Shape Expression Schema のスキーマ進化に対する Property Path 式修正アルゴリズム

赤澤豪樹† 松原尚利†† 鈴木伸崇†††
† 筑波大学図書館情報メディア研究科 〒305-8550 茨城県つくば市春日 1-2
†† 筑波大学システム情報工学研究科 〒305-8573 茨城県つくば市天王台 1-1-1
††† 筑波大学図書館情報メディア系 〒305-8550 茨城県つくば市春日 1-2
E-mail: †s1921621@s.tsukuba.ac.jp, ††s1920841@s.tsukuba.ac.jp, †††nsuzuki@slis.tsukuba.ac.jp

あらまし 本研究では，RDF/グラフデータのスキーマ言語である Shape Expression Schema (ShEx)，および，RDF データに対する問合せ言語の一種である Property Path に着目する．Property Path とは，探索すべき経路を記述するものであり，指定した経路の始点と終点のノードのペアが解となる．正規表現と類似した定義を行うが，幾つかの拡張された表現を持つ．本稿では，ShEx スキーマが更新された場合に，それに応じて Property Path 式の修正を行う手法について考察する．まず，ShEx スキーマに対する更新操作を定義し，次に，Property Path 特有の表現に適応させた Property Path 式修正アルゴリズムを示す．
キーワード Shape Expression Schema，RDF，Property Path

1. はじめに

RDF/グラフデータが広く普及し，このようなデータに対するスキーマ言語（RDF Schema，SHACL，Shape Expression Schema 等）が提案されている．RDF/グラフデータの活用が進むにつれて，スキーマの利用もより拡大していくと考えられる．本研究では，特にスキーマの更新に着目する．時間の経過と共にデータの利用状況等も変化するため，それに応じてスキーマも更新されるのが一般的である．その場合，スキーマ下にあるデータの構造も変化し，更新前の問合せ式が使用できなくなることがある．このような場合，問合せ式の修正が必要になるが，スキーマの構造を正確に把握し，適切な修正を行うのは容易ではない．そこで本研究では，スキーマが更新された際に，その更新に応じて問合せ式を自動修正する手法について考える．

本研究では Shape Expression Schema (以下 ShEx)[1]と呼ばれる，RDF データの構造を定義するスキーマ言語，及び RDF データに対する問合せの一種である Property Path[2]に着目する．ShEx は Shape Expressions Community Group において検討中の新しいスキーマ言語であり，従来のグラフスキーマ[4]等と比べても高い表現力を有する．また，Property Path は，RDF に対する問合せ言語である SPARQL 1.1 において定義されている．Property Path は regular path query の機能に加え，いくつかの拡張した機能が備わっている．

本稿では，ShEx スキーマが更新された際，その更新に応じて Property Path 式を修正する手法を提案する．修正においては，可能な限り元の Property Path 式と同じ解を返す式が得られるようにする．本稿ではまず，スキーマ更新に必要と考えられる操作を網羅的に定義し，それぞれの操作において Property Path 式の修正が必要となるかどうかを考察する．次に，ShEx スキーマの更新に応じて Property Path 式を修正するアルゴリズムを提案する．

関連研究として，XML に関しては，スキーマ更新に応じて XPath 式の充足可能性判定を行うシステム[4]や XSLT style sheet の修正を行う手法[7]など，数多くの研究がなされている．しかし，RDF/グラフデータのスキーマ更新に応じて問合せ式を修正する手法は著者の知る限り提案されていない．

2. 諸定義

本章では，グラフ，ShEx とその型定義に用いられる Regular Bag Expression，および，Property Path に関する定義を行う．

2.1 グラフ

RDF データはラベル付き有向グラフで表現される．そのため本研究でも，ラベル付き有向グラフ（以下，グラフ）を対象とする．グラフは $G = (V, E)$と表され，ここで V はノード集合，E はエッジ集合である．例えば，図 1 のグラフ G_0 の場合，$G_0 = (V, E)$と表され，ここで

$V = \{n_0, n_1, n_2, n_3, n_4\}$

$E = \{(n_0, a, n_1), (n_0, b, n_3), (n_0, c, n_2), (n_1, b, n_3), (n_1, c, n_4), (n_2, c, n_3), (n_4, a, n_3)\}$

である．

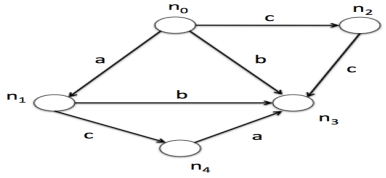

図 1 グラフ G_0

2.2 Regular Bag Expression

ShEx はグラフデータを扱うスキーマであり，兄弟ノード間の順序関係を考慮しない．よって，Regular Expression の代わりに Regular Bag Expression（以下 RBE）を用いる．RBE の大きな特徴は連結「||」において順序が無視されることである．RBE は以下のように定義される．

- ε および任意の $a \in \Sigma$ はRBEである
- $E_1, E_2, \cdots, E_k$ が RBE であるならば，$E_1|E_2|\cdots|E_k$ は RBE である．ここで，‘|’ は選言を表す．
- $E_1, E_2, \cdots, E_k$ が RBE であるならば，$E_1||E_2||\cdots||E_k$ は RBE である．ここで，‘||’ は順序を無視した連結を表す．
- E が RBE であるならば，$E^{[n,m]}$は RBE である．ここで，$[n,m]$は n 回以上 m 回以下の繰り返しを表す．

2.3 Shape Expression Schema

ShEx スキーマは $S = (\Sigma, \Gamma, \delta)$と表現される．ここで，$\Sigma$はラベルの有限集合，$\Gamma$は型の集合，$\delta$は$\Gamma$から$\Sigma \times \Gamma$上の RBE への関数であり，型の内容モデルを定義する．ShEx スキーマではそれぞれのノードに型を付与するが，全てのノードが 1 つの型を持つ場合を single-type，2 つ以上の型を取り得る場合を multi-type と呼ぶ[6]．本研究では single-type の場合を考える．

本研究では ShEx スキーマをグラフとして表現する．ShEx スキーマ $S = (\Sigma, \Gamma, \delta)$に対して，$S$のスキーマグラフを $G_s = (V_s, E_s)$と定義する．ここで，

$V_s = \Gamma$

$E_s = \{(t, l, t') \mid \delta(t)$ が $l :: t'$ を含む$\}$

である．例えば，ShEx スキーマ $S = (\{a, b, c\}, \Gamma, \delta)$ を考える．ここで，$\Gamma = \{n_0, n_1, n_2, n_3, n_4\}$かつ$\delta$は次のように定義される．

$\delta(n_0) \rightarrow a :: n_1 \,||\, b :: n_3 \,||\, c :: n_2$

$\delta(n_1) \rightarrow b :: n_3 \mid c :: n_4$

$\delta(n_2) \rightarrow c :: n_3^*$

$\delta(n_4) \rightarrow a :: n_4$

この時，Sのスキーマグラフは図 1 で表される．

2.4 Property Path

Property Pathは RDFデータに対する問合せ式の一種であり，探索経路を指定する．指定した経路の開始ノードと終端ノードの組が解となる．Σ上のProperty Path式は以下のように定義される．

i. ε および任意の $a \in \Sigma$ はProperty Path式である
ii. * はProperty Path式である
iii. $\{a_1, \ldots, a_k\}$をラベル集合とした時，$!\{a_1, \ldots, a_k\}$は Property Path式である（$a_1, \ldots, a_k$以外のラベルにマッチする）
iv. 任意の $a \in \Sigma$ に対して，a^{-1}はProperty Path式である．これは，ラベル a のエッジを逆向きに辿る
v. $r_1, r_2, \ldots, r_k$ をProperty Path式とする時，$r_1.r_2.\ldots.r_k$および$r_1|r_2|\ldots|r_k$ はPropery Path式である
vi. r を Property Path式とする時，r^* はProperty Path式である

Property Pathの大きな特徴の1つは，定義 iv のように，エッジを逆向きに進めることである．

図1の例を用いると，以下がProperty Path式とその解（始点と終点の組の集合）の例である．

- Property Path式：$a.c$ → 解：$\{(n_0, n_4)\}$
- Property Path式：$a.b.a^{-1}$ → 解：$\{(n_0, n_4)\}$
- Property Path式：$a^{-1}.!\{a,b\}$ → 解：$\{(n_1, n_2), (n_3, n_1)\}$

ただし以下では，Property Path 式に加え，探索を開始する始点の型が 1 つ与えられるものと仮定する．現時点では，本稿のアルゴリズムは上記の i ～ iv を対象としている．

3. 研究手法

本章ではまず，ShEx の更新操作について定義する．次に，Property Path 式修正手法に関して述べる．

3.1 型の木構造表現

ShEx を更新する際，どの型のどの部分を更新するのかを指定する必要がある．そこで，型を木構造で表現し，エッジに順番に番号を付与することにより，型のどの部分を修正するかを指定できるようにする．図 2 に，型 t_0 を木構造で表した例を示す．ここで，$\delta(t_0) \rightarrow a :: t_1 \,||\, b :: t_2$ とする．

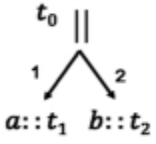

図 2 t_0 の木構造

3.2 型の更新操作

本節では ShEx スキーマの更新に必要となる型の更新操作を定義する．また，それぞれの操作が行われた際，Property Path 式の修正が必要となるか否かについて述べる．型の更新操作として以下の 8 種類を定義する．

- **「ラベル::型」の追加**

内容モデルに「ラベル::型」を追加する処理であり，

以下のように表記する．この更新操作は，スキーマグラフにおいてはエッジの追加に相当する．

add_lt (*t, i, nl*)

ここで，*t* は更新する型，*i* は δ (*t*)における追加位置，*nl* は追加する「ラベル::型」である．

- **「ラベル：:型」の削除**

内容モデルから「ラベル::型」を削除する処理であり，以下のように表記する．この更新操作は，スキーマグラフにおいてはエッジの削除に相当する．

del_lt (*t, i*)

ここで，*t* は更新する型，*i* は δ (*t*)における削除位置である．

- **「ラベル：:型」の変更**

内容モデルにおける「ラベル::型」を変更する処理であり，以下のように表記する．

change_lt (*t, i, cl*)

ここで，*t* は更新する型，*i* は δ (*t*)における変更位置，*cl* は変更後の「ラベル::型」である

- **オペレータの追加，削除，変更**

ここで言うオペレータとは，選言「|」，連接「||」，繰り返しの回数[*n,m*]である．以下のように表記する．

追加処理：add_opr (*t, i, op*)

ここで，*t* は更新する型，*i* は δ (*t*)における追加位置，*op* は追加するオペレータである．

削除処理：del_opr (*t, i*)

ここで，*t* は更新する型，*i* は δ (*t*)における削除位置である．

変更処理：change_opr (*t, i, op*)

ここで，*t* は更新する型，*i* は δ (*t*)における変更位置，*op* は変更後のオペレータである．

- **型の追加**

新たな型を追加する処理であり，以下のように表記する．

add_type(*t*)

ここで，*t* は追加する型であり，δ (*t*) → ε とする．

- **型の削除**

既存の型を削除する処理であり，以下のように表記する．

del_type(*t*)

ここで，*t* は削除する型である．

上記 8 種類の更新操作のうち，追加処理全般とオペレータへの処理全般は，探索経路に影響はないため，Property Path 式の修正に関しては考慮しなくて良い．それに対し，「ラベル::型」の削除，変更処理，型の削除処理は，更新前の探索に用いていたエッジがなくなる，または，ラベル名が変わるといった状況が生じ得るため，Property Path 式の修正が必要となる可能性がある．

3.3 Property Path 式修正アルゴリズム

本節では，Property Path 式修正アルゴリズム，またそれに付随する Procedure DFS 改を記述する．また，修正が必要となる可能性がある場合分けは，大きく以下の 3 つが考えられる．

- 「ラベル :: 型」，または型の削除
- 「ラベル :: 型」の変更において，行き先の型が変更される
- 「ラベル：：型」の変更において，エッジのラベル名が変更される

ShEx スキーマが上記のような操作で更新された場合に，それに応じて Property Path 式を修正するアルゴリズムを示す．

Input : ShEx *S*，*S* に対する更新操作 *op*，Property Path 式 *p*，経路数の上限 *limit*

Output：修正 Property Path 式（集合）

1. *S* のスキーマグラフ G_s を作成する
2. G_s 上で *p* で探索される範囲を表す部分グラフを G_p とする
3. **if** G_p 上で更新操作が行われる **then**
4. 更新操作により *p* の *index* 番目の要素が影響を受けるとする
5. **if** *op* = change_lt (*t, i, l* :: *t'*) **then**
6. δ (*t*)の位置 *i* の「ラベル::型」を *l''*:: *t''*とする
7. **if** *l''* != *l* **and** *t''* = *t'* **then**
8. *p* に出現する *l''*を *l* に置き換える
9. **else**
10. **if** G_p において *t'*が *p* の探索範囲の終端である **then**
11. **if** *index* ≠ 0 **then**
12. *new_p* ← *p*[:*index*] + *l*
13. output {*new_p*}
14. **else**
15. output {*l*}
16. **else**
17. *Paths* ← FindPaths(G_s,*t',t'',p,limit*)
18. output *Paths*
19. **else** #op = del_lt(*t, i*) or del_type(*t*)
20. **if** *op* = del_lt (*t, i*) **then**
21. δ (*t*)の位置 *i* の「ラベル::型」を *l''*:: *t''*とする
22. *start* ← *t*
23. *goal* ← *t''*
24. **else**
25. *start* ← G_p において *t* に入射する型（ノード）
26. *goal* ← G_p において *t* の

```
                                        出力先の型（ノード）
27.     Paths ← FindPaths(Gs,start,goal,p,limit)
28.     output Paths
```

1～2 行目では，S のスキーマグラフ G_S を構成し，G_s のうち p の該当部分を抽出して G_p（G_s から P 該当以外のエッジを削除）を得る．4 行目で，G_p 上で更新操作が行われたと判断した場合，まず p を“.”で区切った要素のどれに影響があるかを判別する（影響がない要素は修正を加えない）．

5～18 行目では，更新操作が change_lt である場合の手順を示しており，またその中でも単にラベル名の変更である場合（7～8 行目），行き先の型の変更の場合（9～18 行目）に分けられる．

10～13 行目では G_p 上で変更後の行き先の型が p の終点の型であり，かつ，更新影響部分の始点の型が p の始点の型と異なる場合を示している．12 行目において，修正対象箇所より前の式要素（修正は加えていない式要素）と l（変更後の行き先の型へと出力されるエッジのラベル）を結合し，13 行目でその *new_p* を出力する．14～15 行目では，G_p 上で変更後の行き先の型が p の終点の型であり，かつ，更新影響部分の始点の型が p の始点の型である場合の処理を示している．つまり 1 本のエッジを辿るだけで p の始点から終点まで向かうことができるので l をそのまま修正式として出力することとなる．16～18 行目は G_p 上で変更後の行き先の型が p の終点の型でなく，かつ，更新影響部分の始点の型が p の始点の型と異なる場合である．G_p において t'（新たな行き先の型）から t''（元々の式の行き先の型）が p で辿れなくなるので，その間の経路を FindPaths で探索して補完する．

19～28 行目は更新操作が del_lt または del_type である場合を示す．「ラベル :: 型」の削除（del_lt(*t, i*)）の場合は，t（削除されるエッジの始点）を *start*，t''（削除されるエッジの終点）を *goal* とする（21～23 行目）．型の削除（del_type(*t*)）の場合は G_p において t に入射する型（ノード）を *start*，t の出力先の型（ノード）を *goal* とする（25～26 行目）．そして FindPaths を呼び出し，*start* から *goal* までの経路集合を求めている（27 行目）．

次に，G_p において *start* から *goal* までの経路を補完する FindPaths を示す．

Input : グラフ G_s，型 *start*，型 *goal*，Property Path 式 p，p における影響要素の位置 *index*，経路数の上限 *limit*
Output : Property Path 式集合 *Paths*

```
1. path ← “”
2. Paths ← Φ
3. for every u ∈ Vs do
4.   Pre_paths(u) ← Φ
5. current ← start
6. next ← current の隣接ノード（のうちの 1 つ）
7. while start から goal に至る単純経路上のノードで
                                未訪問のものが存在する do
8.   if Pre_paths(next) = Φ or
        current が Pre_paths(next)に出現しない then
9.     if next が path の経路上に含まれない then
10.      path の末尾に current と next 間のエッジの
                                        ラベルを追加
11.      Pre_paths(next)に current と next 間の
                                エッジ（逆向きを含む）を追加
12.      current ← next
13.      next ← current の隣接ノード(のうちの 1 つ)
14.      if current = goal then
15.        Paths に path を追加
16.        path の末尾のラベルを削除
17.    else
18.      path の末尾のラベルを削除
19.      current ← 1 つ前の遷移ノード
20.      next ← current の隣接ノード(のうちの 1 つ)
21.  else
22.    path の末尾のラベルを削除
23.    current ← 1 つ前の遷移ノード
24.    next ← current の隣接ノード（のうちの 1 つ）
25. for every path ∈ Paths do
26.   if index = 0
27.     path の後に p の index より後の要素を追加
28.   elseif index = len(p) – 1
29.     path の前に p の index より前の要素を追加
30.   else
31.     path の前に p の index より前の要素，
            path の後に p の index より後の要素を追加
32. if Paths の全ての修正式候補の総経路数が
      limit より大きく，goal が p の終点でない then
33.   goal ← goal に隣接するノードで，
                    p に沿った経路を 1 つ先に進んだもの
34.   1 行目の処理に戻る
35. else
36.   return Paths
```

まず 1～2 行目で，修正 Property Path 式候補を記録するための変数 *path*，最終的な修正 Property Path 式候補を保存するための集合 *Paths* を初期化する．3～4 行目で全てのノードに集合 *Pre_paths* を付与する．これはそのノードまでたどり着く Property Path 式の経路として既に使用されたものを保存するための配列である．5 行目で，遷移中のノードを保存しておく変数 *current*

に初期値として *start* を代入する．6 行目では遷移する可能性のある隣接ノードを保存しておく変数 *next* を示している．

そして *start* から *goal* に至る単純経路上のノードで未訪問のものがなくなるまで（7 行目），8～24 行目の処理が行われる．8～16 行目では *next* のノードにおける *Pre_paths* が空である，または，*current* が *next* のノードにおける *Pre_paths* に含まれない場合，かつ，*next* のノードがその時点の *path* における式の経路に含まれない場合の処理を示している．10 行目で *path* にその隣接ノード間のエッジラベルを追加し，11 行目では *current* と *next* 間のエッジ（逆向きを含む）を *next* のノードにおける *Pre_paths* に追加する．12 行目で *current* に *next* のノードを代入し，13 行目では *next* にはまた次に遷移可能性のある隣接ノードを代入する．この時点で *current* が *goal* のノードである場合（14 行目），15 行目で *Paths* にその時点の *path* を保存する．また，16 行目で *path* の末尾のラベルを削除する．

17～20 行目では，*next* のノードがその時点の *path* における式の経路に含まれる場合や遷移先ノードが葉ノードである場合等の処理を示している． *path* の末尾のラベルを削除して一つ前のノードに戻る，という手順である．

next のノードにおける *Pre_paths* が空である，または，*current* が *next* のノードにおける *Pre_paths* に含まれない場合，かつ， *next* のノードがその時点の *path* における式の経路に含まれてしまっている場合（21～24 行目），17～20 行目と同様に 1 つ前のノードに戻る．

25～31 行目では，修正した部分の式と修正対象外の式を結合している．*p* における影響要素が先頭の場合（26～27 行目）と末尾の場合（28～29 行目），また，それ以外（影響要素が *p* の中間）の場合（30～31 行目）に分けて処理する．

32～34 行目では，適切な *path* を発見できなかった場合，*goal* を 1 つ前のノードに設定し直して探索をやり直している．

4. 評価実験

本章では，提案アルゴリズムの評価実験を示す．使用データは教科書 LOD[5]である．教科書 LOD とは，1992 年施工の学習指導要領以降の検定教科書を対象として書誌事項と教科書の関連情報を LOD 化しているプロジェクトである．

図 3 に教科書 LOD における型間のデータ構造（可読性のため，各エッジのラベルは省略している），図 4 にデータ例[5]を示す．Turtle 形式で記述されており，データサイズは 12MB，計 219,018 トリプルである．

以下に結果を示す．Property Path 式の記述においては，可読性のためラベル名の接頭辞は省略している．この実験では，提案アルゴリズムから導き出される修正 Property Path 式の一致率を算出する．ここでの一致率とは，修正 Property Path 式の解において *p*（元々の Property Path 式）の解を含む割合のことである．

① *p*：catalogue.school
始点型：Textbook，終点型：School
更新操作：del_lt(Textbook,5)
→Textbook 型から Catalog 型に出力されるエッジを削除

修正 Property Path 式	編集距離	一致率
publisher.catalogue.school	2	88.19%

② *p*：$catalogue^{-1}.publisher^{-1}.curriculum.hasSubjectArea.hasSubject$
始点型：Catalogue，終点型：Subject
更新操作：del_lt(Publisher)
→Publisher 型の削除

修正 Property Path 式	編集距離	一致率
$catalogue^{-1}.curriculum.hasSubjectArea.hasSubject$	1	69.93%

上記の結果から，提案アルゴリズムにより得られる修正 Property Path 式は，修正前の式と比較して約 70%以上検索結果が一致しており，概ね良好な結果と考えられる．一致率が 100%とならなかったのは，教科書 LOD ではスキーマ上で「*」や「?」の指定があるエッジが多く，該当するエッジがデータ上に出現せず，Property Path 式がそのエッジのラベルを含む式であった場合，解が一致しなくなるのが一因であると考えられる．

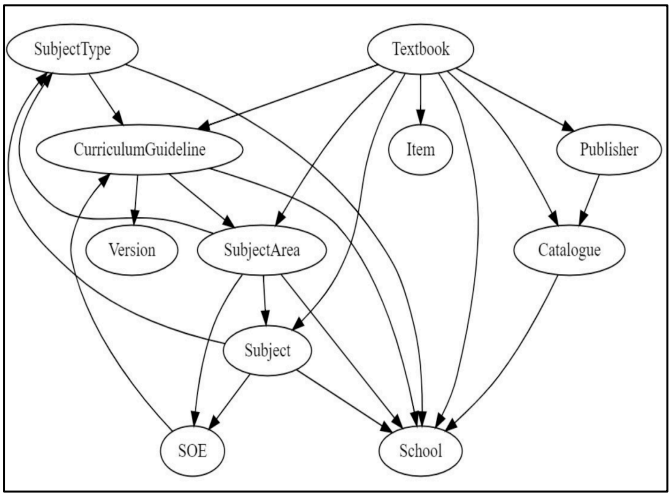

図 3 教科書 LOD における型間のデータ構造

```
@prefix bf: <http://id.loc.gov/ontologies/bibframe/>.
@prefix curriculum: <https://w3id.org/jp-textbook/curriculum/>.
@prefix dct: <http://purl.org/dc/terms/>.
@prefix nier: <http://dl.nier.go.jp/library/vocab/>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix qb: <http://purl.org/linked-data/cube#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix schema: <http://schema.org/>.
@prefix sh: <http://www.w3.org/ns/shacl#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.
@prefix textbook: <https://w3id.org/jp-textbook/>.
@prefix textbook-rc: <http://dl.nier.go.jp/library/vocab/textbook-rc/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
<https://w3id.org/jp-textbook/中学校/1992/保体/701> a textbook:Textbook;
  schema:name "新しい保健体育(中学校全)";
  schema:editor "秋山 房雄, 笠井 恵雄, 斎藤 キ能, ほか１４名";
  schema:publisher <https://w3id.org/jp-textbook/publisher/1992/東書>;
  textbook:item [
  nier:callNumber "K260.49||T8N||92/93";
  nier:recordID "EB10012620"
  ];
  textbook:catalogue <https://w3id.org/jp-textbook/catalogue/中学校/1992>, <https://w3id.org/
<https://w3id.org/jp-textbook/catalogue/中学校/1994>, <https://w3id.org/jp-textbook/catalogue
  textbook:school <https://w3id.org/jp-textbook/school/中学校>;
  textbook:subjectArea <https://w3id.org/jp-textbook/curriculum/中学校/1993/保健体育>;
  textbook:subject <https://w3id.org/jp-textbook/curriculum/中学校/1993/保健体育/保健体育>;
  textbook:grade 1, 2, 3;
  textbook:curriculum <https://w3id.org/jp-textbook/curriculum/中学校/1993>;
  textbook:authorizedYear "1992";
  textbook:usageYear "1993-1996";
  bf:extent "160";
  bf:dimensions "B5";
  textbook:textbookSymbol "保体";
  textbook:textbookNumber "701".
```

図 4 教科書 LOD のデータ例

5. むすび

本稿では, ShEx スキーマに対する更新操作を定義し, 更新に応じて Property Path 式を修正するアルゴリズムを提案した.

今後は，より多くの評価実験を行い，手法の有効性評価や改良を行っていく予定である.

参　考　文　献

[1] World Wide Web Consortium (W3C). “Shape Expression Schema (ShEx) Primer”. http://shex.io/shex-primer/. (accessed 2019-06-18).

[2] World Wide Web Consortium (W3C). “SPARQL 1.1 Property Paths”. https://www.w3.org/TR/sparql11-property-paths/. (accessed 2019-06-18).

[3] M. Fernandez and D. Suciu, “Optimizing regular path expressions using graph schemas,” Proc. ICDE, 1998.

[4] R. Oliveira , P. Genevès , and N. Layaïda. “Toward automated schema-directed code revision,” Proc. ACM DocEng, 2012.

[5] “教科書 LOD”. https://jp-textbook.github.io/, （accessed 2020-2-10）.

[6] S. Staworko, I. Boneva, J. Labra Gayo, S. Hym, E. Prud'hommeaux and H. Sorbrig., “Complexity and Expressiveness of ShEx for RDF,” Proc. ICDT, 2015, pp.195-211.

[7] Y. Wu and N. Suzuki, “An algorithm for correcting XSLT rules according to DTD updates,” Proc. DChanges, 2016.