バージョン化されたXML文書に対する問い合わせの書き換え規則

本村徹太郎† 岩井原瑞穂†† 吉川 正俊††

† 京都大学工学部情報学科 〒606-8501 京都市左京区吉田本町
†† 京都大学情報学研究科社会情報学専攻 〒606-8501 京都市左京区吉田本町
E-mail: †motomura@db.soc.i.kyoto-u.ac.jp, ††{iwaihara,yoshikawa}@i.kyoto-u.ac.jp

あらまし オフィス文書やwikiのコンテンツなどにおいて，最新バージョンとともに過去の全バージョンも提供する応用が増えている．XML文書において注目している文書ノードのバージョン祖先を求めるような，ノード単位のバージョン検索を行うには，XPathにバージョン軸の追加が必要であり，我々が提案しているXVerPathを用いることができる．XPathにおける問合せ最適化のための基本的性質である包含関係，および等価性判定は以前から研究されているが，バージョン軸で拡張された場合は知られていない．本論文では，文書軸とバージョン軸との間で等価変換が可能なパターンを変換規則としてまとめる．変換規則により文書軸とバージョン軸の巡行順序を入れ替えるなどの最適化が可能になる．

キーワード XML，バージョン管理，問合せ言語

1. はじめに

近年，wikiのコンテンツやオフィス文書などにおいて，最新のバージョンだけでなく過去の全バージョンも提供されることが多くなってきた．wikiのような，多数のユーザから共有され自由に改変できるコンテンツは，オープンコンテントと呼ばれ，コンテンツ自体を共有するだけでなく，コンテンツの創造過程および更新過程も共有される．また，オフィス文書では，過去のバージョンの文書を保存しておくことにより，現在と過去との文書を比較することが可能になり，誤った編集を元に戻したり，不正な編集を発見できる．

多くのバージョンが提供されるに従い，バージョン検索が重要性を増している．例えば，代表的なオープンコンテンツとして，Wikipedia [1] がある．Wikipediaを利用するユーザには，記事の削除や差し戻しが可能な管理者，記事を執筆する編集者，記事を閲覧する一般の利用者などが存在する．編集者は，自身が編集したパラグラフの更新の変遷を閲覧する，あるいは，全ての更新履歴ではなく，ある期間中に更新された履歴のみを閲覧したい場合がある．また，一般のユーザは，記事全体ではなく記事のカテゴリ情報のみを閲覧することがある．さらに，管理者は，記事編集権のないユーザに，編集用のページを見せないようにして情報を隠す必要がある．

バージョン検索を行うためのツールとして，バージョン管理システムSubversion [2] がある．Subversionは，どのような変更がなされたかを確認するために，二つのバージョンのファイルおよび文書の比較を行い差分を求める機能を備えている．この機能を用いてバージョン検索を行うことが可能であるが，検索の際はファイル全体を比較して表示するため，バージョン検索の度にファイル中のどの箇所が変更されているかを確認しなければならない．そのため，Subversionは，ファイルの一部および部分文書単位でのバージョン検索には適さない．

Wikipediaのような XML 文書の中から，情報を抽出，あるいは検索する基本的な問合せ言語として，XPath [3] [4] が用いられる．XPath 問合せはパスのパターンによって記述され，XML 文書のノード集合からパターンに適合する部分集合を解とする．しかし，XML 文書において文書ノードのバージョン祖先を求めるといった，ノード単位のバージョン検索を行う機能は備えていない．ノード単位のバージョン検索を行うには，ノードレベルでバージョン管理を行う必要がある．検索の際はXPathにバージョン軸の追加が必要であり，岩井原らによって提案された問合せ言語，XVerPath [5] を用いることができる．

XVerPathを用いて上述したノード単位でのバージョン検索を行う際は，ユーザが頻繁に行う問合せをユーザ定義ビューとして，ユーザが閲覧不可能な箇所をフィルタリングするための問合せをセキュリティビューとして，それぞれを定義することが考えられる．バージョン管理システムでは，ビューを定義する機能は提供されていない．ビューを定義しておくことで，問い合わせる際は，ビューと問合せとの包含関係を判定することにより，実際に問い合わせるのではなくあらかじめ用意してある情報を表示すればよい場合がある．

XPathにおける問合せ最適化のための基本的性質である包含関係，および等価性判定は以前から研究されている．Miloおよび Suciu [6] は，XPath 式がノードテスト，child 軸（/），descendant-or-self 軸（//），および wildcard（*）からなる場合において，二つの XPath 式の包含関係の判定問題は，PTIMEであることを示した．また，Miklau および Suciu [7] は，XPath 式がノードテスト，/，//，*，および predicate（[]）からなる場合において，二つの XPath 式の包含関係の判定問題は，co-NP 完全であることを示した．しかし，バージョン軸も含めた場合は知られていない．本論文で示す問合せ書き換え規則は，バージョン軸で拡張された XPath 問合せにおける，包含関係の判定に応用することができる．

本論文では，問合せの書き換えを行う際に有用であると考えられる，変換規則を挙げる．変換規則は，ある更新操作が行われたときに，文書軸とバージョン軸との間において評価順序が交換可能だと考えられるパターンをまとめたものである．文書木に対する更新操作は，文書木の削除，挿入，更新，置換の四つを考える．このとき，文書軸は XPath の/，//，*で表現可能な問合せを対象とし，バージョン軸はノード間のバージョンの親子関係を表現可能である問合せを対象とする．

我々は，次節で議論の対象とするモデルについて定義し，3. 章では，規則の一覧を提示する．4. 章では，更新操作が delete であるときに適用可能な規則を形式化する．同様に，5. 章では，insert について，6. 章では update について，7. 章では replace について，適用可能な規則をそれぞれ形式化する．8. 章では，本論文で得られた規則の応用として，問合せを非冗長化する例をあげる．最後に，本論文のまとめを述べる．

2. データモデルと問合せ言語

この章では，モデルとする XML 木およびバージョングラフの定義，XPath 問合せおよびその文法，XVerPath 問合せのおよびその文法について述べる．

2.1 XML 文書木

今回議論の対象とする XML 木のモデルは，兄弟ノード間に順序のない，かつ各ノードにラベル付けされた XML 文書木である．XML 文書木は $d < N, E, r >$ で表現される．ただし N はラベル付けされたノードの集合，E は文書枝の集合，r は d のルートノードであり，$r \in N$ を満たす．各ノード $n \in N$ はコンテンツを持っており，$content(n)$ で表す．また，label はラベル名を表す．ノード $n \in N$ に対し，$n.label$ は n のラベルを表し，エッジ $e \in E$ に対し $e.label$ は e に付けられたラベルを表す．コンテンツは要素名あるいは属性名のタグを含むテキストであり，テキストの文字の有限集合 Σ に含まれる．

XML 文書木 $d < N, E, r >$ において，二つのノード $a, b \in N$ が与えられたとき，a が b の祖先ノードであるならば，a は b の**文書祖先**であるという．

2.2 デルタバージョングラフ

最初に，XML 文書に対して適用される更新操作について定義する．

定義 1.（更新操作）. *XML* 文書の更新の際に用いる基本的な更新操作を以下のように定義する．

（*1*） $delete(x)$*:* 文書木からノード x をルートノードとする部分木を削除する．ここで x をルートとする部分木を $subtree(x) = (//*)(x)$ とすると，削除されたノードの集合は $subtree(x)$ と表現できる．

（*2*） $insert(x, y)$*:* 文書木に $subtree(y)$ をノード x の子ノードとして挿入する．

（*3*） $update(x, c)$*:* ノード x のコンテンツを $content(x) = c$ となるように更新する．

（*4*） $replace(x, y)$*:* $subtree(x)$ を $subtree(y)$ で置換する．

上にあげた四つの更新操作の任意の操作を δ とする．ある文書木に δ を適用させるたび新たな文書木のバージョンが生成される，というバージョンモデルを採用する．

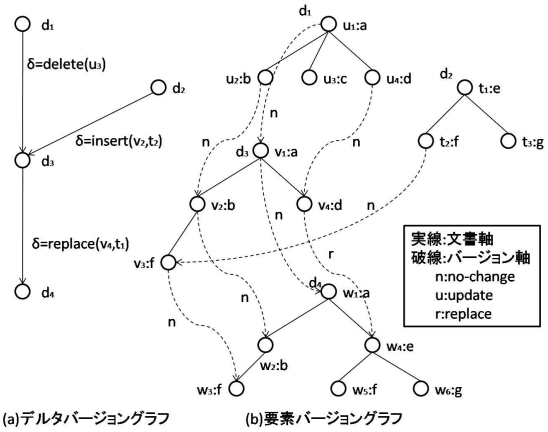

図 1 (a) デルタバージョングラフと (b) 要素バージョングラフ

文書木の集合を D とする．D の要素である文書木 $d_1 \in D$ が，文書木 $d_2 \in D$ に δ を作用させて得られたとする．このとき d_1 は d_2 のバージョン子であるといい，d_2 は d_1 のバージョン親であるという．d_1，d_2 をそれぞれノードとみなし，各文書木ごとに一意に定まる識別子を与えたノードの集合を V_d，ノード d_1 とノード d_2 間のラベル δ を付して張られる有向枝 (デルタ枝という) を $edge(d_1, \delta, d_2)$ と表現し，デルタ枝の集合を E_d とする．V_d，E_d からなる非巡回有向グラフをデルタバージョングラフといい，G_d と表現する．図 1(a) はデルタバージョングラフである．例えば，$edge(d_1, delete(u_3), d_3)$ というバージョン枝は, 文書木 d_1 からノード u_3 が削除され，新たな文書木 d_3 が生成されるということを表している．

2.3 要素バージョングラフ

デルタバージョングラフは，文書レベルでバージョンが割り当てられていた．ここでは，より細粒度の高いノードレベルでバージョンを割り当てられた，要素バージョングラフについて定義する．要素バージョングラフ $G_e < V_e, E_e >$ は，デルタバージョングラフ G_d と XML 文書の集合 D から求めることができる．ただし，V_e はラベル付けされたノード集合，E_e はラベル付けされたバージョン枝の集合である．要素バージョングラフを求めるアルゴリズムは，文献 [5] で示されている通りである．図 1(b) は要素バージョングラフである．例えば，文書木 d_1 のノードの中で，u_3 のみはバージョン子が存在せず，それ以外のノードは，文書木 d_3 にバージョン子が存在する．u_1 から v_1 へのバージョン枝のラベルは”n”であることから，d_3 が生成されたときに，v_1 のコンテンツは u_1 のコンテンツと同一である．一方，v_4 と w_4 間のバージョン枝のラベルは”r”であるため，v_4 のコンテンツは別のコンテンツに置換されて文書木 d_4 が生成される．

要素バージョングラフ $G_e < V_e, E_e >$ において，二つのノード $a, b \in V_e$ が与えられたとき，バージョン軸に関して a が b の祖先ノードであるならば，a は b の**バージョン祖先**であるという．例えば，図 1(b) において，u_4 および v_4 は，w_4 のバージョン祖先である．

2.4 問合せ言語

ここでは，本論文において我々が議論する問合せ言語について述べる．

定義 2.（XPath 問合せ）. 我々は，以下の文法で与える *XPath* のサブセットについて議論する．

$$p \rightarrow \epsilon \mid l \mid * \mid p/p \mid p//p \mid p[q]$$

ここで，ϵ は空のパス式，"l" はラベルである．"$*$"は wildcard であり，任意のラベルを選択する．"/"は child 軸を表し，コンテキストノードの子ノードを選択する．"//"は descendant-or-self 軸を表し，コンテキストノードの子孫ノードを選択する．"[]"は predicate（述語）であり，ノード集合をフィルタリングするものである．predicate は以下で与えられる．

$$q \rightarrow p \mid p\,\theta\,c$$

ここで，c は定数であり，p は上述の式である．θ は比較演算子であり，$<$，$\leqq$，$=$，$\geqq$，$>$，のいずれかである．

このように，XPath のサブセットの中で，//，*，[] からなる問合せクラスを，$XP^{\{//,*,[]\}}$ と表現する．

ある XPath 式の問合せ q と入力の XML 文書木 $d = (N, E, r)$ が与えられたとき，q で r を評価した際に得られるノード集合を $q(r)$ と表現する．2 つの式 q，$q^{'}$ があり q が $q^{'}$ に包含される ($q \subseteqq q^{'}$ と表す) とは，任意のノード集合 R に対して $q(R) \subseteqq q^{'}(R)$ が成り立つときである．2 つの式が等価 ($q \equiv q^{'}$ と表す) であるとは，$q \subseteqq q^{'}$ かつ $q^{'} \subseteqq q$ が成り立つときである．

次に，XPath にバージョン軸を追加した問合せ言語 XVerPath [5] について定義する．

定義 3.（XVerPath 問合せ）. 我々が議論する *XVerPath* のサブセットは，定義 *2* の文法を，以下を含むように拡張したものである．

$$\begin{aligned} p &\rightarrow vpar(el) \mid vchild(el) \\ el &\rightarrow \epsilon \mid ell \\ ell &\rightarrow n \mid u \mid r \mid ell\,,\, ell \end{aligned}$$

ここで，vpar はバージョン親の軸，vchild はバージョン子の軸を表している．el はバージョン枝におけるラベルのリストである．el が ϵ の場合は，任意のラベルとみなす．

v をコンテキストノード，G_e をデルタバージョングラフとする．このとき，$q \in \{vpar(el), vchild(el)\}$ に対し，$q(v)$ は以下で定義される．

$$\begin{aligned} (vpar(el))(v) &= \{\, u \mid edge(u, el, v) \in G_e\} \\ (vchild(el))(v) &= \{\, w \mid edge(v, el, w) \in G_e\} \end{aligned}$$

以降では，特に断らない限り，簡単のために引数の el は添え字として表現する．例えば，$vpar(n)$ は $vpar_n$ として表す．

任意の XVerPath 式を q，デルタバージョングラフを G_d，G_d のノードの部分集合を R とする．このとき q を R に適用した評価結果を $q(R, G_d)$ と表す．ここで，問合せ前後で G_d は変化しないため，議論しているデルタバージョングラフ G_d が明らかな場合は，G_d を省略して $q(R)$ と表す．q はノードの集合を引数とするが，ノードの集合の要素が単一のノード r のみであるときは，$q(\{r\})$ を簡単に $q(r)$ と表記することにする．XVerPath の包含性と等価性は XPath と同様に定義される．

文献 [7] において，任意の問合せ $q \in XP^{\{//,*,[]\}}$ は，同じ意味を持つラベル付けされた木パターンの問合せ（Tree Pattern Query）として表現できることが述べられている．

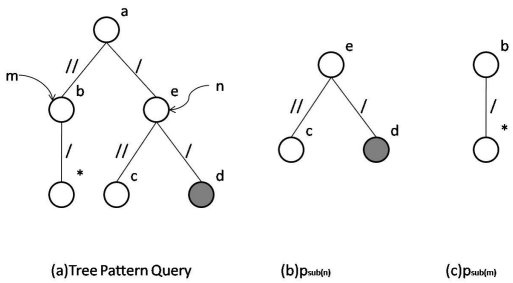

図 2　Tree Pattern Query と subpattern

定義 4. (Tree Pattern Query(TPQ)). V_p をノード集合，E_p をエッジ集合とするとき，*Tree Pattern Query* p は，$p < V_p, E_p, r_p, o_p >$ で表現される．このとき，p は以下を満たす．

- 各ノード $n \in V_p$ は，$\Sigma \cup \{'*'\}$ からなるラベル，あるいは "$m\theta c$"$(m \in \Sigma, \theta \in \{<, \leqq, =, \geqq, >\}, c = const)$ となるラベルを持つ．
- 各エッジ $e \in E_p$ は，$\{'/', '//'\}$ からなるラベルを持つ．
- $r_p, o_p \in V_p$ は，それぞれ p のルートノードと出力ノードである．

例えば，$q = a[//b/*]/e[//c]/d$ は，図 2(a) として表現される．図中の塗りつぶされたノードは，出力ノードに相当する．

$q \in XP^{\{//,*,[]\}}$ を，q に対応する TPQ p に変換する関数を P すると，$p = P(q)$ と書ける．いま，すべての表現可能な TPQ の集合を $P^{\{//,*,[]\}}$ とする．このとき，次の三つを $P^{\{//,*,[]\}}$ の部分集合として定義する．(1):分枝を持たない直線状の TPQ のみの集合である $P^{\{//,*\}}$，(2):ラベルに"$*$"を含まない TPQ の集合である $P^{\{//,[]\}}$，(3):エッジに"//"が存在しない TPQ の集合である $P^{\{*,[]\}}$．特に，これらの部分集合に対して，$XP^{\{//,*\}}$ は $P^{\{//,*\}}$ に，$XP^{\{//,[]\}}$ は $P^{\{//,[]\}}$ に，$XP^{\{*,[]\}}$ は $P^{\{*,[]\}}$ に，それぞれ変換できる．

任意のパターン $p < V_p, E_p, r_p, o_p >$ が与えられたとき，下記の条件を満たすような $p_s < V_s, E_s, r_s, o_s >$ を，r_s をルートノードとする部分パターン（**subpattern**）と呼び，$p_{sub(r_s)}$ と表す．

（1） $V_s \subseteqq V_p$

（2） $E_s = \{V_s \times V_s\} \cap E_p$

（3） $r_s \in V_s$ は p_s のルートノードである．

（4） $o_p \in V_p$ ならば，$o_s = o_p$ であり，かつ o_s は p_s の出力ノードである．

例えば，問合せ $q = a[//b/*]/e[//c]/d$ が与えられたときに，

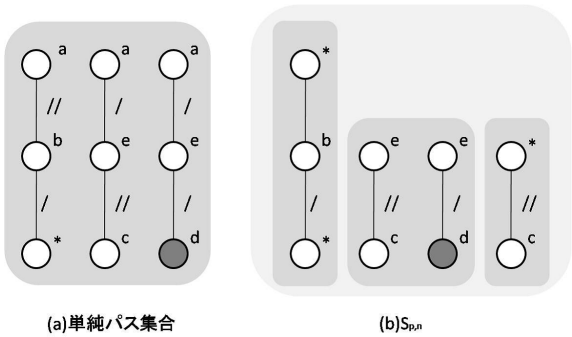

図 3 (a) 単純パス集合と (b)$S_{p,n}$

ノード n を図 2(a) における"e"のラベルのノードとすれば，$p_{sub(n)}$ は，図 2(b) となる．また，ノード m を図 2(a) における"b"のラベルのノードとすれば，$p_{sub(m)}$ は，図 2(c) となる．

TPQ p に対し，*leaf(p)* を，p の葉ノードの集合とする．TPQ p および $n \in leaf(p)$ が与えられたとき，r_p から n までの単純パスを**単純化された TPQ** と呼び，$p_{simple(n)}$ と表現する．このとき，$p_{simple(n)} \in P^{\{//,*\}}$ である．また，$p_{simple} = \{p_{simple(n)} | n \in leaf(p)\}$ を p の**単純パス集合**と呼ぶ．図 3(a) は，図 2(a) の TPQ の単純パス集合である．

XPath 式を非決定性有限オートマトン（NFA）に変換し，NFA を用いて XML 木からノードを得る手法は，文献 [8] [9] [10] などにより示されている．$XP^{\{//,*,[]\}}$ に属する任意の XPath 式と，TPQ は相互に変換可能である．そのため，TPQ の評価は，以下の手順で行う．

（1） TPQ p を単純パス集合 $p_{simple} = \{p_1, ..., p_n\}$ に変換する．

（2） 各 p_i を等価な NFA A_i に変換する．ただし，p_i が出力ノードを保持しない場合は，p_i のパスパターンと適合するノードが存在した場合は真を返し，そうでなければ偽を返すように，A_i を拡張する．

（3） $A_p = \{A_1, ..., A_n\}$ とする．ただし，k 個の NFA $A_1, ..., A_k$ は出力ノードを保持し，$A_{k+1}, ..., A_n$ は保持しないとする．このとき，$A_p(V)$ を以下のように定める．

$$A_p(V) = \{A_1(V) \cap ... \cap A_k(V) | A_{k+1}(V) = ... = A_n(V) = true\}$$

定義 5.（TPQ の分割関数）． *TPQ* $p < V_p, E_p, r_p, o_p > \in P^{\{//,*,[]\}}$，*XML* 木 $d < N, E, r >$ およびノード $v \in N$ が与えられたとき，*TPQ* の分割関数 S は，下記で定義される．

$$\begin{aligned} S(p,v) = \{(p_{sub(n)})_{single} \mid & (n.label = v.label) \\ & \vee (n.label =' *')\} \\ \cup \{(p_{sub(n')})_{single} \mid & \forall n \in V_p (\forall m \in V_p (\exists e \in E_p \\ & ((e.label =' //') \wedge (e = (n,m))) \\ & \wedge (p_{sub(n')} \equiv p_{sub(n)}) \\ & \wedge (n'.label =' *')))\} \end{aligned}$$

$S(p,v)$ は v がルートノードとしてマッチしうる TPQ の集合を返す．以降では，簡単のため，$S(p,n)$ を $S_{p,n}$ と記述する．図 3(b) は，図 2 の TPQ p，および XML 文書木 T 中のノード n（ただし $n.label =' e'$）に対し，$S_{p,n}$ によって得られる単純パス集合の集合を表したものである．

Algorithm 1 $S_{p,v}(V_1, V_2)$

1: TPQ p を NFA $A_p < S, \Sigma, T, s, A_> >$ に変換する．
2: $v.label \in \Sigma$ を，V_1 中に出現しない文字 z で置き換える．
3: 任意の $x \in S$ に対し，$T(x, v.label) = y$ という状態遷移が存在した場合，$T(x,z) = y$ を新たに付け加える．
4: $A_p(V_1)$ を評価する．ただし，評価の途中で z を読み込んだ場合，次の状態での入力は V_2 の各ノードとする．
5: 最終的に得られたノード集合を解集合として返す．

$S_{p,v} = \{P_1, P_2, ..., P_n\}$ と仮定する．また，$t_1 < V_1, E_1, r_1 >$，$t_2 < V_2, E_2, r_2 >$ を任意の文書木とする．このとき，$S_{p,v}(V_1, V_2)$ は，Algorithm 1 に示す方法で評価する．Algorithm 1 において，2 行目では v のラベル $v.label$ を別のラベル z に置き換える．また，3 行目では，z が入力されたときに，$v.label$ が入力されたときと同じ挙動をするように遷移関数を拡張する．その上で，4 行目では，z を読み込んだとき，入力を別のノード集合 V_2 に切り替える．

3. 問合せの等価変換規則

この章では，問合せ書き換えの際に有用であると思われる，文書軸とバージョン軸を入れ替えても成立すると考えられる等価な変換規則を列挙する．

3.1 XVerPath の問合せクラス

$XVP^{\{//,*,[],vpar\}}$ は，$XP^{\{//,*,[]\}}$ にバージョン親の軸 *vpar* を加えた問合せクラスである．すなわち，問合せ $q \in XVP^{\{//,*,[],vpar\}}$ はノードテスト，/，//，*，[]，および *vpar* 軸からなる．以下では，問合せクラスを表記する際は，*vpar* :↑，*vchild* :↓ という記号を用いる．例えば，$XVP^{\{//,*,[],vpar\}}$ は $XVP^{\{//,*,[],\uparrow\}}$ と表す．

3.2 変換規則

我々は，この節で，更新操作が delete，insert，update，replace のいずれかであるときに適用可能な規則を扱う．

以下に，XVerPath 問合せの書き換えにおいて有用であると思われ，かつ成立すると考えられる変換規則を挙げる．いま，V は同じバージョンに属するノード集合とし，U, W はそれぞれ V のバージョン親およびバージョン子のノード集合とする．また，t_0 は V の直前の更新操作 δ の対象となったノード，s_0 は V の直後の更新操作 δ の対象となったノードであり，x は更新操作の適用先のノードである．例えば，更新操作 δ が $delete(s_0)$ ならば，V から s_0 をルートノードとする部分木を削除して W が生成される，という意味である．また，δ が $insert(x, t_0)$ ならば，U に対し t_0 をルートノードとする部分木を x の子として挿入し V が生成される，という意味である．このとき，t_0 に対し，$t_0' \in V$ を t_0 に一致するノードとする．この様子を，図 4 に示す．さらに，$q \in XP^{\{//,*\}}$ を任意の問合せとし，$u_{vpar} \in \{vpar_{n,u}, vpar_{n,r,u}\}$，$u_{vchild} \in \{vchild_{n,u}, vchild_{n,r,u}\}$ とす

V:	同じバージョンに属するノード集合
U:	V のバージョン親のノード集合
W:	V のバージョン子のノード集合
t_0:	V の直前の更新操作の対象となったノード
s_0:	V の直後の更新操作の対象となったノード
x:	更新操作の適用先のノード
q:	$XP^{\{//,*\}}$ に属する任意の問合せ
P:	$XP^{\{//,*,[]\}}$ に属する問合せを対応する TPQ に変換する関数
p:	$p = P(q)$
u_{vpar}:	$u_{vpar} \in \{vpar_{n,u}, vpar_{n,r,u}\}$
u_{vchild}:	$u_{vchild} \in \{vchild_{n,u}, vchild_{n,r,u}\}$

表 1　規則中に出現する記号

る．上述した規則中に出現する記号をまとめ，これを表 1 に記す．

各規則は,「直前（あるいは直後）の更新操作名-バージョン軸-XPath の演算」という順の並びで表現される．例えば，DEL-VPAR-∗は，直前の更新操作が delete であるときに，vpar と wildcard(∗) が交換可能であるという規則である．

規則 1.　DEL-VPAR-∗

規則 2.　DEL-VPAR-//

規則 1 ならびに規則 2 の，前提条件および書き換え則は以下の通りである．

前提条件:U から $subtree(t_0)$ を削除し，V が生成される．

書き換え則:$(/q/vpar_n)(V) = (/vpar_n/q)(V) \setminus subtree(t_0)$

規則 3.　INS-VPAR-∗

規則 4.　INS-VPAR-//

規則 3 ならびに規則 4 の，前提条件および書き換え則は以下の通りである．

前提条件:$x \in U$ の子として $subtree(t_0)$ を挿入し，V が生成される．

書き換え則:$(/q/vpar_n)(V) = (/vpar_n/q)(V) \cup S_{p,x}(U, t_0)$

規則 5.　UPD-VPAR-∗

規則 6.　UPD-VPAR-//

規則 5 ならびに規則 6 の，前提条件および書き換え則は以下の通りである．

前提条件:U 中の t_0 を更新し，V が生成される．

書き換え則:$(/q/u_{vpar})(V) = (/u_{vpar}/q)(V)$

規則 7.　RPL-VPAR-∗

規則 8.　RPL-VPAR-//

規則 7 ならびに規則 8 の，前提条件および書き換え則は以下の通りである．

前提条件:U において $subtree(x)$ を $subtree(t_0)$ を置換し，V が生成される．

書き換え則:$(/q/vpar_{n,r})(V) = (/vpar_{n,r}/q)(V) \setminus (/*//*)(x)$

規則 9.　DEL-VCHILD-∗

規則 10.　DEL-VCHILD-//

規則 9 ならびに規則 10 の，前提条件および書き換え則は以下の通りである．

前提条件:V から $subtree(s_0)$ を削除し，W が生成される．

書き換え則:$(/q/vchild_n)(V) = (/vchild_n/q)(V)$

規則 11.　INS-VCHILD-∗

規則 12.　INS-VCHILD-//

規則 11 ならびに規則 12 の，前提条件および書き換え則は以下の通りである．

前提条件:V に $subtree(s_0)$ を挿入し，W が生成される．

書き換え則:$(/q/vchild_n)(V) = (/vchild_n/q)(V) \setminus subtree(s_0)$

規則 13.　UPD-VCHILD-∗

規則 14.　UPD-VCHILD-//

規則 13 ならびに規則 14 の，前提条件および書き換え則は以下の通りである．

前提条件:V 中の t_0 を更新し，W が生成される．

書き換え則:$(/q/u_{vchild})(V) = (/u_{vchild}/q)(V)$

規則 15.　RPL-VCHILD-∗

規則 16.　RPL-VCHILD-//

規則 15 ならびに規則 16 の，前提条件および書き換え則は以下の通りである．

前提条件:V において $subtree(x)$ を $subtree(s_0)$ を置換し，W が生成される．

書き換え則:$(/q/vchild_{n,r})(V) = (/vchild_{n,r}/q)(V) \setminus (/vchild_r/*//*)(x)$

4. 直前の操作が delete の場合に適用可能な規則

文書木 T_u から文書木 T_v への更新操作 δ が，$\delta = delete(t_0)$ の場合について議論する．この場合 T_u から t_0 をルートとする部分木が削除されることにより，新たな文書木 T_v が生成される．このとき，T_u と T_v の差異は $subtree(t_0)$ を含むか含まないかであるから，ある問合せ q で T_u を評価した場合と q で T_v を評価した場合の差異は，削除されたノード集合 $subtree(t_0)$ を含むか否かである．バージョン親を求める軸 vpar と XPath の問合せの書き換えは，$\tau \in subtree(t_0)$ が解集合に含まれているならば，τ を解集合から削除する操作を加えることで，τ を解集合から削除する操作を先に評価するように順序変更を行ってもよい．

規則 1. *DEL-VPAR-∗*

要素バージョングラフ G_e において，V は同じバージョンに属するノード集合とし，V の属するバージョンの直前の更新操作は $delete(t_0)$ とする．このとき，下記が成り立つ．

$$(/*/vpar_n)(V) = (/vpar_n/*)(V) \setminus subtree(t_0)$$

定理 1. *V が単一のノード v のみからなるとき，規則 1 は成り立つ．*

次に，定理 1 が，単一のノードではなく，ノード集合に拡張しても成り立つことを示すために，以下の定理を示す．

定理 2. *(分配則) V_1，V_2 をそれぞれ任意のノード集合とし，$q \in XVP^{\{//,*,[],\uparrow,\downarrow\}}$ を任意の問合せとする．このとき，以下が成り立つ．*

$$q(V_1 \cup V_2) = q(V_1) \cup q(V_2)$$

定理 3.（DEL-VPAR-∗）. 規則 *1* は成り立つ.

定理 3 は，規則 1 を単一のノードではなく，同じバージョンからなるノード集合に拡張しても成り立つことを示している．

DEL-VPAR-∗により，wildcard(∗) と vpar は交換可能である．delete で削除される部分木と vpar を適用させるノード集合が共通部分を持つ場合，補正項により共通部分を除くことで等価性が成り立つ．逆に共通部分がない場合は，$subtree(t_0)$ の項が不要になり，より明確な形で等価性が成り立つ．

規則 2. *DEL-VPAR-//*

要素バージョングラフ G_e において，V は同じバージョンに属するノード集合とし，V の属するバージョンの直前の更新操作は $delete(t_0)$ とする．また，$q \in XP^{\{//,*,[]\}}$ を任意の問合せとする．このとき，下記が成り立つ.

$$(//vpar_n/q)(V) = (/vpar_n//q)(V) \setminus subtree(t_0)$$

定理 4. V が単一のノード v のみからなるとき，規則 *2* は成り立つ.

定理 4 の証明は，定理 1 の証明と同様にして行う.

定理 5.（DEL-VPAR-//）. 規則 *2* は成り立つ.

DEL-VPAR-//により，//と vpar は交換可能である．DEL-VPAR-∗のときと同様に，補正項により共通部分を除くことで等価性が成り立つ．逆に共通部分がない場合は，より明確な形で等価性が成り立つ.

5. 直前の操作が insert の場合に適用可能な規則

文書木 T_u から文書木 T_v への更新操作 δ が $\delta = insert(x, t_0)$ の場合について議論する．このとき，ある問合せ q で T_u, T_v を評価した時の差異は，$subtree(t_0)$ を含むか含まないかのみである．更新操作が insert の場合は，$t_0^{'} \in T_v$ がバージョン親を持つ場合と持たない場合がある．$t_0^{'}$ がバージョン親を持つ場合は，他の文書木から部分木を挿入する場合である（図 4）．このときは，$vpar_n(t_0^{'}) = \{t_0\}$ が成り立っている．そのため，T_u のみではなく $q(subtree(t_0))$ の評価結果を加えることで，vpar と文書軸の評価順序を交換してもよい．一方，バージョン親を持たない場合は，新たな部分木を挿入する場合である．この場合は，$q(subtree(t_0))$ の評価結果を加える必要がないため，自明な形で評価順序の交換が可能である.

規則 3. *INS-VPAR-∗*

要素バージョングラフ G_e において，V は同じバージョンに属するノード集合とし，V の属するバージョンの直前の更新操作は $insert(x, t_0)$ とする．また，$q \in XP^{\{//,*,[]\}}$ を任意の問合せとする．さらに，p を $/q/*$ を TPQ に変換したものとする．このとき，下記が成り立つ.

$$(/q/*/vpar_n)(V) = (/q/vpar_n/*)(V) \cup S_{p,x}(U, t_0)$$

定理 6.（INS-VPAR-∗）. 規則 *3* は成り立つ.

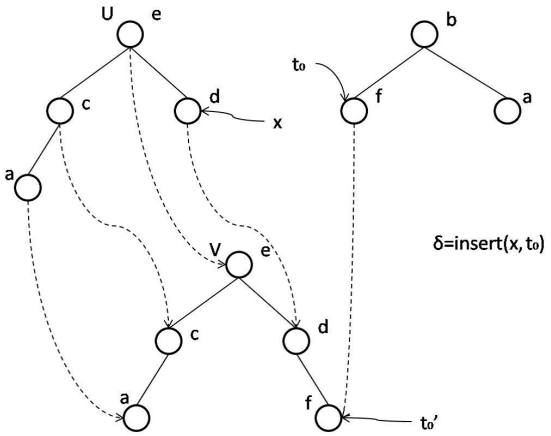

図 4　更新操作が *insert* のときに，$t_0^{'}$ がバージョン親を持つ様子

挿入されるノードがバージョン親を持たない場合は，INS-VPAR-∗により，自明な形で順序交換が可能である．一方，バージョン親を持つ場合は，補正項として $S_{p,x}(U, t_0)$ を加えることで，等価な順序交換が可能となる.

規則 4. *INS-VPAR-//*

要素バージョングラフ G_e において，V は同じバージョンに属するノード集合とし，V の属するバージョンの直前の更新操作は $insert(x, t_0)$ とする．また，$q_1, q_2 \in XP^{\{//,*,[]\}}$ を任意の問合せとする．さらに，p を $/q_1//*$ を TPQ に変換したものとする．このとき，下記が成り立つ.

$$(/q_1//vpar_n/q_2)(V) = (/q_1/vpar_n//q_2)(V) \cup S_{p,x}(U, t_0)$$

定理 7.（INS-VPAR-//）. 規則 *4* は成り立つ.

挿入されるノードがバージョン親を持たない場合は，INS-VPAR-//により，自明な形で順序交換が可能である．一方，バージョン親を持つ場合は，補正項として $S_{p,x}(U, t_0)$ を加えることで，等価な順序交換が可能となる.

例 1. 図 *1* において，次の二つの問合せ式を考える.

$$(//f/vpar_n)(v_1) \tag{1}$$
$$(/vpar_n//f)(v_1) \cup S_{p,v_2}(v_1, t_2) \tag{2}$$

(2) 式は，規則 *4* を用いて *(1)* 式を書き換えた問合せ式である．二つの問合せをそれぞれ評価すると，次のようになる.

$$(/a/*/vpar_n)(v_1) = \{t_2\}$$
$$(/vpar_n/a/*)(v_1) \setminus subtree(u_3) = \{t_2\}$$

これより，二つの問合せ式は等価であることが確認できる．また，下記の二つの問合せ式においては，複数の規則を用いて問合せを等価変換することが可能である.

$$(//*/vpar_n)(v_1) \tag{3}$$
$$(/vpar_n//*)(v_1) \cup S_{p,v_2}(v_1, t_2) \setminus subtree(u_3) \tag{4}$$

(4) 式は，規則 *1*，規則 *2*，規則 *3*，規則 *4* を用いて *(3)* 式を書

き換えた問合せ式である．二つの問合せをそれぞれ評価すると，次のようになる．

$$(//*/vpar_n)(v_1) = \{u_1, u_2, u_4, t_2\}$$
$$(/vpar_n//*)(v_1) \setminus subtree(u_3) = \{u_1, u_2, u_4, t_2\}$$

6. 直前の操作が update の場合に適用可能な規則

この章では，更新操作 δ が $\delta = update(x, c)$ により，文書木 T_u 中のノード x を，$content(x) = c$ となるように更新し，T_v が生成されたときに適用可能な規則について議論する．特に，3.2 節で取り上げた規則のうち，規則 5，規則 6 をそれぞれ下記のように形式化する．

規則 5. 要素バージョングラフ G_e において，V は同じバージョンに属するノード集合とし，V の属するバージョンの直前の更新操作は $update(x, c)$ とする．このとき，下記が成り立つ．

$$(/*/vpar_{n,u})(V) = (/vpar_{n,u}/*)(V)$$

規則 6. 要素バージョングラフ G_e において，V は同じバージョンに属するノード集合とし，V の属するバージョンの直前の更新操作は $update(x, c)$ とする．また，$q \in XP^{\{//,*,[]\}}$ を任意の問合せとする．このとき，下記が成り立つ．

$$(//vpar_{n,u}/q)(V) = (/vpar_{n,u}//q)(V)$$

規則 5 および規則 6 は，以下に示す操作を行うことにより，問合せを等価に変換していることが確認できる．(1): 各バージョン枝 $e \in G_e$ に対し，$e.label =$ "n" あるいは $e.label =$ "u" ならば，$e.label =$ "l" となるように置き換えることにより，$G_e^{'}$ を生成する．(2); 変換規則中に出現するバージョン軸 $vpar_{n,u}$ を，$vpar_l$ と読み替える．(3): $G_e^{'}$ において，2 で読み替えた変換規則に対し，等価性が成り立っているか確認する．

更新操作 δ が update のときは，更新操作適用前後で文書木の構造は変化しない．そのため，更新前の文書木に対して，任意の問合せ $q \in XP^{\{//,*\}}$ を適用して得られるノード集合は，更新後の文書木に対して q を適用して得られるノード集合のバージョン親と等価となる．

以上より，UPD-VPAR-*および UPD-VPAR-//により，自明な形で順序交換が可能である．

7. 直前の操作が replace の場合に適用可能な規則

この章では，更新操作 δ が $\delta = replace(x, t_0)$ により，文書木 T_u 中のノード x をルートとする部分木を，t_0 をルートとする部分木で置換し，T_v が生成されたときに適用可能な規則についてみていく．更新操作が replace の場合は，ある問合せ q で T_u および T_v を評価した時の差異は，置換されたノードを含むか否かである．特に，$subtree(x)$ が $subtree(t_0)$ で置換されるので，x を除く $subtree(x)$ のノード集合（図 5 における U_3）は解集合から取り除く必要がある．また，$t_0^{'}$ を除く $subtree(t_0^{'})$

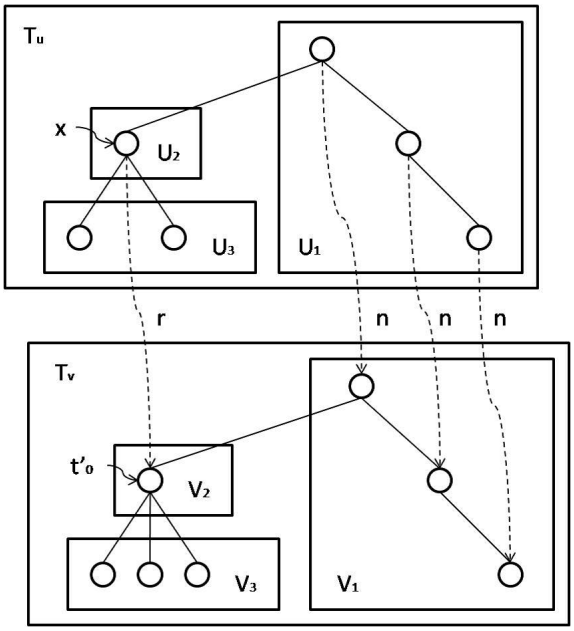

図 5　更新操作が *replace* のときにおける，T_u と T_v の関係

のノード集合（図 5 における V_3）はバージョン親を持たないため，insert におけるバージョン親がないノード集合と同様に扱うことができる．そのため，vpar と XPath 問合せは，U_3 に該当するノード集合を取り除く操作を加えることで，U_3 を解集合から削除する操作を先に評価するように，順序交換を行ってもよい．

規則 7. *RPL-VPAR-**

要素バージョングラフ G_e において，U, V は同じバージョンに属するノード集合とし，U の属するバージョンに対して更新操作 $replace(x, t_0)$ を行い，V のバージョンが生成されたとする．このとき，下記が成り立つ．

$$(/*/vpar_{n,r})(V) = (/vpar_{n,r}/*)(V) \setminus (/*//*)(x)$$

定理 8. (RPL-VPAR-*). 規則 *7* は成り立つ．

規則 8. *RPL-VPAR-//*

要素バージョングラフ G_e において，U, V は同じバージョンに属するノード集合とし，U の属するバージョンに対して更新操作 $replace(x, t_0)$ を行い，V のバージョンが生成されたとする．また，$q \in XP^{\{//,*\}}$ を任意の問合せとする．このとき，下記が成り立つ．

$$(//vpar_{n,r}/q)(V) = (/vpar_{n,r}//q)(V) \setminus (/*//*)(x)$$

定理 9. (RPL-VPAR-//). 規則 *8* は成り立つ．

定理 8，定理 9 により，vpar と*あるいは//は順序交換が可能である．特に，バージョン子を持たないノードが U 中に存在しない場合，自明な形で順序交換が可能となる．

8. 書き換え規則を用いた応用

この章では，本論文で得られた規則を応用して，問合せの非冗長化が可能なことを確認する．

例 2. 図 *1* に示す更新が成されて，最新バージョンとして d_4 の文書木が得られたと仮定する．このとき，二つの問合せ $q_1 = /a/vchild_{n,r}$，$q_2 = /*/vpar_{n,r,u}$ について考える．

いま，q_1 と q_2 を合成した問合せ $q_3 = /a/vchild_{n,r}/*/vpar_{n,r,u}$ を，図 *1* に示す文書木 d_3 に対して作用させたとする．

$$(/a/vchild_{n,r}/*/vpar_{n,r,u})(v_1) \tag{5}$$

ところで，q_3 は，規則 *1* を用いて，下記に示す等価な問合せに書き換えることが可能である．

$$(/a/vchild_{n,r}/vpar_{n,r,u}/*)(v_1) \setminus subtree(u_3) \tag{6}$$

ここで，*(6)* 式において，$vchild_{n,r}/vpar_{n,r,u}$ の部分問合せを含まない問合せ *(7)* でも，等価な解集合を得られる．

$$(/a/*)(v_1) \setminus subtree(u_3) \tag{7}$$

このように，*(5)* 式を *(7)* 式に書き換えることが可能である．

9. 結　　論

本論文では，バージョン化された XML 文書に対する問合せ言語 XVerPath において，文書軸とバージョン軸とで交換可能な組み合わせを列挙した．また，更新操作が文書木の削除，挿入，更新ならびに置換の場合に対し，そのときにバージョン軸 vpar と XPath の演算が交換可能である規則が等価な書き換えであることを示した．これらの規則は，バージョン軸で拡張した XPath において，二つの問合せの包含関係の判定に応用することが可能である．さらなる応用として，包含関係の判定を応用し，問合せを書き換えることが挙げられる．例えば，複数の問合せが与えられ，それらを合成する際に，冗長な部分を削除し合成した問合せを最小化すること，効率の良い問合せに書き換えることなどが考えられる．

今後の課題として，本論文で挙げた規則の中で，バージョン軸 vchild と XPath の演算とを交換することが可能だと予想される規則について，それらの証明を試みることが挙げられる．また，Tree Pattern Query の分割関数を評価する際に，一度 Tree Pattern Query を決定性有限オートマトンに変換し，効率よく評価するように最適化することが挙げられる．さらに，本論文で示した規則を用いて，実際にバージョン軸で拡張された XPath の包含関係の判定を行うことも重要な課題である．

文　　献

[1] Wikipedia. http://en.wikipedia.org/wiki.

[2] Subversion. http://subversion.tigris.org/.

[3] W3C: “Xml path language(xpath) version 1.0” (1999). http://www.w3.org/TR/xpath/.

[4] W3C: “Xml path language(xpath) version 2.0” (2007). http://www.w3.org/TR/xpath20/.

[5] M. IWAIHARA, R. HAYASHI, S. CHATVICHIENCHAI, C. ANUTARIYA and V. WUWONGSE: “Relevancy-based access control and its evaluation on versioned xml documents.”, ACM Transactions on Information and System Security, **10**, 1, pp. 1–31 (2007).

[6] T. Milo and D. SUCIU: “Index structures for xpath expressions”, ICDT, pp. 277–295 (1999).

[7] G. MIKLAU and D. SUCIU: “Containment and equivalence for a fragment of xpath.”, Journal of the ACM, **51**, pp. 2–45 (2004).

[8] T. J. Green, G. Miklau, M. Onizuka and D. Suciu: “Processing xml streams with deterministic automata”, ICDT, pp. 173–189 (2003).

[9] T. J. Green, A. Gupta, G. Miklau, M. Onizuka and D. Suciu: “Processing xml streams with deterministic automata and stream indexes”, ACM TODS, **29**, pp. 752–788 (2004).

[10] A. K. Gupta and D. Suciu: “Stream processing of xpath queries with predicates”, ACM SIGMOD, pp. 419–430 (2003).