社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－DBS－143 (19)
2007／7／2

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

森本 卓爾† 橋本 健二† 石原 靖哲† 藤原 融†

† 大阪大学大学院情報科学研究科 〒565-0871 大阪府吹田市山田丘 1-5
E-mail: †{takuji-m,k-hasimt,ishihara,fujiwara}@ist.osaka-u.ac.jp

あらまし 本稿ではXPath部分クラスの問い合わせを，木埋め込み関係に基づいたXMLスキーマ進化に応じて正しく変換する手法を提案する．正しい変換とは，進化前スキーマに従うXML文書に対する問い合わせ結果と，進化後スキーマに従うように変換した任意のXML文書に対する問い合わせ結果が等しい変換を言う．提案した手法が正しい変換を行っていることを証明する．

キーワード XML，XPath，スキーマ進化，問合せ変換

Translating XPath queries according to XML schema evolution based on the tree-embedding relation

Takuji MORIMOTO†, Kenji HASHIMOTO†, Yasunori ISHIHARA†, and Toru FUJIWARA†

† Graduate School of Information Science and Technology, Osaka University
Yamadaoka 1-5, Suita-shi, Osaka, 565-0871 Japan
E-mail: †{takuji-m,k-hasimt,ishihara,fujiwara}@ist.osaka-u.ac.jp

Abstract A method is proposed for translating queries in a subclass of XPath according to XML schema evolution based on the tree-embedding relation. A query translation is said to be correct if the result of every query to an XML document valid against the schema before evolution is equal to the result of the translated query to the XML document valid against the schema after evolution. Correctness of that the translation method is correct is proved.

Key words XML, XPath, Schema Evolution, Query Translation

1. まえがき

XMLは記述言語としての機能とメタ言語としての機能を備えており，その柔軟性と互換性からデータ交換用のフォーマットとして広く用いられている．また，近年注目を集めている知識管理においても，個人の持つノウハウなどの知識をXML文書で記述して共有するという方法が用いられており，様々な分野でXMLの重要性が高まっている．

データ交換用フォーマットや知識管理でXMLを用いる場合，組織構造やビジネスプロセスに変化があったときにXML文書の構造，つまりスキーマを進化させる必要がある．近年，顧客のニーズの多様化，法制の改変，M&Aなどで組織構造やビジネスプロセスの変更が頻繁に起こっているために，スキーマを頻繁に進化させなければならない．

また，スキーマの進化にあたり，進化前のスキーマで記述したXML文書やそれに対する問合せを進化後のスキーマに対応するように変換することが望まれるが，XML文書や問合せの量が膨大である場合，手作業で変換するには莫大なコストを要してしまう．そこで，本稿では，XML文書間の情報保存の関係を木埋め込み関係としてとらえ，その関係に基づくXMLスキーマ進化[3]に応じて，機械的にXML問合せを変換する手法を提案する．

文献[3]では，木埋め込み関係に基づくXMLスキーマ進化を行うためのスキーマ更新操作群を提案している．本稿では，進化前スキーマとスキーマ更新操作群を用いて，進化前スキーマに従うXML文書に対する問合せを進化後スキーマに従うように変換したXML文書[4]に対する問合せに正しく変換する手法を提案する．そして，その手法による変換が正しいことを証明する．ここで，正しい変換とは，変換前XML文書から変換前問合せで得られるノード集合と変換後の任意のXML文書から変換後問合せで得られるノード集合がノードIDにおいて等しくなるような問合せ変換を言う．

文献[3]のスキーマ進化に従ってXML文書を変換する場合変換後のXML文書は一意に決まらないが，本稿で提案した手法で変換した問合せは，[4]によって変換した任意のXML文書に対して適用可能である．また，本稿では問合せとしてXPathの部分クラスを採用しており，親，子，先祖，子孫軸と比較演算の述語を扱える実用上十分なクラスである．

表 1 問合せクラス XP の構文定義
Table 1 Syntax of the class XP of queries

XP	::=	$Path \mid Path$ "$\cup$" XP
$Path$	::=	$RelativePath \mid AbsolutePath$
$AbsolutePath$	::=	"/" $RelativePath$
$RelativePath$	::=	$LocationStep$
		$\mid LocationStep$ "/" $RelativePath$
$LocationStep$	::=	$Axis$ "::" $Label$ "[" $PredSequence$ "]"
$Axis$	::=	"child" | "desc" | "par" | "anc" | "self"
$Label$	::=	(任意の名前)
$PredSequence$	::=	$Pred \mid Pred\ PredSequence$
$Pred$	::=	$RelativePath$
		$\mid RelativePath\ Op\ Value$
Op	::=	"=" | ">" | "<" | "=>" | "=<"
$Value$	::=	' " ' (『"』以外の任意の名前) ' " '

表 2 問合せクラス XP の意味定義
Table 2 Semantics of the class XP of queries

$S[Path_1$ "$\cup$" $Path_2](N_0)$	:=	$S[Path_1](N_0) \cup S[Path_2](N_0)$
$S[$"/"$](N_0)$	:=	$\{root\}$
$S[$"/" $RPt](N_0)$	:=	$S[RPt](\{root\})$
$S[$"/" LSt "/" $RPt](N_0)$	:=	$S[RPt](S[LSt](N_0))$
$S[\chi$ "::" $l\ PSq](N_0)$	:=	$\chi(N_0) \cap L(l) \cap S_{\leftarrow}[PSq]$
$S_{\leftarrow}[[e_1]...[e_k]]$	:=	$S_{\leftarrow}[e_1] \cap ... \cap S_{\leftarrow}[e_k]$
$S_{\leftarrow}[\chi$ "::" $l[e_1]...[e_k]]$	:=	$\chi^{-1}(S_{\leftarrow}[[e_1]...[e_k]] \cap L(l))$
$S_{\leftarrow}[\chi$ "::" $l[e_1]...[e_k]$ $Op\ Value]$	:=	$\chi^{-1}(S_{\leftarrow}[[e_1]...[e_k]] \cap L(l, Op, Value))$
$S_{\leftarrow}[\chi$ "::" $l[e_1]...[e_k]$ "/" $RPt]$	:=	$\chi^{-1}(S_{\leftarrow}[[e_1]...[e_k]] \cap L(l) \cap S_{\leftarrow}[RPt])$
$S_{\leftarrow}[\chi$ "::" $l[e_1]...[e_k]$ "/" $RPt\ Op\ Value]$	:=	$\chi^{-1}(S_{\leftarrow}[[e_1]...[e_k]] \cap L(l) \cap S_{\leftarrow}[RPt\ Op\ Value])$

2. 諸 定 義

この節では本稿で用いる諸概念の定義を行う．

XML 文書と XML スキーマ: 本稿では XML 文書を木，XML スキーマを正規木文法としてモデル化する．正規木文法は以下のように定義される．

[定義 1] 正規木文法は以下のような四つ組 $G = (N, \Sigma, S, P)$ である．

- N は非終端記号の有限集合．
- Σ は終端記号の有限集合．
- S は開始記号集合．
- P は $X \to a(r)$ の形をした生成規則の有限集合．ただし，$X \in N, a \in \Sigma$ であり，r は N 上の正規表現である．$X, a(r)$ をそれぞれこの規則の左辺，右辺と呼び，a, r をそれぞれこの規則のラベル，内容モデルと呼ぶ．また，ある規則 p の左辺，右辺，ラベル，内容モデルをそれぞれ $left(p)$，$right(p)$，$label(p)$，$cm(p)$ と記述する． □

一般性を失うことなく，本稿では XML 文書の根は特殊なノード $root$，生成規則の開始記号集合 S は $S = \{ROOT\}$，$ROOT$ を左辺に持つ規則は P 中にただ 1 つだけ存在し，その規則は $root$ を生成すると仮定する．XML 文書中のラベルは木のノードのラベルとして表現する．木 t にノード x が含まれることを $x \in t$ と記述する．

スキーマ更新操作: 文献 [3] では 2 種類のスキーマ更新操作群 A と B を提案しており，スキーマ更新操作群 A と B ではくくり出しと呼ぶ操作の能力が異なる．本稿ではスキーマ更新操作群 A を用い，以降，単にスキーマ更新操作群と呼ぶ．提案されているスキーマ更新操作群は，4 つのスキーマ更新操作から成る．以下に 4 つの操作の概要を記す．

- くくり出し: 規則 $p = (A \to a(r)) \in P$ を生成規則の集合から削除し，代わりに規則 $p_1 = A \to a(r_b)$ と $p_2 = X \to x(r_x)$ を追加する．ただし，r_b に出現する X を r_x に置き換えた正規表現は，r と等しくなければならない．
- 挿入: 新たな非終端記号を 1 つ内容モデルに追加する．
- 拡張: 新たな規則を 1 つ追加する．
- 等価変換: 生成可能な XML 文書の集合が等しいという条件を満たすスキーマを変換する．

ノード ID: 木の各ノードにつけられたユニークな整数をノード ID と呼ぶ．ノード x のノード ID を $id(x)$ で表す．ある木 t のスキーマが進化し，それに伴い木 t が t' に変化したとき，進化によって追加されたノードには，$(\max_{x \in t'} id(x)) + 1$ 以上の整数を小さいものから順にノード ID として割り当てる．

スキーマ進化によって木 t が木の集合 T' に変化したとき，スキーマ更新操作列の性質と上述のノード ID 割り当てから，任意のノード $x \in t$ と任意の $t' \in T'$ について，$id(x) = id(y)$ を満たすノード y が木 t' にただ 1 つ存在する．このとき，y は x に対応したノードであるという．あるノード集合 N に対して，集合 $\{id(n) \mid n \in N\}$ をノード ID 集合と呼び，N^{id} と記述する．

問合せ: 問合せは木から条件に従うノードの集合を取り出す，木からノード集合への写像である．木 t に問合せ q を施して得られるノード集合を $q(t)$ と記述する．本稿で提案する変換手法の入力及び出力をクラス XP として定義する．クラス XP の構文定義を表 1 に記す．表 1 中の $Label$ は XPath において通常ノードテストと呼ばれるが，本稿では要素名及び $*$ のみを対象としているので，ラベルと呼ぶ．$*$ は任意の要素名を指し，以降，$*$ をワイルドカードと呼ぶ．また，本稿では /child::を/，/desc::を//と略記する．

文献 [1] では集合演算によって問合せの意味定義を行っている．本稿では [1] の意味定義を利用して，クラス XP の問合せの意味定義を表 2 のように定義する．コンテキストノード集合 N_0 における XP クラスの問合せ X の意味を $S[X](N_0)$ により表す．ここで，N_0 は XML 文書中の全要素集合である．$S_{\leftarrow}[P]$ は述語 P の条件を満たす要素の集合を表す．また，$k = 0$ のとき，$S_{\leftarrow}[e_1] \cap ... \cap S_{\leftarrow}[e_k]$ は全要素集合を返す．χ は軸を表し，$\chi(N_0)$ は N_0 から軸 χ を辿って得られるノード集合である．χ^{-1} は χ の逆軸であり，$self^{-1} = self$，$par^{-1} = child$，$desc^{-1} = anc$ とする．$L()$ は引数として $Label$ のみを取る場合と，$Label$ と Op と $Value$ の 3 つを取る場合がある．$Label$ を取る場合は，XML 文書中の全要素集合のうち，要素名と

表 3　スキーマの例
Table 3　An example of a schema

0	:	$ROOT$	$\rightarrow$	$root(A)$	2	: B	$\rightarrow$ $b(\epsilon)$
1	:	A	$\rightarrow$	$a(B \mid B')$	3	: B'	$\rightarrow$ $b(\epsilon)$

して *Label* を持つ要素の集合を返す．*Label* と *Op* と *Value* の 3 つを取る場合は，XML 文書中の全要素のうち，要素名として *Label* を持ち，かつ，その要素が持つ値 v に対して式 v *Op Value* が成立する要素の集合を返す．また，表中では *Label* を *l*，*RelativePath* を *RPt*，*LocationStep* を *LSt*，*PredSequence* を *PSq* と略記する．

正しい変換:　スキーマ進化に応じて木 t が木の集合 T' 中のいずれかの木に変換され得ると仮定する [4]．問合せ q を問合せ q' に変換するとき，変換が正しいことを次のように定義する．

[定義 2]　任意の $t' \in T'$ について，$q(t)^{id} = q'(t')^{id}$ が成立するならば，q から q' への変換は正しい．　□

3. 問合せ変換

正規木文法により生成した木の各要素と非終端記号との対応は一意に決まらないため，スキーマ更新操作を直接問合せに適用することは難しい．加えて，逆方向の軸やワイルドカードも問合せ変換を難しくする要因となる．

[例 1]　表 3 はスキーマ生成規則の例である．列の一番左の整数は規則に割り当てられた ID である．このスキーマで生成される木に対して **/a/b** という問合せを処理する場合，問合せ中の **b** が生成規則 2 によって生成された **b** であるのか，生成規則 3 によって生成された **b** であるのかが分からない．ここで，くくり出しによって生成規則 1 が削除され，代わりに

$$1' : A \rightarrow a(X \mid B'),\ 4 : X \rightarrow x(B)$$

が追加された場合を考える．このとき，問合せ **/a/b** を **/a/x/b** と変換してしまうと，生成規則 3 で生成された **b** が問合せ結果から除外されてしまう．従って，問合せを **/a/b**∪**/a/x/b** に変換する必要がある．　□

そこで，本稿では問合せを解析して問合せの各ロケーションステップのラベルと生成規則の非終端記号の対応を求め，逆方向の軸やワイルドカードを除去してから問合せを変換する方法を提案する．提案する問合せ変換は 3 つのステップで構成される．まず，与えられた問合せに対して解析を行い，スキーマ生成規則から成る解析木を生成する．次に，与えられたスキーマ更新操作列を用いて解析木を変換する．最後のステップで変換後の解析木を問合せに戻し，求める問合せを得る．以降，スキーマ $G = (N, \Sigma, S, P)$ のスキーマ進化に応じた問合せ変換の手順をステップ毎に説明する．

3.1　ステップ 1: 解析

このステップでは与えられた問合せの各ロケーションステップのラベルが，どのスキーマ生成規則の終端記号であるかを解析する．解析の結果は解析木と呼ばれる木の集合である．解析木の例を図 1 に示す．図中のノードを左肩に記した数字で呼ぶ．解析木のノードはロケーションステップのラベルに対応した生成規則，述語の情報を持ち，辺は軸の情報を持つ．解析木のノード n の持つロケーションステップに対応した生成規則を $rule(n)$，述語の比較演算子を $op(n)$，値を $val(n)$ と記述する．$rule(n)$ は必須であり，$op(n)$ と $val(n)$ は空であっても構わない．図 1 ではノード 5 のみが 3 つの要素を持っている．問合せの最右ロケーションステップのラベルに対応する解析木のノードを終端ノードと呼ぶ．図中では終端ノードを 2 重丸で表している．また，解析木の辺は後述の軸とラベルに応じた処理で用いるためのオートマトンを持つことがあり，親ノード n，子ノード m 間でオートマトン A を持つ辺を (n, m, A) と記述する．オートマトンを持たない場合は，$A = \phi$ である．図 1 ではノード 4 と 7 の間の辺のみがオートマトンを持っている．

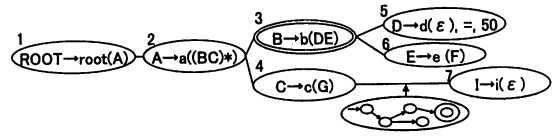

図 1　解析木の例
Fig. 1　An example of an analysis tree

問合せから解析木を求めるアルゴリズムを述べる．*ResultSet*，*MiddleSet* を生成途中の解析木の集合とする．生成途中の解析木はノード処理のためのヘッダを持つ．ヘッダはいずれかのノードを指し，そのノードを基点としてノードの追加やヘッダの移動を行う．*ResultSet* は初期値として，以下のようなただ 1 つの解析木を持つ．

- 解析木はただ 1 つのノードから成り，そのノードは *root* を生成する規則のみを持つ．他の要素 $(Op, Value)$ は空である．
- 解析木のヘッダはそのノードを指す．

ロケーションステップ *ls* の軸，ラベル，述語をそれぞれ $axis(ls)$，$label(ls)$，$pred(ls)$ と記述する．解析のアルゴリズムの概略は以下の通りである．

問合せの先頭から順に，各ロケーションステップ *ls* について
　ResultSet のすべての解析木 t について
　　ロケーションステップに応じた処理 $(ls,\ t)$ の結果を *MiddleSet* に追加
　　if *ls* が問合せ最右のステップ **then**
　　　t のヘッダが指すノードを終端ノードとする
　ResultSet ← *MiddleSet*

ロケーションステップに応じた処理 (ls, t)

解析木の集合を格納するローカルな変数 *TempSet1* と *TempSet2*，ノードへのポインタ m を用意する．ロケーションステップ *ls* の軸とラベルに応じて解析木 t に以下の処理を加え，*TempSet2* を返す．

t のヘッダの指すノードを m に格納
TempSet1 ← **軸とラベルに応じた処理** $(ls,\ t)$
TempSet1 のすべての解析木 s について
　述語の処理 $(ls,\ s)$ の結果を *TempSet2* に追加

TempSet2 のすべての解析木 u について
u のヘッダを m の指すノードに移動

軸とラベルに応じた処理 (ls, t)

解析木の集合を格納するローカルな変数 *TempSet* を用意する．解析木 t に対して t のヘッダの指すノード n を基点とし，ロケーションステップの軸 $axis(ls)$ とラベル $label(ls)$ に応じて処理する．処理は軸に応じて大きく 5 つに分けられる．それぞれの処理は結果を *TempSet* に格納しており，*TempSet* を呼び出し側に返すことで軸とラベルに応じた処理を終了する．

(i) $axis(ls) = child$ の場合は以下の要領で n の子として新たなノードを加える．1 つも子を追加されなかった解析木は *TempSet* に追加されない．

すべての $p \in P$ について
if $left(p)$ が $cm(rule(n))$ に出現し，かつ
$label(ls) = label(p)$ *or* $*$ **then**
n の子に p を追加した解析木を
TempSet に追加（辺 (n, m, A) は $A = \phi$）

(ii) $axis(ls) = desc$ の場合，n から $label(ls)$ への導出をすべて求め，導出の最右の生成規則（$label(ls)$ に対応する生成規則）を持つノードを n の子に追加し，n との辺に導出を表すオートマトンを追加する．このオートマトンは後述の *par* 軸や *anc* 軸の処理によってヘッダを移動する際に用いる．

導出 g は生成規則集合 P 上の正規表現 reg と生成規則 p を用いて $g = reg\ p$ の形で表すことができるので，すべての導出 g について，n の子にノード $v(rule(v) = p)$ を追加した解析木を *TempSet* に追加する．n と v 間の辺は (n, v, A) であり，A は reg が表す言語を受理するオートマトンである．また，一般性を失うことなく，p が同じであるような異なる導出 $g_1 = reg_1\ p$ と $g_2 = reg_2\ p$ は存在しないと仮定できる．それらは 1 つの導出 $g = (reg_1 + reg_2)\ p$ にまとめることができるからである．導出が存在しない場合，解析木 t は *TempSet* に追加されない．

すべての導出 $g = reg\ p$ について，reg が表す言語を受理するオートマトン $A = (Q, \Sigma', \delta, q_0, F)$ を，スキーマ $G = (N, \Sigma, S, P)$，n，$label(ls)$ を用いて以下のように構成する．

- 状態集合 $Q = P$.
- 入力記号 $\Sigma' = P$.
- 遷移関係 $\delta = \{(q, q', q') \mid q, q' \in Q,\ left(q')$ が $cm(p)$ に出現 $\}$. ここで，遷移関係 $\delta(q_1, q_2, q_3)$ の q_1 は遷移前状態，q_2 は入力記号，q_3 は遷移先状態を表す.
- 初期状態 $q_0 = rule(n)$.
- 受理状態集合 $F = \{q \mid \delta(q, p, p),\ label(p) = label(ls)\}$.

(iii) $axis(ls) = anc$ の場合，t のヘッダが指すノード n の先祖ノードの該当箇所に，以下の要領でヘッダを移動させた解析木を *TempSet* に追加する．

t のヘッダが指すノード n の各先祖ノード m について
if $label(rule(m)) = label(ls)$ **then**
t のヘッダを m に移動した木を *TempSet* に追加

さらに，辺の持つオートマトンのある状態に対応する生成規則のラベルが $label(ls)$ と等しい場合を考える．すなわち，n 及び n の先祖ノード間の辺 $(l, m, A)(A \neq \phi)$ について，オートマトン A のある状態 $q \in Q$ が以下の条件を満たすとする．

- $label(q) = label(ls)$
- A の初期状態から q，q から A の受理状態に到達可能

この場合，辺 (l, m, A) に新たなノードを挿入し，そのノードにヘッダを移動させた解析木を *TempSet* に追加する．より形式的に述べると，まずオートマトン A が受理する言語を L として $L \cap P^* q P^* = L_1 q L_2$ を満たす言語 L_1 と L_2 を求める．次に，t にノード o（$rule(o) = q$），辺 (l, o, A_1)，(o, m, A_2) を追加し，(l, m, A) を削除する．A_1 と A_2 はそれぞれ言語 L_1 と L_2 を受理するオートマトンである．よって，この場合の処理は以下の通りである．

n 及び n の先祖ノード間の
すべての辺 (l, m, A) $(A \neq \phi)$ について
$label(q) = label(ls)$ を満たす，
A のすべての状態 q について
上記の要領でノード o と辺を追加
ヘッダを o に移動し，*TempSet* に追加

上記 2 つの処理でヘッダの移動先が 1 箇所もない場合，その解析木は破棄される．また，$label(ls) = *$ の場合，ヘッダの指すノードの先祖ノードすべてがヘッダの移動先となる．加えて，ヘッダの指すノードとそのすべての先祖ノード間の辺が持つオートマトンの（到達可能条件を満たす）状態について上記の処理を行う．

(iv) $axis(ls) = par$ の場合，t のヘッダが指すノードを n，n と親ノード m 間の辺を (m, n, A) とし，A によって 2 種類の処理を行う．$A = \phi$ のときはヘッダを親に移動する．そうでない場合は $axis(ls) = anc$ のときと同様にオートマトンの状態について処理を行う．A の状態 q に処理を加える条件は，以下の通りである．

- $label(q) = label(ls)$
- A の初期状態から q に到達可能
- $\delta(q, f, f)$ を満たす受理状態 $f \in F$ が存在

以上をまとめると次のようになる．

if $A = \phi$ **then**
if $label(m) = label(ls)$ $\cdots$ $(\diamond)$ **then**
t のヘッダを m に移動して *TempSet* に追加
else (if $A \neq \phi$)
処理を加えるべき状態 q について
$axis(ls) = anc$ の場合と同様にノード o と辺を追加
ヘッダを o に移動し，*TempSet* に追加

$label(ls) = *$ の場合は $axis(ls) = anc$ のときと同様に，ラベルに関わらず上記処理を行う．すなわち，$(\diamond)$ を記した条件が常に真となる．

(v) $axis(ls) = self$ の場合，t のヘッダの指すノード n として，$label(ls) = label(n)$ *or* $*$ ならば t を *TempSet* に追加する．

ヘッダの移動は行わない．

述語の処理 (ls, s)

解析木の集合を格納するローカルな変数 *TempSet* を用意する．$pred(ls)$ の *RelativePath* について，解析アルゴリズムを再帰的に使用する．解析アルゴリズムでは，初期状態で *ResultSet* が持つ解析木はただ 1 つのノードを持つような木であったが，ここでは解析木 s を持つことになる．*RelativePath* の解析結果を *TempSet* に格納し，$pred(ls) = RelativePath$ の場合は *TempSet* のすべての解析木の終端ノードを通常のノードに戻す．$pred(ls) = RelativePath\ op\ value$ の場合は，終端ノード l に対して $op(l) = op$，$val(l) = value$ を代入し，通常のノードに戻す．呼び出し側に *TempSet* を返し，処理を終了する．

3.2 ステップ 2: スキーマ更新操作による変換

このステップではスキーマ更新操作列の各操作を先頭から順に，3.1 節で得られた解析木の集合に適用する．スキーマ更新操作は [3] で 4 種類提案されているが，くくり出し以外の変換に関しては以下の理由から変換を行う必要がない．

- 挿入: 挿入では新たにノードを追加するが，[3] の定義では追加されたノードのラベルはスキーマ進化前に従う XML 文書に含まれず，変換前問合せがノードを求めることはない．定義 2 より，変換後の問合せは挿入によって追加されたノードを求めないので，挿入に応じて問合せを変換しない．
- 拡張: [4] の XML 文書変換手法では，スキーマに拡張操作を施しても，XML 文書が変化しない．そのため，問合せを拡張に応じて変換することはない．
- 等価変換: 等価変換は，生成できる XML 文書が等しくなるという条件の下で，生成規則を自由に変換する．XML 文書自体に変換は行われないが，生成規則の終端記号や非終端記号がすべて変換されてしまう可能性があるので，これまでの解析木が使用不能となる．そのため，等価変換が行われた場合はステップ 3 に移行して問合せ変換を終了した後，残りの更新操作列と新たな生成規則を用いて問合せ変換を行う．

では，くくり出しに応じた変換手法を述べる．解析木の集合を格納するローカルな変数 *MiddleSet* と *ResultSet* を用意する．くくり出しによって規則 $p = (A \rightarrow a(r)) \in P$ が生成規則の集合から削除され，代わりに規則 $p_1 = A \rightarrow a(r_b)$ と $p_2 = X \rightarrow x(r_x)$ が追加されるとする．ステップ 1 で得られた解析木集合を G とし，以下の要領で解析木を変換する．

すべての解析木 $g \in G$ について
  MiddleSet を空にする
  g 中のすべての辺 (n, m, A) $(A \neq \phi)$ について
    if $rule(n) = p$ **then**
      if $left(rule(m))$ が r_x に出現 **then**
        $rule(n) = p_1$ とし，$rule(l) = p_2$ である
        ノード l を n,m 間に挿入した g を
        MiddleSet に追加
      if $left(rule(m))$ が r_b に出現 **then**
        g を *MiddleSet* に追加
  ResultSet ← *MiddleSet*

上記の結果得られる *ResultSet* はスキーマ更新操作（くくり出し）1 つに応じて解析木を変換した結果である．*ResultSet* を G とし，スキーマ更新操作列に応じて繰り返し上記の処理を行い，最終的に得られた *ResultSet* がステップ 2 の出力である．

3.3 ステップ 3: 解析木から問合せへの変換

このステップでは，ステップ 2 で変換した解析木の集合を問合せに変換する．ステップ 2 で得られた解析木の集合を G'，解析木 g を問合せに変換したものを $path(g)$ とすると，この問合せ変換で求める問合せ q' は

$$q' = \bigcup_{g' \in G'} path(g') \tag{1}$$

である．$path(g)$ は以下のように求める．

まず，g の根から終端ノードへの経路上のノードを問合せに変換する．問合せを格納する変数を q とする．

経路上のノード n について根から順に
  if n が根 **then** q に / を追加
  else
    if n の親ノードとの辺がオートマトンを持つ **then**
      q の右にロケーションステップ
        / $desc :: label(rule(n))$ を追加
    else
      q の右にロケーションステップ
        / $child :: label(rule(n))$ を追加
    if n が比較演算子と値を持つ **then**
      q に $[self\ \ op(n)\ \ \text{“}val(n)\text{”}]$ を追加

ノード n の子 m が根と終端ノードの経路上にない場合も経路上にある場合と同様に問合せに変換するが，変換結果は n に対応する述語として q に格納される．

[例 2] 図 1 の解析木から問合せへの変換を示す．根から終端ノードまでの経路上にあるノードは 1，2，3 であり，それ以外のノードはすべて述語内のステップとなる．よって，図 1 の解析木は **/a[c//i]/b[d[self=“50”]][e]** に変換される． □

4. 変換が正しいことの証明

3 節で提案した問合せ変換が正しいことを証明する．本稿では紙面の都合上，証明の概略のみを記す．進化前スキーマに従う XML 文書と問合せをそれぞれ t，q とする．スキーマ進化に応じて t を変換して得られる XML 文書の集合を T'，q をスキーマに応じて解析し，得られた解析木の集合を G とする．

解析木の集合 G は問合せ q のラベルと生成規則の非終端記号のすべての組み合わせを含んでいる．さらに，q 内のワイルドカードと逆方向の軸もスキーマ情報を用いて除去されている．逆方向の軸を含む問合せを順方向の軸のみを含む問合せに変換する手法は [2] で提案されている．本稿での逆方向の軸の除去は，[2] の手法と手順が少し異なるが，本質的には同じである．よって，任意の木 t について

$$q(t) = \Big(\bigcup_{g \in G} path(g) \Big)(t) \tag{2}$$

である．ここで，G のすべての要素を更新操作に応じて変換して得られる集合を G' とし，任意の $t' \in T'$ について

$$\left(\left(\bigcup_{g\in G} path(g)\right)(t)\right)^{id} = \left(\left(\bigcup_{g'\in G'} path(g')\right)(t')\right)^{id} \quad (3)$$

であることが証明できたと仮定すると，式 (1)，(2)，(3) より，任意の木 $t' \in T'$ について

$$q(t)^{id} = q'(t')^{id}$$

が成り立つ．定義 2 より，q から q' への変換は正しい．

次に，式 (3) の証明を行う．G の要素数を n，$G = \{g_k \mid 1 \leqq k \leqq n\}$ とする．ある $g_k \in G$ に 1 つ更新操作を施して得られる解析木の集合を G'_k とすると，G を更新操作に対応して変換した結果得られる G' は

$$G' = \bigcup_{k=1}^{n} G'_k \quad (4)$$

である．すべての $t' \in T'$，k について

$$\left(path(g_k)(t)\right)^{id} = \left(\left(\bigcup_{g'\in G'_k} path(g')\right)(t')\right)^{id} \quad (5)$$

が成立することを示せば，式 (4)，(5) より式 (3) が証明できる．

式 (5) は，解析木のサイズによる帰納法で証明する．ノード数 i の解析木 s を任意のくくり出しによって変換した結果得られる解析木の集合を S' とし，

$$\left(path(s)(t)\right)^{id} = \left(\left(\bigcup_{s'\in S'} path(s')\right)(t')\right)^{id} \quad (6)$$

が成立していると仮定する．s にノードを 1 つ追加して構成されるノード数 $i+1$ の解析木について，ノード追加に伴って追加された辺がオートマトンを持つ場合 (i) と，追加された辺がオートマトンを持たず，かつノード追加後の木が分岐を持たず，かつ終端ノードが葉である場合 (ii) とそれ以外の場合 (iii) に分けて考える．(i)，(ii)，(iii) のいずれの場合においても，一般性を失うことなく追加されたノードは葉であると仮定することができ，追加されたノードとその親の間のくくり出しについて考える．帰納法の仮定より，それ以外のすべての辺についてのくくり出しによる変換は正しい．

(i) の場合： s の任意のノード l の子にノードを 1 つ追加して，ノード数 $i+1$ の任意の木を構成する．追加したノードを m とすると，くくり出しに応じた解析木の変換によって，l と m の間にノードが挿入されることはない．くくり出しによって t の要素間にノードが挿入されることがあるが，オートマトンを持つ辺は $path()$ によって $desc$ 軸に変換されるため，問合せの結果は変わらない．よって，ノード数 $i+1$ の任意の木に対して，式 (6) が成立する．

(ii) の場合： s の葉 l に終端ノード m を追加して，ノード数 $i+1$ で分岐を持たない木を構成する．このとき，終端ノードであったノード l は，通常のノードになる．$rule(l) = p_l$，$rule(m) = p_m$ とすると，$cm(p_l)$ に $left(p_m)$ が出現する．くくり出しによって追加される規則を $X \to x(r_x)$ とする．くくり出しによってすべての $left(p_m)$ が X に書き換えられる場合，t' では，規則 p_l と p_m によって生成されるラベル $label(p_l)$ と $label(p_m)$ を持つノードの間には必ずラベル $label(X)$ となるノードが含まれる．一方，提案手法に従い解析木を変換すると，解析木のノード l と m の間に x が挿入される．また，くくり出しによって $cm(p_l)$ の一部の $left(p_m)$ が書き換えられる場合，t' では，規則 p_l と p_m によって生成されるラベル $label(p_l)$ と $label(p_m)$ を持つノードの間に $label(X)$ が含まれる場合と含まれない場合の 2 通りがある．この場合，提案手法による解析木の変換では，問合せのノード l と m の間に x が挿入された解析木と，もとの解析木の 2 通りを変換結果として出力する．また，くくり出しによって書き換えられない場合は，t に対するくくり出しに応じた変換はなく，解析木も変換されない．以上より，くくり出しに応じた木の変換に対応して問合せが変換され，問合せの返す答えが等しくなる．よって，ノード数 $i+1$ の任意の木に対して，式 (6) が成立する．

(iii) の場合： s の任意のノード l の子にノードを 1 つ追加して，ノード数 $i+1$ で分岐を持つ任意の木を構成する．追加したノードを m とすると，このとき，m は終端ノードでなく，$path()$ によって述語に変換されるノードであると仮定しても一般性を失わない．$rule(l) = p_l$，$rule(m) = p_m$ とすると，$cm(p_l)$ に $left(p_m)$ が出現する．くくり出しによって追加される規則を $X \to x(r_x)$ とする．くくり出しによってすべての $left(p_m)$ が X に書き換えられる場合，$cm(p_l)$ の一部の $left(p_m)$ が書き換えられる場合，くくり出しによって書き換えられない場合をそれぞれ (ii) と同様に考える．くくり出しに応じた木の変換に対応して述語が変換され，述語が返す真偽値も変化しない．よって，ノード数 $i+1$ の任意の木に対して，式 (6) が成立する．

5. あとがき

本稿では，木の埋め込み関係に基づくスキーマ進化に応じた XML 問合せ変換手法を提案した．しかし，提案手法にはいくつか改善すべき点が残されている．一つは，中間データとして生成される解析木の集合のサイズが最悪の場合指数的に増加することであり，それを多項式サイズに抑えるための条件を検討することが課題となる．また，一方で，計算量を増やすことなく入力クラスを拡張することも課題である．

文　献

[1] Georg Gottlob, Christoph Koch, and Reinhard Pichler. "Efficient algorithms for proccesing XPath queries," *In Proc. VLDB*, pp. 95-106, 2002.

[2] Michael Benedikt, Wenfei Fan, Gabriel M. Kuper. "XPath: Looking Forward," *In Proc. EDBT 2002 Workshops*, LNCS 2490, pp. 109-127, 2002.

[3] 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，" 電子情報通信学会論文誌 (D) Vol. J90-D, No. 4, pp. 990-1004, 2007.

[4] 吉田 昌起，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文書変換器の生成法，" DBWS（発表予定），2007.