SPARQL Query Containment Under SHI Axioms

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

SPARQL Query Containment under $\mathcal{SHI}$ Axioms

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

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

Pierre Genevès
CNRS
pierre.geneves@inria.fr

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

Abstract

SPARQL query containment under schema axioms is the problem of determining whether, for any RDF graph satisfying a given set of schema axioms, the answers to a query are contained in the answers of another query. This problem has major applications for verification and optimization of queries. In order to solve it, we rely on the μ-calculus. Firstly, we provide a mapping from RDF graphs into transition systems. Secondly, SPARQL queries and RDFS and $\mathcal{SHI}$ axioms are encoded into μ-calculus formulas. This allows us to reduce query containment and equivalence to satisfiability in the μ-calculus. Finally, we prove a double exponential upper bound for containment under $\mathcal{SHI}$ schema axioms.

Introduction

Access to semantic web data expressed in RDF (Resource Description Framework) may 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, in particular for extending the language and optimizing queries. Querying RDF graphs with SPARQL proceeds by matching graph patterns, i.e., triple patterns connected to form graphs by means of joins expressed using several occurrences of the same variable. Since queries in the semantic web are evaluated over huge RDF graphs, optimizations are necessary in order to find minimal queries to reduce the computational cost of query evaluation.

Query optimization aims at improving the runtime performance of query evaluation. Studies have contributed to query optimization using rewriting rules in particular in the relational algebra for databases (Ioannidis 1996; Chandra and Merlin 1977). Similar approaches have also been applied to SPARQL (Schmidt, Meier, and Lausen 2010). These works, however, need at some point to prove the correctness of query optimization, i.e., the semantics of the optimized query is the same as the original one. In other words, 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 vital role in optimization. It can be defined as determining if the result of one query is included in the result of another one for any RDF graph. In addition, query containment can be of independent interest for performing other optimizations. For example, if a query q is contained in q', then q can be evaluated on the materialized view of q' rather than on the whole data graph. To the best of our knowledge, the problem of SPARQL query containment (under a schema) has not been covered in the literature.

The aim of this paper is to address SPARQL query containment under a DL schema (the schema is formulated within the fragments of $\mathcal{SHIQ}$). We apply an approach which has already been successfully applied for XPath (Genevès, Layaïda, and Schmitt 2007). SPARQL is interpreted over graphs, hence we encode it in a graph logic, specifically the alternation-free fragment of the μ-calculus (Kozen 1983) with converse and nominals (Tanabe, Takahashi, and Hagiya 2008) interpreted over labeled transition systems. We show that this logic is powerful enough to deal with query containment for the fragment of SPARQL considered here in the presence of RDFS and $\mathcal{SHI}$ (schema) axioms. Furthermore, this logic admits exponential time decision procedures that is implemented in practice (Tanabe, Takahashi, and Hagiya 2008). Hence, our approach opens a way to use this implementation.

We introduce a translation of RDF graphs into transition systems and SPARQL queries and schema axioms into μ-calculus formulae. Then, we show how query containment in SPARQL can be reduced to unsatisfiability in the μ-calculus. We prove a double exponential upper bound for the problem. An additional benefit of using a μ-calculus encoding is to take advantage of fixpoints and modalities for encoding recursion. They allow to deal with natural extensions of SPARQL such as path queries (Alkhateeb, Baget, and Euzenat 2009) or queries modulo RDF Schema.

Preliminaries

RDF is a language used to express structured information on the Web as graphs. We present a compact formalization of RDF (Hayes 2004). Let U, B, and L be three disjoint infinite sets denoting the set of URIs (identifying a resource), blank nodes (denoting 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, $\mathrm{UBL} = U \cup B \cup L$. A triple of the form $(s, p, o) \in$

$\mathrm{UB} \times U \times \mathrm{UBL}$ is called an *RDF triple*. s is the *subject*, p is the *predicate*, and o is the *object* of the triple. Each triple can 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*. RDF has a model theoretic semantics (Hayes 2004).

Example 1 (RDF Graph)**.** *Consider 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) }

RDFS (RDF Schema) may be considered as a simple ontology language expressing subsumption relations between classes or properties (Hayes 2004). Technically, this is an RDF vocabulary used for expressing axioms constraining the interpretation of graphs. The RDFS vocabulary and its semantics are given in (Hayes 2004).

For our purpose, we consider four RDFS axioms: *subclass*, *subproperty*, *domain* and *range*. The DL fragment of RDFS is a subset of $\mathcal{SHI}$. Hence RDFS axioms can be translated into $\mathcal{SHI}$ axioms.

$\mathcal{SHI}$ Description logics are fragments of first-order logic that model a domain of interest in terms of concepts and roles (Baader et al. 2007). For this study, we consider the description logic $\mathcal{SHI}$ which is a fragment of $\mathcal{SHIQ}$ (Horrocks, Sattler, and Tobies 1999). The satisfiability of $\mathcal{SHIQ}$ logic is proved to be ExpTime. We assume standard notation for the syntax and semantics of $\mathcal{SHI}$ knowledge bases. We refer the reader to (Horrocks, Sattler, and Tobies 1999) for the semantics of $\mathcal{SHI}$.

Syntax In $\mathcal{SHI}$, concepts and roles are formed according to the following syntax:

$$C ::= \bot \mid \top \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists r.C \mid \forall r.C$$
$$r ::= p \mid p^-$$

p denotes an atomic role, $\bot$ represents an empty concept, A denotes an atomic concept, C denotes a complex concept, and r denotes a complex role which is an atomic role or its inverse.

$\mathcal{SHI}$ **Axioms:** The TBox is a finite set of axioms consisting of *concept inclusions* ($C_1 \sqsubseteq C_2$), *role inclusion* ($r_1 \sqsubseteq r_2$), and *transitivity* $trans(r)$ axioms.

SPARQL is a W3C recommended query language for RDF (Prud'hommeaux and Seaborne 2008). It is based on the notion of query patterns defined inductively from triple patterns: a tuple $t \in \mathrm{UBV} \times \mathrm{UV} \times \mathrm{UBLV}$, with V a set of variables disjoint from UBL, is called a triple pattern. Triple patterns grouped together using SPARQL operators AND and UNION form *query patterns* (or graph patterns)[1]. We use an abstract syntax that can be easily translated into the μ-calculus.

Definition 1 (Query Pattern)**.** *A query pattern q is inductively defined as follows :*

$$q ::= t \in \mathrm{UBV} \times \mathrm{UV} \times \mathrm{UBLV} \mid q_1 \text{ AND } q_2 \mid q_1 \text{ UNION } q_2$$

We focus on SELECT queries which are the core of SPARQL queries.

Definition 2. *A SPARQL 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 2 (SPARQL queries)**.** *Consider the following queries $q_1(?x)$ and $q_2(?x)$ on the graph of Example 1:*

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

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

SPARQL has multiset (or bag) semantics, however, when dealing with containment, we consider set semantics. This is due to the undecidability of union of conjunctive queries under bag semantics (**?**).

The semantics of SPARQL queries is given by a partial mapping function ρ from V to 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 $\forall x \in dom(\rho_1) \cap dom(\rho_2)$, $\rho_1(x) = \rho_2(x)$. Further, if ρ_1 and ρ_2 are compatible, then $\rho_1 \cup \rho_2$ is also a mapping. The evaluation of query patterns over an RDF graph G is inductively defined as follows:

$$[\![t]\!]_G = \{\rho \mid dom(\rho) = var(t) \text{ and } \rho(t) \in G\}$$

where $var(t)$ is the set of variables occurring in t.

$$[\![q_1 \text{ AND } q_2]\!]_G = [\![q_1]\!]_G \bowtie [\![q_2]\!]_G$$
$$[\![q_1 \text{ UNION } q_2]\!]_G = [\![q_1]\!]_G \cup [\![q_2]\!]_G \quad [\![q\{\overrightarrow{w}\}]\!]_G = \pi_{\overrightarrow{w}}([\![q]\!]_G)$$

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

Example 3. *The answers to queries q_1 and q_2, of Example 2, on graph G of Example 1 are respectively $\{Poe, Mallarme\}$ and $\{Baudelaire, Poe, Mallarme\}$. Thus, $[\![q_1]\!]_G \subseteq [\![q_2]\!]_G$.*

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

Definition 3 (Containment)**.** *Given a set of axioms $\mathcal{C}$ and two queries q and q' with the same arity, q_1 is contained in q_2 with respect to $\mathcal{C}$, denoted $q \sqsubseteq_{\mathcal{C}} q'$, iff $[\![q]\!]_G \subseteq [\![q']\!]_G$ for every graph G satisfying $\mathcal{C}$.*

Definition 4 (Equivalence)**.** *Two queries q and q' under a set of axioms $\mathcal{C}$ are equivalent, i.e., $q \equiv_{\mathcal{C}} q'$ iff $q \sqsubseteq_{\mathcal{C}} q'$ and $q' \sqsubseteq_{\mathcal{C}} q$.*

[1] We do not consider OPTIONAL and FILTER query patterns because containment over the full SPARQL is undecidable.

The evaluation of SPARQL queries is proved to be PSPACE-complete. However, the evaluation problem is NP-complete for the fragment containing only AND and UNION query patterns (Pérez, Arenas, and Gutierrez 2009).

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.

μ-calculus

The modal μ-calculus (Kozen 1983) 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, and a set of *programs* Prog for navigating in graphs. A μ-calculus formula, φ, can be defined inductively as follows:

$$\varphi \quad ::= \quad \top \mid \bot \mid p \mid X \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu X \varphi \mid \nu X \varphi$$

where $p \in AP, X \in Var$ 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 (Kozen 1983). 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 $R(\bar{a}) = \{(s', s) \mid (s, s') \in R(a)\}$. In addition, a valuation function $V : \text{Var} \rightarrow 2^S$ 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 K (a.k.a. Kripke structure) and a valuation function, is represented by $[\![\varphi]\!]^K_V$. The semantics of basic μ-calculus formulae is defined as follows:

$$[\![\top]\!]^K_V = S \quad [\![p]\!]^K_V = L(p), p \in AP \cup Nom, \; L(p) \text{ is singleton for } p \in Nom$$

$$[\![X]\!]^K_V = V(X), X \in \text{Var} \quad [\![\neg\varphi]\!]^K_V = S \backslash [\![\varphi]\!]^K_V$$

$$[\![\varphi \wedge \psi]\!]^K_V = [\![\varphi]\!]^K_V \cap [\![\psi]\!]^K_V, \quad [\![\varphi \vee \psi]\!]^K_V = [\![\varphi]\!]^K_V \cup [\![\psi]\!]^K_V$$

$$[\![\langle a \rangle \varphi]\!]^K_V = \{s \in S \mid \exists s' \in S.(s, s') \in R(a) \wedge s' \in [\![\varphi]\!]^K_V\}$$

$$[\![[a]\varphi]\!]^K_V = \{s \in S \mid \forall s' \in S.(s, s') \in R(a) \Rightarrow s' \in [\![\varphi]\!]^K_V\}$$

$$[\![\mu X \varphi]\!]^K_V = \bigcap \{S' \subseteq S \mid [\![\varphi]\!]^K_{V[X/S']} \subseteq S'\}$$

$$[\![\nu X \varphi]\!]^K_V = \bigcup \{S' \subseteq S \mid S' \subseteq [\![\varphi]\!]^K_{V[X/S']}\}$$

The next sections introduce a representation of RDF graphs as transition systems and queries as μ-calculus formulas.

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 5 (Transition system associated to an RDF graph)**.** *Given an RDF graph, $G \subseteq UB \times U \times UBL$, the transition system associated to G, $\sigma(G) = (S, R, L)$ over $AP = \text{UBL} \cup \{s', s''\}$, is such that:*

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

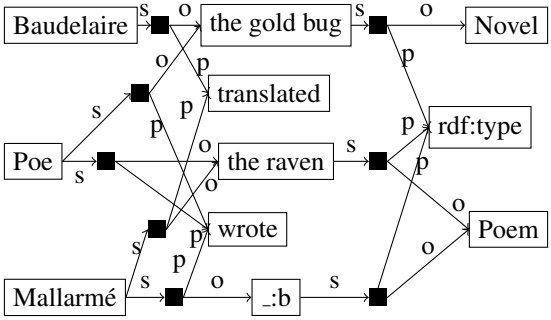


Figure 1: Transition system encoding the RDF graph of Example 1. Nodes in S'' are black anonymous nodes; nodes in S' are the other nodes (d-transitions are not displayed).

When checking for query containment, we consider two constraints: (i) the set of programs is fixed: $Prog = \{s, p, o, d, \bar{s}, \bar{p}, \bar{o}, \bar{d}\}$, and (ii) a model must be a restricted transition system. The 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*

$$\xi = (\neg\langle\bar{s}\rangle\top \wedge \neg\langle o\rangle\top \wedge \neg\langle p\rangle\top \wedge \neg\langle d\rangle\top \wedge \neg\langle\bar{d}\rangle\top \wedge \neg\langle s\rangle s' \wedge \neg\langle\bar{o}\rangle s' \wedge \neg\langle\bar{p}\rangle s).$$

The formula φ_r ensures that θ and κ hold in every node reachable by a d edge, i.e. in every triple node. The formula θ forces each s'' node to have one and only one 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.

If a μ-calculus formula ψ appears under the scope of a least μ or greatest ν fixed point operator over all the programs $\{s, p, o, d, \bar{s}, \bar{p}, \bar{o}, \bar{d}\}$ as, $\mu X.\psi \vee \langle s\rangle X \vee \langle p\rangle X \vee \cdots$ or $\nu X.\psi \wedge \langle s\rangle X \wedge \langle p\rangle X \wedge \cdots$, then, for the sake of legibility, we denote the recursion components of the respective formulae as $mu(X)$ for the μ recursion part and $nu(X)$ for the ν recursion part. Thus, the formulae become $\mu X.\psi \vee mu(X)$ and $\nu X.\psi \wedge nu(X)$.

SPARQL Query Containment

In this section, we encode queries and schema axioms as μ-calculus formulas. Then, we reduce query containment under schemas to μ-calculus unsatisfiability and prove the correctness of this reduction.

Encoding Queries as μ-calculus Formulae

Queries are translated into 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 or some formula that are satisfied when they are at the corresponding position in such triple relations. A function called $\mathcal{A}$ is used to encode queries inductively on the structure of query patterns. AND and UNION are translated into boolean connectives $\wedge$ and $\vee$ respectively. When encoding $q \sqsubseteq q'$, we call q left-hand side query and q' right-hand side query.

Encoding left-hand side query: the encoding of a left-hand side query q is done such that every term (distinguished and non-distinguished variables and constants) in the query becomes a nominal in the μ-calculus. Hence, the encoding of q is $\mathcal{A}(q)$ such that:

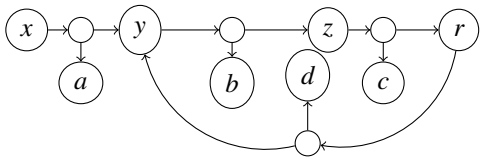
$$\mathcal{A}((x, y, z)) = \mu X.\big(\langle\bar{s}\rangle x \wedge \langle p\rangle y \wedge \langle o\rangle z\big) \vee mu(X)$$
$$\mathcal{A}(q_1 \text{ AND } q_2) = \mathcal{A}(q_1) \wedge \mathcal{A}(q_2)$$
$$\mathcal{A}(q_1 \text{ UNION } q_2) = \mathcal{A}(q_1) \vee \mathcal{A}(q_2)$$

In order to encode the right-hand side query, we need the notion of cyclic queries.

Definition 6 (Cyclic Query)**.** *A SPARQL query is referred to as cyclic if a transition graph induced from the query patterns is cyclic. The transition graph* [2] *is constructed in the same way as done in Definition 5.*

[2]The transition graph is similar to the tuple-graph used in (Calvanese, De Giacomo, and Lenzerini 2008) to detect the dependency among variables.

Example 4. *q is cyclic, as shown graphically,* $q(x) = (x, a, y), (y, b, z), (z, c, r), (r, d, y)$



Encoding right-hand side query: the encoding of the right-hand side query q' is different from that of the left due to the non-distinguished variables that appear in cycles in the query. The distinguished variables and constants are encoded as nominals whereas the non-distinguished variables $ndvar(q')$ are encoded as follows:

- First, for each triple $t_i \in q'$, introduce a fresh nominal n_i, i.e., $t(t_i) = n_i$. This nominal is satisfied in a triple node S'' in a restricted transition system.
- Second, we use a function that assigns a formula for each $x \in ndvar(q')$ as follows:
  - If x occurs only once in q', x is encoded as $\top$.
  - If x appears multiple times in q' and $x \in t_i \in q'$, then

$$m_i = \{x \mapsto \varphi \mid \begin{cases} \varphi = \langle s\rangle t(t_i) \text{ if } subject(x) \text{ or} \\ \varphi = \langle\bar{p}\rangle t(t_i) \text{ if } predicate(x) \text{ or} \\ \varphi = \langle\bar{o}\rangle t(t_i) \text{ if } object(x) \quad \} \end{cases}$$

    Note that there is an exponential number of m_i's in terms of the number of non-distinguished variables. More precisely, there are at most $O(n^k)$ mappings, where n is the number of triples where non-distinguished variables appear, and k is the number of non-distinguished variables.

- Finally, the function $\mathcal{A}$ uses t and m to encode the query inductively:

$$\mathcal{A}(q, m) = \bigvee_{i=1}^{|m|} \mathcal{A}(q, m_i)$$
$$\mathcal{A}((x, y, z), m) = \mu X.\big(t((x, y, z)) \wedge \langle\bar{s}\rangle d(m, x) \wedge \langle p\rangle d(m, y) \wedge \langle o\rangle d(m, z)\big) \vee mu(X)$$
$$\mathcal{A}(q_1 \text{ AND } q_2, m) = \mathcal{A}(q_1, m) \wedge \mathcal{A}(q_2, m)$$
$$\mathcal{A}(q_1 \text{ UNION } q_2, m) = \mathcal{A}(q_1, m) \vee \mathcal{A}(q_2, m)$$
$$d(m, x) = \begin{cases} \varphi & \text{if } (x \mapsto \varphi) \in m \\ \top & \text{if } unique(x) \\ x & \text{otherwise} \end{cases}$$

Example 5. *Consider the encoding of $q_1 \sqsubseteq q_2$ of Example 2. To encode q_1, freeze the variables and constants and proceed with $\mathcal{A}$ such that $\mathcal{A}(q_1) =$*

$$\big((\mu X.(\langle\bar{s}\rangle x \wedge \langle p\rangle \texttt{translated} \wedge \langle o\rangle l) \vee mu(X)) \vee (\mu X.(\langle\bar{s}\rangle x \wedge \langle p\rangle \texttt{wrote} \wedge \langle o\rangle l) \vee mu(X))\big) \wedge (\mu X.(\langle\bar{s}\rangle l \wedge \langle p\rangle \texttt{type} \wedge \langle o\rangle \texttt{Poem}) \vee mu(X))$$

To encode q_2, one first computes t and m. Hence, $t((x, translated, l)) = n_1$, $t((l, type, Poem)) = n_2$, $t((x, wrote, l)) = n_3$ and $m = \{m_1, m_2, m_3\}$ where $m_1 = \{y \mapsto \langle\bar{o}\rangle n_1\}$, $m_2 = \{y \mapsto \langle s\rangle n_2\}$, and $m_3 = \{y \mapsto \langle\bar{o}\rangle n_3\}$. Finally, $\mathcal{A}(q_2, m) = \bigvee_{i=1}^{|m|} \mathcal{A}(q, m_i) =$

$$
\begin{aligned}
&\big((\mu X.(n_1 \wedge \langle\bar{s}\rangle x \wedge \langle p\rangle\texttt{translated} \wedge \langle o\rangle\langle\bar{o}\rangle n_1) \vee mu(X)\\
&\quad \wedge \mu X.(n_2 \wedge \langle\bar{s}\rangle\langle\bar{o}\rangle n_1 \wedge \langle p\rangle\texttt{type} \wedge \langle o\rangle\texttt{Poem}) \vee mu(X))\\
&\quad \vee \mu X.(n_3 \wedge \langle\bar{s}\rangle x \wedge \langle p\rangle\texttt{wrote} \wedge \langle o\rangle\langle\bar{o}\rangle n_1) \vee mu(X)\big) \vee\\
&\big((\mu X.(n_1 \wedge \langle\bar{s}\rangle x \wedge \langle p\rangle\texttt{translated} \wedge \langle o\rangle\langle s\rangle n_2) \vee mu(X)\\
&\quad \wedge \big(\mu X.(n_2 \wedge \langle\bar{s}\rangle\langle s\rangle n_2 \wedge \langle p\rangle\texttt{type} \wedge \langle o\rangle\texttt{Poem}) \vee mu(X))\\
&\quad \vee \mu X.(n_3 \wedge \langle\bar{s}\rangle x \wedge \langle p\rangle\texttt{wrote} \wedge \langle o\rangle\langle s\rangle n_2) \vee mu(X)\big) \vee\\
&\big((\mu X.(n_1 \wedge \langle\bar{s}\rangle x \wedge \langle p\rangle\texttt{translated} \wedge \langle o\rangle\langle\bar{o}\rangle n_3) \vee mu(X)\\
&\quad \wedge \mu X.(n_2 \wedge \langle\bar{s}\rangle\langle\bar{o}\rangle n_3 \wedge \langle p\rangle\texttt{type} \wedge \langle o\rangle\texttt{Poem}) \vee mu(X))\\
&\quad \vee \mu X.(n_3 \wedge \langle\bar{s}\rangle x \wedge \langle p\rangle\texttt{wrote} \wedge \langle o\rangle\langle\bar{o}\rangle n_3) \vee mu(X)\big)
\end{aligned}
$$

Encoding Axioms

Besides the encoding of the queries, we introduce the encoding of $\mathcal{SHI}$ schema axioms as below.

Definition 7. *Given a set of axioms $c_1, c_2, ..., c_n$ of a schema $\mathcal{C}$, the μ-calculus encoding of $\mathcal{C}$ is:*

$$\eta(\mathcal{C}) = \eta(c_1) \wedge \eta(c_2) \wedge ... \wedge \eta(c_n).$$

Where η translates each axiom into an equivalent formula using ω which in turn recursively encodes concepts and roles:

$$
\begin{aligned}
\eta(r_1 \sqsubseteq r_2) &= \nu X.\ \big(r_1 \Rightarrow r_2\big) \wedge nu(X)\\
\eta(C_1 \sqsubseteq C_2) &= \nu X.\ \big(\omega(C_1) \Rightarrow \omega(C_2)\big) \wedge nu(X)\\
\omega(A) &= A \quad \omega(\neg C) = \neg\omega(C) \quad \omega(\bot) = \bot\\
\omega(C_1 \sqcap C_2) &= \omega(C_1) \wedge \omega(C_2)\\
\omega(\exists r.C) &= \langle s\rangle\big(\langle p\rangle r \wedge \langle o\rangle(\langle s\rangle\langle o\rangle\omega(C))\big)\\
\omega(\forall r.C) &= [s]\big([p]r \Rightarrow [o]([s][o]\omega(C))\big)\\
\omega(\exists r^-.C) &= \langle\bar{o}\rangle\big(\langle p\rangle r \wedge \langle\bar{s}\rangle(\langle s\rangle\langle o\rangle\omega(C))\big)\\
\omega(\forall r^-.C) &= [\bar{o}]\big([p]r \Rightarrow [\bar{s}]([s][o]\omega(C))\big)\\
\eta(trans(r)) &= \nu X.\langle s\rangle\big(\langle p\rangle r \wedge \langle o\rangle(y \wedge \langle s\rangle(\langle p\rangle r \wedge \langle o\rangle z))\\
&\qquad \Rightarrow (\langle p\rangle r \wedge \langle o\rangle z)\big) \wedge nu(X)
\end{aligned}
$$

So far we proposed various functions to produce formulas corresponding to the encodings of queries and schema axioms. Hence, the problem of containment under a schema can be reduced to formula unsatisfiability in μ-calculus as:

$q \sqsubseteq_{\mathcal{C}} q' \Leftrightarrow \eta(\mathcal{C}) \wedge \mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \varphi_r$ is unsatisfiable.

For the sake of legibility in writing, we use $\Phi(\mathcal{C}, q, q')$ to denote $\eta(\mathcal{C}) \wedge \mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \varphi_r$.

Reducing Containment to Unsatisfiability

We prove the correctness of reducing query containment to unsatisfiability test.

Lemma 1. *Given a set of schema axioms $\mathcal{C} = \{c_1, \cdots, c_n\}$, $\mathcal{C}$ has a model iff $\eta(\mathcal{C})$ is satisfiable.*

Lemma 2. *For any SPARQL query q, q is satisfiable iff $\mathcal{A}(q)$ and $\mathcal{A}(q, m)$ are satisfiable.*

Proof. (sketch) We prove for $\mathcal{A}(q, m)$, the proof for $\mathcal{A}(q)$ is immediate.
($\Rightarrow$) a canonical instance of q can be converted into a transition system that satisfies $\mathcal{A}(q, m)$.
($\Leftarrow$) any formula corresponding to a query encoding is satisfiable. However, each satisfying model may not be a restricted transition system. Thus, we use $\mathcal{A}(q, m) \wedge \varphi_r$ (Proposition 1), to guarantee that satisfying models are restricted transition systems. As such, it can be shown that a model of the formula $\mathcal{A}(q, m) \wedge \varphi_r$ can be turned into a graph G that satisfies q. □

Theorem 1 (Soundness)**.** *Given queries q and q', and a set of axioms $\mathcal{C}$, if $\Phi(\mathcal{C}, q, q')$ is unsatisfiable, then $q \sqsubseteq_{\mathcal{C}} q'$.*

Proof. (sketch) We show the contrapositive. If $q \not\sqsubseteq_{\mathcal{C}} q'$, then $\Phi(\mathcal{C}, q, q')$ is satisfiable. It can be verified that every model of $\mathcal{C}$ in which there is at least one tuple satisfying q but not q' can be a satisfying model for $\Phi(\mathcal{C}, q, q')$. □

Theorem 2 (Completeness)**.** *Given queries q and q', and a set of axioms $\mathcal{C}$, if $\Phi(\mathcal{C}, q, q')$ is satisfiable, then $q \not\sqsubseteq_{\mathcal{C}} q'$.*

Proof. $\Phi(\mathcal{C}, q, q')$ is satisfiable $\Rightarrow \exists K.[\![\Phi(\mathcal{C}, q, q')]\!]^K \neq \emptyset$. Consequently, K is a restricted transition system due to $[\![\varphi_r]\!]^K \neq \emptyset$ (cf. Proposition 1). Using $K = (S' \cup S'', R, L)$ we construct a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{C}$ such that $q \not\sqsubseteq q'$ holds:

- $\Delta^{\mathcal{I}} = S'$, $A^{\mathcal{I}} = [\![A]\!]^K$ for each atomic concept A,
- $\top^{\mathcal{I}} = [\![\top]\!]^K$, for a top concept,
- $r^{\mathcal{I}} = \{(s, s') \mid \forall t \in [\![r]\!]^K \wedge t' \in S'' \wedge (s, t') \in R(s) \wedge (t', t) \in R(p) \wedge (t', s') \in R(o)\}$ for each atomic role r,
- for each constant c in q and q', $c^{\mathcal{I}} = [\![c]\!]^K$,
- for each distinguished and non-distinguished variable v in q, $v^{\mathcal{I}} = [\![v]\!]^K$, and
- for each distinguished variable v in q', $v^{\mathcal{I}} = [\![v]\!]^K$.

One can utilize Lemma 1, to verify that indeed $\mathcal{I}$ is a model of $\mathcal{C}$. Thus, it remains to show that $[\![q]\!]_{\mathcal{I}} \not\subseteq [\![q']\!]_{\mathcal{I}}$. From our assumption, one anticipates the following:

$$
\begin{aligned}
&[\![\mathcal{A}(q) \wedge \neg\mathcal{A}(q')]\!]^K \neq \emptyset\\
&\quad \Rightarrow [\![\mathcal{A}(q)]\!]^K \neq \emptyset \text{ and } [\![\neg\mathcal{A}(q', m)]\!]^K \neq \emptyset\\
&\quad \Rightarrow [\![\mathcal{A}(q)]\!]^K \neq \emptyset \text{ and } [\![\mathcal{A}(q', m)]\!]^K = \emptyset
\end{aligned}
$$

Note here that, if a formula φ is satisfiable in a restricted transition system K_r, then $[\![\varphi]\!]^{K_r} = S$. We use a function f to construct an RDF graph G from the interpretation $\mathcal{I}$. f uses assertions in $\mathcal{I}$ to form triples:

$$
\begin{aligned}
f(a \in A^{\mathcal{I}}) &= (a, \texttt{type}, A) \in G\\
f((a, b) \in r^{\mathcal{I}}) &= (a, r, b) \in G\\
f((a, b) \in (r^-)^{\mathcal{I}}) &= (b, r, a) \in G\\
f((x, y, z)) &= (x, y, z) \in G,\ \forall (x, y, z) \in q
\end{aligned}
$$

As a consequence, $[\![q]\!]_G \neq \emptyset$ and $[\![q']\!]_G = \emptyset$ because G contains all those triples that satisfy q and not q'. Therefore, we get $[\![q]\!]_G \not\subseteq [\![q']\!]_G$. Fundamentally, there are two issues to be addressed (i) when q' contains a cycle and (ii) when q' is not cyclic. (i) can be dealt with nominals, i.e., since cycles can be expressed by a formula in a μ-calculus extended with nominals and inverse, cyclic queries can be encoded by such a formula. Hence, the constraints expressed by $\neg\mathcal{A}(q', m)$ are satisfied in a transition system containing cycles. On the other hand, (ii) if there are no cycles in q', then replacing non-distinguished variables with $\top$ suffices (cf. Lemma 2). □

Complexity In the following, we establish the complexity of the containment problem under schema axioms. The schema axioms can be formed using the fragments of $\mathcal{SHIQ}$. More specifically, the fragments without number restrictions. The expressiveness of the schema language is limited as such due to the expressive power of the logic used for the encoding: μ-calculus with nominals and converse becomes undecidable when extended with graded modalities (Bonatti et al. 2006).

Proposition 2. *SPARQL query containment under $\mathcal{SHI}$ schema axioms can be determined in a time of $2^{\mathcal{O}(n^2 log\ n)}$ where $n = \mathcal{O}(|\eta(\mathcal{C})| + |\mathcal{A}(q)| + |\mathcal{A}(q')|)$ is the size of the formula, and $\eta(\mathcal{C})$, $\mathcal{A}(q)$ and $\mathcal{A}(q')$ denote the encodings of schema axioms $\mathcal{C}$, and queries q and q'.*

Note that due to duplication in the encoding of q_2, the size of $|\mathcal{A}(q_2)|$ is exponential in terms of the non-distinguished variables that appear in cycles in the query. Hence, we obtain a 2ExpTime upper bound for containment. As pointed out in (Calvanese, De Giacomo, and Lenzerini 2008), the problem is solvable in ExpTime if there is no cycle on the right hand side query. This complexity is a lower bound due to the complexity of satisfiability in μ-calculus which is $2^{\mathcal{O}(n^2 log\ n)}$ (Sattler and Vardi 2001; Tanabe, Takahashi, and Hagiya 2008).

Related Works

In the following we briefly review works that previously established closely related results for related query languages. We took a similar approach as (Genevès, Layaïda, and Schmitt 2007) that established the optimal complexity for XPath query containment and provided an effective implementation.

Studies on the translation of SPARQL into relational algebra and SQL (Cyganiak 2005; Chebotko et al. 2006) indicate a close connection between SPARQL and relational algebra in terms of expressiveness. In (Polleres 2007), a translation of SPARQL queries into a datalog fragment (non-recursive datalog with negation) that is known to be equally expressive as relational algebra (RA) was presented. This translation makes the close connection between SPARQL and rule-based languages explicit and shows that RA is at least as expressive as SPARQL. Tackling the opposite direction, it was recently shown in (Angles and Gutierrez 2008) that SPARQL is relationally complete, by providing a translation of the above-mentioned datalog fragment into SPARQL. As argued in (Angles and Gutierrez 2008), the results from (Polleres 2007) and (Angles and Gutierrez 2008) taken together imply that SPARQL has the same expressive power as relational algebra. From early results on query containment in relational algebra and first-order logic, one can infer that containment in relational algebra is undecidable (contrary to the results in (Chekol et al. 2011)). Therefore, containment of SPARQL queries is also undecidable. Hence, in this paper, we considered a fragment of SPARQL containing only conjunction and disjunction for this study.

The most closely related work is (Calvanese, De Giacomo, and Lenzerini 2008) in which query containment under description logic constraints is studied based on an encoding in propositional dynamic logic with converse (CPDL). They establish 2ExpTime upper bound complexity for containment of queries consisting of union of conjunctive queries under $\mathcal{DLR}$ schema axioms. 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 languages and ontologies (for instance OWL-DL). Besides, the two languages are different since SPARQL allows for predicates to be used as subject or object of other triple patterns and can be in the scope of a variable. This is not directly allowed in $\mathcal{DLR}$ (union) of conjunctive queries. Our encoding of RDF graphs and SPARQL queries preserves this capability.

Other related results come from the study of query entailment and query answering. Query entailment (and hence containment) in DLs ranging from $\mathcal{ALCI}$ to $\mathcal{SHIQ}$ is shown to be 2ExpTime-hard in (Lutz 2008; Glimm et al. 2008; Eiter et al. 2009). In this paper we do not deal with the same query language than the one dealt with in (Glimm et al. 2008). In fact, the supported SPARQL fragment is strictly larger than the one studied in (Glimm et al. 2008). Specifically, UCQs in (Glimm et al. 2008) are made of $C(x), R(x, y)$ for an atom C, a role R, and variables x and y, whereas we do also support queries capable of querying concept and role names at the same time, such as $q(x) = (x, y, z)$. Further, the purpose of reducing the problem to μ-calculus is exactly about extending query containment to even more features (such as SPARQL 1.1 paths with recursion, entailment regimes, and negation). For instance, it is known that recursive paths can be easily supported in mu-calculus (using fixpoints) whereas it is known that extending previous approaches with this feature is notoriously difficult. Beyond this, the novelty of the study is the reduction of the SPARQL containment problem to μ-calculus satisfiability, and the advantages of using such a logic: great expressivity, good computational properties, extensibility. The main focus of the contribution is not the complexity bound by itself but rather a new approach with a broader logic, paving the way for future extensions as it was never done before.

Here, we would like to emphasize that, in addition to the complexity bound we provide, no implementation has been reported in previous works, whereas in our case our work opens the way to use an implementation like the one in (Tanabe, Takahashi, and Hagiya 2008) or (Genevès, Layaïda, and Schmitt 2007).

Finally, the evaluation of SPARQL query under schema constraints is considered by W3C under the entailment regime principle in which SPARQL queries are evaluated by taking into account the semantics of a schema language (Kollia, Glimm, and Horrocks 2011). It is possible to define query containment under such entailment regimes (cf. (**?**) for instance). However, because the schema is not made explicit in entailment regimes, this would not allow to consider containment under a particular schema as we did here. And this could be very useful particularly because (1) schema are very often separated from the data and (2) this allows for compiling the schema.

Conclusion

We have introduced a mapping from RDF graphs into transition systems and the encodings of queries and schema axioms in the μ-calculus. We proved that this encoding is correct and can be used for checking query containment. We have provided implementable algorithms, as a consequence, this work opens a way to use available implementations of μ-calculus satisfiability solvers from (Tanabe, Takahashi, and Hagiya 2008) and (Genevès, Layaïda, and Schmitt 2007). Beyond this, we have established a double exponential upper bound for containment test under $\mathcal{SHI}$ axioms.

As a future work, we plan to extend the schema language with nominals (becomes $\mathcal{SHOI}$) and analyse the optimality of the complexity. Because nominals are part of the logic, the complexity of containment under $\mathcal{SHOI}$ axioms has already a 2ExpTime upper bound. Furthermore, what would be interesting is, to identify the fragments of SPARQL queries and DLs that can be encoded in the fragments of μ-calculus with nominals and converse, graded modalities and converse, and nominals and graded modalities (Bonatti et al. 2006; Tanabe, Takahashi, and Hagiya 2008).

References

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

Angles, R., and Gutierrez, C. 2008. The expressive power of sparql. *The Semantic Web-ISWC 2008* 114–129.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. F., eds. 2007. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press. ISBN 9780511717383.

Bonatti, P. A.; Lutz, C.; Murano, A.; and Vardi, M. Y. 2006. The Complexity of Enriched μ-calculi. *Automata, Languages and Programming* 540–551.

Calvanese, D.; De Giacomo, G.; and Lenzerini, M. 2008. Conjunctive Query Containment and Answering under Description Logics Constraints. *ACM Trans. on Computational Logic* 9(3):22.1–22.31.

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

Chebotko, A.; Lu, S.; Jamil, H.; and Fotouhi, F. 2006. Semantics preserving sparql-to-sql query translation for optional graph patterns. Technical report, Technical Report TR-DB-052006-CLJF.

Chekol, M. W.; Euzenat, J.; Genevès, P.; and Layaïda, N. 2011. PSPARQL query containment. DBPL'11.

Cyganiak, R. 2005. A relational algebra for sparql. *Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170*.

Eiter, T.; Lutz, C.; Ortiz, M.; and Šimkus, M. 2009. Query answering in description logics with transitive roles. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 759–764.

Genevès, P.; Layaïda, N.; and Schmitt, A. 2007. Efficient Static Analysis of XML Paths and Types. PLDI '07, 342–351.

Glimm, B.; Horrocks, I.; Lutz, C.; and Sattler, U. 2008. Conjunctive query answering for the description logic $shiq$. *J Artif Intell Res* 31:157–204.

Hayes, P. 2004. RDF Semantics. W3C Recommendation.

Horrocks, I.; Sattler, U.; and Tobies, S. 1999. Practical reasoning for expressive description logics. In *Logic for Programming and Automated Reasoning*, 161–180. Springer.

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

Kollia, I.; Glimm, B.; and Horrocks, I. 2011. SPARQL query answering over OWL ontologies. In *Proc. 8th ESWC, Hersounissos (GR)*, volume 6643 of *LNCS*, 382–396.

Kozen, D. 1983. Results on the propositional μ-calculus. *Theor. Comp. Sci.* 27:333–354.

Lutz, C. 2008. The complexity of conjunctive query answering in expressive description logics. *Automated Reasoning* 179–193.

Pérez, J.; Arenas, M.; and Gutierrez, C. 2009. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)* 34(3):16.

Polleres, A. 2007. From SPARQL to rules (and back). WWW '07, 787–796.

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

Sattler, U., and Vardi, M. Y. 2001. The Hybrid μ-Calculus. In *IJCAR*, 76–91.

Schmidt, M.; Meier, M.; and Lausen, G. 2010. Foundations of SPARQL Query Optimization. ICDT '10, 4–33.

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