# スキーマ進化に伴う XPath 式修正アルゴリズム

筑波大学
図書館情報メディア研究科
2014 年 3 月
長谷川数馬

# 目次

# 第 1 章

# はじめに

XML[19] は Web において事実上の標準といえる文書フォーマットである．XML 文書は，多くの場合データ構造を定義するスキーマとともに格納される．一般に，スキーマは利用者の要求や使用状況の変化に伴い，要素の削除や新たな要素の追加などの更新が行われる．これを**スキーマ進化**という．スキーマが更新された場合，問合せ式はスキーマ進化により更新後のスキーマに対し妥当ではなくなる可能性がある．この場合，問合せ式を更新後のスキーマに対し妥当となるように修正する必要がある．しかしながら，スキーマの巨大化や複雑化により，この修正は容易でない．

本論文では，スキーマ進化に伴う XPath 式修正アルゴリズムを提案する．ここで，XPath[5] は，代表的な XML への問合せ言語である．スキーマ $S$，$S$ に対する更新操作を $op$，XPath 式を $p$ としたとき，本アルゴリズムは $p$ を XPath 式 $p'$ に変換する．ただし，$p'$ は $op$ を $S$ に適用した更新後のスキーマ $op(S)$ に対して検索した結果が，$S$ に対して $p$ で検索した際の結果と (可能な限り) 等価となる式である．

本アルゴリズムについて説明するために，簡単な例を示す．図 1.1(a) の DTD に対し，要素 `students` が要素 `school` の子要素として挿入されたとする (図. 1.1(b))．このスキーマ進化に従い，本アルゴリズムは次の XPath 式

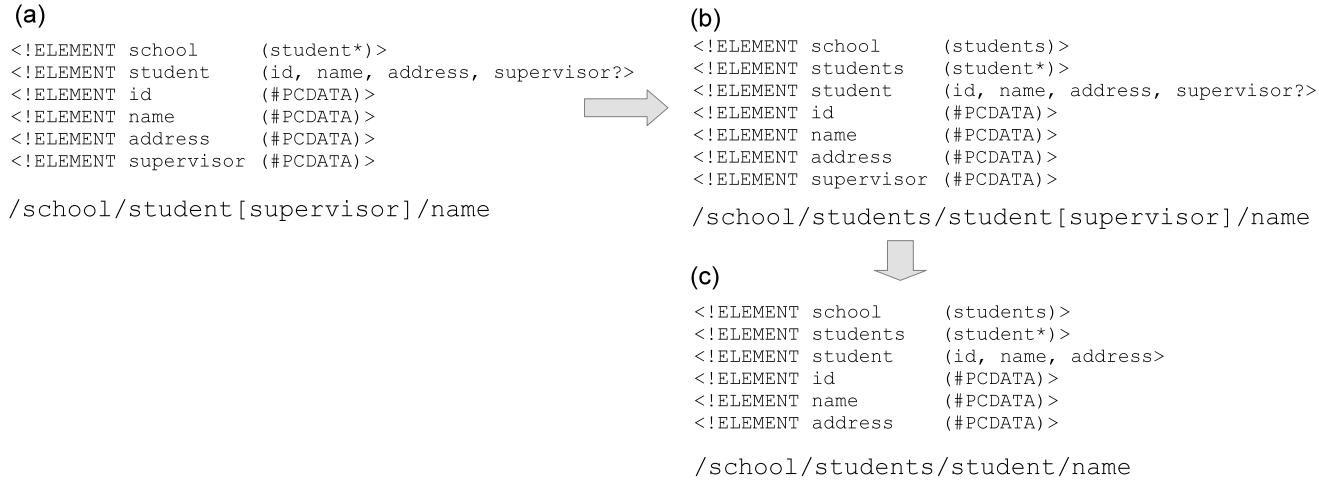


図 1.1　XPath 式変換例

```
/school/student[supervisor]/name
```

を次の式に変換する．

```
/school/students/student[supervisor]/name
```

次に，図 1.1(b) の DTD から要素 `supervisor` が削除されたとする．この場合，上記の式を等価な式に変換することは不可能である．このような場合，本アルゴリズムは代替として上記の式から `supervisor` を削除し，次の式を返す．

```
/school/students/student/name
```

上記の例に使用した DTD は小さいが，実際に使用されているスキーマはより大きくそして複雑である [7]．それゆえ，ユーザはスキーマ全体の構造を正確に理解することが困難である．したがって，本アルゴリズムはスキーマ進化に伴う XPath 式の修正に有用であると考えられる．

本アルゴリズムは，スキーマを正規木文法と等価な有限木オートマトンとして扱う．有限木オートマトンは RELAX NG の正式なモデルであるとともに，XML Schema や DTD もまた有限木オートマトンでモデル化することができる [13]．さらに，有限木オートマトンは specialized DTD と等価である [15]．以上から，本アルゴリズムは XML の主要なスキーマに対して適用可能である．また本アルゴリズムは，child 軸，descendant-or-self 軸，preceding-sibling 軸，following-sibling 軸を用い，述語を含む XPath 式を対象とする．本アルゴリズムでは，祖先への軸を対象としていないが，実際に使用されている XPath 式の多くはこれらの軸のみを用いているため，実用上の問題は小さいと考えられる [10]．

## 関連研究

本論文に最も関連が強いのは，森本らの研究 [22] である．この研究は，スキーマ進化に伴う XPath 式の変換アルゴリズムを提案している．しかし，この研究ではスキーマは単調増加すると仮定している．すなわち，スキーマから要素が削除されることはない．一方，本論文では上記のような仮定はせず，より一般的な更新操作を対象としている．実際のスキーマ進化では，スキーマから要素が消失する更新は行われているため，本アルゴリズムの方がより現実的な状況に対応していると言える．また著者の知る限り，森本らの研究を除いて，スキーマ進化に伴う XPath 式の修正を扱った研究はない．しかしながら，スキーマに対する更新を扱った研究はいくつか存在する．例えば，文献 [17] は DTD に対する「完全な」更新操作を提案している．文献 [12, 9] は，スキーマに対する更新操作および 2 つのスキーマ間の差分を抽出するアルゴリズムを提案している．文献 [8, 23, 18] は，更新前のスキーマを包含することを保証する更新操作を提案している．文献 [14] は，スキーマの更新によって起こり得る問題を列挙して分類し，それらの問題を検知するアルゴリズムを提案している．文献 [11] は，問合せ修正の独立性の分析をし，$\mu$ 論理を使用することで [6] の実行性能を大幅に改善できることを示している．

# 第 2 章

# 諸定義

本章では，有限木オートマトン，有限木オートマトンの積演算および XPath 式を定義する．

## 2.1 有限木オートマトン

XML 文書はラベル付き順序木として，スキーマは有限木オートマトンとしてそれぞれ表される．有限木オートマトンは 4 次組 $TA = (N, \Sigma, s, P)$ と定義される．ここで，

- $N$ は**状態**の有限集合
- $\Sigma$ は**要素名**の有限集合
- $s \in N$ は**初期状態**
- $P$ は $X \to a(arg)$ または $X \to Y$ の形をした**遷移規則**の有限集合， ただし， $X, Y \in N$，$a \in \Sigma$ であり，$reg$ は $N$ 上の正規表現である

遷移規則 $X \to a(arg)$ および $X \to Y$ において， $X$ は規則の**左辺**， $Y$ および $a(reg)$ を**右辺**といい， $a$ を規則の**ラベル**， $reg$ を**内容モデル**という．例えば，図 2.1 の有限木オートマトン $TA_a$ は，図 1.1(a) の DTD と等価である．$N$ における “School” は要素 “school” の型を表す．要素 “pcdata” は任意の文字列を表すものとする．$L(TA)$ は有限木オートマトン $TA$ の**言語**を表す，すなわち $TA$ に対して妥当な木の集合である．

$N = \{\text{School}, \text{Student}, \text{ID}, \text{Name}, \text{Address}, \text{Supervisor}, \text{Pcdata}\}$,
$\Sigma = \{\text{school}, \text{student}, \text{id}, \text{name}, \text{address}, \text{supervisor}, \text{pcdata}\}$,
$P = \{\text{School} \to \text{school}(\text{Student}^*), \text{Student} \to \text{student}(\text{ID Name Address Supervisor?}), \text{ID} \to \text{id}(\text{Pcdata}),$
$\text{Name} \to \text{name}(\text{Pcdata}), \text{Address} \to \text{address}(\text{Pcdata}), \text{Supervisor} \to \text{supervisor}(\text{Pcdata}),$
$\text{Pcdata} \to \text{pcdata}(\epsilon)\}$.

図 2.1 有限木オートマトン $TA_a = (N, \Sigma, \text{School}, P)$

## 2.2　有限木オートマトンの積演算

有限木オートマトンの**積演算**を定義する [21]．$TA_1 = (N_1, \Sigma_1, s_1, P_1)$ と $TA_2 = (N_2, \Sigma_2, s_2, P_2)$ を有限木オートマトンとする．一般性を失うことなく，$\Sigma_1 = \Sigma_2 = \Sigma$ と仮定する．まず，$N_1$ 上の正規表現 $reg_1$ と $N_2$ 上の正規表現 $reg_2$ に対して，それらの積 $reg_1 \oplus reg_2$ を構成する．$reg_1 \oplus reg_2$ は $N_1 \times N_2$ 上の正規表現であり，$n_1^1 n_1^2 \cdots n_1^i$ が $reg_1$ と一致しかつ $n_2^1 n_2^2 \cdots n_2^i$ が $reg_2$ と一致する場合に，$N_1 \times N_2$ 中の状態の系列 $[n_1^1, n_2^1][n_1^2, n_2^2] \cdots [n_1^i, n_2^i]$ は $reg_1 \oplus reg_2$ と正確に一致する．

このとき，$reg_1$ と $reg_2$ の積を受理する正規表現の構成方法は以下の通りである：

1. $reg_1$ から $reg_1'$($N_1 \times N_2$ 上の正規表現) を構成する．これは $N_1$ 中の各 $n$ を $[n_1, n_2^1]|[n_1, n_2^2]| \cdots |[n_1, n_2^k]$ で置き換えることで行われる．ただし，$n_2^1, n_2^2, \cdots, n_2^k$ は $N_2$ の列挙である．同様に $reg_2$ から $reg_2'$($N_1 \times N_2$ 上のもう一方の正規表現) を構成する．
2. $reg_1'$ と $reg_2'$ にそれぞれ等価な 2 つのオートマトンの $A_1$ と $A_2$ を構成する．
3. $A_1$ と $A_2$ の積オートマトンを構成する．
4. この積オートマトンから正規表現 $reg_1 \oplus reg_2$ を構成する．

このとき，$TA_1$ と $TA_2$ の積オートマトンは $TA_3 = (N_1 \times N_2, \Sigma, s_1 \times s_2, P_3)$ である．ただし，

$$\begin{aligned} P_3 = \{&[n_1, n_2] \to a(reg_1 \oplus reg_2) \mid (n_1 \to a(reg_1) \in P_1, \\ &n_2 \to a(reg_2) \in P_2) \\ &\cup \{[n_1, n_2] \to [n_1, n_2'] \mid (n_1 \in N_1, n_2 \to n_2' \in P_2)\} \\ &\cup \{[n_1, n_2] \to [n_1', n_2] \mid (n_1 \to n_1' \in P_1, n_2 \in N_2)\}\} \end{aligned}$$

木 $t$ に対して，$t \in L(TA_3)$ であるとき，かつそのときのみ $t \in L(TA_1)$ かつ $t \in L(TA_2)$ が成立する．

## 2.3　XPath 式

要素名の有限集合を $\Sigma$ とする．XPath 式 $p$ を以下のように定義する．

- $p ::= /p'$
- $p' ::= \chi :: l \mid p'/p' \mid p'[q]$ $(l \in \Sigma)$
- $\chi$ ::=child | descendant-or-self | preciding-sibling | following-sibling
- $q ::= p'$

各 $\chi \in \{\text{child}, \text{descendant-or-self}, \text{preciding-sibling}, \text{following-sibling}\}$ を**軸**，$q$ を**述語**という．

# 第 3 章

# 更新操作

本章では，有限木オートマトンに対する更新操作を定義する．

## 3.1 $reg$ 内での状態の位置

$reg$ を正規表現とする．有限木オートマトンに対する更新操作を定義するために，$reg$ 内での位置集合 $pos(reg)$ を次のように定義する．$reg$ は前置記法で表す．

- $reg = \epsilon$ または $reg = a$ $(a \in \Sigma)$ のとき，$pos(reg) = \{\lambda\}$.
- 上記以外の場合，$pos(reg) = \{\lambda\} \cup \{u \mid u = vw, 1 \leq v \leq n, w \in pos(reg_v)\}$

ここで，$reg_v$ は $reg$ における位置 $v$ の子孫から成る部分式を表す．

例えば，$reg = (A|B)^*$ の場合，前置記法により $reg =^* (|(AB))$ と表される．$pos(reg) = \{\lambda, 1, 11, 12\}$ であり，図 3.1 において，$\lambda$ は $*$, 1 は $|$, 11 は $A$, 12 は $B$ の位置を表す．

次に，一般性を失うことなく，状態 $A$ と要素名 $a$ に対して，左辺が $A$ でラベルが $a$ の遷移規則は高々一つであると仮定する．もし 2 つの遷移規則 $A \to a(reg_1)$ と $A \to a(reg_2)$ が存在した場合，これらの式は遷移規則 $A \to a(reg_1|reg_2)$ に統合することができる．

## 3.2 有限木オートマトンに対する更新操作

有限木オートマトン $TA = (N, \Sigma, s, P)$ に対する更新操作を次のように定義する．

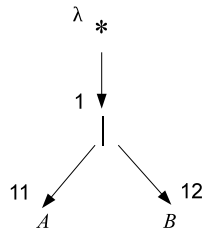


図 3.1 $(A|B)^*$ の木構造

$N = \{\text{School}, \text{Students}, \text{Student}, \text{ID}, \text{Name}, \text{Address}, \text{Supervisor}, \text{Pcdata}\}$,
$\Sigma = \{\text{school}, \text{students}, \text{student}, \text{id}, \text{name}, \text{address}, \text{supervisor}, \text{pcdata}\}$,
$P = \{\text{School} \to \text{school}(\text{Students}), \text{Students} \to \text{students}(\text{Student}^*)$,
$\text{Student} \to \text{student}(\text{ID Name Address Supervisor?}), \text{ID} \to \text{id}(\text{Pcdata})$,
$\text{Name} \to \text{name}(\text{Pcdata}), \text{Address} \to \text{address}(\text{Pcdata}), \text{Supervisor} \to \text{supervisor}(\text{Pcdata})$,
$\text{Pcdata} \to \text{pcdata}(\epsilon)\}$.

図 3.2　有限木オートマトン $TA_b = (N, \Sigma, \text{School}, P)$

$N = \{\text{School}, \text{Students}, \text{Student}, \text{ID}, \text{Name}, \text{Address}, \text{Pcdata}\}$,
$\Sigma = \{\text{school}, \text{students}, \text{student}, \text{id}, \text{name}, \text{address}, \text{pcdata}\}$,
$P = \{\text{School} \to \text{school}(\text{Students}), \text{Students} \to \text{students}(\text{Student}^*), \text{Student} \to \text{student}(\text{ID Name Address})$,
$\text{ID} \to \text{id}(\text{Pcdata}), \text{Name} \to \text{name}(\text{Pcdata}), \text{Address} \to \text{address}(\text{Pcdata}), \text{Pcdata} \to \text{pcdata}(\epsilon)\}$.

図 3.3　有限木オートマトン $TA_c = (N, \Sigma, \text{School}, P)$

- 状態の追加 $ins\_state(A, a, B, i)$：左辺が $A$ でラベルが $a$ である遷移規則の内容モデル内の位置 $i$ に新しい状態 $B$ を追加する．
- 更新操作の追加 $ins\_opr(A, a, opr, i)$：左辺が $A$ でラベルが $a$ である遷移規則の内容モデル内の位置 $i$ に演算子 $opr$ ($*$，$+$，?) を追加する．
- 状態の削除 $del\_state(A, a, i)$：左辺が $A$ でラベルが $a$ である遷移規則の内容モデル内の位置 $i$ の状態を削除する．
- 更新操作の削除 $del\_opr(A, a, i)$：左辺が $A$ でラベルが $a$ である遷移規則の内容モデル内の位置 $i$ の演算子を削除する．
- 状態の挿入 $nest\_state(A, a, B, b, i)$：左辺が $A$ でラベルが $a$ である遷移規則の内容モデル内の位置 $i$ の部分式 $E$ を状態 $B$ に置換する．さらに遷移規則 $B \to b(E)$ を $P$ に追加する．
- 状態の抜き取り $unnest\_state(A, a, b, i)$：この更新操作は $nest\_state$ の反対の操作である．すなわち，左辺が $A$ でラベルが $a$ である遷移規則の内容モデル内の位置 $i$ の状態 $B$ を，左辺が $B$ でラベルが $b$ の遷移規則の内容モデル $reg$ に置換する．
- 状態の置換 $replace\_state(A, a, B, i)$：左辺が $A$ でラベルが $a$ である遷移規則の内容モデル内の位置 $i$ の状態を $B$ に置換する．本論文では，この更新操作は $unnest\_state(A, a, i)$ と $nest\_state(A, a, B, b, i)$ のペアとして扱う．この更新操作は「要素名の変更」として使われる．

ここで更新操作の例を示す．図 2.1 の有限木オートマトン $TA_a$ に対し，
$nest\_state(\text{School}, \text{school}, \text{Students}, \text{students}, \lambda)$ を適用し，図 3.2 の有限木オートマトン $TA_b$ を得る．次に，$del\_opr(\text{Student}, \text{student}, 4)$ と $del\_state(\text{Student}, \text{student}, 4)$ を $TA_b$ に適用し，図 3.3 の $TA_c$ を得る．

# 第 4 章

# XPath 式の有限木オートマトンへの変換

本アルゴリズムでは，XPath 式を有限木オートマトンとして扱う．本章では，XPath 式の有限木オートマトンへの変換を定義する．この変換は文献 [21] を参考にしている．

XPath 式は次のように有限木オートマトンへ変換される．まず，ロケーションステップ $\chi :: l$ を有限木オートマトンに変換する．各変換後の有限木オートマトンは**入力**と**出力**状態をもつものとする．**入力**状態は，$\chi :: l$ に対する「コンテキストノード」，**出力**状態は $\chi :: l$ によって指定される「結果のノード」である．$p_1/p_2$，$p'[q]$，もしくは $/p'$ の形をした XPath 式 $p$ に対して，$p$ の構造に沿って再帰的に $p$ を有限木オートマトンに変換する．

## ロケーションステップに対する有限木オートマトンへの変換

ロケーションステップ $\chi :: l$ に対する有限木オートマトンを $(N, \Sigma, s, P)$ と定義する．以下，$A \to \sigma(\cdots)$ という記述は $\{A \to \sigma(\cdots) \mid \sigma \in \Sigma\}$ を表すものとする．

- $\chi =$ child の場合
  - 初期状態 $s$ : $B$
  - $P$ は以下の遷移規則を持つ
    - $A \to \sigma(A^*)$
    - $B \to \sigma(A^*(B|C)A^*)$
    - $B \to C$
    - $C \to \sigma(A^*DA^*)$
    - $D \to l(A^*)$
- $\chi =$ descendant-or-self の場合
  - 初期状態 $s$ : $B_1$
  - $P$ は以下の遷移規則を持つ
    - $A \to \sigma(A^*)$
    - $B_1 \to \sigma(A^*(B_1|C)A^*)$
    - $B_1 \to C$

    - $C \to \sigma(A^*(B_2|D)A^*)$
    - $C \to D$
    - $B_2 \to \sigma(A^*(B_2|D)A^*)$
    - $D \to l(A^*)$
- $\chi =$ preceding-sibling の場合
  - 初期状態 $s$ : $B_1$
  - $P$ は以下の遷移規則を持つ
    - $A \to \sigma(A^*)$
    - $B_1 \to \sigma(A^*(B_1|B_2)A^*)$
    - $B_1 \to B_2$
    - $B_2 \to \sigma(A^*DA^*CA^*)$
    - $C \to \sigma(A^*)$
    - $D \to l(A^*)$
- $\chi =$ following-sibling の場合
  - 初期状態 $s$ : $B_1$
  - $P$ は以下の遷移規則を持つ
    - $A \to \sigma(A^*)$
    - $B_1 \to \sigma(A^*(B_1|B_2)A^*)$
    - $B_1 \to B_2$
    - $B_2 \to \sigma(A^*CA^*DA^*)$
    - $C \to \sigma(A^*)$
    - $D \to l(A^*)$

有限木オートマトンの入力状態と出力状態の有限集合はそれぞれ $\{C\}$ と $\{D\}$ である．$C$ はコンテキストノードを表し，$D$ は $C$ からロケーションステップ $\chi :: l$ で探索して得られるノードである．

## XPath 式の有限木オートマトンへの変換

XPath 式 $p$ の有限木オートマトンへの変換を定義する．$p = p_1/p_2$，$p = p[q]$，$p = /p$ の場合を考える．$p$ の有限木オートマトン $TA_p$ の状態集合を $N_p$，入力状態集合を $NI_p \subset N_p$，出力状態集合を $NO_p \subset N_p$ と表す．

### $p_1/p_2$ の場合

$p_1$，$p_2$ はそれぞれ有限木オートマトン $TA_{p_1} = (N_{p_1}, \Sigma, s_{p_1}, P_{p_1})$，$TA_{p_2} = (N_{p_2}, \Sigma, s_{p_2}, P_{p_2})$ に変換済みであるとする．さらに，$TA = (N, \Sigma, s, P)$ は $TA_{p_1}$ と $TA_{p_2}$ の積オートマトンとする．このとき，$p_1/p_2$ の有限木オートマトン $TA_{p_1/p_2}$ は $TA_{p_1/p_2} = (N_{p_1/p_2}, \Sigma, s, P)$ と定義され，ここで状態集合

$N_{p_1/p_2}$ は次のように定義される．

$$N_{p_1/p_2} = \{[n_{p_1}, n_{p_2}] \mid (n_{p_1} \in NO_{p_1}, n_{p_2} \in NI_{p_2}) \vee (n_{p_1} \in (N_{p_1} - NO_{p_1}), n_{p_2} \in (N_{p_2} - NI_{p_2}))\}$$

また，$TA_{p_1/p_2}$ の入力状態集合 $NI_{p_1/p_2}$ と出力状態集合 $NO_{p_1/p_2}$ は次のように定義される．

$$NI_{p_1/p_2} = \{[n_{p_1}, n_{p_2}] \mid n_{p_1} \in NI_{p_1} n_{p_2} \in (N_{p_2} - NI_{p_2})\}$$
$$NO_{p_1/p_2} = \{[n_{p_1}, n_{p_2}] \mid n_{p_1} \in (N_{p_1} - NO_{p_1}), n_{p_2} \in NO_{p_2}\}$$

### $/p$ の場合

$TA_p = (N_p, \Sigma, s_p, P_p)$ を $p$ の有限木オートマトンとする．このとき，$/p$ の有限木オートマトン $TA_{/p}$ は $TA_{/p} = (N_{/p}, \Sigma, R, P_{/p})$ と定義され，状態集合 $N_{/p}$ は次のように定義される．

$$N_{/p} = N_p \cup \{R\}$$
$$P_{/p} = P_p \cup \{R \to root(D)\}$$

ここで，$R$ は $TA_{/p}$ の初期状態，$root$ はルートノードに対応する要素名である．$TA_{/p}$ の入力状態集合 $NI_{/p}$ と出力状態集合 $NO_{/p}$ は次のように定義される．

$$NI_{/p} = \{R\}$$
$$NO_{/p} = NO_p$$

$NO_p$ は $TA_p$ の出力状態集合である．

# 第 5 章

# XPath 式修正アルゴリズム

本章では，スキーマ進化に伴う XPath 式修正アルゴリズムについて述べる．

アルゴリズムの詳細を示す前に，まず XPath 式に関する定義を行う．XPath 式 $p$ に対して，$p$ から全ての述語を除いた XPath 式を $p$ の**主式**とする．また，$p$ が $/p_1[q]/p_2$ のように述語を持つ場合，$/p_1/q$ を $p$ の**従式**とする．例えば，$p = /a[f]/b[c/e]/d$ の場合，$/a/b/d$ が**主式**，$/a/f$，$/a/b/c/e$ が**従式**となる．

図 5.1 にアルゴリズムを示す．$op$ は有限木オートマトン $TA$ に適用される更新操作である．$p$ は入力された XPath 式，$p_0$ は**主式**，$p_1, \cdots, p_k$ は**従式**である．本アルゴリズムは，$p_0, p_1, \cdots, p_k$ を $op$ に従い $p'_0, p'_1, \cdots, p'_k$ に変換し，$p'_0, p'_1, \cdots, p'_k$ を統合し式 $p'$ を返す．より具体的には，本アルゴリズムは，まず $p$ を $p_0, p_1, \cdots, p_k$ に分割する (1 行目)．次に，更新操作に従い関数 Transform(後述) で $p_0$ を $p'_0$ に変換する．$del\_state$ により $p_0$ が更新後のスキーマで結果が空になる場合，$p'_0 = nil$ とし，$p$ に対する変換を終了する (3,4 行目)．上記以外の場合は，$p_1, \cdots, p_k$ を関数 Transform で $p'_1, \cdots, p'_k$ に変換する (6〜8 行目)．最後に，$p'_0, p'_1, \cdots, p'_k$ を統合し結果として返す．**従式** $p'_i$ が $nil$ となった場合，$p'_i$ を統合する際に除外する．例えば，$p = /a[f]/b[c/e]/d$ を考える．$p_0 = /a/b/c$，$p_1 = /a/f$，$p_2 = /a/b/c/e$ と分割される．$c$ が $unnest\_state$ により抜き取られるとする．本アルゴリズムにより，$p'_0 = /a/b/d$，$p'_1 = /a/f$，$p'_2 = /a/b/e$ となり，これらの式が統合され $p' = /a[f]/b[e]/d$ が結果となる．

次に，関数 Transform について述べる．$TA = (N, \Sigma, s, P)$ は有限木オートマトン，$op$ は $TA$ に対す

**Input** : XPath 式 $p$，有限木オートマトン $TA$，$TA$ に対する更新操作 $op$
**Output** : XPath 式もしくは $nil$

1. $p$ を**主式** $p_0$，**従式** $p_1, \cdots, p_k$ に分割する．
2. $p'_0 \leftarrow$ Transform$(p_0, TA, op)$;
3. **if** $p'_0 = nil$ **then**
4. **return** $nil$;
5. **else**
6. **for** $i \leftarrow 1$ **to** $k$ **do**
7. $p'_i \leftarrow$ Transform$(p_i, TA, op)$;
8. **end**
9. $p'_0, p'_1, \cdots, p'_k$ を $p'$ に統合する．
10. **return** $p'$;

図 5.1　Main アルゴリズム

Transform$(p, TA, op)$

**Input** : XPath 式 $p = /ls_1/\cdots/ls_m$，有限木オートマトン $TA = (N, \Sigma, s, P)$，$TA$ に対する更新操作 $op$.
**Output** : XPath 式または $nil$

1. $p$ の有限木オートマトン $TA_p = (N_p, \Sigma, s_p, P_p)$ を構築する.
2. $TA$ と $TA_p$ の積オートマトン $TA'' = (N_D \times N_p, \Sigma, s_D \times s_p, P'')$ を構築する.
3. **switch** $op$
4. **case** $ins\_state(A, a, B, i)$
5. **case** $ins\_opr(A, a, opr, i)$
6. **case** $del\_opr(A, a, i)$
7. **return** $p$;
8. **case** $del\_state(A, a, i)$
9. Delete$(A, a, i)$
10. **case** $nest\_state(A, a, B, b, i)$
11. Nest(A, a, B, b, i)
12. **case** $unnest\_state(A, a, b, i)$
13. Unnest(A, a, b,i)
14. **end**
15. **return** $p$;

図 5.2　関数 Transform

る更新操作，$p$ は述語を持たない XPath 式である．本アルゴリズムの目的は，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 $p$ を $p'$ に変換することである．ただし，$TA$ から状態及びそれに対応する要素が削除された場合，これが不可能となることがある．そこで，関数 Transform を以下の方針で構築する.

- $op$ が $ins\_state$，$ins\_opr$，$del\_opr$ の場合，$p$ は変換不要．ただし，ある要素 $l$ に対して，$p$ の途中のロケーションステップで $l$ が探索された後，続いて兄弟軸で $l$ が探索される場合，$l$ の演算子 (*または +) を削除するような更新は本アルゴリズムの対象外とする．例えば，図 1.1(a) に対する $p = /school/student/$following-sibling::$student$ を考える．このとき，$del\_opr(School, school, \lambda)$ により student に付与されている*が削除される場合，この更新は本アルゴリズムの対象外となる.
- $op$ が $nest\_state(A, a, B, b, i)$ の場合，$p$ に対してロケーションステップ child :: $b$ の追加が必要かどうか判定し，必要ならば $p$ の適切な箇所に追加する.
- $op$ が $unnest\_state(A, a, b, i)$ の場合，$p$ からノードテストが $b$ のロケーションステップの削除が必要かどうか判定し，必要ならば当該ロケーションステップを削除する.
- $op$ が $del\_state$ の場合，$p$ を次のように修正する.
  - $op$ により更新後の $TA$ から $p$ の検索対象要素が全て削除された場合，$p$ の結果は空となる．そのため，本アルゴリズムは $nil$ を返す.
  - 上記以外の場合，すなわち更新後の $TA$ に $p$ の検索対象要素が残っている場合，本アルゴリズムは $p$ を残っている要素を検索する XPath 式に変換する.

図 5.2 に関数 Transform，図 5.3〜5.5 に手続き Delete，Nest，Unnest をそれぞれ示す．図 5.3 の 6 行目，図 5.4 の 1 行目，図 5.5 の 1 行目の $state(A, a, i, P)$ は遷移規則 $r$ の内容モデル内の $i$ 番目の状態を

Delete$(A, a, i)$

1. 更新後のスキーマの有限木オートマトン $TA' = (N', \Sigma, s', P')$ を構築する.
2. $TA'$ と $TA_p$ の積オートマトン $TA'_p = (N' \times N_p, \Sigma, s' \times s_p, P'_p)$ を構築する.
3. **if** $L(TA_p) \subseteq L(TA'_p)$ **then**
4. 　**return** $p$;
5. **else if** ($ls_m$ に対応する状態) $\in N' \times N_p$ **then**
6. 　$C \leftarrow state(A, a, i, P)$　// $C$ は削除される状態
7. 　$P_p$ 内の $A$ から $C$ への遷移規則を $r$ とする.
8. 　$r$ に対応する $p$ のロケーションステップを $ls_j$ とする. $ls_j$ は削除される要素の親要素をノードテストに指定しているロケーションステップである.
9. 　$axis \leftarrow ls_{j+2}$ の軸;
10. 　$l \leftarrow ls_{j+2}$ のノードテスト;
11. 　$p \leftarrow /ls_1/\cdots/ls_j/axis :: l/ls_{j+3}/\cdots/ls_m$;
12. **else if** $p$ が**主式 then**
13. 　**return** $nil$;
14. **else**
15. 　$C \leftarrow state(A, a, i, P)$　// $C$ は削除される状態
16. 　**if** $P''$ 内に $A$ から $C$ への遷移規則 $r$ が存在する **then**
17. 　　$r$ に対応する $p$ のロケーションステップを $ls_j$ とする. $ls_j$ は削除される要素の親要素をノードテストに指定しているロケーションステップである.
18. 　　$p \leftarrow /ls_1/\cdots/ls_j$;
19. 　**end**
20. **end**

図 5.3　手続き Delete

Nest$(A, a, B, b, i)$

1. $C \leftarrow state(A, a, i, P)$　// $C$ は新しい状態の子になる状態
2. **if** $P''$ 内に $A$ から $C$ への遷移規則 $r$ が存在する.　**then**
3. 　$r$ に対応する $p$ のロケーションステップを $ls_j$ とする. $ls_j$ は新しい要素の親要素になる要素をノードテストに指定しているロケーションステップである.
4. 　**if** $ls_{j+2}$ の軸が preceding-sibling または following-sibling **then**
5. 　　**if** $ls_{j+2}$ に対応する状態が $C$ に含まれていない **then**
6. 　　　$p \leftarrow /ls_1/\cdots/ls_j/\text{child} :: b[ls_{j+1}]/ls_{j+2}/\cdots/ls_m$;
7. 　　**else if** $ls_j$ に対応する状態が $C$ に含まれていない **then**
8. 　　　$axis \leftarrow ls_{j+2}$ の軸;
9. 　　　$l \leftarrow ls_{j+2}$ のノードテスト;
10. 　　　$p \leftarrow /ls_1/\cdots/ls_j/ls_{j+1}/axis :: b/\text{child} :: l/\cdots/ls_m$;
11. 　　**else**
12. 　　　$p \leftarrow /ls_1/\cdots/ls_j/\text{child} :: b/ls_{j+1}/\cdots/ls_m$;
13. 　　**end**
14. 　**else**
15. 　　$p \leftarrow /ls_1/\cdots/ls_j/\text{child} :: b/ls_{j+1}/\cdots/ls_m$;
16. 　**end**
17. **end**

図 5.4　手続き Nest

Unnest$(A, a, b, i)$

1. $C \leftarrow state(A, a, i, P)$　// $C$ は抜き取られる状態
2. **if** $P''$ 内に $A$ から $C$ への遷移規則 $r$ が存在する． **then**
3. 　$r$ に対応する $p$ のロケーションステップを $ls_j$ とする．$ls_j$ は抜き取られる要素の親要素をノードテストに指定しているロケーションステップである．
4. 　**if** $ls_{j+2}$ の軸が preceding-sibling または following-sibling **then**
5. 　　**if** $ls_{j+2}$ に対応する状態が $C$ に含まれていない **then**
6. 　　$axis \leftarrow ls_{j+1}$ の軸;
7. 　　$l \leftarrow C$ の子の状態に対応する要素名の内 1 つ;
8. 　　$p \leftarrow /ls_1/\cdots/ls_j/axis :: l/ls_{j+2}/\cdots/ls_m$;
9. 　　**else if** $ls_{j+1}$ に対応する状態が $C$ に含まれていない **then**
10. 　　$axis \leftarrow ls_{j+2}$ の軸;
11. 　　$l_1 \cdots l_n \leftarrow C$ の子の状態に対応する要素名;
12. 　　$p \leftarrow /ls_1/\cdots/ls_{j+1}/axis :: l_1|\cdots|/ls_1/\cdots/ls_{j+1}/axis :: l_m$;
13. 　　**end**
14. 　**else**
15. 　　**if** $ls_{j+1}$ の軸が descendant-or-self **and** $ls_{j+2}$ が述語の先頭のロケーションステップ **then**
16. 　　$p \leftarrow /ls_1/\cdots/ls_{j-1}$;
17. 　　**else**
18. 　　$axis \leftarrow ls_{j+1}$ の軸;
19. 　　$l \leftarrow ls_{j+2}$ のノードテスト;
20. 　　$p \leftarrow /ls_1/\cdots/ls_j/axis :: l/ls_{j+3}/\cdots/ls_m$;
21. 　　**end**
22. 　**end**
23. **end**

図 5.5　手続き Unnest

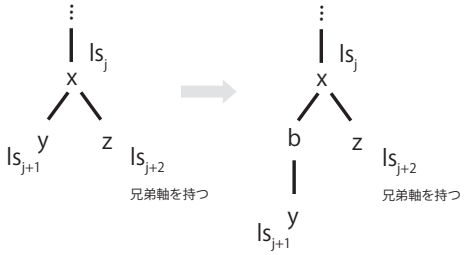


図 5.6　nest_state の例 1

意味する．ここで，$r$ は $P$ における左辺が $A$ でラベルが $a$ の遷移規則である．$op$ が $ins\_state$，$ins\_opr$ または $del\_opr$ の場合，$p$ を修正しない (4〜7 行目)．

$op$ が $del\_state$ の場合，更新後のスキーマと $p$ の積オートマトン $TA'_p$ が $TA_p$ を全て含むかを否か，すなわち $L(TA_p) \subseteq L(TA'_p)$ であるかを判定する (図 5.3 の 3 行目)．$L(TA_p) \subseteq L(TA'_p)$ である場合，$p$ の探索経路上の要素に対して更新が行われていないので，$p$ を修正しない (3,4 行目)．$L(TA_p) \subseteq L(TA'_p)$ ではなくかつ，$N' \times N_p$ に $ls_m$(XPath 式の最後尾のロケーションステップ) に対応する状態が含まれている場合，すなわち，$p$ が最終的に検索する要素が更新後のスキーマ上に存在する場合，更新操作により削除される要素をノードテストに指定しているロケーションステップ $ls_{j+1}$ を削除し，$ls_{j+2}$ の軸を $ls_{j+1}$

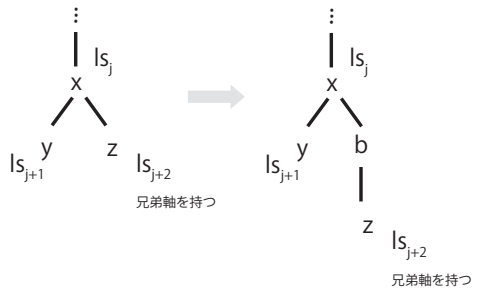


図 5.7　nest_state の例 2

の軸に置換する (5〜11 行目)．7 行目の $r$ は，$A$ を左辺に持ち内容モデルに $C$ を持つ遷移規則である．すなわち，$A$ から $C$ への遷移を含む遷移規則である．8 行目の $ls_j$ は，$r$ に対応するロケーションステップ，すなわち状態 $C$ のラベル $c$ を指定しているロケーションステップである．$L(TA_p) \subseteq L(TA'_p)$ ではなくかつ，$p$ が**主式**の場合，本アルゴリズムは $nil$ を返す (12,13 行目)．上記以外の場合，すなわち $p$ が**従式**の場合，$op$ により消失する部分に対応する $p$ のロケーションステップを削除する (14〜19 行目)．

$op$ が $nest\_state(A, a, B, b, i)$ の場合，$p$ に対して修正が必要か否かを判定する (図 5.4 の 2,3 行目)．兄弟軸に更新が影響するか否かを判定する (4〜16 行目)．5,6 行目，7〜10 行目の修正は，それぞれ図 5.6，5.7 のような更新が起きた場合に対応する．5,6 行目では，$p$ の $j+1$ 番目のロケーションステップを child :: $b[ls_{j+1}]$ に置換する．7〜10 行目では，$ls_{j+2}$ の軸 $axis$ を持つ新しいロケーションステップ $axis :: b$ を挿入し，$ls_{j+2}$ の軸を child 軸に変更する．上記以外の場合，すなわち $ls_{j+1}$，$ls_{j+2}$ 両方ともが $C$ に含まれている場合，$p$ に新しいロケーションステップ child :: $b$ を挿入する．また，兄弟軸に影響を及ぼさない場合，$p$ に新しいロケーションステップ child :: $b$ を挿入する (14,15 行目)．

$op$ が $unnest\_state(A, a, b, i)$ の場合，$p$ に対して修正が必要か否かを判定する (図 5.5 の 2,3 行目)．次に，兄弟軸に更新が影響するか否かを判定する (4〜16 行目)．抜き取られる要素を $ls_{j+1}$ で指定している場合，$p$ の $j+1$ 番目のロケーションステップを $ls_{j+1}$ の軸 $axis$，抜き取られる要素の子要素の内 1 つ $l$ をノードテストに持つ $axis :: l$ に置換する (5〜8 行目)．兄弟軸で指定している要素が抜き取られる場合，$ls_{j+2}$ の軸で抜き取られる要素の全ての子要素それぞれ指定した式の和集合を作成する (9〜13 行目)．また，兄弟軸に影響を及ぼさず，削除されるロケーションステップ $ls_{j+1}$ が述語 $q$ を持ち，かつ，$ls_{j+1}$ の軸が descendant-or-self である場合，$q$ を削除する (15,16 行目)．上記以外の場合，$ls_{j+1}$ を $p$ から削除する (18〜20 行目)．

最後に，本アルゴリズムの時間計算量について述べる．$TA = (N, \Sigma, s, P)$ をスキーマの有限木オートマトンとし，$p$ を $p_0, p_1, \cdots, p_k$ に分割できる XPath 式とする．また，$|p|$ を $p$ のロケーションステップ数，$|TA|$ を $TA$ のサイズとする．図 5.2 の 1 行目の $p$ の有限木オートマトン $TA_p$ の構築に $O(|p|)$ かかり，図 5.2 の 2 行目と図 5.3 の 2 行目の積オートマトンの構築に $O(|p| \cdot |TA|)$ かかる．そして，図 5.1 の 6〜8 行目より変換を行う XPath 式の数は $k$ である．図 5.3 の 3 行目の $L(TA_p) \subseteq L(TA'_p)$ のような有限木オートマトン同士の比較は，一般には時間のかかる演算 (EXPTIME 困難)[16] である．しかし，本研究において $TA'_p$ は $TA'$ と $TA_p$ の積オートマトンであるため，$L(TA'_p)$ のサイズは $L(TA_p)$ のサイズ以下となることが分かっている．そのため，$L(TA_p) \subseteq L(TA'_p)$ の判定は $O(|p|^2)$ で済む．以上より，本アルゴリズムの時間計算量は $O(k \cdot |p|^4 \cdot |TA|)$ となる．

# 第 6 章

# 評価実験

本章では，本アルゴリズムに対する評価について述べる．

本アルゴリズムが実際に起きたスキーマ進化に対して適切に XPath 式を修正するかを確かめる．そのため，本アルゴリズムを Ruby で実装し，2 組のスキーマを用いて評価実験を行った．実験環境は次の通りである．

OS : Windows 7 Home Premium
CPU : IntelCore2 Quad Q8400 2.66GHz
メモリ : 4.00GB
使用言語 : Ruby 1.9.3

使用したスキーマは，MSRMEDOC (version 2.1.1 and 2.2.2)[3]，NLM Journal Publishing Tag Set Tag Library (version 2.3 and 3.0)[1] である．MSRMEDOC は，製造・供給間の開発過程での情報交換のためのフォーマットである [2]．NLM は，米国医学図書館が開発した学術情報誌を XML で表現するためのフォーマットである．NLM に若干修正を加えた JATS は，米国情報標準化機構 (NISO) に採択されている [4].

まず，MSRMEDOC を用いた評価実験について述べる．MSRMEDOC version 2.1.1，version 2.2.2 をそれぞれ $D_{211}$，$D_{222}$ とする．$D_{211}$ の要素数は 185 個，$D_{222}$ の要素数は 205 個である．表 6.1 に $D_{211}$ から $D_{222}$ へスキーマ進化する際に行われた更新操作を示す．ここで，「その他」は属性の追加であり，本アルゴリズムでは対象としていない操作である．

$D_{211}$ に対する XPath 式を，XQgen[20] を用いて 100 式生成した．これらの式には述語や兄弟軸が含まれないため，手作業で 4 式に述語を付与し，5 式を兄弟軸を含む式に修正した．XPath 式の平均ロケーションステップ数は 5.96，最小ロケーションステップ数は 2，最大ロケーションステップ数は 7 である．

表 6.1　$D_{211}$・$D_{222}$ 間の更新操作

| *ins_state* | *del_state* | *nest_state* | *unnest_state* | *replace_state* | *ins_opr*/*del_opr* | その他 | 合計 |
|---|---|---|---|---|---|---|---|
| 61 | 11 | 27 | 0 | 0 | 60 | 32 | 191 |

descendant-or-self 軸は一つの式に対して高々一回，preceding-sibling 軸と following-sibling 軸は一つの式に対してどちらか一つが高々一回使用されている．100 式のうち descendant-or-self 軸は 84 式に含まれている．また，100 式のうち 7 式が $D_{222}$ で結果が空になる．

上記の 100 式を本アルゴリズムで変換する．93 式は $D_{211}$ の検索結果と同じものが $D_{222}$ でも得られた．残る 7 式の XPath 式では，$D_{211}$ 上で検索されていた要素が $D_{222}$ で全て消失するため，本アルゴリズムは $nil$ を返した．この 7 式を表 6.2 に示す．これらの式は，$p$ 内のノードテストで指定している要素 LIST，NOTE，FIGURE，FORMULA(REMARK の子要素)，PRIVATE-CODES(COMPANY-DOC-INFO の子要素)，REMARK(COMPANY-REVISION-INFO の子要素)，NCOI-1(SDG の子要素) が更新により消失したため，$D_{222}$ で結果が空となる．述語を持つ式 $p_1, p_2, p_3, p_4$ の変換は次の通りである (表 6.3).

- COMPANY-REVISION-INFO の子要素 REMARK が削除されたため，$p_1$ の述語を削除する．
- REMARK が削除されたため，$p_2$ の述語内の最後のロケーションステップを削除する．
- $D_{211}$ で検索していた要素 REMARK が削除されたため， $p_3$ は $D_{222}$ で結果が空となる．そのため，本アルゴリズムは $nil$ を返す．
- P と FT，P と STD の間に要素 L-1 が追加されるため，$p_4$ に新しいロケーションステップ L-1 を追加する．

同様に，兄弟軸を含む式 $p_5, p_6, p_7, p_8, p_9$ の変換は次の通りである (表 6.4).

- FIGURE と兄弟軸で指定している GRAPHIC の間に要素 L-GRAPHIC が追加されるため，$p_5$ において兄弟軸で新しいロケーションステップ L-GRAPHIC を指定し，child 軸で GRAPHIC を指定する．
- FIGURE と GRAPHIC の間に要素 L-GRAPHIC が追加されるため，$p_6$ において GRAPHIC に対応するロケーションステップを L-GRAPHIC[GRAPHIC] に置換する．
- 兄弟軸で指定している SDG の子要素 NCOI-1 が削除されたため，$p_7$ は $D_{222}$ で結果が空となる．そのため，本アルゴリズムは $nil$ を返す．
- XREF が削除され，兄弟軸を使用できなくなるため，$p_8$ の SDG-CAPTION を指定する軸を child 軸に置換する．
- P と STD，P と TT の間に要素 L-1 が追加されるため，$p_9$ に新しいロケーションステップ L-1 を追加する．

要素の削除により，XPath 式上に出現する要素が消失したにもかかわらず本アルゴリズムは適切に XPath 式を修正した．

上記の XPath 式 100 式と 191 個の更新操作での，総実行時間は 2.840 秒，1 式当りの実行時間は 0.027 秒である．

次に，NLM Journal Publishing Tag Set Tag Library を用いた評価実験について述べる．The NLM Journal Publishing Tag Set Tag Library version 2.3，version 3.0 をそれぞれ $D_{23}$，$D_{30}$ とする．$D_{23}$ の要素数は 211 個，$D_{30}$ の要素数は 233 個である．表 6.5 に $D_{23}$ から $D_{30}$ へスキーマ進化する際に行われた更新操作を示す．

$D_{23}$ に対する XPath 式を，XQgen を用いて 100 式生成した．そのうち手作業で，5 式に述語を付与し，9 式を兄弟軸を含む式に修正した．XPath 式の平均ロケーションステップ数は 13.55，最小ロケーションステップ数は 3，最大ロケーションステップ数は 16 である．

descendant-or-self 軸は一つの式に対して高々一回，preceding-sibling 軸と following-sibling 軸は一つの式に対してどちらか一つが高々一回使用されている．100 式のうち descendant-or-self 軸は 98 式に含まれている．また，100 式のうち 11 式が $D_{30}$ で結果が空になる．

上記の 100 式を本アルゴリズムで変換する．89 式は $D_{23}$ の検索結果と同じものが $D_{30}$ でも得られた．残る 11 式の XPath 式では，$D_{23}$ 上で検索されていた要素が $D_{30}$ で全て消失するため，本アルゴリズムは $nil$ を返した．この 11 式を表 6.6 に示す．これらの式は，$p$ 内のノードテストで指定している要素 citation, contract-num, contract-sponsor, copyright-year が更新により消失したため，$D_{30}$ で結果が空となる．述語を持つ式 $p_{10}, p_{11}, p_{12}, p_{13}, p_{14}$ の変換は次の通りである (表 6.7).

- time-stamp が削除されたため，$p_{10}$ の述語を削除する．
- target が削除されたため，$p_{11}$ の述語内の target 以降の部分式を削除する．
- contract-num が削除されたため，$p_{12}$ は $D_{30}$ で結果が空となる．そのため，本アルゴリズムは $nil$ を返す．
- chem-struct-wrapper が chem-struct-wrap に置換されたため，$p_{13}$ の述語内の chem-struct-wrapper を chem-struct-wrap に置換する．
- custom-meta-wrap が custom-meta-group に置換されたため，$p_{14}$ の述語内の custom-meta-wrap を custom-meta-group に置換する．

同様に，兄弟軸を含む式 $p_{15}, p_{16}, p_{17}, p_{18}, p_{19}, p_{20}, p_{21}, p_{22}, p_{23}$ の変換は次の通りである (表 6.8).

- copyright-year が削除され， 兄弟軸を使用できなくなるため，$p_{15}$ の counts を指定する軸を child 軸に置換する．
- 兄弟軸で指定している article-meta の子要素 copyright-year が削除されたため，$p_{16}$ は $D_{30}$ で結果が空となる．そのため，本アルゴリズムは $nil$ を返す．
- journal-meta と兄弟軸で指定している journal-title の間に要素 journal-title-group が追加されるため，$p_{17}$ において兄弟軸で新しいロケーションステップ journal-title-group を指定し，child 軸で journal-title を指定する．
- journal-meta と journal-title の間に要素 journal-title-group が追加されるため，$p_{18}$ において journal-title に対応するロケーションステップを journal-title-group[journal-title] に置換する．
- chem-struct-wrapper が chem-struct-wrap に置換されたため，$p_{19}$，$p_{20}$ の chem-struct-wrapper を chem-struct-wrap に置換する．
- article-categories の子要素 subj-group が抜き取られるため，$p_{21}$ を $D_{23}$ で検索されていた要素，すなわち subj-group の子要素を全て兄弟軸で指定した式の和集合の式に修正する．
- article-categories の子要素 subj-group が抜き取られるため，$p_{22}$ において subj-group に対応するロケーションステップを削除し，subj-group を指定していた軸で subject を指定する．
- article-categories の子要素 subj-group が抜き取られるため，$p_{23}$ において subj-group に対応する

ロケーションステップを subj-group を指定していた軸で subject を指定したロケーションステップに置換する．

要素の削除により，XPath 式上に出現する要素が消失したにもかかわらず本アルゴリズムは適切に XPath 式を修正することができている．これらの結果より，本アルゴリズムは実際に起きたスキーマ進化に適用可能であると考えられる．上記の XPath 式 100 式と 733 個の更新操作での，総実行時間は 7.036 秒，1 式当りの実行時間は 0.067 秒である．

表 6.2　$D_{222}$ に対して結果が空になる XPath 式

| |
|---|
| /MSRREP/MATCHING-DCIS/MATCHING-DCI/REMARK/*LIST*/ITEM |
| /MSRREP/MATCHING-DCIS/MATCHING-DCI/REMARK/*NOTE* |
| /MSRREP/MATCHING-DCIS/MATCHING-DCI/REMARK/*FIGURE*/DESC |
| /MSRREP/MATCHING-DCIS/MATCHING-DCI/REMARK/*FORMULA*/FORMULA-CAPTION/LONG-NAME |
| //ADMIN-DATA/COMPANY-DOC-INFOS/COMPANY-DOC-INFO/*PRIVATE-CODES*/PRIVATE-CODE |
| //DOC-REVISION/COMPANY-REVISION-INFOS[COMPANY-DOC-INFO/PRIVATE-CODES/PRIVATE-CODE]/DOC-REVISIONS/DOC-REVISION/COMPANY-REVISION-INFOS/COMPANY-REVISION-INFO/*REMARK* |
| //SPECIAL-DATA/SDG/SD/following-sibling::*NCOI-1*/P |

表 6.3　$D_{211}$ と $D_{222}$ に対する述語を含む XPath 式

| | |
|---|---|
| XPath 式 $p_1$: | //DOC-REVISION/COMPANY-REVISION-INFOS/COMPANY-REVISION-INFO[*REMARK*]/COMPANY-REF |
| 変換結果: | //DOC-REVISION/COMPANY-REVISION-INFOS/COMPANY-REVISION-INFO/COMPANY-REF |
| XPath 式 $p_2$: | //DOC-REVISION[COMPANY-REVISION-INFOS/COMPANY-REVISION-INFO/*REMARK*]/MODIFICATIONS/MODEFICATION |
| 変換結果: | //DOC-REVISION[COMPANY-REVISION-INFOS/COMPANY-REVISION-INFO]/MODIFICATIONS/MODEFICATION |
| XPath 式 $p_3$: | //DOC-REVISION[COMPANY-REVISION-INFOS/COMPANY-DOC-INFO/PRIVATE-CODES/PRIVATE-CODE]/DOC-REVISIONS/DOC-REVISION/COMPANY-REVISION-INFOS/COMPANY-REVISION-INFO/*REMARK* |
| 変換結果: | *nil* |
| XPath 式 $p_4$: | //P[FT]/STD |
| 変換結果: | //P[*L-1*/FT]/*L-1*/STD |

表 6.4　$D_{211}$ と $D_{222}$ に対する兄弟軸を含む XPath 式

| | |
|---|---|
| XPath 式 $p_5$: | //FIGURE/FIGURE-CAPTION/following-sibling::GRAPHIC |
| 変換結果: | //FIGURE/FIGURE-CAPTION/following-sibling::*L-GRAPHIC*/GRAPHIC |
| XPath 式 $p_6$: | //FIGURE/GRAPHIC/following-sibling::MAP |
| 変換結果: | //FIGURE/*L-GRAPHIC*[GRAPHIC]/following-sibling::MAP |
| XPath 式 $p_7$: | //SPECIAL-DATA/SDG/SD/following-sibling::*NCOI-1*/P |
| 変換結果: | *nil* |
| XPath 式 $p_8$: | //SPECIAL-DATA/SDG/*XREF*/following-sibling::SDG-CAPTION |
| 変換結果: | //SPECIAL-DATA/SDG/SDG-CAPTION |
| XPath 式 $p_9$: | //MATCHING-DCIS/MATCHING-DCI/REMARK/P/STD/following-sibling::TT |
| 変換結果: | //MATCHING-DCIS/MATCHING-DCI/REMARK/P/*L-1*/STD/following-sibling::TT |

表 6.5　$D_{23}$・$D_{30}$ 間の更新操作

| *ins_state* | *del_state* | *nest_state* | *unnest_state* | *replace_state* | *ins_opr*/*del_opr* | その他 | 合計 |
|---|---|---|---|---|---|---|---|
| 504 | 82 | 3 | 0 | 24 | 0 | 120 | 733 |

表 6.6　$D_{30}$ に対して結果が空になる XPath 式

| |
|---|
| //table-wrap-group/table-wrap/speech/p/*citation*/conf-name |
| //table-wrap-group/table-wrap/speech/p/*citation*/conf-loc |
| //table-wrap-group/table-wrap/speech/p/*citation*/comment |
| //table-wrap-group/table-wrap/speech/p/*citation*/article-title |
| //table-wrap-group/table-wrap/speech/p/*citation*/annotation |
| //p/*contract-num*/named-content/ack/boxed-text/sec |
| //p/*contract-sponsor*/named-content/verse-group/verse-group/verse-group |
| //statement/p/*contract-sponsor*/named-content/verse-group/verse-group |
| //statement/p/*contract-sponsor*/named-content/verse-group/verse-line |
| //list-item/p/*contract-num*/named-content/ack[label]/p |
| //article-meta/article-id/following-sibling::*copyright-year* |

表 6.7　$D_{23}$ と $D_{30}$ に対する述語を含む XPath 式

| | |
|---|---|
| XPath 式 $p_{10}$: | //inline-supplementary-material/overline/related-article[*time-stamp*]/sans-serif |
| 変換結果: | //inline-supplementary-material/overline/related-article/sans-serif |
| XPath 式 $p_{11}$: | //sub/abbrev/def/p/array/label/sup/named-content/boxed-text/sec-meta/contrib-group/contrib[collab/*target*/sup]/name |
| 変換結果: | //sub/abbrev/def/p/array/label/sup/named-content/boxed-text/sec-meta/contrib-group/contrib[collab]/name |
| XPath 式 $p_{12}$: | //list-item/p/*contract-num*/named-content/ack[label]/p |
| 変換結果: | *nil* |
| XPath 式 $p_{13}$: | //named-content[*chem-struct-wrapper*/label]/media/permissions/license/p/fig |
| 変換結果: | //named-content[*chem-struct-wrap*/label]/media/permissions/license/p/fig |
| XPath 式 $p_{14}$: | /article/front/journal-meta/*custom-meta-wrap*[custom-meta/meta-name]/custom-meta/meta-value |
| 変換結果: | /article/front/journal-meta/*custom-meta-group*[custom-meta/meta-name]/custom-meta/meta-value |

表 6.8　$D_{23}$ と $D_{30}$ に対する兄弟軸を含む XPath 式

| | |
|---|---|
| XPath 式 $p_{15}$: | //article-meta/*copyright-year*/following-sibling::counts |
| 変換結果: | //article-meta/counts |
| XPath 式 $p_{16}$: | //article-meta/article-id/preceding-sibling::*copyright-year* |
| 変換結果: | *nil* |
| XPath 式 $p_{17}$: | //journal-meta/journal-id/preceding-sibling::journal-title |
| 変換結果: | //journal-meta/journal-id/preceding-sibling::*journal-title-group*/journal-title |
| XPath 式 $p_{18}$: | //journal-meta/journal-title/following-sibling::issn |
| 変換結果: | //journal-meta/*journal-title-group*[journal-title]/following-sibling::issn |
| XPath 式 $p_{19}$: | //named-content/*chem-struct-wrapper*/preceding-sibling::fig |
| 変換結果: | //named-content/*chem-struct-wrap*/preceding-sibling::fig |
| XPath 式 $p_{20}$: | //named-content/array/following-sibling::*chem-struct-wrapper* |
| 変換結果: | //named-content/array/following-sibling::*chem-struct-wrap* |
| XPath 式 $p_{21}$: | //article-categories/series-title/preceding-sibling::*subj-group* |
| 変換結果: | //article-categories/series-title/preceding-sibling::*subj-group* \| //article-categories/series-title/following-sibling::*subject* |
| XPath 式 $p_{22}$: | //article-categories/series-title/following-sibling::*subj-group*/subject |
| 変換結果: | //article-categories/series-title/following-sibling::*subject* |
| XPath 式 $p_{23}$: | //article-categories/*subj-group*/following-sibling::series-title |
| 変換結果: | //article-categories/*subject*/following-sibling::series-title |

# 第 7 章

# 結論

本論文では，要素の削除を許すスキーマ進化に伴う XPath 式修正アルゴリズムを提案した．評価実験の結果，現実世界で実際に起こるスキーマ進化に対しても適用可能であると考えられる．

今後の課題としては，対象とする XPath 式の拡張，例えば parent 軸，ancestor-or-self 軸への対応である．また，複数の更新操作で一つの意味を持つ更新への対応が考えられる．更に，スキーマの属性や実体参照への対応，評価実験の拡充が挙げられる．

# 謝辞

本研究を進めるに当たり，適切なご指導・ご助言をしていただきました，指導教員の鈴木伸崇先生に心から感謝致します．森嶋厚行先生には，発表会や合同ゼミにて本研究に関してご助言をいただきました．また，阪口哲男先生には，日頃の研究活動に関してご助言をいただきました．ここに感謝致します．そして，日頃のゼミ等にてご助言をしていただきました，池田光雪さん，水本弘貴さんありがとうございました．活発な議論に付き合っていただいた，阪口研の瀬尾崇一郎さんありがとうございました．

# 参考文献


[1] *"Journal Publishing Tag Set" NLM Journal Archiving and Interchange Tag Suite*. http://dtd.nlm.nih.gov/publishing/.

[2] *"MEDOC-HOME" MSR Home*. http://www.msr-wg.de/medoc/index.html.

[3] *"MSR Download" MSR Home*. http://www.msr-wg.de/medoc/downlo.html.

[4] *"NLM DTD* から *Journal Article Tag Suite* への進展：これまでの経過整理 *" PDF*、組版と文書変換のアンテナハウス株式会社. http://www.antenna.co.jp/xml/xmllist/JATS-status.html.

[5] B. Anders, B. Scott, C. Don, F. F. Mary, K. Michael, R. Jonathan, and S. Jerome, editors. *XML Path Language (XPath) 2.0 (Second Edition)*. http://www.w3.org/TR/xpath20/.

[6] M. Benedikt and J. Cheney. Destabilizers and independence of XML updates. In *Proc. VLDB ENDOW*, volume 3, pages 906–917, 2010.

[7] C. Byron. What are real DTDs like? In *Proc. WebDB*, pages 43–48, 2002.

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

[9] K. Horie and N. Suzuki. Extracting differences between regular tree grammars. In *Proc. ACM SAC*, pages 859–864, 2013.

[10] Z. G. Ives, A. Y. Halevy, and D. S. Weld. An XML query engine for network-bound data. *The VLDB Journal*, 11(4):380–402, 2002.

[11] M. Junedi, P. Genevès, and N. Layaïda. XML query-update independence analysis revisited. In *Proceedings of the 2012 ACM symposium on Documnt engineering*, pages 95–98, 2012.

[12] E. Leonardi, T. T. Hoai, S. S. Bhowmick, and S. Madria. DTD-Diff: a change detection algorithm for DTDs. In *Proc. DASFAA*, pages 817–827, 2006.

[13] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology*, 5:660–704, 2005.

[14] R. Oliveira, P. Genevès, and N. Layaïda. Toward automated schema-directed code revision. In *Proceedings of the 2012 ACM symposium on Documnt engineering*, pages 103–106, 2012.

[15] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. ACM PODS*, pages 35–46, 2000.

[16] H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput*, 19(3):424–437, 1990.

[17] H. Su, D. Kramer, L. Chen, K. Claypool, and E. A. Rundensteiner. XEM: managing the

evolution of XML documents. In *Proc. IEEE RIDE*, pages 103–110, 2001.

[18] N. Suzuki. An edit operation-based approach to the inclusion problem for DTDs. In *Proc. ACM SAC*, pages 482–488, 2007.

[19] B. Tim, P. Jean, C. M. Sperberg-McQueen, M. Eve, and Y. Francois, editors. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. http://www.w3.org/TR/xml/.

[20] Y. Wu, N. Lele, R. Aroskar, S. Chinnusamy, and S. Brenes. XQGen: an algebra-based XPath query generator for micro-benchmarking. In *Proc. CIKM*, CIKM '09, pages 2109–2110, 2009.

[21] 大野敦司， 石原靖哲， 藤原融. XML スキーマで定義された型と XPath 式との対応の解析手法. **情報処理学会研究報告**, 2009-MPS-75(20):1–7, 2009.

[22] 森本卓爾，橋本健二， 石原靖哲， 藤原融. 木埋め込み関係に基づく XML スキーマ進化に応じた XPath 問合せ変換. **電子情報通信学会技術研究報告**, 107(131):DE2007–40, 109–114, 2007.

[23] 橋本健二， 石原靖哲， 藤原融. XML データベースにおけるスキーマ進化のための更新操作群とそれらのスキーマ表現能力保存に関する性質. **電子情報通信学会和文論文誌** *D*, J90-D(4):990–1004, 2007.

# 本論文に関連する発表論文

- 国際会議論文

  K. Hasegawa, K. Ikeda, and N. Suzuki. An Algorithm for Transforming XPath Expressions According to Schema Evolution. First International Workshop on (Document) Changes: Modelling Detection, Storage and Visualization (co-located with ACM DocEng 2013), 8p, 2013.

- 国内研究会論文

  長谷川数馬, 池田光雪, 鈴木伸崇. スキーマ進化に伴う XPath 式修正アルゴリズムの提案. 第 5 回データ工学と情報マネジメントに関するフォーラム (DEIM2013), B2-2, 2013.

- 口頭発表

  長谷川数馬, 池田光雪, 鈴木伸崇. スキーマ進化に伴う XPath 式修正アルゴリズム. 情報処理学会第 74 回全国大会, 4P-6, 2p, 2012.