SN Computer Science (2022) 3:196
https://doi.org/10.1007/s42979-022-01086-0

ORIGINAL RESEARCH

An Algorithm for Transforming Property Path Query Based on Shape Expression Schema Update

Goki Akazawa[1] · **Naoto Matsubara**[1] · **Nobutaka Suzuki**[1]

Received: 7 July 2021 / Accepted: 5 March 2022 / Published online: 23 March 2022

Abstract

In this paper, we consider transforming queries automatically according to schema update. Suppose that we have a query q under schema S and that S is updated. Then due to the schema update we have to update q accordingly, otherwise q no longer provides 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 the 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 Property Path query q and schema update s to ShEx schema S, the proposed algorithm checks how s affects the structure of q under S, and transforms q according to the result. Our experimental result suggests that the proposed algorithm transforms Property Path queries appropriately according to ShEx schema updates.

Keywords ShEx · Property path · Schema update

Introduction

For over years, schema plays an important role in managing various types of data, and the importance holds for RDF/graph data as well. Since user requirements for RDF data may change over time, schemas are continuously updated 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; for example, (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 attempts to use q again, and so on. In such cases, we have to update q according to the update of S, otherwise q no longer reports correct answer. However, updating q manually is a difficult and time-consuming task, since users do not fully understand the schema definition or are not aware of the details of the schema update.

To address the problem, we consider transforming queries automatically according to schema update. We focus on Shape Expression (ShEx) [3] and Property Path as schema and query language, respectively. Here, ShEx is a novel schema language for RDF whose specification is considered under the Shape Expression Community Group. Although ShEx is a relatively recent schema language, it is already applied to various areas including Wikidata and FHIR [10, 17, 20]. For RDF data, there is another novel schema language, called SHACL [15]. ShEx and SHACL share a number of similar properties [11], and the main results of this paper can be applied to SHACL as well. On the other hand, they have a few differences. SHACL schema description tends to be more complicated due to its strict definition, while ShEx has higher readability and is easy to handle, although the vocabulary has few limitations. In addition, recursion is formally supported by ShEx but not in the case of SHACL (depending on the implementation). As for query language, Property Path is a well-known path query language included in SPARQL 1.1. Property Path is defined similar to regular path query, but is extended in several aspects, e.g., backward navigation and negation.

An earlier version of this paper, Akazawa et al. [2], was presented in the 16th International Conference on Web and Information Systems and Technologies (WEBIST 2020). We have elaborated the explanation throughout this paper and also expanded the evaluation experiments.

This article is part of the topical collection "Web Information Systems and Technologies 2021" guest edited by Joaquim Filipe, Francisco Domínguez Mayo and Massimo Marchiori.

✉ Nobutaka Suzuki
nsuzuki@slis.tsukuba.ac.jp

[1] University of Tsukuba, Tsukuba, Japan

In this paper, we first introduce update operations to ShEx schema, and then propose an algorithm for transforming a given query into a new query according to schema update. We take a structural approach to transform queries. For a query q and a sequence of update operations s to ShEx schema S, our algorithm checks how s 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 behavior of q as much as possible; i.e., the answer of q' should be as close to that of the original query q as possible. To examine the effectiveness of the proposed structural approach, we conducted an experiment. The result suggests that transformed queries obtained by the proposed algorithm exhibited rather good behaviors on this aspect.

The rest of this paper is organized as follows. Section 2 gives some preliminary definitions. Section 3 describes operations to ShEx schema and proposes our algorithm. Section 4 presents the results of our evaluation experiment. Section 5 describes some related works. Section 6 concludes the study.

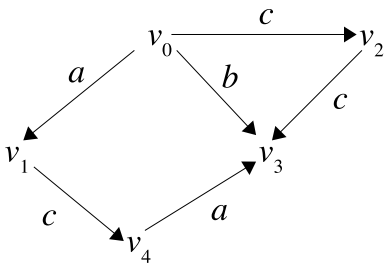

Fig. 1 A valid graph G of S

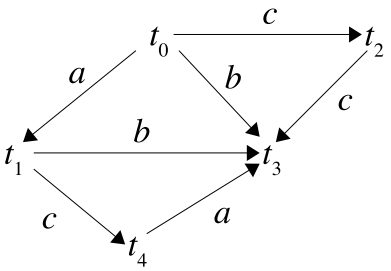

Fig. 2 Schema graph of S

Preliminaries

Graph and ShEx Schema

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 $E \subseteq V \times \Sigma \times V$ is a set of *edges*. Let $e \in E$ be an edge labeled by $l \in \Sigma$ from node $v \in V$ to node $v' \in V$. 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 is less significant. Thus ShEx uses regular bag expression (RBE) to represent content model of type [18]. RBE is similar to regular expression except that RBE uses *unordered* concatenation instead of ordered concatenation. Let Γ be a set of types. Then RBE over $\Sigma \times \Gamma$ is recursively defined as follows:

- ϵ and $a :: t \in \Sigma \times \Gamma$ are RBEs.
- If $r_1, r_2, \cdots, r_k$ are RBEs, then $r_1 | r_2 | \cdots | r_k$ is an RBE, where $|$ denotes disjunction.
- If $r_1, r_2, \cdots, r_k$ are RBEs, then $r_1 \parallel r_2 \parallel \cdots \parallel r_k$ is an RBE, where $\parallel$ denotes unordered concatenation.
- If r is an RBE, then $r^{[n,m]}$ is an RBE, where $n \leq m$. In particular, $r^? = r^{[0,1]}$, $r^* = r^{[0,\infty]}$, and $r^+ = r^{[1,\infty]}$.

For example, let $r = a :: t_1 \parallel (b :: t_2 | c :: t_3)$ be an RBE. Since $\parallel$ is unordered, r matches not only $a :: t_1\ b :: t_2$ and $a :: t_1\ c :: t_3$ but also $b :: t_2\ a :: t_1$ and $c :: t_3\ a :: t_1$.

A *ShEx schema* is denoted $S = (\Sigma, \Gamma, \delta)$, where Γ is a set of *types* and δ is a function from Γ to the set of RBEs over $\Sigma \times \Gamma$. For example, let $S = (\Sigma, \Gamma, \delta)$ be a ShEx schema, where $\Sigma = \{a, b, c\}$, $\Gamma = \{t_0, t_1, t_2, t_3, t_4\}$, and δ is a function defined as follows:

$$\delta(t_0) = a :: t_1 \parallel b :: t_3 \parallel (c :: t_2)^*,$$
$$\delta(t_1) = b :: t_3 | c :: t_4,$$
$$\delta(t_2) = c :: t_3,$$
$$\delta(t_3) = \epsilon,$$
$$\delta(t_4) = a :: t_3.$$

Let $\lambda : V \to \Gamma$ be a function that associates every node $v \in V$ with a type $\lambda(v)$. Let

$$\text{out-lab-type}^{\lambda}_G(v) = \{|a :: \lambda(v) \mid (v, a, v') \in E|\},$$

where $\{|\cdots|\}$ denotes a bag. Then $G = (V, E)$ is *valid* for S if there is a function λ such that for every node $v \in V$, $\delta(\lambda(v))$ matches out-lab-type$^{\lambda}_G(v)$. For example, consider the graph G shown in Fig. 1. Assuming that $\lambda(v_i) = t_i$ for $0 \leq i \leq 4$, it is easy to verify that $\delta(t_i)$ matches out-lab-type$^{\lambda}_G(v_i)$. Thus G is a valid graph of S.

The *schema graph* of ShEx schema $S = (\Sigma, \Gamma, \delta)$ is a graph $G_S = (V_S, E_S)$, where $V_S = \Gamma$ and $E_S = \{(t, a, t') \mid \delta(t) \mathit{containsa} :: t'\}$. For example, Fig. 2 shows the schema graph of S.

Property Path and its Traversal Area

We use Property Path as a query language. Formally, *Property Path query* (*query* for short) over Σ is defined as follows:

- ϵ and any $a \in \Sigma$ 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 $\{a_1, a_2, \cdots, a_k\}$, $!\{a_1, a_2, \cdots, a_k\}$ is a query. Here, ! denotes negation and this query matches an edge whose label is not in $\{a_1, a_2, \cdots, a_k\}$.
- For label $a \in \Sigma$, a^{-1} is a query, which matches the *inverse* of an edge labeled by a.
- For queries $q_1, q_2, \ldots, q_k$, $q_1.q_2.\ldots.q_k$ and $q_1|q_2|\ldots|q_k$ are queries. The former matches path $p = p_1.p_2.\ldots.p_k$ if q_i matches subpath p_i for every $1 \leq i \leq k$. The latter matches path p if one of $q_1, q_2, \ldots, q_k$ matches p.
- For query q, q^* is a query. This query matches a path $p = p_1.p_2.\ldots.p_k$ if q matches subpath p_i for every $1 \leq i \leq k$ $(k \geq 0)$.

In this paper, we focus on single source query traversal. For graph $G = (V, E)$, query q, and *start node* $v_s \in V$, the *answer* of q from v_s over G, denoted $Ans(G, q, v_s)$, is the set of nodes v such that G contains a path from v_s to v whose sequence of labels is matched by q. For example, if $q = a^{-1}.!\{a, b\}$ and G is the graph in Fig. 1, then $Ans(G, q, v_1) = \{v_2\}$.

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. 2, $q = b.(c^{-1})^*.(a|b)$. Then $G_S(q, t_1)$ is shown in Fig. 3. t_4 and the two edges incident to t_4 are not in the traversal area. By $Ans(G_S(q, t))$, we mean the "answer" types of $G_S(q, t)$, that is, the "answer" types obtained by traversing q from t over G_S. For example, in Fig. 3 $Ans(G_S(q, t_1)) = \{t_1, t_3\}$.

Query Transformation

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

Operation on Types and Problem Definition

To represent schema update, we introduce update operations (operations for short) to types. To update types, we need to specify explicitly the positions of labels and operators in RBEs. Thus, we introduce tree representation of type and assign an ID based on Dewey ordering to each node. For example, let

$$\delta(t_0) = a :: t_1^+ \parallel (b :: t_2 | c :: t_3) \parallel a :: t_4^*.$$

Then the tree representation of t_0 is shown in Fig. 4. The ID associated with each node is the *position* of the node.

It is desirable that the operations are "complete," that is, any ShEx schema can be updated to arbitrary ShEx schema using the operations. Thus, we define the following eight operations so that they are complete. Let t be a type of ShEx schema S.

- Updating label::type pair of type:
  - *add_lt*$(t, i, l' :: t')$: it adds label::type pair $l' :: t'$ to $\delta(t)$ at position i, where i is a Dewey order. The operation corresponds to adding an edge (t, l', t') to the schema graph of S.
  - *del_lt*(t, i): it deletes label::type pair at position i of $\delta(t)$. Let $l' :: t'$ be the pair to be deleted. Then the operation corresponds to the deletion of edge $(t, , t', l')$ from the schema graph of S.
  - *change_lt*$(t, i, l' :: t')$: it replaces label::type pair at position i of $\delta(t)$ with $l' :: t'$. Let $l'' :: t''$ be the pair to be replaced. Then the operation corresponds to the replacement of edge (t, l'', t'') with an edge (t, l', t') in the schema graph of S.
- Updating operator (|, ||, $[n, m]$) of type:

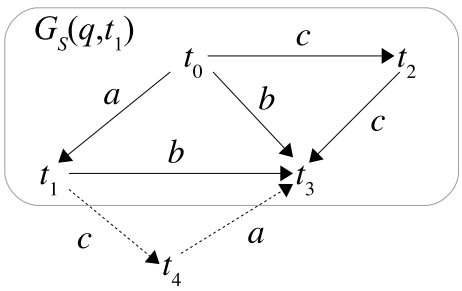

Fig. 3 Traversal area $G_S(q, t_1)$

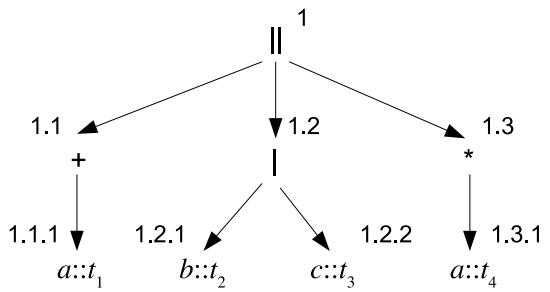

Fig. 4 Tree representation of t_0

  - *add_opr*(t, i, op): it adds operator *op* to to $\delta(t)$ at position i.
  - *del_opr*(t, i): it deletes the operation at position i from $\delta(t)$.
  - *change_opr*(t, i, op): it replaces the operation at position i of $\delta(t)$ with *op*.

- Adding/deleting type of schema:
  - *add_type*(t): it adds a new type, i.e., t, to S. Initially, $\delta(t) = \epsilon$.
  - *del_type*(t): it deletes type t from S.

An *update script* is a sequence $s = op_1 op_2 \cdots op_n$ of operations. For example, consider t_0 in Fig. 4 and let

$$s = change\_lt(t_0, 1.2, \|)\, add\_lt(t_0, 1.3, d :: t_3)$$

be an update script. By applying s to t_0, we obtain $\delta(t_0) = a :: t_1^+ \,\|\, (b :: t_2 \,\|\, c :: t_3) \,\|\, d :: t_3 \,\|\, a :: t_4^*$ (Fig. 5).

For given ShEx schema S, we can add arbitrary types to S, delete any types from S, and modify RBEs in S arbitrarily. Thus we have the following:

Theorem 1 *The eight operations are complete, that is, for any ShEx schemas S and S', there is an update script s that can update S to S'.*

When S is updated to S', the data G under S may no longer be valid for S'. In such a case, G must be updated according to the schema update so that updated version of G becomes valid for S'. How to update G depends on the user's intention which is difficult to predict precisely, but in this paper we make the following minimum assumptions:

- *add_lt*$(t, i, l' :: t')$: for each node v of type t, edges (v, l', v') are added to G so that G becomes valid for S', where v' is a new or existing node of type t'.
- *del_lt*(t, i): let $l' :: t'$ be the pair to be deleted. Then for each node v of type t, edges (v, l', v') are deleted from G so that G becomes valid for S', where v' is a node of type t'.

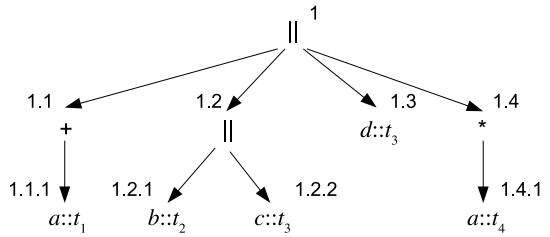

Fig. 5 Tree representation of t_0 after applying s

- *change_lt*$(t, i, l' :: t')$: let $l'' :: t''$ be the pair to be replaced.
  - If $t' \neq t''$, then this is a combination of the above two cases; deleting edges (v, l'', v'') and adding edges (v, l', v') so that G becomes valid for S', where v'' is a node of type t'' and v' is a node of type t'.
  - If $t' = t''$, then for each edge whose source is v and label is l'', the label is replaced by l'.

- *add_opr*(t, i, op), *del_opr*(t, i), *change_opr*(t, i, op): if edge(s) need to be added to G (e.g., ? is deleted or $*$ is replaced by $+$), the edges are added to G so that G becomes valid for S'. If edge(s) need to be deleted from G (e.g., $+$ or $*$ is deleted), the edges are deleted from G so that G becomes valid for S'.
- *add_type*(t): no update is made for G since G remains valid for S'.
- *del_type*(t): every node of type t is deleted from G. We assume that, prior to this update, any label-type pair $l :: t$ of $l \in \Sigma$ and t is already deleted from S.

Now we define our query transformation problem as follows:

Input:
ShEx schema $S = (\Sigma, \Gamma, \delta)$, graph $G = (V, E)$, update script s to S, query q, and start node v_s of q
Problem:
transform q to new query q' so that $Ans(G, q, v_s)$ and $Ans(G', q', v_s)$ are as "close" as possible, where G' is a graph obtained by updating G according to s under the above assumption.

We have three additional points. First, since G' is not uniquely determined since it depends on the user's intention, it is difficult to find optimum q' in general. Thus we propose a heuristic algorithm in the following subsection. Second, the criteria for the closeness above can be various depending on user's requirement, e.g., accuracy, recall, and precision. In the experiments presented in the next section, we use recall and precision to evaluate our algorithm. Third, in the aforementioned operations, *add_lt*(), *add_opr*(), *del_opr*(), *change_opr*(), *add_type*() do not affect q in that q remains "valid" against the update schema of S. On the other hand, *del_lt*(), *change_lt*(), *del_type*() may affect q, that is, q may become "invalid" under the updated schema of S in that q may lost some part of answers that were obtained under S. Thus, our algorithm below transforms q when *del_lt*(), *change_lt*(), or *del_type*() is applied to S.

Algorithm

The proposed algorithm comprises Algorithms 1 and 2. In the following, the type of start node is called *start type*. Algorithm 1

is the main part of our algorithm. For a given update script $s = op_1.op_2.\cdots.op_n$ 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 prepare copies H_S and G'_S of $G_S(q, t_s)$ and G_S, respectively (line 3). Here, H_S maintains the current traversal area of q. Then for each operation op_i of s, the algorithm modifies H_S according to op_i (lines 4–27), and converts H_S to the transformed query q' (line 28). The for loop in lines 4–27 proceeds as follows: The algorithm does nothing if op_i does not affect the traversal area H_S (lines 5–7). Otherwise, H_S (and G'_S) is modified according to op_i in lines 8–26, as follows:

- Lines 8–14 deal with $change\_lt(t, i, l' :: t')$. This operation changes label::type pair $l_i :: t_i$ of $\delta(t)$ at position i to $l' :: t'$. According to this, we replace edge (t, l_i, t_i) with (t, l', t') in H_S and G'_S. If $t_i = t'$, then the target node of the edge does not change. Otherwise, since t_i is changed to t', a path from t_s to some accepting node via t_i may be disconnected by this change. To repair this, we determine a set of simple paths P from t' to t_i in G'_S using FindPaths and add each path $p \in P$ to H_S to connect t_i and t'. Here, FindPaths is a method for finding simple paths based on depth first traversal; for given schema graph G_S and types t, t', FindPaths finds the set P of simple paths p from t to t' over G'_S allowing inverse edge traversal.
- Lines 15–19 deal with $del\_lt(t, i)$. It deletes the label::type pair $l_i :: t_i$ at position i of $\delta(t)$. According to this, we delete edge (t, l_i, t_i) from H_S and G'_S. By the edge deletion, t and t_i may be disconnected, thus we determine paths from t to t_i over G'_S using FindPaths and add the paths to H_S.
- Lines 20–26 deal with $del\_type(t)$. 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_s of nodes outgoing to t and the set T_g of nodes incoming from t and determine paths from T_s to T_g 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 $Ans(G_S(q, t_s))$ of accept states (line 2), constructing DFA M' equivalent to M (line 3), and then converting M' into query q' (line 4). The conversion process is performed using an extension of the state elimination method for DFA.

Algorithm 1 Query Transformation

Input: ShEx schema $S = (\Sigma, \Gamma, \delta)$, update script $s = op_1 op_2 \cdots op_n$ to S, query q, type $t_s \in \Gamma$
Output: query q'

1: construct the schema graph G_S of S
2: construct the traverse area $G_S(q, t_s)$ of q from t_s on G_S
3: $H_S \leftarrow G_S(q, t_s)$; $G'_S \leftarrow G_S$
4: **for** $i = 1, 2, \cdots, n$ **do**
5: **if** op_i does not affect H_S **then**
6: **continue**
7: **end if**
8: **if** $op_i = change\_lt(t, i, l' :: t')$ **then**
9: let $l_i :: t_i$ be the label::type pair at position i of $\delta(t)$
10: replace (t, l_i, t_i) with (t, l', t') in H_S and G'_S
11: **if** $t_i \neq t'$ **then**
12: $P \leftarrow$ FindPaths(G'_S, t', t_i)
13: add all $p \in P$ to H_S
14: **end if**
15: **else if** $op_i = del\_lt(t, i)$ **then**
16: let $l_i :: t_i$ be the label::type pair at position i of $\delta(t)$
17: delete (t, l_i, t_i) from H_S and G'_S
18: $P \leftarrow$ FindPaths(G'_S, t, t_i)
19: add all $p \in P$ to H_S
20: **else if** $op_i = del\_type(t)$ **then**
21: $T_s \leftarrow \{t_1 \mid (t_1, l, t)$ is an edge from t_1 to t in $G'_S\}$
22: $T_g \leftarrow \{t_2 \mid (t, l, t_2)$ is an edge from t to t_2 in $G'_S\}$
23: delete t and every edge adjacent to t from H_S and G'_S
24: $P \leftarrow \{p \mid p \in$ FindPaths$(G'_S, t_1, t_2), t_1 \in T_s, t_2 \in T_g\}$
25: add all $p \in P$ to H_S
26: **end if**
27: **end for**
28: $q' \leftarrow$ ConstructPropertyPath$(H_S, t_s, ans(G_S(q, t_s)))$
29: **return** q'

Algorithm 2 ConstructPropertyPath

Input: traversal area H_S, start type t_s, set of types Ans
Output: query q'
1: let V and E be the sets of nodes and edges of H_S, respectively
2: construct an NFA $M = (Q, \Sigma, \delta, t_s, Ans)$, where $Q = V$ and δ 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'

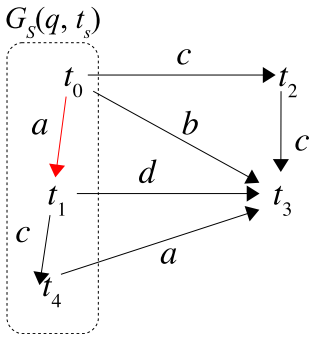

Fig. 6 Schema graph of S

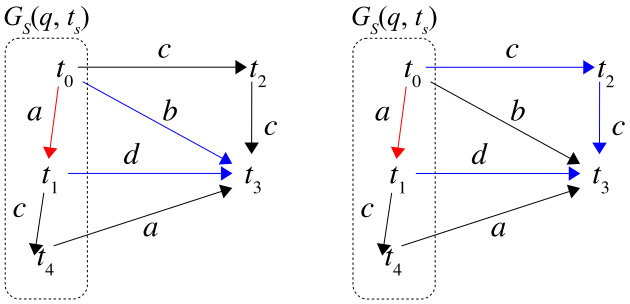

Fig. 7 Two paths obtained by FindPaths

To exhibit the behavior of the algorithm, we provide the following example: Let $S = (\Sigma, \Gamma, \delta)$ be a ShEx schema, where $\Sigma = \{a, b, c\}$, $\Gamma = \{t_0, t_1, t_2, t_3, t_4\}$, and δ is defined as follows:

$$\delta(t_0) = a :: t_1 \parallel b :: t_3 \parallel (c :: t_2)^*,$$
$$\delta(t_1) = b :: t_3 | c :: t_4,$$
$$\delta(t_2) = c :: t_3,$$
$$\delta(t_3) = \epsilon,$$
$$\delta(t_4) = a :: t_3.$$

Let $q = a.c, t_s = t_0$, and $s = del\_(t_0, 1)$. The schema graph G_S and $G_S(q, t_s)$ are shown in Fig. 6. Since $s = del\_(t_0, 1)$, the edge from t_0 to t_1 is deleted (the red edge in Fig. 6). Thus the condition in line 15 of Algorithm 1 holds, and the edge is deleted from H_S in line 17. Then FindPaths is applied in line 18 and returns two paths which are denoted by blue in Fig. 7. Thus the two paths are added to H_S in line 19. Finally, H_S is converted by using ConstructPropertyPath, and we obtain the following query:

$$q' = (b|c.c).d^{-1}.c.$$

Fig. 8 Data structure of Japanese Textbook LOD

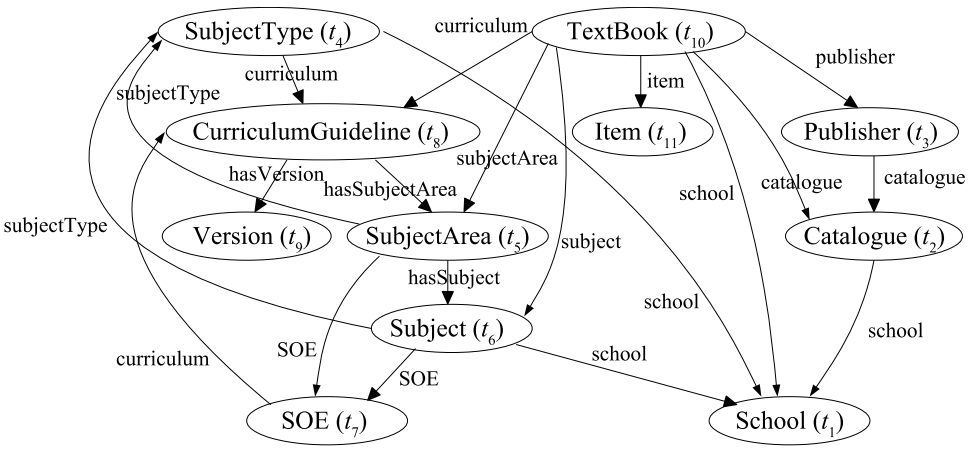

Table 1 Original query and update script

No	(a) start type, original query, and (b) update script
1	(a) t_{10}, catalogue.school (b) $del\_lt(t_{10}, 6)$ $add\_lt(t_{10}, 1, \text{subjectType} :: t_4)$
2	(a) t_2, $\text{catalogue}^{-1}.\text{publisher}^{-1}.\text{curriculum.hasSubjectArea.hasSubject}$ (b) $del\_type(t_3)$ $del\_lt(t_2, 1)$
3	(a) t_8, $\text{curriculum}^{-1}.\text{school}$ (b) $del\_lt(t_{10}, 5)$ $del\_type(t_5)$
4	(a) t_{10}, (catalogue subjectArea).school (b) $del\_lt(t_{10}, 6)$ $add\_lt(t_4, 3, \text{hasSubject} :: t_6)$
5	(a) t_5, $\text{subjectArea}^{-1}.\text{curriculum.hasSubjectArea.hasSubject.school}$ (b) $del\_type(\text{Subject})$ $change\_lt(t_8, 1, \text{version} :: t_9)$
6	(a) t_8, $\text{curriculum}^{-1}.\text{catalogue.school}$ (b) $change\_lt(t_{10}, 4, \text{curriclumguideline} :: t_8) del\_lt(t_{10}, 3.1)$
7	(a) t_{10}, curriculum.hasSubjectArea.hasSubject (b) $change\_lt(t_8, 2.1, \text{subjectArea} :: t_5)$ $change\_lt(t_5, 1, \text{subject} :: t_6)$
8	(a) t_{10}, subjectArea.hasSubject.school (b) $del\_lt(t_{10}, 17) change\_lt(t_8, 2.1, \text{hasArea} :: t_5)$ $del\_type(t_4)$
9	(a) t_8, $\text{curriculum}^{-1}.(\text{catalogue publisher.catalogue})^*.\text{school}$ (b) $del\_lt(t_{10}, 14)$ $del\_lt(t_{10}, 3.1)$ $del\_type(t_4)$
10	(a) t_2, $\text{catalogue}^{-1}.(\text{publisher}^{-1})^*.(\text{subjectArea.hasSubject}\mid \text{subject}).\text{school}$ (b) $change\_lt(t_{10}, 3.1, \text{list} :: t_2)$ $change\_lt(t_3, 1.1, \text{list} :: t_2)$ $change\_lt(t_8, 2.1\text{subjectArea} :: t_5)$ $change\_lt(t_5, 1.1, \text{subject} :: t_6)$ $del\_lt(t_{10}, 17)$ $del\_type(t_4)$

Table 2 Transformed query

No	Transformed query
1	Publisher.catalogue.school
2	$\text{catalogue}^{-1}.\text{curriculum.hasSubjectArea.hasSubject}$
3	$\text{curriculum}^{-1}.\text{publisher}^*.\text{catalogue.school}$
4	(publisher.catalogue subjectArea). School
5	$(\text{subjectArea}^{-1}.\text{curriculum.hasSubjectArea})^*.\text{subjectType}^*.\text{school}$
6	$\text{curriculumguideline}^{-1}.\text{publisher.catalogue.school}$
7	Curriculum.subjectArea.subject
8	Curriculum.hasArea.hasSubject.school
9	$\text{curriculum}^{-1}.(\text{publisher.catalogue subjectArea.hasSubject}^*\mid\text{subject}).\text{school}$
10	$\text{list}^{-1}.(\text{publisher}^{-1})^*.(\text{curriculum.subjectArea})^*.\text{subject.school}$

Experimental Results

The proposed algorithm transforms a query when ShEx schema is updated by *del_lt*(), *change_lt*(), or *del_type*(). The answer may be different from the one before and after the transformation, especially when the schema is updated by *del_lt*() or *del_type*(). The reason for this is that when a node (type) or an edge in a schema graph is deleted by *del_lt*() or *del_type*(), a path in the schema graph is disconnected and an "alternative path" is obtained by FindPaths, but the alternative path does not always match the nodes reached by the original path. Therefore, in our experiments we mainly focus on the case where *del_lt*() and *del_type*() are included in the update script. After considering the structure (connection between types) of several RDF data, we selected Japanese Textbook LOD [8, 9] as the dataset that allows us to try such various alternative paths. Then we applied the proposed algorithm to several queries to examine if the transformed queries exhibit "good" behavior in the sense that the answers of the original queries are maintained after schema update.

Table 3 Recall, precision, and F-measure

No	Recall	Precision	F-measure
1	0.88	0.99	0.93
2	0.70	0.50	0.59
3	1.00	1.00	1.00
4	0.88	0.77	0.82
5	1.00	1.00	1.00
6	0.87	0.98	0.90
7	1.00	1.00	1.00
8	0.97	0.78	0.87
9	0.87	0.78	0.83
10	0.90	0.74	0.82
Mean	0.90	0.85	0.87

Japanese Textbook LOD is RDF data compiled from a collection of textbooks that have been organized over the years by NIER Education Library and Textbook Research Center Library. Its data structure is illustrated in Fig. 8. The LOD comprises 233,001 triples of the Turtle format. The data size is 12 MB.

In this experiment, we manually created ten queries with start type and short schema updates (Table 1). We transformed each query using the proposed algorithm (and the data are also transformed according to the schema update under the assumption in Sec. 3.1). Table 2 lists the transformed queries of the original queries. Then, using each node of the start type as the start node, we executed the original and transformed queries over the original and updated data, respectively. To evaluate the outputs of the algorithm, we need a metric to evaluate how "close" the results of transformed query are to the result of its original query. To do this, we calculated recall and precision. Let G be a graph, G' be the transformed graph obtained from G, q be a query, q' be the transformed query of q, and v_s be the start node of q, q'. The recall of q' w.r.t. q from v_s over G and G' is defined as follows:

$$recall_{v_s,G,G'}(q,q') = \frac{|Ans(G,q,v_s) \cap Ans(G',q',v_s)|}{|Ans(G,q,v_s)|}.$$

Similarly, the precision of q' w.r.t. q from v_s over G and G' is defined as follows:

$$precision_{v_s,G,G'}(q,q') = \frac{|Ans(G,q,v_s) \cap Ans(G',q',v_s)|}{|Ans(G',q',v_s)|}.$$

Table 3 lists the mean values of the obtained recall, precision, and F-measure values. The F-measure of the ten queries is 0.87, which means that the results of transformed queries are close to those of their original queries. Thus, we can say that our algorithm shows a good behavior in terms of our research objective.

However, some of the transformed queries missed a few correct answers. The main reason for this is due to *del_lt*() and *del_type*(), as expected: when an edge or a node (type) on a path in a schema graph is deleted, the path is disconnected. Thus the proposed algorithm tries to find an alternative path by FindPaths method. However, the alternative path does not always return the same results as the original path, which is the reason why precision and recall are reduced. For example, consider the second query in Table 1. The query passes though the Publisher (t_3) in the opposite direction on the way from Catalogue (t_2) to TextBook (t_{10}) (i.e., "catalogue^{-1}.publisher^{-1}"), but t_3 is deleted by the update script. Thus the algorithm finds an alternative path, which is a direct inverse path labeled by "catalogue^{-1}" from Catalogue to TextBook. However, the alternative path does not necessarily return the same TextBook nodes as the original path, so the results of the transformed query are also different. Similar situations are also observed for the other queries whose F-measure is less than one, i.e., queries 1, 4, 6, and 8 to 10, although to a lesser extent.

Related Work

For many years, RDF Schema (RDFS) has been used as a schema language for RDF data. However, RDFS is essentially an ontology description language, which has a different orientation from schema description language for defining the structure of data and specifying vocabulary. Hence, ShEx was proposed as a user-friendly and high-level schema language for RDF. Validation under ShEx is naturally the most fundamental and important problem, and several methods have been proposed [5, 11, 18]. Although ShEx is a relatively recent schema language, it is already applied to various areas, e.g., Wikidata and FHIR [10, 17, 20]. Since ShEx is highly expressive, static analysis of query under ShEx schema, e.g., query containment, is also an interesting problem and some pioneering studies have been made [1, 19]. Since ShEx is a recently proposed language, some existing RDF data are not given any ShEx schema. However, methods for extracting ShEx schema from RDF data has been proposed [10, 21]. Using these methods, RDF data for which ShEx schema was not defined can benefit from ShEx schema.

Focusing on schema update of RDF data, Chirkova and Fletcher proposed a model for RDF schema (RDFS) evolution [7]. However, no query transformation was considered in that study. Gutierrez et al. proposed procedures for computing schema and instance RDFS updates, where schema and instance updates are treated separately [13]. Bonifati et al. discussed the evolution of the property graph schema based on graph rewriting operations [6]. ShEx is also used for model transformation; a transformation method from FHIR model to ShEx has been proposed, and a system based on the method has been constructed [10, 17].

In addition to RDF data, a number of studies on schema updates for XML documents have been conducted. Guerrini et al. proposed update operations that assures any updated schema contains its original schema so that documents under an original schema remains valid under its updated schema [12]. Junedi et al. studied query-update independence analysis and showed that the performance of [4] can be drastically enhanced in the use of μ-calculus [14]. Oliveira et al. proposed an algorithm for detecting possible problems that affect XQuery code according to XML Schema update [16]. Wu et al. proposed an algorithm for correcting XSLT stylesheet according to DTD update [22].

To the best of our knowledge, no studies on the transformation of Property Path query according to ShEx schema update have been conducted.

Conclusion

In this paper, we considered transforming a given query according to schema update. We used ShEx as the schema language, and Property Path as the query language. First, we introduced update operations to ShEx schema and describe the problem based on the operations. Then we proposed an algorithm for transforming a given query into a new query according to schema update. Here, the algorithm is intended to make the answer of the original query as close as possible to that of the transformed query. We conducted an experiment, and the results showed that the transformed queries exhibited good behavior in that the answers were close to that of the original queries.

However, we still have some works to do. Among them, the main things that should be done are two parts: expansion of evaluation experiments and extension of the algorithm.

- First of all, the dataset used in our experiment was limited, i.e., only Japanese Textbook LOD. Thus, we need to conduct more experiments with other various datasets. Furthermore, the queries and the update scripts used in the experiment are also limited. As shown in the experiments, the performance of "alternative path" directly affects that of transformed queries. Therefore, we need to conduct more experiments using various datasets and queries, especially affected by "alternative paths," and evaluate the performance of transformed queries.
- We also need to consider extending the algorithm. In particular, the experiments showed that queries containing "alternative paths" may miss some coronet answers. The fundamental problem with the algorithm is that query transformation is based *only* on the operations applied to ShEx schema. To achieve a better query transformation, we need to analyze how the data under the ShEx schema is updated and incorporate that into the query transformation.

Acknowledgements The authors express sincere thanks to anonymous reviewers for their invaluable and insightful comments that greatly help us to improve this paper.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abbas A, Genevès P, Roisin C, Layaïda N. SPARQL query containment with ShEx constraints. In ADBIS 2017 - 21st European Conference on Advances in Databases and Information Systems , 2017;343–356.
2. Akazawa G, Matsubara N, Suzuki N. Ransforming property path query according to shape expression schema update. In Proc. the 16th International Conference on Web Information Systems and Technologies (WEBIST 2020), 2020;292–298.
3. Baker T, Prud'hommeaux Eric. Shape expressions (ShEx) primer.http://shexspec.github.io/primer/, 2019.
4. Benedikt M, Cheney J. Destabilizers and independence of XML updates. Proc VLDB. 2010;3(1–2):906–17.
5. Boneva I, Gayo JEL, Prud'hommeaux EG. Semantics and validation of shapes schemas for RDF. In International Semantic Web Conference, 2017;104–120.
6. Bonifati A, Furniss P, Green A, Harmer R, Oshurko E, Voigt H. Schema validation and evolution for graph databases. In Proc. 38th International Conference on Conceptual Modeling (ER 2019), 2019;448–456.
7. Chirkova R, Fletcher Geroge HL. Towards well-behaved schema evolution. In Proc. 12th International Workshop on the Web and Databases (WebDB 2009), 2009.
8. Egusa Y, Takaku M. Building and publishing Japanese textbook linked open data. J Inf Sci Technol. 2018;68(7):361–7.
9. Egusa Y, Takaku M. Japanese textbook LOD. https://jp-textbook.github.io/en/about 2018.
10. Fernandez-Álvarez D, Labra-Gayo JE, Gayo-Avello D. Automatic extraction of shapes using sheXer. Knowl-Based Syst 2022;238.
11. Gayo JEL, Prud'hommeaux E, Boneva I, Kontokostas D. Validating RDF data. Synthesis Lect Semant Web: Theory Technol. 2018;7(1):1–330.
12. Guerrini G, Mesiti M, Rossi D. Impact of XML schema evolution on valid documents. In Proc. International Workshop on Web Information and Data Management (WIDM'05), 2005;39–44.
13. Gutierrez C, Hurtado C, Vaisman A. RDFS update: From theory to practice. In Proc. The Semanic Web: Research and Applications, 2011;99–107.
14. Junedi M, Genevès P, Layaïda N. XML query-update independence analysis revisited. In Proc. ACM Symposium on Document Engineering (DocEng'12), 2012; 95–98.
15. Knublauch H, Kontokostas D. Shapes constraint language (SHACL). https://www.w3.org/TR/shacl/ 2017.
16. Oliveira R, Genevès P, Layaïda N. Toward automated schema-directed code revision. In Proc. the 2012 ACM Symposium on Document Engineering (DocEng'12), 2012;103–106.
17. Solbrig HR, Prud'hommeaux E, Grieve G, McKenzie L, Mandel JC, Sharma DK, Jiang G. Modeling and validating HL7 FHIR profiles

using semantic web shape expressions (ShEx). J Biomed Inform. 2017;67:90–100.
18. Staworko S, Boneva I, Gayo JEL, Hym S, Prud'hommeaux EG, Solbrig HR. Complexity and expressiveness of ShEx for RDF. In Proc. 18th In-ternational Conference on Database Theory (ICDT 2015), 2015;17.
19. Staworko S, Wieczorek P. Containment of shape expression schemas for rdf. In Proc. the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '19, 2019;303—319.
20. Thornton K, Solbrig H, Stupp GS, Labra GJE, Mietchen D, Prud'hommeaux E, Waagmeester A. Using shape expressions (ShEx) to share RDF data models and to guide curation with rigorous validation. In Proc. the European Semantic Web Conference 2019:606–620.
21. Tsuboi Y, Suzuki N. An algorithm for extracting shape expression schemas from graphs. In Proc. the ACM Symposium on Document Engineering 2019;1–4.
22. Wu Y, Suzuki N. An algorithm for correcting XSLT rules according to dtd updates. In Proc. the 4th International Workshop on Document Changes: Modeling, Detection, Storage and Visualization, DChanges '16, 2016.