SPARQL Query Containment with ShEx Constraints

Abdullah Abbas, Pierre Genevès, Cécile Roisin, Nabil Layaïda

SPARQL Query Containment with ShEx Constraints

Abdullah Abbas, Pierre Genevès[(1)], Cécile Roisin, and Nabil Layaïda

Univ. Grenoble Alpes, CNRS, Grenoble INP*, Inria, LIG, F-38000 Grenoble France
[firstname.lastname]@inria.fr, @cnrs.fr[(1)]

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 propose a sound and complete procedure for the problem of containment with ShEx, considering several SPARQL fragments. Particularly our procedure considers OPTIONAL query patterns, that turns out to be an important fragment to be studied with schemas. We then show the complexity bounds of our problem with respect to the fragments considered. To the best of our knowledge, this is the first work addressing SPARQL query containment in the presence of ShEx constraints.

1 Introduction

ShEx (or Shape Expressions) is intended to be an RDF constraint language [19]. It can be used to validate documents and communicate expected graph patterns. Static analysis and query optimisation can make a considerable benefit from the presence of schemas when used to infer satisfiability/unsatisfiabliity of queries and relations between queries (such as containment and equivalence) by utilising 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, our purpose is to statically analyse 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 [1]. A solution mapping is a mapping from a set of variables to a set of values, thus designating an answer for a query. A solution mapping σ_1 is subsumed by another solution mapping σ_2 written as $\sigma_1 \sqsubseteq \sigma_2$ if all the variables of σ_1 are also in σ_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 $\sigma_1 \in \Omega_1$ there exists $\sigma_2 \in \Omega_2$ such that $\sigma_1 \sqsubseteq \sigma_2$.

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:

* Institute of Engineering Univ. Grenoble Alpes

Q_1: {?x :producer :p1 . ?x :feature "feature1"}
OPT {?x :feature "feature2" . ?x :expiryDate ?d}
Q_2: {?x :producer ?y . ?x :feature "feature1"}

Without constraints, no containment relation holds between these two queries. However, consider the following ShEx constraints defined for a $\langle Product \rangle$ 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 $\langle Company \rangle$, 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. In such case, we can deduce that the containment relation $Q_1 \sqsubseteq Q_2$ holds between the two queries.

There are several kinds of ShEx constraint violations that may lead to a new conclusion about the containment of two queries. These include (1) cardinality constraint violations, (2) basic data type constraint violations (like xsd:string, xsd:data ...), and (3) ShEx type definition violations (like @$\langle Company \rangle$ type).

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. While these slight differences in data shapes may become a hassle for users to track individually, the use of OPT patterns in SPARQL provides a way to ask for constraints that are not necessarily applicable, and that is why the study of the optional fragment is particularly interesting.

In this work we define a sound and complete procedure for containment of SPARQL query fragments in the presence of a ShEx schema, based on the usage of ShEx validators and query containment solvers that don't consider any schema constraints. We also study the complexity of the problem. The results vary from NP-c to Π_2^P-c according to the fragment considered. We also provide a higher NEXP complexity bound for further fragment extensions (FILTERS, MINUS, and property path patterns).

Paper Outline. In Sect. 2 we comment on the related works. In Sect. 3 we introduce some preliminaries necessary for understanding the rest of the paper. In Sect. 4 we define a query transformation function which is necessary for the definition of the containment procedure. In Sect. 5 a sound and complete

containment procedure is given for different SPARQL fragments. In Sect. 6 we derive complexity bounds of our problem. Finally, we conclude in Sect. 7.

2 Related Works

In [8], the authors proposed a schema language for edge-labeled data graphs (like RDFs), and then studied the satisfiability of 3 different classes of query languages (RPQs, NREs, and CRPQs) when such constraints are considered, but this study did not include containment. In [10, 11] the authors studied static analysis aspects of XPath using mu-calculus and Monadic Second-order Logic respectively, then the authors provided in [12] 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, yet these works inspire our work for SPARQL fragment extensions using logical formulas.

The work in [5] studied containment of PSPARQL, an extension of SPARQL 1.0 with paths and path constraints. In [14], the authors explored the complexity of containment and evaluation problems for fragments of SPARQL 1.1 property paths. The study in [18] provides complexity analysis for several fragments of SPARQL. Additionally, in [16] the containment of well-designed OPT queries is investigated. None of these works consider schemas in the study of containment.

The works in [4, 6, 7] study the containment problem with ontology languages and entailment regimes (SHI, RDFS, OWL...). Ontology languages put constraints on data, like schemas, but also allows for entailment of implicit data relations. The works on containment with ontology languages focus on entailment regimes employed in these languages, but not on the fragments of SPARQL with optional patterns which we want to consider.

For the works on ShEx, in [2, 20, 23] the expressiveness and validation complexity of ShEx was studied. The work in [9] proposes an implementation of shape expressions to RDF graphs.

With the deep static analysis of SPARQL queries - including containment - on one side, and the emergence of schema languages for RDF (like ShEx) on the other side, we find a value in investigating these targets in a common framework.

3 Definitions

3.1 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 [21, 15]. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions [22].

A SPARQL graph pattern is defined inductively from triple patterns. Given disjoint infinite sets of IRIs - Internationalised Resource Identifiers - (I), blank nodes (B), literals (L), and variables (V), we define a triple pattern as an instance

of $(I \cup B \cup V)(I \cup V)(I \cup B \cup L \cup V)$ denoted by $IBV \times IV \times IBLV$. A SPARQL graph pattern q is defined inductively from triple patterns as follows:

$q ::= t \mid q\ AND\ q' \mid q\ UNION\ q' \mid q\ OPT\ q' \mid q\ MINUS\ q' \mid q\ FILTER\ C$

where t is a triple pattern and C is a condition or a conjunction and/or disjunction of conditions on variables.

In order to reference different SPARQL fragments later, we define them as follows:

- BGP: This is the conjunctive fragment of SPARQL, i.e. the fragment that only allows using the AND operator between triples.
- AND-OPT: The fragment of SPARQL allowing the AND and OPT operators only. We particularly consider the well-designed patterns within this fragment (defined just after this list).
- AND-OPT-(UNION): The AND-OPT fragment extended with UNION on the top level only (external).
- AND-OPT-(UNION)-FILTER: The AND-OPT-(UNION) fragment extended with the FILTER operator. For this fragment we only consider filters that are decidable for query satisfiability [24], which is a necessary requirement for query containment, namely $FILTER(bound, =, \neq_c)$ and $FILTER(bound, \neq, \neq_c)$. Where $bound(?x)$ means that the variable $?x$ should be bound to a value in the query results. $=/\neq$ are the equality/inequality relations between variables. $\neq_c$ is the inequality of variable with respect to a constant belonging to $I \cup L$.
- AND-OPT-(UNION)-PP: The AND-OPT-(UNION) fragment extended with property path patterns from the SPARQL 1.1 syntax. These are regular expressions allowed in the predicate position.
- AND-OPT-(UNION)-MINUS: The AND-OPT-(UNION) fragment extended with the MINUS operator which puts constraints on situations that must not occur in the results.

Well-Designed OPT Patterns. Well-designed OPT patterns define a class of OPTIONAL patterns that have several desired properties [17], such as evaluation performance advantages.

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, and each O_j has the same form (also in normal form).*

These normal forms can be represented as pattern trees as described in [16]. For example, a query of the form $((P_1\ OPT\ (P_{11}\ OPT\ P_{111}\ OPT\ P_{112}))\ OPT\ P_{12})\ OPT\ P_{13}$, where each P_i is a BGP, can be represented as a pattern tree as follows:

In our work, we use pattern tree representations of the queries in order to study their containment with ShEx.

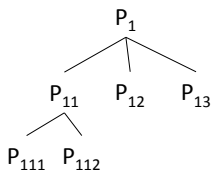

Fig. 1. Pattern tree example

3.2 ShEx

ShEx (or Shape Expressions) is intended to be an RDF constraint language. Logical operators in Shape Expressions such as 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 [20]. 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, $\langle Shape1 \rangle$ is a definition of a shape in ShEx, where a ShEx document contains definitions of several shapes.

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

In the previous example, ex:name and ex:phone are NameClasses 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.

Abstract Syntax of ShEx. 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: $e ::= \epsilon \mid \Sigma \times \Gamma \mid e^* \mid (e \text{“|”} e) \mid (e \text{“||”} e)$, where “|” is a disjunction, “||” is an unordered concatenation, and “$*$” is an unordered Kleene star. This 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. In the sequel we write $(a, t) \in \Sigma \times \Gamma$ simply as $a :: t$.

A shape expression schema (ShEx), or simply schema, is a tuple $S = (\Sigma, \Gamma, \delta)$, where Σ is a finite set of edge labels, Γ is a finite set of types, and δ is a type definition function that maps elements of Γ to shape expressions e over $\Sigma \times \Gamma$. If the δ is not defined for some type $t \in \Gamma$, the default definition is $\delta(t) = \epsilon$.

Semantics of ShEx. [20] Semantically, an RDF graph is valid against a ShEx 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.

We assume a fixed graph $G = (V, E)$ which resembles an RDF graph, and a fixed schema $S = (\Sigma, \Gamma, \delta)$. A typing of G w.r.t. S is a function $\lambda : V \to 2^\Gamma$ that associates with every node of G a set of types.

Next, the conditions that a typing needs to satisfy are identified. Given a typing λ and a node $n \in V$ we define the neighborhood-typing of n w.r.t. λ as bag over $\Sigma \times 2^{\Gamma}$ as $neighborTyping^{\lambda}_{G}(n) = \{|a :: \lambda(m) \mid (n, a, m) \in E|\}$.

Now, λ is a valid typing of S on G if and only if every node satisfies the type definitions of its associated type i.e., for every $n \in V$, $neighborTyping^{\lambda}_{G}(n) \in \delta(t)$, for all $t \in \lambda(n)$.

4 Query Transformation

Query transformation is a process in which we rewrite a query, where the resulting query is equivalent to the original query given that the ShEx constraints hold on the data sets. Two queries are considered to be equivalent if they always give the same execution results.

The resulting query transformations defined in this section has several utilisations, namely for optimisation purposes, especially that they are equivalent to and smaller than the original queries. We use them in this work particularly for defining containment in Sect. 5.

Before defining the transformation procedures, we give some preliminary definitions.

Definition 1. *Given a set of triple patterns P, $\mathcal{RDF}(P)$ is a function that yields a set of RDF triples by replacing each variable in P by a fresh IRI. The replacement is unique for each variable name.*

According to the previous definition, there always exists a homomorphism from the triples graph of P to the triples graph of $P' = \mathcal{RDF}(P)$. In fact, P' is an RDF data set that can be validated against a ShEx schema.

Definition 2. *Given two sets of RDF triples D_1 and D_2 and a ShEx schema S, we say that D_2 is a complement of D_1 w.r.t. S, if:*

1. *$D_1 \subseteq D_2$*
2. *D_2 is valid w.r.t. S*

Definition 3. *Given a ShEx schema S, the minimals discarding ShEx schema of S is given by the function $\mathcal{MIN}_0(S)$, and is defined by replacing all minimal cardinality constraints of S by zeros. (i.e. all cardinality constraints $[m, n]$, $+$ and 1 respectively, are replaced with $[0, n]$, $*$ and ? (optional) respectively).*

4.1 BGP Transformation

Query transformation of a BGP query is based on the RDF document validation. RDF validation against ShEx is defined with its NP-complete complexity in [23].

Definition 4 (Query Transformation). *For a BGP SPARQL query Q and a ShEx schema S, the query transformation function $\mathcal{T}_S$ is defined as follows:*

$$\mathcal{T}_S(Q) = \begin{cases} Q, & \text{if } \mathcal{RDF}(Q) \text{ is valid w.r.t. } \mathcal{MIN}_0(S) \\ \text{empty query}, & \text{otherwise} \end{cases}$$

The validation against $\mathcal{MIN}_0(S)$ is due to the fact that the query triples do not catch the complete data structure. Indeed, queries by nature are just partial representations of the constraints on the data that should be extracted.

4.2 AND-OPT Transformation

We extend the BGP transformation to a more interesting SPARQL fragment for our problem, the AND-OPT fragment. The results in this case will be a modified AND-OPT query that is equivalent to the original query, by applying two steps: (1) Eliminating non-valid OPT patterns, and (2) replacing some OPT operators with AND operators.

For the step (1), if we find out that some OPTIONAL pattern will never return results due to the ShEx constraints, the new query that results from this transformation is by omitting this OPTIONAL pattern.

Consider the following SPARQL query:
Q: {`:p1 :producer ?y`} `OPT` {`:p1 :review ?z`}
and the following ShEx schema (a minimals discarding ShEx schema):

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

We consider two RDF triple sets for validation against the ShEx schema, {`:p1 :producer :y`} which is valid, and {`:p1 :producer :y. :p1 :review :z`} - the optional pattern with its parent - which is not valid. As a result of this validation step, we rewrite the query by removing the optional pattern which corresponds to the RDF triple set which is not valid, and thus we get:
Q': {`:p1 :producer ?y`}

For step (2), we check if it is possible to replace some OPT operators with the AND operator. Considering the pattern tree representation of a query this operation can be described by uniting two directly connected nodes into one node, one of which is a child node, and the other is a parent node.

To show this by example, consider the following query:
Q: {`?x :name ?n`} `OPT` {`?x :phone ?p`}
and the following ShEx shape:

```
<Person> {
:name xsd:string ,
:phone xsd:string }
```

According to this ShEx shape definition, we know that :name and :phone will always occur together. Thus the right hand side of the OPT pattern will always occur with the left hand side of it. We therefore deduce that the previous query Q is equivalent to another query Q' without an OPT pattern.
Q: {`?x :name ?n. ?x :phone ?p`}

Two nodes in a query pattern tree must be merged into one node (the parent node), if and only if the triples of the child node will necessarily return results whenever the parent node returns results. We apply this check on every

pair of of parent-child nodes in the query pattern tree in order to get the final transformation of the query.

The transformations described in the latter examples for the AND-OPT SPARQL fragment are given formally in Definition 6.

Definition 5. *Given a pattern tree $\mathcal{P}$, and a node n of P, we define $\mathcal{R}_{\mathcal{P}}(n)$ to be the union of the set of triples of n and the set of triple of all its parent nodes up to the root node.*

Definition 6 (Query Transformation). *For an AND-OPT SPARQL query Q, its pattern tree representation $\mathcal{P}$, and a ShEx schema S, the query transformation function $\mathcal{T}_S$ is defined by the following steps:*

1. *For each node n of $\mathcal{P}$, if $\mathcal{R}_{\mathcal{P}}(n)$ is not valid w.r.t. $\mathcal{MIN}_0(S)$, then eliminate n and all its descendants from $\mathcal{P}$. Let $\mathcal{P}'$ be the new pattern tree after the validation of all the nodes of $\mathcal{P}$.*
2. *For each pair of nodes n_1 and n_2 of $\mathcal{P}'$, such that n_1 is the parent of n_2, if it is necessary for every complement of $\mathcal{RDF}(n_1)$ to include the RDF triples of $\mathcal{RDF}(n_2)$ according to S, then merge n_1 and n_2 into one node. Let P'' be the new pattern tree obtained.*

We define $\mathcal{T}_S(\mathcal{P}) = \mathcal{P}''$.

4.3 AND-OPT-(UNION) Transformation

For the AND-OPT SPARQL fragment extended with UNION at the top level, the same procedure can be applied on each UNION pattern separately.

5 Query Containment with ShEx

In this section we show how SPARQL query containment with ShEx can be done by benefiting from the transformations of Sect. 4.

We first apply the transformation procedure on the two queries to be checked for containment based on a given ShEx schema. The resulting transformations are then checked for containment without considering the ShEx document using query containment solvers as the one proposed in [18]. If the containment of the query transformations hold, then the containment of the original queries with the consideration of ShEx holds.

We briefly describe the containment procedure - without ShEx, displayed in the following lemma taken from [16]. The lemma provides the necessary and sufficient conditions for the deciding containment of a well-designed OPT SPARQL queries. The conditions are formulated in terms of pattern trees.

Lemma 1. *Consider two well-designed pattern trees $\mathcal{T}_1$ and $\mathcal{T}_2$ with roots r_1 and r_2, respectively. Then $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ if and only if for every subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1, there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ rooted at r_2 such that:*

1. *$vars(\mathcal{T}_1') \subseteq vars(\mathcal{T}_2')$, and*

2. *there exists a homomorphism from the triples in* $\mathcal{T}_2'$ *to the triples in* $\mathcal{T}_1'$ *that is the identity over* $vars(\mathcal{T}_1')$.

We notice that a pattern tree containment relation $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ also yields the query containment corresponding to these pattern trees (lets say $q_1 \sqsubseteq q_2$) [16].

Definition 7. *Given two queries* q_1 *and* q_2*, we define the relation* $q_1 \sqsubseteq_S q_2$ *to mean that* $q_1 \sqsubseteq q_2$ *holds in the presence of a ShEx schema* S.

Theorem 1. *Given two queries* q_1 *and* q_2*, and their corresponding transformations* q_1' *and* q_2' *according to a ShEx schema* S*, the containment relation* $q_1 \sqsubseteq_S q_2$ *holds if and only if* $q_1' \sqsubseteq q_2'$ *holds.*

Proof. The soundness of our procedure is evident from the fact that the transformations are equivalent to the original queries in the presence of the ShEx constraints. We use the empty schema as a transformation, or we eliminate parts of the queries only when we are sure that these parts will not return results according to the given schema.

For the completeness of the procedure, we prove it according to the corresponding fragment for each case.

1. For the BGP SPARQL fragment, assume we have two BGP queries q_1 and q_2 and their corresponding transformations q_1' and q_2' according to a ShEx schema S. For the completeness of the procedure, our purpose now is to prove that if $q_1' \not\sqsubseteq q_2'$, then $q_1 \not\sqsubseteq_S q_2$. For the case where q_1' is an empty query, $q_1' \sqsubseteq q_2'$ always holds, since the empty query is contained in every other query, and therefore the assumption condition can never happen. For the case where only q_2' is an empty query, that means that also q_2 will never return results due to a violation to the ShEx rules. No query can be contained in a query that does not return results except the empty query, and since we know that q_1 may return results due to the absence of any ShEx violation, then $q_1 \not\sqsubseteq_S q_2$ always holds. The final case is when both q_1' and q_2' are kept exactly the same as q_1 and q_2. In the latter case, if $q_1' \not\sqsubseteq q_2'$, then there exists no homomorphism from q_2' to q_1'. Given that the triples of q_1' don't violate the ShEx schema rules, then there exists a data set D which is a complement of $\mathcal{RDF}(q_1')$ w.r.t. S. BGP query solving is based on homomorphism from the set of query triple patterns to the set of RDF triples ([22]). A solution for q_1 necessarily exists in the proposed data set since the homomorphism exists by our proposal. Now we assume that the same solution also holds for q_2 and conclude a contradiction. If the same solution holds for q_2, then there exists a homomorphism from its triples patterns to D. Since all variables of q_1 are replaced with fresh IRIs, then a homomorphism is also necessary to hold from the triple patterns of q_2' to the triple patterns of q_1', and thus we conclude a contradiction because this homomorphism is a sufficient condition for deriving that the containment $q_1' \sqsubseteq q_2'$ holds (condition from [18]).
2. For the AND-OPT SPARQL fragment, assume we have two AND-OPT queries q_1 and q_2 and their corresponding transformations q_1' and q_2' according to a ShEx schema S. We show that for a transformation q' of any

query q as proposed in Sect. 4, the containment $q' \sqsubseteq q$ always holds. This follows from the fact that our transformation includes only elimination of optional parts of the query and transformation of other optional conditions into necessary conditions (transformation of OPT operators into AND operators). Both of the transformations make the query more restrictive in the meaning that it eliminates some solutions but never adds solutions to the original query. Assume $q'_1 \not\sqsubseteq q'_2$, our purpose is to show that $q_1 \not\sqsubseteq_S q_2$. Since $q'_1 \not\sqsubseteq q'_2$, then for some subtree $\mathcal{T}'_1$ of q'_1, there doesn't exist a subtree of q'_2 with the homomorphism condition of Lemma 1. On the other hand, there exists a data set D which is a complement of $\mathcal{RDF}(\mathcal{T}'_1)$ w.r.t. S. A solution for q'_1 necessarily exists in D. If this solution is also a solution for q'_2, and thus for q_2, then a homomorphism must hold from $\mathcal{T}'_2$ of q'_2 to the D, and thus there exists a homomorphism from $\mathcal{T}'_2$ to $\mathcal{T}'_1$, that necessarily doesn't hold due to the fact that $q'_1 \not\sqsubseteq q'_2$, and therefore a contradiction is derived.

3. For the AND-OPT-(UNION) SPARQL fragment, the same proof holds as for the AND-OPT fragment, except that instead of proposing complement data set that has a solution for q_1, leading to a contradiction when assuming it to have a solution for q_2, we alternatively propose multiple data sets, each corresponding to a top level UNION part of the query, and deriving a contradiction for each of the proposed data sets. □

6 Complexity

In this section we study the complexity of SPARQL query containment with ShEx with respect to different SPARQL fragments.

We show that the complexity varies from NP-complete to Π_2^P-complete for the SPARQL fragments BGP, AND-OPT, and AND-OPT-(UNION). We also extend these fragments to include filter, property path patterns, and the MINUS operator whose containment problem is in the NEXP time complexity class, yet this is not shown to be an upper bound.

6.1 SPARQL AND-(OPT)-(UNION) Fragments

Theorem 2. *Containment with ShEx for the SPARQL BGP fragment is NP-complete.*

Proof. The complexity of containment of the SPARQL BGP fragment is NP-complete [3]. In the presence of ShEx constraints, a sufficient procedure to check containment is to first validate the BGP of each of the considered queries against the ShEx document. RDF validation against ShEx is NP-complete. An invalid query will return no results, and thus is contained in any other query. Otherwise, the normal containment procedure (without ShEx) is applied. Then the BGP fragment containment with ShEx is also in NP.

To show the NP-hardness of the problem, we argue that containment with ShEx is at least as hard as containment without ShEx which is shown to be

NP-complete for the considered fragment. A reduction from the containment problem to the containment with ShEx problem can be easily shown by assuming an empty schema. □

Theorem 3. *Containment with ShEx for the well-designed OPT SPARQL fragment is NP-complete.*

Proof. In [18], the authors studied the problem of containment of well-designed OPT SPARQL queries. The authors provide a procedure for solving the problem, and show the complexity of the problem to be NP-complete for this fragment.

The procedure we follow for deciding query containment of this SPARQL fragment with ShEx is based on both the query transformation described previously in this work, and the query containment procedures of [18]. Given two SPARQL queries in the well designed OPT fragment, their containment with ShEx can be decided by the two following steps:

1. Transform both queries. The results of these transformations are two new queries equivalent to the original queries respectively.
2. The two new resulting queries from the first step are used as an input of a general SPARQL containment solver (like the solver described in [18] for this fragment).

Validation of an RDF document against a ShEx document is NP-complete [23]. Step (1) of the procedure is a series of ShEx validations each of which is in NP. The number of validation considered is polynomial since for a given query pattern tree, the validation occurs on all possible branches, rather than subtrees. While the number of subtrees is exponential in a pattern tree, the number of branches is polynomial. As each branch is validated independently, so the result of each branch validation doesn't affect the ones of other branches. Step (2), which is the query containment problem, is NP-complete for the well-designed OPT fragment. Thus the complexity of containment with ShEx is in NP for this fragment.

To show the NP-hardness of the problem, we argue that containment with ShEx is at least as hard as containment without ShEx which is shown to be NP-complete for the considered fragment. A reduction from the containment problem to the containment with ShEx problem can be easily shown by assuming an empty schema. □

Theorem 4. *Containment with ShEx for the well-designed OPT SPARQL fragment extended with top level UNION is Π_2^P-complete.*

Proof. In [18], the authors also studied the problem of containment of the AND-OPT-(UNION) fragment. The authors provide a procedure for solving the problem, and show the complexity of the problem to be Π_2^P-complete.

The procedure we follow to for deciding query containment of this fragment with ShEx is similar to the one followed for the AND-OPT fragment, except that in step (2) we use the solver designed for the corresponding fragment. The usage of such solver will rise the complexity to Π_2^P.

To show the Π_2^P-hardness of the problem, we argue that containment with ShEx is at least as hard as containment without ShEx which is shown to be Π_2^P-complete for the considered fragment. A reduction from the containment problem to the containment with ShEx problem can be easily shown by assuming an empty schema. □

6.2 SPARQL AND-OPT-(UNION)-FILTER/PP/MINUS Fragment

For extending the query containment problem with ShEx to the SPARQL fragments including filters, property path patterns, and the MINUS operator, we use an imitation of our procedures with first-order logic (FOL). The basic idea is based on generating an FOL formula corresponding to our procedure and checking its validity with existing FOL theorem provers.

We use a decidable fragment of FOL with only 2 variables, known as FOL^2, whose satisfiability (and thus validity) is NEXP-complete [13]. An advantage of this method is that it allows to benefit from the highly optimised implementations of theorem provers.

A drawback of the problem solving with FOL is that the Kleene closure can be expressed only on atomic ShEx rules, but not on compound rules, which restricts the ShEx fragment allowed. We notice that we can avoid the Kleene closure limitation by adopting the normal procedure and using FOL particularly for the fragment extensions, and giving FOL validity feedback for the original procedure which supports Kleene closure everywhere.

Table 1 summarises our complexity results for the containment problem studied for the different fragments.

Table 1. Containment complexity

SPARQL Fragment	**No ShEx**	**ShEx-All**
BGP	*[Chandra, 1977]* NP-c	NP-c
AND-OPT	*[Pichler, 2014]* NP-c	NP-c
AND-OPT-(UNION)	*[Pichler, 2014]* Π_2^P-c	Π_2^P-c
AND-OPT-(UNION)-Minus	*[FOL^2]* NEXP	*[FOL^2]* NEXP
AND-OPT-(UNION)-FILTER	*[FOL^2]* NEXP	*[FOL^2]* NEXP
AND-OPT-(UNION)-PP	*[FOL^2]* NEXP	*[FOL^2]* NEXP
AND-OPT-(UNION)-FILTER-PP-MINUS	*[FOL^2]* NEXP	*[FOL^2]* NEXP

We currently have two working implementation prototypes, one directly based on the procedures described in this paper utilising existing ShEx validators and containment solvers, and another implementation based on the FOL imitation of the problem, with the drawback and advantages of each as mentioned previously.

7 Conclusion

In this paper we studied the problem of SPARQL query containment with ShEx constraints, and the OPT patterns were shown to be particularly interesting

to this study, due to its flexibility with constraints and the absence of similar studies in the literature. We showed how transformation of queries can be done based on customised validation procedures. Then we proposed a procedure for the problem of containment with ShEx, and the complexity related to the AND-OPT SPARQL fragment was shown to be NP-complete, and that of the AND-OPT SPARQL fragment extended with external UNION to be Π_2^P-complete. We finally mentioned that other fragment extensions can be adopted with FOL.

As a perspective for future work, we manage to adopt modified versions of the same techniques provided in this work for query constructs other than the SELECT. Other SPARQL constructs that can be adopted with further discussions include INSERT, DELETE, and CONSTRUCT.

References

1. Marcelo Arenas and Jorge Pérez. Querying semantic web data with sparql. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 305–316, New York, NY, USA, 2011. ACM.
2. Iovka Boneva, José Emilio Labra Gayo, Samuel Hym, Eric G. Prud'hommeaux, Harold R. Solbrig, and Slawek Staworko. Validating RDF with shape expressions. *CoRR*, abs/1404.1270, 2014.
3. Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, New York, NY, USA, 1977. ACM.
4. Melisachew Wudage Chekol. On the containment of sparql queries under entailment regimes. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 936–942. AAAI Press, 2016.
5. Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. Psparql query containment. In *DBPL*, 2011.
6. Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. Sparql query containment under rdfs entailment regime. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning: 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, pages 134–148, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
7. Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. Sparql query containment under shi axioms. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, pages 10–16. AAAI Press, 2012.
8. Dario Colazzo and Carlo Sartiani. Typing regular path query languages for data graphs. In *Proceedings of the 15th Symposium on Database Programming Languages*, DBPL 2015, pages 69–78, New York, NY, USA, 2015. ACM.
9. Jos Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, Slawek Staworko, Harold R. Solbrig, and Samuel Hym. *Towards an RDF Validation Language Based on Regular Expression Derivatives*, pages 197–204. 2015.
10. Pierre Genevès and Nabil Layaïda. A system for the static analysis of xpath. *ACM Trans. Inf. Syst.*, 24(4):475–502, October 2006.

11. Pierre Genevès and Nabil Layaïda. Deciding {XPath} containment with {MSO}. *Data & Knowledge Engineering*, 63(1):108 – 136, 2007. Data Warehouse and Knowledge Discovery (DAWAK ?05)7th International Congress on Data Warehouse and Knowledge Discovery (DAWAK?05).
12. Pierre Genevès and Nabil Layaïda. Xml reasoning made practical. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 1169–1172, March 2010.
13. Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 003 1997.
14. Egor V. Kostylev, Juan L. Reutter, Miguel Romero, and Domagoj Vrgoč. Sparql with property paths. In *Proceedings of the 14th International Conference on The Semantic Web - ISWC 2015 - Volume 9366*, pages 3–18, New York, NY, USA, 2015. Springer-Verlag New York, Inc.
15. Markus Lanthaler, Richard Cyganiak, and David Wood. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`.
16. Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. pages 89–100, 2012.
17. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, September 2009.
18. Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed sparql. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, pages 39–50, New York, NY, USA, 2014. ACM.
19. Eric Prud'hommeaux. Shape expressions (shex) primer. Technical report, W3C and MIT, February 2017. `http://shexspec.github.io/primer/`.
20. Eric Prud'hommeaux, Jose Emilio Labra Gayo, and Harold Solbrig. Shape expressions: An rdf validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems*, SEM '14, pages 32–40, New York, NY, USA, 2014. ACM.
21. Guus Schreiber and Yves Raimond. RDF 1.1 primer. W3C note, W3C, June 2014. `http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`.
22. Andy Seaborne and Steven Harris. SPARQL 1.1 query language. W3C recommendation, W3C, March 2013. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.
23. Slawek Staworko, Iovka Boneva, Jose E. Labra Gayo, Samuel Hym, Eric G. Prud'hommeaux, and Harold Solbrig. Complexity and Expressiveness of ShEx for RDF. In Marcelo Arenas and Martín Ugarte, editors, *18th International Conference on Database Theory (ICDT 2015)*, volume 31 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 195–211, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
24. Xiaowang Zhang, Jan Van Den Bussche, and François Picalausa. On the satisfiability problem for sparql patterns. *J. Artif. Int. Res.*, 56(1):403–428, May 2016.