YARR!: Yet Another Rewriting Reasoner

Joerg Schoenfisch and Jens Ortmann

Softplant GmbH, Agnes-Pockels-Bogen 1, 80992 Munich, Germany
{joerg.schoenfisch, jens.ortmann}@softplant.de

Abstract. In this paper we present our implementation of an OWL 2 QL reasoner using query rewriting to answer SPARQL queries in a relational database. To answer queries in the database through rewriting, ontologies are limited to the OWL 2 QL profile. The rewriting algorithm internally produces a non-recursive Datalog program from the given SPARQL query. This program is then translated to SQL and executed by the database.

Keywords: reasoning, query rewriting, Presto, OWL 2 QL

1 Introduction

In this paper we present YARR, our implementation of a reasoner using query rewriting which is part of the Living Semantic Platform [10]. The platform consists of a relational database as storage for ontologies, an importer for OWL 2 QL[1] and RDF[2] data, a web-based GUI to edit the knowledge base, and our reasoner. YARR exposes a SPARQL[3] endpoint and answers queries through the database.

To enable processing of semantic queries by a relational database some restrictions apply and several steps are performed. The ontologies stored in the database are limited to the expressiveness of the OWL 2 QL profile which is specifically designed to facilitate query answering through query rewriting and processing in a relational database.

Query rewriting is employed to retrieve complete and sound answers from the database. The rewriting which incorporates knowledge from the TBox into the query to also enable extraction of implicit knowledge from the ABox takes place in three steps. First, the SPARQL query is parsed and translated to Datalog. Second, this query is then rewritten into a non-recursive Datalog program to include the knowledge from the TBox. Third, the program is translated to SQL and executed in the database system.

YARR fully supports ontologies formulated in OWL 2 QL and most of the SPARQL 1.1 SELECT syntax. Some built-ins and complex mathematical expressions in FILTER and ORDER BY clauses still have to be implemented. Furthermore, there is no reasoning on datatypes.

[1] http://www.w3.org/TR/owl2-profiles/
[2] http://www.w3.org/TR/rdf-primer/
[3] http://www.w3.org/TR/sparql11-overview/

2 Related Work

The OWL 2 QL profile is based on the description logic DL-Lite [1]. Calvanese et al. [1] proposed PerfectRef as one of the first rewriting algorithms. A second, popular rewriting algorithm is REQUIEM [6], of which an optimized version called Blackout [7] is used in Stardog[4]. Pérez-Urbina et al. also provide an overview over other rewriting algorithms, for instance Nyaya [5], Clipper [4], Rapid [3], Presto [9] and Prexto [8].

YARR is based on the Presto algorithm, which produces a non-recursive Datalog program.

3 Query Rewriting

Query rewriting is the process of transforming a query over a knowledge base in such a way that it produces sound and complete answers without the need for any reasoning in the knowledge base itself. In the case of OWL 2 QL and similar description logics this means that the query is expanded with knowledge from the TBox so that implicit knowledge can be extracted from the ABox without the need for any information about the ABox itself. This is achieved by limiting the expressiveness of the description logic. A thorough overview is given in the description of the DL-Lite family by Calvanese et al. [2].

An opposing approach to this is materialization. Here, all knowledge that can be inferred from known facts is explicitly stored when the data is loaded. Thus, no reasoning is needed later on, as long as the data is not modified. If the data is modified the materialization has to be computed anew which can be quite expensive, e.g. when removing a subclass-of relation, the class assertion with the super class has to be removed from every single instance of the subclass.

There are three reasons why we chose query rewriting in our implementation. First, rewriting allows the query to be processed by regular RDBMS, which are readily available in enterprise environments and require well-known effort concerning administration, maintenance, backup, etc.

The omission of a reasoning step in the ABox during query answering is the second point in favor of query rewriting. This way, changes in the ABox, which might happen quite frequently due to the collaborative setting in which we want to deploy YARR, can directly be reflected in the answers to a query.

Third, in an ontology-based data access scenario it is often not feasible or possible to modify or expand the ABox due to its size or missing write permissions.

Presto rewrites a Datalog query in three major steps. The first step splits each Datalog rule into its independent join components. This produces more, smaller Datalog rules, resulting in smaller rewritings in the end. This is beneficial as every Datalog rule is later translated to SQL and must be processed by the RDBMS. Thus, smaller rewritings positively affect the speed of query answering.

[4] http://stardog.com/

The second step removes redundant rules. These rules would produce answers other rules already did, so there is no need to rewrite, translate and execute them. An example would be two rules, one asking for all individuals, and the other asking for individuals of a specific type. The individuals of a specific type are already included in all individuals, and thus the specific rule is superfluous.

The last step defines TBox views for each concept and role, e.g. a view for a concept would be the union of all individuals directly of this type or of the type of one of its subclasses.

As an example, the simple SPARQL query that selects all triples is translated to a single Datalog rule with one atom, and results in an SQL query consisting of a union over 3 sub-selects with 3 joins each. If the rewriting step is left out, the size of the SQL grows linearly with the number of triple patterns in the SPARQL query.

4 Architecture

The two major parts of YARR's architecture are on the one hand the steps and libraries involved to transform and rewrite a query from SPARQL to the corresponding SQL, and the database architecture on the other hand.

Other parts include the import and export functionality for which we utilize the OWL API[5] to parse and write OWL documents, and consistency checking which is currently transferred to Jena[6]. Jena is the only reasoner freely available for commercial use. However, it does not officially support OWL 2 as of yet and its performance is behind that of other reasoners like Pellet or HermiT.

4.1 Rewriting Architecture

The rewriting takes place in several steps, along which our architecture is split. We try to use existing libraries as much as possible for the steps not directly related to the rewriting.

The first step is the parsing of the SPARQL query and its translation to our internal Datalog model. We are using the parser implementation of Sesame[7]. Their parser produces an abstract model of the query which we translate to Datalog rules. The rules incorporate some additional information, e.g. aggregate functions or mathematical expressions for FILTER clauses, which are not directly represented in Datalog.

These rules are then passed on to the Presto algorithm, which is a straightforward implementation of the rewriting procedure described by Rosati et al. [9]. The resulting Datalog program includes all the knowledge needed to produce sound and complete answers to the query.

The process is finished after the final translation from Datalog to SQL. To gain some syntax and type checking, and to be as agnostic to the underlying

[5] http://owlapi.sourceforge.net/
[6] http://jena.apache.org/
[7] http://www.openrdf.org/

database systems as possible, we us jOOQ[8], a domain specific language for SQL in Java. Each rule of the Datalog program is translated to an SQL query. The queries are then aggregated into a single query as subqueries. This SQL query can then be passed to the database to retrieve the answers.

4.2 Database Design

Our database design is loosely based on the way in which facts are stated in OWL. Overall, it consists of 22 tables. There are individual tables for each type of assertions, for subclass and subproperty axioms, disjointness and equivalency axioms, range and domain axioms, literals, and one table for all types of entities (classes, properties, and individuals).

Another quite obvious choice would have been a design which follows the structure of RDF. There would have been one (or to optimize for performance several partitioned) table(s) storing only triples. In fact, this is the layout several triple stores, like Sesame, Jena, or OWLIM, chose. However, as our reasoner is part of a platform that focuses on OWL semantics and also offers an editor, historization, and versioning, we opted for the initially more complex layout to ease these other tasks.

5 Expected Performance

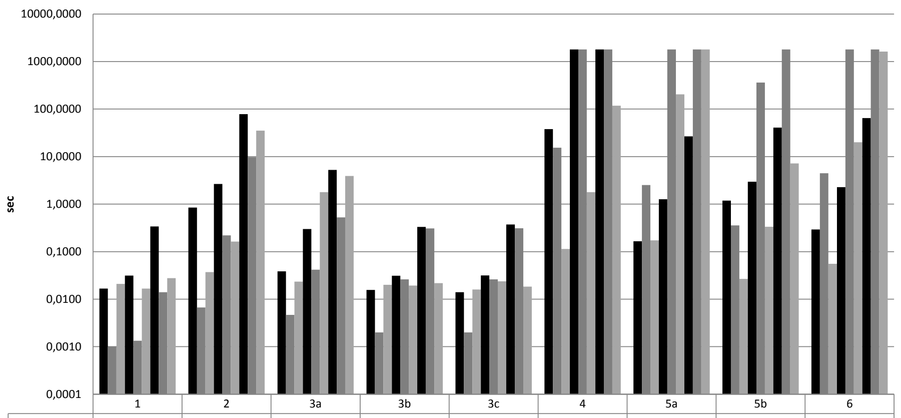
We conducted a benchmark to compare YARR to state-of-the-art triple stores. As benchmark we chose SP2Bench[9] and various sizes of the data. The other stores we used for comparison are OWLIM-Lite 5.2[10] and Stardog 1.0.7 Community Edition. OWLIM uses a materialization approach, which means all inferences are computed and stored before a query is processed. Stardog uses a query rewriting algorithm similar to our approach.

Figures 1 and 2 show a comparison to OWLIM and Stardog for different sizes of SP2Bench (10k, 250k, and 5M triples). This benchmark was run on a desktop machine with a Intel Core 2 Duo at 3 GHz and 8GB RAM. As database backend for YARR we used Oracle 11g Express Edition[11]. Note that to make for a fair comparison the times include the time needed to send the results to the client.

Most of the time that YARR needs for query answering is spent by the RDBMS for query planning, processing and result serialization. The overhead of the rewriting step, and the translation from SPARQL to Datalog and SQL is negligible for larger datasets and complex queries (well below 50ms).

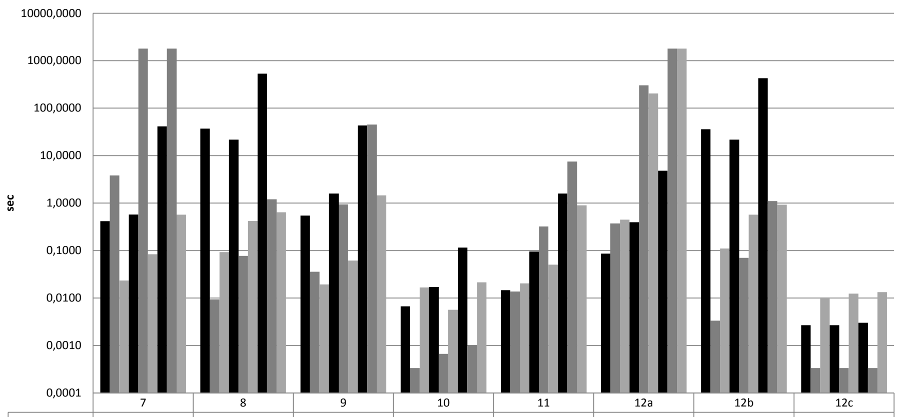
The diagram clearly shows that our implementation behaves similarly in terms of scalability as OWLIM and Stardog. We have some disadvantages for very fast queries due to the rewriting step and the translation to SQL, and the round-trip to the database (Queries 1, 12c). Although the database is hosted on

[8] http://www.jooq.org/
[9] http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B
[10] http://www.ontotext.com/owlim
[11] http://www.oracle.com/technetwork/products/express-edition/overview/



	1	2	3a	3b	3c	4	5a	5b	6
■ 10k LSP	0,0167	0,8453	0,0383	0,0157	0,0140	37,8023	0,1647	1,1867	0,2933
■ 10k OWLIM	0,0010	0,0067	0,0047	0,0020	0,0020	15,3383	2,5103	0,3570	4,4750
■ 10k Stardog	0,0210	0,0370	0,0233	0,0200	0,0160	0,1137	0,1723	0,0267	0,0557
■ 250K LSP	0,0313	2,6503	0,3000	0,0310	0,0317	1800,0000	1,2663	2,9617	2,2693
■ 250k OWLIM	0,0013	0,2200	0,0417	0,0260	0,0260	1800,0000	1800,0000	360,8620	1800,0000
■ 250k Stardog	0,0167	0,1630	1,7893	0,0193	0,0237	1,7873	203,2773	0,3327	20,0597
■ 5M LSP	0,3380	78,3443	5,2533	0,3317	0,3733	1800,0000	26,6530	40,6263	64,5100
■ 5M OWLIM	0,0140	9,6743	0,5257	0,3077	0,3097	1800,0000	1800,0000	1800,0000	1800,0000
■ 5m Stardog	0,0277	35,1750	3,9043	0,0217	0,0183	117,5650	1800,0000	7,2007	1617,1033

Fig. 1. Benchmark SP2Bench Queries 1 - 6



	7	8	9	10	11	12a	12b	12c
■ 10k LSP	0,4143	36,9597	0,5450	0,0067	0,0147	0,0863	35,9443	0,0027
■ 10k OWLIM	3,8273	0,0093	0,0357	0,0003	0,0137	0,3727	0,0033	0,0003
■ 10k Stardog	0,0233	0,0927	0,0193	0,0167	0,0203	0,4457	0,1103	0,0100
■ 250K LSP	0,5723	21,6380	1,5933	0,0170	0,0957	0,3947	21,7037	0,0027
■ 250k OWLIM	1800,0000	0,0773	0,9390	0,0007	0,3207	302,2110	0,0703	0,0003
■ 250k Stardog	0,0833	0,4183	0,0613	0,0057	0,0503	204,4030	0,5690	0,0123
■ 5M LSP	41,4237	533,6507	43,1820	0,1153	1,5877	4,7917	427,6407	0,0030
■ 5M OWLIM	1800,0000	1,1983	45,1623	0,0010	7,5273	1800,0000	1,1003	0,0003
■ 5m Stardog	0,5690	0,6397	1,4493	0,0213	0,8983	1800,0000	0,9240	0,0133

Fig. 2. Benchmark SP2Bench Queries 7 - 12

the same machine it is running in a different process outside the JVM and has to be accessed over the network which is considerably slower than to access a database in the same process.

However, more importantly, we are on par with OWLIM's performance for some queries on the larger datasets (Queries 3b, 3c, 9) and sometimes considerably faster (Queries 5a, 5b, 6, 7, 11, 12a). For six of these OWLIM is not able to complete the request within the 30 minutes timeout that SP2Bench imposes on reasoners, whereas this only happens for one query in YARR (Query 4).

The comparison to Stardog shows similar results. The overall impression is that Stardog performs slightly better than the other two. It is especially fast on query 4, where it is the only reasoner without a timeout, and queries 7 and 9, but also seems to have some penalty if the result of the query can be computed very fast (Queries 1, 10, 12c).

Further investigation is needed to determine the source for YARRs slow performance on the remaining queries (Queries 2, 3a, 8, 10, 12b). Possible reasons are bad query plans, missing indexes, or queries inherently hard for relational database systems.

We also did benchmarks for different database backends, i.e. Oracle, Postgres[12], HSQLDB[13] and H2[14]. Oracle and Postgres showed comparable performance for all benchmarks. The tests on HSQLDB and H2 were only done in-memory and for small datasets, so the performance values we have for those are not conclusive, yet. However, it was quite surprising that for some queries, H2 was not able to find a query plan for the SQL and did not produce any results.

6 Future Work and Conclusion

We presented our implementation of an OWL 2 QL reasoner using the Presto algorithm to answer SPARQL queries in a relational database. The first benchmarks we conducted to compare it to state-of-the-art triple stores are promising. We still have planned further optimization but we already expect our reasoner to compete well with other implementations.

Our future work is focused on a more thorough support of SPARQL 1.1, mainly for SELECT, ASK and CONSTRUCT queries. Built-ins defined by the recommendation will also be implemented as the need for them arises.

Furthermore, we have planned several optimizations, e.g. caching of TBox statistics to reduce the size of the SQL queries or to improve the join order.

In a wider perspective there is ongoing work to implement adapters for ontology-based data access (OBDA) to support arbitrary database schemes. One possibility herein is the use of R2RML[15], which provides a mapping language from relational schemas to RDF.

[12] http://www.postgresql.org/
[13] http://hsqldb.org/
[14] http://www.h2database.com/html/main.html
[15] http://www.w3.org/TR/r2rml/

References

1. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Dl-lite: Tractable description logics for ontologies. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 602. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.
2. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *Journal of Automated reasoning*, 39(3):385–429, 2007.
3. Alexandros Chortaras, Despoina Trivela, and Giorgos Stamou. Optimized query rewriting for owl 2 ql. In *Automated Deduction–CADE-23*, pages 192–206. Springer, 2011.
4. Thomas Eiter, Magdalena Ortiz, M Simkus, Trung-Kien Tran, and Guohui Xiao. Towards practical query answering for horn-shiq. *Description Logics*, 846, 2012.
5. Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Ontological queries: Rewriting and optimization. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 2–13. IEEE, 2011.
6. Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, 8(2):186–209, 2010.
7. Héctor Pérez-Urbina, Edgar Rodrıguez-Dıaz, Michael Grove, George Konstantinidis, and Evren Sirin. Evaluation of query rewriting approaches for owl 2. In *Joint Workshop on Scalable and High-Performance Semantic Web Systems (SSWS+ HPCSW 2012)*, page 32, 2012.
8. Riccardo Rosati. Query rewriting under extensional constraints in dl-lite. In *Proceedings of the international workshop on description logics, DL-2012*, 2012.
9. Riccardo Rosati and Alessandro Almatelli. Improving query answering over dl-lite ontologies. *Proc. of KR*, 2010, 2010.
10. Joerg Schoenfisch, Florian Lautenbacher, Julian Lambertz, and Willy Chen. *Living Semantic Platform.* 10th International Semantic Web Conference - Industry Track, 25 October 2011. Available at http://www.softplant.de/innovation/konferenzteilnahmen.html.