Towards In-Memory RDFS Entailment

Javier D. Fernández[1,2], Claudio Gutiérrez[3], Miguel A. Martínez-Prieto[1], Jorge Pérez[3]

[1] DataWeb Research, Department of Computer Science, Univ. de Valladolid (Spain)
{jfergar,migumar2}@infor.uva.es
[2] Ontology Engineering Group (OEG), Univ. Politécnica de Madrid (Spain)
jdfernandez@fi.upm.es
[3] Department of Computer Science, Univ. de Chile (Chile)
{cgutierr,jperez}@dcc.uchile.cl

1 Motivation

Massive publication efforts have enriched the Web with huge amounts of semantic data represented in RDF [7], and reasoning tasks at such scale are a formidable challenge. RDF Schema (RDFS) [6] defines the most simple inference in RDF introducing a vocabulary with predefined semantics to describe relationships such as typing of entities and hierarchy relations in classes and properties. This vocabulary allows one to infer new facts, not originally explicit in the RDF graph, by means of a process called *RDFS entailment*.

Traditionally, two types of solutions tackle RDFS entailment. On the one hand, all facts which can be inferred from an RDF graph can be materialized and added to the graph. This is referred to as the *graph closure*, which allows to easily check if a triple is inferred from the graph. However, the closure can be of size quadratic in the size of the initial graph which is not a practical bound from a database point of view [9]. Although some approaches can compute huge closures on the basis of distributed systems [11, 12], the final (potentially massive) closure has to be still managed and queried, paying costly latencies. On the other hand, one could maintain the original graph and check the entailment on-demand. Unfortunately, these solutions have to pay a potentially large number of I/O accesses at huge scale [5], which disregard a broader adoption whenever a fast response prevails.

This scenario claims for forms of lightweight entailment which could make reasoning feasible at Web scale. Solutions will necessarily involve finding space/time tradeoffs that (i) save storage requirements, and (ii) minimize I/O costs. Some clever form of compression would address both issues which are closely related. In fact, an in-memory solution enables these two objectives to be achieved if the compressed data can be directly accessed without prior decompression, optimizing the memory footprint. This solution would dissuade to maintain graph closures and allows to design an efficient on demand algorithm which performs triple checking in main memory.

Our ongoing work, described in the next section, follows the above ideas by compressing the RDF graph using RDF/HDT [3], a data structure known to assure a reduced memory footprint while providing fast triple pattern resolution in main memory [8, 4].

2 Ongoing Research

In our work we consider the minimal RDFS subset proposed by Muñoz, *et al.* [9], which includes all the relevant RDFS keywords `rdfs:subPropertyOf` (`sp`), `rdfs:subClassOf` (`sc`), `rdf:type` (`type`), `rdfs:domain` (`dom`), and `rdfs:range` (`range`). This fragment preserves the original RDFS semantics [6], and allows RDFS inference by just checking the existence of some paths in the original graph, obtaining a good theoretical upper-bound on the cost of RDFS entailment [9, 10]. Although some proposals have implemented this minimal RDFS inference [5], their performance is far from matching the theoretical complexity bounds mainly because of the I/O cost aspects. Thus, we address an in-memory compressed solution for huge graphs, minimizing I/O costs and approaching the promised theoretical optimal performance.

RDFS-HDT. HDT (*Header-Dictionary-Triples*) is a succinct data structure,and serialization format, designed for storing, exchanging and basic querying of RDF compressed data [8, 2]. It is based on two main components: the *HDT Dictionary* maps each term in an RDF graph to a unique identifier (ID), enabling the *HDT Triples* to manage RDF as a more efficient graph of IDs. Both components are represented and indexed succinctly on the basis of compressed string dictionaries and compact graph encodings. For instance, the standard corpus of *DBpedia 3.8* (which comprises more than 431 million triples and takes more than 60GB of space in plain format) can be managed in less than 7 GB of memory, and resolves SPARQL triple patterns at the level of microseconds (see [2] for further details).

Our ongoing approach for RDFS entailment is referred to as *RDFS-HDT*. Given an RDF graph G, RDFS-HDT maintains the original *HDT Dictionary* concept, but it divides the graph encoded in the *HDT triples* in three different subgraphs:

- The *subproperty* subgraph, $G(sp)$ comprises all triples `(subject, sp, object)`.
- The *subclass* subgraph, $G(sc)$ comprises all triples `(subject, sc, object)`.
- All the remaining triples are managed in the *general* subgraph G'.

First, this decision assures that the space required by RDFS-HDT is, in the worst case, similar to that required by a general HDT. Then, this data reorganization enables RDFS entailment to be performed in main memory following the approach in [9, 10]:

(a) The original HDT retrieval features are used to resolve those triple patterns involving `dom`, or `range` properties. In order to speed up these queries, the triples involving `dom` and `range` can be extracted from G' and indexed independently.
(b) The location of elements related by paths of `sc` or `sp` properties is performed over the specific $G(sp)$ or $G(sc)$ subgraphs respectively; RDFS-HDT resolves successive patterns of the form `(subject, sc, ?subclass)` (similar for `sp`) until the required triple is entailed, or the corresponding path is finished without success.

(c) Addressing `type` properties, and general triples of the form (a, p, b), with p not an RDFS keyword, needs a combination of both approaches, using the original HDT retrieval plus `sc` and `sp` path inference.

The next step in our ongoing project is to leverage compressed tree representations [1] for encoding the `sp` and `sc` hierarchies. This would allow us to store $G(sp)$ and $G(sc)$ in minimal space while efficiently supporting *ancestor-queries* over these hierarchies, which is the main feature needed in the algorithms proposed in [9, 10]. These advances will report some space/time tradeoffs allowing RDFS-HDT adoption under different computational configurations at Web scale.

Acknowledgments

Claudio Gutiérrez and Jorge Pérez are funded by the Millennium Nucleus Center for Semantic Web Research, Grant NC120004.

References

1. D. Arroyuelo, R. Cánovas, G. Navarro, and K. Sadakane. Succinct Trees in Practice. In *Proc. of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 84–97, 2010.
2. J.D. Fernández. *Binary RDF for Scalable Publishing, Exchanging and Consumption in the Web of Data*. PhD thesis, University of Valladolid, Spain, 2014.
3. J.D. Fernández, M.A. Martínez-Prieto, C. Gutiérrez, and A. Polleres. *Binary RDF Representation for Publication and Exchange (HDT)*. W3C Member Submission, 2011. `http://www.w3.org/Submission/2011/03/`.
4. J.D. Fernández, M.A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary RDF Representation for Publication and Exchange. *Journal of Web Semantics*, 19:22–41, 2013.
5. W.J. Haffmans and G.H.L. Fletcher. Efficient RDFS Entailment in External Memory. In *Proc. of the OnTheMove Workshops (OTM)*, pages 464–473, 2011.
6. P. Hayes. *RDF Semantics*. W3C Recommendation, 2004. `http://www.w3.org/TR/rdf-mt/`.
7. F. Manola and E. Miller, editors. *RDF Primer*. W3C Recommendation, 2004. `http://www.w3.org/TR/rdf-primer/`.
8. M.A. Martínez-Prieto, M. Arias, and J.D. Fernández. Exchange and Consumption of Huge RDF Data. In *Proc. of the 9th Extended Semantic Web Conference (ESWC)*, pages 437–452, 2012.
9. S. Muñoz, J. Pérez, and C. Gutiérrez. Simple and Efficient Minimal RDFS. *Journal of Web Semantics*, 7(3):220–234, 2009.
10. J. Pérez, M. Arenas, and C. Gutiérrez. nSPARQL: A navigational language for RDF. *Journal of Web Semanticas*, 8(4):255–270, 2010.
11. J. Urbani, S. Kotoulas, J. Maassen, F. Van Harmelen, and H. Bal. OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples. In *Proc. of the 7th Extended Semantic Web Conference (ESWC)*, pages 213–227, 2010.
12. J. Weaver and J.A. Hendler. Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples. In *Proc. of the 8th International Semantic Web Conference (ISWC)*, pages 682–697, 2009.