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-

Akazawa, G., Matsubara, N. and Suzuki, N.
Transforming Property Path Query According to Shape Expression Schema Update.
DOI: 10.5220/0010173002920298
In *Proceedings of the 16th International Conference on Web Information Systems and Technologies (WEBIST 2020)*, pages 292-298
ISBN: 978-989-758-478-7

posed update operations that assures any updated schema contains its original schema so that documents under an original schema remains valid under its updated schema (Guerrini et al., 2005). 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, Chirkova and Fletcher proposed a model of RDF schema evolution (Chirkova 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 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 a node $v \in V$ to a 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 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 $\Sigma \times \Gamma$ 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

$$
\begin{aligned}
\delta(t_0) &= a :: t_1 \parallel b :: t_3 \parallel (c :: t_2)^*, \\
\delta(t_1) &= b :: t_3 | c :: t_4, \\
\delta(t_2) &= c :: t_3, \\
\delta(t_3) &= \varepsilon, \\
\delta(t_4) &= a :: t_3.
\end{aligned}
$$

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 v_i is of type t_i, $\delta(t_i)$ matches the outgoing edges of v_i. Then it is easy to verify that G is a valid graph of S.

The *schema graph* of a ShEx schema $S = (\Sigma, \Gamma, \delta)$ is a graph $G_S = (V_S, E_S)$, where $V_S = \Gamma$ and $E_S = \{(t,a,t') \mid \delta(t) \text{ contains } a :: t'\}$. For example, Fig. 1(right) shows the schema graph of S.

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 a label $a \in \Sigma$, a^{-1} is a query, which matches the *inverse* of an edge labeled by a.
- For queries $q_1, q_2, \cdots, q_k$, $q_1.q_2.\cdots.q_k$ and $q_1|q_2|\cdots|q_k$ are queries. The former matches a path $p = p_1.p_2.\cdots.p_k$ if q_i matches subpath p_i for every $1 \leq i \leq k$. The latter matches a path p if one of $q_1, q_2, \cdots, q_k$ matches p.
- For a query q, q^* is a query. This query matches a path $p = p_1.p_2.\cdots.p_k$ 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 a graph G, query q, and node v_s, the *answer* of q from v_s over G is a 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 $q = a^{-1}.!\{a,b\}$, then the answer of q from v_1 over G in Fig.1 is $\{v_2\}$.

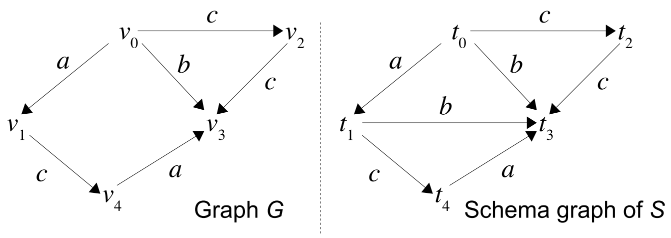

Figure 1: Valid graph G and schema graph of S.

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), $q = b.(c^{-1})^*.(a|b)$. Then $G_S(q,t_1)$ is shown in Fig. 2. By $Ans(G_S(q,t))$, 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 $Ans(G_S(q,t_1)) = \{t_1, t_3\}$.

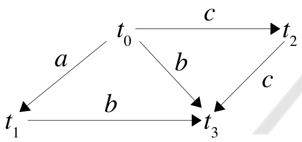

Figure 2: Traversal area $G_S(q,t_1)$.

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

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

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:
  - $add\_lt(t,i,l' :: t')$: this operation adds label::type pair $l' :: t'$ to $\delta(t)$ at position i, where

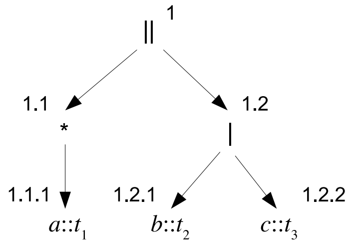

Figure 3: Tree representation of t_0.

    i is a Dewey order. This operation corresponds to adding an edge (t, l', t') to the schema graph of S.
  - $del\_lt(t,i)$: this operation deletes label::type pair at position i of $\delta(t)$. Let $l' :: t'$ be the pair to be deleted. Then this operation corresponds to deleting an edge $(t, , t', l')$ from the schema graph of S.
  - $change\_lt(t,i,l' :: t')$: this operation replaces label::type pair at position i of $\delta(t)$ with $l' :: t'$. Let $l'' :: t''$ be the pair to be replaced. Then this operation corresponds to replacing an edge (t, l'', t'') with an edge (t, l', t') in the schema graph of S.
- Changing operator ($|$, $\parallel$, $[n,m]$) of type:
  - $add\_opr(t,i,op)$: this operation adds an operator op to to $\delta(t)$ at position i.
  - $del\_opr(t,i)$: this operation deletes the operation at position i from $\delta(t)$.
  - $change\_opr(t,i,op)$: this operation replaces the operation at position i of $\delta(t)$ with op.
- Adding/deleting type of schema:
  - $add\_type(t)$: this operation adds a new type t to S. Initially, $\delta(t) = \varepsilon$.
  - $del\_type(t)$: this operation 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. 3 and let

$$s = change\_lt(t_0, 1.2, \parallel)\, add\_lt(t_0, 1.1, d :: t_3)$$

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

Let q be a query and S be a ShEx schema. In the above 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, i.e., 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 shown below transforms q when $del\_lt()$, $change\_lt()$, or $del\_type()$ 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 $s = op_1.op_2.\cdots.op_n$ 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 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 $change\_lt(t,i,l'::t')$. 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 we are done. 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 find a set of simple paths P from t' to t_i in G'_S by FindPaths and add each path $p \in P$ to H_S to connect t_i and t'.

  Here, for given types t,t', FindPaths (not shown) is a method for finding the set P 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 $del\_lt(t,i)$. This operation 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 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 $del\_type(t)$. 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 then find 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 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 $Ans(q)$ be the set of obtained answer nodes of q. The recall of q' w.r.t. q is defined as follows.

$$recall(q,q') = \frac{|Ans(q) \cap Ans(q')|}{|Ans(q)|}.$$

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

$$precision(q,q') = \frac{|Ans(q) \cap Ans(q')|}{|Ans(q')|}.$$

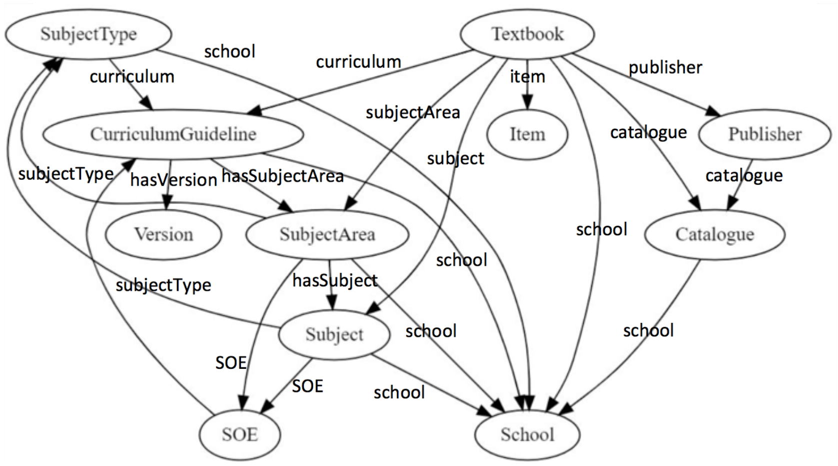

Figure 4: Data structure of Japanese Textbook LOD.

Table 1: Original query and update script.

No.	(a) original query and (b) update script
1	(a) catalogue.school
	(b) $del\_lt(Textbook, 6)$ $add\_lt(Textbook, 1, subjectType :: SubjectType)$
2	(a) catalogue^{-1}.publisher^{-1}.curriculum.hasSubjectArea.hasSubject
	(b) $del\_type(Publisher)$ $del\_lt(Catalogue, 1)$
3	(a) curriculum^{-1}.school
	(b) $del\_lt(Textbook, 5)$ $del\_type(SubjectArea)$
4	(a) (catalogue | subjectArea).school
	(b) $del\_lt(Textbook, 6)$ $add\_lt(SubjectType, 3, hasSubject :: Subject)$
5	(a) subjectArea^{-1}.curriculum.hasSubjectArea.hasSubject.school
	(b) $del\_type(Subject)$ $change\_lt(CurriculumGuideline, 1, version :: Version)$

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.

No.	Transformed Query	recall	precision	F-measure
1	publisher.catalogue.school	0.88	0.99	0.93
2	catalogue^{-1}.curriculum.hasSubjectArea.hasSubject	0.70	0.50	0.59
3	$\text{curriculum}^{-1}.\text{publisher}^{*}$.catalogue.school	1.00	1.00	1.00
4	(publisher.catalogue | subjectArea).school	0.88	0.77	0.82
5	$(\text{subjectArea}^{-1}.\text{curriculum.hasSubjectArea})^{*}.\text{subjectType}^{*}$.school	1.00	1.00	1.00
average of five queries		0.89	0.85	0.87

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'

construct the schema graph G_S of S
construct the traverse area $G_S(q,t_s)$ of q from t_s on G_S
$H_S \leftarrow G_S(q,t_s)$; $G'_S \leftarrow G_S$
for $i = 1, 2, \cdots, n$ **do**
  if op_i does not affect H_S **then**
    continue
  end if
  if $op_i = change\_lt(t, i, l' :: t')$ **then**
    let $l_i :: t_i$ be the label::type pair at position i of $\delta(t)$
    replace (t, l_i, t_i) with (t, l', t') in H_S and G'_S
    if $t_i \neq t'$ **then**
      $P \leftarrow FindPaths(G'_S, t', t_i)$
      add all $p \in P$ to H_S
    end if
  else if $op_i = del\_lt(t, i)$ **then**
    let $l_i :: t_i$ be the label::type pair at position i of $\delta(t)$
    delete (t, l_i, t_i) from H_S and G'_S
    $P \leftarrow FindPaths(G'_S, t, t_i)$
    add all $p \in P$ to H_S
  else if $op_i = del\_type(t)$ **then**
    $T_s \leftarrow \{t_1 \mid (t_1, l, t)$ is an edge from t_1 to t in $G'_S\}$
    $T_g \leftarrow \{t_2 \mid (t, l, t_2)$ is an edge from t to t_2 in $G'_S\}$
    delete t and every edge adjacent to t from H_S and G'_S
    $P \leftarrow \{p \mid p \in FindPaths(G'_S, t_1, t_2), t_1 \in T_s, t_2 \in T_g\}$
    add all $p \in P$ to H_S
  end if
end for
$q' \leftarrow$ ConstructPropertyPath$(H_S, t_s, ans(G_S(q,t_s)))$
return q'

Algorithm 2: ConstructPropertyPath.

Input: traversal area H_S, start type t_s, set of types Ans
Output: query q'

let V and E be the sets of nodes and edges of H_S, respectively
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 $(t, a, t') \in E$
construct a DFA M' equivalent to M
construct a query q' from M'
return q'

REFERENCES

Baker, T. and Prud'hommeaux, E. (2019). Shape expressions (ShEx) primer. http://shexspec.github.io/primer/.

Benedikt, M. and Cheney, J. (2010). Destabilizers and independence of XML updates. *Proc. VLDB Endow.*, 3(1-2):906–917.

Bonifati, A., Furniss, P., Green, A., Harmer, R., Oshurko, E., and Voigt, H. (2019). Schema validation and evolution for graph databases. In *Conceptual Modeling*, pages 448–456.

Chirkova, R. and Fletcher, G. H. (2009). Towards well-behaved schema evolution. In *Proc. 12th International Workshop on the Web and Databases (WebDB 2009)*.

Egusa, Y. and Takaku, M. (2018a). Building and publishing japanese textbook linked open data. *The journal of Information Science and Technology*, 68(7):361–367.

Egusa, Y. and Takaku, M. (2018b). Japanese textbook LOD. https://jp-textbook.github.io/en/about.

Guerrini, G., Mesiti, M., and Rossi, D. (2005). Impact of XML schema evolution on valid documents. In *Proc. WIDM*, pages 39–44.

Junedi, M., Genevès, P., and Layaïda, N. (2012). XML query-update independence analysis revisited. In *Proc. ACM DocEng'12*, pages 95–98.

Knublauch, H. and Kontokostas, D. (2017). Shapes constraint language (SHACL). https://www.w3.org/TR/shacl/.

Oliveira, R., Genevès, P., and Layaïda, N. (2012). Toward automated schema-directed code revision. In *Pro-*

ceedings of the 2012 ACM Symposium on Document Engineering, DocEng '12, page 103–106.

Staworko, S., Boneva, I., Gayo, J. E. L., Hym, S., Prud'hommeaux, E. G., and Solbrig, H. R. (2015). Complexity and expressiveness of ShEx for RDF. In *Proceedings of 18th In-ternational Conference on Database Theory (ICDT 2015)*, page 17p.

Thornton, K., Solbrig, H., Stupp, G. S., Labra Gayo, J. E., Mietchen, D., Prud'hommeaux, E., and Waagmeester, A. (2019). Using shape expressions (ShEx) to share RDF data models and to guide curation with rigorous validation. In Hitzler, P., Fernández, M., Janowicz, K., Zaveri, A., Gray, A. J., Lopez, V., Haller, A., and Hammar, K., editors, *In Proceedings of the European Semantic Web Conference*, pages 606–620.

Wu, Y. and Suzuki, N. (2016). An algorithm for correcting xslt rules according to dtd updates. In *Proceedings of the 4th International Workshop on Document Changes: Modeling, Detection, Storage and Visualization*, DChanges '16.