A Benchmark for Semantic Web Query Containment, Equivalence and Satisfiability

Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, Nabil Layaïda

A Benchmark for Semantic Web Query Containment, Equivalence and Satisfiability

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

[1]*INRIA & LIG*

[2]*CNRS*

firstname.lastname@inria.fr

***Abstract*— The problem of SPARQL query containment has recently attracted a lot of attention due to its fundamental purpose in query optimization and information integration. New approaches to this problem, have been put forth, that can be implemented in practice. However, these approaches suffer from various limitations: coverage (size and type of queries), response time (how long it takes to determine containment), and the technique applied to encode the problem. In order to experimentally assess implementation limitations, we designed a benchmark suite offering different experimental settings depending on the type of queries, projection and reasoning (RDFS). We have applied this benchmark to three available systems using different techniques highlighting the strengths and weaknesses of such systems.**

I. Introduction

Since its recommendation by W3C, SPARQL has come to widespread prominence in the semantic web world. Broad and diverse studies are being conducted in order to improve and extend this language. To this end, recently, static analysis of the language has attracted reasonable attention, due to its foundational advantages in query optimization, satisfiability and information integration. This problem has long been observed and extensively studied for relational database query languages. On the other side, in the semantic web, where data is stored sporadically, large datasets are being made available due to the rapid emergence of linked data, queries are executed at remote endpoints, it is obvious that static analysis of SPARQL queries is relevant. Beyond this, in SPARQL 1.1 federated queries are introduced, these are queries that are evaluated at several remote endpoints and the final result is merged at the sender's premises. Thus, checking the satisfiability of these queries before sending them to the remote endpoint saves communication time. Furthermore, often queries are embedded in programs (Java, C++, and other programming languages), static analysis is useful to avoid runtime errors in these settings.

Static analysis covers containment, equivalence and satisfiability. Several studies of these problems have been carried out for different query languages: from conjunctive queries in relational databases, conjunctive queries in description logic, to SPARQL queries most recently. These studies are often supported by sound mathematical theorems and complexity results. However, a shortcoming of these studies is that they have little or no experimental evaluation. To overcome this shortcoming, we test the containment and equivalence of tree-structured SPARQL queries using a prototype implementation.

In general, a query is referred to as cyclic if the graph structure induced from the query is cyclic. It has been long noted that cyclic queries contribute to a further jump in the complexity of containment and equivalence problems. Most notably, Chandra and Merlin 1977 [1] proved that containment and equivalence of relational conjunctive queries is NP-complete. However, this complexity reduces to P if the queries are acyclic [2], [3]. Further, the study in [4] demonstrated that containment of $\mathcal{DLR}$ (description logics with n-ary relations) conjunctive queries is double exponential but this complexity bound reduces to exponential if the query on the right-hand side of the containment has a tree structure. In other words, cycles among the non-distinguished variables contribute to a further jump in containment and equivalence complexity. Meanwhile, we observed that most of the queries currently used are acyclic. Hence, considering acyclic query containment is relevant.

The overall purpose of this paper is designing a benchmark tailored for tree-structured queries and evaluating the performance of the current state-of-the-art using this benchmark. In previous work [5] and [6], we developed techniques for encoding queries and schema axioms into the μ-calculus [7] in order to determine the containment of SPARQL queries in the presence of schema axioms. As such, we have proved a double exponential upper bound for containment and equivalence problems. Interestingly, this bound reduces to exponential if the right-hand side query has a tree-structure. Using this encoding scheme and exponential time satisfiability solvers from [8] and [9], we conduct an experiment to test the containment and equivalence of tree-structured SPARQL queries. In doing so, we modify the encoding in [5], [6] in order to use the solver from [9]. We compare the performance of these two satisfiability solvers, [8] and [9]. Moreover, we also evaluate the performance of the subsumption and equivalence solver from [10]. However, as it is not an exponential time solver like the other two ([8] and [9]), we judge its performance on its own.

a) Outline:: after presenting the foundations of RDF(S) and SPARQL (§**??**), we provide an in depth analysis of the structure of benchmark (§III), followed by a description of the current state-of-the-art of query containment systems (§IV),

using these systems we conduct experiments and explain the results (§V). We finalize this study with a summary concluding remarks (§**??**).

II. Preliminaries

We present the foundations of RDF, schema language $\mathcal{SHI}$, and Tree-structured SPARQL queries.

A. RDF

Resource Description Framework (RDF) is a W3C standard language to create machine-readable content in the semantic web. RDF describes resources using a directed graph with a *subject* node connected to an *object* node and is labelled by a *predicate*. The subject, predicate and object make up a triple and a set of triples form a graph.

Example 1: An RDF graph that models a university domain.

```
Jerome a Professor.
SemanticWeb a Course.
Jerome givesCourse SemanticWeb.
Manuel memberOf (Hadas Exmo).
Asan a Student.
Professor subClassOf Person.
Asan knows Manuel.
givesCourse domain Person.
```

1) RDFS: RDF has a simple schema language called RDF Schema (RDFS) that enables to express concept and property hierarchies as well as typing restrictions. In this work, we consider a benchmark setting containing the containment problem under the presence of core RDFS axioms namely subclass (sc), subproperty (sp), domain (dom), and range (range). Nonetheless, RDFS lacks some useful constructs (negation, inverse properties, existential concept and so on) in order to be a complete schema language. Since a description logic fragment of RDFS is a subset of $\mathcal{SHI}$, we consider it for this study.

B. $\mathcal{SHI}$

RDF has a simple schema language called RDF Schema (RDFS) that allows to express concept and property hierarchies as well as typing restrictions. Nonetheless, RDFS lacks some useful constructs (negation, inverse properties, existential concept and so on) in order to be a complete schema language. Since a description logic fragment of RDFS is a subset of $\mathcal{SHI}$, we consider it for this study.

Example 2: An $\mathcal{SHI}$ schema modeling a university domain is shown in Figure **??**.

$$\begin{array}{rr}
\text{PostgradStudent} \sqsubseteq \text{Student} & \text{UndergradStudent} \sqsubseteq \text{Student} \\
\text{Department} \sqsubseteq \text{Faculty} & \text{Faculty} \sqsubseteq \text{University} \\
\text{Staff} \sqcap \text{Student} & \sqsubseteq \bot \\
\text{Professor} \sqsubseteq \text{Person} & \text{Student} \sqsubseteq \text{Person} \\
\text{Chair} \equiv \text{Person} \sqcap \exists \text{headOf.Department} & \\
\text{Student} \equiv \text{Person} \sqcap \exists \text{takesCourse.Course} & \\
\text{Professor} \equiv \text{Person} \sqcap \exists \text{givesCourse.Course} & \\
\exists \text{headOf.} \top \sqsubseteq \text{Professor} & \text{takesCourse} \equiv \text{givesCourse}^- \\
\text{Staff} \sqsubseteq \text{AdministrativeStaff} \sqcup \text{Professor} &
\end{array}$$

C. SPARQL

SPARQL is W3C recommended query language based on simple graph patterns. It allows variables to be bound to components in the input 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 from triple patterns: a tuple $t \in \text{UBV} \times \text{UV} \times \text{UBLV}$, with V a set of variables disjoint from UBL (URIs, Blank nodes and Literals), is called a triple pattern. Triple patterns grouped together using operators AND (**.**) and UNION form *query patterns*. We do not consider OPTIONAL and FILTER query patterns because containment over the full SPARQL is undecidable. We use an abstract syntax that can be easily translated into the μ-calculus.

Definition 1: A query pattern q is inductively defined as follows :

$$q ::= \text{UBV} \times \text{UV} \times \text{UBLV} \mid q_1 \ . \ q_2 \mid \{q_1\} \text{ UNION } \{q_2\} \mid q_1 \text{ OPTIONAL } \{q_2\}$$

Example 3 illustrates this. We refer the readers to [11] for a formal presentation and semantics of SPARQL.

Example 3: Consider the following queries, taken from the benchmark, that retrieve students' information from the university dataset in Example **??**.

Select all students' information.

```
SELECT ?x ?y ?z ?t ?ssn
?a ?sex ?e ?d ?c
WHERE {                                   Q7a
   ?x a :Student . ?x :name ?y .
   ?x :nickName ?z . ?x :telephone ?t .
   ?x :ssn ?ssn .  ?x :age ?a .
   ?x :sex ?sex . ?x :emailAddress ?e .
   ?x :memberOf ?d . ?x :takesCourse ?c .
}
```

Select master students' information.

```
SELECT ?x ?y ?z ?t ?ssn ?a ?sex ?e ?d ?c
WHERE {                                   Q7b
   ?x a :Student . ?x :name ?y .
   ?x :nickName ?z . ?x :telephone ?t .
   ?x :ssn ?ssn .  ?x :age ?a .
   ?x :sex ?sex .  ?x :emailAddress ?e .
   ?x :memberOf ?d . ?x :takesCourse ?c .
   ?x :masterDegreeFrom ?master .
}
```

Definition 2 (Containment): Given a set of RDFS axioms $\mathcal{S}$ and queries Q_1 and Q_2 with the same arity, Q_1 is *contained in* Q_2 with respect to $\mathcal{S}$, denoted $Q_1 \sqsubseteq_{\mathcal{C}} Q_2$, if and only if the answers of Q_1 are included in the answers of Q_2 for every graph G satisfying $\mathcal{S}$.

Example 4: From the queries in Example 3, $Q7b \sqsubseteq Q7a$ and $Q7a \not\sqsubseteq Q7b$.

In order to benchmark and assess the current state-of-the-are, for this study, we have identified a class of SPARQL queries called tree-structured queries. These are queries that have a tree structure when seen as a graph. The following section introduces this notion.

1) Tree-structured SPARQL Queries: In relational databases, Bernstein and Chiu [2] classified queries into two types: *tree* (is an undirected graph in which any two vertices are connected by exactly one simple path) and *cyclic* queries. An algorithm to decide whether a query is cyclic or not was presented in their paper. This algorithm is based on the idea of representing queries as hypergraphs. Infact, queries have often been considered as hypergraphs (see [12], [2] for instance). It is possible to represent queries as hypergraphs where the nodes of the hypergraph are the variables and constants in the query. There is one hyperedge corresponding to each query subgoal that includes the variables and constants occurring in that subgoal. These studies proved that (Boolean) conjunctive query evaluation, while NP-hard in general, is polynomial in case of acyclic queries, i.e., queries whose associated hypergraphs are acyclic [3]. This distinction of queries as tree and cyclic has its advantages: for example, the evaluation of acyclic (Boolean) conjunctive queries is highly parallelizable [13].

Borrowing the same notion from databases, we propose to view SPARQL queries as graphs. More specifically, a SPARQL query is represented as a bipartite graph, with two kinds of nodes: triple nodes and terms nodes (are URIs, blank nodes, and literals). Using this graph, one is able to determine whether a query is cyclic or not, a formal definition is given below.

Definition 3 (Cyclic Query): A SPARQL query is referred to as *cyclic* if and only if a graph constructed from the query patterns is cyclic and *acyclic* otherwise.

Example 5: The query shown graphically in Figure 1 is a cyclic query.

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 analysed the DBpedia (`http://dbpedia.org`) query log and obtained 378530 real world queries, out of which 15.8% were syntactically incorrect. Using the remaining 74.2%, we tested the cyclicness and acyclicness of queries and found out that: more than 87% of semantic web queries, as visible through DBpedia logs, are acyclic, i.e., a graph constructed from query elements is acyclic, and the remaining 13% are cyclic. This lays the foundation that acyclic queries make up a major part of the real world queries, thus, their separate study is fundamental. Detailed results on the analysis of the structure of queries (tree, directed acyclic graph (DAG), and cyclic) are presented in Table II-C.1.

Beyond the cyclic and acyclic tests, we checked how many of the queries have projection, i.e., not all variables in the graph pattern are distinguished, or not. In this setting, we found out that 63.4% of the queries have projection and 36.6% of the queries have no projection. Further, all of the cyclic queries have projection and out of the tree-structured ones, 65% of the queries have projection and the rest have no projection. These results motivate the design principles for the benchmark.

III. Benchmark

In this section, we present the query containment setup, structure of the benchmark suite and the benchmark description.

A. Query Containment Setup

Query containment is the problem of determining if the result of one query is included in the result of another for any RDF graph. Its advantages are broad: query optimizations, information integration, satisfiability for instance. A test case for containment comprises two queries Q_1 and Q_2, and an optional schema $\mathcal{S}$. The query containment solver (QC solver) produces yes or no answers, i.e., yes if Q_1 is contained in Q_2 and no otherwise. A general workflow diagram designed for this purpose is illustrated in Figure 4.

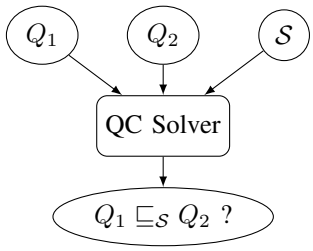

Fig. 2. General workflow of query containment test

B. Structure of the Benchmark

The four key requirements of a benchmark laid out by the benchmark handbook [14] are: understandability i.e., the queries and hence axioms should be understandable, scalability, portability i.e., being able to run on different platforms, and finally relevance i.e., testing typical operations such as joins, disjunctions, and typing restrictions. Thus, we designed the benchmark following these principles.

Benchmark queries are chosen according to the following criteria: projection (or no projection), operator nesting, number of connectives (joins and disjunctions), and requiring RDFS reasoning. Our queries also vary in general characteristics like selectivity, query size, and different types of joins. One thing that should be noted is that, the benchmark criteria are selected in line with the capacity of the current state-of-the-art. The benchmark contains three test suites:

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

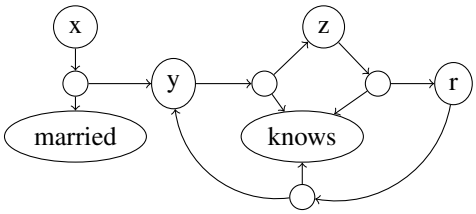

Fig. 1. Cyclic query

Query graph		# of queries
Acyclic	Tree	150145
	DAG	146077
Cyclic		22591
Incorrect syntax		59717

Query graph	Query type	# of queries
DAG	UCQs	50355
	Other	95722
Tree	UCQs	100001
	Other	50144

TABLE I
CYCLE DETECTION

- **U**nion of **C**onjunctive **Q**ueries with **P**rojection (UCQProj) this setting comprises 28 queries which have variables in the SELECT clause (aka distinguished variables) hence the name projection. In addition to the projection, the queries are chosen according to selectivity, number of conjunction and unions, and size (in terms of the number of connectives). For instance, consider the queries in Example 3 which contain 9 (Q7a) and 10 (Q7b) conjunctions respectively. This test suite assesses containment solvers that support projection and union of conjunctive queries.
- **C**onjunctive **Q**ueries with **N**o **P**rojection (CQNoProj) this suite is designed to address the containment of basic graph patterns (conjunctive queries). It contains 20 queries differing in the number of conjunctions and variables they contain. This test suite is developed for containment solvers that do not support projection of variables in queries. Also, in this test suite, disjunctive (union) queries are not allowed.
- **U**nion of **C**onjunctive **Q**ueries under RDFS reasoning (UCQrdfs) this test suite, as its name implies, is designed for containment of queries that require RDFS reasoning. Thus, a set of RDFS schemas are selected in order to be able to test containment in the presence of these constraining schemas. The four schemas cover concept and property hierarchies, and typing (domain and range) restriction axioms. Along with the schemas, the test suite contains 18 queries differing in the: number of operators, number of distinguished variables, and having projection (or no projection).

C. Benchmark Description

We have identified three test suites: UCQProj, CQNoProj, and UCQrdfs, for containment of tree-structured union of conjunctive SPARQL queries under RDFS axioms. In this section, we detail each of these test suites. The benchmark is available on line at `http://sparql-qc-bench.inrialpes.fr/`.

1) UCQProj: In this test suit, there are 28 test cases (p1 – p28 shown in Table II). Each test case comprises two tree-structured union of conjunctive queries. The test cases differ in the number of distinguished variables (*Dvars*) and connectives (conjunction or union). To this end, in the table, the size of the queries measured in terms of the number of connectives is depicted. Tests are carried out using this benchmark with the solvers AFMU and TreeSolver.

2) CQNoProj: as discussed in Section III-B, this test suite contains conjunctive queries with no projection (denoted as *nop* followed by *case number*, shown in Table III) is designed for containment of basic graph patterns. For this case, we have identified 20 different test cases (nop1 – nop20), each case represents containment between two queries. All the test cases in this setting are shown in Table III, along with the number of connectives and variables in the queries.

3) UCQrdfs: in this test suite, there are 4 different schemas, C1–C4 in Table IV. The table also shows that the type and number of axioms used in each schema.

In query containment under RDFS reasoning, there are 28 test cases (rdfs1 – rdfs28 of Table V). In comparison to the test cases in UCQProj and CQNoProj setting, the query sizes are small. Each test case is composed of two tree-structured UCQs and a schema.

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

TABLE IV
AXIOM TYPES FOR EACH OF THE FOUR SCHEMAS.

		AND		UNION		Dvars
p1 p2	Q1a ⊑ Q1b Q1b ⊑ Q1a	1	0	0	0	1
p3 p4	Q2a ⊑ Q2b Q2b ⊑ Q2a	5	5	0	0	3
p5 p6	Q3a ⊑ Q3b Q3b ⊑ Q3a	2	1	0	0	2
p7 p8	Q4c ⊑ Q4b Q4b ⊑ Q4c	3	5	0	0	1
p9 p10	Q5a ⊑ Q5b Q5b ⊑ Q5a	0	0	0	0	2
p11 p12	Q6a ⊑ Q6b Q6b ⊑ Q6a	2	2	0	0	1
p13 p14	Q6a ⊑ Q6c Q6c ⊑ Q6a	2	1	0	0	1

		AND		UNION		Dvars
p15 p16	Q7a ⊑ Q7b Q7b ⊑ Q7a	9	10	0	0	10
p17 p18	Q8a ⊑ Q8b Q8b ⊑ Q8a	3	2	0	0	4
p19 p20	Q9a ⊑ Q9b Q9b ⊑ Q9a	4	4	0	0	2
p21 p22	Q9c ⊑ Q9b Q9b ⊑ Q9c	4	4	0	0	2
p23 p24	Q10a ⊑ Q10b Q10b ⊑ Q10a	2	9	7	1	10
p25 p26	Q11a ⊑ Q11b Q11b ⊑ Q11a	6	7	2	0	2
p27 p28	Q12a ⊑ Q12b Q12b ⊑ Q12a	3	3	1	1	2

TABLE II
UCQPROJ

		AND		Vars
nop1 nop2	Q1a ⊑ Q1b Q1b ⊑ Q1a	1	0	1
nop3 nop4	Q2a ⊑ Q2b Q2b ⊑ Q2a	5	5	3
nop5 nop6	Q3a ⊑ Q3b Q3b ⊑ Q3a	2	1	2
nop7 nop8	Q4c ⊑ Q4b Q4b ⊑ Q4c	3	5	3
nop9 nop10	Q6a ⊑ Q6b Q6b ⊑ Q6a	2	2	3

		AND		Vars
nop11 nop12	Q6a ⊑ Q6c Q6c ⊑ Q6a	2	1	3
nop13 nop14	Q6b ⊑ Q6c Q6c ⊑ Q6b	2	1	3
nop15 nop16	Q7a ⊑ Q7b Q7b ⊑ Q7a	9	10	10
nop17 nop18	Q8a ⊑ Q8b Q8b ⊑ Q8a	3	2	4
nop19 nop20	Q9a ⊑ Q9b Q9b ⊑ Q9a	4	4	3

TABLE III
CQNOPROJ

IV. TESTED QUERY CONTAINMENT SOLVERS

In this section, we present three state-of-the-art query containment solvers. These systems are tested using the benchmark suite.

A. AFMU

AFMU, stands for Alternation Free two-way MU-calculus [8], is a satisfiability (SAT) solver for the alternation fragment of μ-calculus. It is a prototype implementation which determines the satisfiability of a μ-calculus formula by producing a yes or no answer.

B. TreeSolver

TreeSolver: the XML tree logic solver[1] performs static analysis of XPath queries which comprises containment, equivalence and satisfiability. To perform these tasks, the solver translates XPath queries into μ-calculus formulas and then it tests the unsatisfiability of the formula. Unlike the AFMU solver, the unsatisfiability test is performed in a time of $2^{\mathcal{O}(n)}$ whereas it is $2^{\mathcal{O}(n \log n)}$ for AFMU, where n is the size of the formula. The SAT solver component of this system is taken for our purpose.

Around the SAT solvers, AFMU and TreeSolver, we have implemented a wrapper, depicted in Figure 3, used for exper-

[1] http://wam.inrialpes.fr/websolver/

	Axiom		AND		UNION		Dvars
rdfs1	C1	Q9a $\sqsubseteq$ Q9c	0	0	0	1	1
rdfs2	C1	Q9c $\sqsubseteq$ Q9a					
rdfs3	C1	Q9a $\sqsubseteq$ Q9b	0	0	0	0	1
rdfs4	C1	Q9b $\sqsubseteq$ Q9a					
rdfs5	C1	Q9b $\sqsubseteq$ Q9c	0	0	0	1	1
rdfs6	C1	Q9c $\sqsubseteq$ Q9b					
rdfs7	C1	Q9d $\sqsubseteq$ Q9e	4	4	0	0	1
rdfs8	C1	Q9e $\sqsubseteq$ Q9d					
rdfs9	C2	Q10b $\sqsubseteq$ Q10d	0	0	0	0	1
rdfs10	C2	Q10d $\sqsubseteq$ Q10b					
rdfs11	C2	Q10e $\sqsubseteq$ Q10b	1	0	0	0	1
rdfs12	C2	Q10b $\sqsubseteq$ Q10e					
rdfs13	C3	Q11b $\sqsubseteq$ Q11c	0	0	0	0	1
rdfs14	C3	Q11c $\sqsubseteq$ Q11b					

	Axiom		AND		UNION		Dvars
rdfs15	C3	Q11b $\sqsubseteq$ Q11d	0	0	0	0	1
rdfs16	C3	Q11d $\sqsubseteq$ Q11b					
rdfs17	C3	Q11c $\sqsubseteq$ Q11d	0	0	0	0	1
rdfs18	C3	Q11d $\sqsubseteq$ Q11c					
rdfs19	C3	Q11b $\sqsubseteq$ Q11a	0	0	0	0	1
rdfs20	C3	Q11a $\sqsubseteq$ Q11b					
rdfs21	C3	Q11e $\sqsubseteq$ Q11a	0	0	1	0	1
rdfs22	C3	Q11a $\sqsubseteq$ Q11e					
rdfs23	C4	Q13a $\sqsubseteq$ Q13b	3	1	3	1	2
rdfs24	C4	Q13b $\sqsubseteq$ Q13a					
rdfs25	C4	Q13a $\sqsubseteq$ Q13c	3	1	3	1	2
rdfs26	C4	Q13c $\sqsubseteq$ Q13a					
rdfs27	C4	Q13b $\sqsubseteq$ Q13c	3	1	3	1	2
rdfs28	C4	Q13c $\sqsubseteq$ Q13b					

TABLE V
UCQRDFS

imentation. This wrapper, together with the solvers, is used to determine query containment.

C. SPARQL-algebra

SPARQL-algebra is an implementation of SPARQL query subsumption and equivalence based on the theoretical results in [10]. 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/`.

A summary of the features supported by state-of-the-art of query containment solvers is presented in Table VI.

Out of the three systems used in the experiments, AFMU and TreeSolver are μ-calculus satisfiability solvers that need an intermediate system that translates queries into formulas (based on previous works) to determine containment whereas SPARQL-algebra is self contained.

Using the containment solvers we designed an experimental setup that comprises several software components. This setup is illustrated in Figure 4, it is mainly based on the theoretical results developed in our previous works [5], [6], [15]. The components of the architecture are detailed as follows:

- *Jena SPARQL parser*[2] - Jena is an open source semantic web framework for Java. It provides an API, ARQ, for parsing and evaluating SPARQL queries. We used this API to parse queries and create μ-calculus formulas.
- *Mu-calculus Encoder* - this component consists of a package for graph construction, cycle detection, and formula creation from queries and axioms ($\eta(\mathcal{S}) \wedge \mathcal{A}(Q_1) \wedge \neg\mathcal{A}(Q_2)$). It has two parts, the first component produces the formulas for the solver AFMU and the second one computes the formulas for the TreeSolver.
- *SAT solvers* - once the formulas are created, satisfiability solvers are used to determine the unsatisfiability of the formula and hence containment. Presently, for tree-structured queries, two SAT solvers, AFMU and TreeSolver, are available.

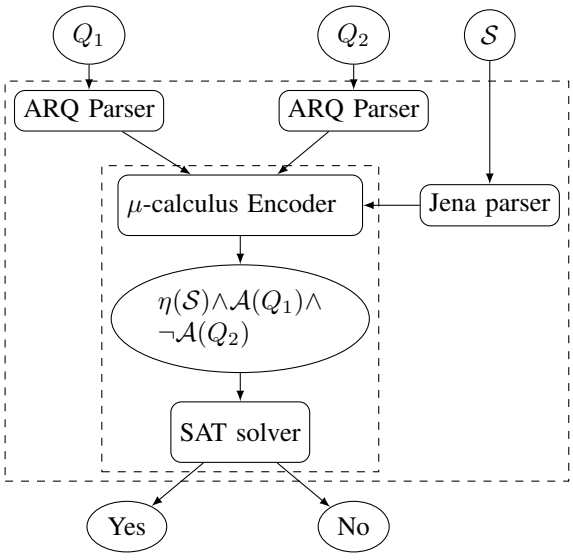

Fig. 3. Query containment wrapper around SAT solvers

Next, experimental results produced using the benchmark are explained.

[2] `http://jena.sourceforge.net/documentation.html`

	No projection	*Projection*	*Reasoning*
CQ	AFMU, TreeSolver, SPARQL-algebra	AFMU, TreeSolver	AFMU, TreeSolver
UCQ	AFMU, TreeSolver	AFMU, TreeSolver	AFMU, TreeSolver
OPTIONAL	SPARQL-algebra	-	-
Blank nodes	AFMU, TreeSolver	AFMU, TreeSolver	AFMU, TreeSolver

TABLE VI

A COMPARISON OF FEATURES SUPPORTED BY CURRENT SYSTEMS

V. EXPERIMENTATION

All the experiments were conducted in a MacBook Pro V10.6.8, Intel Core i7, 2GHz processor speed, and 4GB memory. We carried out two sets of experiments. The first one involves cycle detection – this is done in order to assess the relevance and the percentage of tree-structured real world queries thereby underlining the purpose of the study. The second one involves three experiments on the containment of tree-structured UCQs. The details are explained in the following sections. It is beyond the scope of this paper to provide an in-depth comparison of existing solvers. Rather, we want to give first insights into the state-of-the art and highlight deficiencies of engines based on the benchmark outcome.

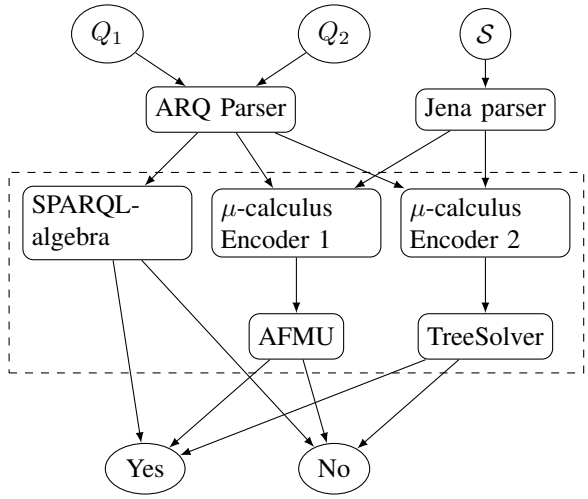

Fig. 4. Experimental setup for query containment test

We conducted tests to compare the performances of the containment solvers AFMU, TreeSolver and SPARQL-algebra.

A. 1st Experimental setting: UCQProj

The test suite used for this experiment is UCQProj (see Section III-C.1). We compared two systems: TreeSolver and AFMU. As can be seen in Figure 5, the TreeSolver outperforms AFMU in all of the benchmark test cases. Further, in one of the cases (p16), where the queries have more than 9 joins, AFMU was not responsive after running 1.5 hours. This behaviour is not unique to AFMU, as whenever there are more than 10 joins in the query, the same situation is observed in the TreeSolver. For instance, for the test case p16, the TreeSolver was non-responding in some runs and in some others it eventually runs out of space.

B. 2nd Experimental setting: CQNoProj

In the benchmark, the test suite used for this experiment is CQNoProj (see Section III-C.2). In light of the fact that, SPARQL-algebra supports only CQs with no projection, we deemed it as necessary to evaluate its performance against TreeSolver and AFMU. The results of this comparison are depicted in Figure 6. From the graph, it can be seen that the performance of SPARQL-algebra is much faster than that of AFMU or TreeSolver. In fact, this comes as no surprise, due to the reason that the latter are exponential time solvers whereas the first is polynomial time solver. Note also that, whenever containment is determined between queries that contain more than 10 joins, Tree Solver and AFMU become non-responsive (do not terminate).

C. 3rd Experimental setting: UCQrdfs

The test suite used for this experiment is UCQrdfs (cf. Section III-C.3). In this experiment, the containment of tree-structured UCQs under RDFS reasoning is tested. Three approaches that are used to encode the containment problem under RDFS axioms is presented in [5]. For this task, we have chosen the schema encoding approach – amounts to encoding the schema axioms as μ-calculus formulas [5].

The results presented in Figure 7 show that except for one case (rdfs13) the TreeSolver out performs AFMU. In fact, like the other two experimental settings, when the query size gets larger both solvers either take a substantial amount of time or run out of space or become non-responsive.

In summary, all the solvers under all of the experimental settings responded positively i.e., they all determined containment correctly except for SPARQL-algebra. It responded negatively, in test cases nop7 and nop8 cf. Table III, when blank nodes are used in the queries.

D. Discussion

As it can be observed in the experiments, a lot needs to be done in order to alleviate the shortcomings of the current systems. Here, we discuss the pros and cons of these systems.

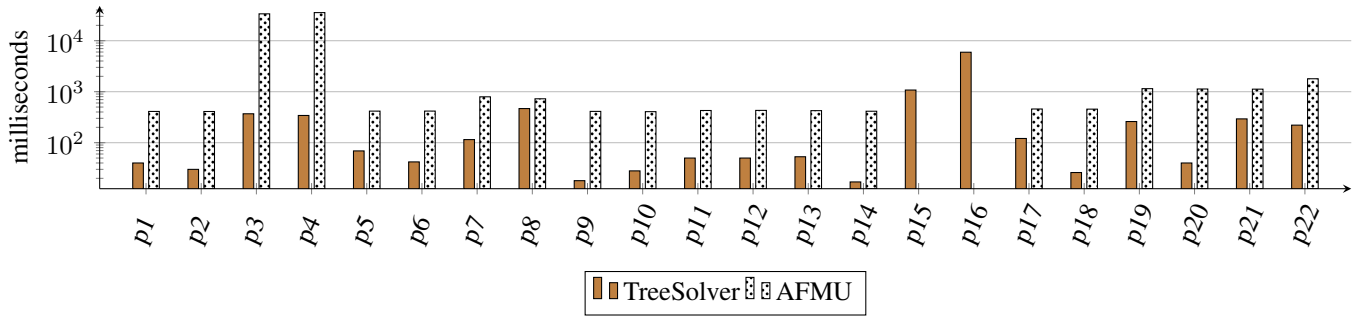

Fig. 5. Results for UCQProj test suite (logarithmic scale)

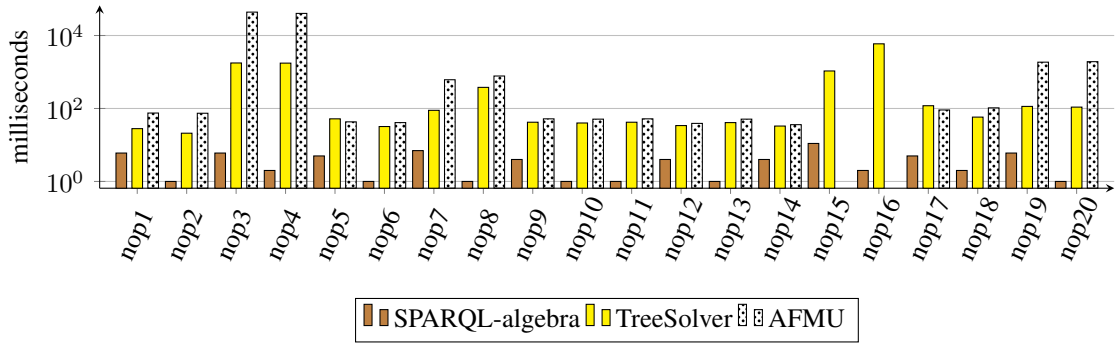

Fig. 6. Results for CQNoProj test suite (logarithmic scale)

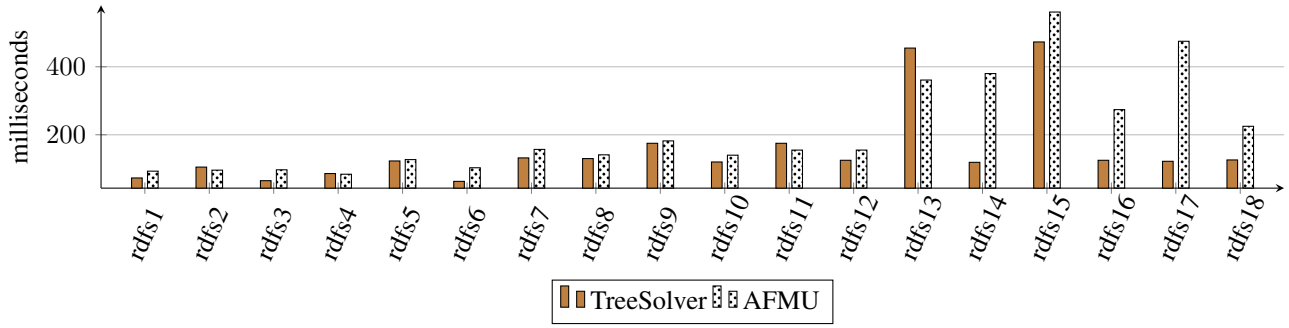

Fig. 7. Results for UCQrdfs test suite

1) AFMU: In terms of capacity AFMU, is able determine containment of acyclic UCQs under ontological axioms. Specially, for queries of reasonable size, the solver determined their containment correctly. The problem is that when queries have large size (for instance, more than 8 joins), the solver takes more than 1/2 hour or becomes non-responsive. This is shown in the result graph in Figure 5 for test cases p15 and p16, also in Figure 6 for test cases fnop15 and nop16. However, the implementation of this solver is very poor, even the authors had documented that it can be improved so as to have faster response times. In fact, this is reflected in the experiments that it has the worst performance out of the three. Therefore, reimplementing or improving the solver should be noted as a perspective.

In another note, to be able to determine containment of general UCQs (beyond the tree-structured ones), an extension of the solver is compulsory.

2) TreeSolver: this solver works well with tree-structured queries and RDFS reasoning if the queries have reasonable size, it has an on line version, and it also supports static analysis of XPath queries among others. Its limitations are similar to that of AFMU: no support for cyclic queries, and queries of large size (for example, nop16 and p16).

3) SPARQL-algebra: the advantages of this solver compared to the others are that, it supports subsumption of OPTIONAL query patterns and also cyclic CQs. Its limitations include: no support for distinguished variables, i.e., no projection, UNION, and reasoning. Moreover, basic graph patterns (AND queries) with blank nodes are not supported, in fact, this is reflected in the experiment cf. section V-B test cases

nop7 and nop8 in Figure 6.

In this paper, we have first studied the profile of real-world queries with respect to theoretical characterisation 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, we have designed a containment benchmark made of three first query containment test suites testing projection, no projection and RDFS reasoning.

We have applied these to existing containment solvers (AFMU, TreeSolver and SPARQL-algebra). Obviously, current μ-calculus solvers are not optimised for conjunctive queries without projection which are better dealt with by SPARQL-algebra. Hence, so far the best strategy is to use one solver when queries are conjunctive and do not have projection and another when they have projection, union and axioms.

These analyses confirm that our benchmarks are well-suited for pinpointing the theoretical shortcomings of containment solvers. These experimental results show that 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. In the future, we plan to work towards this direction.

Additionally, we will improve and extend this benchmark, e.g., adding other test suites designed for containment of queries under expressive description logic axioms such as OWL2.

VI. Related Works

Recently, static analysis and optimization of SPARQL queries has attracted widespread attention, notably [5], [6], [15], [10] for static analysis and [16], [17], [18], [10] for optimization. These studies have grounded the theoretical aspects of these fundamental problems. However, to the best of our knowledge, there is only one implementation from [10] and it supports only conjunction and OPTIONAL queries with no projection.

On the other hand, in databases, containment of union conjunctive queries (UCQs) is well studied and has a well know NP-complete complexity. The importance of the study of this problem goes beyond the field of databases, it has its fair share from the description logic community. Many of the works, from description logics, concentrated on the problem of query answering as containment follows from it. These works, have sound theoretical proofs, algorithms, mathematical explanations, and so on. However, they lack an implementation (or experimentation) of their approaches.

In line with CQs in databases and description logic worlds, we have regular path queries— languages that are used to query arbitrary length paths in graph databases — in semi structured data. Like CQs, they have been used and studied widely. They are different from CQs in that, they allow recursion by using regular expression patterns. The problem of containment has been addressed for extensions of this language. In this regard, a prominent language used in semi-structured data is XPath. This language has been studied extensively over the last decade. These studies are all round, from extending or limiting to static analysis. Static analysis of XPath queries has been studied in [9], encompassing containment, equivalence, coverage, and satisfiability of XPath queries. This work is motivated by [9] in that the approach to study these problems using a graph logic and providing an implementation which has been put to practice.

Finally, various SPARQL query evaluation performance benchmarks have been proposed [19], [20], [21], but no SPARQL query containment benchmark to our knowledge.

VII. Conclusion

In this paper, we have first studied the profile of real-world queries with respect to theoretical characterisation 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, we have designed a containment benchmark made of three first query containment test suites testing projection, no projection and RDFS reasoning.

We have applied these to existing containment solvers (AFMU, TreeSolver and SPARQL-algebra). Obviously, current μ-calculus solvers are not optimised for conjunctive queries without projection which are better dealt with by SPARQL-algebra. Hence, so far the best strategy is to use one solver when queries are conjunctive and do not have projection and another when they have projection, union and axioms.

These analyses confirm that our benchmarks are well-suited for pinpointing the theoretical shortcomings of containment solvers. These experimental results show that 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. In the future, we plan to work towards this direction.

Additionally, we will improve and extend this benchmark, e.g., adding other test suites designed for containment of queries under expressive description logic axioms such as OWL2.

References

[1] 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*. ACM, 1977, pp. 77–90.

[2] P. Bernstein and D. Chiu, "Using semi-joins to solve relational queries," *Journal of the ACM (JACM)*, vol. 28, no. 1, pp. 25–40, 1981.

[3] M. Yannakakis, "Algorithms for acyclic database schemes," in *VLDB'81*, vol. 7, 1981, pp. 82–94.

[4] D. Calvanese, G. De Giacomo, and M. Lenzerini, "Conjunctive Query Containment and Answering under Description Logics Constraints," *ACM Trans. on Computational Logic*, vol. 9, no. 3, pp. 22.1–22.31, 2008.

[5] M. W. Chekol, J. Euzenat, P. Genevès, and N. Layaïda, "SPARQL query containment under RDFS entailment regime," in *IJCAR'12*. Springer, 2012, pp. 134–148.

[6] ——, "SPARQL query containment under SHI axioms," in *AAAI'12*, vol. 1, 2012, pp. 10–16.

[7] D. Kozen, "Results on the propositional μ-calculus," *Theor. Comp. Sci.*, vol. 27, pp. 333–354, 1983.

[8] 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*, 2005, pp. 277–291.

[9] P. Genevès, N. Layaïda, and A. Schmitt, "Efficient Static Analysis of XML Paths and Types," in *PLDI '07*. New York, NY, USA: ACM, 2007, pp. 342–351.
[10] A. Letelier, J. Pérez, R. Pichler, and S. Skritek, "Static analysis and optimization of semantic web queries," in *PODS'12*. ACM, 2012, pp. 89–100.
[11] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," W3C Rec., 2008.
[12] C. Chekuri and A. Rajaraman, "Conjunctive query containment revisited," *Database Theory–ICDT'97*, pp. 56–70, 1997.
[13] G. Gottlob, N. Leone, and F. Scarcello, "The complexity of acyclic conjunctive queries," *Journal of the ACM (JACM)*, vol. 48, no. 3, pp. 431–498, 2001.
[14] J. Gray, *Benchmark handbook: for database and transaction processing systems*. Morgan Kaufmann Publishers Inc., 1992.
[15] M. W. Chekol, J. Euzenat, P. Genevès, and N. Layaïda, "PSPARQL query containment," in *DBPL'11*, Aug. 2011.
[16] J. Groppe, S. Groppe, and J. Kolbaum, "Optimization of SPARQL by using coreSPARQL," in *ICEIS (1)*, 2009, pp. 107–112.
[17] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds, "SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation," in *Proceeding of the 17th international conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 595–604.
[18] M. Schmidt, M. Meier, and G. Lausen, "Foundations of SPARQL Query Optimization," in *ICDT '10*. New York, NY, USA: ACM, 2010, pp. 4–33.
[19] 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.
[20] ——, "The Berlin SPARQL benchmark," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5, no. 2, pp. 1–24, 2009.
[21] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, "SPˆ 2Bench: a SPARQL performance benchmark," in *ICDE'09*. Ieee, 2009, pp. 222–233.