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

2006－FI－82（3）
2006－DD－54（3）
2006／3／22

# XMLストリームに対する時制問合せの一手法

浜野泰男[†]　宮崎　純[†]
中島伸介[†]　植村俊亮[†]

本論文では，XPath で指定した XML 部分文書に対してイベント代数の演算を適用することにより，XML ストリームに対する時制問合せを行う手法を提案する．

情報交換のフォーマットとして XML の利用が拡大している．中でも連続的に送信される時系列な XML 文書が急増している．そのような XML 文書を XML ストリームと呼ぶ．XML ストリームに対する問合せ言語には XPath や XQuery がある．しかし，これらは XML の部分文書を検索するための言語であり，時間の概念を含んだ問合せは表現できない．

そこで本論文では上記のような問合せを実現するために，XPath で指定した XML 部分文書に対して，イベント代数 Snoop の演算を適用することにより，XML ストリームに対する時制問合せを行う手法を提案する．また，提案する時制問合せの処理をする有限状態変換器の構築方法を示す．

本手法により，XML ストリームデータをシーケンシャルに一度だけ読み込むことで時制問合せが可能になる．

# A Method of Temporal Queries for XML Streams

Yasuo HAMANO,[†] Jun MIYAZAKI,[†] Shinsuke NAKAJIMA[†]
and Shunsuke UEMURA[†]

In this paper, we propose a method of temporal queries for XML streams using XPath results as primitive events in an event specification language. XML is widely used as data format for information exchange. In particular, continuously arriving XML data is rapidly increasing. We refer to such XML data as an XML stream. There are demands on queries with aspects of time for an XML stream. However existent query languages such as XPath or XQuery can only retrieve part of the information in an XML document. To express such query with aspects of time, in this paper, we show how to generate a finite state pushdown transducer which can process the temporal queries, where Snoop to process event detection and XSQ to process XPath queries are used. The transducer can process temporal queries over an XML stream reading the data sequentially only once.

## 1. はじめに

XML(Extensible Markup Language) [7] は企業間取引など，Web 上で使われる情報の標準交換方式として広く普及している．中でも XML ストリームと呼ばれるストリーム指向の XML 文書が増加している．一般的に XML ストリームとは送信側と受信側で順序が一意で，一度読んだデータを再読することができない XML 文書のことを言う．これに加えて本論文で扱う XML ストリームには，データの生起順に XML 文書が生成されるという制約を加える．XML ストリームの例として，株価情報を表す XML 文書を図 1 に示す．

このような XML ストリームに対して，時間の概念を含んだ問合せの要求がある．例えば図 1 の XML 文書に対して，“A 社の株価が上がってから，B 社の株価が下がるまでの C 社のすべての株価を取得する” という問合せ要求が考えられる．これを XPath で表すと例えば次のようになる．初めに

```
//stock[@name = "A 社"]/info[diff>0]/time
//stock[@name = "B 社"]/info[diff<0]/time
```

の二つの問合せによって，A 社の株価が上がった時間と B 社の株価が下がった時間を求めてから，その区間のデータに対して

```
//stock[company = "C 社"]/info/price
```

のように問い合わせることになる．XPath ではこのような二段階で問合せをする必要が生じる．ストリームデータの場合，元に戻ってデータを読み直すことができない可能性があるので，二段階の問合せは一般に許容できない．

本論文では上記のような問合せが可能な，XML ストリームに対する時制問合せを行う手法を提案する．提案手法では，XPath で指定した XML 部分文書に対してイベント代数の演算を適用することにより，時間

† 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, NAIST

```
<root>
<stocklist>
  <stock name="A 社">
    <info>
      <time>1020</time>
      <price >8130</price>
      <diff>2</diff>
    </info>
  </stock>
  ...
  <stock name="B 社">
    <info>
      <time>1800</time>
      <price>4880</price>
      <diff>-3</diff>
    </info>
  </stock>
  ...
</stocklist>
</root>
```

**図 1** 株価情報の XML ストリームの例

の概念を含む問合せを行う．これによって，XPath よりも表現力の高い問合せが可能になる．また提案する時制問合せの処理のために，XML ストリームを先頭から逐次に一度だけ読むことで結果を返すことのできる単一の有限状態変換器 (以降，単に変換器と呼ぶ) の構築方法を示す．単一の変換器で問合せ処理を行うことには，変換器の状態数が削減でき，処理の高速化が期待できる．提案する時制問合せでは，イベント代数を適用するイベントの単位として，与えられた XPath にマッチする XML の部分文書を要求する．したがって，提案する時制問合せ処理器の XPath 評価部分は，XML の部分文書を返すことのできる処理手法が必須である．

本論文の構成を以下に示す．第 2 節では本研究に関連する研究を紹介する．第 3 節では本研究で利用するストリーム指向 XPath 問合せ処理器の XSQ と，イベント代数の Snoop について詳しく述べる．第 4 節では提案手法である時制問合せの表現方法と，その処理を行なう変換器の構築アルゴリズムを示す．最後に第 5 節で本論文をまとめ，今後の課題について述べる．

## 2. 関 連 研 究

XPath とイベント代数を組み合わせた問合せの研究は，我々の知る限りなされていない．しかし，XML ストリームに対する XPath 問合せ手法は数多く研究されている．既存の手法は大きく二種類に分けられる．

一つはフィルタリングとしての XPath 処理手法であり，YFilter [2] や XPush Machine [5] が挙げられる．これらは，多数の XML 文書に対して複数の XPath 問合せを条件として設定し，それにマッチした XML 文書全体を結果として返すものであり，XML 部分文書を扱うことはできない．よって本研究での XPath 処理部分に利用できない．

もう一方の手法は，従来の XPath 処理と同じように，XPath 式にマッチした XML 部分文書を結果として返すものであり，XSQ [6] や XMatch [8] が挙げられる．XSQ は，階層化されたバッファ付きプッシュダウン変換器によって XPath 式を評価する．XMatch はパスオートマトンと制御スタックを用いて問合せ式を検出する．

イベント代数とは，原始イベントと呼ぶ不可分なイベントの列から，それらをある演算子で組み合わせた複合イベントを表現するための代数である．イベント代数はアクティブデータベースでのイベント記述や，蓄積されたイベントログからの知識発見などに用いられている．イベント代数の既存手法には Snoop [1], Ode [4], Samos [3] などがある．Snoop は豊富な演算子と，parameter context と呼ばれる状態削減手法を組み合わせて，複合イベントを効率よく検出できる．Ode や Samos は検出できる複合イベントが Snoop より少なく，表現能力の点で劣る．

## 3. XPath 処理器とイベント代数

本研究では，構造が単純な XML ストリーム指向の XPath 処理器 XSQ と，複合イベントの表現能力に優れる Snoop を利用して，XML ストリームの時制問合せを実現する．

### 3.1 XSQ

XSQ [6] はストリーム指向の XPath 処理器であり，複数述語，“//”，集約関数に対応しており，他手法と比べて表現できる XPath の範囲が広い．

XSQ で処理する XML ストリームのデータモデルは，深さ情報を追加した SAX イベント列として (tag, attrs, type, depth) の四項組で表される．ただし，

- tag: タグ名．
- attrs: (a, v) で表される属性のリスト．a は属性名，v は属性値を表す．属性がないときは NULL になる．
- type: B(開始イベント), E(終了イベント), T(テキストイベント)．type が T のとき，attrs は一つの (text(),v) の組を持つ．v はテキストノードの内容である．
- depth: XML 木中の要素の深さ．属性ノードとテキストノードは親ノードと同じ深さを持つ．

図 1 の XML ストリームの最初の 5 行は図 2 のように表される．

XSQ で対応する XPath は，XPath 1.0 から reverse 軸 (preceding 軸など) と position 関数 (last() など) を除いたものである．図 3 に，XSQ が対応する XPath の文法を EBNF 表記で示す．

$N_i$ はロケーションステップ，$O$ は出力関数としたと

(root, $\phi$, B, 0)
(stocklist, $\phi$, B, 1)
(stock, name, “A 社”, B, 1)
(info, $\phi$, B, 2)
(time, $\phi$, B, 3)
(time, (text(), “1020”), T, 3)
(time, $\phi$, E, 3)

**図 2** 図 1 の XML ストリームの SAX イベント列

$Q ::= N + [/O]$
$N ::= [/|//]tag[F]$
$F ::= @attribute|tag[@attribute]|text()$
$O ::= @attribute|text()|count()|sum()$
$OP ::=> | \geq | = | < | \leq | \neq |contains$

**図 3** XSQ で使われる XPath の EBNF 表記

き，XPath を形式的に $N_1N_2\cdots N_n/O$ と表す．

XPath のロケーションステップは，バッファ付きプッシュダウン変換器 (BPDT) で処理される．バッファはキューとして定義され，述語の評価の時に，結果の候補になる要素をバッファリングするのに使われる．BPDT は述語評価が偽である NA 状態と，真である TRUE 状態を持つ．バッファ操作は次の四つが定義されている．

(1) enqueue(v): v をキューの最後尾に加える．
(2) clear(): キュー中のすべての要素を削除する．
(3) flush(): キュー中のすべての要素を，FIFO 順に出力する．
(4) upload(): キュー中のすべての要素を，HPDT 中での親 BPDT のバッファの最後尾に送る．HPDT は BPDT を二分木の構造で階層化したものである．

ロケーションステップから BPDT を生成するとき，その種類によって次の五種類のテンプレートが適用される．

- /tag[@attr op val], /tag[@attr] の形で表されるロケーションステップ．
- /tag[text() op val] の形で表されるロケーションステップ．
- /tag[child@attr op val], /tag[child@attr] の形で表されるロケーションステップ．
- /tag[child] の形で表されるロケーションステップ．
- /tag[child op val] の形で表されるロケーションステップ．

任意の問合せはこれらのテンプレートの組み合わせで表される．一つの XPath 全体は HPDT で表される．

HPDT は次のアルゴリズムにより構成される．

- root の BPDT をつくり，ID を bpdt(0,0) とする．
- それぞれの bpdt(i-1, k) に対して
  - NA 状態を持っていれば，bpdt(i, 2k) を右の子供として生成する．bpdt(i-1, k) の NA 状態を初期状態とする．NA 状態がなければ，bpdt(i, 2k) を NULL とする．
  - bpdt(i, 2k+1) を bpdt(i, 2k) の左の子供として生成する．bpdt(i-1,k) の TRUE 状態を初期状態とする．

bpdt(n, $2^n-1$) の出力関数を output() とする．output() は値をバッファにためずに直接出力する．bpdt(n, $2^n-1$) では全ての述語を満たしているため，すぐにバッファ中のデータが出力される．

ここで上記のアルゴリズムの初期状態を満たすために，XML 文書の最上位階層に述語を含まない root 要素が必要である．HPDT は，BPDT の位置によって述語の真偽の判断をする．HPDT の状態が bpdt(i, 2k) にあれば，i 番目の述語の評価は真である．

HPDT の状態は，(i, d) の二項組で表される．i は状態の識別番号，d は深さスタックである．例えば状態 (\$3(013))) は，状態番号が \$3 で深さスタックが先頭から 0,1,3 と入っている状態を表す．深さスタックは同じ識別番号の状態 i に至る複数のパスのマッチを区別する．深さスタックに対する操作は，通常のスタック操作 (push, pop, peek) がある．それに加えて，スタックの上位 k 個の要素を取り除く remove(k) が定義される．

HPDT の状態遷移は次の 4 種類に分けられる．ここで，入力記号を e，遷移元の状態を q = (i, d) 遷移後の状態を q’ = (i’, d’) とする．

- self-closure 遷移: e.type = B かつ e.depth > d.peek() のとき，自己遷移をする．つまり q’ = q となる．状態遷移図では “//” と表される．
- closure 遷移: e.type = B かつ e.depth > d.peek() のときに状態遷移する．深さスタックは d’ = d.push(e.depth) になる．遷移元の状態 q は活性状態集合に留まり，かつ新しい活性状態 q’ が追加される．状態遷移図では “=” で飾られた矢印で表される．
- regular 遷移: 入力記号 e が
$$e.depth = \begin{cases} d.peek()+1 & (e.type = B \text{ のとき}) \\ d.peek() & (e.type = T \text{ または } E \text{ のとき}) \end{cases}$$
の条件を満たすとき，regular 遷移が行われる．活性状態集合から q が削除され，新しい状態 q’ が追加される．q’ の深さスタック d’ は次のようになる．
$$d' = \begin{cases} d.push(e.depth) & (e.type = B \text{ のとき}) \\ d & (e.type = T \text{ のとき}) \\ d.pop() & (e.type = E \text{ のとき}) \end{cases}$$
状態遷移図では，通常の矢印で表される．
- catch-all 遷移: 入力記号 e が次の条件を満たすときに catch-all 遷移が行なわれる．
$$\begin{cases} e.depth > d.peek() & (e.type = B \text{ または } E \text{ のとき}) \\ e.depth \geq d.peek() & (e.type = T \text{ のとき}) \end{cases}$$

新しい状態は q' = q となる．つまり元の状態にとどまる．状態遷移図では $*$ と表される．

### 3.2 Snoop

Snoop は多くの種類の複合イベントを表現できるイベント代数であり，AND，OR，SEQ，ANY，NOT，A，P の七つの演算子が定義されている．このうち，以降の議論で用いる三つの演算子のみを解説する．それ以外の演算子の意味論については，文献 [1] を参照されたい．

- AND ($\triangle$): イベント $E_1$，$E_2$ の論理積を取り，$(E_1 \triangle E_2)$ と表す．$E_1$ と $E_2$ の両方が生起したときに検出される．
- OR ($\bigtriangledown$): イベント $E_1$，$E_2$ の論理和を取り，$(E_1 \bigtriangledown E_2)$ と表す．$E_1$ と $E_2$ のどちらかが生起したときに検出される．
- SEQ (;): イベント $E_1$ が生起した後に $E_2$ が生起したとき $(E_1 ; E_2)$ が検出される．

ただし $E_i$ はイベントの種類を表すもの (イベントタイプと呼ぶ) であり，そのインスタンスが原始イベント $e_i^k$ である．$k$ は原始イベントの生起時間順を表す．複合イベントの検出を開始するイベントを initiator，検出を終わらせるイベントを terminator という．複合イベントの生起時刻は terminator の生起時刻とする．

次に，上述の演算子を用いて表される複合イベント X の，原始イベント列 H に対する検出例を示す．

$X = ((E_1; E_2) \triangle E_3)$
$H = \{\{e_1^1\}, \{e_1^2\}, \{e_2^1\}, \{e_3^1\}, \{e_2^2\}\}$
$(E_1)[H] = \{\{e_1^1\}, \{e_1^2\}\}$
$(E_2)[H] = \{\{e_2^1\}, \{e_2^2\}\}$
$(E_3)[H] = \{\{e_3^1\}\}$
$(E_1; E_2)[H] = \{\{e_1^1, e_2^1\}, \{e_1^1, e_2^2\}, \{e_1^2, e_2^1\}, \{e_1^2, e_2^2\}\}$
$X[H] = \{\{e_1^1, e_2^1, e_3^1\}, \{e_1^2, e_2^1, e_3^1\},$
$\{e_1^1, e_3^1, e_2^2\}, \{e_1^2, e_3^1, e_2^2\}\}$

イベント演算子の定義をそのまま適用すると，複合イベントの検出が行われる領域によっては多くのイベントを検出しすぎてしまう．そこで，複合イベントの検出が行われる領域に応じてイベントの検出方法を変える，Parameter context が導入されている．Parameter context は Recent, Chronicle, Continuous, Cumulative の四つが定義されているが，ここでは検出が単純な Recent についてのみに着目する．Recent context は最新の initiator のみがイベントを検出することができ，古いイベントがあまり意味をなさないアプリケーションで特に有効である．例えば気象情報のイベント列に対して，何日も前の気温が必要ないようなアプリケーションの場合に有効である．他の Parameter context の詳細については，文献 [1] を参照されたい．

Snoop において複合イベントはイベント木という木構造のモデルによって検出される．原始イベントが葉から入力されていき，それぞれのノードで演算子の評価がされる．根まで伝播した原始イベント列が複合イベントとして検出される．上記の複合イベント X に対する Recent context での検出過程を図 4 に示す．図では $e_3^1$ が生起した時点のイベント木の状態を表している．

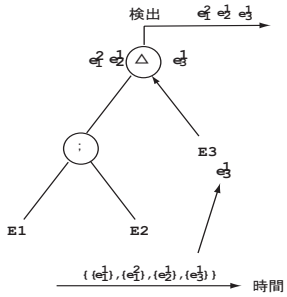


**図 4** recent context でのイベント X の検出

以降の時制問合せの議論では，簡単のため Snoop の演算子は高々一つ含まれると仮定する．

## 4. XML ストリームに対する時制問合せ手法

本節では，提案する時制問合せの表記方法および問合せを処理する変換器の構成方法について述べる．

### 4.1 提案する時制問合せの表記方法

時制問合せ式として，Snoop のイベント演算子のオペランドに XPath 式を指定することで表記するものとする．例として，AND，OR，SEQ 演算子のみの表記を以下に示すが，他の演算子についても同様である．

- $(Q_1 \triangle Q_2)$
- $(Q_1 \bigtriangledown Q_2)$
- $(Q_1; Q_2)$

ただし，$Q_i$ は XPath 式である．扱える XPath 式は XSQ で扱えるもの (図 3) と同じである．これらの問合せは，XPath 式の結果として返される XML 部分文書を，原始イベントとして扱う．

本手法での XML ストリームのデータモデルは，XSQ での XML ストリームのモデルに時間情報を追加した (tag, attrs, type, depth, time) の五項組で表す．tag, attrs, type, depth の意味は，XSQ でのモデルと同様である．time は XML ストリームの生起した時刻を表す文字列である．time は 絶対時間を必要とする P 演算子の評価に利用される．

### 4.2 複合イベントを処理する変換器の構成

本節では Snoop の複合イベントを検出する変換器を，各演算子に対して構築する方法を示す．Snoop では，複合イベントの検出にイベント木を用いている．提案する時制問合せの処理を単一の変換器で行うために，その部品になるイベント検出部分も変換器で行う．各変換器は一つのバッファを持つ．バッファ操作は次の四つを定義する．

- append (E): E のインスタンスをバッファの最後尾に加える．

- delete ($E_1$, $E_2$, ⋯): バッファ中の $E_1$, $E_2$, ⋯ のインスタンスを削除する．delete() と書くときは，バッファ中のすべてのインスタンスを削除する．
- replace (E): delete(E) に続けて append(E) を行う．
- output (m): バッファ中の最後尾 m 個のインスタンスを出力する．output() と書くときは，バッファ中のすべてのインスタンスを出力する．出力後もインスタンスはバッファ内に残る．

構築した変換器を図 5 から図 7 の状態遷移図で示す．紙面の都合上 SEQ, OR, AND 演算子の変換器のみを示す．他の Snoop で定義されている演算子も容易に変換器で表現可能である．

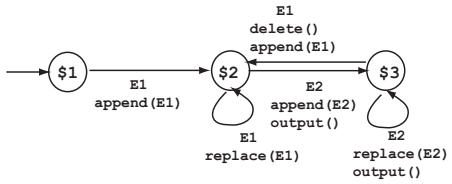


**図 5** ($E_1; E_2$) を処理する変換器

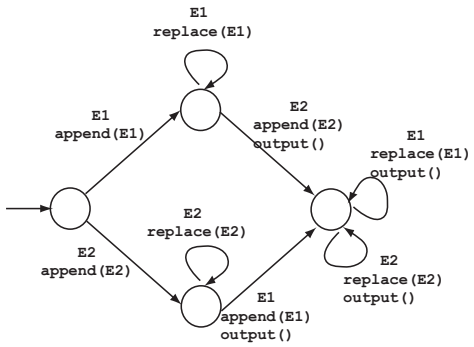


**図 6** ($E_1 \triangle E_2$) を処理する変換器

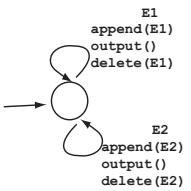


**図 7** ($E_1 \bigtriangledown E_2$) を処理する変換器

状態遷移図中で明記した入力記号以外を読んだ場合，その状態に留まりバッファ操作も状態遷移も行なわない．これは，通常の有限状態機械の動作と異なり，注意が必要である．

図 5 の SEQ 演算子を例に変換器の動作を説明する．この変換器に，入力記号列 $\{e_1^1, e_1^2, e_2^1, e_1^3, e_2^2\}$ を入力したときの動作を表 1 に示す．まず初期状態 \$1 から動作を始める．initiator である $e_1^1$ が入力されると $e_1^1$ をバッファに追加し，状態 \$2 に遷移する．状態 \$2 はバッファに $E_1$ のインスタンスだけが入っている状態を表す．状態 \$2 で $e_1^2$ を読むと replace() 操作によって，すでに入っている $e_1^1$ と置き換えられる．Recent context であるため，古い initiator は捨てられる．状態 \$2 で $e_2^1$ のインスタンスを読むと，そのインスタンスをバッファに加えた後，バッファ中のアイテムを出力する．状態 \$3 は $E_1, E_2$ のインスタンスが順にバッファに入っている状態を表す．バッファ中の四角で囲ったものは，output() 操作によって出力されるアイテムを表す．

**表 1** 図 5 の状態遷移の例

| 入力記号 | 活性状態 | バッファ |
|---|---|---|
| $e_1^1$ | \$1→\$2 | $\{e_1^1\}$ |
| $e_1^2$ | \$2→\$2 | $\{e_1^2\}$ |
| $e_2^1$ | \$2→\$3 | $\boxed{\{e_1^2, e_2^1\}}$ |
| $e_1^3$ | \$3→\$2 | $\{e_1^3, e_2^1\}$ |
| $e_2^2$ | \$2→\$3 | $\boxed{\{e_1^3, e_2^2\}}$ |

### 4.3 複数の XPath を処理する変換器の構成

提案する時制問合せを処理するためには，前節で構築した Snoop の演算子を処理する変換器と，XPath 式を検出する変換器とを合成する必要がある．その際，Snoop は複数の XPath 式をオペランドとしてとるため，一つの変換器で全てを処理するためには，複数の XPath 式を並行に検出する必要がある．しかし XSQ で用いられる HPDT は，単一の XPath 問合せのみを処理する．そこで本節では，任意の $n$ 個の XPath 式を並行して処理する単一の HPDT を合成する方法を示す．

**準備:** 合成する $n$ 個の XPath 式を $Q_1, Q_2, \cdots, Q_n$ とする．各 XPath 式 $Q_i$ は $N_1^i N_2^i \cdots N_m^i / O^i$ と表される．各 XPath 式に対応する HPDT を $H_1, H_2, \cdots, H_n$ とする．合成後の変換器を $H_c$ とする．並行して処理する XPath 式 に関して，XML 文書中の同じ部分を返す XPath 式は，本論文では考えないものとする．なぜなら，Snoop の演算子は，同じタイミングで複数のイベントが発生するときの動作を定義していないためである．例えば，/a/b と //a/b の合成は考えない．

**ステップ 1 (共通接頭辞の決定):** $n$ 個の XPath 式に対して共通接頭辞を決定する．共通接頭辞とは，各 XPath 式の先頭から共通であるロケーションステップである．例えば /a/b[c]/d, /a/b[c]/e, /a/b[c] の共通接頭辞は /a/b[c] である．決定した共通接頭辞を $N_1^0 \cdots N_k^0$ とする．

**ステップ 2 (共通接頭辞に対応する状態の共有):** 共通接頭辞に対応する変換器を構築する．XSQ の HPDT

構築アルゴリズムに従い，共通接頭辞 $N_1^0 \cdots N_k^0$ の変換器を構築する．このとき，各々の変換器 $H_i$ のうち，共通接頭辞 $N_1^0 \cdots N_k^0$ 中の各ロケーションステップ $N_i^0$ を処理する BPDT を考える．この $N_i^0$ を処理する BPDT 中の各状態遷移に属するバッファ操作は，合成した変換器 $H_c$ 中の $N_i^0$ を処理する BPDT に継承する必要がある．従って，これらのバッファ操作を合成した換器 $H_c$ 中の対応する状態遷移に加える．

**ステップ 3 (共通接頭辞以降の変換器の連結):** 共通接頭辞以外のロケーションステップに対応する変換器を，各 XPath ごとに独立に構築する．共通接頭辞 $N_1^0 \cdots N_k^0$ 中の述語の数を $p$ とするとき，$N_k^0$ に対応する BPDT には $2^p$ 個の NA 状態または TRUE 状態が存在する．これらのそれぞれの状態に対して，共通接頭辞以降のロケーションステップに対する BPDT を連結していく．

以上の手順で合成した変換器において，“//” 以外で非決定的な動作をする場合が二つある．一つは複数の $Q_i$ において，$N_{k+1}^i$ のロケーションステップで，同じ子要素を検出する場合である．例えば $Q_1$:/a/b/c と $Q_2$:/a/b[c]/d は，共通接頭辞が $N_1^0 =$ /a であり，$N_2^1$ と $N_2^2$ がともに b という子要素を検出する．このとき，b という入力記号での遷移先が二つ存在する．このときは，3.1 節で説明した XSQ の “//” のときの動作のように，活性状態を増やして，それぞれの状態ごとに XPath 検出を行う．もう一つは，$n$ 個の XPath 式中で，$N_1^0 \cdots N_k^0$ で表されるものがある場合，つまり共通接頭辞そのままの XPath 式 が存在する場合である．このとき，$N_k^0$ に対応する BPDT の TRUE 状態と NA 状態には，catch-all 遷移がある．例えば $Q_1$:/a/b と $Q_2$:/a/b/c を合成する際，/b に対応する BPDT の TRUE 状態において $Q_1$ の catch-all 遷移と，$Q_2$c の開始イベントでの遷移が存在する．この TRUE 状態で c の開始イベントが入力された場合，$Q_1$ の catch-all 遷移と，$Q_2$c の開始イベントでの遷移を非決定的に行う．

次に $Q_1$:/a/b, $Q_2$:/a/c の場合を例にとり，HPDT の合成とその動作を示す．$Q_1$ を処理する HPDT は図 8，$Q_2$ を処理する HPDT は図 9 のようになる．合成後の変換器は図 10 のようになる．$output_i()$, $enqueue_i()$ はそれぞれ，$Q_i$ の変換器に対応するバッファ操作を表す．

まず $Q_1$ と $Q_2$ の共通接頭辞を /a と決定する．次に，共通接頭辞に対応する部分の変換器を共有する．図 10 でいうと $11, $22, $33 を共有化している．次に，共通接頭辞以降の変換器は $33 から枝分かれし，$Q_1$ と $Q_2$ それぞれに対する BPDT がつくられる．

図 10 の変換器に対して図 11 の XML ストリームを入力したときの動作の詳細を，表 2 に示す．表で開始イベントは <tag>，終了イベントは </tag>，テキストイベントはタグで囲まない文字列として表す．tag はタグ名である．表 2 の 5, 8 行目でそれぞれ，$H_1$ に対応するバッファの $output_1()$ 操作が行われる．11, 14 行目ではそれぞれ，$H_2$ に対応するバッファの $output_2()$ 操作が行われる．最終的にこの変換器の出力結果は `{{<b>B1</b>}, {<b>B2</b>}, {<c>C1</c>}, {<c>C2</c>}}` となる．

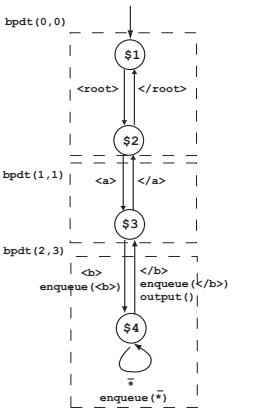


**図 8** /a/b を処理する HPDT

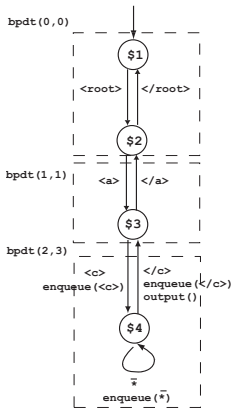


**図 9** /a/c を処理する HPDT

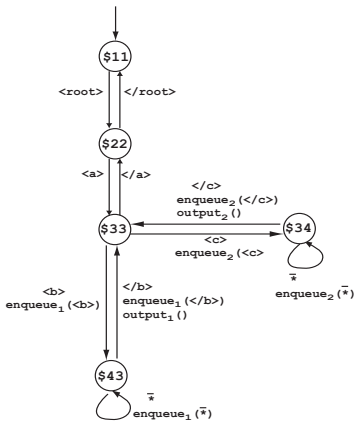


**図 10** /a/b と /a/c を並行に処理する HPDT

```
<root>
<a>
  <b>B1</b>
  <b>B2</b>
  <c>C1</c>
  <c>C2</c>
</a>
</root>
```

**図 11** XML ストリームの例

### 4.4 提案する時制問合せを処理する変換器の構成

本節では Snoop の演算子を処理する変換器 S と，XPath 式を並行処理する変換器 $H_c$ を合成し，提案する時制問合せを処理する変換器 T を構築する方法を示す．

変換器 T は以下の手順で合成する．

**表 2** /a/b と /a/c の並行処理をする変換器の動作

| 行 | イベント | 深さ | 活性状態 | $H_1$ のバッファ | $H_2$ のバッファ |
|---|---|---|---|---|---|
| 1 | <root> | 0 | ($1()$1()) → ($2(0)$2(0)) | | |
| 2 | <a> | 1 | ($2(0)$2(0)) → ($3(01)$3(01)) | | |
| 3 |  | 2 | ($3(01)$3(01)) → ($4(012)$3(01)) | () | |
| 4 | B1 | 2 | ($4(012)$3(01)) → ($4(012)$3(01)) | (B1) | |
| 5 |  | 2 | ($4(012)$3(01)) → ($3(01)$3(01)) | (B1) | |
| 6 |  | 2 | ($3(01)$3(01)) → ($4(012)$3(01)) | () | |
| 7 | B2 | 2 | ($4(012)$3(01)) → ($4(012)$3(01)) | (B2) | |
| 8 |  | 2 | ($4(012)$3(01)) → ($3(01)$3(01)) | (B2) | |
| 9 | <c> | 2 | ($3(01)$3(01)) → ($3(01)$4(012)) | | (<c>) |
| 10 | C1 | 2 | ($3(01)$4(012)) → ($3(01)$4(012)) | | (<c>C1) |
| 11 | </c> | 2 | ($3(01)$4(012)) → ($3(01)$3(01)) | | (<c>C1</c>) |
| 12 | <c> | 2 | ($3(01)$3(01)) → ($3(01)$4(012)) | | (<c>) |
| 13 | C2 | 2 | ($3(01)$4(012)) → ($3(01)$4(012)) | | (<c>C2) |
| 14 | </c> | 2 | ($3(01)$4(012)) → ($3(01)$3(01)) | | (<c>C2</c>) |
| 15 | <a> | 1 | ($3(01)$3(01)) → ($2(0)$2(0)) | | |
| 16 | </root> | 0 | ($2(0)$2(0)) → ($1()$1()) | | |

- 変換器 S の各状態 $s_i$ に対して次の操作を行う．
  - 状態 $s_i$ において検出すべき XPath 問合せを並行処理する変換器のみを，前節のアルゴリズムにしたがって構築する．これにより構築された変換器の状態数を削減できる．合成した変換器を $H_{ic}$ とする．
  - 変換器 S の $s_i$ を $H_{ic}$ に置き換える．この部分変換器を $A_i$ とする．
  - 前ステップで合成された部分変換器 $A_i$ は，変換器 S のバッファを継承する．これを $B_S$ とする．また $H_{ic}$ のバッファも $A_i$ に継承する．これを $B_{H_{ic}}$ とする．
  - $A_i$ 中の状態遷移に $B_{H_{ic}}$ に対するバッファ操作 output()，flush() がある場合は，$s_i$ の次状態へ遷移するように遷移先を変更する．そのときに出力された中間結果は，$B_S$ に入力するようにバッファ操作を加える．
- root の終了イベントより後に入力イベントが発生することはないため，root の終了イベントによって遷移する先は一つにまとめる．

上記のアルゴリズムでは単一の Snoop の演算子のみの合成方法を示したが，このアルゴリズムを再帰的に適用することによって，任意の数の Snoop の演算子の合成をすることは容易であると考えられる．

次に時制問合せ ((/a/b) ; (/a/c)) を例に，変換器の合成とその動作を説明する．((/a/b) ; (/a/c)) は，SEQ 演算 $(Q_1;Q_2)$ に XPath $Q_1$:/a/b, $Q_2$:/a/c を代入した形になっている．$Q_1$ に対する変換器は図 10, $Q_2$ に対する変換器は図 5 で示したとおりである．SEQ 演算子を処理する変換器 (図 5) の状態 $1 では入力は $E_1$ のみをとる．したがって $1 では $Q_1$ を処理する変換器のみを合成すればよい．状態 $2 では，$Q_1$ と $Q_2$ の XPath 式を評価する必要があるので，$Q_1$ と $Q_2$ を並行に処理する変換器を合成する．$3 に対しても同様に合成を行う．root の終了イベントでの遷移先は全て初期状態とする．合成された変換器の状態遷移図を図 12 に示す．図中で，Snoop の変換器に対応するバッファ操作は “→” の後に書かれている． この変換器に対して図 11 の XML ストリームを入力したときの動作の詳細を表 3 に示す．この例では問合せに “//” または複数述語を含まないため深さスタックの管理をする必要がない．よって深さスタック情報は省略している．表 3 の 5 行目では $H_1$ のバッファに対する output() 操作が行われ，S のバッファに `<b>B1</b>` が append() される．続いて 8 行目でも $H_1$ のバッファに対する output() 操作が行われる．この場合は S でのバッファ操作が replace() であるため，`<b>B1</b>` は`<b>B2</b>` に置き換えられる．11 行目で，$H_2$ のバッファに対する output() 操作が行われ，S のバッファに `<c>C1</c>` が append() される．さらに S の output() 操作により，S のバッファ中の要素 `<b>B2</b>,<c>C1</c>` が出力される．この問合せへの最終的な答えとして，

`{{(<b>B2</b>),(<c>C1</c>)},`
`{(<b>B2</b>),(<c>C2</c>)}}` が得られる．

## 5. おわりに

本研究では，XPath 問合せとイベント代数を組み合わせることで時制問合せを行う手法を提案し，時制問合せを処理する変換器の構築アルゴリズムを示した．また，変換器を合成する際に状態数の削減を行った．これにより，直積を取って合成する方法よりも効率のよい変換器を生成することができた．

今後の課題として，より多くの領域で有効な時制問合せを行うために Recent 以外の parameter context への対応についても検討したい．


## 謝　　辞

本研究の一部は，科学技術振興事業機構戦略的基礎研究推進事業「高度メディア社会の生活情報技術」プログラム，ならびに日本学術振興会科学研究費補助金基盤研究 (A)(2) (課題番号: 15200010)，若手研究 (B) (課題番号: 17700109) によるものである．ここに記して謝意を表す．

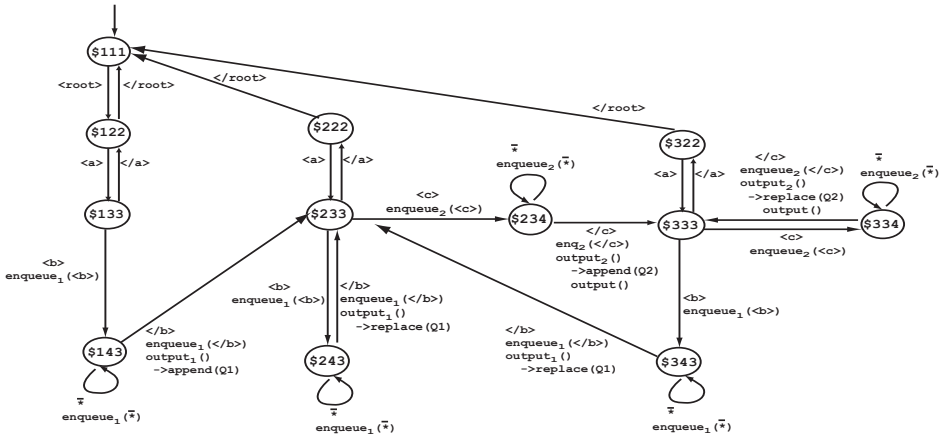


**図 12** $((/a/b);(/a/c))$ を処理する変換器

**表 3** ((/a/b) ; (/a/c)) へ図 11 の XML ストリームを入力したときの変換器の動作

| 行 | イベント | 深さ | 活性状態 | $H_1$ のバッファ | $H_2$ のバッファ | S のバッファ |
|---|---|---|---|---|---|---|
| 1 | <root> | 0 | $111 → $222 | | | |
| 2 | <a> | 1 | $222 → $333 | | | |
| 3 |  | 2 | $333 → $143 | () | | |
| 4 | B1 | 2 | $143 → $143 | (B1) | | |
| 5 |  | 2 | $143 → $233 | (B1) | | (B1) |
| 6 |  | 2 | $233 → $243 | () | | (B1) |
| 7 | B2 | 2 | $243 → $243 | (B2) | | (B1) |
| 8 |  | 2 | $243 → $233 | (B2) | | (B2) |
| 9 | <c> | 2 | $233 → $234 | | (<c>) | (B2) |
| 10 | C1 | 2 | $234 → $234 | | (<c>C1) | (B2) |
| 11 | </c> | 2 | $234 → $333 | | (<c>C1</c>) | ((B2), (<c>C1</c>)) |
| 12 | <c> | 2 | $333 → $334 | | (<c>) | ((B2), (<c>C1</c>)) |
| 13 | C2 | 2 | $334 → $334 | | (<c>C2) | ((B2), (<c>C1</c>)) |
| 14 | </c> | 2 | $334 → $333 | | (<c>C2</c>) | ((B2), (<c>C2</c>)) |
| 15 | <a> | 1 | $333 → $222 | | | |
| 16 | </root> | 0 | $222 → $111 | | | |

## 参 考 文 献


1) Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar, and S.-K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In *Proceedings of the Very Large Database*, 1994.

2) Yanlei Diao, Mehmet Altinel, Michael J. Franklin, Hao Zhang, and Peter Fischer. Path sharing and predicate evaluation for high-performance xml filtering. *ACM Trans. Database Syst.*, Vol.28, No.4, pp. 467–516, 2003.

3) Stella Gatziu and Klaus R. Dittrich. Events in an active object-oriented database system. In *Rules in Database Systems*, pp. 23–39, 1993.

4) Narain H. Gehani, H. V. Jagadish, and Oded Shmueli. Compose: A system for composite specification and detection. In *Advanced Database Systems*, pp. 3–15, 1993.

5) Ashish Kumar Gupta and Dan Suciu. Stream processing of xpath queries with predicates. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 419–430, New York, NY, USA, 2003. ACM Press.

6) Feng Peng and Sudarshan S. Chawathe. Xsq: A streaming xpath engine. In *Technical Report CS-TR-4493 (UMIACS-TR-2003-62)*, 2003.

7) World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/REC-xml, October 2000. W3C Recommendation 6 October 2000.

8) 森川裕章, 浅井達哉, 有村博紀. 半構造データ変換に基づく効率良いストリーム指向 XML データ処理系. 電子情報通信学会第 15 回データ工学ワークショップ (DEWS2004), March 2004.