4P-6

# スキーマ進化に伴う XPath 式修正アルゴリズム

長谷川数馬† 池田光雪†† 鈴木伸崇‡

筑波大学 情報学群 知識情報・図書館学類† 筑波大学大学院 図書館情報メディア研究科††
筑波大学 図書館情報メディア系‡

## 1 はじめに

XML データを蓄積・管理する場合，スキーマは利用者の要求や使用状況に応じて更新される(スキーマ進化)．スキーマが更新された場合，検索対象のXMLデータの構造が変化するため，XPath式の修正が必要となる場合がある．しかし，DTDやXPath式が複雑な場合など，XPath式の修正は必ずしも容易でない．本稿では，DTDに対する更新操作に基づいてXPath 式を修正するアルゴリズムを提案する．

関連研究として，文献[1]ではスキーマとして正規木文法を用いているが，本研究では DTD を用いる．一方，文献[1]ではスキーマの内容が単調に増加すると仮定しており，要素の削除を伴う更新操作を定義していないが，本研究では削除も含めて考える．

## 2 諸定義

DTD に対する更新操作として，文献[2]で定義されている要素の削除(内容モデルに出現する指定された要素を削除する)，挿入，抜き取り(内容モデルに出現する要素 $a$ を $a$ の内容モデルで置換する)に加えて，要素名の変更を用いる． DTD $D$ において次の条件が成り立つ場合，$D$ における要素 $a$ から要素 $b$ までの経路があるという．

- $D$ において $b$ が $a$ の子として定義されている
- ある要素 $c$ に対し，$a$ から $c$ までの経路が存在し，かつ $b$ が $c$ の子として定義されている．

本研究において使用する XPath 式は /*ax*[1]::*l*[1]/…/*ax*[*m*]::*l*[*m*]の形をした式であり，ここで， *ax*[*i*]は child(↓)， descendant-or-self(↓*)， parent(↑) ， ancestor-or-self(↑*) ， preceding-sibling(←+)，following-sibling(→+)のいずれか，*l*[*i*]は要素名である． $D$ と $p$ = /*ax*[1]::*l*[1]/…/*ax*[*m*]::*l*[*m*]に対して，以下の(1)と(2)が成立するならば $p$ は $D$ に対して妥当であるという．

(1) *ax*[1] = ↓かつ *l*[1] = *s*，または，*ax*[1] = ↓*かつ *l*[1]は $D$ 中に出現する．ここで，$s$ は文書要素である．

(2) $2 \leqq i \leqq m$ に対して以下が成り立つ

- *ax*[*i*] = ↓かつ *l*[*i*]が $D$ において *l*[*i*-1]の子要素として出現する
- *ax*[*i*] = ↓*かつ *l*[*i*-1]から *l*[*i*]までの経路がある
- *ax*[*i*] = ↑かつ *l*[*i*-1]が $D$ において *l*[*i*]の子要素として出現する
- *ax*[*i*] = ↑*かつ *l*[*i*]から *l*[*i*-1]までの経路がある
- *ax*[*i*] = ←+かつ $D$ において *l*[*i*]の親要素の子要素として *l*[*i*]が *l*[*i*-1]よりも先に出現する
- *ax*[*i*] = →+かつ $D$ において *l*[*i*]の親要素の子要素として *l*[*i*]が *l*[*i*-1]よりも後に出現する

## 3 XPath 式修正アルゴリズム

本アルゴリズムでは，更新前のスキーマと更新後のスキーマにおいて検索結果が一致し，かつ妥当であるように入力 XPath 式 $p$ を修正する．ただし，更新操作が要素の削除であった場合，必ずしも検索結果が同じになるように修正できるとは限らない．そこで，以下の方針でアルゴリズムを構成する．

- 更新操作が要素の削除以外の場合
  - スキーマ進化前と同じ結果を得られるように $p$ を修正する．
- 更新操作が要素の削除の場合
  - スキーマ進化前に検索される要素がスキーマ進化後もすべて存在する場合は，それらを検索できるように $p$ を修正する．
  - スキーマ進化前には $p$ で検索されていた要素がスキーマ進化後に一部もしくはすべて消失する場合，スキーマ進化後に残っている要素を検索できるように $p$ を修正する．検索する要素がすべて消失した場合，その旨を表示する．
  - $p$ の途中で検索の条件(ある要素が存在するか否かの検査など)として使用されていた要素が削除された場合，スキーマ進化前で検索される要素をすべて解に含むように $p$ を修正する．

以下にアルゴリズムの概要を示す．ただし，紙面の都合上，各更新操作に関する処理は概要のみ説明する．

---

An algorithm for transforming XPath expressions according to schema evolution
†Kazuma Hasegawa
†College of Knowledge and Library Sciences, School of Informatics, University of Tsukuba
††Kosetsu Ikeda
††Graduate School of Library, Information and Media Studies, University of Tsukuba
‡Nobutaka Suzuki
‡Faculty of Library, Information and Media Studies

入力： DTD *D*，妥当なXPath 式 *p*,
　　*D* に対する更新操作 *op*
出力：妥当な XPath 式
1. **for** *i* = 0 ... n **do**
2.　**switch** *op*
3.　**case** del_elm　　//要素の削除
4.　　*p* ← del_elm'(*D, p, op, i*)
5.　**case** ext_elm　　//要素の抜き取り
6.　　*p* ← ext_elm'(*D, p, op, i*)
7.　**case** agg_elm　　//要素の挿入
8.　　*p* ← agg_elm'(*D, p, op, i*)
9.　**case** change_elm　　//要素名の変更
10.　　*p* ← change_elm'(*D, p, op, i*)
11.　**end**
12.　**return** *p*
13. **end**

本アルゴリズムは，*p*のロケーションステップを先頭から順に見ていき，*p*に修正が必要か否かを判定する．修正が必要と判定された場合は適宜修正を行う．以下，各更新操作に対する処理の概要を説明する．

要素の削除では，まず，削除する要素または要素を削除することにより到達可能でなくなる要素を含むロケーションステップ(*dls*) を探し，*p* から*dls* を削除する．次に，*dls* が削除されたことによって到達可能でなくなるロケーションステップを探し，*p*から削除する．また，*p* において，*dls* よりも後にあるロケーションステップ(*bls*) と *dls* よりも前にあるロケーションステップ(*pls*)に対して，*dls* の削除後も*D* において*pls* から*bls* への経路がある場合，その経路を探索するための修正(ロケーションステップの追加や軸の変更等)を*pls* と*bls* の間に施す．

要素の抜き取りでは，まず抜き取られる要素を含むロケーションステップ(*els*) を探す．次に，*els* と*els* の次のロケーションステップで使用している軸の組合せから，*els* の削除や別のロケーションステップへの置換等の修正を行う．

要素の挿入では，まず挿入する要素の親要素となる要素を含むロケーションステップ(*als*) を探す．次に，*als* の直前・直後のロケーションステップで使用している軸・ノードテストと*als* で使用している軸の組合せから，*p*に修正が必要か否かを判定する．修正が必要と判定された場合，適切な軸で挿入する要素を指定したロケーションステップを*p*に挿入する．

要素名の変更では，名前を変更する要素を含むロケーションステップを探し，新しい要素名に置換する．

本アルゴリズムの時間計算量は *O*(|*p*|・|*D*|)である．ここで|*p*|は *p* のロケーションステップ数，|*D*|は *D* のサイズである．

## 4　評価実験

本アルゴリズムを Ruby で実装し，評価実験を行った． 実験に使用した DTD は MSR MEDOC[3]にある MSRREP の V2.1.1(進化前のスキーマ)と V2.2.2(進化後のスキーマ)である．実験に必要なデータ等は次のように作成した．まず，手作業でスキーマ進化の際に行われた更新操作を抽出した(対象更新操作数：39 個)．次に，スキーマ進化前の DTD から XMLSpy で XML データを生成し，このデータと抽出した更新操作に基づいてスキーマ進化後の XML データを作成した．また， XQgen を用いて 100 個の XPath 式を生成し，一部の式を↑，↑*，←+，→+を含む式に変換した．

実験手順は以下の通りである．

1. スキーマ進化前の XML データに対して 100 個の XPath 式で問合せを行う
2. 本アルゴリズムで各 XPath 式の修正を行う
3. 修正された各 XPath 式でスキーマ進化後の XML データに対して問合せを行う

スキーマ進化前とスキーマ進化後で，80 個の式で同じ結果，20 個の式で問合せ結果の数が減少した．問合せ結果が同じとなった式のうち，10 個の式が要素の削除の影響を受けたが，提案アルゴリズムにより適切に修正された．20 個の式で問合せ結果の数が減少したのは，スキーマ進化前で検索されていた要素の一部がスキーマ進化後の XML データから消失し，スキーマ進化後も存在した要素のみが検索されたためである．
以上から，スキーマ進化における要素の削除により相当数の XPath 式が影響を受けること，そして本アルゴリズムにより XPath 式が適切に修正されることが分かった．

## 5　むすび

本稿では，DTD に対する更新操作に基づいて，XPath 式を修正するアルゴリズムを提案した．今後の課題として，XPath 式の述語等への対応が挙げられる．

## 参考文献

**[1]**森本, 橋本, 石原, 藤原. “スキーマ進化に応じた XPath 問合せ変換手法における入力問合せクラスの拡張と問合せ簡単化処理の改良”. DEWS2008.
**[2]**N. Suzuki. “An Algorithm for Inferring K Optimum Transformations of XML Document from Update Script to DTD”. IEICE Trans. Inf. & Syst. 2010, vol.E93-D, no.8, pp.2198-2212.
[3]“MSR Download” MSR Home. http://www.msr-wg.de/medoc/downlo.html, (accessed 2011-11-01).