Transforming Property Path Query According to Shape Expression Schema Update

Goki Akazawa∗, Naoto Matsubara† and Nobutaka Suzuki‡

University of Tsukuba, Tsukuba, Japan

Keywords: ShEx, Property Path, Schema Update.

Abstract: Suppose that we have a query $q$ under schema $S$ and then $S$ is updated. Then we have to update $q$ according to the update of $S$, since otherwise $q$ no longer reports correct answer. However, updating $q$ manually is often a difficult and time-consuming task since users do not fully understand the schema definition or are not aware of the details of schema update. In this paper, we consider transforming queries automatically according to schema update. We focus on Shape Expression (ShEx) and Property Path as schema and query language, respectively, and we take a structural approach to transform Property Path query. For a Property Path query $q$ and a schema update op to an ShEx schema $S$, our algorithm checks how op affects the structure of $q$ under $S$, and transforms $q$ according to the result.

1 INTRODUCTION

Schema plays an important role in management of various kinds of data, and the importance holds for RDF/graph data as well. Since user requirements to RDF data may change over time, schema tends to be updated continuously to meet the requirements. Here, suppose that we have a query $q$ written for data under schema $S$, then $S$ is updated, and that $q$ is (re)executed after the update. Such a situation often arises, e.g., (a) $q$ is embedded in a program code and the code is executed after a schema update, (b) $q$ is recorded in a user’s history and she/he tries to use $q$ again, and so on. In such cases, we have to update $q$ according to the update of $S$, since otherwise $q$ no longer reports correct answer. However, updating $q$ manually is often a difficult and time-consuming task, since users do not fully understand the schema definition or are not aware of the details of schema update.

To address the problem, in this paper we consider transforming queries automatically according to schema update. We focus on Shape Expression (ShEx) (Baker and Prud’hommeaux, 2019) and Property Path as schema and query language, respectively. Here, ShEx is a novel schema language for RDF and is already used in a number of areas (Thornton et al., 2019). For RDF data, there is another schema language, called SHACL (Knublauch and Kontokostas, 2017), but it has some differences. SHACL schema description tends to be more complicated due to its strict definition. On the other hand, ShEx has higher readability and is easy to handle although the vocabulary has some limitation. In addition, recursion is formally supported in the ShEx specification, but not in that of SHACL (depending on the implementation). As for query language, Property Path is a well-known path query language included in SPARQL 1.1.

In this paper, we first define update operations to ShEx schema, then propose an algorithm for transforming a given query into a new query according to schema update.

In this paper we take a structural approach to transform query. For a query $q$ and an update operation(s) $op$ to ShEx schema $S$, our algorithm checks how $op$ affects the structure of $S$, examines how the changes to $S$ affects the structure of $q$, and then transforms $q$ into new query $q'$ according to the result. Here, it is desirable that the transformed query $q'$ preserves the behaviour of $q$ as much as possible, i.e., the answer of $q'$ should be as close to that of its original query $q$ as possible. To examine the effectiveness of our structural approach, we made a small preliminary experiment. The result suggests that transformed queries obtained by our algorithm show rather good behaviors on this respect.

1.1 Related Work

For XML documents, a number of studies on schema updates have been made so far. Guerrini et al. pro-
posed update operations that assure any updated schema contains its original schema so that documents under an updated schema can be processed by existing query evaluation systems. Junedi et al. studied query-update independence analysis and showed that the performance of (Benedikt and Cheney, 2010) can be drastically enhanced in the use of μ-calculus (Junedi et al., 2012). Oliveira et al. proposed an algorithm for detecting possible problems affecting XQuery code according to XML Schema update (Oliveira et al., 2012). Wu et al. proposed an algorithm for correcting XSLT stylesheet according to DTD update (Wu and Suzuki, 2016).

For RDF/graph schema update, Chirкова and Fletcher proposed a model of RDF schema evolution (Chirкова and Fletcher, 2009) but no query transformation was considered. Bonifati et al. discussed evolution of property graph schema by using graph rewriting operations (Bonifati et al., 2019).

To the best of our knowledge, however, no studies on transforming Property Path query according to ShEx schema update have been made so far.

The rest of this paper is organized as follows. Section 2 gives some preliminary definitions. Section 3 shows some operations to types of ShEx schema and our algorithm. Section 4 presents the result of our preliminary evaluation experiment. Section 5 shows our conclusion.

2 PRELIMINARIES

Let Σ be a set of labels. A labeled directed graph (graph for short) is denoted G = (V, E), where V is a set of nodes and E ⊆ V × Σ × V is a set of edges. Let e ∈ E be an edge labeled by l ∈ Σ from a node v ∈ V to a node v′ ∈ V. Then e is denoted (v, l, v′), v is called source, and v′ is called target.

Unlike XML documents, in RDF/graph data the order among sibling nodes are less significant. Thus ShEx uses regular bag expression (RBE) to represent content model of type (Staworko et al., 2015). RBE is defined similarly to regular expression except that RBE uses unordered concatenation instead of ordered concatenation. Let Γ be a set of types. Then RBE over Σ × Γ is recursively defined as follows.

• ε and a :: t ∈ Σ × Γ are RBEs.

• If r1, r2, · · · , rk are RBEs, then r1[r2 · · · rk] is an RBE, where | denotes disjunction.

• If r1, r2, · · · , rk are RBEs, then r1 || r2 || · · · || rk is an RBE, where || denotes unordered concatenation.

• If r is an RBE, then r[i,m] is an RBE, where n ≤ m. In particular, r[1,1] = r[0,0], r[0,0] = r[0,1], and r[k] = r[k,1].

For example, let r = a :: t1 || (b :: t2[c :: t3]) be an RBE. Since || is unordered, r matches not only a :: t1 b :: t2 and a :: t1 c :: t3 but also b :: t2 a :: t1 and c :: t3 a :: t1.

A ShEx schema is denoted S = (Σ, Γ, δ), where Γ is a set of types and δ is a function from Γ to the set of RBEs over Σ × Γ. For example, let S = (Σ, Γ, δ) be a ShEx schema, where Σ = {a, b, c}. Γ = {t0, t1, t2, t3, t4}, and

δ(t0) = a :: t1 || b :: t3 || (c :: t2)*,
δ(t1) = b :: t3[c :: t4],
δ(t2) = c :: t3,
δ(t3) = ε,
δ(t4) = a :: t3.

For example, consider the graph G shown in Fig. 1(left). In RBE, a :: t matches an edge e if the label of e is a and the target node of e is of type t. Thus, assuming that each node vi of type tk matches the outgoing edges of vi. Then it is easy to verify that G is a valid graph of S.

The schema graph of a ShEx schema S = (Σ, Γ, δ) is a graph Gs = (Vs, Es), where Vs = Γ and Es = { t, a, t′ | δ(t) contains a :: t′ }. For example, Fig. 1(right) shows the schema graph of S.

Property Path query (query for short) over S is defined as follows.

• ε and any a ∈ Σ is a query. Here, query a matches an edge labeled by a.

• * is a “wildcard” query, which matches any edge.

• For a set of labels {a1, a2, · · · , ak}, |{a1, a2, · · · , ak} is a query. Here, | denotes negation and this query matches an edge whose label is not in {a1, a2, · · · , ak}.

• For a label a ∈ Σ, a−1 is a query, which matches the inverse of an edge labeled by a.

• For queries q1, q2, · · · , qk, q1.q2. · · · .qk and q1[q2. · · · .qk] are queries. The former matches a path p = p1.p2. · · · . pk if qj matches subpath pj for every 1 ≤ i ≤ k. The latter matches a path p if one of q1.q2. · · · .qk matches p.

• For a query q, q∗ is a query. This query matches a path p = p1.p2. · · · . pk if q matches subpath pj for every 1 ≤ i ≤ k (k ≥ 0).

In this paper, we focus on single source query traversal. For a graph G, query q, and node v0, the answer of q from v0 over G is a set of nodes v such that G contains a path from v0 to v whose sequence of labels is matched by q. For example, if q = a−1.[a, b], then the answer of q from v0 over G in Fig.1 is {v2}.
Figure 1: Valid graph $G$ and schema graph of $S$.

Let $G_S = (V_S, E_S)$ be the schema graph of $S$, $q$ be a query, and $t$ be a type of $S$. By $G_S(q, t)$ we mean the traversal area of $q$ from $t$ over $G_S$, that is, the subgraph of $G_S$ traversed by $q$ from $t$ over $G_S$. For example, let $G_S$ be the schema graph shown in Fig. 1(right), $q = \langle c^{-1} \rangle.a.b$. Then $G_S(q, t_1)$ is shown in Fig. 2. By $\text{Ans}(G_S(q, t))$, we mean the “answer” types of $G_S(q, t)$, i.e., the “answer” types obtained by traversing $q$ from $t$ over $G_S$. For example, in Fig. 2 $\text{Ans}(G_S(q, t_1)) = \{t_1, t_3\}$.

For example, let $G$ be a query, and Fig. 1(right), $q$ the subgraph of $S$ traversed by $q$ from $t$ over $G_S$. Then the tree representation of $q$ is shown in Fig. 3.

![Diagram](image)

3 QUERY TRANSFORMATION

In this section, we first define operations to types of ShEx schema. Then we present an algorithm for transforming a given query according to schema update.

3.1 Operation on Types

To represent schema update, we introduce update operations (operations for short) to types. First, we introduce tree representation of type. To identify the position of each node, an id based on Dewey ordering is given to each node. For example, let

$$ \hat{\delta}(t_0) = a::t''_1 \parallel (b::t_2|c::t_3). $$

Then the tree representation of $t_0$ is shown in Fig. 3. The id associated with each node is the position of the node.

We define the following eight operations to types of ShEx schema. Let $t$ be a type of a ShEx schema $S$.

- Changing label::type pair of type:
  - $\text{add}_l(t, i, t'')$: this operation adds label::type pair $t''$ at position $i$ of $\hat{\delta}(t)$ with $t''$.

- Adding/deleting type of schema:
  - $\text{add}_s(t, i, op)$: this operation adds an operator $op$ to $\hat{\delta}(t)$ at position $i$.  
  - $\text{del}_s(t, i)$: this operation deletes the operation at position $i$ from $\hat{\delta}(t)$.  
  - $\text{change}_s(t, i, op)$: this operation replaces the operation at position $i$ of $\hat{\delta}(t)$ with op.

- Changing operator ($|, ||, [n, m])$ of type:
  - $\text{add}_o(t, i, op)$: this operation adds a new type $t$ to $S$. Initially, $\hat{\delta}(t) = e$.

An update script is a sequence $s = op_1op_2\cdots op_n$ of operations. For example, consider $t_0$ in Fig. 3 and let $s = \text{add}_s(t_0, 1.1, d::t_3)$ be an update script. By applying $s$ to $t_0$, we obtain $\hat{\delta}(t_0) = d::t_3 \parallel a::t''_1 \parallel (b::t_2|c::t_3)$.  

![Diagram](image)
Let \( q \) be a query and \( S \) be a ShEx schema. In the above operations, \( addLt() \), \( addOp() \), \( delOp() \), \( changeOp() \), \( addType() \) do not affect \( q \) in that \( q \) remains “valid” against the update schema of \( S \). On the other hand, \( delLt() \), \( changeLt() \), \( delType() \) may affect \( q \), i.e., \( q \) may become “invalid” under the updated schema of \( S \) in that \( q \) may lose some part of answers that were obtained under \( S \). Thus, our algorithm shown below transforms \( q \) when \( delLt() \), \( changeLt() \), or \( delType() \) is applied to \( S \).

We finally note that when a ShEx schema \( S \) is updated, the data under \( S \) must also be updated according to the schema update. Thus we have developed a method for updating data according to schema update (details are omitted).

### 3.2 Algorithm

Our algorithm consists of Algorithms 1 and 2. Algorithm 1 is the main part of our algorithm. For a given update script \( s = op_1, op_2, \ldots, op_n \) on ShEx schema \( S \) and start type \( t_s \), the algorithm transforms a given query \( q \) according to \( s \). Let \( G_S \) be the schema graph of \( S \), and let \( G_S(q, t_s) \) be the traversal area of \( q \) from \( t_s \) (lines 1 and 2). First, we take copies \( H_S \) and \( G'_S \) of \( G_S(q, t_s) \) and \( G_S \), respectively (line 3). Then for each operation \( op_i \) of \( s \) the algorithm modifies \( H_S \) according to \( op_i \) (lines 4 to 27), and converts \( H_S \) to transformed query \( q' \) (line 28). The for loop in lines 4 to 27 proceeds as follows. The algorithm does nothing if \( op_i \) does not affect the traversal area \( H_S \) (lines 5 to 7). Otherwise, \( H_S \) (and \( G'_S \)) is modified according to \( op_i \) in lines 8 to 26, as follows.

- Lines 8 to 14 deal with \( changeLt(t_s, i, i', \cdot : i') \). This operation changes label: type pair \( i \) of \( \delta(t) \) at position \( i \) to \( i' \). According to this, we replace edge \( (t_s, i, t) \) with \( (t_s, i', t) \) in \( H_S \) and \( G'_S \). If \( i = i' \), then we are done. Otherwise, since \( t_s \) is changed to \( t' \), a path from \( t_s \) to some accepting node via the path may be disconnected by this change. To repair this, we find a set of simple paths \( P \) from \( t_s \) to \( t_i \) in \( G'_S \) by FindPaths and add each path \( p \in P \) to \( H_S \) to connect \( t_s \) and \( t_i \).

Here, for all types \( t, t' \), FindPaths (not shown) is a method for finding the set of simple paths \( P \) from \( t \) to \( t' \) over \( G'_S \) with inverse edge traversal allowed. But if the length of every simple path \( p \) exceeds a given threshold, FindPaths also traverses paths from \( t \) to the neighbours of \( t' \) and if shorter simple path(s) is found, then the FindPaths reports the shorter paths instead of \( P \).

- Lines 15 to 19 deal with \( delLt(t_s, i) \). This operation deletes the label: type pair \( i \) of \( \delta(t) \). According to this, we delete edge \( (t_s, i, t_i) \) from \( H_S \) and \( G'_S \). By this edge deletion \( t \) and \( t_i \) may be disconnected, thus we find paths from \( t \) to \( t_i \) over \( G'_S \) by FindPaths and add the paths to \( H_S \).

- Lines 20 to 26 deal with \( delType(t) \). This operation deletes type \( t \) from \( S \). Thus \( t \) and every edge incident to \( t \) is deleted from \( H_S \) and \( G'_S \). To repair this, we find the set \( T_t \) of nodes outgoing to \( t \) and the set \( T'_t \) of nodes incoming from \( t \), and then find paths from \( T_t \) to \( T'_t \) and add the paths to \( H_S \).

In line 28, ConstructPropertyPath (Algorithm 2) converts \( H_S \) to new query \( q' \). This is done by regarding \( H_S \) as an NFA \( M \) with start state \( t_s \) and the set \( \text{Ans}(G_S(q, t_s)) \) of accept states (line 2), constructing a DFA \( M' \) equivalent to \( M \) (line 3), and then converting \( M' \) into a query \( q' \) (line 4). The conversion from \( M' \) to \( q' \) is done by using an extension of the state elimination method for DFA.

### 4 PRELIMINARY EXPERIMENT

In this section, we present the result of our preliminary evaluation experiment. We applied our algorithm to several queries in order to examine if the transformed queries show “good” behaviour in the sense that the answers of the original queries are maintained after schema update.

The data used in this experiment is Japanese Textbook LOD (Egusa and Takaku, 2018a; Egusa and Takaku, 2018b). Here, Japanese Textbook LOD is RDF data compiled from a collection of textbooks that has been organized over the years by NIER Education Library and Textbook Research Center Library. The data structure of Japanese Textbook LOD is illustrated in Fig. 4. Japanese Textbook LOD consists of 233,001 triples of the Turtle format. The data size is 12MB.

In this experiment, we manually created five queries and short schema updates shown in Table 1. We transformed each query by the algorithm (and the data is also transformed according to the schema update), executed the original and transformed queries over the original and updated data, respectively, and calculated the recall, precision and F-measure values. Let \( q \) be a query, \( q' \) be the transformed query of \( q \), and \( \text{Ans}(q) \) be the set of obtained answer nodes of \( q \). The recall of \( q' \) w.r.t. \( q \) is defined as follows.

\[
\text{recall}(q, q') = \frac{|\text{Ans}(q) \cap \text{Ans}(q')|}{|\text{Ans}(q)|}.
\]

Similarly, the precision of \( q' \) w.r.t. \( q \) is defined as follows.

\[
\text{precision}(q, q') = \frac{|\text{Ans}(q) \cap \text{Ans}(q')|}{|\text{Ans}(q')|}.
\]
Table 1: Original query and update script.

<table>
<thead>
<tr>
<th>No.</th>
<th>(a) original query and (b) update script</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(a) catalogue.school</td>
</tr>
<tr>
<td></td>
<td>(b) del JL(Textbook, 6) add JL(Textbook, 1, subjectType :: SubjectType)</td>
</tr>
<tr>
<td>2</td>
<td>(a) catalogue^{-1}.publisher^{-1}.curriculum.hasSubjectArea.hasSubject</td>
</tr>
<tr>
<td></td>
<td>(b) del JL type(Publisher) del JL(Catalogue, 1)</td>
</tr>
<tr>
<td>3</td>
<td>(a) curriculum^{-1}.school</td>
</tr>
<tr>
<td></td>
<td>(b) del JL(Textbook, 5) del JL(SubjectArea)</td>
</tr>
<tr>
<td>4</td>
<td>(a) catalogue</td>
</tr>
<tr>
<td></td>
<td>(b) del JL(Textbook, 6) add JL(SubjectType, 3, hasSubject :: Subject)</td>
</tr>
<tr>
<td>5</td>
<td>(a) subjectArea^{-1}.curriculum.hasSubjectArea.hasSubject.school</td>
</tr>
<tr>
<td></td>
<td>(b) del JL type(Subject).change JL(CurriculumGuideline, 1, version :: Version)</td>
</tr>
</tbody>
</table>

Table 2 lists the transformed queries for the original queries and their recall, precision and F-measure values. The average F-measure of the five queries is 0.87, and thus the transformed queries showed rather good behaviors overall. However, the first, second, and fourth transformed queries missed some correct answers, especially the second one. A reason for this is as follows. Japanese Textbook LOD schema contains many edges associated with "*" or "?". Such edges are “optional” and their corresponding edge may not appear in the RDF data. Therefore, if transformed query contains a label of such “optional” edges, the answers obtained by the transformed queries do not coincide with those of their original queries. In the experiment, “hasSubject” of the second query has such optional edges.

The results show some potential of our approach, however, the queries and schema updates used in the experiment are very limited and we need to conduct more experiments by using more queries and update operations. This is left as a future work.

5 CONCLUSION

In this paper, we first defined update operations to ShEx schema, and then proposed an algorithm for transforming a given query into a new query according to schema update. We made a small preliminarily experiment and the results showed that queries transformed by our algorithm shows good behaviour in that their answers were close to that of the original queries.

However, we have to some works to do. First, the dataset used in our experiment is limited. Thus we need to conduct more experiments with a variety kinds of datasets. In the experiment, each schema update consists of only two update operations. However, we need to examine schema update consisting of more update operations in order to reflect real schema update situations. Moreover, there are some ShEx elements missing in our paper, e.g., negation. Thus we plan to consider more broader class of ShEx schema.
Table 2: Transformed Query and Recall, Precision, F-measure.

<table>
<thead>
<tr>
<th>No.</th>
<th>Transformed Query</th>
<th>recall</th>
<th>precision</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>publisher.catalogue.school</td>
<td>0.88</td>
<td>0.99</td>
<td>0.93</td>
</tr>
<tr>
<td>2</td>
<td>catalogue⁻¹.curriculum.hasSubjectArea.hasSubject</td>
<td>0.70</td>
<td>0.50</td>
<td>0.59</td>
</tr>
<tr>
<td>3</td>
<td>curriculum⁻¹.publisher⁺.catalogue.school</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>(publisher.catalogue</td>
<td>subjectArea).school</td>
<td>0.88</td>
<td>0.77</td>
</tr>
<tr>
<td>5</td>
<td>(subjectArea⁻¹.curriculum.SubjectArea)⁻¹.subjectType⁺.school</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>average of five queries</td>
<td>0.89</td>
<td>0.85</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Algorithm 1: Query Transformation.

Input: ShEx schema $S = (\Sigma, \Gamma, \delta)$, update script $s = \sigma_1 \sigma_2 \cdots \sigma_n$ to $S$, query $q$, type $t_0 \in \Gamma$

Output: query $q'$

1: construct the schema graph $G_S$ of $S$
2: construct the traversal area $G_S(q, t_0)$ of $q$ from $t_0$ on $G_S$
3: $H_S \leftarrow G_S(q, t_0)$; $G'_S \leftarrow G_S$
4: for $i = 1, 2, \ldots, n$ do
5: if $\sigma_i$ does not affect $H_S$ then
6: continue
7: end if
8: if $\sigma_i = \text{change}_J t_i$ then
9: let $l_i :: t_i$ be the label:type pair at position $i$ of $\delta(t)$
10: replace $(t_i, l_i, t_i')$ with $(t_i', l_i', t_i')$ in $H_S$ and $G'_S$
11: if $t_i \neq t_i'$ then
12: $P \leftarrow \text{FindPaths}(G'_S, t_i, t_i')$
13: add all $p \in P$ to $H_S$
14: end if
15: else if $\sigma_i = \text{del}_J t_i$ then
16: let $l_i :: t_i$ be the label:type pair at position $i$ of $\delta(t)$
17: delete $(t_i, l_i, t_i')$ from $H_S$ and $G'_S$
18: $P \leftarrow \text{FindPaths}(G'_S, t_i, t_i')$
19: add all $p \in P$ to $H_S$
20: else if $\sigma_i = \text{del}_J \text{type}(t)$ then
21: $T_S \leftarrow \{ t_i \mid (t_i, l_i, t_i') \text{ is an edge from } t_i \text{ to } t \text{ in } G'_S \}$
22: $T_S' \leftarrow \{ (t_i, l_i, t_i') \mid (t_i, t_i') \text{ is an edge from } t_i \text{ to } t_i' \text{ in } G'_S \}$
23: delete $t_i$ and every edge adjacent to $t_i$ from $H_S$ and $G'_S$
24: $P \leftarrow \{ (p \mid p \in \text{FindPaths}(G'_S, t_i, t_i') \mid T_S, T_S' \in T_S) \}$
25: add all $p \in P$ to $H_S$
26: end if
27: end for
28: $q' \leftarrow \text{ConstructPropertyPath}(H_S, t, \text{ans}(G_S(q, t)))$
29: return $q'$

Algorithm 2: ConstructPropertyPath.

Input: traversal area $H_S$, set of types $T$, query $q'$

Output: query $q'$

1: let $V$ and $E$ be the sets of nodes and edges of $H_S$, respectively
2: construct an NFA $M = (Q, \Sigma, \delta, t_0, \text{Ans})$, where $Q = V$ and $\delta$ is a transition function s.t. $\delta(t, a) = t'$ iff $(t, a, t') \in E$
3: construct a DFA $M'$ equivalent to $M$
4: construct a query $q'$ from $M'$
5: return $q'$

REFERENCES


