Evaluating and Benchmarking SPARQL Query Containment Solvers

Melisachew Wudage Chekol[1,3], Jérôme Euzenat[1],
Pierre Genevès[2], and Nabil Layaïda[1]

[1] INRIA and LIG
[2] CNRS
[3] LORIA
{melisachew.chekol|firstname.lastname}@inria.fr

Abstract. Query containment is the problem of deciding if the answers to a query are included in those of another query for any queried database. This problem is very important for query optimization purposes. In the SPARQL context, it can be equally useful. This problem has recently been investigated theoretically and some query containment solvers are available. Yet, there were no benchmarks to compare theses systems and foster their improvement. In order to experimentally assess implementation strengths and limitations, we provide a first SPARQL containment test benchmark. It has been designed with respect to both the capabilities of existing solvers and the study of typical queries. Some solvers support optional constructs and cycles, while other solvers support projection, union of conjunctive queries and RDF Schemas. No solver currently supports all these features or OWL entailment regimes. The study of query demographics on DBPedia logs shows that the vast majority of queries are acyclic and a significant part of them uses UNION or projection. We thus test available solvers on their domain of applicability on three different benchmark suites. These experiments show that (i) tested solutions are overall functionally correct, (ii) in spite of its complexity, SPARQL query containment is practicable for acyclic queries, (iii) state-of-the-art solvers are at an early stage both in terms of capability and implementation.

1 Introduction

Since its recommendation by the W3C, SPARQL has come to widespread use in the semantic web. Large datasets are being made available due to the rapid emergence of linked data and queries are executed at remote endpoints. It becomes more important to be able to optimize queries, which requires analyzing them before evaluating them. We concentrate here on query containment analysis since query equivalence and query satisfiability can be reduced to query containment.

The benefits of static analysis are well known for database query optimization and information integration. Indeed, the static analysis of queries may reveal, without trying to evaluate it, that a query will not return any answer. It may also be used for evaluating queries against precompiled views, which requires to know if the answers to the query against the view are the same as the answers to the database itself. The same benefits can apply for SPARQL queries. It is even more important in a distributed setting when federated queries are evaluated at different SPARQL endpoints. Before sending a query to an endpoint, it is useful to know if this query has any chance to receive answers: this saves communication time.

SPARQL static analysis has attracted some attention recently [7, 8, 13]. These studies are supported by sound mathematical analysis and complexity results. Although they provide results for implementing query containment, very few tools are available (to the exception of [13]). Moreover, to our knowledge, no experimental evaluation of SPARQL query containment has been performed. Due to the relatively high complexity of this problem, it is important to know if the proposed solutions can be applied in practice. Moreover, well established benchmarks for query containment would help fostering the development and improvement of solvers.

The overall purpose of this paper is to design a benchmark suite for testing SPARQL query containment and to evaluate the performance of current solvers. For that purpose, we analyze state-of-the-art solver capabilities as well as actual queries being asked on the web. This allows us to identify two classes of solvers addressing different types of problems: some solvers [13] are restricted to conjunctive queries without projections, some other techniques based on query translations into the μ-calculus [8] are so far restricted to acyclic queries without OPTIONAL. The analysis of DBPedia query logs show that more than 90% of correct queries are acyclic, queries being distributed in large sets of DAG and tree queries.

We thus designed three test sets involving: conjunctive queries without projection, union of conjunctive queries with projection and union of conjunctive queries to be analyzed with respect to an RDF Schema. We also provide an evaluation protocol allowing to run the exact same queries through a specific interface and wrapped the three considered tools. We run these tests and observe the capabilities of different solvers in the different categories of tests.

Hence the contribution of this paper is threefold: (*i*) the analysis of the demographics of SPARQL queries actually produced on the web, (*ii*) the design of benchmark suites for evaluating query containment solver capabilities, (*iii*) the evaluation of three different solvers through these benchmarks. The proposed benchmark suites as well as our testing and analysis tools are available on the web and we expect that this will motivate others to produce even better query containment solvers.

Outline: after presenting the foundations of SPARQL and query containment (§2), we discuss related work both from databases and the semantic web (§3). Then, we present the state of the art in SPARQL query containment solvers, before analyzing the query landscape (§4). From this, we design our benchmark suite (§5). Finally, we report evaluation experiments for the three available systems and discuss the results (§6).

2 SPARQL query containment

2.1 SPARQL

SPARQL is a W3C recommended query language [16] based on simple graph patterns. It allows variables to be bound to components in the queried RDF graph. In addition, operators akin to relational joins, unions, left outer joins, selections, and projections can be combined to build more expressive queries. Queries are formed from query patterns which in turn are defined inductively. A triple pattern is an RDF triple that may contain variables. Triple patterns grouped together using operators AND (.), UNION and OPTIONAL form *query patterns*. We do not consider FILTER query patterns here.

Definition 1. *A graph pattern p is inductively defined as follows:*

$$\begin{aligned} p ::= \ & (U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V) \\ & \mid \ p_1 \ . \ p_2 \mid \{p_1\} \ \textit{UNION} \ \{p_2\} \mid p_1 \ \textit{OPTIONAL} \ \{p_2\} \end{aligned}$$

such that U is the set of URI references, B the set of blank nodes, L the set of literals and V a set of variables, all disjoints.

SPARQL queries are built around the classical pattern SELECT U FROM G WHERE P such that FROM identifies an RDF graph G on which the query will be evaluated, WHERE contains a graph pattern P that the query answers should satisfy and SELECT singles out the distinguished variables $U \subseteq V$ in the graph pattern (see Example 1). An answer for such a query is an assignment of values to the distinguished variables such that the graph pattern is satisfied in the graph. Because query containment works independently from the queried graph, we consider such a query q independently from the graph G, as in Example 1.

We denote the set of answers with respect to a graph G as $ANS(q, G)$. We refer the readers to [16] for a formal presentation and semantics of SPARQL. We consider SPARQL under set semantics which is practical in most cases and for which query containment is decidable.

Example 1. Consider the following queries that retrieve student information from a university dataset.

Select the URI and name of all students taking a course in either computer science or mathematics.

```
SELECT ?x ?y
WHERE {                                  Q1
    ?x a :Student . ?x :name ?y .
    ?x :takesCourse ?c .
    { ?c rdf:type :CsCourse . }
    UNION
    { ?c rdf:type :MathCourse . }
}
```

Select the URI and name of all master students taking a course in computer science.

```
SELECT ?x ?y
WHERE {                                  Q2
    ?x a :Student . ?x :name ?y .
    ?x :masterDegreeFrom ?master .
    ?x :takesCourse ?c .
    ?c rdf:type :CsCourse .
}
```

Acyclic SPARQL queries. Following, the tradition from databases (see §3), we consider SPARQL queries as graphs. More specifically, a SPARQL query is represented as a bipartite graph, with two kinds of nodes: triple nodes and term nodes (representing URIs, blank nodes, and literals), as in Figure 1. If this graph contains a cycle going uniquely through variable and triple nodes, then the query is cyclic.

```
SELECT ?x WHERE {
  ?x :married ?y . ?y :knows ?z .
  ?z :knows ?r . ?r :knows ?y .
}
```

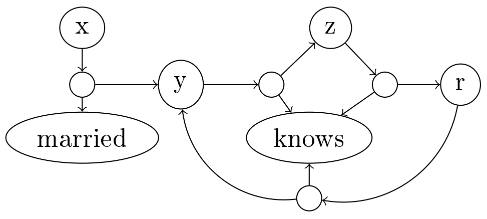

Fig. 1. Cyclic query.

2.2 The query containment problem

The arity of a query is defined as the arity of the query answers. When there is an outer projection it is defined by its distinguished variables, otherwise, it is defined by all free variables of the query. This is analogous to the arity of a predicate in logic.

A query is said to be contained into another if for any RDF graph, its answers are included in those of the other query.

Definition 2 (SPARQL query containment). *Given queries q and q' with the same arity, q is contained in q', denoted $q \sqsubseteq q'$, if and only if $ANS(q, G) \subseteq ANS(q', G)$ for every RDF graph G.*

From the queries in Example 1, $Q2 \sqsubseteq Q1$ and $Q1 \not\sqsubseteq Q2$.

The definitions of SPARQL query answers and of query containment exclude testing containment of query of differing arities. Indeed, it makes little sense in practice to replace a query with another query of a different arity. Projection can be used for making queries comparable.

When the queried graph is assumed to satisfy a particular ontology or schema, it is useful to take this schema into account because two queries may not be contained in each other without the schema, but they may under the schema constraints.

Definition 3 (SPARQL query containment with respect to a schema). *Given a set of RDFS axioms $\mathcal{S}$ and two queries q and q' with the same arity, q is contained in q' with respect to $\mathcal{S}$, denoted $q \sqsubseteq_{\mathcal{S}} q'$, if and only if $ANS(q,G) \subseteq ANS(q',G)$ for every RDF graph G satisfying $\mathcal{S}$.*

3 Related work

In relational databases, query containment is well studied. Notably, [6] proved that containment and equivalence of relational conjunctive queries is NP-complete. Union of conjunctive queries (UCQs) containment is also NP-complete. The complexity reduces to P, if the right-hand side query is acyclic [2, 21].

Beyond databases, the query containment problem has been considered in description logics. [5] demonstrated that containment of $\mathcal{DLR}$ (description logics with n-ary relations) conjunctive queries is double exponential but this complexity reduces to exponential if the query on the right-hand side of the containment has a tree structure.

The containment problem has also been studied for regular path queries — languages that are used to query arbitrary length paths in graph databases. There are different such query languages. A prominent language used in semi-structured data is XPath developed for navigating in XML documents. Static analysis of XPath queries has been studied in [10], encompassing containment, equivalence, coverage, and satisfiability, as well as containment of XPath queries under XML Schema constraints.

Recently, static analysis and optimization of SPARQL queries has attracted attention, notably for static analysis [7, 8, 13] and for optimization [11, 19, 18, 13]. The query containment problem for full SPARQL is undecidable [15, 1]. Hence, it is necessary to reduce SPARQL in order to consider it. A double exponential upper bound has been proven for the containment and equivalence problems of SPARQL queries without OPTIONAL, FILTER and under set semantics [7]. The double exponential upper bound reduces to simple exponential if the right-hand side query is acyclic. In addition, we studied SPARQL query containment under schema constraints, either through entailment regimes [7] or query rewriting [8].

These studies have grounded the theoretical aspect of these fundamental problems, but led to few implementations (see §4). Given the computational complexity of the query containment problem, it is important to evaluate containment test performances in practice. Various SPARQL query evaluation performance benchmarks have been proposed [3, 4, 17, 14]. The DBPSB framework is based on analysing query logs and automatically generating queries which are representative of actual queries [14]. However, no SPARQL query containment benchmark is available to our knowledge.

4 Query containment solvers

We briefly present three state-of-the-art query containment solvers used in the experiments. Our goal is to characterize their capabilities in order to design appropriate benchmarks. In order to do so, we also analyze actual queries used on the semantic web.

Out of the three systems, SPARQL-Algebra is self contained whereas the other two are μ-calculus satisfiability solvers that need an intermediate query translation into formulas to determine containment.

4.1 SPARQL-Algebra

SPARQL-Algebra is an implementation of SPARQL query subsumption and equivalence based on the theoretical results in [13]. This implementation supports AND and OPTIONAL queries with no projection. An on-line version of the solver is available at `http://db.ing.puc.cl/sparql-algebra/`.

4.2 AFMU

AFMU (Alternation Free two-way μ-calculus) [20] is a satisfiability solver for the alternation-free fragment of the μ-calculus [12]. It is a prototype implementation which determines the satisfiability of a μ-calculus formula by producing a yes-or-no answer.

To turn it into a query containment solver, it is necessary to turn the problem into a μ-calculus satisfiability problem.

We developed techniques for encoding queries into the μ-calculus ($\mathcal{A}$) in order to determine the containment of SPARQL queries [7, 8]. Of the three approaches introduced in [7] to deal with RDFS, we have chosen the encoding of the schema (η) into the μ-calculus. We use these encoding schemes for deciding $q \sqsubseteq_{\mathcal{S}} q'$: it is necessary to check if the encoding of its negation, $\eta(\mathcal{S}) \wedge \mathcal{A}(q) \wedge \neg\mathcal{A}(q')$, is satisfiable. If this is the case, then containment does not hold, otherwise, it is established.

4.3 TreeSolver

The XML tree logic solver *TreeSolver*[4] performs static analysis of XPath queries which comprise containment, equivalence and satisfiability. To perform these tasks, the solver translates XPath queries into μ-calculus formulas and then tests the unsatisfiability of the formula. Unlike AFMU, the unsatisfiability test

[4] `http://wam.inrialpes.fr/websolver/`

is performed in time of $2^{\mathcal{O}(n)}$ whereas it is $2^{\mathcal{O}(n \log n)}$ for AFMU, such that n is the size of the formula.

Using TreeSolver follows the same procedure as using AFMU with a slightly different encoding. Indeed, because TreeSolver is restricted to tree-shaped models, we use a specific encoding of query formulas.

4.4 Features supported by solvers

A summary of the features supported by these query containment solvers is presented in Table 1.

Table 1. Comparison of features supported by current systems.

System	projection	UCQ	optional	blanks	cycles	RDFS
SPARQL-Algebra			√		√	
AFMU	√	√		√		√
TreeSolver	√	√		√		√

Part of the query structures can be transformed into concept expressions in description logics and submitted to satisfiability (or subsumption) tests as well. So, in principle, query containment solvers based on description logic reasoners could be designed. However, we do not know any such solver.

4.5 State of the query landscape

To the best of our knowledge, no experimental work has been conducted to verify how many of real world queries are acyclic or cyclic. To answer this question, we analyzed DBpedia query logs[5]. We report on two log sets because there is a lot of variation between them. 2 905 035 queries from the logs were syntactically correct (90%). The results are given in Table 2. We tested the cyclicness of queries and found out that more than 90% of these queries are acyclic (94% of the small sample and 99% on the total). This justifies designing and evaluating acyclic queries. Projection is used in 11% of the large log and 22% of the smaller one, but such figures more than double if only "SELECT *" queries are counted as projection-free queries.

OPTIONAL are used in around 30% of queries, whereas UNION is used in 18% of those in the full log and 43% in the large one. Union of conjunctive queries with optional are 15 to 30% of the logs. This make them operators to be supported in query containment.

[5] DBpedia 3.5.1 logs (`ftp://download.openlinksw.com/support/dbpedia/`) contain 3 210 368 queries between 30/04/2010 and 20/07/2010 and 378 530 queries of 13/07/2010 only.

Table 2. Query characteristics of the full DBPedia logs.

	projection			no projection		
operator	tree	dag	cycle	tree	dag	cycle
none	175 220	562	1	1 534 150	1 761	1 748
union	9	26 625	547	24	29 629	1 166
opt	2 052	685	0	311 608	722	1
filter	7 912	711	6	264 821	340	1
un-opt	0	306	0	0	12 659	1
opt-filt	7 991	779	0	4 933	52 401	0
filt-un	2	183	0	23 802	12 286	0
un-opt-filt	0	102 765	0	0	302 657	23 969

5 Benchmark design

We first present the design of containment benchmark suites. Each test suite is made of elementary test cases asking for the containment of one query into another. We then introduce the principles and software used for evaluating containment solvers. The benchmark and software is available on-line at `http://sparql-qc-bench.inrialpes.fr/`.

5.1 Structure of the benchmark

There are three qualitative dimensions along which tests can be designed: the type of graph pattern connectors (AND, UNION, MINUS, Projection, OPTIONAL, FILTER etc.), the type of ontology: (no schema, RDFS, SHI, OWL, etc.) and the query structure (tree, DAG, cyclic). In addition to these dimensions, quantitative measures are:

- the number of triple patterns,
- the number of variables,
- the number of triple patterns involving more than one variable (Tjoins),
- the size of the ontology.

We designed test suites of homogeneous qualitative dimensions selected with respect to the capacity of the current state-of-the-art solvers. The benchmark contains three test suites:

- **C**onjunctive **Q**ueries with **No Proj**ection (CQNoProj)
- **U**nion of **C**onjunctive **Q**ueries with **Proj**ection (UCQProj)
- **U**nion of **C**onjunctive **Q**ueries under **RDFS** reasoning (UCQrdfs)

The test suites are designed to model increasing expected difficulty by using more constructors. We did not provide tests of cyclic queries since only one solver is

Table 3. The CQNoProj testsuite. In the AND column, figures correspond to the number of AND in the left-hand side query of the test. Vars is the number of variables in each queries and Tjoin the number of triples in which occurs at least two variables.

Test case	Problem	AND	Vars	Tjoin	Test case	Problem	AND	Vars	Tjoin
nop1	Q1a $\sqsubseteq$ Q1b	1	1	0	nop11	Q6a $\sqsubseteq$ Q6c	2	3	1
nop2	Q1b $\sqsubseteq$ Q1a	0		0	nop12	Q6c $\sqsubseteq$ Q6a	0		1
nop3	Q2a $\sqsubseteq$ Q2b	5	3	3	nop13	Q6b $\sqsubseteq$ Q6c	2	3	1
nop4	Q2b $\sqsubseteq$ Q2a	5		3	nop14	Q6c $\sqsubseteq$ Q6b	0		1
nop5	Q3a $\sqsubseteq$ Q3b	2	2	2	nop15	Q7a $\sqsubseteq$ Q7b	9	10	9
nop6	Q3b $\sqsubseteq$ Q3a	1		1	nop16	Q7b $\sqsubseteq$ Q7a	10		9
nop7	Q4c $\sqsubseteq$ Q4b	5	3	2	nop17	Q8a $\sqsubseteq$ Q8b	3	4	3
nop8	Q4b $\sqsubseteq$ Q4c	3		2	nop18	Q8b $\sqsubseteq$ Q8a	2		3
nop9	Q6a $\sqsubseteq$ Q6b	2	3	1	nop19	Q9a $\sqsubseteq$ Q9b	4	3	2
nop10	Q6b $\sqsubseteq$ Q6a	2		1	nop20	Q9b $\sqsubseteq$ Q9a	4		2

currently able to deal with them. However, this would be a natural addition to these tests.

Each test suite contains tests of different quantitative measures. Most of them are used for conformance testing, i.e., testing that solvers return the correct answer, but we also identify some stress tests trying to evaluate solvers at or beyond their limits. We discuss these test suites below.

CQNoProj This test suite is designed for containment of basic graph patterns. It contains conjunctive queries with no projection. We have identified 20 different test cases (nop1–nop20), each one testing containment between two queries. All the cases in this setting are shown in Table 3, along with the number of connectives and variables in the queries. The more difficult test used for stress testing are nop3, nop4, nop15, and nop16. The two former ones have a larger number of conjunction (and of Tjoin), while the two latter ones have an even larger number of conjunctions and variables. We have selected Tjoins (triples having two variables) as a measure of difficulty because simple triple joins may be compiled efficiently as tuples.

The worse case complexity of CQ without projection containment is PTime [9].

UCQProj This test suite is made of 28 test cases, each comprising two acyclic union of conjunctive queries with projection. In fact, 14 tests contain projection only, 6 tests contains union only and 2 tests contains both (see Table 4). The test cases differ in the number of distinguished variables (*Dvars*) and connectives (conjunction or union). Particular stress tests are p3, p4 (without union nor projection), p15, p16, p23, and p24.

The worse case complexity of UCQ with projection containment is NP-complete with cycles [6], it is unknown for acyclic queries.

Table 4. The UCQProj test suite.

Test case	Problem	AND	UNION	Dvars	Vars	Tjoin
p1	Q11a $\sqsubseteq$ Q11b	1	0	1	1	0
p2	Q11b $\sqsubseteq$ Q11a	0	0		1	0
p3	Q12a $\sqsubseteq$ Q12b	5	0	3	3	3
p4	Q12b $\sqsubseteq$ Q12a	5	0		3	3
p5	Q13a $\sqsubseteq$ Q13b	2	0	2	2	2
p6	Q13b $\sqsubseteq$ Q13a	1	0		2	1
p7	Q14c $\sqsubseteq$ Q14b	3	0	1	3	2
p8	Q14b $\sqsubseteq$ Q14c	5	0		3	2*
p9	Q15a $\sqsubseteq$ Q15b	0	0	2	3	1
p10	Q15b $\sqsubseteq$ Q15a	0	0		2	1
p11	Q16a $\sqsubseteq$ Q16b	2	0	1	3	1
p12	Q16b $\sqsubseteq$ Q16a	2	0		3	1
p13	Q16a $\sqsubseteq$ Q16c	2	0	1	3	1
p14	Q16c $\sqsubseteq$ Q16a	0	0		3	1
p15	Q17a $\sqsubseteq$ Q17b	9	0	10	10	9
p16	Q17b $\sqsubseteq$ Q17a	10	0		11	10
p17	Q18a $\sqsubseteq$ Q18b	3	0	4	4	3
p18	Q18b $\sqsubseteq$ Q18a	2	0		4	3
p19	Q19a $\sqsubseteq$ Q19b	4	0	2	3	2
p20	Q19b $\sqsubseteq$ Q19a	4	0		3	2
p21	Q19c $\sqsubseteq$ Q19b	4	0	2	4	3
p22	Q19b $\sqsubseteq$ Q19c	4	0		3	2
p23	Q20a $\sqsubseteq$ Q20b	2	7	10	10	9
p24	Q20b $\sqsubseteq$ Q20a	8	1		10	9
p25	Q21a $\sqsubseteq$ Q21b	6	2	2	4-6	5
p26	Q21b $\sqsubseteq$ Q21a	8	0		6	5
p27	Q22a $\sqsubseteq$ Q22b	3	1	2	2	2
p28	Q22b $\sqsubseteq$ Q22a	3	1		2	2

UCQrdfs In query containment under RDFS reasoning, there are 28 test cases (Table 5). In comparison to the test cases in UCQProj and CQNoProj setting, the size of the queries is small. Each test case is composed of two acyclic UCQs and a schema. There are 4 different small schemas, C1–C4 whose characteristics, with respect to the type and number of axioms, are presented in Table 5. These small schemas are used for testing the correctness of solvers and are not realistic schemas. The most difficult tests are supposed to be rdfs23, rdfs24, rdfs25, rdfs26, rdfs27 and rdfs28 with both projection and union.

The worse case complexity of UCQ under RDFS has an ExpTime upper bound [7].

5.2 Benchmarking software architecture

For testing containment solvers, we designed an experimental setup which comprises several software components. This setup is illustrated in Figure 2. It simply considers a containment checker as a software module taking as input two

Table 5. The UCQrdfs test suite.

Schema	Axiom types
C1	subclass (2)
C2	domain (1) and range (1)
C3	subclass (1), subproperty (2) and domain (1)
C4	subclass (1)

Test	Ontology	Problem	AND	UNION	Dvars	Vars	Tjoin
rdfs1	C1	Q39a $\sqsubseteq$ Q39c	0	0	1	1	0
rdfs2		Q39c $\sqsubseteq$ Q39a	0	1		1	0
rdfs3	C1	Q39a $\sqsubseteq$ Q39b	0	0	1	1	0
rdfs4		Q39b $\sqsubseteq$ Q39a	0	0		1	0
rdfs5	C1	Q39b $\sqsubseteq$ Q39c	0	0	1	1	0
rdfs6		Q39c $\sqsubseteq$ Q39b	0	1		1	0
rdfs7	C1	Q39d $\sqsubseteq$ Q39e	4	0	1	3	2
rdfs8		Q39e $\sqsubseteq$ Q39d	4	0		3	2
rdfs9	C2	Q40b $\sqsubseteq$ Q40d	0	0	1	2	1
rdfs10		Q40d $\sqsubseteq$ Q40b	0	0		1	0
rdfs11	C2	Q40e $\sqsubseteq$ Q40b	1	0	1	2	2
rdfs12		Q40b $\sqsubseteq$ Q40e	0	0		1	0
rdfs13	C3	Q41b $\sqsubseteq$ Q41c	0	0	1	2	1
rdfs14		Q41c $\sqsubseteq$ Q41b	0	0		2	1
rdfs15	C3	Q41b $\sqsubseteq$ Q41d	0	0	1	2	1
rdfs16		Q41d $\sqsubseteq$ Q41b	0	0		2	1
rdfs17	C3	Q41c $\sqsubseteq$ Q41d	0	0	1	2	1
rdfs18		Q41d $\sqsubseteq$ Q41c	0	0		2	1
rdfs19	C3	Q41b $\sqsubseteq$ Q41a	0	0	1	2	1
rdfs20		Q41a $\sqsubseteq$ Q41b	0	0		1	0
rdfs21	C3	Q41e $\sqsubseteq$ Q41a	0	1	1	2	2
rdfs22		Q41a $\sqsubseteq$ Q41e	0	0		1	0
rdfs23	C4	Q43a $\sqsubseteq$ Q43b	3	1	2	2	2
rdfs24		Q43b $\sqsubseteq$ Q43a	3	1		2	2
rdfs25	C4	Q43a $\sqsubseteq$ Q43c	3	1	2	2	2
rdfs26		Q43c $\sqsubseteq$ Q43a	3	1		2	2
rdfs27	C4	Q43b $\sqsubseteq$ Q43c	3	3	2	2	2
rdfs28		Q43c $\sqsubseteq$ Q43b	3	1		2	2

SPARQL queries (q and q'), eventually an RDF Schema ($\mathcal{S}$), and returning true or false depending if q' is entailed by q (under the constraints of $\mathcal{S}$).

This has been provided as a Java interface using Jena to express queries and RDF Schema. We have developed three wrappers implementing this interface for the three tested systems. Other systems may be wrapped in the same interface (dashed rectangles in Figure 2) and tested in the same conditions. This platform may also be used for providing non regression tests for containment solvers.

Tests proceeds by providing test cases to the interface, timing the execution of the containment test around this common interface call. So timing occurs at the frontier of the dashed box of Figure 2, i.e., after query and schema parsing. This advantages SPARQL-Algebra, because it works directly on the ARQ representation, whereas the two other solvers have first to translate the ARQ representation into a μ-calculus formula which is then parsed and transformed in each solver's internal representation.

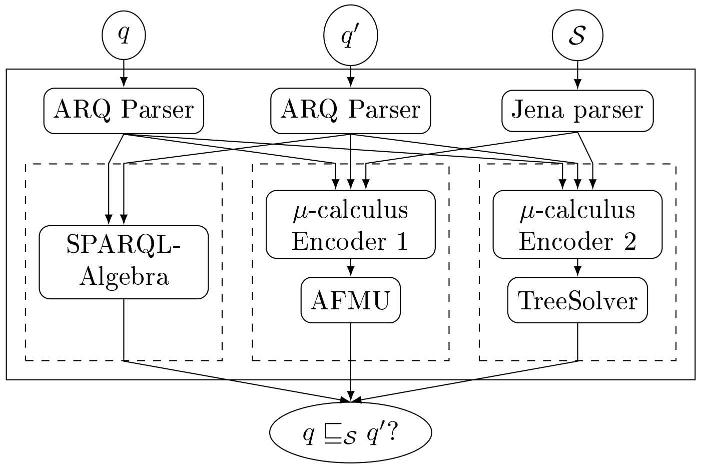

Fig. 2. Experimental setup for testing query containment. The tester (plain rectangle) parses queries and schemas and passes them to a solver wrapper (dashed rectangle).

6 Experiments

We evaluated the three identified query containment solvers with the three test suites. Rather than a definitive assessment of these solvers, our goal is to give first insights into the state-of-the-art and highlight deficiencies of engines based on the benchmark outcome. None of these systems is sharply optimized. However, their behavior is sufficient for highlighting test difficulty.

We run the experiments on a Debian Linux virtual machine configured with four processors and 20GB of RAM running under a Dell PowerEdge T610 with 2*Intel Xeon Quad Core 2.26GHz E5607 processors and 32GB of RAM, under Linux ProxMox 2 (Debian).

The solvers were not genuinely reentrant. Hence, each test case has been run in a separate process after that the first case of each suite has been run as a warm up.

All solvers are Java programs. The Java virtual machines were run with maximum heap size of 2024MB and a timeout at 20s (20000ms). Raising memory size to 1GB and timeout to 40s does not change timeout results. The μ-calculus solvers take advantage of a native BDD library. Using the native implementation doubles the speed of these solvers, however, it also brings large initialization time (in spite of warm-up set up).

Reported figures are the average of 5 runs (we run the tests 7 times and ruled out each time the best and worse performance).

6.1 CQNoProj Results

On the conjunctive queries without projection, the SPARQL Algebra implementation is at least 10 times faster than the μ-calculus implementations (Figure 3).

This comes as no surprise, since the latter are exponential time solvers whereas the former is a polynomial time solver.

AFMU times out on stress tests (nop3, nop4, nop15 and nop16). This happens whenever containment is determined between queries that contain more than 10 joins, such as in test cases nop15 and nop16. TreeSolver is able to deal with such cases albeit at the price of longer response time.

The fact that SPARQL-Algebra does not suffer from these sets, shows that the encoding of the μ-calculus solvers can be improved for such practical cases.

SPARQL-Algebra responded incorrectly, in test case nop7 (cf. Table 3), when blank nodes are used in the queries. It is not expected to deal with blank nodes. The other solvers are able to take them into account.

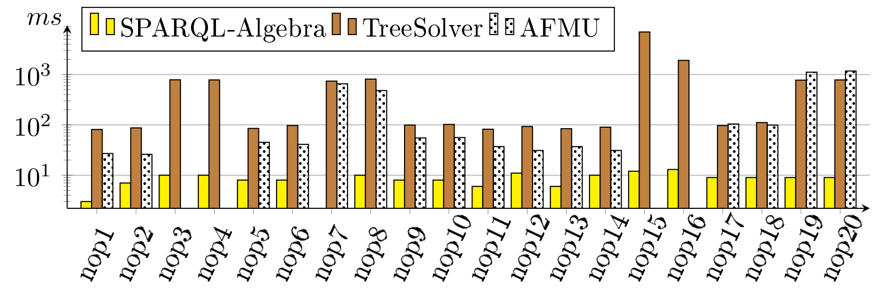

Fig. 3. Results for the CQNoProj test suite (logarithmic scale).

6.2 UCQProj Results

On the UCQProj test suite (see Section 5.1), we compared the two systems able to deal with UNION: TreeSolver and AFMU. Figure 4 shows AFMU is nearly always slightly faster than TreeSolver, but on tests p17 and p18. This is surprising given the difference in complexity of both solvers. However, TreeSolvers times out only on tests p23 and p24, while AFMU cannot deal with all stress tests: p3-p4, p15-p16, p23-p26. The necessary run time tends to be far longer as it often ends up in filling the available heap. For some of these tests (p15-16), this could also be improved by adopting a better encoding of triples.

6.3 UCQrdfs Results

The results for containment of acyclic UCQs under RDFS (cf. Section 5.1) are better. Figure 5 shows that both solvers answer containment queries within a few hundreds of milliseconds. In this tests, AFMU and TreeSolver have contrasted results. AFMU is faster in 15 tests, TreeSolver faster in 9 tests, and there is a tie on 3 tests. Tests rdfs7 and rdfs8 have been difficult for both solvers. Tests rdfs23-28 have been more difficult for TreeSolver than for AFMU. This should not be

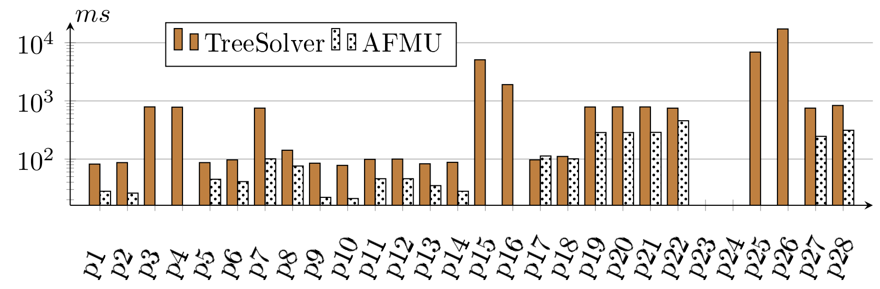

Fig. 4. Results for the UCQProj test suite (logarithmic scale).

due to the C4 ontology which is reduced to only one subsumption assertion, but rather to the presence of UNION and projection. AFMU returns an incorrect answer for rdfs9 which seems to be a bug in the solver.

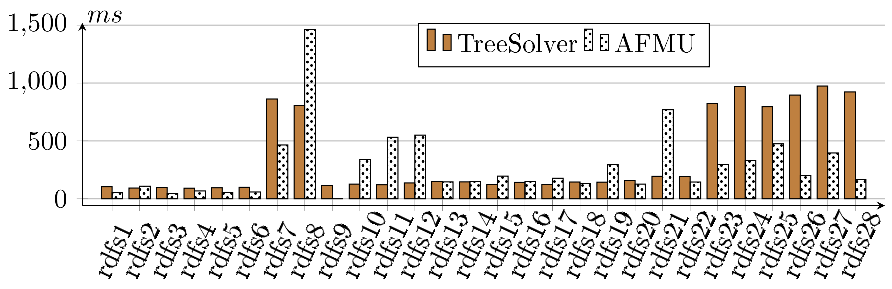

Fig. 5. Results for the UCQrdfs test suite.

6.4 Discussion

In summary, all solvers under all experimental settings responded positively i.e., they all determined containment correctly under their stated application limits (we tested this independently). However, from these experiments, a lot remains to be done in order to alleviate the shortcomings of the current systems.

SPARQL-Algebra is faster on its domain of application. The advantages of this solver compared to the others are that it supports subsumption of OPTIONAL query patterns and also cyclic CQs. However, blank nodes are not supported.

AFMU is able to determine containment of acyclic UCQs under ontological axioms. For queries of reasonable size, the solver determined their containment correctly. The problem is that when queries have a larger size, e.g., more than 8 joins, the solver saturates memory. This is shown for test cases nop15 and nop16 (Figure 3) as well as for test cases p15 and p16 (Figure 4). However, the implementation of this solver is not optimal: the authors have documented improvements. Moreover, determining containment of general UCQs (beyond the acyclic ones) will require extending the solver.

TreeSolver has similar limitations as AFMU: no support for cyclic queries and difficulty with queries of large size, such as nop16 and p16. This is surprising with respect to its worse case complexity.

In addition, determining the type of queries to compare (cyclic, disjunctive, with blank nodes, with projections, etc.) is easy. Hence, it is possible to build a system assembling these solvers and providing the best possible performances for each case.

7 Conclusion

The evaluation of SPARQL query containment should help developers to produce more and better solvers. This paper has contributed to the evaluation of SPARQL query containment in several ways:

- We have studied the demographics of SPARQL queries on a large example and found that (1) a large part of these queries are acyclic, and (2) those parts that either contain projections (effective SELECT) or not, are significant;
- From this study and the state of the art in query containment solvers, we have designed a benchmark suite made of three suites testing conjunctive queries without projection, union of conjunctive queries with projection and with RDFS reasoning;
- We have proposed and implemented a methodology for evaluating this problem;
- Finally, we have applied these to existing containment solvers (SPARQL-Algebra, AFMU and TreeSolver) and we can report the following lessons:
  - All tested solutions perform correctly with respect to their declared applicability limits (which are easily testable);
  - SPARQL query containment can be practically performed, in spite of its complexity, in a reasonable time with respect to network communication costs,
  - the current state-of-the-art is at its early stage and requires improvement and new ways to determine containment and equivalence of queries, in order to become a useful tool for query optimizers.

These benchmark suites are well-suited for pinpointing the theoretical shortcomings of containment solvers. These have already benefited TreeSolver in diagnosing bad use of memory.

We plan to improve and extend this benchmark, in particular by adding other test suites designed for containment of cyclic queries and queries under expressive description logic axioms such as OWL2.

References

1. R. Angles and C. Gutierrez. The expressive power of SPARQL. In *Proc. ISWC*, pages 114–129, 2008.
2. P. Bernstein and D. Chiu. Using semi-joins to solve relational queries. *Journal of the ACM*, 28(1):25–40, 1981.
3. C. Bizer and A. Schultz. Benchmarking the performance of storage systems that expose SPARQL endpoints. In *Proc. 4 th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)*, 2008.
4. C. Bizer and A. Schultz. The Berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems*, 5(2):1–24, 2009.
5. D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive Query Containment and Answering under Description Logics Constraints. *ACM Trans. on Computational Logic*, 9(3):22.1–22.31, 2008.
6. A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90. ACM, 1977.
7. M. W. Chekol, J. Euzenat, P. Genevès, and N. Layaïda. SPARQL query containment under RDFS entailment regime. In *Proc. IJCAR*, pages 134–148, 2012.
8. M. W. Chekol, J. Euzenat, P. Genevès, and N. Layaïda. SPARQL query containment under SHI axioms. In *Proc. AAAI*, pages 10–16, 2012.
9. C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *Proc. ICDT*, pages 56–70, 1997.
10. P. Genevès, N. Layaïda, and A. Schmitt. Efficient static analysis of XML paths and types. In *Proc. PLDI*, pages 342–351. ACM, 2007.
11. J. Groppe, S. Groppe, and J. Kolbaum. Optimization of SPARQL by using coreSPARQL. In *Proc. ICEIS*, pages 107–112, 2009.
12. D. Kozen. Results on the propositional μ-calculus. *Theor. Comp. Sci.*, 27:333–354, 1983.
13. A. Letelier, J. Pérez, R. Pichler, and S. Skritek. Static analysis and optimization of semantic web queries. In *Proc. PODS*, pages 89–100. ACM, 2012.
14. M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngomo Ngonga. DBpedia SPARQL benchmark: performance assessment with real queries on real data. In *Proc. 10th ISWC*, pages 454–469, Bonn (DE), 2011. Springer-Verlag.
15. A. Polleres. From SPARQL to rules (and back). In *Proc. WWW Conference*, pages 787–796, 2007.
16. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation, 2008.
17. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP^{2}Bench: a SPARQL performance benchmark. In *Proc. ICDE*, pages 222–233, 2009.
18. M. Schmidt, M. Meier, and G. Lausen. Foundations of SPARQL query optimization. In *Proc. ICDT*, pages 4–33. ACM, 2010.
19. M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds. SPARQL basic graph pattern optimization using selectivity estimation. In *Proc. WWW Conference*, pages 595–604, 2008.
20. Y. Tanabe, K. Takahashi, M. Yamamoto, A. Tozawa, and M. Hagiya. A decision procedure for the alternation-free two-way modal μ-calculus. In *TABLEAUX*, pages 277–291, 2005.
21. M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. VLDB*, pages 82–94, 1981.