PSPARQL Query Containment

Melisachew Wudage Chekol
INRIA and LIG
melisachew.chekol@inria.fr

Jérôme Euzenat
INRIA and LIG
jerome.euzenat@inria.fr

Pierre Genevès
CNRS and LIG
pierre.geneves@inria.fr

Nabil Layaïda
INRIA and LIG
nabil.layaida@inria.fr

ABSTRACT

Querying the semantic web is mainly done through SPARQL. This language has been studied from different perspectives such as optimization and extension. One of its extensions, PSPARQL (Path SPARQL) provides queries with paths of arbitrary length. We study the static analysis of queries written in this language, in particular, containment of queries: determining whether, for any graph, the answers to a query are contained in those of another query. Our approach consists in encoding RDF graphs as transition systems and queries as μ-calculus formulas and then reducing the containment problem to testing satisfiability in the logic.

1. INTRODUCTION

Access to semantic web data expressed in Resource Description Framework (RDF) can be achieved through querying. Currently, querying RDF graphs is done mainly with the SPARQL query language. It has been a source of research from various perspectives mainly extending the language with new features and optimizing queries automatically. Querying RDF graphs with SPARQL amounts to matching graph patterns that are sets of triples of subjects, predicates and objects. These triples are usually connected to form graphs by means of joins expressed using several occurrences of the same variable. On the other hand, PSPARQL (Path SPARQL) allows querying of arbitrary length paths by using regular expression patterns. Regular path queries (RPQs) are useful for expressing complex navigations in a graph. In particular, union and transitive closure are crucial when one does not have a complete knowledge of the structure of the knowledge base. SPARQL 1.0 lacks recursion mechanism and supports a simple form of RPQs however its extensions such as PSPARQL [2] and its successor SPARQL1.1 support this feature.

Query optimization aims at improving the performance of query evaluation. Since queries in the semantic web are evaluated over huge RDF graphs, optimizations are crucial. Many studies contributed to query optimization using in particular the relational algebra from the database community [15]. This is very often achieved by using rules for rewriting queries into equivalent but faster ones. All these works, however, need at some point to prove the correctness of query optimization, i.e., the semantics of the optimized query remains the same as the original one. In other terms, the results of a given query are exactly the same as the optimized one regardless of the considered database. This can be reduced to query containment. Thus query containment plays a central role in database and knowledge base query optimization [15, 8, 4]. In addition, query containment can be of independent interest for performing other optimizations. For example, if a query q_1 is contained in q_2, then q_1 can be evaluated on the materialized view of q_2 rather than on the whole data graph.

Such approaches have also been applied to SPARQL [19], but not yet for PSPARQL.

We address the problem of static analysis of PSPARQL queries, encompassing satisfiability, containment and equivalence of queries. We introduce an approach which has already been successfully applied for XPath [11]. PSPARQL is interpreted over graphs, hence we encode it in a graph logic, specifically the alternation-free fragment of the μ-calculus [16] with converse and nominals [22] interpreted over labeled transition systems. We show that this logic is powerful enough to deal with query containment where queries are made of regular expression patterns which allow navigation through the graph. One benefit of using a μ-calculus encoding is to take advantage of fixpoints and modalities for encoding recursion. Furthermore, this logic admits exponential time decision procedures that can be implemented efficiently in practice [23, 11].

After presenting RDF, PSPARQL and the μ-calculus (§2), we show how to translate RDF graphs into transition systems (§3.1) and PSPARQL queries into μ-calculus formulas (§3.2). Therefore, query containment in PSPARQL can be reduced to unsatisfiability test in μ-calculus (§4).

2. PRELIMINARIES

This section introduces the basics of RDF and PSPARQL.

2.1 RDF: Resource Description Framework

RDF is a language used to express structured information on the Web as graphs. Here we present a compact formalization of RDF [14]. Let U, B, and L be three disjoint infinite sets denoting the set of URIs (identify a resource), blank nodes (denote an unidentified resource) and literals (a character string or some other type of data) respectively. We abbreviate any union of these sets as for instance, $\text{UBL} = U \cup B \cup L$. A triple of the form $<s, p, o> \in \text{UB} \times U \times \text{UBL}$ is called an *RDF triple*. s is the *subject*, p is the *predicate*, and o is the *object* of the triple. Each triple may be thought of as an edge between the subject and the object labelled by the predicate, hence a set of RDF triples is often referred to as an *RDF graph*.

EXAMPLE 1 (RDF GRAPH). *Here are 8 triples of an RDF graph about writers and their works: (all identifiers correspond to URIs, _:b is a blank node):*

Poe wrote thegoldbug .	*Baudelaire translated thegoldbug .*
Poe wrote theraven .	*Mallarmé translated theraven .*
theraven type Poem .	*Mallarmé wrote _:b .*
_:b type Poem .	*thegoldbug type Novel .*

RDF has a model theoretic semantics [14].

2.2 RDFS

RDF Schema (RDFS) [14] may be considered as a simple ontology language expressing subsumption relations between classes or properties. Technically, this is an RDF vocabulary used for expressing axioms constraining the interpretation of graphs. The RDFS vocabulary and its semantics are given in [14]. We consider a core fragment of RDFS called ρdf [17] which contains the minimal vocabularies, $\rho\text{df} = \{$`sp,sc,type,dom,range`$\}$. Where `sp` denotes *subproperty* relation, `sc` is *subclass*, and `dom` is for *domain*. The authors in [17] proved this fragment to be minimal and well-behaved. Moreover, its semantics is equivalent to that of the full RDFS.

In [14], a set of rules are given which allow to deduce or infer new triples using RDF Schema assertions. For our purposes, we consider a subset of RDFS inference or deduction rules, shown in Table 1.

Subclass	*Subproperty*	*Typing*
$\frac{<a,\text{sc},b> \quad <b,\text{sc},c>}{<a,\text{sc},c>}$	$\frac{<a,\text{sp},b> \quad <b,\text{sp},c>}{<a,\text{sp},c>}$	$\frac{<a,\text{dom},b> \quad <x,a,y>}{<x,\text{type},b>}$
$\frac{<a,\text{sc},b> \quad <x,\text{type},a>}{<x,\text{type},b>}$	$\frac{<a,\text{sp},b> \quad <x,a,y>}{<x,b,y>}$	$\frac{<a,\text{range},b> \quad <x,a,y)}{<y,\text{type},b>}$

Implicit Typing	
$\frac{<a,\text{dom},b> \quad <c,\text{sp},a> \quad <x,c,y>}{<x,\text{type},b>}$	$\frac{<a,\text{range},b> \quad <c,\text{sp},a> \quad <x,c,y>}{<y,\text{type},b>}$

Table 1: RDFS inference rules

EXAMPLE 2. *This example shows the usage of RDFS inference rules, consider the graph {<John,*`type`*,Student>, <Student,*`sc`*,Person>}. By applying Subclass rule, rule [2], it can be deduced that <John,*`type`*,Person>.*

2.3 PSPARQL

PSPARQL (short for Path SPARQL) extends SPARQL with regular expression patterns. SPARQL [18] is a W3C recommended query language for RDF. PSPARQL overcomes the limitation of the current version of SPARQL which is the inability to express path queries. Before presenting the syntax and semantics of PSPARQL, let us briefly introduce the notion of regular expression patterns (cf. [2] for detailed discussion).

2.3.1 Regular Expressions

Regular expressions are patterns used to describe languages (i.e., sets of strings) from a given alphabet. Let $\Sigma = \{a_1, ..., a_n\}$ be an alphabet. A *string/word* is a finite sequence of symbols from the alphabet Σ. A *language* $\mathcal{L}$ is a set of words over Σ which is a subset of Σ^*, i.e, $\mathcal{L}(\Sigma) \subseteq \Sigma^*$. A word can be either empty ϵ or a sequence of alphabet symbols $a_1...a_n$. If $A = a_1...a_n$ and $B = b_1...b_m$ are two words over some alphabet Σ, then $A.B$ is a word over the same alphabet defined as: $A.B = a_1...a_nb_1...b_m$.

DEFINITION 1 (REGULAR EXPRESSION PATTERN). *Given an alphabet Σ and a set of variables V, a regular expression $\mathcal{R}(\Sigma, V)$ can be constructed inductively as follows:*

$$e := uri \mid x \mid e_1 \shortmid e_2 \mid e_1.e_2 \mid e^+ \mid e^*$$

Where $e \in \mathcal{R}(\Sigma, V)$ and x denotes a variable, $e_1 \shortmid e_2$ denotes disjunction, $e_1.e_2$ denotes concatenation, e^+ denotes positive closure, and e^* denotes Kleene closure. Let U be a set of URIs and V a set of variables, a regular expression over $\mathcal{R}(U, V)$ can be used to define a language over the alphabet $U \cup V$.

2.3.2 PSPARQL Syntax

The only difference between the syntax of SPARQL and PSPARQL is on triple patterns. Triple patterns in PSPARQL contain regular expressions in property positions instead of only URIs or variables as it is the case of SPARQL. Queries are formed based on the notion of query patterns defined inductively from triple patterns: a tuple $t \in \text{UBV} \times \mathcal{R}(U, V) \times \text{UBLV}$, with V a set of variables disjoint from UBL, is called a triple pattern. Triple patterns grouped together using connectives (AND, UNION, OPT) form *graph patterns* (a.k.a query patterns). We use an abstract syntax that can be easily translated into μ-calculus.

DEFINITION 2 (QUERY PATTERN). *A PSPARQL query pattern q is inductively defined as follows :*

$$q = \; t \in \text{UBV} \times \mathcal{R}(U, V) \times \text{UBLV}$$
$$\mid q_1 \textit{ AND } q_2 \mid q_1 \textit{ UNION } q_2 \mid q_1 \textit{ OPT } q_2 \mid q_1 \text{ MINUS } Q_2$$

Where $\mathcal{R}(U, V)$ is a regular expression pattern defined over URIs U and query variables V.

DEFINITION 3. *A PSPARQL SELECT query is a query of the form $q\{\overrightarrow{w}\}$ where $\overrightarrow{w}$ is a tuple of variables in V which are called distinguished variables, and q is a query pattern.*

EXAMPLE 3 (PSPARQL QUERIES). *Consider the following queries $q_1\{?x\}$ and $q_2\{?x\}$ on the graph of Example 1:*

q_1

```
SELECT ?x
  WHERE {
      ?x    (translated | wrote) . type   Poem.
  }
```

q_2

```
SELECT ?x
   WHERE {
       { ?x  (translated  .  type)    Poem }
         UNION
       { ?x wrote  ?l .}
   }
```

2.3.3 PSPARQL Semantics

The semantics of PSPARQL queries is given by a partial mapping function $\rho : V \mapsto UBL$. The domain of ρ, $dom(\rho)$, is the subset of V on which ρ is defined. Two mappings ρ_1 and ρ_2 are said to be *compatible* if $\forall x \in dom(\rho_1) \cap dom(\rho_2)$, $\rho_1(x) = \rho_2(x)$. Hence, $\rho_1 \cup \rho_2$ is also a mapping. This allows for defining the join, union, and difference operations between two sets of mappings M_1, and M_2 as shown below:

$$\begin{aligned} M_1 \bowtie M_2 &= \{\rho_1 \cup \rho_2 \mid \rho_1 \in M_1, \rho_2 \in M_2 \\ &\qquad \text{are compatible mappings } \} \\ M_1 \cup M_2 &= \{\rho \mid \rho \in M_1 \text{ or } \rho \in M_2\} \\ M_1 \setminus M_2 &= \{\rho \in M_1 \mid \forall \rho_1 \in M_2, \rho \text{ and } \rho_1 \\ &\qquad \text{are not compatible } \} \end{aligned}$$

Now, we are ready to define the evaluation of PSPARQL triple patterns recursively as follows:

$$\begin{aligned} [\![<x, uri, y>]\!]_G &= \{\rho \mid \ <\rho(x), \rho(uri), \rho(y)> \in G\} \\ [\![<x, z, y>]\!]_G &= \{\rho \mid \ <\rho(x), \rho(z), \rho(y)> \in G\} \\ [\![<x, e \mid e', y>]\!]_G &= [\![<x, e, y>]\!]_G \cup [\![<x, e', y>]\!]_G \\ [\![<x, e.e', y>]\!]_G &= [\![<x, e, n>]\!]_G \bowtie [\![<n, e', y>]\!]_G \\ [\![<x, e^+, y>]\!]_G &= \{\rho \mid \exists <n_0, e, n_1>, <n_1, e, n_2>, ..., \\ &\qquad <n_{k-1}, e, n_k> \in G \text{ such that } n_0 = \rho(x), \\ &\qquad n_k = \rho(y) \text{ and } e...e \in \mathcal{L}(e^+)\} \\ [\![<x, e^*, y>]\!]_G &= \{\rho \mid \rho(x) = \rho(y)\} \cup [\![<x, e^+, y>]\!]_G \end{aligned}$$

The evaluation of query patterns over an RDF graph G is inductively defined by:

$$\begin{aligned} [\![.]\!]_G &: q \rightarrow 2^{\mathrm{V} \times \mathrm{UBL}} \\ [\![q_1 \text{ AND } q_2]\!]_G &= [\![q_1]\!]_G \bowtie [\![q_2]\!]_G \\ [\![q_1 \text{ UNION } q_2]\!]_G &= [\![q_1]\!]_G \cup [\![q_2]\!]_G \\ [\![q_1 \text{ OPT } q_2]\!]_G &= ([\![q_1]\!]_G \bowtie [\![q_2]\!]_G) \cup ([\![q_1]\!]_G \setminus [\![q_2]\!]_G) \\ [\![q_1 \text{ MINUS } q_2]\!]_G &= [\![q_1]\!]_G \setminus [\![q_2]\!]_G \\ [\![q\{\overrightarrow{w}\}]\!]_G &= \pi_{\overrightarrow{w}}([\![q]\!]_G) \end{aligned}$$

Where the projection operator $\pi_{\overrightarrow{w}}$ selects only those part of the mappings relevant to variables in $\overrightarrow{w}$.

EXAMPLE 4 (ANSWERS TO SPARQL QUERIES). *The answers to query q_1 and q_2 of Example 3 on graph G of Example 1 are respectively $\{Poe, Mallarme\}$ and $\{Baudelaire, Poe, Mallarme\}$. Hence, $[\![q_1]\!]_G \subseteq [\![q_2]\!]_G$.*

Beyond this particular example, the goal of query containment is to determine whether this holds for any graph.

DEFINITION 4 (CONTAINMENT). *Given queries q_1 and q_2 with the same arity, q_1 is contained in q_2, denoted $q_1 \sqsubseteq q_2$, iff for any graph G, $[\![q_1]\!]_G \subseteq [\![q_2]\!]_G$.*

DEFINITION 5 (EQUIVALENCE). *Two queries q_1 and q_2 are equivalent, $q_1 \equiv q_2$, iff $q_1 \sqsubseteq q_2$ and $q_2 \sqsubseteq q_1$.*

3. ENCODINGS

In this section, encodings of RDF graphs as transition systems, and regular expressions and PSPARQL queries as μ-calculus formulas are explained.

3.1 Encoding RDF graphs as Transition Systems

Before presenting the encoding of RDF graphs as transition systems over which the μ-calculus is interpreted, we introduce the syntax and semantics of the μ-calculus.

3.1.1 μ-calculus

The modal μ-calculus [16] is an expressive logic which adds recursive features to modal logic using fixpoint operators.

The syntax of the μ-calculus is composed of countable sets of *atomic propositions* AP, a set of *nominals* Nom, a set of *variables* Var, a set of *programs* $Prog$ for navigating in graphs. A μ-calculus formula, φ, can be defined inductively as follows:

$$\begin{aligned} \varphi \quad ::= &\quad \top \mid \bot \mid p \mid X \mid \neg\varphi \mid \varphi \vee \psi \mid \\ &\quad \varphi \wedge \psi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu X \varphi \mid \nu X \varphi \end{aligned}$$

where $p \in AP \cup Nom, X \in Var$ and $a \in Prog$ is either an atomic program or its converse $\bar{a}$. The greatest and least fixpoint operators (ν and μ) respectively introduce general and finite recursion in graphs [16].

The semantics of the μ-calculus is given over a transition system, $K = (S, R, L)$ where S is a non-empty set of nodes, $R : Prog \rightarrow 2^{S \times S}$ is the transition function, and $L : AP \rightarrow 2^S$ assigns a set of nodes to each atomic proposition or nominal where it holds, such that $L(p)$ is a *singleton* for each nominal p. For converse programs, R can be extended as $R(\bar{a}) = \{(s', s) \mid (s, s') \in R(a)\}$. Besides, a valuation function $V : \mathrm{Var} \rightarrow S$ is used to assign a set of nodes to each variable. For a valuation V, variable X, and a set of nodes $S' \subseteq S$, $V[X/S']$ is the valuation that is obtained from V by assigning S' to X. The semantics of a formula in terms of a transition system (a.k.a. Kripke structure) and a valuation function is represented by $[\![\varphi]\!]^K_V$. The semantics of basic μ-calculus formula is defined as follows:

$$\begin{aligned} [\![\mathtt{p}]\!]^K_V &= L(p), p \in AP \cup Nom \\ [\![\mathtt{X}]\!]^K_V &= V(X), X \in Var \\ [\![\neg\varphi]\!]^K_V &= S \backslash [\![\varphi]\!]^K_V \\ [\![\varphi \wedge \psi]\!]^K_V &= [\![\varphi]\!]^K_V \cap [\![\psi]\!]^K_V \\ [\![\varphi \vee \psi]\!]^K_V &= [\![\varphi]\!]^K_V \cup [\![\psi]\!]^K_V \\ [\![\langle \mathtt{a} \rangle \varphi]\!]^K_V &= \{s \in S \mid \exists s' \in S.(s, s') \in R(a) \ \wedge s' \in [\![\varphi]\!]^K_V\} \\ [\![[\mathtt{a}]\varphi]\!]^K_V &= \{s \in S \mid \forall s' \in S.(s, s') \in R(a) \Rightarrow s' \in [\![\varphi]\!]^K_V\} \\ [\![\mu \mathtt{X} \varphi]\!]^K_V &= \bigcap \{S' \subseteq S \mid [\![\varphi]\!]^K_{V[X/S']} \subseteq S'\} \\ [\![\nu \mathtt{X} \varphi]\!]^K_V &= \bigcup \{S' \subseteq S \mid S' \subseteq [\![\varphi]\!]^K_{V[X/S']}\} \end{aligned}$$

Once providing the syntax and semantics of μ-calculus, the next subsections introduce representation of RDF graphs as transition systems and queries as formulas.

3.1.2 Encoding of RDF graphs

An RDF graph is encoded as a transition system in which nodes correspond to RDF entities and RDF triples. Edges relate entities to the triples they occur in. Different edges are used for distinguishing the functions (subject, object, predicate). expressing predicates as nodes, instead of atomic programs, makes it possible to deal with full RDF expressiveness in which a predicate may also be the subject or object of a statement.

DEFINITION 6 (TRANSITION SYSTEM FOR RDF GRAPH). *Given an RDF graph, $G \subseteq \mathrm{UB} \times U \times \mathrm{UBL}$, the transition system associated G, $\sigma(G) = (S, R, L)$ over $AP = \mathrm{UBL} \cup \{s', s''\}$, is such that:*

- $S = S' \cup S''$ *with* S' *and* S'' *the smallest sets such that* $\forall u \in U_G, \exists n^u \in S'$, $\forall b \in B_G, \exists n^b \in S'$, $\forall l \in L_G, \exists n^l \in S'$ *and* $\forall t \in G, \exists n^t \in S''$,
- $\forall t = <s, p, o> \in G$, $\langle n^s, n^t \rangle \in R(s)$, $\langle n^t, n^p \rangle \in R(p)$, *and* $\langle n^t, n^o \rangle \in R(o)$,
- $L : \text{UBL} \to 2^S; \forall u \in U_G, L(u) = \{n^u\}, \forall b \in B_G, L(b) = S'$, $L(s') = S'$, $\forall l \in L_G, L(l) = \{n^l\}$ *and* $L(s'') = S''$,
- $\forall n^t, n^{t'} \in S''$, $\langle n^t, n^{t'} \rangle \in R(d)$.

The program d is introduced to render each triple accessible to the others and thus facilitate the encoding of queries. The function σ associates what we call a *restricted transition system* to any RDF graph. Formally, we say that a transition system K is a *restricted transition system* iff there exists an RDF graph G such that $K = \sigma(G)$.

A restricted transition system is thus a bipartite graph composed of two sets of nodes: S', those corresponding to RDF entities, and S'', those corresponding to RDF triples. For example, Figure 1 shows the restricted transition system associated with the graph of Example 1.

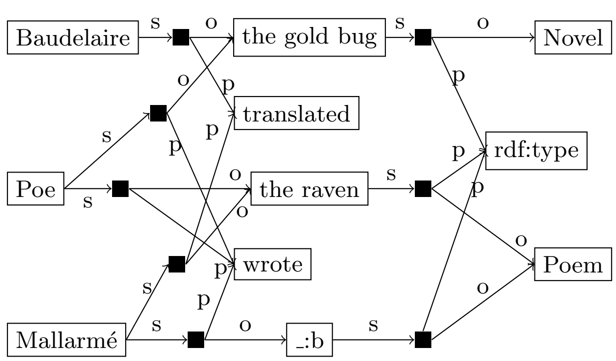

Figure 1: Transition system encoding the RDF graph of Example 1. Nodes in S'' are black anonymous nodes; nodes in S' are the other nodes (d-transitions are not displayed).

When checking for query containment, we consider the following restrictions:

- The set of programs is fixed: $Prog = \{s, p, o, d, \bar{s}, \bar{p}, \bar{o}, \bar{d}\}$ (note that $d = \bar{d}$).
- A model must be a restricted transition system.

This last constraint can be expressed in the μ-calculus as follows:

Proposition 1 (RDF restriction on transition systems). *A formula φ is satisfied by some restricted transition system if and only if $\varphi \wedge \varphi_r$ is satisfiable by some transition system, i.e.* $\exists K_r [\![\varphi]\!]^{K_r} \neq \emptyset \iff \exists K [\![\varphi \wedge \varphi_r]\!]^K \neq \emptyset$, *where:*

$$\varphi_r = \quad \nu X.\theta \wedge \kappa \ \wedge (\neg \langle d \rangle \top \vee \langle d \rangle X)$$

in which $\theta = \langle \bar{s} \rangle s' \wedge \langle p \rangle s' \wedge \langle o \rangle s' \wedge \neg \langle s \rangle \top \wedge \neg \langle \bar{p} \rangle \top \wedge \neg \langle \bar{o} \rangle \top$ *and* $\kappa = [\bar{s}]\xi \wedge [p]\xi \wedge [o]\xi$ *with*

$$\xi = \begin{cases} \neg \langle \bar{s} \rangle \top \wedge \neg \langle o \rangle \top \wedge \neg \langle p \rangle \top \wedge \neg \langle d \rangle \top \wedge \neg \langle \bar{d} \rangle \top \\ \wedge \neg \langle s \rangle s' \wedge \neg \langle \bar{o} \rangle s' \wedge \neg \langle \bar{p} \rangle s'. \end{cases}$$

The formula φ_r ensures that θ and κ hold in every node reachable by a d edge, i.e. in every S'' node. The formula θ forces each S'' node to have a subject, predicate and object. The formula κ navigates from a s'' node to every reachable s' node, and forces the latter not to be directly connected to other subject, predicate or object nodes.

Proof. ($\Rightarrow$) Assume that $\exists K_r [\![\varphi]\!]^{K_r} \neq \emptyset$, since φ_r is satisfied by only transition systems, one gets $[\![\varphi_r]\!]^{k_r} \neq \emptyset$. Hence it follows that, $\exists K_r [\![\varphi]\!]^{K_r} \neq \emptyset$ and $[\![\varphi_r]\!]^{k_r} \neq \emptyset$ which implies $\exists K_r [\![\varphi]\!]^{k_r} \wedge [\![\varphi_r]\!]^{K_r} \neq \emptyset$. From this, using the semantics of μ-calculus formula, one obtains $\exists K_r [\![\varphi \wedge \varphi_r]\!]^{K_r} \neq \emptyset$. Since a restricted transition system is also a transition system, $K_r \subseteq K$, it follows that $\exists K . [\![\varphi \wedge \varphi_r]\!]^K \neq \emptyset$.
($\Leftarrow$) Assume that $\exists K [\![\varphi \wedge \varphi_r]\!]^K \neq \emptyset$. We construct a restricted transition system model $K_r = (S_r, R_r, L_r)$ and a function $f : K_r \to K$ from $K = (S, R, L)$. Add a node n'_0 to S_r with $f(n'_0) = n_0$ where $\varphi \wedge \varphi_r$ is satisfied in K. Supppose we have constructed a node n_r of S_r. For $j \in \{s, p, o\}$, if there is $n \in S$ with $(f(n_r), n) \in R(j)$, then pick one such n and add a node n'_r to S_r with $f(n'_r) = n$. Finally, for an atomic proposition p, $L_r(p) = \{n_r \in S_r \mid f(n_r) \in L(p)\}$.
The RDF triple structure is maintained in K_r i.e.
$\langle (s, s''), (s'', p), (s'', o) \rangle$ is valid through out the graph. If there were node pairs outside of this structure, then φ_r will not be satisfied. Throught out the graph, θ and κ ensure that for each triple node $s'' \in S_r$, there exists exactly one incoming subject edge, one outgoing property edge, and one outgoing object edge. Hence, $[\![\varphi_r]\!]^{K_r} \neq \emptyset$
To verify that $[\![\varphi]\!]^{K_r} \neq \emptyset$, it is enough to show $[\![\varphi]\!]^K \Rightarrow [\![\varphi]\!]^{K_r}$ by induction on the structure of φ. □

3.2 Encoding PSPARQL Queries as μ-calculus Formulas

Queries are translated to μ-calculus formulas. The principle of the translation is that each triple pattern is associated with a sub-formula stating the existence of the triple somewhere in the graph. Hence, they are quantified by μ so as to put them out of the context of a state. In this translation, variables are replaced by nominals which will be satisfied when they are matched in such triple relations. For that purpose, we use a function $\lambda : VUBL \to UBL$ such that:

$$\lambda(x) = \begin{cases} v_x & \text{if } x \in V \\ x & \text{if } x \in \text{UBL} \end{cases}$$

The function $\mathcal{A}$ encodes queries inductively on the structure of query patterns. AND and UNION are replaced by boolean connectives $\wedge$ and $\vee$ respectively. The MINUS operator is translated as $\wedge$ and $\neg$. OPT queries carry implicit negation in that they can be expressed as a logic formula in the following form: q_1 OPT $q_2 = (q_1 \wedge q_2) \vee (q_1 \wedge \neg q_2)$. Unfortunately, this formula can be reduced to just q_1 which is not the intended semantics of the operator. Hence, we need another approach in order to correctly encode this operator. To do so, we rely on the interpretation given below:

$$q_1 \ \mathsf{OPT} \ q_2 = \begin{cases} q_1 \ \mathsf{AND} \ q_2 & \text{if } \rho(q_2) \in G \\ q_1 & \text{if } \rho(q_2) \notin G \end{cases}$$

The above interpretation of OPT operator dictates that: q_1 OPT q_2 evaluates as q_1 AND q_2 if there exists a mapping ρ for q_2, otherwise it evaluates as q_1. Based on this, the μ-calculus encoding of the operator can be obtained. The formula $ew(q_1 \text{ AND } q_2)$ evaluates to all nodes S if q_2 exists in the graph, it evaluates to $\emptyset$ otherwise. Further, the function f translates query patterns into formulas recursively.

DEFINITION 7. *The encoding of a PSPARQL query pattern q is $\mathcal{A}(q)$ such that:*

$$
\begin{aligned}
\mathcal{A}(<x,e,z>) &= \mu X.(\langle \bar{s}\rangle \lambda(x) \wedge \mathcal{R}(\lambda(e),\lambda(z))) \\
&\qquad \vee \langle d\rangle X \vee \langle s\rangle X \vee \langle \bar{p}\rangle X \vee \langle \bar{o}\rangle X \\
\mathcal{A}(q_1 \textit{ AND } q_2) &= \mathcal{A}(q_1) \wedge \mathcal{A}(q_2) \\
\mathcal{A}(q_1 \textit{ UNION } q_2) &= \mathcal{A}(q_1) \vee \mathcal{A}(q_2) \\
\mathcal{A}(q_1 \textit{ MINUS } q_2) &= \mathcal{A}(q_1) \wedge \neg \mathcal{A}(q_2) \\
\mathcal{A}(q_1 \textit{ OPT } q_2) &= ew(f(q_1) \wedge \langle d\rangle f(q_2)) \wedge \mathcal{A}(q_1 \textit{ AND } q_2) \;\; \vee \\
&\qquad ew(f(q_1) \wedge \neg\langle d\rangle f(q_2)) \wedge \mathcal{A}(q_1) \\
ew(\varphi) &= \mu X.\varphi \vee \langle d\rangle X \vee \langle s\rangle X \vee \langle \bar{p}\rangle X \vee \langle \bar{o}\rangle X \\
f(<x,e,z>) &= \langle \bar{s}\rangle \lambda(x) \wedge \mathcal{R}(\lambda(e),\lambda(z)) \\
f(q_1 \textit{ AND } q_2) &= f(q_1) \wedge f(q_2) \\
f(q_1 \textit{ UNION } q_2) &= f(q_1) \vee f(q_2) \\
f(q_1 \textit{ OPT } q_2) &= f(q_1)
\end{aligned}
$$

In definition 7, a regular expression encoding function $\mathcal{R}$ is introduced. It takes two arguments (the predicate which is a regular expression pattern, and the object of a triple). This function is inductively defined as follows:

DEFINITION 8. *Regular expressions are encoded recursively using the function $\mathcal{R}$, detailed below:*

$$
\begin{aligned}
\mathcal{R}(uri,y) &= \langle p\rangle uri \;\wedge \langle o\rangle y \\
\mathcal{R}(x,y) &= \langle p\rangle x \wedge \langle o\rangle y \\
\mathcal{R}(e_1 \mid e_2,y) &= (\mathcal{R}(e_1,y) \vee \mathcal{R}(e_2,y)) \\
\mathcal{R}(e_1.e_2,y) &= \mathcal{R}(e_1,\langle s\rangle \mathcal{R}(e_2,y)) \\
\mathcal{R}(e^+,y) &= \mu X.\mathcal{R}(e,y) \vee \mathcal{R}(e,\langle s\rangle X) \\
\mathcal{R}(e^*,y) &= \mathcal{R}(e^+,y) \vee \langle \bar{s}\rangle y
\end{aligned}
$$

EXAMPLE 5. *This example shows a recursive encoding of query 1 of Example 3 as a μ-calculus formula.*

$$
\begin{aligned}
\mathcal{A}(q_2) =& A(<x,translated.type,Poem> \textit{ UNION } <x,wrote,l>) \\
=& \; A(<x,translated.type,Poem>) \vee \mathcal{A}(<x,wrote,l>) \\
=& \; \mu X.(\langle \bar{s}\rangle \lambda(x) \wedge \mathcal{R}(\lambda(translated\ .\ type),\lambda(Poem))) \\
& \qquad \vee \langle d\rangle X \vee \langle s\rangle X \vee \langle \bar{p}\rangle X \vee \langle \bar{o}\rangle X \vee \\
& \; \mu X.(\langle \bar{s}\rangle \lambda(x) \wedge \mathcal{R}(\lambda(wrote),\lambda(l))) \\
& \qquad \vee \langle d\rangle X \vee \langle s\rangle X \vee \langle \bar{p}\rangle X \vee \langle \bar{o}\rangle X \\
=& \; \mu X.(\langle \bar{s}\rangle v_x \wedge \langle p\rangle translated \wedge \langle o\rangle\langle s\rangle(\langle p\rangle type \\
& \quad \wedge \langle o\rangle Poem)) \vee \langle d\rangle X \vee \langle s\rangle X \vee \langle \bar{p}\rangle X \vee \langle \bar{o}\rangle X \vee \\
& \; \mu X.(\langle \bar{s}\rangle v_x \wedge \langle p\rangle wrote \wedge \langle o\rangle v_l) \\
& \qquad \vee \langle d\rangle X \vee \langle s\rangle X \vee \langle \bar{p}\rangle X \vee \langle \bar{o}\rangle X
\end{aligned}
$$

4. REDUCING QUERY CONTAINMENT TO UNSATISFIABILITY

In this section, we address the problem of query containment, $q_1\{\overrightarrow{w}\} \sqsubseteq q_2\{\overrightarrow{w}\}$, by reducing it to the problem of unsatisfiability in the logic. The first theorem expressed the correctness and completeness of the encodings.

THEOREM 2. *For any graph G and PSPARQL query $q\{\vec{w}\}$,*
$\forall \rho.(\rho \in [\![q\{\vec{w}\}]\!]_G \iff [\![\mathcal{A}(\rho(q\{\vec{w}\}))]\!]^{\sigma(G)} \neq \emptyset)$

PROOF. This is proved inductively:
(Base case) The base case is proved for triple patterns containing regular expression patterns of the form: $y \mid e_1.e_2 \mid e^+$. First, when $q\{x,y,z\} = <x,y,z>$.
$\forall G.\forall \rho.(\rho \in [\![<x,y,z>]\!]_G \iff [\![\mathcal{A}(\rho(<x,y,z>))]\!]^{\sigma(G)} \neq \emptyset)$
($\Rightarrow$) If $\rho \in [\![<x,y,z>]\!]_G$, then $<\rho(x),\rho(y),\rho(z)> \in G$. Hence $\sigma(G) = (S,R,L)$ contains:

- $t \in S'', n^{\rho(x)}, n^{\rho(y)}, n^{\rho(z)} \in S'$,
- $(n^{\rho(x)},t) \in R(s), (t,n^{\rho(y)}) \in R(p), (t,n^{\rho(z)}) \in R(o)$, and
- $L(\rho(x)) = n^{\rho(x)}, L(\rho(y)) = n^{\rho(y)}, L(\rho(z)) = n^{\rho(z)}$.

$<\rho(x),\rho(y),\rho(z)>$ can be encoded as a μ-calculus formula. This encoding when evaluated over the transition system $\sigma(G)$ is non empty because if the triple exists in G, it also exists in the transition system. Consequently,

$$
\begin{aligned}
\Rightarrow\;& [\![\mu X.(\langle \bar{s}\rangle \lambda(\rho(x)) \wedge \langle p\rangle \lambda(\rho(y)) \wedge \langle o\rangle \lambda(\rho(z))) \\
& \vee \langle d\rangle X \vee \langle s\rangle X \vee \langle \bar{p}\rangle X \vee \langle \bar{o}\rangle X]\!]^{\sigma(G)} \neq \emptyset \\
\Rightarrow\;& [\![\mathcal{A}(<\rho(x),\rho(y),\rho(z)>)]\!]^{\sigma(G)} \neq \emptyset \\
\Rightarrow\;& [\![\mathcal{A}(\rho(<x,y,z>))]\!]^{\sigma(G)} \neq \emptyset
\end{aligned}
$$

($\Leftarrow$) $[\![\mathcal{A}(\rho(<x,y,z>))]\!]^{\sigma(G)} \neq \emptyset$ entails that there is a state $n^t \in S''$ and $n^{\rho(x)}$, $n^{\rho(y)}$, $n^{\rho(z)} \in S'$, such that $\langle n^{\rho(x)}, n_t\rangle \in R(s)$, $\langle n_t, n^{\rho(y)}\rangle \in R(p)$, and $\langle n_t, n^{\rho(z)}\rangle \in R(o)$ and $n^{\rho(x)} \in L(\lambda(\rho(x)))$, $n^{\rho(y)} \in L(\lambda(\rho(y)))$ and $n^{\rho(z)} \in L(\lambda(\rho(z)))$. Since the transition system $\sigma(G)$ is the encoding of an RDF graph G, this means that $<\lambda(\rho(x)),\lambda(\rho(y)),\lambda(\rho(z))> \in G$. Subsequently, $[\![<x,y,z>]\!]_G \neq \emptyset$, thus there exists a mapping ρ such that $\rho \in [\![<x,y,z>]\!]_G$. This concludes the proof for the base case.

(Inductive case) Query patterns: q_1 AND q_2 | q_1 UNION q_2 | q_1 OPT q_2 | q_1 MINUS q_2. We provide the transcriptions of AND and OPT. The proof of UNION and MINUS follows similarly. First, consider when $q\{\vec{w}\} = q_1$ AND q_2.
$\rho \in [\![q_1 \text{ AND } q_2]\!]_G$

$$
\begin{aligned}
\Leftrightarrow\;& \rho \in [\![q_1]\!]_G \text{ and } \rho \in [\![q_2]\!]_G \\
\Leftrightarrow\;& [\![\mathcal{A}(\rho(q_1))]\!]^{\sigma(G)} \neq \emptyset \text{ and } [\![\mathcal{A}(\rho(q_2))]\!]^{\sigma(G)} \neq \emptyset \\
& \text{by induction hypothesis.} \\
\Leftrightarrow\;& [\![\mathcal{A}(\rho(q_1)) \wedge \mathcal{A}(\rho(q_2))]\!]^{\sigma(G)} \neq \emptyset \;\; * \\
\Leftrightarrow\;& [\![\mathcal{A}(\rho(q_1) \text{ AND } \rho(q_2))]\!]^{\sigma(G)} \neq \emptyset \\
\Leftrightarrow\;& [\![\mathcal{A}(\rho(q_1 \text{ AND } q_2))]\!]^{\sigma(G)} \neq \emptyset
\end{aligned}
$$

* this formula remains satisfiable because there exists ϕ a satisfiable subformula of $\mathcal{A}(\rho(q_1))$ and $\mathcal{A}(\rho(q_2))$ in $\sigma(G)$. In fact, ϕ is a nominal obtained by encoding a variable which is common to both q_1 and q_2.

Inductive case for OPT i.e., when $q\{\vec{w}\} = q_1$ OPT q_2.

$\rho \in [\![q_1 \text{ OPT } q_2]\!]_G$

$\Leftrightarrow \rho \in ([\![q_1]\!]_G \bowtie [\![q_2]\!]_G)$ or $\rho \in ([\![q_1]\!]_G \setminus [\![q_2]\!]_G)$

$\Leftrightarrow \rho \in ([\![q_1]\!]_G \bowtie [\![q_2]\!]_G)$ if $\rho(q_2) \in G$ or

$\rho \in [\![q_1]\!]_G$ if $\rho(q_2) \notin G$

$\Leftrightarrow [\![\mathcal{A}(\rho(q_1 \text{ AND } q_2))]\!]^{\sigma(G)} \neq \emptyset$ if $\rho(q_2) \in G$ or

$[\![\mathcal{A}(\rho(q_1))]\!]^{\sigma(G)} \neq \emptyset$ if $\rho(q_2) \notin G$ by induction hypothesis.

$\Leftrightarrow [\![\mathcal{A}(\rho(q_1 \text{ AND } q_2))]\!]^{\sigma(G)} \neq \emptyset \cap [\![ew(q_1 \text{ AND } q_2)]\!]^{\sigma(G)} = S \cup$

$[\![\mathcal{A}(\rho(q_1))]\!]^{\sigma(G)} \neq \emptyset \cap [\![ew(q_1 \text{ AND } \neg q_2)]\!]^{\sigma(G)} = S$ *

$\Leftrightarrow [\![\mathcal{A}(\rho(q_1 \text{ AND } q_2)) \wedge ew(q_1 \text{ AND } q_2)]\!]^{\sigma(G)} \neq \emptyset \cup$

$[\![\mathcal{A}(\rho(q_1)) \wedge ew(q_1 \text{ AND } \neg q_2)]\!]^{\sigma(G)} \neq \emptyset$

$\Leftrightarrow [\![\mathcal{A}(\rho(q_1 \text{ OPT } q_2))]\!]^{\sigma(G)} \neq \emptyset$

* here we added a formula which evaluates to the entire set of states S if both $q_1$and q_2 are found in the transition system. Hence, the first part of the disjunction evaluates to a non empty result whereas if q_2 does not exist in the transition system, the second part of the disjunction is non empty. Thereby, retaining the semantics of the OPT operator. □

A detailed version of this proof is published in [9]. This theorem is the key to reduce query containment to satisfiability. For a proper translation of the query encodings, we use a variable renaming function $\phi_q^{\vec{w}}$ which renames all variables in q, but the distinguished variables in $\vec{w}$, to independent variables.

THEOREM 3. *Given PSPARQL queries $q_1\{\vec{w}\}$ and $q_2\{\vec{w}\}$, $q_1\{\vec{w}\} \sqsubseteq q_2\{\vec{w}\}$ iff $\mathcal{A}(q_1) \wedge \neg\phi_{q_1}^{\vec{w}}(\mathcal{A}(q_2)) \wedge \varphi_r$ is unsatisfiable.*

PROOF. It can be proved as follows:
$q_1\{\vec{w}\} \sqsubseteq q_2\{\vec{w}\}$

$\Leftrightarrow \forall G.[\![q_1\{\vec{w}\}]\!]_G \subseteq [\![q_2\{\vec{w}\}]\!]_G$

$\Leftrightarrow \forall G.\forall\rho.\ (\rho \in [\![q_1]\!]_G \Rightarrow \rho \in [\![q_2]\!]_G)$

$\Leftrightarrow \forall G.\forall\rho.([\![\mathcal{A}(\rho(q_1))]\!]^{\sigma(G)} \neq \emptyset) \Rightarrow ([\![\mathcal{A}(\rho(q_2))]\!]^{\sigma(G)} \neq \emptyset)$
by Theorem 2

$\Leftrightarrow \forall G.\forall\rho.([\![\mathcal{A}(\rho(q_1))]\!]^{\sigma(G)} \neq \emptyset) \Rightarrow ([\![\phi_{q_1}^{w}(\mathcal{A}(\rho(q_2)))]\!]^{\sigma(G)} \neq \emptyset)$
by transparent renaming

$\Leftrightarrow \forall G.\forall\rho[\![\neg\mathcal{A}(\rho(q_1)) \vee \phi_{q_1}^{w}(\mathcal{A}(\rho(q_2)))]\!]^{\sigma(G)} \neq \emptyset$

$\Leftrightarrow \forall G.\forall\rho[\![\mathcal{A}(\rho(q_1)) \wedge \neg\phi_{q_1}^{w}(\mathcal{A}(\rho(q_2)))]\!]^{\sigma(G)} = \emptyset$

$\Leftrightarrow \forall G.[\![\mathcal{A}(q_1) \wedge \neg\phi_{q_1}^{w}(\mathcal{A}(q_2))]\!]^{\sigma(G)} = \emptyset$ *

$\Leftrightarrow \forall K.[\![\mathcal{A}(q_1) \wedge \neg\phi_{q_1}^{w}(\mathcal{A}(q_2)) \wedge \varphi_r]\!]^{K} = \emptyset$ by Proposition 1

$\Leftrightarrow \mathcal{A}(q_1) \wedge \neg\phi_{q_1}^{w}(\mathcal{A}(q_2)) \wedge \varphi_r$ unsatisfiable

□

(*) From Theorem 2, it follows that if there exists a set of mappings for an evaluation of a query over a graph, then the encoding of the query over the transition system obtained from the graph is satisfiable.

5. QUERY CONTAINMENT OVER RDFS

The current version of SPARQL cannot query RDFS however it can be done by PSPARQL using complex regular expressions. RDFS graphs can be queried either by computing their closure or rewriting the queries as done in [2]. In Table 1 a set of rules are given that can be used to infer new triples from an RDF graph using RDF Schema. These rules are used in [1] to rewrite queries so that during querying the inferred triples can be included in the query result set.

Containment over RDF Schema can be done by first rewriting queries using schema assertions (or RDFS rules from Table 1) and then reducing the encoding of the rewriting to unsatisfiability test. The rewriting is done using regular expressions as explained in the following definition.

DEFINITION 9. *Given a query q, a rewriting function τ produces its rewriting by translating each triple pattern $t \in q$ into a query.*
$\tau : t \rightarrow q'$
$\tau(<s, sc, o>) = \ <s, sc^+, o>$
$\tau(<s, sp, o>) = \ <s, sp^+, o>$
$\tau(<s, p, o>) = \ <s, x, o>$ *AND* $<x, sp^*, p>$
$p \notin \{sc, sp, type\}$
$\tau(<s, type, o>) = \ <s, type.sc^*, o>$ *UNION* $(<s, p, y>$ *AND* $<p, sp^*, q>$ *AND* $<q, dom.sc^*, o>)$ *UNION* $(<y, p, s>$ *AND* $<p, sp^*, q>$ *AND* $<q, range.sc^*, o>)$
$\tau(<s, x, o>) = \ <s, x, o>$ *where x is a variable*

DEFINITION 10 (CONTAINMENT OVER RDFS). *Given queries q_1 and q_2 with the same arity and their respective rewritings q_1' and q_2'. $q_1 \sqsubseteq q_2$ over RDFS entailment iff $q_1' \sqsubseteq q_2'$ iff $\mathcal{A}(q_1') \wedge \mathcal{A}(q_2') \wedge \varphi_r$ is unsatisfiable.*

5.1 Complexity

Our translation of the query containment problem does not involve duplication of logical formulas of variable size, except for the OPT operator. Therefore, if we omit the OPT operator, the translation produces a logical formula of linear-size in terms of the size of the queries. Thus, We linearly reduced the problem of OPT-free query containment to unsatisfiability of a μ-calculus formula.

PROPOSITION 4. *Query containment for OPT-free queries can be solved in a time of $2^{\mathcal{O}(n^2 log\ n)}$ where $n = |\mathcal{A}(q_1)| + |\mathcal{A}(q_2)|$ is the size of the formula, and $\mathcal{A}(q_1)$ and $\mathcal{A}(q_2)$ denote the encodings of queries q_1 and q_2.*

The translation of OPT query patterns produces duplicates. Therefore, the size of the translated logical formula is exponential in terms of the size of the original queries.

PROPOSITION 5. *Query containment can be solved in a time of 2EXPTIME for queries containing OPT query patterns.*

In another note, the EXPTIME complexity is only an upper bound for containment.

5.2 Experimentation

In order to experiment with the proposed approach, the μ-calculus satisfiability solver from [23] is used to test containment and equivalence among different queries. A set of queries are tested for their containment and equivalence having running times between 190ms and 515ms. In fact, the running time is dependent on the processor speed and the size of the queries. Note that, queries are encoded manually. However, automatic encoding of queries as μ-calculus formulas is under way by using the Jena SPARQL API [1].

[1]http://jena.sourceforge.net/ARQ/

6. RELATED WORK

Query optimization has been the subject of an important research effort for many types of query languages, with the common goal of speeding up query processing. To the best of our knowledge, so far the problem of SPARQL with path query optimization has not been addressed. However, the works found in [21, 12, 19] considered the problem of SPARQL query optimization. So, the present work can be used to prove the correctness of query rewriting techniques.

An early formalization of RDF(S) graphs has been presented in [13], in which the complexity of query evaluation and containment is also studied. The authors investigate a Datalog-style, rule-based query language for RDF(S) graphs. In particular, they establish an NP-completeness result for query containment over simple RDF graphs. The work found in [20] provides algorithms for the containment and minimization of RDF(S) query patterns utilizing concept and property hierarchies for the query language RQL (RDF Query Language). The NP-completeness is established for query containment concerning conjunctive and union of conjunctive queries. Query containment is studied in [4] based on an encoding in propositional dynamic logic with converse (CPDL). They establish an upper bound 2EXPTIME complexity for containment of union of conjunctive queries under description logic constraints. Our work is similar in spirit, in the sense that the μ-calculus is a logic that subsumes CPDL and may open the way for extensions of the query language. In particular, we consider the OPT operator, previously overlooked, and regular graph patterns including paths of arbitrary length.

Most notably and closely related results on query containment come from the study of regular path queries (RPQs) [5]. The difference between [5] and our work lies in the features supported by the languages. While RPQs in [5] support backward navigation and conjunction, PSPARQL supports variables in paths, union, and negation of queries (as implicit negation is carried by the query operator OPT).

Conjunctive RPQs have been studied in [10, 7] where an EXPSPACE algorithm for query containment is proved. On the other hand, containment of conjunctive RPQs with inverse have an EXPSPACE worst case complexity [5]. Most recently, containment of RPQs under description logic constraints have been studied in [6], and it has been show that the problem is 2EXPTIME complete. Furthermore, containment has also been addressed for various forms of recursive queries over graph databases, i.e. databases consisting of binary relations only [3]. In this setting, which is receiving increased attention the basic querying mechanism is (two-way) regular path queries (2RPQs). These queries ask for all pairs of objects connected by a path conforming to a regular language over the binary relations, and thus support a restricted form of recursion. Containment over this kind of queries is shown to be undecidable [3].

7. CONCLUSIONS

In this paper we addressed query containment of SPARQL queries with paths. We took a similar approach to [11] that established the optimal complexity for XPath query containment. The problem of PSPARQL query containment has been reduced to satisfiability test in μ-calculus. For that purpose, we encoded RDF graphs as transition systems and PSPARQL queries as formulas. The reduction is proved to be sound and complete and the problem is shown to be EXPTIME. In addition, we implemented the proposed approach via an encoding using the μ-calculus solver of [23] and this demonstrated the effectiveness of the encoding.

Paths are included in the new version of SPARQL[2] which is currently under standardization by W3C hence our results are a step towards query containment for SPARQL 1.1. The proposed encodings are not specific to PSPARQL. The same RDF encoding can be used for SPARQL query containment. Further, this work is also relevant for determining containment of SPARQL queries under RDFS entailment regime.

8. REFERENCES

[1] F. Alkhateeb. *Querying RDF (S) with regular expressions*. PhD thesis, Université Joseph Fourier, 2008. thesis.

[2] F. Alkhateeb, J.-F. Baget, and J. Euzenat. Extending SPARQL with regular expression patterns (for querying RDF). *J. Web Semantics*, 7(2):57–73, 2009.

[3] P. Barceló, C. Hurtado, L. Libkin, and P. Wood. Expressive languages for path queries over graph-structured data. pages 3–14. ACM, 2010.

[4] 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.

[5] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of Conjunctive Regular Path Queries with Inverse. In *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 176–185, 2000.

[6] D. Calvanese, M. Ortiz, and M. Simkus. Containment of regular path queries under description logic constraints. In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011)*, 2011. To appear.

[7] D. Calvanese and R. Rosati. Answering Recursive Queries under Keys and Foreign Keys is Undecidable. In *Proc. of the 10th Int. Workshop on Knowledge Representation meets Databases (KRDB 2003)*, volume 79, pages 3–14, 2003.

[8] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*, pages 77–90, 1977.

[9] M. W. Chekol, J. Euzenat, P. Genevès, and N. Layaïda. PSPARQL query containment. Research report 7641, June 2011. http://hal.inria.fr/inria-00598819/PDF/RR-7641.pdf.

[10] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. PODS '09, pages 139–148. ACM, 1998.

[11] P. Genevès, N. Layaïda, and A. Schmitt. Efficient Static Analysis of XML Paths and Types. PLDI '07, pages 342–351, New York, NY, USA, 2007.

[12] J. Groppe, S. Groppe, and J. Kolbaum. Optimization of SPARQL by using coreSPARQL. In *ICEIS (1)*, pages 107–112, 2009.

[13] C. Gutierrez, C. Hurtado, and A. O. Mendelzon. Foundations of Semantic Web Databases. PODS '04, pages 95–106, New York, NY, USA, 2004.

[14] P. Hayes. RDF Semantics. W3C Rec., 2004.

[2] http://www.w3.org/TR/sparql11-query

[15] Y. E. Ioannidis. Query Optimization. *ACM Comput. Surv.*, 28(1):121–123, 1996.

[16] D. Kozen. Results on the Propositional μ-Calculus. *Theor. Comp. Sci.*, 27:333–354, 1983.

[17] S. Muñoz, J. Pérez, and C. Gutierrez. Minimal Deductive Systems for RDF. volume 4519 of *LNCS*, pages 53–67, 2007.

[18] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Rec., 2008.

[19] M. Schmidt, M. Meier, and G. Lausen. Foundations of SPARQL Query Optimization. ICDT '10, pages 4–33, New York, NY, USA, 2010.

[20] G. Serfiotis, I. Koffina, V. Christophides, and V. Tannen. Containment and Minimization of RDF/S Query Patterns. In *The Semantic Web ISWC 2005*, volume 3729 of *LNCS*, pages 607–623, 2005.

[21] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds. SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation. In *Proc.*, WWW '08, pages 595–604, New York, NY, USA, 2008.

[22] Y. Tanabe, K. Takahashi, and M. Hagiya. A Decision Procedure for Alternation-Free Modal μ-calculi. In *Advances in Modal Logic*, pages 341–362, 2008.

[23] 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.