PSPARQL Query Containment
Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, Nabil Layaida

To cite this version:
Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, Nabil Layaida. PSPARQL Query Containment. DBPL 2011 The 13th International Symposium on Database Programming Languages, Aug 2011, Seattle, United States. 8 p. hal-00619342

HAL Id: hal-00619342
https://hal.science/hal-00619342
Submitted on 6 Sep 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 Layaida  
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 $\mu$-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 $\mu$-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 $\mu$-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 $\mu$-calculus (§2), we show how to translate RDF graphs into transition systems (§3.1) and PSPARQL queries into $\mu$-calculus formulas (§3.2). Therefore, query containment in PSPARQL can be reduced to unsatisfiability test in $\mu$-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, \( UBL = U \cup B \cup L \). A triple of the form \(<s,p,o>\) \( UBL \times U \times UBL \) is called an \( RDF \) triple. \( s \) is the \textit{subject}, \( p \) is the \textit{predicate}, and \( o \) is the \textit{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.

\textbf{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 .
- Poe wrote thegoldbug .
- Baudelaire translated thegoldbug .
- Poe wrote thegoldbug .
- Mallarmé wrote .b .

\( RDF \) has a model theoretic semantics [14].

\subsection{RDFS}

\( RDFS \) (RDF Schema) [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 \( pdf \) [17] which contains the minimal vocabularies, \( pdf = \{ \text{sp, sc, type, dom, range} \} \). Where \( sp \) denotes \textit{subproperty} relation, \( sc \) is \textit{subclass}, and \( dom \) is \textit{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.

<table>
<thead>
<tr>
<th>Subclass</th>
<th>Subproperty</th>
<th>Typing</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;s,sc,sc&gt;)</td>
<td>(&lt;s,sc,sc&gt;)</td>
<td>(&lt;s,sc,sc&gt;)</td>
</tr>
<tr>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
</tr>
<tr>
<td>(&lt;s,dom,dom&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
</tr>
<tr>
<td>(&lt;s,sc,sc&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
</tr>
<tr>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
</tr>
<tr>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
</tr>
<tr>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
</tr>
<tr>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
<td>(&lt;s,sp,sp&gt;)</td>
</tr>
</tbody>
</table>

\textbf{Implicit Typing}

| \(<s,sp,sp>\) | \(<s,sp,sp>\) | \(<s,sp,sp>\) |
| \(<s,sp,sp>\) | \(<s,sp,sp>\) | \(<s,sp,sp>\) |

\textbf{Table 1: RDFS inference rules}

\textbf{Example 2.} This example shows the usage of \( RDFS \) inference rules, consider the graph \(<\text{John, type, Student}, \ <\text{Student, sc, Person}>\). By applying \textit{Subclass} rule, rule [2], it can be deduced that \(<\text{John, type, Person}>\).

\subsection{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).

\subsubsection{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 \textit{string/word} is a finite sequence of symbols from the alphabet \( \Sigma \). A \textit{language} \( L \) is a set of words over \( \Sigma \) which is a subset of \( \Sigma^* \), i.e., \( L(\Sigma) \subseteq \Sigma^* \).

A word can be either empty \( e \) 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 \( \Sigma \), then \( A.B \) is a word over the same alphabet defined as: \( A.B = a_1...a_n b_1...b_m \).

\textbf{Definition 1 (Regular expression pattern).} Given an alphabet \( \Sigma \) and a set of variables \( V \), a regular expression \( R(S, V) \) can be constructed inductively as follows:

\( e := uri \mid x \mid e_1 \ e_2 \mid e^+ \mid e^* \)

Where \( e \in R(\Sigma, V) \) and \( x \) denotes a variable, \( e_1 \ e_2 \) denotes disjunction, \( e^+ \) denotes concatenation, \( e^\star \) 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 \( (U \cup V) \) can be used to define a language over the alphabet \( U \cup V \).

\subsubsection{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 \( t \in UBL \times UBL \times UBL \) with \( V \) is a set of variables disjoint from UBL, is called a triple pattern. Triple patterns grouped together using connectives \( \text{AND, UNION, OPT} \) form graph patterns (a.k.a query patterns). We use an abstract syntax that can be easily translated into \( \mu \)-calculus.

\textbf{Definition 2 (Query Pattern).} A PSPARQL query pattern \( q \) is inductively defined as follows:

\( q = t \cup UBL \times R(U, V) \times UBL \)

\( \mid q_1 \ \text{AND} \ q_2 \mid q_1 \ \text{UNION} \ q_2 \mid q_1 \ \text{OPT} \ q_2 \mid q_1 \ \text{MINUS} \ q_2 \)

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

\textbf{Definition 3.} A PSPARQL \textit{SELECT} query is a query of the form \( q\{w\} \) where \( w \) is a tuple of variables in \( V \) which are called distinguished variables, and \( q \) is a query pattern.

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

\begin{verbatim}
SELECT ?x WHERE {
  ?x (translated / wrote) . type Poem .
}
\end{verbatim}

\begin{verbatim}
SELECT ?x WHERE {
  { ?x (translated . type) Poem }
  UNION
  { ?x wrote ?l .}
}
\end{verbatim}
2.3.3 PSPARQL Semantics

The semantics of PSPARQL queries is given by a partial mapping function \( \rho : V \rightarrow UBL \). The domain of \( \rho \), \( \text{dom}(\rho) \), is the subset of \( V \) on which \( \rho \) is defined. Two mappings \( \rho_1 \) and \( \rho_2 \) are said to be compatible if \( \forall x \in \text{dom}(\rho_1) \cap \text{dom}(\rho_2), \rho_1(x) = \rho_2(x) \). 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:

\[ M_1 \cup M_2 = \{ \rho \cup \rho' \mid \rho \in M_1, \rho' \in M_2 \} \]

\[ M_1 \setminus M_2 = \{ \rho \cup \rho' \mid \rho \in M_1, \rho' \in M_2, \rho \neq \rho' \} \]

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

\[ \llangle x, \text{uri}, y \rrangle_G = \{ \rho \mid \llangle \rho(x), \rho(\text{uri}), \rho(y) \rrangle \in G \} \]

\[ \llangle x, z, y \rrangle_G = \{ \rho \mid \llangle \rho(x), \rho(z), \rho(y) \rrangle \in G \} \]

\[ \llangle x, e \cdot e', y \rrangle_G = \{ \llangle \llangle x, e, y \rrangle_G \cup \llangle x, e', y \rrangle_G \} \]

\[ \llangle x, e \cdot e', y \rrangle_G = \{ \llangle \llangle x, e, m \rrangle_G \cup \llangle n, e', y \rrangle_G \} \]

\[ \llangle x, e^+, y \rrangle_G = \{ \rho \mid \exists n_0, n_1, \ldots, n_k \in G \text{ such that } n_0 = \rho(x), n_1 = \rho(y) \} \]

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

\[ [\llangle q \rrangle_G : q \rightarrow 2^{V \times UBL}] \]

\[ \llangle q \land q_2 \rrangle_G = [\llangle q \rrangle_G] \cap [\llangle q_2 \rrangle_G] \]

\[ \llangle q \lor q_2 \rrangle_G = [\llangle q \rrangle_G] \cup [\llangle q_2 \rrangle_G] \]

\[ \llangle q \ominus q_2 \rrangle_G = ([\llangle q \rrangle_G] \cap [\llangle q_2 \rrangle_G]) \cup ([\llangle q \rrangle_G] \setminus [\llangle q_2 \rrangle_G]) \]

3.1 Encoding RDF graphs as Transition Systems

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

3.1.1 \( \mu \)-calculus

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

The syntax of the \( \mu \)-calculus is composed of countable sets of atomic propositions \( AP \), a set of nominals \( Nom \), a set of variables \( Var \), and a set of programs \( Prog \) for navigating in graphs. A \( \mu \)-calculus formula, \( \varphi \), can be defined inductively as follows:

\[ \varphi ::= T \mid \perp \mid p \mid X \mid \neg \varphi \mid \varphi \lor \psi \mid \varphi \land \psi \mid (\alpha)\varphi \mid [\alpha]\varphi \mid \mu X\varphi \mid \nu X\varphi \]

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

The semantics of the \( \mu \)-calculus is given over a transition system, \( \mathcal{K} = (S, R, L) \) where \( S \) is a non-empty set of nodes, \( R : Prog \rightarrow 2^{S \times S} \) is the transition function, and \( L : AP \rightarrow 2 \) 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 \( R(\bar{\alpha}) = \{ (s', s) \mid (s, s') \in R(\alpha) \} \). Besides, a valuation function \( V : Var \rightarrow S \) 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 \( [v]_X^H \). The semantics of basic \( \mu \)-calculus formula is defined as follows:

\[ [p]_X^H = L(p), p \in AP \cup Nom \]

\[ [X]_X^H = V(X), X \in Var \]

\[ [-\varphi]_X^H = S \setminus [\varphi]_X^H \]

\[ [(\varphi \land \psi)]_X^H = [\varphi]_X^H \cap [\psi]_X^H \]

\[ [(\varphi \lor \psi)]_X^H = [\varphi]_X^H \cup [\psi]_X^H \]

\[ [(\alpha)\varphi]_X^H = \{ s \in S \mid \exists s' \in S.(s, s') \in R(\alpha) \wedge s' \in [\varphi]_X^H \} \]

\[ [[\alpha]\varphi]_X^H = \{ s \in S \mid \forall s' \in S.(s, s') \in R(\alpha) \Rightarrow s' \in [\varphi]_X^H \} \]

\[ [\mu X\varphi]_X^H = \bigcup \{ S' \subseteq S \mid [\varphi]_X^{V[X/S']} \subseteq S' \} \]

\[ [\nu X\varphi]_X^H = \bigcap \{ S' \subseteq S \mid [\varphi]_X^{V[X/S']} \subseteq S' \} \]

Once providing the syntax and semantics of \( \mu \)-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, \( G \subseteq UBL \times U \times UBL \), the transition system associated \( G, \sigma(G) = (S, R, L) \) over \( AP = UBL \cup \{ s', s'' \} \), is such that:

"
This last constraint can be expressed in the following way:

\[ S = S' \cup S'' \text{ with } S' \text{ and } S'' \text{ the smallest sets such that } \forall u \in U_G, \exists n^u \in S', \forall b \in B_G, \exists n^b \in S', \forall l \in L_G, \exists n^l \in S' \text{ and } \forall t \in G, \exists n^t \in S'', \]

\[ \forall t = s \preceq p \preceq o \in G, \langle n^s, n^t \rangle \in R(s), \langle n^t, n^o \rangle \in R(p), \text{ and } \langle n^p, n^r \rangle \in R(o), \]

\[ L : \text{UBL} \to 2^S; \forall u \in U_G, L(u) = \{ n^u \}, \forall b \in B_G, L(b) = \{ S' \}, L(s') = \{ S' \}, \forall l \in L_G, L(l) = \{ n^l \} \text{ and } L(s'') = S'', \]

\[ \forall n^l, n^t \in S'', \langle n^l, n^t \rangle \in R(d). \]

The program \( d \) is introduced to render each triple accessible to the others and thus facilitate the encoding of queries. The function \( \sigma \) 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 if it has 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.

![Diagram](image_url)

**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: \( \text{Prog} = \{ s, p, o, d, s, \bar{p}, \bar{o}, \bar{d} \} \) (note that \( d = \bar{d} \)).
- A model must be a restricted transition system.

This last constraint can be expressed in the \( \mu \)-calculus as follows:

**Proposition 1 (RDF restriction on transition systems)**

A formula \( \varphi \) is satisfied by some restricted transition system if and only if \( \varphi \land \varphi_r \) is satisfiable by some transition system, i.e., \( \exists K, [\varphi]_K^T \neq \emptyset \iff \exists K [\varphi \land \varphi_r]_K^T \neq \emptyset \), where:

\[ \varphi_r = \nu X. \theta \land \kappa \land \neg (d) \top \lor (d) X \]

in which \( \theta = \langle \bar{s} \rangle s' \land (p) s' \land (o) s' \land \neg (\bar{p}) \top \land \neg (\bar{o}) \top \) and \( \kappa = [\bar{s}] \xi \land [p] \xi \land [o] \xi \) with

\[ \xi = \langle \neg (s) \top \land \neg (o) \top \land \neg (p) \top \land \neg (d) \top \land \neg (\bar{d}) \top \land \neg (\bar{p}) \top \land \neg (\bar{o}) \top \land \neg (\bar{p}) s'. \]

The formula \( \varphi_r \) ensures that \( \theta \) and \( \kappa \) hold in every node reachable by a \( d \) edge, i.e. in every \( S'' \) node. The formula \( \varphi \) forces each \( S'' \) node to have a subject, predicate, and object. The formula \( \kappa \) 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 \( \exists K, [\varphi]_K^T \neq \emptyset \), since \( \varphi \) is satisfied by only transition systems, one gets \( [\varphi \land \varphi_r]_K^T \neq \emptyset \). Hence it follows that, \( \exists K, [\varphi]_K^T \neq \emptyset \) and \( [\varphi \land \varphi_r]_K^T \neq \emptyset \) which implies \( \exists K, [\varphi \land \varphi_r]_K^T \neq \emptyset \). From this, using the semantics of \( \mu \)-calculus formula, one obtains \( \exists K, [\varphi \land \varphi_r]_K^T \neq \emptyset \). Since a restricted transition system is also a transition system, \( K_r \subseteq K \), it follows that \( \exists K, [\varphi \land \varphi_r]_K^T \neq \emptyset \).

\((\Leftarrow)\) Assume that \( \exists K, [\varphi \land \varphi_r]_K^T \neq \emptyset \). We construct a restricted transition system \( K_r = (S_r, R_r, L_r) \) 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 \( \varphi \land \varphi_r \) is satisfied in \( K \). Suppose we have constructed a node \( n_r \) of \( S_r \). For \( j \in \{ s, p, o \} \), if there is \( n \in S \) with \( f(n_{j\downarrow}) \in R(j) \), then pick one such \( n \) and add a node \( n_j \) of \( S_r \) with \( f(n_j) = n \). 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. \( (s, s'), (s', p), (s', o) \) is valid throughout the graph. If there were node pairs outside of this structure, then \( \varphi_r \) will not be satisfied. Throught out the graph, \( \theta \) and \( \kappa \) 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, \( [\varphi \land \varphi_r]_K^T \neq \emptyset \)

To verify that \( [\varphi]_K^T \neq \emptyset \), it is enough to show \( [\varphi]_K^T \Rightarrow [\varphi]_K^T \) by induction on the structure of \( \varphi \). \( \square \)

### 3.2 Encoding SPARQL Queries as \( \mu \)-calculus Formulas

Queries are translated to \( \mu \)-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 \( \mu \) 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 this purpose, we use a function \( \lambda : \text{VUBL} \to \text{UBL} \) such that:

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

The function \( \lambda \) encodes queries inductively on the structure of query patterns. \textsc{AND} and \textsc{UNION} are replaced by boolean connectives \( \land \) and \( \lor \) respectively. The \textsc{MINUS} operator is translated as \( \land \) and \( \neg \). \textsc{OPT} queries carry implicit negation in that they can be expressed as a logic formula in the following form: \( q_1 \text{ OPT } q_2 = (q_1 \land \neg q_2) \lor (q_1 \land \neg q_2) \).

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:

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

The above interpretation of \textsc{OPT} operator dictates that: \( q_1 \text{ OPT } q_2 \) evaluates as \( q_1 \) if \( q_2 \) is empty otherwise it evaluates as \( q_1 \). Based on this, the \( \mu \)-calculus encoding of the operator can be obtained. The
formula \( ev(q_1 \text{ AND } q_2) \) 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 P SparQL query pattern \( q \) is \( A(q) \) such that:

\[
A(<x,e,z>) = \mu X.((\bar{s})(\lambda(x) \wedge R(\lambda(e),\lambda(z)))) \\
\wedge <d,X \wedge \langle \bar{p},X \wedge \langle \bar{o},X
\]

\[
A(q_1 \text{ AND } q_2) = A(q_1) \cap A(q_2) \\
A(q_1 \text{ UNION } q_2) = A(q_1) \cup A(q_2) \\
A(q_1 \text{ MINUS } q_2) = A(q_1) \setminus A(q_2) \\
A(q_1 \text{ OPT } q_2) = ev(f(q_1) \wedge (d)f(q_2)) \wedge A(q_1 \text{ AND } q_2) \wedge ev(f(q_1) \wedge \neg(d)f(q_2)) \wedge A(q_1)
\]

\[
ev(\varphi) = \mu X.\varphi \wedge <d,X \wedge \langle \bar{p},X \wedge \langle \bar{o},X
\]

f(<x,e,z>) = (\bar{s})(\lambda(x) \wedge R(\lambda(e),\lambda(z)))

f(q_1 \text{ AND } q_2) = f(q_1) \wedge f(q_2)

f(q_1 \text{ UNION } q_2) = f(q_1) \vee f(q_2)

f(q_1 \text{ OPT } q_2) = f(q_1)

In definition 7, a regular expression encoding function \( R \) 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 \( R \), detailed below:

\[
R(\text{uri}, y) = \langle \bar{p}\rangle y \\
R(x,y) = \langle \bar{p}\rangle x \wedge \langle \bar{o}\rangle y \\
R(e_1 \cdot e_2, y) = (R(e_1, y) \vee R(e_2, y)) \\
R(e_1, e_2) = R(e_1, (s)R(e_2, y)) \\
R(e^+, y) = \mu X.R(e, y) \vee R(e, (s)X) \\
R(e^+, y) = R(e^+, y) \vee \langle \bar{s}\rangle y
\]

**Example 5.** This example shows a recursive encoding of query 1 of Example 3 as a \( \mu \)-calculus formula.

\[
A(q_2) = A(<x, \text{translated.type}, \text{Poem}> \text{ UNION } <x, \text{wrote}, t>) = A(<x, \text{translated.type}, \text{Poem}> \wedge (A(<x, \text{wrote}, t>)) = \mu X.((\bar{s})(\lambda(e) \wedge R(\lambda(e),\lambda(z)))) \\
\wedge <d,X \wedge \langle \bar{p},X \wedge \langle \bar{o},X
\]

**4. Reducing Query Containment to Unsatisfiability**

In this section, we address the problem of query containment, \( q_1 \{ \bar{w} \} \subseteq q_2 \{ \bar{w} \} \), 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 P SparQL query \( q(\bar{w}) \),

\[
\forall \rho (\rho \in [q[\bar{w}]]) \Longleftrightarrow [A(\rho(q(\bar{w})))]^\sigma(G) \neq \emptyset
\]

**Proof.** This is proved inductively:

**Base case.** The base case is proved for triple patterns containing regular expression patterns of the form: \( \{ y \mid e_1 \cdot e_2 \cdot e^+ \cdot c^+ \} \). First, when \( q\{x,y,z\} = \langle x,y,z > \).

\[
\forall G \forall \rho (\rho \in [q\{x,y,z\}]) \Longleftrightarrow [A(\rho(q\{x,y,z\}))]^\sigma(G) \neq \emptyset
\]

\( (\Rightarrow) \) If \( \rho \in [q\{x,y,z\}] \), 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

- \( L(\rho(x)) = n^{\rho(y)}, L(\rho(y)) = n^{\rho(z)}, L(\rho(z)) = n^{\rho(o)} \).

\( \langle x,y,z,z > \rangle \) can be encoded as a \( \mu \)-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,

\[
A(\langle x,y,z,z > \rangle)^\sigma(G) \neq \emptyset
\]

\( \langle \Rightarrow \rangle \) If \( \langle x,y,z,z > \rangle \) entails that there is a state \( n^1 \in S'' \) and \( n^{\rho(x)}, n^{\rho(y)}, n^{\rho(z)} \in S' \), such that \( (n^{\rho(x)}, n^{\rho(y)}) \in R(s), (n^{\rho(y)}, n^{\rho(z)}) \in R(p), \) and \( (n^{\rho(z)}, n^{\rho(o)}) \in R(o) \) and \( n^{\rho(z)} \in L(\lambda(\rho(x))), n^{\rho(y)} \in L(\lambda(\rho(y))), \) and \( n^{\rho(o)} \in L(\lambda(\rho(z))) \). 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, \( [q\{x,y,z\}] \) does not exist, there exists a mapping \( \rho \) such that \( \rho \in [q\{x,y,z\}] \). This concludes the proof for the base case.

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

\[
\Longleftrightarrow \rho \in [q_1] \text{ and } \rho \in [q_2] \\
\Longleftrightarrow [A(\rho(q_1))]^\sigma(G) \neq \emptyset \text{ and } [A(\rho(q_2))]^\sigma(G) \neq \emptyset
\]

by induction hypothesis.

\[
\Longleftrightarrow [A(\rho(q_1))] \wedge [A(\rho(q_2))]^\sigma(G) \neq \emptyset
\]

\[
\Longleftrightarrow [A(\rho(q_1) \text{ AND } \rho(q_2))]^\sigma(G) \neq \emptyset
\]

* this formula remains satisfiable because there exists \( \phi \) a satisfiable subformula of \( A(\rho(q_1)) \) and \( A(\rho(q_2)) \) in \( \sigma(G) \). In fact, \( \phi \) 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 \( q(\bar{w}) = q_1 \text{ OPT } q_2 \).
Thereby, retaining the semantics 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 $2^{O(n^{2\log n})}$ 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 $2\text{EXPTIME}$ for queries containing OPT query patterns. In another note, the EXPTIME complexity is only an upper bound for containment.

5.2 Experimentalation

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 is 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.

1http://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 $\mu$-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 shown 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 $\mu$-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 $\mu$-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 RDF(S) entailment regime.

8. REFERENCES


http://www.w3.org/TR/sparql11-query


