HAL
open science

SPARQL Query Containment with ShEx Constraints

Abdullah Abbas, Pierre Genevès, Cécile Roisin, Nabil Layaïda

To cite this version:

Abdullah Abbas, Pierre Genevès, Cécile Roisin, Nabil Layaïda. SPARQL Query Containment with ShEx Constraints. 2016. hal-01414509v1

HAL Id: hal-01414509

https://inria.hal.science/hal-01414509v1

Preprint submitted on 12 Dec 2016 (v1), last revised 29 Nov 2017 (v2)

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.

SPARQL Query Containment with ShEx Constraints

Abdullah Abbas, Pierre Genevès*, Cécile Roisin, Nabil Layaïda
Université Grenoble Alpes, CNRS, INRIA, LIG
Grenoble, France
[firstname.lastname]@inria.fr, [firstname.lastname]@cnrs.fr*

***Abstract*—ShEx (Shape Expressions) is a language for expressing constraints on RDF graphs. We consider the problem of SPARQL query containment in the presence of ShEx constraints. We first investigate the complexity of the problem according to the fragments considered for SPARQL queries and for ShEx constraints. In particular, we show that the complexity of SPARQL query containment remains the same with or without ShEx constraints. We develop two radically different approaches for solving the problem and we evaluate them. The first approach relies on the joint use of a ShEx validator and a tool for checking query containment without constraints. In a second approach, we show how the problem can be solved by a reduction to a fragment of first-order logic with two variables. This alternative approach allows to take advantage of any of the many existing FOL theorem provers in this context. We evaluate how the two approaches compare experimentally, and report on lessons learned. To the best of our knowledge, this is the first work addressing SPARQL query containment in the presence of ShEx constraints.**

Keywords—static analysis; containment; queries; shape expressions

I. Introduction

ShEx (or Shape Expressions) is intended to be an RDF constraint language [1] [2]. It can be used to validate documents and communicate expected graph patterns. Static analysis and query optimization can make a considerable benefit from the presence of schemas when used to infer satisfiability/unsatisfiablity of queries and relations between queries (such as containment and equivalence) by utilizing the additional information provided by the schemas.

In this work we investigate the SPARQL query containment with ShEx constraints. Given two SPARQL queries, and a set of ShEx constraints being considered on the RDF data to be queried, our purpose is to statically analyze such queries, namely determining the containment relation between them before being actually executed on the data.

For the fragments of SPARQL including OPTIONAL patterns, the containment of queries is normally investigated with the notion of subsumption [3]. A solution mapping ϱ_1 is subsumed by another solution mapping ϱ_2 written as $\varrho_1 \sqsubseteq \varrho_2$ if all the elements of ϱ_1 are also in ϱ_2 and have the same mapping values. Given a set of mappings Ω_1 (resembling a SPARQL query solution), it is subsumed by another set of mappings Ω_2 written as $\Omega_1 \sqsubseteq \Omega_2$ if for every $\varrho_1 \in \Omega_1$ there exists $\varrho_2 \in \Omega_2$ such that $\varrho_1 \sqsubseteq \varrho_2$.

Deciding containment (as static analysis in general) helps us take wise actions before executing the queries to improve the performance and efficiency by reducing access to resources where such access is not needed as a result. For example if two queries Q_1 and Q_2 are to be executed on a huge dataset, and we know that $Q_1 \sqsubseteq Q_2$, then Q_2 can be executed on the results of Q_1 instead of the whole dataset.

The consideration of ShEx constraints in query containment is important, because such constraints may affect the results of containment checking. Consider the following two SPARQL query graph patterns:

Q_1: {?x :producer :p1 . ?x :feature "feature1"}
OPT {?x :feature "feature2" . ?x :expiryDate ?d}

Q_2: {?x :producer ?y . ?x :feature "feature1"}

With no constraints, no containment relation holds between these two queries. Now consider the following ShEx constraints defined for a "product" node type:

<product> {
:name xsd:string ,
:expiryDate xsd:date ? ,
:producer @<company> + ,
:feature xsd:string }

The previous ShEx shape definition means that a node of type "product" should have a name of type string, optionally have an expiry date, have at least one producer which belongs to a another ShEx shape definition <company>, and have exactly one feature of type string. Given that these ShEx constraints apply to the data, we can deduce that a containment relation $Q_1 \sqsubseteq Q_2$ holds between the two queries. This is due to the constraint that a "feature" predicate is allowed to occur only once, and thus in query Q_1 the right hand side of the optional pattern will never return results. With the results from the right hand side only, given that the previous ShEx constraints applies to the data, we can deduce that a containment relation $Q_1 \sqsubseteq Q_2$ always hold between the two queries as long as the constraints apply.

There are several kinds of constraint violations that may lead to a new conclusion about the containment of two queries. These include cardinality constraint violations, basic data type constraint violations (like xsd:string, xsd:data …), and ShEx type definition violations (like @<company> type). Additionally, there have already been a discussion about open and closed shape semantics for ShEx [4]. An open shape would match a subject if all rules in the shape are passed, however not

all triples have to be matched. For a closed shape however each triple on the subject has to be matched. With closed semantics, even more violation cases may appear that will also affect the decision of containment between two queries.

Now consider the following two queries graph patterns and the same previous ShEx schema.

Q_3: {:p1 :producer ?y} OPT {:p1 :rating ?z}

Q_4: {?x :producer ?y} OPT {?x :review ?z}

With open semantics, no containment relation holds between these two queries, while with closed semantics, $Q_3 \sqsubseteq Q_4$ holds due to the fact that the right side of the OPT patterns will never return results in both queries, because rating and review predicates are not defined in the <product> ShEx shape.

Data on the web are getting larger, and distribution of data is getting more applicable. Different data sources are often being managed by different authorities. The need of schemas becomes increasingly necessary in order to manage the big amounts of data. While different sources in the same domain may share the same vocabulary, their constraints on data may vary according to the specific needs of each authority taking responsibility of its own set of data. While these slight differences in data shapes may become a hassle for users to track individually, the OPT patterns in SPARQL provides a way to ask for constraints that are not necessarily applicable, granting flexibility for users to define a common framework for queries even when schemas may differ with different sources, and that is why the study of the optional fragment is particularly important in our study, concerning static analysis of queries before executing them on a particular set of data in order to improve their execution.

Our purpose in this work is automate the process of determining containment relations in the presence of the ShEx schema constraints.

In [5], the authors proposed a schema language for edge-labeled data graphs (like RDFs), and then studied the satisfiablity of 3 different classes of query languages (RPQs, NREs, and CRPQs) when such constraints are considered, but this study did not include containment. In [6], the authors explored the complexity of containment and evaluation problems for fragments of SPARQL 1.1 property paths. The study in [7] provides complexity analysis for several fragments of SPARQL. Additionally, in [8] the containment of OPTIONAL queries is investigated. The work in [9], [10], and [11] studies the containment problem with entailment regimes (SHI, RDFS, OWL…). For the works on ShEx, in [12] and [13] the expressiveness and validation complexity of ShEx was studied.

In this work we draw several research directions to cope with ShEx expressions while solving containment between queries. First, we define a methodology for performing containment of SPARQL with ShEx and we show that the complexity of this problem is Π_2^P-complete for the fragment of SPARQL with well-designed optional patterns, and NP-complete for fragments of SPARQL without optional patterns, and we explain why ShEx does not increase the complexity of the original containment problem (without ShEx). Then we propose an alternative method for solving the problem based on encoding to FOL. We investigate several logics to check their suitability for performing the task (by expressivity and complexity). We also provide complete logical encoding method for our problem with FOL with only 2 variables, a decidable fragment of FOL, while having restriction on Kleene star in ShEx to atomic symbols only. We finally provide 2 implementations, based on the two previously mentioned approaches, and we evaluate how they compare experimentally. This work is the first work to deal with containment in the presence of ShEx constraints, and also the first work to deal with containment of SPARQL optional patterns with a constraint or schema language.

A. Paper Outline

In section (II), we introduce preliminary notions from the literature related to our work. We develop two radically different approaches for solving the problem. In section (III) we define our first approach which relies on the joint use of a ShEx validator and a tool for checking query containment without constraints. We use this approach as bases to show the complexity of the problem in the same section. In section (IV) we introduce our second approach which is based on reduction to a fragment of first-order logic with two variables. We then evaluate how the two approaches compare experimentally in section (V). In section (VI) we mention and compare other previous works related to our work, and we end up with a conclusion.

II. Preliminaries

A. SPARQL

SPARQL is an RDF query language and a W3C Recommendation, where RDF is a directed, labeled graph data format for representing information in the Web [14] [15]. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions [16].

1) SPARQL Abstract Syntax

For this work on SPARQL containment, we consider a fragment of SPARQL without property paths, and thus the definition will be held accordingly. Property paths may be integrated later, but we exclude them because they don't have a major value for the specificity of our study.

A SPARQL graph pattern is defined inductively from triple patterns. Given disjoint infinite sets of IRIs (**I**), blank nodes (**B**), literals (**L**), and variables (**V**), we define a triple pattern as an instance of $(I \cup B \cup V) \times (I \cup V) \times (I \cup B \cup L \cup V)$ denoted by $IBV \times IV \times IBLV$. A SPARQL graph pattern q is defined inductively from triple patterns as follows:

$q ::= t \mid q\ AND\ q' \mid q\ UNION\ q' \mid q\ OPT\ q'$ where t is a triple pattern.

We also define a basic graph pattern BGP as:

$BGP ::= t \mid q\ AND\ q'$ where t is a triple pattern and q and q' are also BGPs, i.e. a BGP is conjunctive graph pattern.

2) Well-Designed OPT Patterns

Well-Designed OPT patterns are SPARQL patterns representing a class of OPTIONAL patterns that has several desired properties [17], such as evaluation performance

advantages and possibility to have a common normal form of such queries.

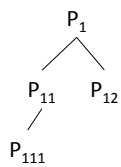
A query q is well-designed if for every subpattern $q' = (q_1\ OPT\ q_2)$ of q and every variable x occurring in q, it holds that: if x occurs inside q_2 and outside q', then x also occurs inside q_1.

It is also shown in [17] that any well-designed graph pattern can be equivalently rewritten in the normal form:

$$(\ldots(t_1\ AND\ \ldots\ AND\ t_k)\ OPT\ O_1)\ OPT\ O_2)\ \ldots)\ OPT\ O_n)$$

Where each t_i is a triple pattern, $n \geq 0$ and each O_j has the same form (also in normal form).

These normal forms has good properties regarding query evaluation [17], and can be represented as pattern trees as described in [8]. For example, a query of the form $(P_1\ OPT\ (P_{11}\ OPT\ P_{111}))\ OPT\ P_{12}$ can be represented as a pattern tree as follows:



In our work, we use pattern tree representations of the queries in order to study containment and parse normal form queries in our implementations.

B. ShEx

ShEx (or Shape Expressions) is intended to be an RDF constraint language. It can be used to validate documents and communicate expected graph patterns. The logical operators in Shape Expressions, grouping, conjunction, disjunction and cardinality constraints, are defined to make as closely as possible to their counterparts in regular expressions and grammar languages like BNF [18]. Shape Expressions correlate an ordered pattern of pairs of predicate and object classes (called *NameClass* and *ValueClass*) and logical operators against an unordered set of edges in a graph. For example, *Shape1* in the following is a definition of a shape in ShEx, where a ShEx document contains a definition for several shapes.

```
<Shape1> {
  ex:name xsd:string,
  ex:phone xsd:string
}
```

In the previous example, ex:name and ex:phone are *NameClass*es and xsd:string is a *ValueClass*. This definition means that for a node belonging to this shape there must strictly exist the predicates ex:name and ex:phone, each once. The objects corresponding to these predicates must be of type xsd:string.

1) ShEx Abstract Syntax

Given a finite set of edge labels Σ and a finite set of types Γ, we define a *shape expression* e over $\Sigma \times \Gamma$ as follows: $e ::= \epsilon \mid \Sigma \times \Gamma \mid e^* \mid (e"|"e) \mid (e" \parallel "e)$, where "|" is a disjunction, "||" is an unordered concatenation, and "*" is an unordered Kleene star. The previous definition also allows us to further define as macros $e^?$ (optional), e^+ (positive closure), and $(\Sigma \times \Gamma)^{[m;n]}$ (interval from m to n), which are all parts of the ShEx syntax.

A *shape expression* as defined above allows us to define a shape in ShEx. A shape definition (for convenience with the abstract syntax we also call it a *type definition*) is a mapping from the set Γ to the set of shape expressions. A ShEx schema is a set of *type definitions*. If for some type $t \in \Gamma$ a rule is missing, the default rule is $t \rightarrow \epsilon$, which means a node belonging to this type must have no relations at all (useful for pre-defined types such as xsd:string, and not for user defined types).

For simplicity we write $(a, t) \in \Sigma \times \Gamma$ as $a :: t$. The following is an example of a ShEx schema in abstract syntax.

$$S: \quad t_1 \rightarrow a :: t_2 \parallel b :: t_3$$
$$t_2 \rightarrow (a :: t_2 \mid c :: t_1)^*$$

Semantically, a graph is valid against a schema if it is possible to assign types to the nodes of the graph in a manner that satisfies the type definitions of the schema.

C. FOL with two Variables

FOL (First Order Logic) is in general a logic that in addition to the classical truth values $(\top, \bot)$ and logical connectives (conjunction $\wedge$, disjunction $\vee$, implication $\rightarrow \ldots$), it allows negation, predicate symbols, function symbols, variables, and quantification over variables ($\exists$ and $\forall$).

FOL^2 (FOL with two variables) is a decidable fragment of FOL that allows no more than 2 interacting variables. This fragment is attractive from a theoretical point of view due to its expressivity, adding that it has been proven that it is still decidable with equality/inequality and counting quantifiers (e.g. $\exists^{<n}$). [19] [20] [21] [22]

The presence of more than two variables in a formula is not an evidence that the formula does not belong to FOL^2. Thus the expressivity of FOL^2 may not be intuitive. Consider the following example expression that can be expressed in FOL^2: "There exists a path of length 4"

Form 1: $\exists x \exists y \exists z \exists w \exists v \big(R(x,y) \wedge R(y,z) \wedge R(z,w) \wedge R(w,v)\big)$

Form 2: $\exists x \exists y \Big(R(x,y) \wedge \exists z \Big(R(y,z) \wedge \exists w \big(R(z,w) \wedge \exists v R(w,v)\big)\Big)\Big)$

Form 3: $\exists x \exists y \Big(R(x,y) \wedge \exists x \Big(R(y,x) \wedge \exists y \big(R(x,y) \wedge \exists x R(y,x)\big)\Big)\Big)$

The three forms are equivalent to each other, and they belong to FOL^2.

III. Problem Definition and Complexity

The well-designed OPT fragment of SPARQL have been well studied, as well as its containment problem, in several works from the literature [17] [8]. Our purpose in this section is to generalize this containment problem to the presence of ShEx constraints, by defining a methodology for solving the problem, and studying the complexity associated with it.

Given two well-designed SPARQL queries and a ShEx schema which defines constraints on the RDF data, our purpose is to study whether subsumption always holds from one query to the other in the presence of the schema constraints.

SPARQL containment with ShEx is not more complex than containment without ShEx for the considered well-designed OPT fragment of SPARQL. Indeed the containment problem with ShEx can be understood as two steps:

• Step 1: Validating the two queries against ShEx and generating two new derived queries

• Step 2: Then, checking containment between the two new queries

We note out that the validation of a query (with variables) in the 1st step is not more complex than the validation of an RDF document. For the purpose of validation we can always assume fresh IRIs assigned to the variables of the queries. In fact, the validation of the queries will be done with more loose constraints than that of an RDF document. It only checks that what is present in the query does not violate the ShEx type and cardinality constraints. This does not require forcing the presence of the complete patterns required by the set of ShEx constraints, which may only be considered when studying the complete RDF data set. To provide such more loose constraints when validating the queries, we employ a simple trick, by modifying all the ShEx minimal cardinality constraints, allowing them to tend to zero.

Although semantically the consequences of the validation would change the queries in question (by suppressing some optional parts for example, and thus disallowing some results from appearing), yet the new queries are not more expressive than the original queries. It can be understood as a modification of the query within the same fragment. The 2nd step is the containment of modified queries resulting from the 1st step. For example, if in the validation step we find out that a certain part of the query (some OPTIONAL part for example) will never return results due to the ShEx constraints, the modified query that results from this validation is by omitting this OPTIONAL part. Usually the first step is less complex than the containment alone in the second step, and that is why the containment complexity will hold the same in most of the cases, and in all the fragments we considered in our study (as defined in the previous table).

To show how the two steps interact in practice, and how we use the results of this procedure to check containment, we show the application of the two steps on the following query graph pattern examples (given in the introduction):

Q_3: {:p1 :producer ?y} OPT {:p1 :rating ?z}

Q_4: {?x :producer ?y} OPT {?x review ?z}

and the following ShEx Schema:

```
<product> {
:name xsd:string ? ,
:expiryDate xsd:date ? ,
:producer @<company> * ,
:feature xsd:string ? }
```

We notice that because we must not consider minimal cardinalities, those are disregarded in the above ShEx schema in comparison to the schema given in the introduction. Minimal cardinalities must be allowed to tend to zero instead, and that is why ? (and *) are used instead of single cardinality (and + respectively) in the above schema.

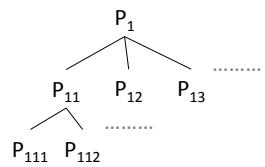
In the first step, we need to validate the parts of both queries. For the validation purpose, we assign to each variable a fresh IRI. For query Q_1, we consider two triple sets for validation against the ShEx schema, {:p1 :producer :y} which is valid, and {:p1 :producer ?y. :p1 :rating :z}, the optional pattern with its parent, which is not valid. As a result of this validation step, we change the query by removing the optional pattern which corresponds to the triple set which is not valid, and thus we get:

Q_1': {:p1 :producer ?y}

By following the same validation step, we also get:

Q_2': {?x :producer ?y}

In the second step, we check the containment of the two resulting modified queries, without any consideration of the ShEx constraints. This step can be done by strategies followed for simple containment, as in [8], which gives us that $Q_1' \sqsubseteq Q_2'$ holds. We use this information to conclude that $Q_1 \sqsubseteq Q_2$ holds in the presence of the ShEx schema example given above.



In general, given a well-designed OPT query pattern in normal form, where P_i is a BGP for all values of *i*, for each node of the pattern tree representation we have to consider a validation step. The validation of a node considers its own triples, and all the triples of its ancestors up to root. For example if the BGP P_1 is not valid, then the whole query will never return results. If the BGP $(P_1 \cup P_{11})$ is not valid, then node P_{11} and all its descendants must be eliminated. If the BGP $(P_1 \cup P_{11} \cup P_{111})$ is not valid, then node P_{111} must be eliminated, and so on. Thus a new well-designed pattern tree will be obtained. The new resulting pattern tree resembles the graph pattern of the new query on which the containment in the second step should be done.

The complexity of the problem is given by the following theorem.

Theorem. *Given two SPARQL queries in well-designed OPT form and a ShEx document, the query subsumption problem on data complying to the given ShEx constraints is Π_2^P-complete.*

A. Complexity Proof

We first prove the membership of the problem the complexity class Π_2^P, and then we prove its hardness.

The membership can be argued by following the previous mention procedure in the current section, consisting of the two consecutive steps, each of which can be proven to be Π_2^P. The ShEx validation problem to be used in step 1 is proven to be in NP [12]. Given the repeated application of the validation procedure on all the nodes of the pattern tree, the query

validation becomes in Π_2^P (by definition, Π_2^P is coNP With NP Oracle). The second step, which is applied after completely accomplishing the first step, is proven in [8] to be Π_2^P-complete.

The correctness of the procedure for solving our problem is argued in the following.

We mention first that if a containment relation between two queries holds in general, then it also holds in the presence of any ShEx schema because these constraints are just a special case of the general situation where containment holds. Thus we also conclude the containment in the presence of ShEx without any manipulation. We concentrate on studying the cases where containment does not originally hold for general cases (i.e. without constraints).

By the SPARQL semantics, any mapping result following from the execution of a well-designed OPT query, should comply to a set of branches all of which are rooted at the root P_1 of the pattern tree. Such result may be suppressed if one of these branches is eliminated from the pattern tree, which is what we do in the first step by eliminating non valid nodes of the pattern tree according to the ShEx constraints.

We first show the effect of node elimination presented previously due to query validation. If a node's BGP is not valid, then we know that the query will not return results for this optional pattern, and thus can be eliminated, resulting in an equivalent query under the considered ShEx constraints. For a node validation, we consider all the ancestor nodes because they are necessary for the studied node, where by semantics of the optional pattern, it just gives complementary results, where the ancestor results must occur first. This means the combination up to the root must be considered in the validation of each node because all triples included will be considered for the solving of this optional pattern, where the occurrences combined together may violate the ShEx constraints, and thus must be checked all together.

Now when considering two queries for containment, more factors must be considered to prove the correctness of the two step procedure. Yet we show how our procedure avoids these problematic factors. While considering query validation, two factors are considered, the type constraints, and the cardinality constraints, and thus will be studied repectively.

Type constraints: A ShEx schema which is disjunction free may not lead to a contradiction of types. The only way a contradiction may occur in validation of queries (where each one is separately valid), is when there is a disjunction rule in the schema with a determined cardinality, and each of the two queries is fulfilling a different side of the disjunction rule - where by the ShEx semantics either the first may occur or the second, but not both simultaneously. For example when we have the ShEx rule **(a::t_1 | a::t_2)**, and **a::t_1** is occurring in query Q_1, and **a::t_2** is occurring in query Q_2. In such case, the different type definitions will appear explicitly in the IRI objects of the triples fulfilling the type definitions. Explicitly different IRIs will not hold for simple containment test [8], and thus a positive result will not be granted in the second step (containment) even if the validation passes in the first step.

Cardinality constraints: A cardinality contradiction may occur when both quires do not violate the cardinality constraints, but the patterns of the two queries combined violates them. We remind that we only need to consider maximal cardinalities. Given a single ShEx rule with cardinality *k*, assume that the rule is fulfilled *n* times in the Q_1 and *m* times in Q_2.

- case 1: If the *n* occurrences counted are all included in the *m* occurrences in the second query, this will never rise a problem since the validation of the second query will be enough for maintaining cardinality constraints in the whole scenario.

- case 2: A problem scenario may only occur if at least one occurrence of the *n* occurrences is not present in the *m* occurrences and given that $m+n>k$. For this case to occur, the counts that are considered are actually the ones that are explicit by distinct IRI objects. The presence of such distinct IRI in one query but not in the other will yield the queries not to hold for simple containment test [8], and thus a positive result will not be granted in the second step (containment) even if the validation passes in the first step.

Now we prove the Π_2^P hardness of the problem by arguing that any well-designed SPARQL containment problem without ShEx can be reduced to well-designed SPARQL containment problem with ShEx where the empty ShEx schema is considered. The well-designed SPARQL containment problem alone is proven in [8] to be Π_2^P hard as well. ■

Given results from the literature related to containment, and with the consideration of the specificity of our problem, we summarize the complexity results of the SPARQL containment problem with ShEx in the following table, with respect to the mentioned SPARQL features.

SPARQL			**SPARQL Containment with ShEx (Problem Complexity)**
Cycles	***OPT (well designed)***	***Union (external)***	
✓	×	×	**NP-complete**
✓	×	✓	**NP-complete**
✓	✓	×	**Π_2^P-complete**
✓	✓	✓	**Π_2^P-complete**

The complexity results in the table above hold the same if ShEx was not considered, which means the problem being considered as the containment between two queries only. This allows us to extract such complexities directly from literature [7] [8].

IV. A Reduction to FOL Validity

In this section we provide another method for solving our previously defined problem by encoding it into an FOL formula and studying its validity with existing automated theorem provers. Our final purpose is compare the two different methods for solving our problem in application.

Encoding the problem into a logical formula provides an alternative method for solving our problem, allowing the use of existing and highly optimized automated theorem provers to achieve better performance, which we experimentally validate in section (V).

The problem encoding is divided into two formulas. An axiom formula which resembles the encoding of the ShEx

document constraints, and another conjecture formula expressing the problem of the containment of two well-designed OPT queries. These two formulas (the axiom and the conjucture) can be given to an automated theorem prover to check the validity of the whole problem.

To best handle such problem, we first devise a survey including the decidable logics or fragments to be considered, and then we choose the most suitable fragment for our problem. Encoding our containment problem is possible with several logics found out to be used for similar problems. For example we know how to practically encode the problem with ∃MSO (Existential Monadic Second Order logic), μ-calculus, K-modal logic, and FOL (First Order Logic). The complexity of our encodings is the same for all the mentioned logics, given in the table below with respect to the mentioned SPARQL features.

SPARQL			Encoding Complexity
Cycles	**OPT (well designed)**	**Union (external)**	
✓	×	×	**PTIME**
✓	×	✓	
✓	✓	×	**EXPTIME**

Only some fragments of FOL are decidable, whereas other mentioned logics are entirely decidable. While the complexity of the satisfiablity problem of ∃MSO is elementary in general (and quadratic time for finite simple graphs), for μ-calculus is 2EXPTIME, and for K-modal logics is PSPACE-complete, there exist better complexity bounds for some decidable fragments of FOL.

In the table below we give a summary of the decidable fragments of FOL, by a survey from the literature, with their characteristics and their known satisfiability complexity bounds.

FOL Fragment	**Functions allowed/ arity**	**Predicate arity**	**# of vars**	**Counting quantifiers**	**(In)equality allowed**	**Finite model property**	**Tree model property**
FOL guarded fragment The fragment with only guarded quantification. **Sat:** 2EXPTIME-c		∞	∞		✓	✓	✓ (g)
FOL with only 2 variables (L^2) **Sat:** NEXPTIME-c		∞	2		✓	✓	
FOL with only 2 variables (C^2) Is an extension of (L^2) with counting quantifiers **Sat:** NEXPTIME-c		∞	2	✓	✓		
Monadic predicate logic The fragment in which all relation symbols in the signature are monadic.	1	1	∞		✓	✓	
The prefix class ($\exists^*\forall^*$)		∞	∞	✓	✓		
The prefix class ($\exists^*\forall\exists^*$)	✓	∞	∞	✓	✓		
The prefix class ($\exists^*\forall^2\exists^*$)		∞	∞				
The prefix class ($\exists^*$)	✓	∞	∞	✓	✓		
Propositional Modal Logic **Sat:** PSPACE-c						✓	✓

Some of the mentioned FOL fragments presented in the table above can handle our problem encoding, while others fail, relative to the expressivity of each. Considering the feasibility of encoding we encounter, and complexities, we find FOL^2 (FOL with 2 variables only) to be most suitable for our problem encoding. Accordingly, in the rest of the paper we will be using this logic to describe our problem and its solution.

Furthermore, there are several FOL theorem provers available, with good and competent performances, which makes our choice also suitable from the point of view of applicability.

For the purpose of our work in this section, it is convenient to put a restriction on the usage of the Kleene star "*" in ShEx. We consider a ShEx fragment that allows Kleene star on atomic symbols only. Kleene star on non-atomic symbols can't be expressed in any of the previously mentioned logics.

A. *ShEx Document Encoding (ζ)*

For convenience, in this section we will consider a fragment of ShEx that does not allow a predicate to be used multiple times in a shape definition. In a section later, we explain how to extend our encoding for the full fragment of ShEx by introducing the notion of determinism and the three different fragments tracing this definition.

To start defining the encoding, we first define for each ShEx shape a primitive relation $S_i(x)$ meaning that the RDF term x has the shape S_i.

Let P_i be the set of predicates allowed in a shape S_i. Let R be a relation of arity 3 for expressing triple pattern relations (subject, predicate, object). Let O_p be a relation of arity 1, describing the type of restriction of the predicate p on the object (e.g. String, Date, IRIInstance…). For each shape S_i we define the following implication rule: *[→ is the logical implication]*

$$S_i(x) \rightarrow \left(\forall y\left(S_{Allowed}(x,y)\right) \wedge S_{Cardinality}(x)\right)$$

Where:

$$S_{Allowed}(x,y) \equiv \bigwedge\{R(x,p,y) \rightarrow O_p(y) \mid p \in P_i\}$$

The cardinality formula depends on the form of the shape definition expression, and thus must be defined inductively taking into account the ShEx connectives that may occur.

$$S_{Cardinality}(x) \equiv \delta(e)$$
$$\delta(e_1 \parallel e_2) \equiv \delta(e_1) \wedge \delta(e_2)$$
$$\delta(e_1 \mid e_2) \equiv \delta(e_1) \vee \delta(e_2)$$
$$\delta\left((p :: o)^{[m;n]}\right) \equiv \exists^{\leq n} y\, R(x,p,y)$$
$$\delta((p :: o)^*) \equiv \top$$

Notice that we only care about maximal cardinality constraints, because if this number is exceeded in the query, we

know that the query will not return results for this part of the query. While in the case of minimal cardinality constraints, it doesn't matter for the query (the constraints should apply on the RDF document, not the query).

Now we define the ShEx constraints by the disjunction of all the shapes:

$$\forall x\big(S_1(x) \vee S_2(x) \vee \ldots \vee S_n(x)\big)$$

By analyzing the previous encoding scheme and its unfolding, one can see that the result is always within the scope of the FOL^2 fragment (it does not need more than two variables for all cases).

B. Query Encodings

In [8] the authors provide a method for well-designed OPT query containment based on homomorphism between two pattern trees corresponding to the two queries in question, and they provide a proof for the correctness of their method. In this section we formulate the homomorphism between two pattern trees in the FOL logic, allowing us to use the formula as a conjecture in our problem to check the containment in the presence of the ShEx constraints axiom. The final containment conjecture formula, similar to the work in [11], is given a the following combined encoding, where $\mathcal{A}(Q1)$ is the encoding of the first query, and $\mathcal{A}(Q2)$ is the encoding of the second query:

$$\mathcal{A}(Q1) \wedge \neg\mathcal{A}(Q2)$$

In the following, we define the encoding $\mathcal{A}(Q)$ of a query Q.

Given a query, each variable in it will be given an IRI referring to its name. This encoding will not use FOL variables, it is just a set of axioms. We define the encoding of a query $\mathcal{A}(Q)$ as follows, where P is a BGP (basic graph pattern):

$$\mathcal{A}(P\ ) = \bigwedge\{R(s,p,o) \mid \{s,p,o\} \in P\}$$
$$\mathcal{A}(P\ OPT\ P_1\ OPT \ldots OPT\ P_n) = \mathcal{A}(P\ ) \vee \Big(\mathcal{A}(P\ ) \wedge \big(\mathcal{A}(P_1\ ) \vee \ldots \vee \mathcal{A}(P_n\ )\big)\Big)$$

This encoding assumes that there are no variables in the predication position. We next explain how such variables can be adopted in our encoding.

1) Predicates as Variables

Normally, if a predicate variable is substituted by an IRI as followed for other variables, this IRI will not match to any of the predicates of the ShEx document, and thus will be treated as a violation of the ShEx rules.

This problem can be solved by substitution. For the set of predicate variables, we substitute them by all the possible combinations of the predicate URIs that are mentioned in the ShEx document. This means that the containment test should be repeated several times, as much as there are different combinations. While it is possible to be expressed (by disjunctions), yet the size of the encoding will increase dramatically.

Without using substitution, expressing predicate variables as variables in the logic can serve our need except that there is a limitation by the FOL^2 fragment. Some encodings in this case may give a formula out of the scope of FOL^2, and thus make our problem undecidable.

Although several solution may be adopted, we propose a restriction on the query form that keeps the encoding size with no expansion at all. The restriction states that the predicate variables may only appear in the predicate positions in the query. In this case, it would be possible to treat these variables as variables in FOL and existentially quantify them ($\exists p_1 \exists p_2 \exists p_3 \ldots$). Although more than two predicate variables may occur in the query, yet the encoding still belongs to FOL^2 because these variables are not interacting in a single relation *R*, and thus can be separated into sets of triples each including only a single predicate variable.

Yet a less tight restriction can be used instead, which states that a predicate variable can be connected to at most two other predicate variables by the relation R. Connected to at most 2 other variables, this allows us to represent the connections as a chain, and thus make it possible to be expressed in FOL^2. (*Check back the example "There exist a path of length 4" which gives a close approach*)

2) "Containing Query" Variables

By containing query, we mean the query on the right hand side of the containment symbol ($Q_{left} \subseteq Q_{right}$). The variables of Q_{right} which also appear in Q_{left} will be substituted by IRIs corresponding to them. For other variables of Q_{right} we substitute them by all the possible combinations from the IRIs of Q_{left}.

To reduce the number of combinations to be considered, one can adopt techniques previously used in the literature, such as limiting the substitutes of a variable relative to the position it occurs on (subject, predicate, or object position).

C. Non-Determinism of Predicate Rules in ShEx

We mention the notion of non-determinism because it affects the encoding according to the ShEx fragment considered. This notion has been used in a previous study on the complexity of ShEx [12]. In our work we define 3 different fragments that are useful to determine an encoding accordingly.

The encoding as described previously in this work for the purpose of containment can be considered as a basic encoding that works for an unambiguous simple fragment, which is the fragment that allows a predicate *p* to appear only once in a ShEx shape definition. We call this *fragment 1*.

The source of ambiguity in ShEx shape definition analysis may rise when a predicate *p* is used multiple times. More precisely this ambiguity occurs either when the disjunction in ShEx is used arbitrarily, or when the same predicate *p* corresponds to two (or more) different object values. In both cases our encoding will lose track of cardinality constraints. To deal with these cases separately, we further define two more fragments of ShEx.

Fragment 2 is defined as the fragment that allows a predicate to appear multiple times, yet is associated to at most 1 object value. As an example for this fragment, the following shape definition: **(a::t_1|b::t_2) || (a::t_1|c::t_3)**. In order to deal with this fragment, we transform the set of rules into another equivalent set of rules in disjunctive normal form (DNF), in analogy to logical formulas. Thus the transformation of the above example

becomes as follows: **(a::t_1^2)|(a::t_1 || c::t_3)|(b::t_2 a::t_1)**. The new form can be encoded directly by the previous encoding rules with no ambiguities.

Fragment$_{all}$ is the fragment with no restrictions regarding the use of predicates and their corresponding values. For a given predicate, this fragment allows multiple object values. As an example for this fragment, the following shape definition: **(a::t1 || a::t2)**. In order to deal with the encoding ambiguity in this case, we propose to make the object values (i.e. t_1 and t_2) as a part of the primitive relation *R* instead of being just an argument in it. This means that instead of defining a single relation *R(sub, pred, obj)*, we should define a relation for each object value that occurs in the ShEx document, in our case Rt_1 and Rt_2, with *(sub, pred, obj)* being their arguments as well. Notice that in general (more complicated examples may occur), a DNF transformation is still needed in this fragment. This technique with DNF transformation will avoid encoding ambiguities in *fragment$_{all}$*.

D. In Application: Problem Encoding without Counting Quantifiers

The encoding defined previously in this work uses counting quantifiers with the FOL fragment considered. Although the results are theoretically correct and it holds from the literature that such encoding is decidable, yet it is useful to study the encoding without using counting quantifiers for practical reasons. In fact, we don't know any FOL solver implementation available with the support of counting quantifiers.

In this section we show how the encoding can be adjusted to deal with cardinality without using counting quantifiers. To achieve our purpose, we introduce a modification to the ShEx document encoding, and also a slight modification to the query encoding, to be described next respectively.

1) ShEx Encoding (Without Counting Quantifiers)

In order keep track of the cardinality constraints, we need to introduce new primitive relations in addition to the original general relation *R(sub,pred,obj)*. We introduce the relations R_1, R_2, R_3… to resemble the cardinality constraints of values 1, 2, 3 … respectively.

Given the whole implication rule for a shape (as defined previously), we modify it by making the universal quantification $\forall y$ apply to $S_{Cardinality}(x)$ also, as follows:

$$S_i(x) \rightarrow \left(\forall y \ \left(S_{Allowed}(x,y) \wedge S_{Cardinality}(x)\right)\right)$$

Then a further modification will take place in the $S_{Cardinality}(x)$ part of the encoding (only), where the δ encoding function's basic case becomes as follows:

$$\delta\left((p :: o)^{[m;n]}\right) \equiv \neg R_{n+1}(x,p,y)$$

This encoding is forbidding the cardinality that is higher than the maximal cardinality allowed. This will implicitly also forbid all cardinalities higher than $(n+1)$, since the appearance of R_{n+1} is necessary for the appearance of R_{n+2} in the query encoding as will be seen in the next subsection.

2) Query Encoding (Without Counting Quantifiers)

The query encoding defined previously in this work does not itself use any counting quantifiers. These quantifiers were only used in the ShEx encoding where they are implicitly being able to keep track of the number of occurrences of relations of the query. Without using counting quantifiers in the ShEx encoding, it is necessary to adjust the query encoding in order to meet the new semantics of the encoding. We define the new query encoding as follows:

$$\mathcal{A}(P) = \bigwedge \{R(s,p,o) \wedge R_i(s,p,o)\}$$

such that $\{s,p,o\} \in P$, and it is the ith occurrence of p as predicate

$$\mathcal{A}(P \ OPT \ P_1 \ OPT \ \ldots OPT \ P_n) = \mathcal{A}(P) \vee \left(\mathcal{A}(P) \wedge \left(\mathcal{A}(P_1) \vee \ldots \vee \mathcal{A}(P_n)\right)\right)$$

V. Implementation and Experimentation

Previously in this work we presented two methods for solving our problem. We first provided a method based on the using of ShEx validator and a SPARQL query containment solver. The second method is based on encoding to an FOL formula with two variables.

We also implemented both methods as a part of this work in order to validate our procedures in practice, and in order to compare the performance of the different implementations.

For the first method, we used a ShEx validator provided by the authors of [13], and a containment solver for well-designed SPARQL queries (called SPAM tool) provided by the authors of [7]. The whole procedure is combined into a common Java framework, while using Jena ARQ [23] as a SPARQL parser.

For the second method, we also used Jena ARQ as a query parser in Java in order to write our encoding in the fof TPTP syntax [24], which is a FOL syntax accepted by the majority of automated theorem provers. For the ShEx constraints document encoding into a TPT axiom, we use a ShEx syntax expressed in JSON, where there are several tools for available for converting from/to ShEx JSON syntax to/from ShEx compact syntax [25]. The resulting TPTP axiom and conjecture are given in our experiments to two different automated theorem provers, namely SPASS [26] and E Theorem Prover [27], to validate our encodings in practice.

We then compare the performance of the implementations of the two methods by preparing a benchmark of 5 pairs of different queries among which we want to check the containments. The containment is considered for each pair against the empty schema (equivalent to containment without schema), in addition to 4 prepared different ShEx schemas, and we comment on the results.

The queries used are a collection from the Berlin SPARQL Benchmark [28], where the 4 different schemas are constructed by us with varying type and cardinality constraints in order to study the effect of such constraints.

In the appendix at the end of this paper we present the queries and the schemas used in our benchmark. The expected (valid) results are as follows:

	No Schema	S1	S2	S3	S4
Q1a ⊑ Q1b	✓	✓	✓	✓	✓
Q2a ⊑ Q2b	×	×	×	×	✓
Q3a ⊑ Q3b	×	✓	×	×	×
Q4a ⊑ Q4b	×	×	✓	×	×
Q5a ⊑ Q5b	×	×	×	✓	×

With both implementation we get the same results as above. We compare the two methods by comparing the execution time on the different cases above in practice.

A. Results on First Implementation

	Con. without Schema (ms)	Valid. Time (ms)	Con. with Schema (ms)	Total (Valid+Con.) (ms)
QaQb1:S1	45	723	44	767
QaQb1:S2	45	714	42	756
QaQb1:S3	45	738	41	779
QaQb1:S4	45	724	42	766
QaQb2:S1	51	837	53	890
QaQb2:S2	51	851	52	903
QaQb2:S3	51	840	57	897
QaQb2:S4	51	870	53	923
QaQb3:S1	44	740	44	784
QaQb3:S2	44	725	43	768
QaQb3:S3	44	766	44	810
QaQb3:S4	44	754	44	798
QaQb4:S1	47	786	43	829
QaQb4:S2	47	736	0	736
QaQb4:S3	47	815	45	860
QaQb4:S4	47	861	45	906
QaQb5:S1	59	897	57	954
QaQb5:S2	59	913	57	970
QaQb5:S3	59	861	47	908
QaQb5:S4	59	942	53	995

B. Results on Second Implementation (FOL)

	Queries encoding time (ms)
QaQb1	367
QaQb2	371
QaQb3	366
QaQb4	368
QaQb5	368

	Schemas encoding time (ms)
Schema 1	25
Schema 2	25
Schema 2	25
Schema 2	25

Formula execution on SPASS: (time in ms)

QaQb1	≈10 (with all schemas)
QaQb2	≈20 (with all schemas)
QaQb3	≈10 (S0, S1) ≈20 (with other schemas)
QaQb4	≈20 (with all schemas)
QaQb5	≈20 (S0) ≈30 (with other schemas)

(The approximate results above is due to the lack of accuracy of time measuring to the level of ones in milliseconds)

Formula execution on E Theorem Prover: (time in ms)

	No Schema	S1	S2	S3	S4
QaQb1	7	8	8	8	8
QaQb2	10	12	11	12	12
QaQb3	9	10	10	10	10
QaQb4	11	12	12	13	13
QaQb5	15	18	17	18	17

The previous results shows a better performance for the cases considered with the FOL implementation. We notice that although the theoretical execution of the second method is 3-EXP (resulting from EXP encoding and 2-EXP for the Satisfiability problem of the FOL formula), yet in practice we get better execution time by almost half the execution time compared to the other method which is theoretically in Π_2^P. We expect such results due to the high optimization techniques used in the automated theorem provers, and due to the fact that realistic queries (even relatively big queries) are small with respect to the problem considered. Validation of a whole set of data in contrast with FOL may be tedious to the huge amount of data deployed in such cases, while by nature, queries are relatively concise. It is also clear from the previous results that most of the time is spent on parsing the queries and encoding them, and not in the automated theorem prover itself.

VI. Related Works

In [5], the authors proposed a schema language for edge-labeled data graphs (like RDFs), and then studied the satisfiablity of 3 different classes of query languages (RPQs, NREs, and CRPQs) when such constraints are considered, but this study did not include containment. In [29] and [30] the authors studied static analysis aspects of XPath using mu-calculus and Monadic Second-order Logic respectively, then the authors provided in [31] a tool related to their studies. XPath is a query language on tree structures while in our work SPARQL is a query language on graphs.

The work in [32]studied containment of PSPARQL, an extension of SPARQL 1.0 with paths and path constraints. In [6], the authors explored the complexity of containment and evaluation problems for fragments of SPARQL 1.1 property paths. The study in [7] provides complexity analysis for several fragments of SPARQL. Additionally, in [8] the containment of well-designed OPTIONAL queries is investigated, and in [33] the same authors provided a tool called SPAM for static analysis and particularly containment of SPARQL queries. Recently related to containment, a work was published on the containment considering an extension of SPARQL 1.1 with navigational queries called EPPs (Extended Property Paths) which is more expressive than the former [34]. None of these works consider schemas in the study of containment.

The work in [9], [10], and [11] studies the containment problem with ontology languages and entailment regimes (SHI, RDFS, OWL…). Ontology languages consider an open world assumption, while schemas (like ShEx) consider closed word assumption on data. In general, ontology languages and schemas serve different purposes from a semantic point of view. These works on containment with ontology languages, focuses on entailment regimes employed in these languages, but not on the fragments of SPARQL with optional patterns.

The authors in [35] provide a benchmark for evaluating SPARQL containment.

For the works on ShEx, in [18], [12], and [13] the expressiveness and validation complexity of ShEx was studied. The work in [36] proposes an implementation of shape expressions to RDF graphs. Recently, an early working draft

became available on the comparative expressiveness of ShEx and SHACL (which is less expressive than the former) [37].

VII. Conclusion

In this work we studied SPARQL query containment in the presence of Shape Expressions, a schema language for RDF. We focused on a fragment of SPARQL including OPTIONAL patterns. We focus on well-designed SPARQL patterns, and their representations as pattern trees. We define the problem and show that the complexity for the studied fragment is Pi_2^p-complete, and that in general the complexity is not more than containment without ShEx. We develop two radically different approaches for solving the problem and we evaluate them. The first approach relies on the joint use of a ShEx validator and a tool for checking query containment without constraints. In a second approach, we show how the problem can be solved by a reduction to a fragment of first-order-logic with two variables. This alternative approach allows to take advantage of any of the many existing FOL theorem provers in this context. We evaluate how the two approaches compare experimentally. It was shown that the FOL implementation has a better practical performance for the cases considered although it has a higher theoretical complexity.

As a perspective for future work, we plan to investigate extensions of our work to other fragments of SPARQL, and studying their effects on containment with ShEx.

References

[1] "Shape Expressions Primer," W3C, [Online]. Available: https://www.w3.org/2013/ShEx/Primer. [Accessed 1 9 2016].

[2] "ShEx," W3C, [Online]. Available: https://www.w3.org/2001/sw/wiki/ShEx. [Accessed 1 9 2016].

[3] M. Arenas and J. Pérez, "Querying Semantic Web Data with SPARQL," in *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2011.

[4] "ShEx/CurrentDiscussion," W3C, 2014.

[5] D. Colazzo and C. Sartiani, "Typing Regular Path Query Languages for Data Graphs," in *Proceedings of the 15th Symposium on Database Programming Languages*, 2015.

[6] E. V. Kostylev, J. L. Reutter, M. Romero and D. Vrgoč, "SPARQL with Property Paths," in *Proceedings of the 14th International Conference on The Semantic Web - ISWC 2015 - Volume 9366*, 2015.

[7] R. Pichler and S. Skritek, "Containment and Equivalence of Well-designed SPARQL," in *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2014.

[8] A. Letelier, J. Pérez, R. Pichler and S. Skritek, "Static Analysis and Optimization of Semantic Web Queries," *ACM Trans. Database Syst.*, vol. 38, no. 4, pp. 1-45, 2013.

[9] M. W. Chekol, J. Euzenat, P. Genevès and N. Layaïda, "SPARQL Query Containment Under SHI Axioms," in *AAAI Conference on Artificial Intelligence*, 2012.

[10] M. W. Chekol, J. Euzenat, P. Genevès and N. Layaïda, "SPARQL Query Containment under RDFS Entailment Regime," in *Automated Reasoning: 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, 2012.

[11] M. W. Chekol, "On the Containment of SPARQL Queries under Entailment Regimes," in *AAAI Conference on Artificial Intelligence*, 2016.

[12] S. Staworko, I. Boneva, J. E. L. Gayo, S. Hym, E. G. Prud'hommeaux and H. R. Solbrig, "Complexity and Expressiveness of ShEx for RDF," in *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, 2015.

[13] I. Boneva, J. E. L. Gayo, S. Hym, E. G. Prud'hommeau, H. Solbrig and S. Staworko, "Validating RDF with Shape Expressions," *CoRR*, 2014.

[14] "RDF 1.1 Primer," W3C, [Online]. Available: https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/. [Accessed 1 9 2016].

[15] "RDF 1.1 Concepts and Abstract Syntax," W3C, [Online]. Available: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/. [Accessed 1 9 2016].

[16] S. Harris and A. Seaborne, "SPARQL 1.1 Query Language," W3C, 3 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[17] J. Pérez, M. Arenas and C. Gutierrez, "Semantics and Complexity of SPARQL," *ACM Trans. Database Syst.*, 2009.

[18] E. Prud'hommeaux, J. E. Labra Gayo and H. Solbrig, "Shape Expressions: An RDF Validation and Transformation Language," in *Proceedings of the 10th International Conference on Semantic Systems*, 2014.

[19] E. Grädel, P. G. Kolaitis and M. Y. Vardi, "On the Decision Problem for Two-Variable First-Order Logic," vol. 3, 1997.

[20] E. Grädel and M. Otto, "On logics with two variables," vol. 224, 1999.

[21] K. Etessami, M. Y. Vardi and T. Wilke, "First-Order Logic with Two Variables and Unary Temporal Logic," vol. 179, 2002.

[22] H. Ganzinger, C. Meyer and M. Veanes, "The Two-Variable Guarded Fragment with Transitive Relations," in *14th Annual {IEEE} Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, 1999.

[23] "Apache Jena," 19 1 2016. [Online]. Available: https://www.w3.org/2001/sw/wiki/Apache_Jena. [Accessed 18 10 2016].

[24] G. Sutcliffe, "The TPTP Problem Library and Associated Infrastructure," *Journal of Automated Reasoning*, vol. 43, 2009.

[25] E. Prud'hommeaux, "Shape Expressions (ShEx) JSON Formats," [Online]. Available: http://shex.io/primer/ShExJ. [Accessed 18 10 2016].

[26] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda and P. Wischnewski, "SPASS Version 3.5," in *Automated Deduction -- CADE-22: 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, 2009.

[27] Schulz¸Stephan, "System Description: E 1.8," in *Proc. of the 19th LPAR, Stellenbosch*, 2013.

[28] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5, no. 2, 2009.

[29] P. Genevès and N. Layaïda, "A system for the static analysis of XPath," *ACM Trans. Inf. Syst.*, vol. 24, pp. 475-502, 2006.

[30] P. Genevès and N. Layaïda, "Deciding XPath Containment with MSO," *Data & Knowledge Engineering*, vol. 63, pp. 108-136, 2007.

[31] P. Genevès and N. Layaïda, "XML reasoning made practical," in *Proceedings of the 26th International Conference on Data Engineering*, 2010.

[32] M. W. Chekol, J. Euzenat, P. Genevès and N. Layaïda, "PSPARQL Query Containment," in *DBPL*, 2011.

[33] A. Letelier, J. Pérez, R. Pichler and S. Skritek, "SPAM: A SPARQL Analysis and Manipulation Tool," *PVLDB*, vol. 5, pp. 1958-1961, 2012.

[34] M. W. Chekol and G. Pirrò, "Containment of Expressive SPARQL Navigational Queries," in *The Semantic Web -- ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17--21, 2016, Proceedings, Part I*, 2016.

[35] M. W. Chekol, J. Euzenat, P. Genevès and N. Layaïda, "Evaluating and Benchmarking SPARQL Query Containment Solvers," in *Proceedings of the 12th International Semantic Web Conference - Part II*, 2013.

[36] J. E. Labra Gayo, E. Prud'hommeaux, I. Boneva, S. Staworko, H. R. Solbrig and S. Hym, "Towards an RDF Validation Language Based on Regular Expression Derivatives," in *Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), Brussels, Belgium, March 27th, 2015.*, 2015.

[37] I. Boneva, "Comparative expressiveness of ShEx and SHACL (Early working draft)," 2016.

APENDIX A: QUERIES

Q1a:

```
SELECT * WHERE {
 :product1 :is :Product .
 ?product :label  "label1".
 ?product :productFeature "ProductFeature1" .
 ?product :productFeature "ProductFeature2" . }
```

Q1b:

```
SELECT * WHERE {
 :product1 :is :Product .
 ?product :label ?label .
 ?product :productFeature "ProductFeature1" .
 ?product :productFeature "ProductFeature2" . }
```

Q2a:

```
SELECT * WHERE {
 :product1 :is :Product .
 :product1 :label ?label .
 :product1 :comment ?comment .
 :product1 :producer ?p .
 ?p :label ?producer .
 :product1 :publisher ?p .
 :product1 :productFeature "ProductFeature1"^^xsd:string .
 :product1 :productPropertyTextual1 ?propertyTextual1 .
 :product1 :productPropertyNumeric1 ?propertyNumeric1 .
 OPTIONAL
 { :product1 :productPropertyTextual2 ?propertyTextual2 }
 OPTIONAL
 { :product1 :productPropertyTextual3 ?propertyTextual3 }
 OPTIONAL
 {:product1 :productPropertyNumeric2 ?propertyNumeric2 .
  ?propertyNumeric2 :value "123" }
}
```

Q2b:

```
SELECT * WHERE {
 :product1 :is :Product .
 :product1 :label ?label .
 :product1 :comment ?comment .
 :product1 :producer ?p .
 ?p :label ?producer .
 :product1 :publisher ?p .
 :product1 :productFeature "ProductFeature1" .
 :product1 :productPropertyTextual1 ?propertyTextual1 .
 :product1 :productPropertyNumeric1 ?propertyNumeric1 .
 OPTIONAL
 { :product1 :productPropertyTextual2 ?propertyTextual2 }
 OPTIONAL
 { :product1 :productPropertyTextual3 ?propertyTextual3 }
}
```

Q3a:

```
SELECT * WHERE {
 ?product :is :Product .
 ?product :label ?label .
 ?product :productFeature "ProductFeature1" .
 ?product :productPropertyNumeric1 ?p1 .
 ?product :productPropertyNumeric3 ?p3 .
 OPTIONAL
 {?product :productFeature "ProductFeature2" .
  ?product :rating ?rating }
}
```

Q3b:

```
SELECT * WHERE {
 ?product :is :Product .
 ?product :label ?label .
 ?product :productFeature "ProductFeature1" .
 ?product :productPropertyNumeric1 ?p1 .
 ?product :productPropertyNumeric3 ?p3 .
}
```

Q4a:

```
SELECT *  WHERE {
 ?review :reviewFor :product1 .
 ?review :title ?title .
 ?review :text ?text .
 ?review :reviewDate ?reviewDate .
 ?review :reviewer ?reviewer .
 ?reviewer :is :Reviewer .
 ?reviewer :name  "Name1".
 ?reviewer :name  "Name2".
 OPTIONAL { ?review :rating ?rating1 . }
 OPTIONAL { ?review :rating ?rating2 . }
 OPTIONAL { ?review :rating ?rating3 . }
 OPTIONAL { ?review :rating ?rating4 . }
}
```

Q4b:

```
SELECT *  WHERE {
 ?review :reviewFor :product1 .
 ?review :title ?title .
 ?review :text ?text .
 ?review :reviewDate ?reviewDate .
 ?review :reviewer ?reviewer .
 ?reviewer :is :Reviewer .
 ?reviewer :name ?reviewerName .
}
```

Q5a:	Q5b:

SELECT * WHERE {
:product1 :is :Product .
:product1 :label ?productLabel .
OPTIONAL {
?offer :product :product1 .
?offer :price ?price .
?offer :vendor ?vendor .
?vendor :label ?vendorTitle .
?vendor :country :FR .
?offer :publisher ?vendor .
?offer :validTo ?date .
}
OPTIONAL {
?review :is :Review .
?review :reviewFor :product1 .
?review :reviewer ?reviewer .
?reviewer :name ?revName .
?review :title ?revTitle .
?revTitle :value "title1" .
OPTIONAL { ?review :rating1 ?rating1 . }
OPTIONAL { ?review :rating2 ?rating2 . }
}
}
	SELECT * WHERE {
	:product1 :is :Product .
	:product1 :label ?productLabel .
	OPTIONAL {
	?offer :product :product1 .
	?offer :price ?price .
	?offer :vendor ?vendor .
	?vendor :label ?vendorTitle .
	?vendor :country :FR.
	?offer :publisher ?vendor .
	?offer :validTo ?date .
	}
	OPTIONAL {
	?review :is :Review .
	?review :reviewFor :product1 .
	?review :reviewer ?reviewer .
	?reviewer :name ?revName .
	?review :title ?revTitle .
	?revTitle :value "title2" .
	OPTIONAL { ?review :rating1 ?rating1 . }
	OPTIONAL { ?review :rating2 ?rating2 . }
	}
	}

APENDIX B: SCHEMAS

Schema 1:	Schema 2:	Schema 3:	Schema 4:

:Product {
:is [:Product] ?,
:label xsd:string ? ,
:productFeature xsd:string ?
}
 	
:Reviewer {
:is [:Reviewer] ?
}
 	
:Review {
:is [:Review] ?
}
	:Product {
	:is [:Product] ?,
	:label xsd:string ? ,
	:productFeature xsd:string *
	}
	
	:Reviewer {
	:is [:Reviewer] ?,
	:name xsd:string ?
	}
	
	:Review {
	:is [:Review] ?
		:Product {
		:is [:Product] ?,
		:label xsd:string ? ,
		:productFeature xsd:string *
		}
		
		:Reviewer {
		:is [:Reviewer] ?,
		:name xsd:string *
		}
		
		:Review {
		:is [:Review] ?,
		:reviewer @:Reviewer ?,
		:title xsd:string ?
		}
			:Product {
			:is [:Product] ?,
			:label xsd:string ?,
			:productFeature xsd:string *,
			:productPropertyNumeric2 @:PropertyNumeric ?
			}
			
			:Reviewer {
			:is [:Reviewer] ?,
			:name xsd:string *
			}
			
			:Review {
			:is [:Review] ?
			}
			
			:PropertyNumeric {
			:is [:PropertyNumeric] ?,
			:value xsd:integer ?
			}