. Most of the current query engines (with the notable exception of ARQ [2]) adopt a *pre-materialization* approach, where entailment rules are pre-computed and the initial triplestore is enriched with their consequences before carrying out any querying activity.

Unfortunately, triplestores based on the pre-materialization approach outlined above have some drawbacks:

• Inferred information is available only after the often long lasting pre-materialization step. Pre-materialization is unpractical if massive amounts of data are involved in the inferencing process; in fact, inferred information is usually much bigger in size than the original one. As an example, if only the ρDF fragment of RDFS is considered, this growth has been empirically estimated (in, e.g., [32]) as more than twice the original size of the dataset. Actually, a materialized dataset can be cubically larger in theory [27]. If the full normative RDF(S) semantics is considered, then the set of inferred triples is infinite and cannot be materialized at all.

• Entailment rules are "statically" programmed by coupling a parametric reasoner (designed "ad-hoc") with the original triplestore code. This prevents the possibility to dynamically

[2]Several datasets are nowadays available in RDF format, such as DBLP [12] and Wikipedia [4]. Also, it is possible to convert legacy data into RDF by means of ad-hoc services [7].

[3]The reader may find in [30] a good starting point to the vast applicative and academic research currently under development for RDF.

[4]E.g. [2, 1] and [33]. The reader may refer to [34] for a thorough survey.

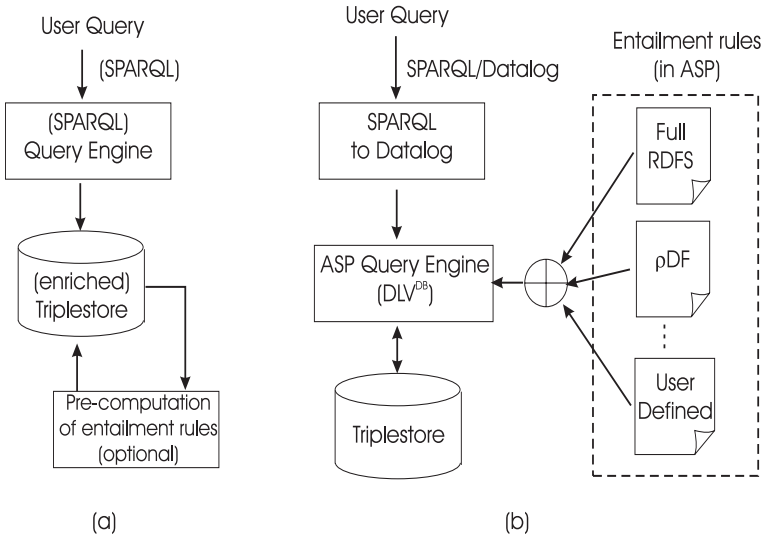

Figure 1: *(a)* State-of-the-art architectures for querying RDF(S) triplestores. *(b)* Our proposal.

prototype new inference rules, and to activate/de-activate inference rules depending on the given application. For instance, one might want to restrict RDF(S) inference only to the known ρDF fragment, or to enlarge inference with other (normative or not) entailment rules such as the D-entailment rules introduced in [29].

• The basic reasoning machinery of RDF(S) prescribes a heavy usage of transitive closure (recursive) constructs. Roughly speaking, given a class taxonomy, an individual belonging to a leaf class must be inferred to be member of all the ancestor classes, up to the root class. This prevents a straightforward implementation of RDF(S) over RDBMSs, since RDBMSs usually feature very primitive, and inefficient, implementations of recursion in their native query languages.

In this paper, we propose and experiment with a new architecture for querying triplestores, based on Answer Set Programming (ASP), which overcomes all the drawbacks outlined above and improves both efficiency and expressiveness of current state-of-the-art systems.

The proposed architecture is shown in Figure 1(b). The user is allowed to express queries in both SPARQL and Datalog[5]. A SPARQL query is first translated into Datalog by a corresponding module, which implements the approach proposed in [26], whereas a Datalog query is passed unchanged to the next module. Entailment rules are expected to be expressed in ASP, and different sets of entailment rules can be "attached" to the triplestore. As an example, in Figure 1(b) three sets are shown, namely full normative RDFS, ρDF and user-defined entailment rules. The user can choose on the fly, at query time, which set

[5]As it will be clear in the following, Datalog is more expressive than SPARQL.

of entailment rules must be applied on the triplestore, and/or design his own. Note that, rather than materializing the whole output of the application of entailment rules, the rules are used as an inference mechanism for properly answering the specific input query at hand. The query engine is then implemented by means of a database oriented version of an ASP evaluator, which can both access and modify data in the triplestore.

In order to let ASP provide the abovementioned benefits, theoretical and practical issues must be solved:

1. A faithful translation of the whole normative RDF(S) into ASP has not been attempted yet, due to some important semantic differences between the two languages. As an example, RDF(S) has the capability to deal with unnamed individuals, using so called *blank nodes*. A blank node can be seen as a limited form of existential quantification. ASP semantics is usually derived from function-free Datalog, and has no direct possibility to deal with unnamed individuals (objects whose existence is known but whose identity cannot be reconduced to a known constant symbol) in a context where unique name assumption is not assumed. One of the main contributions of this paper is the translation of RDF(S) (especially of its entailment rules) into a suitable extension of ASP we recently proposed, and the proof that this translation is faithful.

2. Another key issue in the proposed architecture is whether it could be possible to achieve an efficient interaction of the ASP engine with the database. In fact, it is well known that current (extended) Datalog-based systems present important limitations when the amount of data to reason about is large: *(i)* reasoning is generally carried out in main-memory and, hence, the quantity of data that can be handled simultaneously is limited; *(ii)* the interaction with external (and independent) DBMSs is not trivial and, in several cases, not allowed at all, but in order to effectively share and elaborate large ontologies these must be handled with some database technology; *(iii)* the efficiency of present datalog evaluators is still not sufficient for their utilization in complex reasoning tasks involving large amounts of data.

As far as the second issue is concerned, we recently proposed a new database-oriented extension of the well known Answer Set Programming system DLV, named DLV^{DB} [35], which presents the features of a Deductive Database System (DDS) and can do all the reasoning tasks directly in mass-memory; DLV^{DB} does not have, in principle, any practical limitation in the dimension of input data, is capable of exploiting optimization techniques both from the DBMS field (e.g. join ordering techniques [17]) and from the DDS theory (e.g. magic sets [24]), and can easily interact (via ODBC) with external DBMSs. DLV^{DB} turned out to be particularly effective for reasoning about massive data sets (see benchmark results presented in [35]) and supports a rich query and reasoning language including stratified recursion, true negation, negation as failure, and all built-in and aggregate functions already introduced in DLV [15].

Summarizing, the contributions of the paper are:

• A faithful translation of RDF(S) entailment rules into Answer Set Programming extended with external predicates (ASP^{EX}) is developed. Proofs of equivalence and computability of

the translation are also given. The translation can be fully implemented in an Answer Set Programming system.

• RDF(S) semantics can now be taken into account without the necessity of pre-materializing inferrable data.

• The set of entailment rules to be adopted for the inference can be dynamically chosen at query time.

• Recursive constructs (both for querying and inferencing) are natively supported by ASP and, thus, directly available to both the internal engine and the end user wishing to design sophisticated queries (if Datalog is exploited as query language).

• Management of massive amounts of data is effectively handled by the mass-memory based evaluation of our DLV^{DB} query engine.

• The proposed architecture has been implemented and tested to compare both expressiveness and efficiency of our ASP-based approach with state-of-the-art triplestore querying systems, giving very encouraging results.

Related Work. To the best of our knowledge, this is the first attempt to provide a complete translation of all the normative RDF(S) semantics into declarative logic programming. The work is similar in spirit to [8], where it is shown how RDF, RDFS and ERDFS can be ported to F-Logic (and, to a large extent, to Datalog), and thus implemented in a standard reasoner. The above work does not address explicitly the problem of treating the infinite set of axiomatic triples of RDFS in a finite context. Also, in [28] a deductive calculus is presented which is sound and complete for the ρDF fragment of the language, and paves the way to its implementation in a deductive system.

It is worth pointing out also that current important efforts in the Semantic Web community aim at integrating Ontologies with Rules under stable model semantics (e.g. [13, 14, 23]). In this context, the abovementioned works highlight that the possibility of exploiting a Datalog-like language to express (or integrate) ontologies with a query/rule language provides important benefits. A work explicitly addressing RDF and proposing the idea of extending RDF graphs with negation and stable models is [3].

The paper is organized as follows. In the next Section we briefly introduce some background knowledge. Then, in Section 3, we formally introduce the mapping between RDFS and ASP^{EX}. In Section 4 we provide some implementation details of the proposed architecture and discuss experimental results. Finally, in Section 5 we draw some conclusions.

2 Background

In this section we provide some background knowledge needed to introduce our approach. In particular, we first recall basic notions on Answer Set Programming with extensions to support external function calls, then we briefly introduce RDF(S).

2.1 Answer Set Programming with External Predicates

As reference language for the internal querying engine, we adopt Answer Set Programming extended with external predicates (hereafter ASP^{EX}). Informally, an external predicate models knowledge that is external to a given logic program, and whose extension might be infinite. The usage of external predicates is crucial for modeling some aspects of the normative RDF(S) that require to deal with infinite sets and/or that are difficult to be encoded using plain ASP. The reader is referred to [10] for a complete reference on Answer Set Programming extended with external predicates. An ASP^{EX} program P is a finite set of rules of the form

$$\alpha_1 \vee \cdots \vee \alpha_k :\!- \beta_1, \ldots, \beta_n, \text{ not } \beta_{n+1}, \ldots, \text{ not } \beta_m. \tag{1}$$

where $m, k \geq 0$, $\alpha_1, \ldots, \alpha_k$, are ordinary atoms, and $\beta_1, \ldots, \beta_m$ are (ordinary or external) atoms. An external atom predicate name is conventionally preceded by "#". Rules with $k = 0$ and $m > 0$ are called *constraints*, whereas rules such that $k = 1$ and $m = 0$ are called *facts*.

Atoms and external atoms have the usual structure $p(t_1, \ldots, t_n)$ where p is the predicate name and $t_1, \ldots, t_n$ are either variables or constants ranging over a (possibly) infinite set $\mathcal{C}$. Elements of $\mathcal{C}$ are supposed to be strings, i.e. they are a subset of Σ^* for a given finite vocabulary Σ.

The *Herbrand base* $HB_{\mathcal{C}}(P)$ is the possibly infinite set of ground atoms, whose predicate name appears in P, that can be constructed using symbols appearing in $\mathcal{C}$. An *interpretation* I for P is a pair $\langle S, F \rangle$ where:

• $S \subseteq HB_{\mathcal{C}}(P)$ is a set of ordinary atoms; we say that I (or by small abuse of notation, S) is a *model* of an ordinary atom $a \in HB_{\mathcal{C}}(P)$ if $a \in S$.

• F is a mapping which associates every external predicate name $\#e$ with a decidable n-ary function (which we call *oracle*) $F(\#e)$; it assigns each tuple $(x_1, \ldots, x_n)$ to either 0 or 1, where n is the fixed arity of $\#e$, and $x_i \in \mathcal{C}$. I (or, by small abuse of notation, F) is a *model* of a ground external atom $a = \#e(x_1, \ldots, x_n)$ if $F(\#e)(x_1, \ldots, x_n) = 1$.

An interpretation I is an *answer set* for a program P if it coincides with a minimal model of P^I, where P^I is the traditional Gelfond-Lifshitz reduct of $grnd(P)$. $grnd(P)$ is the ground version of P constructed using constant symbols from $\mathcal{C}$ [18]. Let $ans(P)$ be the set of answer sets of P. Given a ground atom a, we say that $P \models a$ (P cautiously models a), if for each $A \in ans(P)$, then $a \in A$.

An example of external atom could be $\#concat(X, Y, Z)$, which has an oracle such that this atom is true whenever Z is the concatenation of the strings X and Y.

Note that, given that $\mathcal{C}$ is not assumed to be finite, in principle $HB_{\mathcal{C}}(P)$, $grnd(P)$, as well as interpretations and answer sets are not finite.

However a large class of ASP^{EX} programs is proven to have finite answer sets, namely vi-restricted (*value invention-restricted*) programs, whose definition is given in [10]. Intuitively, external predicates may be associated with functions having an unknown (and possibly

infinite) co-domain. A vi-restricted program is such that new values brought in the logic program by means of external predicates do not propagate through recursion; this avoids the existence of answer sets of infinite size.

2.2 RDF and RDFS

RDF (without S), has per se a very basic semantics, which has been extended to RDFS. The syntactic format of RDF and RDFS is equivalent, although the latter gives semantics to some special keywords. In this paper we will use the notation RDF(S) for generally referring to RDF with or without its extended semantics called RDFS (short for RDF-Schema). The semantics of RDF(S) is outlined below, following the normative specification given in [29]. An RDF *tripleset* (or *graph*) is a set of triples $(s, p, o) \in (I \cup B \cup L) \times I \times (I \cup B \cup L)$ where I, B and L are pairwise disjoint sets of IRIs (Internationalized Resource Identifiers), blank nodes, and literals, respectively[6]. s is called the *subject*, p the *predicate* and o the *object* of the triple, respectively. As commonly done in the literature (see, e.g. [28]), we enlarge our focus to graphs where literals are allowed to appear also in the subject position within a triple. We occasionally denote a blank node $b \in B$ as starting with prefix "_", such as $\_b$.

The meaning of RDF(S) graphs is given in terms of first order interpretations without the unique name assumption. For instance, the triple ($\_b$,*hasName*,*GB*), can be seen as the first order sentence $\exists B\ hasName(B, GB)$, that is, "there exists an object in the domain of discourse having name *GB*". The first order semantics of RDF(S) is difficult to be implemented in practice without a concrete proof theory. Thus, the official specification of RDF(S) states equivalence theorems between the first order semantics and the notion of *graph entailment*. In the following we directly define RDF(S) semantics in terms of graph entailment, referring to the two notions of *simple* entailment and *RDFS* entailment[7].

Intuitively, graph entailment is built on the notion of subgraph isomorphism, that is, it amounts to finding a function mapping a graph to another. This mapping function must have the following characteristics.

A *mapping* μ is a function mapping elements from $I \cup B \cup L$ to elements in the same set, subject to the restriction that an element $i \in I \cup L$ must be such that $\mu(i) = i$. We indicate as $\mu(G)$ the set of triples $\{(\mu(s), \mu(p), \mu(o)) \mid (s, p, o) \in G\}$.

Note that elements of B (blank nodes), are used to model existential quantification and can be mapped to any element, while elements of I and L preserve their identity through the mappings.

Given two graphs G_1 and G_2, we say that $G_1 \models G_2$ (G_1 *simply entails* G_2) if there is a mapping μ such that $\mu(G_2)$ is a subgraph of G_1. In general, deciding this kind of entailment is NP-complete [29] with respect to the combined sizes of G_1 and G_2, while it has been observed that entailment is polynomial in the size of G_1 [9].

The above notion is used as a tool for formalizing query languages for RDF, as follows. One can see G_1 as the data to be queried, and G_2 as a query pattern that should be matched

[6] In practice, I, B and L are strings of a given vocabulary and are subject to syntactic rules.

[7] [29] includes also the notion of RDF-entailment and D-entailment.

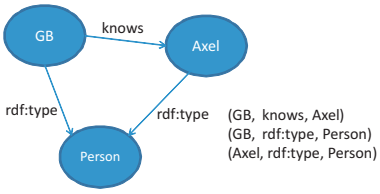

Figure 2: The example graph G'_1 and its corresponding set of triples.

to G_1. Rules establishing matching criteria can be those of simple entailment (which basically consist in checking subgraph isomorphism between G_1 and G_2) or more complex ones.

For instance, the graph $G'_2 = \{(GB, knows, \_b)\}$, where $\_b$ is a blank node, is entailed by the graph in Figure 2 (a matching mapping is obtained by associating $\_b$ to *Axel*). Notably, blank nodes in G'_2 are seen as variables to be matched, not differently from variables in an SQL-like query language as actually SPARQL is. Note that query answering and entailment are strictly related: the set of matching mappings constitutes the possible answers to the pattern query, while entailment is decided by the existence of at least one of such mappings.

For instance, the possible answers to the pattern graph $G''_2 = (\_b1, rdf{:}type, \_b2)$ with respect to the graph G'_1 in Figure 2, are the two mappings μ' and μ'' where $\mu'(\_b1) = GB$, $\mu'(\_b2) = Person$, and $\mu''(\_b1) = Axel$, $\mu''(\_b2) = Person$. Both μ' and μ'' testify that G'_1 entails G''_2.

Simple entailment is not directly used as a notion for defining semantics for RDF graphs. In fact, the two notions of RDF-entailment and RDFS-entailment, are originally defined by means of first order logic. However, the RDFS-entailment Lemma (sec. 7.3 of [29]) brings back RDFS-entailment of two graphs G_1 and G_2 to simple entailment between a graph $R(G_1)$ and G_2. The graph $R(G_1)$ is the *closure* of G_1, which is built *(i)* by applying, until saturation, so called RDF and RDFS entailment rules, and *(ii)* by adding normative axiomatic triples to G_1. RDF and RDFS entailment rules are reported in Figure 4 (the reader is referred to [29], sec. 7.3, for all details). Axiomatic triples are composed of a small finite set (which is tabulated in [29]) and an infinite portion, as we explicitly show in Figure 3.

$\forall i \in \mathbb{N}$,
(rdf:_i, rdf:type, rdfs:ContainerMembershipProperty),
(rdf:_i, rdfs:domain, rdfs:Resource),
(rdf:_i, rdfs:range, rdfs:Resource)

Figure 3: The infinite portion of RDFS axiomatic triples A.

We can classify normative RDFS semantics as follows:

• Taxonomy rules. Such rules regard the keywords *rdfs:subClassOf* and *rdfs:subPropertyOf*. Two separate taxonomies can be defined in an RDFS graph: a taxonomy of classes (a class c is a set of individuals C such that *(i, rdf:type, c)* holds for each $i \in C$), and a taxonomy of properties (a property p is a binary relation P, where each couple $(s, o) \in P$ is encoded by

the triple *(s,p,o)*). *rdfs:subClassOf* and *rdfs:subPropertyOf* are the special keywords to be used for defining such taxonomies. The semantic properties of these keywords are enforced by entailment rules such as RDFS5,7,9 and 11 (see Figure 4), which implement a simple (and monotonic) inheritance inference mechanism. For instance, if a graph G contains the triple *(person,rdfs:subClassOf,animal)* and the triple *(g,rdf:type,person)* then the closure $R(G)$ must contain *(g,rdf:type,animal)* as prescribed by entailment rule RDFS9.

• Typing rules. The second category of entailment rules strictly regard properties: the *rdfs:range* and *rdfs:domain* keywords allow the declaration of the class type of s and o for a given couple (s, o) when *(s,p,o)* holds (rules RDFS2 and 3 as well as IRDFS2 and 3), while rules RDF2A,2B, RDFS1A and 1B assess literal values[8].
For instance, from *(g,hasFather,c)* and *(hasFather,rdfs:domain,person)* it can be inferred *(c,rdf:type,person)*, by means of rule RDFS2. Note that the application of typing rules (and of the RDFS entailment rules in general) cannot lead to contradiction. This has two important consequences: first, range and domain specifications are not seen as integrity constraints, as it is usually assumed in the database field. Second, a triple graph cannot, usually, contain contradictory information. In fact, inconsistency is triggered only if a graph contains some ill-formed literal (or XMLLiteral) (e.g. a constant symbol $l \in L$ of type *rdfs:Literal* or *rdf:XMLLiteral*, which does not comply with syntactic prescriptions for this type of objects). In such a case, a graph G is assumed to be inconsistent (rules RDF2B and RDFS1B), and it is normatively prescribed that $R(G)$ must coincide with the set of all the possible triples.

• Axiomatic triples. These triples hold "a priori" in the closure of any graph, and give special behavior to some other keywords of the language. For instance, the triple *(rdfs:subClassOf, rdfs:domain, rdfs:Class)* enforces the domain of the property *rdfs:subClassOf* to be *rdfs:Class*. In particular, axiomatic triples contain an infinite subset (shown in Figure 3) which we call A. This set of triples regards special keywords used for denoting collections of objects (*Containers*) such as Bags (the *rdf:Bag* keyword) and Lists (the *rdf:Seq* keyword). For denoting the i-th element of a container, the property *rdf:_i* is used. These keywords have no special meaning associated besides the axiomatic group of triples *(rdf:_i, rdf:type, rdfs:cmp)*, *(rdf:_i, rdfs:domain, rdfs:Resource)*, *(rdf:_i, rdfs:range, rdfs:Resource)*, where *rdfs:cmp* represents a shortcut for *rdfs:ContainerMembershipProperty*. The set A enforces that the i-th element of a container is of type resource and that each *rdf:_i* is a Container Membership Property. A is of infinite size, and thus requires some care when introduced in an actual implementation of an RDF(S) query system.

Finally, we say that a graph G_1 *rdfs-entails* a graph G_2 ($G_1 \models_s G_2$) if the RDFS closure $R(G_1)$ of G_1 entails G_2. In practice, this means that if one wants to use G_2 for querying information about G_1 (which is expected to correspond to a large set of triples) under RDFS entailment, G_2 is actually matched to $R(G_1)$, which contains also inferred knowledge.

[8] IRDFS2 and 3 do not appear in the normative RDFS document, but were noted to be necessary for preserving the RDFS entailment Lemma validity in [22]. "I" stands for "integrative".

For instance, if we add the triple (*Person*,*rdf:subClassOf*,*Animal*) to G'_1, we have that, under RDFS semantics, the triple (*GB*,*rdf:type*,*Animal*), belongs to $R(G'_1)$, by rule RDFS9 of Figure 4. This means that, if entailment is interpreted under RDFS semantics, other mappings, such as μ''' such that $\mu'''(\_b1) = \textit{rdf:type}$ and $\mu'''(\_b2) = \textit{Animal}$, are a proof for entailment of G''_2 by G'_1.

In the following, we assume that $I \cup L \cup B \subseteq \mathcal{C}$. That is IRI, literals and blank nodes appearing in an RDF graph can be freely used as constant symbols in logic programs.

3 From RDF(S) to ASP^{EX}

In order to make the vision depicted in Figure 1(b) concrete, a crucial step has to be carried out. In fact, one might ask whether it is possible to express the full RDFS entailment semantics using a language based on stable models. In particular, there are some important problems to be faced:

• Usage of blank nodes. Usage of anonymous variables might, apparently, require the usage of a language with existential constructs. Although ASP is not, in general, equipped with this kind of knowledge representation tool, we show that blank nodes can be transformed into either anonymous constant symbols or universally quantified variables.

• Inconsistency. Most deductive databases, founded on stratified Datalog, do not allow modelling of contradiction as first order logic does. A stratified Datalog program has always a model (possibly empty), whereas RDF graphs might be inconsistent in a first order sense (i.e. following the *ex-falso quodlibet* principle). Conversely, this is not a problem under stable models semantics (rather than stratified Datalog), applied to non-stratified programs under brave or cautious reasoning. Indeed constraints (which are, under stable model semantics, particular, non-stratified, programs) allow modelling inconsistency in a similar way as first order logic does.

• Axiomatic triples. Although modelling the set A of infinite triples might raise practical concerns, we show how these problems can be circumvented by restricting "relevant" axiomatic triples to a finite subset.

We remark that, when an entailment and/or a query answer has to be computed, a graph can have two different roles: it can play the role of the dataset to be queried, or that of the pattern representing the query at hand.

In order to encode in ASP the problem of deciding RDFS entailment between two graphs G_1 and G_2, we adopt two translations T_1 and T_2, and an ASP^{EX} program D, which simulates RDFS entailment rules. T_1 transforms G_1 into a corresponding ASP^{EX} program; intuitively, it transforms each triple $t \in G_1$ into a corresponding fact. T_1 maps blanks nodes to themselves, as it is conceptually done in the Skolemization Lemma of [29]. This transformation is intended to be applied on graphs which encode a dataset subject to querying, that is, the current reference ontology. On the other hand, when a graph is seen as the description of a query pattern, we adopt the second transformation T_2.

Name	**If G contains:**	**then add to $R(G)$:**
RDF1	(u, a, y)	(a, rdf:type, rdf:Property)
RDF2A	(u, a, l) where l is a well-typed XML-literal	(l, rdf:type, rdf:XMLLiteral)
RDF2B	(u, a, l) where l is a ill-typed XML-literal	$(I \cup B \cup L) \times I \times (I \cup B \cup L)$
RDFS1A	(u, a, l) where l is a plain literal	(l, rdf:type, rdfs:Literal)
RDFS1B	(u, a, l) where l is a ill-typed literal	$(I \cup B \cup L) \times I \times (I \cup B \cup L)$
RDFS2	(a, rdfs:domain, x), (u, a, y)	(u, rdf:type, x)
RDFS3	(a, rdfs:range, x), (u, a, v)	(v, rdf:type, x)
IRDFS2	(a, rdfs:domain, b), (c,rdfs:subproperty,b), (x, c, y)	(x, rdf:type, b)
IRDFS3	(a, rdfs:range, b), (c,rdfs:subproperty,b), (x, c, y)	(y, rdf:type, b)
RDFS4A	(u, a, x)	(u, rdf:type, rdfs:Resource)
RDFS4B	(u, a, v)	(v, rdf:type, rdfs:Resource)
RDFS5	(u, rdfs:subPropertyOf, v), (v, rdfs:subPropertyOf, x)	(u, rdfs:subPropertyOf, x)
RDFS6	(u, rdf:type, rdf:Property)	(u, rdfs:subPropertyOf, u)
RDFS7	(a, rdfs:subPropertyOf, b), (u, a, y)	(u, b, y)
RDFS8	(u, rdf:type, rdfs:Class)	(u, rdfs:subClassOf, rdfs:Resource)
RDFS9	(u, rdfs:subClassOf, x), (v, rdf:type, u)	(v, rdf:type, x)
RDFS10	(u, rdf:type, rdfs:Class)	(u, rdfs:subClassOf, u)
RDFS11	(u, rdfs:subClassOf, v), (v, rdfs:subClassOf, x)	(u, rdfs:subClassOf, x)
RDFS12	(u, rdf:type, rdfs:ContainerMembershipProperty)	(u, rdfs:subPropertyOf, rdfs:member)
RDFS13	(u, rdf:type, rdfs:Datatype)	(u, rdfs:subClassOf, rdfs:Literal)

Figure 4: The RDF and RDFS Entailment rules for extending a graph G

RDF1'	t(A,*rdf:type*,*rdf:Property*)	:–	t(U,A,Y).
RDF2A'	t(L,*rdf:type*,*rdfs:XMLLiteral*)	:–	t(U,A,L), *#XMLLiteral*(L).
RDF2B'		:–	t(L,*rdf:type*,*rdfs:XMLLiteral*), not *#XMLLiteral*(L).
RDFS1A'	t(L, *rdf:type*, *rdfs:Literal*)	:–	t(U,A,L), *#Literal*(L).
RDFS1B'		:–	t(L, *rdf:type*, *rdfs:Literal*), not *#Literal*(L).
RDFS2'	t(U,*rdf:type*,X)	:–	t(A,*rdfs:domain*,X), t(U,A,V).
RDFS3'	t(V,*rdf:type*,X)	:–	t(A,*rdfs:range*,X), t(U,A,V).
IRDFS2'	t(X,*rdf:type*B)	:–	t(A,*rdfs:domain*,B), t(X,C,Y), t(C,*rdfs:subproperty*,B).
IRDFS3'	t(Y,*rdf:type*B)	:–	t(A,*rdfs:range*,B), t(X,C,Y), t(C,*rdfs:subproperty*,B).
RDFS4A'	t(U,*rdf:type*,*rdfs:Resource*)	:–	t(U,A,X).
RDFS4B'	t(V,*rdf:type*,*rdfs:Resource*)	:–	t(U,A,V).
RDFS5'	t(U,*rdfs:subPropertyOf*,X)	:–	t(U,*rdfs:subPropertyOf*,V), t(V,*rdfs:subPropertyOf*,X).
RDFS6'	t(U,*rdfs:subPropertyOf*,U)	:–	t(U,*rdf:type*,*rdf:Property*).
RDFS7'	t(U,B,Y)	:–	t(A,*rdfs:subPropertyOf*,B), t(U,A,Y).
RDFS8'	t(U,*rdfs:subClassOf*, *rdfs:Resource*)	:–	t(U,*rdf:type*, *rdfs:Class*).
RDFS9'	t(V,*rdf:type*,X)	:–	t(U,*rdfs:subClassOf*,X), t(V,*rdf:type*,U).
RDFS10'	t(U,*rdfs:subClassof*,U)	:–	t(U,*rdf:type*,*rdfs:Class*).
RDFS11'	t(U,*rdfs:subClassof*,X)	:–	t(U,*rdfs:subClassOf*,V), t(V,*rdfs:subClassOf*,X).
RDFS12'	t(U,*rdfs:subPropertyOf*,*rdfs:member*)	:–	t(U,*rdf:type*,*rdfs:cmp*).
RDFS13'	t(U,*rdfs:subClassof*,*rdfs:Literal*)	:–	t(U,*rdf:type*,*rdfs:Datatype*).
A'	*symb*(X)	:–	t(X,Y,Z).
	symb(Y)	:–	t(X,Y,Z).
	symb(Z)	:–	t(X,Y,Z).
	CMProperty(P)	:–	*symb*(P), *#concat*("rdf:_",N,P).
	t(X, *rdf:type*, *rdfs:cmp*)	:–	*CMProperty*(X).
	t(X, *rdfs:domain*, *rdfs:Resource*)	:–	*CMProperty*(X).
	t(X, *rdfs:range*,*rdfs:Resource*)	:–	*CMProperty*(X).

Figure 5: Translation D of entailment rules shown in Figure 4 in ASP^{EX}. *#concat* is defined as in Sec. 2, whereas *#Literal* and *#XMLLiteral* are defined as follows: $F_{\#Literal}(X) = 1$ (resp. $F_{\#XMLLiteral}(X) = 1$) iff X is compliant with the syntax allowed for literals (resp. XML Literal syntax) [29]. *rdfs:cmp* is a shortcut for *rdfs:ContainerMembershipProperty*.

T_2 transforms G_2 in a conjunction of atoms, whose truth values depend on whether G_2 is entailed by G_1 or not. T_2 maps blank nodes to variables.

Definition 1 *The translation $T_1(G)$ of an RDF graph G is the set of facts $\{t(s,p,o) \mid (s,p,o) \in G\}$. The translation $T_2(G)$ of an RDF graph G is a program containing:*

- *the rule $q =$ `entail` $\leftarrow H$. Here,* `entail` *is a fresh literal whereas H is the conjunction of all the atoms in the set $\{t(h(s),h(p),h(o)) \mid (s,p,o) \in G\}$, where h is a transformation function such that $h(a) = a$ if $a \in I \cup L$, $h(a) = A_a$ otherwise; A_a is a fresh variable associated with a if $a \in H$.*
- *the set of facts $\{symb(u) \mid u$ is an element of $I \cup L$ appearing in $G\}$.*

The program D, simulating RDFS entailment rules shown in Figure 4, is given in Figure 5. In particular, for each rule R in Figure 4 there is a counterpart R' in D (notably, RDF2B' and RDFS1B' are constraints rather than rules). The bottommost group of six rules in D (which we call A') models a finite portion of the infinite set of triples A shown in Figure 3. The set of facts of the form $symb(u)$ denotes all the elements explicitly appearing in G_1 or in G_2.

Note that A' derives only a finite portion of the axiomatic triples: the axiomatic triples regarding a given identifier *rdf:_i* are derived if $symb$(*rdf:_i*) is true. In such a case the atom *CMProperty*(rdf:_i) holds by means of the 4th rule of A', and this makes the head of the last but three rules of A' true.

In the following, let P be the ASP^{EX} program $P = D \cup T_1(G_1) \cup T_2(G_2) \cup K$, for two given RDF graphs G_1 and G_2, where K is the set of facts

$\{symb($*rdf:*$_i_u$$) \mid u$ is a blank node appearing in G_2 and i_u is a distinguished natural number such that neither G_1 nor G_2 talk about *rdf:*$\_i_u\}$.

The meaning of P in terms of its answer sets is given by the following Lemma. Roughly speaking the Lemma states that the unique answer set of P encodes in the extension of the predicate t a finite portion of the closure $R(G_1)$ of G_1. Namely, t contains only those triples of $R(G_1)$ that can be constructed using values belonging to S, where S is the set of constant symbols explicitly appearing in P. Note that S will contain all the symbols explicitly appearing in G_1 or G_2.

Lemma 1 *Let S be the set of constant symbols appearing in P. If P is consistent then its unique model M contains an atom $t(s,p,o)$ iff $(s,p,o) \in (S \times S \times S) \cap R(G_1)$.*

Proof. ($\Rightarrow$) If P is consistent, then its model M is clearly unique (P is a positive program except for the two constraints RDF2B' and RDFS1B'). We focus now on atoms of the form $t(s,p,o)$ contained in M. Let a be one of such atoms. Observe that, by construction of P, s, p and o belong to S. By observing the structure of P, we note that (s,p,o) corresponds to a triple of $R(G_1)$ for three possible reasons: either, *(i)* a corresponds to a fact in $T_1(G_1)$ (and thus $(s,p,o) \in G_1 \subseteq R(G_1)$), or *(ii)* a is the byproduct of the application of some of the rules

from RDF1' to RDFS13' in D, except RDF2B' and RDFS1B' (and thus $(s,p,o) \in R(G_1)$), or *(iii)* a corresponds to an axiomatic triple modelled by the subprogram A' of D.

($\Leftarrow$). Consider a triple $a = (s,p,o) \in (S \times S \times S) \cap R(G_1)$. Clearly, a is either *(i)* a triple belonging to G_1 (and thus it has a corresponding atom $t(s,p,o) \in T_1(G_1)$), or, *(ii)* it is the byproduct of the application of some RDFS entailment rules, which are in one-to-one correspondence with rules RDF1' to RDFS13' in D, or *(iii)* it is an axiomatic triple. In the latter case, the subprogram A' guarantees that $a \in (S \times S \times S) \cap R(G_1) \Rightarrow t(s,p,o) \in M$. □

The following trivial Lemma, used in the subsequent Theorem 1, shows that if graphs do not refer explicitly to IRIs of the form *rdf:_i* (which are symbols appearing in the infinite set of axiomatic triples), then such IRIs are "interchangeable" in the sense explained by the Lemma, yet preserving RDFS entailment.

Lemma 2 *(***Interchangeability Lemma***) Given an RDF graph G, we say that G* talks *about an element $e \in I \cup L$ if there exists a triple $t \in G$ containing e. Let G_1 and G_2 be two RDF graphs for which there is μ such that $\mu(G_2) \subseteq R(G_1)$ (i.e. $G_1 \models_s G_2$).*

For each element $e \in I \cup B$ such that $\mu(e) =$ rdf:_i ($i \in \mathbb{N}$), and such that G_1 and G_2 do not talk about e, then $\mu'(G_2) \subseteq R(G_1)$, for any μ' coinciding with μ except that $\mu'(e) =$ rdf:_i' where $i' \neq i$ and G_1 and G_2 do not talk about rdf:_i'.

Theorem 1 *Let G_1 and G_2 be two RDF graphs. Then $G_1 \models_s G_2$ iff $P \models$* `entail`.

Proof. ($\Leftarrow$). Assume that $P \models$ `entail`. Observe that P is a stratified program not containing disjunctive rules, thus it may entail `entail` in two cases: *(a)* P is inconsistent. P can have no model only if G_1 is inconsistent, i.e. G_1 contains some literal which is not syntactically valid. This makes the body of either constraint RDFS1B' or RDFS2B' true (see Figure 5). Note that an inconsistent graph normatively entails any other graph by definition. *(b)* P contains `entail` in its unique answer set M. This means that the body H of the rule $q =$ `entail` $\leftarrow H$ has some ground instantiation which is modelled by M. By Lemma 1, it follows that $R(G_1) \models G_2$.

($\Rightarrow$). Given that $G_1 \models_s G_2$, then, if G_1 is inconsistent, P has no answer sets, thus trivially entailing the literal `entail`. If G_1 is consistent, there is a mapping μ from G_2 to $R(G_1)$. It is possible to show that the single answer set M of P contains `entail`. For proving this, we need to show that $t(\mu(s),\mu(p),\mu(o)) \in M$ for each $(s,p,o) \in G_2$, thus showing that H has some ground instantiation which is true in M. Consider $a = (s,p,o) \in G_2$. Clearly, given that $G_1 \models_s G_2$, a is such that $(\mu(s),\mu(p),\mu(o)) \in R(G_1)$. Let S be the set of symbols appearing in P as in Lemma 1. Note that, if $(\mu(s),\mu(p),\mu(o)) \in S \times S \times S$, then by Lemma 1 $t(\mu(s),\mu(p),\mu(o))$ belongs to M.

However, consider the set $V = \{v \text{ appears in } G_2 \mid \mu(v) \notin S\}$. Note that V might be nonempty and its cardinality not larger than $|K|$. In such a case, we cannot directly show that $t(\mu(s),\mu(p),\mu(o)) \in M$, since μ might map some element of G_2 to a value outside S. Also note that all the elements $v \in V$ are such that $\mu(v) =$ *rdf:_i* for some $i \in \mathbb{N}$, and that G_1 and G_2 do not talk about $\mu(v)$ (otherwise, it would be that $\mu(v) \in S$). By Lemma 2, we

can replace μ with a mapping μ' such that: *(i)* $\mu'(v) = v'$ and $v' = rdf{:}\_i'_v$, for all $v \in V$, where i' (and thus v') can be arbitrarily chosen; *(ii)* $\mu \equiv \mu'$ elsewhere. We also define μ' such that its restriction over V is a bijection to the elements of $K' = \{k \mid symb(k) \in K\}$. Note that μ' is a mapping to elements of S and is such that $\mu'(G_2) \subseteq R(G_1)$. By Lemma 1, $t(\mu'(s), \mu'(p), \mu'(o))$ belongs to M for any $(s, p, o) \in G_2$. □

Theorem 2 *Determining whether* $P \models$ `entail` *is decidable.*

Proof. Theorem 5 of [10], states that answer sets of a vi-restricted program P' can be computed by computing the answer set of a finite, ground program G'_P, and that G'_P is computable. P is vi-restricted. Then, entailment of `entail` can be decided in finite time[9]. □

4 Implementation and experiments

By adapting the encoding obtained in the previous section the way is paved to an actual implementation of an RDF query system based on ASP. In particular, given the graph G_1 (the dataset to be queried) and a graph G_2 (a query pattern), and considering the program $P = T_1(G_1) \cup T_2(G_2) \cup D$:

• query answering is strictly related to entailment: indeed it amounts to finding all the possibilities to match G_2 to G_1. In our case, it is enough to modify the rule `entail` $\leftarrow H$ in $T_2(G_2)$, by changing `entail` to an atom $answer(X_1, \ldots, X_n)$ where variables $X_1, \ldots, X_n$ correspond to a subset of choice of blank nodes in G_2[10].

• The core of the SPARQL language is a syntactic method for expressing G_2 and the variables exposed in the predicate *answer*. A basic graph pattern written in SPARQL represents indeed a query pattern like G_2, which is converted to $T_2(G_2)$. Further, more involved features of SPARQL can be accommodated in our encoding by changing the transformation T_2, as explained in [26]. The same holds also for query patterns expressed in other non-standard query languages.

• D can be easily customized to the entailment regime of interest. For instance, if one is interested in ρDF entailment only, it is sufficient to remove from D all the rules except RDF1', RDFS2 and 3', IRDF2' and 3', RDFS5',6',7',8',9',10' and 11'.

In the following of this section we first provide some details on our implementation of the architecture presented in Figure 1(b); then we present the experimental framework we set up and obtained results.

[9] P is an ordinary program, except for the fourth rule of A', containing possible value invention. Intuitively, P is a "safe" rule, in the sense that all the variables in its head appear also in a positive predicate in its body. This prevents actual value invention. For space reasons, we cannot include here details about vi-restrictedness. The interested reader is referred to [10].

[10] As a simple Corollary of Theorem 1, we note that under RDFS entailment, the predicate *answer* encodes a finite subset F of all the possible answers Ans. If Ans is finite, then $F = Ans$.

4.1 Implementation issues

In order to implement the architecture we proposed in Figure 1(b) only two software components are needed: the `SPARQL to Datalog` translator module and the `ASP Query Engine` module.

Translation from SPARQL to Datalog is carried out by a module independently developed at DERI-Galway, and whose behavior is described in [26].

The ASP Query Engine is implemented by an improved version of the DLV^{DB} system we recently developed, and which we briefly describe next. DLV^{DB} is an extension of the well known ASP system DLV [19] designed both to handle input and output data distributed on several databases, and to allow the evaluation of logic programs directly on the databases. It combines the expressive power of DLV with the efficient data management features of DBMSs [17]. DLV^{DB} captures a wide fragment of the ASP^{EX} language outlined in Section 2.1. The detailed description of DLV^{DB} is out of the scope of the present paper; here we briefly outline the main peculiarities which, coupled with the results presented in the previous section, make it a suitable ASP-based ontology querying engine. The interested reader can find a complete description of DLV^{DB} and its functionalities in [35]; moreover, the system, along with documentation and examples, are available for download at `http://www.mat.unical.it/terracina/dlvdb`.

Generally speaking, DLV^{DB} allows for two kinds of execution: *(i)* direct database execution, which evaluates logic programs directly on the database, with a very limited usage of main-memory but with some limitations on the expressiveness of the queries, and *(ii)* main-memory execution, which loads input data from different (possibly distributed) databases and executes the logic program directly in main-memory. In both cases, interoperation with databases is provided by ODBC connections; these allow, in a quite simple way, the handling of data residing on various databases over the network.

For the purposes of this paper, it is particularly relevant the application of DLV^{DB} in the direct database execution modality for the querying of large ontologies. In fact, usually, the user has his data stored in (possibly distributed) triplestores and wants to carry out some reasoning on them; however the amount of this data can be such that the evaluation of the query cannot be carried out in main-memory. Then, it must be evaluated directly in mass-memory. These characteristics perfectly fit the capabilities of DLV^{DB}, provided that RDF(S) triplestores and inference rules are specified as described in Section 3. Moreover, DLV^{DB} turned out to be particularly efficient for reasoning about massive data sets (see benchmark results presented in [35]) and supports a sufficiently rich reasoning language for querying ontologies.

Three main features characterize the DLV^{DB} system in the direct database execution modality: *(i)* its ability to evaluate logic programs directly and completely on databases with a very limited usage of main-memory resources, *(ii)* its capability to map program predicates to (possibly complex and distributed) database views, and *(iii)* the possibility to easily specify which data is to be considered as input or as output for the program. In the application context considered in this paper, these characteristics allow the user to have a wide flexibility in querying available ontologies.

In order to exploit DLV^{DB} as an RDF(S) query engine, we tailored it to the peculiarities of the specific application context. In particular, we observed that the standard representation format of triplestores is a single, almost static, big table composed of three columns. Then, queries over the triplestore mostly consist of several joins involving the same table. As a consequence, we improved DLV^{DB} so as to automatically create both *views* and *indexes* over the underlying triplestore to improve querying performance.

4.2 Benchmark framework and results

In this section, we present the results of our experiments aiming at comparing the performance of DLV^{DB} with several state-of-the-art triplestores. The main goal of our experiments was to evaluate both the scalability and the query language expressiveness of the tested systems. All tests have been carried out on a Pentium 4 machine with a 3.00 GHz CPU, 1.5 Gbytes of RAM, and 500GB of mass memory, under Windows XP SP2 Operating System.

4.2.1 Compared Systems

In our tests we compared DLV^{DB} with three state-of-the-art triplestores, namely: Sesame [33], ARQ [2], and Mulgara [1]. The first two systems allow both in-memory and RDBMS storage and, consequently, we tested them on both execution modalities. In the following, we refer the in-memory version of Sesame (resp. ARQ) as Sesame-Mem (resp. ARQ-Mem) and the RDBMS version as Sesame-DB (resp. ARQ-DB). RDBMS versions of all systems (including DLV^{DB}) have been tested with Microsoft SQL Server 2005 as underlying database. As it will be clear in the following, we also tested a version of Sesame which works on files; we refer this version of Sesame as Sesame-File. For each system we used the latest available stable release at the time of writing. We next briefly describe them.

Sesame is an open source Java framework with support for storage and querying of RDF(S) data. It provides developers with a flexible access API and several query languages; however, its native language (which is the one adopted in our tests) is SeRQL – Sesame RDF Query Language. Actually, the current official release of Sesame does not support the SPARQL language yet. Some of the query language's most important features are: *(i)* expressive path expression syntax that match specific paths through an RDF graph, *(ii)* RDF Schema support, *(iii)* string matching. Furthermore, it allows simplified forms of reasoning on RDF and RDFS. In particular, inferences are performed by pre-materializing the closure $R(G)$ of the input triplestore G.

The latest official release is version 1.2.7. However, during the preparation of this manuscript, version 2.0 of Sesame became available as Release Candidate and, in order to guarantee fairness in the comparison, we considered also this version in our tests. In fact, version 2.0 supports the SPARQL Query Language and features an improved inferencing support, but does not support RDBMS management yet, allowing only files. In the following, we indicate Sesame 1.2.7 as `Sesame1` and Sesame 2.0 as `Sesame2`.

ARQ is a query engine for Jena (a framework for building Semantic Web applications, which is distributed at `http://jena.sourceforge.net`) that supports the SPARQL Query language. ARQ includes a rule-based inference engine and performs non materialized inference. As for Sesame, ARQ can be executed with data loaded both in-memory and on an RDBMS.

We executed SPARQL queries from Java code on the latest available version of ARQ (2.1) using the Jena's API in both execution modalities[11].
Mulgara is a database system specifically conceived for the storage and retrieval of RDF(S) (note that it is not a standard relational DBMS). Mulgara is an Open Source active fork of the Kowari project[12]. The adopted query language is *iTQL* (Interactive Tucana Query Language), a simple SQL-like query language for querying and updating Mulgara databases. A compatibility support with SPARQL is declared, yet not implemented. The Mulgara Store offers native RDF(S) support, multiple databases per server, and full text search functionalities. The system has been tested using its internal storage data structures (XA Triplestore). The latest release available for Mulgara is mulgara-1.1.1.

4.2.2 Benchmark Data Set

In order to provide an objective and comprehensive set of tests we adopted two different data sets: one coming from real data, that is, the DBLP database [21] and one coming from synthetic information, i.e. the LUBM benchmark suite [20].

DBLP is a real database containing a large number of bibliographic descriptions on major computer science journals and proceedings; the DBLP server indexes more than half a million articles and features several thousand links to home pages of computer scientists. Recently, an OWL ontology has been developed for DBLP data. A corresponding RDF snapshot has been downloaded at the Web address `http://sw.deri.org/~aharth/2004/07/dblp/`. The main classes represented in this ontology are *Author*, *Citation*, *Document*, and *Publisher*, where a *Document* can be one of: Article, Book, Collection, Inproceedings, Mastersthesis, Phdthesis, Proceedings, Series, WWW. In order to test the scalability of the various systems we considered several subsets of the entire database, each containing an increasing number of statements and constructed in such a way that the greater sets strictly contain the smaller ones. Generated data sets contain from 70000 to 2 million RDF triples.

The Lehigh University Benchmark (LUBM) has been specifically developed to facilitate the evaluation of Semantic Web triplestores in a standard and systematic way. In fact, the benchmark is intended to evaluate the performance of those triplestores with respect to extensional queries over large data sets that commit to a single realistic ontology. It consists of a university domain ontology with customizable and repeatable synthetic data. The LUBM ontology schema and its data generation tool are quite complex and their description is out of the scope of this paper. We used the Univ-Bench ontology that describes (among others) universities, departments, students, professors and relationships among them. The interested reader can find all information in [20]. Data generation is carried out by the Univ-Bench data generator tool (UBA) whose main generation parameter is the number of universities to consider. Also in this case, we generated several data sets, each containing an increasing number of statements and constructed in such a way that the greater sets strictly contain the smaller ones. Generated data sets are named as: `lubm-5`, corresponding to 5 universities and about 640000 RDF triples, `lubm-10` (10 universities, about 1 million

[11] Distributed at `https://jena.svn.sourceforge.net/svnroot/jena/ARQ/`
[12] `http://www.kowari.org/`

triples), `lubm-15` (15 universities, about 2 million triples), `lubm-30` (30 universities, about 4 million triples), `lubm-45` (45 universities, about 6 million triples).

4.2.3 Tested Queries

As previously pointed out, the expressiveness of the query language varies for each tested system. In order to compare both scalability and expressiveness, we exploited queries of increasing complexity, ranging from simple selections to queries requiring different forms of inferences over the data. Due to space constraints we cannot show here the encodings of all the tested queries for all the tested systems. The interested reader can find them in the Appendix available at `http://www.mat.unical.it/terracina/dlvdb/JLC/Appendix.pdf`.

Queries on DBLP We ran the following five queries over DBLP:

– Q_1: Select the names of the Authors and the URI of the corresponding Articles they are author of.
– Q_2: Select the names of the Authors which published at least one Article in year 2000.
– Q_3: Select the names of the Authors which are creators of at least one document (i.e. either an Article, or a Book, or a Collection, etc.).
– Q_4: For each author in the database, select the corresponding name and count the number of Articles he published.
– Q_5: Given a pair of Authors A_1 and A_2, compute the "collaborative distance" between them; the collaborative distance can be computed as the minimum length of a path connecting A_1 and A_2 in a graph where Authors are the nodes of this graph and an edge between A_i and A_j indicates that A_i co-authored at least one document with A_j.

Here, queries Q_1 and Q_2 are simple selections; Q_3 requires a simple form of inference; in fact articles, books, etc. must be abstracted into documents. Query Q_4 requires the capability to aggregate data. Finally, Q_5 requires to perform a transitive closure over the underlying data and, consequently, the corresponding query must be recursive.

It is worth observing that queries Q_1, Q_2, and Q_3 can be executed by all the evaluated systems. As for Q_3, we exploited the Krule engine for Mulgara, the inferencing repository in Sesame and the inferencing Reasoner in ARQ. Note that Mulgara and Sesame-DB materialize the possible inferenced data just during the loading of the RDF dataset in the database; however, in our tests, we measured only the query answering times. Query Q_4 cannot be evaluated by Sesame because the SeRQL query language does not support aggregate operators. Finally, query Q_5 can be evaluated only by DLV^{DB}, because it is the only system allowing for recursive queries.

Queries on LUBM The LUBM benchmark provides 14 test queries. Many of them are slight variants of one another. Most of the queries basically select subsets of the input data and require, in some cases, basic inference processes (e.g. class-subclass inferences). Few of them are intended to verify the presence of certain reasoning capabilities (peculiar

of OWL ontologies rather than RDFS) in the tested systems; in fact, they require the management of transitive properties. However, some important querying capabilities, such as data aggregation and recursion, are not addressed by the queries in the LUBM benchmark. As a consequence we designed also other queries over the LUBM data set to test these higher expressive features provided by DLV^{DB} and some other systems.

Queries taken from the LUBM benchmark are listed below (in parentheses, we give the official query number as given by LUBM):

• Q_6 (LUBM Query1): Select Graduate Students attending a given Course. It is assumed that no hierarchy information or inference exists or is required (that is, no RDFS entailment is required).

• Q_7 (LUBM Query2): Select the Graduate Students X, Universities Y and Departments Z such that X graduated in Y and is a member of Z, and Z is a sub-organization of Y. Note that this query involves three classes and three properties; moreover, there is a triangular pattern of relationships between the objects involved.

• Q_8 (LUBM Query3): Select the Publications of a given Assistant Professor. Here Publication has a wide hierarchy of subclasses, subject to RDFS entailment.

• Q_9 (LUBM Query4): Select the Professors working for a specific Department, showing their names, telephone numbers and email addresses. This query covers multiple properties of the single class Professor, having many subclasses, and requires a long join pattern.

• Q_{10} (LUBM Query5): Select the Persons which are member of a specific Department; this query assumes a *subClassOf* relationship between Person and its subclasses and a *subPropertyOf* relationship between memberOf and its subproperties.

• Q_{11} (LUBM Query6): Select all instances of the class Student, considering also its subclasses Graduate Student and Undergraduate Student.

• Q_{12}: (LUBM Query7): Select the Students taking some Course held by a specific Associate Professor.

• Q_{13}: (LUBM Query8): Select the Students having an email address, which are member of a Department that is a sub-organization of a given University.

• Q_{14} (LUBM Query9): Select the Students attending some Course held by a teacher who is also their advisor. This query involves the highest number of classes and properties in the query set; moreover, there is a triangular pattern of relationships.

• Q_{15} (LUBM Query14): Select all the Undergraduate Students. This query is very simple, yet characterized by a large input and a low selectivity.[13]

Additional queries that we have tested, not included in the LUBM benchmark, are listed below.

[13] A more exhaustive description of these queries can be found at `http://swat.cse.lehigh.edu/projects/lubm/query.htm`.

System/Query	Q_6	Q_7	Q_8	Q_9	Q_{10}	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}	Q_{16}	Q_{17}
DLV^{DB}	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Mulgara	Y	Y	Y							Y	Y	
Sesame1-Mem	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		
Sesame1-DB	Y	Y	Y	Y						Y		
Sesame2-Mem	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		
Sesame2-File	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		
ARQ-Mem	Y		Y	Y		Y				Y	Y	
ARQ-DB												

Table 1: Summary of the systems capable of computing the results for the data sets in LUBM queries, under the specified constraints. Y=Yes, blank=No.

• Q_{16}: For each Author in the database count the number of Papers she/he published. This query requires the capability to aggregate data.

• Q_{17}: Given a pair of authors A_1 and A_2, compute the "collaborative distance" between them (see query Q_5 for the definition of collaborative distance). This query requires the capability to express recursion.

4.2.4 Results and Discussion

Results on DBLP Figure 6 shows the results we have obtained for the DBLP queries. In the figure, the chart of a system is absent whenever it is not able to solve the query due to some system's fault or if its response time is (except for Q_5) greater than 3600 seconds (1 hour). Moreover, if a system's query language is not sufficiently expressive to answer a certain query, it is not included in the graph. From the analysis of the figure, we can draw the following observations.

DLV^{DB} shows always the best performance for all the queries, yet it is characterized by the highest language expressiveness. Mulgara and ARQ have, after DLV^{DB}, the more expressive query language; however, Mulgara is not able to complete the computation of Q_3 and Q_4 already after 220000 triples and is not able to express Q_5, whereas ARQ always shows higher time responses than both DLV^{DB} and Mulgara, except for Q_4 where it performs sensibly better than Mulgara. Sesame turns out to be competitive in version Sesame2-File, especially for the biggest data sets in Q_1 and Q_3, but scores the worst performance among all the systems for all queries in version Sesame1-DB. As far as Sesame is concerned, it is particularly interesting to observe the very different behaviour of its various versions; in particular, as for Sesame1 the in-memory version works much better than its DB version; vice versa, the file version of Sesame2 performs much better than its in-memory version. In any case, Q_4 and Q_5 could not be tested with Sesame due to lack of language expressiveness.

Results on LUBM Figures 7 and 8 show the results we have obtained for the LUBM queries. In the figure, the chart of a system is absent whenever it is not able to solve the query due to some system's fault, or if its response time is (except for Q_{17}) greater than 3600 seconds (1 hour). The maximum allowed size for pre-materialized inference has been set to 4Gb (note that the greatest considered triple set occupied 1Gb in the database). Finally, if

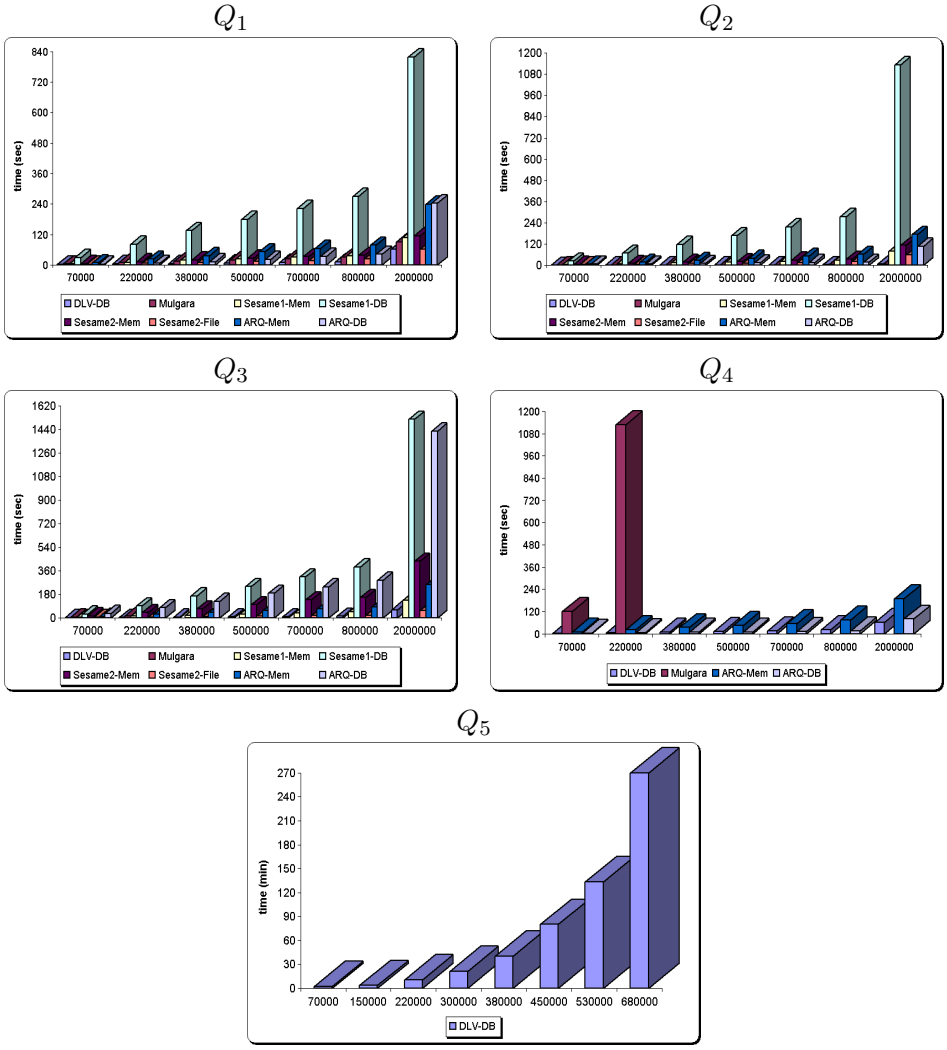

Figure 6: Results for queries Q_1 - Q_5

a system's query language is not sufficiently expressive to answer a certain query, it is not included in the graph.

Table 1 summarizes the observed capability of each system to complete the computation of query results over all the considered data sets, under the limitations specified above.

From the analysis of Figures 7 and 8 and of Table 1, we can draw the following observations. DLV^{DB} is the only system capable of completing the computation of all the queries under the time and space constraints specified previously. Only Sesame is competitive with DLV^{DB} on this aspect; on the other hand, queries Q_{16} and Q_{17} cannot be computed by this system due to lack of language expressiveness.

DLV^{DB} scores also the best performance over all queries, except query Q_{14} where Sesame has the best time responses. As far as this aspect is concerned, however, it is worth recalling that Sesame and Mulgara pre-materialize inferences during loading of data into the reposi-

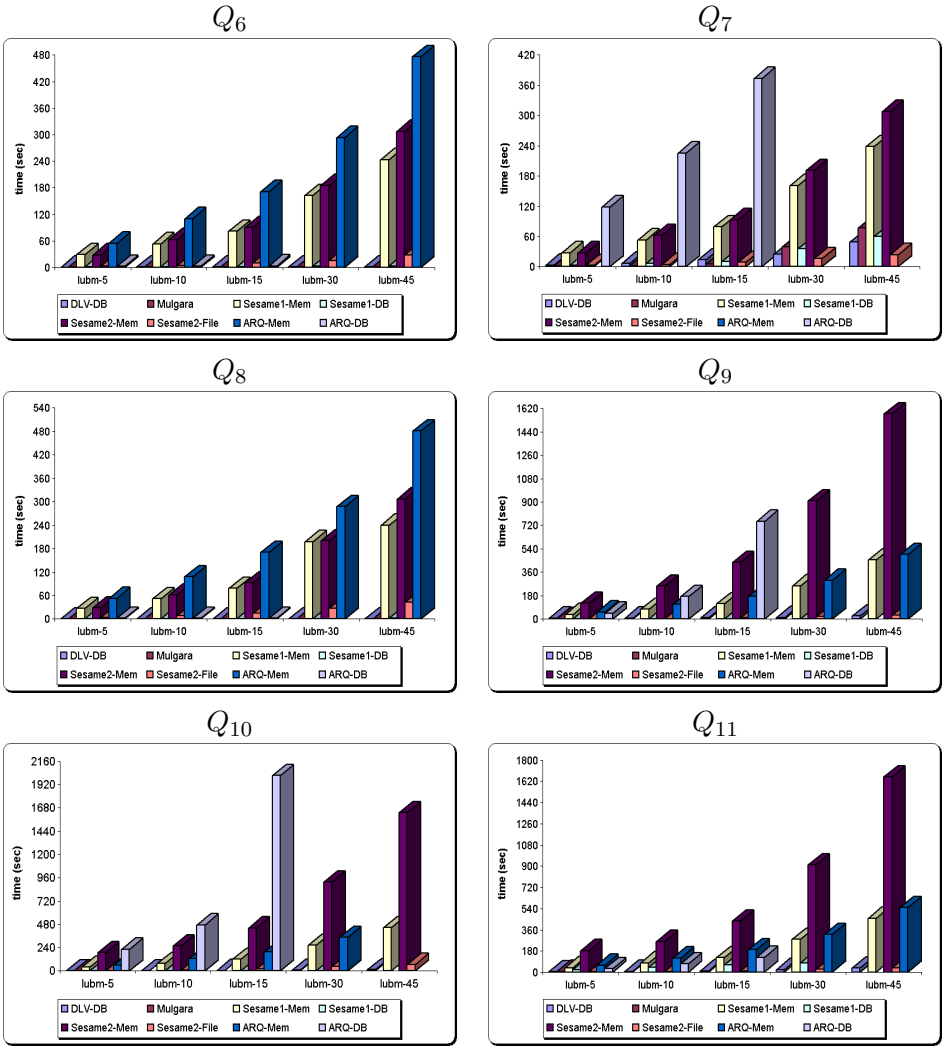

Figure 7: Results for queries Q_6 - Q_{11}

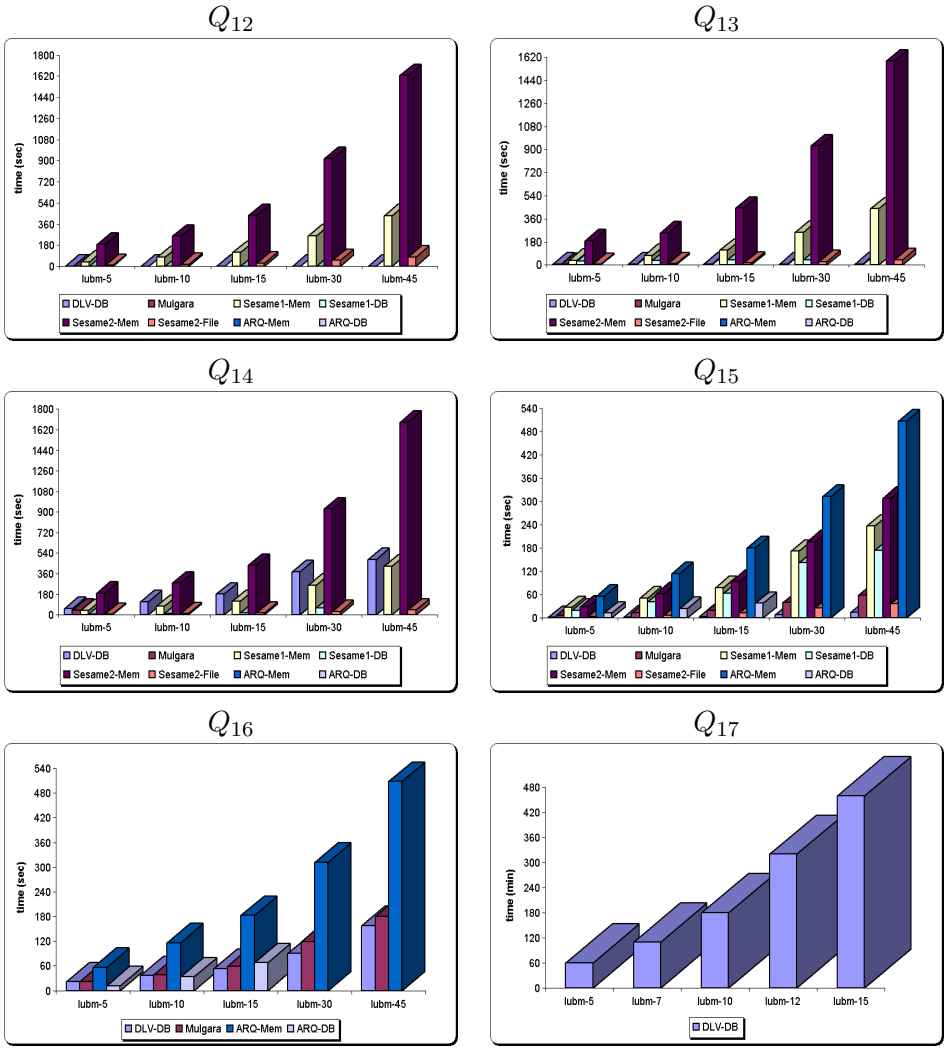

Figure 8: Results for queries Q_{12} - Q_{17}

tory; as previously pointed out, pre-materialization times are not considered in the graphs, since this task can be carried out once for each data set[14]. For the sake of completeness, however, we show in Table 2 the times (in minutes) required by the various systems for loading test data in the corresponding repositories; for systems pre-materializing inferences, we distinguish between the loading of data with and without pre-materialization. From this table and Table 1 we can also observe that Mulgara is not able to compute inference in a reasonable time (we stopped it after 14 hours) already for `lubm-10`; this caused it to be unable to compute queries Q_9-Q_{14} for data sets `lubm-10`, `lubm-15`, `lubm-30`, `lubm-45`.

	DLV	ARQ	Sesame1-DB		Mulgara		Sesame2-File	
Triples				**inference**		**inference**		**inference**
lubm-5	5	10	13	45	5	46	3	12
lubm-10	10	21	28	116	14	**	5	30
lubm-15	15	108	61	221	30	**	9	50
lubm-30	31	*	102	869	106	**	19	138
lubm-45	46	*	161	*	196	**	31	261

Table 2: Loading times (minutes) for each system and data set. Whenever needed we distinguish between loading with and without inference pre-materialization. *: DB space exceeded. **: More than 14 hours.

As for the behaviour of the other systems, it is worth noting that for the LUBM data sets DB versions of the various systems perform generally better than the corresponding main-memory versions. This behaviour can be explained by considering that LUBM data sets are quite large and, consequently, they could not fit in main memory, thus causing swapping of main-memory systems.

In order to further characterize tested systems, we have measured their main-memory footprint for some queries. Table 3 shows the results obtained for queries Q_6, Q_9, Q_{11}, and Q_{15} run on the dataset `lubm-15`. Recall that Q_9 and Q_{11} require inference. From the analysis of this table it clearly emerges that DLV^{DB} is the lightest system in terms of memory occupation. Moreover, it comes with no surprise that, in general, the systems requiring the lower amount of main-memory are those implementing a mass-memory based evaluation; the only exception is made by ARQ-DB in Q_9 and Q_{11} (requiring inference). This can be explained by considering that ARQ applies inference rules at query time and, visibly, this task is carried out in main memory. These results, coupled with the execution times previously analyzed, make DLV^{DB} the system requiring overall less time and space among the tested ones.

5 Conclusions

In this paper we presented an efficient and reliable ASP-based architecture for querying RDF(S) ontologies. We have first formally shown how RDF(S) can be mapped (without loss of semantics) to an Answer Set Programming language. Then, we experimentally proved that our solution, based on a database oriented implementation of ASP, improves both

[14]Recall that Q_9-Q_{14} are indeed the LUBM queries requiring inference.

System/Query	Q_6	Q_9	Q_{11}	Q_{15}
DLV^{DB}	4.88	5.07	5.08	5.03
Mulgara	53.12	-	-	74.23
Sesame1-Mem	256.98	318.52	320.11	268.89
Sesame1-DB	23.31	23.64	38.98	37.47
Sesame2-Mem	291.83	368.65	370.73	292.22
Sesame2-File	15.87	16.05	16.00	15.91
ARQ-Mem	309.38	329.92	331.80	312.78
ARQ-DB	17.31	343.14	407.73	154.98

Table 3: Memory footprint (Megabytes) of tested systems for some queries

scalability and expressiveness of several state-of-the-art systems. Currently, RDF data are stored in our databases with the standard triple format, by collecting all the information in a single table of triples; this, obviously, worsens performance. The representation of data in some more structured form (as already some of the tested systems do) could significantly improve performance. Another promising research line consists in using database integration techniques in the ontology context such as in [11].

References

[1] T. Adams, G. Noble, P. Gearon, and D. Wood. MULGARA homepage. Available at `http://www.mulgara.org/` (Last accessed on 22 July 2008).

[2] ARQ homepage. Available at `http://jena.sourceforge.net/ARQ/` (Last accessed on 22 July 2008).

[3] A. Analyti, G. Antoniou, C. V. Damásio, and G. Wagner. Stable model theory for extended RDF ontologies. In *ISWC*, pages 21–36. Springer, Berlin/Heidelberg, 2005.

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In Proceedings of *ISWC/ASWC*, pages 722–735, 2007. Springer, Berlin/Heidelberg, 2007.

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

[6] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[7] C. Bizer. D2r map - a database to RDF mapping language. In *WWW (Posters)*, 2003.

[8] J. de Bruijn and S. Heymans. Logical foundations of (E)RDF(S): Complexity and reasoning. In Proceedings of *ISWC/ASWC*, pages 86–99. Springer, Berlin/Heidelberg, 2007.

[9] J. de Bruijn, E. Franconi, and S. Tessaris. Logical reconstruction of RDF and ontology languages. In Proceedings of *PPSWR*, pages 65–71. Springer, Berlin/Heidelberg, 2005.

[10] F. Calimeri , S. Cozza , and G. Ianni. External sources of knowledge and value invention in logic programming. *AMAI*, 50(3-4):333–361, 2007.

[11] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati MASTRO-I: Efficient integration of relational data through DL Ontologies DL workshop. In Proceedings of *DL workshop*, CEUR Electronic Workshop Proceedings, pp. 227234. RWTH Aachen University, Germany, 2007. Available at http://ceur-ws.org/Vol-250/.

[12] D2r server publishing the DBLP bibliography database. Available at `http://www4.wiwiss.fu-berlin.de/dblp/` (Last accessed on 22 July 2008).

[13] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. . Artificial Intelligence Journal, 172, 14951539, 2008.

[14] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In Proceedings of *IJCAI*, pp. 9096. Professional Book Center, Denver, CO, USA, 2005.

[15] W. Faber and G. Pfeifer. `DLV` homepage. Available at `http://www.dlvsystem.com/` (Last accessed on 22 July 2008).

[16] D. Fensel, W. Wahlster, H. Lieberman, and J. Hendler, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, Cambridge, MA, 2002.

[17] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database System Implementation*. Prentice Hall, Upper Saddle River, NJ, 2000.

[18] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *ICLP/SLP*, pages 1070–1080, 1988. MIT Press, Cambridge, MA, 1988.

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

[20] LUBM homepage. Available at `http://swat.cse.lehigh.edu/projects/lubm/` (Last accessed on 22 July 2008).

[21] M. Ley. Digital bibliography and library project. Available at `http://dblp.uni-trier.de/` (Last accessed on 22 July 2008).

[22] D. Marin A Formalization of RDF. TR Dept. Computer Science, Universidad de Chile, TR/DCC-2006-8. Available at `http://www.dcc.uchile.cl/cgutierr/ftp/draltan.pdf` (Last accessed on 22 July 2008).

[23] B. Motik and R. Rosati. A faithful integration of description logics with logic programming. In Proceedings of International Joint Conferences on Artificial Intelligence, pages 477–482, 2007. California, USA, 2007.

[24] I.S. Mumick, S.J. Finkelstein, H. Pirahesh, and R. Ramakrishnan. Magic conditions. *ACM TODS*, 21(1):107–155, 1996.

[25] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL web ontology language semantics and abstract syntax. W3C recommendation. Available at http://www.w3.org/TR/owl-semantics/ (Last accessed on 22 July 2008).

[26] A. Polleres. From SPARQL to rules (and back). In Proceedings of WWW, ACM Press, NewYork, NY, USA, 2007. Extended technical report version available at `http://www.polleres.net/publications/GIA-TR-2006-11-28.pdf`

[27] A. Polleres. Personal Communication 2007.

[28] S. Muñoz, J. Pérez, and C. Gutiérrez. Minimal deductive systems for RDF. In Proceedings of ESWC, pp. 5367. Springer, Berlin/Heidelberg, 2007.

[29] RDF semantics. W3C recommendation 10 february 2004. Available at `http://www.w3.org/TR/rdf-mt/` (Last accessed on 22 July 2008).

[30] RDF resource guide. Available at `http://planetrdf.com/guide/` (Last accessed on 22 July 2008).

[31] A. Seaborne E. Prud'hommeaux. SPARQL query language for RDF. W3C candidate recommendation, 14 June 2007. Available at `http://www.w3.org/TR/rdf-sparql-query/` (Last accessed on 22 July 2008).

[32] H. Stuckenschmidt and J. Broekstra. Time - space trade-offs in scaling up rdf schema reasoning. In Proceedings of WISE Workshops, pp. 172181. Springer, Berlin/Heidelberg, 2005.

[33] SESAME homepage. Available at `http://www.openrdf.org/` (Last accessed on 22 July 2008).

[34] SPARQL implementations. Available at `http://esw.w3.org/topic/sparql` implementations (Last accessed on 22 July 2002).

[35] G. Terracina, N. Leone, V. Lio, and C. Panetta. Experimenting with recursive queries in database and logic programming systems. *TPLP, 8(2):129–165, 2008. Available on-line at* `http://arxiv.org/abs/0704.3157`.