OBDA with *Ontop*

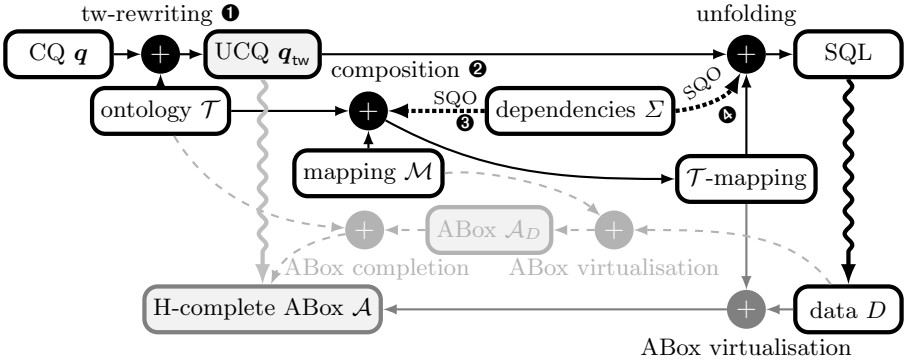
Mariano Rodríguez-Muro[1], Roman Kontchakov[2] and Michael Zakharyaschev[2]

[1] Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
[2] Dept. of Computer Science and Inf. Syst., Birkbeck, University of London, U.K.

Abstract. We evaluate the performance of the OBDA system *Ontop* and compare it with other systems. Our experiments show that (*i*) the *Ontop* tree-witness query rewriter is fast and outperforms the competitors, and (*ii*) query evaluation in *Ontop* using the semantic index is as efficient as in materialisation-based systems (but without the materialisation overhead).

1 Introduction

Ontop (`ontop.inf.unibz.it`) is an ontology-based data access (OBDA) system implemented at the Free University of Bozen-Bolzano and available as a plugin for Protégé 4, SPARQL end-point and OWLAPI and Sesame libraries. The architecture of *Ontop* is as follows:



A user formulates a conjunctive query (CQ) $\boldsymbol{q}$ in the signature of an OWL 2 QL ontology $\mathcal{T}$. The ontology $\mathcal{T}$ is related to the data D by means of a GAV mapping $\mathcal{M}$, which is a collection of database queries defining the vocabulary of $\mathcal{T}$. The mapping $\mathcal{M}$ could be applied to the data D to obtain the so-called *virtual ABox* $\mathcal{A}_D$ [9]. *Ontop* does not materialise it. Instead, it constructs a $\mathcal{T}$-mapping [9] by taking the composition (❷) of $\mathcal{M}$ with the taxonomy in $\mathcal{T}$ and simplifying it using SQL features (disjunction and interval expressions) and Semantic Query Optimisation (SQO ❸) with database integrity constraints (dependencies Σ). By applying the resulting $\mathcal{T}$-mapping to the data D, we could obtain another virtual ABox, $\mathcal{A}$, which is *H-complete with respect to* $\mathcal{T}$ in the sense that:

$$A(a) \in \mathcal{A} \quad \text{if} \quad A'(a) \in \mathcal{A}, \mathcal{T} \models A' \sqsubseteq A \quad \text{or} \quad R(a,b) \in \mathcal{A}, \mathcal{T} \models \exists R \sqsubseteq A,$$
$$P(a,b) \in \mathcal{A} \quad \text{if} \quad R(a,b) \in \mathcal{A} \text{ and } \mathcal{T} \models R \sqsubseteq P.$$

Ontop constructs the tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}$ of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes [5], and then uses the $\mathcal{T}$-mapping to unfold $\boldsymbol{q}_{\mathsf{tw}}$ to an SQL query, which is then simplified with SQO (❹) and passed to a database system for evaluation.

In the remainder of the paper, we evaluate the performance of *Ontop* and compare it with other OBDA systems (more details can be found at `tinyurl.com/ontop-benchmark` and `www.dcs.bbk.ac.uk/~roman/tw-rewriting`). We begin by analysing the structure of tree-witness rewritings over H-complete ABoxes and show that dealing with concept/role hierarchies (taxonomies) is the most critical component of any OBDA system. We then concentrate on a particular shape of $\mathcal{T}$-mappings, called the semantic index [9], which is most suitable for triple stores or data in the form of universal tables in a database and which allows *Ontop* to deal with taxonomies efficiently.

2 Tree Witnesses: The Topology of *Ontop* Rewritings

We have run the *Ontop* tree-witness rewriter on the usual set of CQs and ontologies: Adolena, StockExchange and the extension $\text{LUBM}^{\exists}_{20}$ [6] of LUBM [7] tailored to stress query answering techniques for OWL 2 QL. Our aim was to understand the size of the *topological* part of the rewritings reflecting matches (tree witnesses) in the anonymous part of the canonical models (as opposed to the taxonomical one). The tables below provide the statistics on the tree-witness rewritings over H-complete ABoxes: the number of tree witnesses, the number of CQs in the rewriting (which is a UCQ), the number of atoms in the original query, and the number of atoms in each of the CQs in the rewriting.

	Adolena					StockExchange				
	s_1	s_2	s_3	s_4	s_5	s'_1	s'_2	s'_3	s'_4	s'_5
tree witnesses	1	1	0	1	0	0	0	0	0	0
CQs in $\boldsymbol{q}_{\mathsf{tw}}$	**2**	**2**	**1**	**2**	**1**	**1**	**1**	**1**	**1**	**1**
atoms in $\boldsymbol{q}$	2	3	5	3	5	1	3	5	5	7
atoms in $\boldsymbol{q}_{\mathsf{tw}}$	2+2	1+3	5	2+3	5	1	1	3	2	4

For $\text{LUBM}^{\exists}_{20}$, r_1–r_5 are the queries from the Requiem evaluation [7], q_1–q_6 from the combined approach evaluation [6], and q_7–q_9 from the Clipper evaluation [11]:

	r_1	r_2	r_3	r_4	r_5	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
tree witnesses	0	0	0	0	0	1	1	0	1	0	0	0	3	1
CQs in $\boldsymbol{q}_{\mathsf{tw}}$	**1**	**1**	**1**	**1**	**1**	**2**	**2**	**1**	**2**	**1**	**1**	**1**	**1**	**1**
atoms in $\boldsymbol{q}$	2	3	6	3	4	8	4	6	8	5	8	13	13	34
atoms in $\boldsymbol{q}_{\mathsf{tw}}$	2	1	4	1	2	4+6	3+4	5	5+8	4	6	12	6	33

Note first that these CQs and ontologies have very few tree witnesses. More precisely, in 67% of the cases there are *no tree witnesses* at all, and in 29% we have only one tree witness. Even for the specially designed q_8, the structure of tree witnesses is extremely simple (they do not even overlap). Note also that, although q_8 and q_9 do have tree witnesses, the resulting UCQs contain only one CQ because these tree witnesses are generated by other atoms of the queries.

The size of each of the CQs in the rewritings does not exceed the size of the input query; moreover, the domain/range optimisation simplifies the obtained CQs further for r_2–r_5, q_1, q_3, q_5–q_8 and most of the StockExchange queries. To illustrate, consider the following subquery of q_8:

$$\boldsymbol{q}_0(x_0) = \mathit{Publication}(x_0),\ \mathit{publicationAuthor}(x_0, x_{11}),\ \mathit{Subj1Professor}(x_{11}),\\ \mathit{worksFor}(x_{11}, x_{12}),\ \mathit{Department}(x_{12}),$$

where x_{11}, x_{12} do not occur in the remainder of q_8. This CQ has a tree witness with the last two atoms because of the $\text{LUBM}^{\exists}_{20}$ axiom $\mathit{Faculty} \sqsubseteq \exists \mathit{worksFor}$. However, *Subj1Professor* is a subconcept of *Faculty*, and so any of its instances will anyway be connected to *Department* by *worksFor* (either in the ABox or in the anonymous part). It follows that the last two atoms of $\boldsymbol{q}_0$ do not change answers to the CQ and can be ignored. The first atom is also redundant because the ontology contains the domain axiom $\exists \mathit{publicationAuthor} \sqsubseteq \mathit{Publication}$. As $\boldsymbol{q}_0$ represents a natural and common pattern for expressing queries—select a *Publication* whose *publicationAuthor* is a *Subj1Professor*, etc.—any OBDA system should be able to detect such redundancies automatically. Finally, we note that all of the rewritings in our experiments contain at most two CQs.

For comparison, we computed the rewritings of the CQs over $\text{LUBM}^{\exists}_{20}$ using Requiem [7], Nyaya [4], IQAROS (v 0.2) [10], Clipper (v 0.1) [3] and Rapid (v 0.3) [2]. The first 3 return UCQ rewritings, while the last 2 nonrecursive datalog rewritings, with Rapid having an option to rewrite into UCQs, too. The rewritings are over arbitrary ABoxes; to make them comparable with *Ontop*, in the last 2 cases we omitted the taxonomical rules for completing the ABoxes.

	r_1	r_2	r_3	r_4	r_5	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
UCQ (number of CQs)														
Requiem	2	1	23	2	10	DNF	2	DNF	14,880	690	DNF	DNF	DNF	DNF
Nyaya	2	1	23	2	10	DNF	2	DNF	DNF	690	DNF	DNF	8	DNF
IQAROS	2	1	23	2	10	DNF	**1**	15,120	14,400	990	23,552	DNF	DNF	DNF
Rapid	2	1	23	2	10	3,887	2	15,120	14,880	690	23,552	DNF	DNF	16
datalog (number of non-taxonomical rules)														
Rapid	1	1	1	1	1	2	3	1	2	1	1	1	27	1
Clipper	1	1	1	1	1	8	7	1	5	1	1	1	512	16
tw-rewriter	1	1	1	1	1	2	2	1	2	1	1	1	1	1

The UCQs returned by Requiem, Nyaya, IQAROS and Rapid correspond to our tree-witness rewritings with every concept/role replaced by all of its subconcept/subroles (IQAROS's rewritings of q_2 and possibly of q_4, q_5 are incorrect): for instance, q_7 produces 216,000 ($= 30^3 \times 2^3$) CQs, q_3 produces 15,120 ($=4\times5\times21\times36$) CQs and q_1 produces 3,887 ($= 23 + 2\times4\times21\times23$) CQs because *Student*, *Faculty* and *Professor* have 23, 36 and 30 subconcepts, respectively, *worksFor* has 2 subroles, etc. Evidently, these large UCQs are generated by the taxonomies and none of the query subsumption algorithms can reduce the number of CQs in them. In fact, all four systems require substantial resources (in particular, for query subsumption checks): e.g., Nyaya rewrites q_8 in 91s and runs out of memory on q_1, q_3, etc.; IQAROS rewrites q_6 in 546s and runs out

of memory on q_1, q_7–q_9; Rapid rewrites q_6 in 247s, q_9 in 90.2s and runs out of memory on q_7, q_8 (but is quite fast in all other cases); Requiem takes 232s on r_3, 236s on q_4, 64.7s on q_5 and does not terminate in 600s on q_1, q_3, q_6–q_8.

On the other hand, *Ontop* and the datalog-based systems produce rewritings very quickly: Rapid needs < 0.1s for all CQs except q_8 (0.41s) and q_9 (10.8s); it was impossible to extract the rewriting time in Clipper, but each query was processed in < 2s; *Ontop* computes the rewritings in < 0.025s (except the last two that take < 0.055s). Clearly, the huge UCQs could be produced in fractions of a second by simply unfolding a few CQs by means of the taxonomy. Interestingly, Clipper and Rapid return single-CQ rewritings in the cases without tree witnesses, but generate more CQs than *Ontop* (e.g., q_8 and q_9) otherwise.

3 $\mathcal{T}$-mappings: Concept and Role Hierarchies

We compare the query execution time in *Ontop*, Stardog 1.2 [8] and OWLIM [1]. Both Stardog and OWLIM use internal data structures to store data in the form of triples. Stardog is based on rewriting into UCQs (as we saw in Section 2, such systems can run out of memory during the rewriting stage, even before accessing data). OWLIM is based on materialising the inferences (forward chaining); but the implemented algorithm is incomplete for OWL 2 QL [1]. In the presented experiments, *Ontop* uses DB2 and MySQL to store the data as triples based on the *semantic index* technique, which chooses concept/role IDs in such a way that the resulting $\mathcal{T}$-mapping uses SQL interval expressions (e.g., (ID>=1) AND (ID<=10)) rather than UNIONs to obtain all instances of a concept including its subconcepts (and similarly for roles). It was impossible to compare *Ontop* with other systems: Rapid and IQAROS are just query rewriting algorithms; the publicly available Clipper v 0.1 supports only Datalog engines that read queries and triples at the same time, which would be a serious disadvantage for large datasets. All experiments were run on an HP Proliant with 144 cores 3.47GHz in 24 Intel Xeon CPUs, 106GB RAM and a 1TB@15000rpm HD under 64-bit Ubuntu 11.04 with Java 7, MySQL 5.6 and DB2 10.1.

We took $\text{LUBM}^{\exists}_{20}$ with the data created by the modified LUBM data generator [6] for 50, 100 and 200 universities (with 5% incompleteness) with 7m, 14m and 29m triples, respectively. OWLIM requires a considerable amount of time for loading and materialising the inferences—14min, 33min and 1h 23min, respectively—producing about 93% additional triples and resulting in 13m, 26m and 52m triples (neither Stardog nor *Ontop* need an expensive loading stage). The results of executing the queries from Section 2 are presented in Table 1. We first note that Stardog runs out of memory on 50% of queries, with a likely cause being the query rewriting algorithm, which is an improved version of Requiem (cf. the results in Section 2). On the remaining queries, however, Stardog is fast, which might be due to its optimised triple store. In contrast to Stardog, both OWLIM and *Ontop* return answers to all queries (although the former is incomplete) and their performance is comparable: in fact, in 76% of the cases *Ontop* with DB2 outperforms OWLIM (the lines 'DB2 *' give the execution times for

		r_1	r_2	r_3	r_4	r_5	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
50 universities															
Ontop	DB2 fetch	0	0.02	0.76	0.04	0	1.38	4.16	0	1.50	1.77	1.24	1.83	1.47	0
Ontop	DB2 *	**0**	**0.11**	**0.93**	**0.03**	**0**	125.0	0.86	**0**	0.39	**2.13**	**0.24**	**0.21**	0.48	**0**
Ontop	MySQL	**0**	1.53	12.40	1.57	**0**	681.6	6.91	**0**	1.76	5.52	1.26	8.88	2.69	**0**
OWLIM		0.00	1.85	2.81	0.58	0.16	**20.64**	2.83	0.24	**0.34**	6.16	0.87	0.26	**0.27**	0.04
Stardog		0.01	0.79	1.56	0.34	0.10	DNF	**0.10**	DNF	DNF	DNF	DNF	DNF	DNF	0.04
result size		–	102k	12k	34k	–	1.1m	–	–	–	302k	–	–	–	–
100 universities															
Ontop	DB2 fetch	0	0.02	0.81	0.05	0	1.78	5.06	0	2.15	3.41	1.36	1.85	2.11	0
Ontop	DB2 *	**0**	**0.28**	**1.38**	**0.29**	**0**	131.0	5.15	**0**	2.24	**3.98**	**1.09**	1.26	2.10	**0**
Ontop	MySQL	**0**	2.98	24.89	2.90	**0**	1445	12.63	**0**	3.37	11.08	2.52	17.71	5.19	**0**
OWLIM		0.00	3.72	5.51	1.20	0.38	**41.12**	5.82	0.49	**0.67**	12.92	1.83	**0.51**	**0.55**	0.04
Stardog		0.01	1.78	2.56	0.60	0.38	DNF	**0.20**	DNF	DNF	DNF	DNF	DNF	DNF	0.03
result size		–	205k	24k	69k	–	2.4m	–	–	–	607k	–	–	–	–
200 universities															
Ontop	DB2 fetch	0	0.02	0.97	0.05	0	2.05	6.37	0	3.97	6.84	1.47	1.86	5.97	0
Ontop	DB2 *	**0**	**0.34**	**1.79**	**0.37**	**0**	157.0	6.26	**0**	**3.79**	**7.94**	**1.20**	1.26	5.83	**0**
Ontop	MySQL	**0**	5.73	58.24	7.78	**0**	2888	27.68	**0**	6.66	20.96	4.35	36.21	11.16	**0**
OWLIM		0.00	8.49	11.85	2.73	0.64	**95.87**	11.37	1.03	17.64	29.61	3.60	**1.01**	**1.00**	0.04
Stardog		0.01	3.27	2.92	1.12	0.27	DNF	**0.33**	DNF	DNF	DNF	DNF	DNF	DNF	0.06
result size		–	410k	48k	137k	–	4.7m	–	–	–	1.2m	–	–	–	–

Table 1. Query execution time (in seconds) and the result size over $\text{LUBM}^{\exists}_{20}$.

queries that return the size of the result). Moreover, some of the slowest queries return enormous results (e.g., 4.7m tuples for q_1 over 200 universities). Such queries are hardly typical for databases, and both DB2 and MySQL show a significant degradation in performance. However, as can be seen from the lines 'DB2 fetch' that give the time required to plan the query and fetch the first 500 answers (which does not depend much on the size of the result), DB2 is very fast. It is to be emphasised that *Ontop* can work with a variety of database engines and that, as these experiments demonstrate, *Ontop* with MySQL is considerably worse in executing queries than with DB2 (but is still competitive with OWLIM). Finally, observe that some queries do not need evaluation because *Ontop* simplifies them to empty queries: in fact, r_1, r_5, q_3, q_6 contain atoms that have no instances in the generated data and only 5 out of the 14 CQs return any answers (which probably reflects the artificial nature of the benchmark).

These experiments confirm once again that rewritings into UCQs over arbitrary ABoxes can be prohibitively large even for high-performance triple stores such as Stardog. The materialisation approach should obviously cope with large taxonomies. However, the semantic index used in *Ontop* is able to cope with this problem as well as (and often better than) inference materialisation, but does not incur its considerable extra costs.

We have also evaluated the performance of $\mathcal{T}$-mappings when answering queries over the Movie Ontology (MO, `www.movieontology.org`) and the data

from the SQL version of the Internet Movie Database (IMDb, `www.imdb.com/interfaces`). The reader can find all the results at `tinyurl.com/ontop-benchmark`. Those experiments demonstrate that on-the-fly inference over real databases by means of the tree-witness rewriting and $\mathcal{T}$-mappings is efficient enough to compete (and often outperform) materialisation-based techniques. Moreover, the usual problems associated with approaches based on query rewriting do not affect *Ontop*: $\mathcal{T}$-mappings efficiently deal with hierarchical reasoning, avoiding the exponential blowup, which is usually associated with query rewriting, and the SQO is able to improve performance of the produced SQL queries by taking account of the structure and integrity constraints of the database.

4 Conclusions

To conclude, this paper shows that—despite the negative theoretical results on the worst-case query rewriting and sometimes disappointing experiences of the first OBDA systems—high-performance OBDA is achievable in practice when applied to standard ontologies, queries and data stored in relational databases. In such cases, query rewriting together with SQO and SQL optimisations is fast, efficient and produces SQL queries of high quality whose performance makes materialisation of inferences unnecessary in the OWL 2 QL setting.

Acknowledgements. We thank Giorgio Orsi for providing Nyaya, Guohui Xiao for providing queries and the *Ontop* development team (Josef Hardi, Timea Bagosi and Mindaugas Slusnys) for their help with the experiments.

References

1. B. Bishop and S. Bojanov. Implementing OWL 2 RL and OWL 2 QL rule-sets for OWLIM. In *Proc. of OWLED*, 2011.
2. A. Chortaras, D. Trivela, and G. Stamou. Optimized query rewriting for OWL 2 QL. In *Proc. of CADE-23*, volume 6803 of *LNCS*, pages 192–206. Springer, 2011.
3. T. Eiter, M. Ortiz, M. Šimkus, T.-K. Tran, and G. Xiao. Query rewriting for Horn-SHIQ plus rules. In *Proc. of AAAI 2012*. AAAI Press, 2012.
4. G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proc. of ICDE 2011*, pages 2–13. IEEE Computer Society, 2011.
5. S. Kikot, R. Kontchakov, and M. Zakharyaschev. Conjunctive query answering with OWL 2 QL. In *Proc. of KR 2012*. AAAI Press, 2012.
6. C. Lutz, İ. Seylan, D. Toman, and F. Wolter. The combined approach to OBDA: Taming role hierarchies using filters. In *Proc. of SSWS+HPCSW*, 2012.
7. H. Pérez-Urbina, B. Motik, and I. Horrocks. A comparison of query rewriting techniques for DL-lite. In *Proc. of DL 2009*, volume 477 of *CEUR-WS*, 2009.
8. H. Pérez-Urbina, E. Rodríguez-Díaz, M. Grove, G. Konstantinidis, and E. Sirin. Evaluation of query rewriting approaches for OWL 2. In *SSWS+HPCSW*, 2012.
9. M. Rodríguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work. In *Proc. of AMW 2011*, volume 749. CEUR-WS.org, 2011.
10. T. Venetis, G. Stoilos, and G. Stamou. Query extensions and incremental query rewriting for OWL 2 QL ontologies. *Journal on Data Semantics*, 2013.
11. G. Xiao. Personal communication, 2013.