データストリーム処理のための効率良いXPath問合せ機構

森川　裕章†　　浅井　達哉†　　有村　博紀†

† 九州大学大学院システム情報科学府・研究院
〒 812–8581 福岡市東区箱崎 6–10–1
E-mail: †{h-mori,t-asai,arim}@i.kyushu-u.ac.jp

あらまし　最近研究が盛んなスタック有限状態機械 (pushdown stack finite state machine) 方式のオンライン XPath 問合せ機構について，既存方式に比べて簡潔で拡張性が高い方式を提案し，実装と評価実験を行なった．

キーワード　XML 検索, XPath, データストリーム, 半構造データ, パターン照合

Efficient XPath Processing for Semi-structured Data Streams

Hiroaki MORIKAWA†, Tatsuya ASAI†, and Hiroki ARIMURA†

† Department of Informatics, Kyushu University
Hakozaki 6–10–1, Higashi-ku, Fukuoka 812–8581, Japan
E-mail: †{h-mori,t-asai,arim}@i.kyushu-u.ac.jp

Abstract　In this paper, we consider the problem of efficient processing of XML data streams online. We give another version of XML scanner called ASAX (active SAX), and then develop an efficient lightweight XPath pattern matching engine XMatch for fast XML data stream processing on Internet.

Key words　XML search, XPath, Semi-structured Data Streams, Pattern Matching, Online algorithms

1. はじめに

ウェブやインターネットをはじめとするさまざまな応用において，XML 検索技術が重要な技術となってきている．とくに，データストリーム処理に適し，多様な実行環境で軽快に動作する高速な XPath パターン照合手法が求められている．

そのため，本稿では XML データストリーム処理のための効率良い XPath パターン照合エンジンの開発を行なう．はじめに，効率良い XML ストリーム処理のための新しい走査器 ASAX を提案する．次に，XML の配送問題を中心に，最近研究が盛んになってきたイベント列の一方向逐次走査とスタック付き有限状態機械を用いたストリーム指向の XPath パターン照合アルゴリズムについて概観する．

ここから Chan 等 [9] が採用している複数パターン照合機械を用いたパターン照合機械構築法を選び，プログラミング言語の実行時処理系のアイディアを借りて，効率よくかつ拡張性をもつストリーム指向の XPath パターン照合アルゴリズム XMatch を開発する．

計算機実験では，代表的な XPath パターン処理系である Xalan+DOM および，Peng 等のストリーム指向 XPath パターン照合アルゴリズム XSQ+SAX [8] と比べて，XMatch が著しく（5〜10 倍程度）高速であることを確認した．

2. 準　　備

2.1 XML データと XPath パターンのモデル

はじめに，XML のデータモデルである**データ木**（または **DOM 木**）を導入する．データ木は，ノードラベルをもつ順序木である．データ木の節点は，**要素**(element) と**属性**(attribute) の二つの種類に区別され，それぞれ，XML データの階層的構造と内容を表す．さらに，要素と属性のそれぞれについて，それらの節点ラベルは，**名前**(name) と**値**(value) の 2 種の

情報をもっている．まとめると，データ木の節点は，表 1 のように 4 つの型に区別される．

表 1　データ木とパターン木のラベルの型

節点	ラベル	ラベル名称
V_{elem}	$\mathcal{L}_{elem}$	要素名 (タグ名)
V_{elem}	Δ_{elem}	要素値 (テキスト)
V_{attr}	$\mathcal{L}_{attr}$	属性名
V_{attr}	Δ_{attr}	属性値

形式的には，**データ木**(data tree) は，5 つ組 $D = (V, E, B, r, label)$ で表される．ここに，D は次を満たす：

- $V = V_{elem} \cup V_{value}$ は，**節点**(node) の有限集合である．ここに，集合 V_{elem} は**名前節点**の集合であり，V_{value} は**値節点**の集合である．節点 $r \in V$ は根と呼ばれる特別な節点である．V_{elem} と V_{value} は互いに共通部分をもたないと仮定する．
- $E \subseteqq V \times V$ は，有向辺（単に**辺**(edge) と呼ぶ）の集合であり，**親子関係**と呼ばれる．もし $(v, w) \in E$ ならば，v は w の**親**(parent) であるといい，w は v の**子**(child) であるという．根以外のすべての節点は，ちょうど一つの親をもつ．
- $B \subseteqq V \times V$ は，**兄弟関係**と呼ばれる 2 項関係である．各節点の子は，兄弟関係 B によって左から右に順序付けられる．もし $(v, w) \in B$ ならば，v は w の**前兄**または**先行者**(predecessor) であるといい，w は v の**次弟**または**後続者**(successor) であるという．
- 関数 $label : (V_{elem} \to \mathcal{L}) \cup (V_{value} \to \Delta)$ は，**ラベル関数**(label function) である．ラベル関数 $label$ は，名前節点に名前を割り当て，値節点に値を割り当てる．
- 要素節点 $v \in V_{elem}$ は，もし $label(v) \in \mathcal{L}_{elem}$ ならば，子として 0 個以上の要素節点をもつ．もし $label(v) \in \mathcal{L}_{attr}$ ならば，子としてちょうど 1 個の値節点をもつ．値節点 $v \in V_{value}$ は，そのラベルに関係なく，つねに子をもたず葉となる．

先にみたように，実際の XML データでは，節点は表 1 のように 4 つの型に区別される．しかし，本稿の手法では，いったん走査器によって XML データをイベント列に変換した後は，要素節点と属性節点を区別したり，テキスト節点と属性値節点を区別する必要はない．したがって今後は，節点の型を名前節点と値節点として扱う．

2.2　XML 文書

XML 文書は，タグ付きテキストによるデータ木のテキストによる符号化である．与えられたデータ木 D に対して，D の **XML データによる符号化** $\mathcal{X}(D)$ を次のように定める．

D の深さ優先探索（または，D のオイラー路）π において，各節点は葉方向へ下降するときに初めて 1 回訪問され (pre-order)，つづいて通りすぎるときに 0 回以上 (in-order)，最後に根方向へ上昇するときに 1 回訪問する (post-order)．このとき，節点 $v \in V$ を訪問する際に，次のようにタグを出力することで，

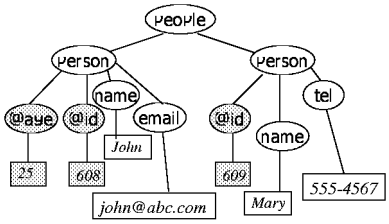


図 1　データ木の例．円は名前節点を表し，四角は値節点を表す．各節点の色は，それが要素型か属性型かを示す．

```
<people>
  <person age=“ 25 ” id=“ 608 ”>
    <name>John</name>
    <email>john@abc.com</email>
  </person>
  <person id=“ 609 ”>
    <name>Mary</name>
    <tel>555-4567</tel>
  </person>
</people>
```

図 2　XML 文書の例

XML 文書 $\mathcal{X}(D)$ を生成する：

- 名前節点を初めて訪問するとき．その節点が名前 $name \in \mathcal{L}_{elem}$ をもち，その子として名前が $attr_1, \ldots, attr_m \in \mathcal{L}_{attr}$ である属性名前節点と，さらにその下に値が $val_1, \ldots, val_m \in \Delta_{attr}$ である属性値節点をもつとき，次の開始タグを出力する：

  <$name$ $attr_1$=''val_1'' $\cdots$ $attr_m$=''val_m''>

- 名前節点を最後に訪問するとき．その節点の名前が $name \in \mathcal{L}_{elem}$ ならば，終了タグ</$name$>を出力する：
- 値節点 v を訪問するとき．v は子をもたないので，一度だけ訪問される．このとき，その値文字列 $label(v) \in \Sigma^*$ をそのまま出力する：

この走査を，すべての内部節点に対してくり返すことで，開始タグと終了タグが入れ子上に対応した XML データ $\mathcal{X}(D)$ を得る．XML 文書からデータ木への逆の対応も一意に定まる．図 2 の右側に，例として，左側のデータ木に対応する XML 文書を示す．

2.3　XPath パターン

ステップ(step) とは組 $S = (\chi, c) \in \{/, //\} \times \mathcal{L}$ をいう．χ を**軸**(axis) といい，/を子供軸といい，//を子孫軸という．**パス**(path) は，ステップの有限列 $\pi = S_1 \cdots S_m$ $(m \geqq 0)$ である．**XPath パターン**は，次のように帰納的に定義される．

- 基本テスト B は XPath パターンである．ここに，基本テストとは，文字列照合''str''($str \in \Sigma^*$), 属性テスト@$attr$=''val'' ($attr \in \mathcal{L}_{attr}, val \in \Delta_{attr}$), 位置テスト@position=$p$, @first, @last ($p \in \mathbf{N}$) の任意の一つである．基本的テスト $t \in \mathbf{B}$ の意味は，現在の節点で，テスト t を評価し，真偽を返すものである．

• パス $\pi = S_1 \cdots S_m$ と，XPath パターン $P_1, \ldots, P_n$ $(n \geqq 0)$ に対して，表現

$$P = \pi[P_1, \ldots, P_n]$$

は XPath パターンである．

部分集合 $T \subseteqq \{/, //, [], str, @, pos\}$ に対して，$XPath(T)$ で T の型の生成規則で定められる XPath パターンの部分族を表す．本稿では，ancestor と，parent(..), following-sibling，preceding sibling の軸は扱わない．

XPath パターンの意味論については，文献 [6] に簡潔かつ正確な定義があるので，そちらを参照されたい．直感的には，XPath パターン $P = \pi[P_1, \ldots, P_n]$ は，与えられた節点（文脈節点という）から出発して，パス π で到達可能な節点で，述語 $P_1, \ldots, P_n$ の条件を満たすものの集合全体を返す．ここに，部分パターン P_i はそれが基本的テストなら，その節点でテストが真を返せばよく，それが再び複合パターンであれば，そこから検索を開始して，一つでも照合する節点があればよい．

本稿では，XPath パターンをしばしば木とみなす．$\mathsf{B}(\Delta)$ を値領域 Δ 上の基本テストの集合とする．**パターン木**(pattern tree) は，節点ラベルをもつ根付き木 $P = (V, E, r, label)$ である．ここに，$V = V_{elem} \cup V_{value}$ は節点集合であり，E は辺の集合（親子関係），$r \in V$ は根，関数 $label : (V_{elem} \to (\{/, //\} \times \mathcal{L})) \cup (V_{value} \to \mathsf{B}(\Delta))$ はラベル関数である．XPath パターンとパターン木の対応の正確な定義は略する．

［例 1］ XPath パターン $P =$ a/b[c[//b,d],//d/c] に対するパターン木を図 5 の左側に示す．

3. XML 文書走査器

本節では，われわれのアプローチに特徴的な XML 文書走査器 ASAX(Active SAX, A-SAX) を説明する．

最も広く使用されている XML 文書走査器は，SAX(Simple Application Interface for XML) である．SAX を用いたオンライン処理では，XML 文書を左から右に走査しながら，イベントの出現ごとに，対応する文字列 $e \in \Sigma^*$ を切り出し，必要ならこれをハッシュ辞書で一意な識別番号 $id(e)$ に変換し，応用プログラムに渡して後処理を行なう．しかし，XPath 処理系では，問合わせパターンに含まれるほんの一部のイベントだけを検出すれば良いので，このやり方は一般的すぎて無駄が多い．

そこで，ASAX では，あらかじめ必要なイベントだけを走査器に登録し，前処理によって各クラスのイベントを検出する数種の有限状態変換機を構成する．実行時には，これらの変換機を切り替えつつ，入力 XML 文書を左から右に一度だけ走査しながら，実時間でイベントとその識別番号を検出する．XPath パターン処理と文字列パターン照合を効率よく融合可能なことも，この方式の利点である．

ASAX の基本イベントは，次のいずれかである．ここに，$\mathcal{I}$ を問合わせ XPath パターンに出現する $id \in \mathcal{I}$ である．

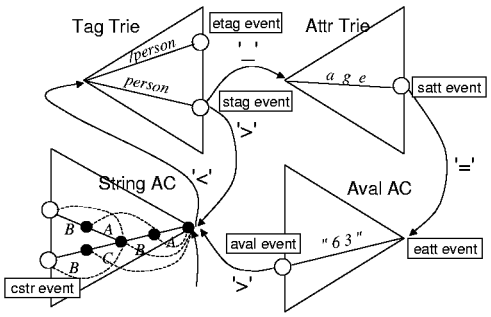


図 3　XML 走査器 ASAX の構成

- タグ開始イベント $(\mathtt{stag}, id)$.
- タグ終了イベント $(\mathtt{etag}, id)$.

それぞれ，識別番号 id をもつタグ名 $name$ に対する開始タグと終了タグの出現を通知する．

- 属性開始イベント $(\mathtt{satt}, id)$.
- 属性終了イベント $(\mathtt{eatt}, id)$.

それぞれ，識別番号 id をもつ属性名 $name$ の開始と終了を通知する．

- テキスト照合イベント $(\mathtt{cstr}, id)$.

識別番号 id をもつ文字列パターンの要素値テキスト中での部分文字列としての出現を通知する．

- 属性値照合イベント $(\mathtt{aval}, id)$.

識別番号 id をもつ属性値の出現を通知する．属性値は，具体的な値だけでなく，整数型や，実数型，西暦型や，列挙型等の値クラスを記述する正則言語として与えてよい．

図 5 に ASAX の構成を示す．各イベントクラスに対応した検出器は次のとおりである．

- Tag Trie は登録されたタグ名文字列集合に対する**トライ**(trie, 前綴り木) [4] である．未登録タグに対しては，$(\mathtt{stag}, default)$ や $(\mathtt{etag}, default)$ のように特別な識別番号 $default$ を返す．図では簡略化されているが，小さな DFA を末尾に付加し，インラインタグ<$name$/>を検出する．
- Attr Trie は，同様に，登録された属性名集合に対するトライである．未登録の属性名に対しては，識別番号 $default$ を返す．
- String AC は登録された文字列パターン集合 S に対する対する**複数パターン照合機械**(Aho-Corasick 機械) [2] である．これは，テキスト中の任意の S のパターンの部分語としての出現を検出する．
- Aval AC は，同様に，登録された属性値集合に対する AC 機械またはトライである．

これらの検出器は，それぞれ決定性有限状態機械（DFA）であり，受理状態に検出した文字列の識別番号リストを格納する．また図 5 のように，特別な記号 $\{<, >, =, \sqcup\}$ によって，有限状態機械と現在の状態を次々に切り替えながら，XML 文書を走査し，登録されたイベントを実時間で検出する．

4. XPath パターン照合機械

本節では，最近研究が盛んになってきているスタッ

ク付き有限状態機械を用いたデータストリーム指向の XPath パターン照合アルゴリズムについて概観する [7]〜[9].

4.1 パターン NFA

XPath パターン P に対して，P が表すパターン NFA $G(\pi)=(Q,\Sigma,\delta,I,F,\tau)$ を次のように帰納的に定義する．ただし，Q は状態の集合，Σ は入力アルファベット，$\delta \subseteq Q\times\Sigma\times Q$ は遷移関係，$I,F\subseteq Q$ は初期状態と終状態の集合，$\tau : Q \to 2^{\mathsf{B}}$ は状態に基本テストの集合を割り当てる関数である．

（1） パス $\pi = S_1\cdots S_m$ $(m \geqq 0)$ に対して，そのパス NFA を一本鎖状の NFA $G_\pi=(Q,\Sigma,\delta,I,F,\tau)$ と定義する．ここに，$S_i=(\chi_i,c_i)$ $(i=1,\ldots,m)$ であり，次が成立する：

- $Q=\{q_0,\ldots,q_m\}$ かつ，$I=\{q_0\}, F=\{q_m\}$.
- G_π の遷移関係は，文字列 $c_1\cdots c_m$ を受理する DFA の遷移関係と，任意の文字での自己ループを子孫軸に対応する状態 q_i に付加したもの：

$\delta = \{\,(q_{i-1},c_i,q_i) \mid i=1,\ldots,m\,\}$
$\cup\{\,(q_i,c,q_i) \mid i=1,\ldots,m,\ \chi_i=\text{“//”}\,\}$

- テスト集合は空：$\tau(q)=\emptyset$ $(\forall q\in Q)$.

（2） 基本テスト $B\in\mathsf{B}$ だけからなる XPath パターンに対して，そのパターン NFA を，ただ一つの状態をもつ NFA $G_B=(\{q_0\},\Sigma,\emptyset,\{q_0\},\{q_0\},\tau)$ と定義する．ここに，$\tau(q_0)=\{B\}$ と定義する．

（3） $\pi = S_1\cdots S_m$ $(m \geqq 0)$ をパスとし，$P_1,\ldots,P_n$ $(n\geqq 0)$ を XPath パターンとする．パス π のパターン NFA を $G_\pi=(Q,\Sigma,\delta,\{p_0\},\{p_m\},\emptyset)$ とし，各部分パターン P_i のパターン NFA を，$G_i=(Q_i,\Sigma,\delta_i,\{q_i^0\},F_i,\tau_i)$ とする $(i=1,\ldots,n)$. このとき，XPath パターン

$$P = \pi[P_1,\ldots,P_n]$$

に対して，そのパターン NFA を $G_P = (Q,\Sigma,\delta,I,F,\tau)$ と定義する．ここに，

- G の状態は，$G_\pi, G_1,\ldots,G_n$ の状態の和である：

$Q = Q_\pi \cup \bigcup_i Q_i$.

- G の遷移関係は，$G_\pi, G_1,\ldots,G_n$ の遷移関係の和と，それに加えて，パス π の末端（終状態）から各 NFAG_i の初期状態に空遷移を加えたもの：

$\delta = \delta_\pi \cup \bigcup_i \delta_i \cup \{\,(p_m,\varepsilon,q_i^0) \mid i=1,\ldots,n\,\}$.

- 初期状態は，G_π の初期状態である：$I=\{p_0\}$.
- 終状態は，P 中のすべてのパスの末端からなる：

$F = \{p_m\} \cup \bigcup_i F_i$.

- テスト集合は全体の和：$\tau(q_0) = \bigcup_i \tau_i$.

［例 2］ 図5右側に，パターン $P=$ `a/b[c[//b, d], //d/c]` のパターン NFA G_P を示す．右側の G_P の終状態（二重の円）が左側のパターン木 T_P の状態に対応していることがわかる．

4.2 パターン照合の基本方針

$\mathcal{X}(D)$ を，データ木 D に対応する XML 文書とする．このとき，XPath パターン $P=\pi[P_1,\ldots,P_n]$ のパターン NFA を $G_P=(Q,\Sigma,\delta,I,F,\tau)$ とおく．G_P を用いた XPath パターン照合を説明する．

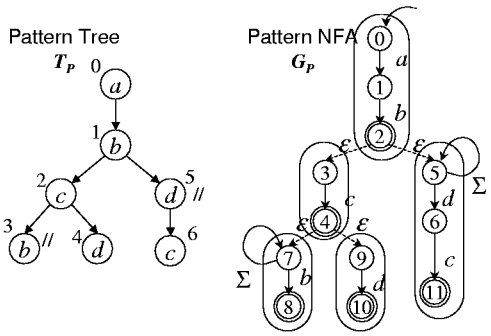


図 4 パターン木とパターン NFA の例．XPath パターンは $P=$ `a/b[c[//b,d],//d/c]`.

説明の都合上，遷移関数 δ を $\delta=\beta\cup\gamma\cup\lambda$ と 3 つに分割する．ここに，$\beta=(q_{i-1},c_i,q_i)\subseteq Q\times\Sigma\times Q$ はパス上での文字を読んで次の状態に進む順方向の遷移の集合である．$\gamma=(q,c,q)\subseteq Q\times\Sigma\times Q$ はパス上での自己ループによる遷移の集合である．$\lambda=(p,\varepsilon,q)\subseteq Q\times\{\varepsilon\}\times Q$ はあるパスの終端から子供のパスの始端に進む空遷移の集合である．

パターン照合機械は，下降状態集合と上昇状態集合と呼ばれる二つの状態集合 $D,U\subseteq Q$ を用いて，イベント系列上で対応するデータ木の仮想的な深さ優先巡回を模倣する．パターン NFA を評価する．仮想的な深さ優先巡回では，D の各節点 v を下降時と上昇時の 2 回訪問する．

このとき，パターン照合機械は，次が常に成立するように状態集合 D と U を管理する．問合わせパターンの任意の部分パターン $P=\pi[P_1,\ldots,P_n]$ $(n\geqq 0)$ に対して，最上部のパス π の末端に対応するパターン NFA G_P の状態を $q\in F_P$ とする．

- 下降時に q が活性状態である $(q\in D)$ ならば，対応するパターンのパス π は D の根から v へのパスに照合し，π の末端が節点 v に対応する．
- 上昇時に q が活性状態である $(q\in U)$ ならば，対応するパターンの部分木 $c[P_1,\ldots,P_n]$ は，その根 r が節点 v に対応するよう D の部分木 $D(v)$ に照合する．ここに，c は π の末端のラベル．

4.3 基本的評価アルゴリズム

基本評価アルゴリズムは，D と U を状態リストとして実装する．状態集合への非決定的遷移の適用 $D := \delta(D)$ は，状態リストを走査しながら，パターン NFA の枝をたどることで行なわれる．アルゴリズムは次のとおりである．

Algorithm Basic:

（1） $(D,U) := (\emptyset,\emptyset)$ と初期化する．

（2） 新しい XML イベント e が来る限り，以下を繰り返す：

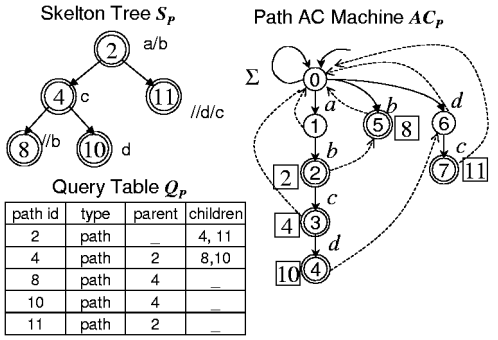
(2.a) 開始タグイベント $e=(\mathtt{stag},c)$ の時：

- $Push(Stack,(D,U))$ で，活性状態集合を退避する．
- 下降状態集合の遷移を行なう．

$D := \beta(D,c)\cup\gamma(D,c)$; $D := \lambda(D,c)$.

(2.b) 終了タグイベント $e=(\mathtt{etag},c)$ の時：

- $(D_{old},U_{old}) := (D,U)$; $(D,U) :=$



path id	type	parent	children
2	path	_	4, 11
4	path	2	8,10
8	path	4	_
10	path	4	_
11	path	2	_

図 5　質問木とパス AC 機械．XPath パターンは P = `a/b[c[//b,d],//d/c]`.

$Pop(Stack)$ で，以前の状態集合に復帰する．

- 各 $p \in D \cap F$ に対して，$\lambda(p) \subseteqq U$（すべての子供が上昇状態）であるかチェックし，成立するならば $U := U \cup \{p\}$.

(2.c)　値イベント $e = (type, id)$ の時 $(type \in \{\texttt{cstr}, \texttt{aval}\})$：$id \in \tau(p)$ を満たす終状態 $p \in D \cap F$ に対して，$U := U \cup \{p\}$.

（3）　もし根に対して $q_0^P \in U$ ならば，パターン照合は成功.

以上により，長さ n の XML 文書とサイズ m の XPath パターン P に対して，$O(mn)$ 時間でパターン評価を行なう基本的なアルゴリズムを得た.

4.4　関連アルゴリズムの導出

状態集合 D と U の実装や，更新方法の改良により，基本アルゴリズムを改良することができる：

Gupta と Suciu [7] は，パターン NFA を決定性有限機械 (DFA) に変換することで，実行時の高速化を図っている．これは，状態集合 $D, U \subseteqq Q$ をビットベクトルで表現し，各ビットベクトルを一つの状態だと考えることで，Q の各部分集合を，新たな一つの状態だと考えて NFA を DFA 化する．DFA サイズの指数的増大を避けるため，Gupta 等は，必要になったときに初めて状態を決定化し，2 度目以降は再利用する遅延決定化技法を提案している．

Peng と Chawathe [8] は，パターン照合に明示的にスタックを使わずに，NFA の状態遷移でスタック機械の動きを模倣するアイディアを提案している．決定化は行なっていない．この手法の短所は，各節点が基本テストしかもたないような鎖状の XPath パターンに対してさえ，変換後の NFA のサイズが指数的になってしまう点である．また，NFA が複雑になるため，高速化の効果にも疑問が残る．

Chan, Felber, Garofalkis, Rastogi 等 [9] は，NFA による基本手法を，複数パターン照合機械を用いて部分的に決定化する手法を提案し，XTrie システムを開発している．この手法については，次節で詳細に検討する．

5.　XMatch アルゴリズム

本節では，イベント列上をスタックを用いて走査する間に，与えられた XPath 問合わせを行なう XPath パターン評価アルゴリズム XMatch を与える．これは，前節で紹介した基本アルゴリズムを改良しており，複数パターン照合機械と動的な環境管理を用いている．

AC 機械を用いたスタック機械方式 XPath 照合機械は，XTrie [9] で提案された．ここでは，プログラミング言語処理系で使われる技法である動的なメモリ割付とディスプレイによる値参照 [3] をスタック方式照合機械と組み合わせることで，簡潔かつ拡張が容易な XPath パターン照合機械を構成する．

5.1　問合わせ木とパス AC 機械

XPath パターン P のパターン NFA $G_P = (Q, \Sigma, \delta, I, F, \tau)$ が与えられたと仮定する．XMatch の基本的な考えは，NFA G_P を質問の構造を表す小さな NFA と，複数の長いパスの出現を同時に検出する DFA に分解することにある．前者の NFA を問合わせ木 (Skelton Tree) と呼び，後者の DFA をパス AC 機械 (Path AC machine) と呼ぶ.

前節でみたように，パターン NFA $G_P = (Q, \Sigma, \delta, I, F, \tau)$ は，一つ以上のパス NFA から構成される．各パス $\pi \in \Pi$ に対して，パス NFA G_π は，ちょうど一つずつの初期状態 s_π と終状態 t_π をもつ．パス $\pi \in \Pi$ に対して，そのラベルを連結して得られる文字列を $str(\pi)$ と書く．ここで，問合わせパターン P は，子孫軸をパスの先頭にだけ持つ $P = //A_1/\cdots/A_m[P_1, \ldots, P_n]$ ような形のパターンに変換されていると仮定する．新たな述語 [] の導入で，このような変換はいつでも可能である.

パターン P の**問合わせ木**は，G_P の各パスを節点とし，パス間の空遷移を唯一の枝とする根付き木 S_P である．すなわち，$S_P = (Q, E, r, label)$ は次のように定義される：

- $Q = \{\, t_\pi \mid \pi \in \Pi \,\}$ はパターン NFA の終状態の集合.
- $(t_\pi, t_{\pi'}) \in E$ iff $(t_\pi, \varepsilon, s_{\pi'}) \in \delta$.
- $r = t_{\pi 0}$ は，G_P の最上位パスの終状態.
- $label(t_\pi) = str(\pi)$.

問合わせ木 S_P から，S_P の各節点に対して，$type, parent, children$ の値をもつ問合わせ表（Query Table）と呼ぶ表 QT_P を作成しておく．節点の型は，ここでは $path$ のみである．これは，集約質問や，属性テスト，計数，変数の導入等の拡張に利用する.

パス AC 機械は，文字列集合 $str(\Pi) = \{str(\pi) | \pi \in \Pi\}$ に対する複数パターン照合機械 (Aho-Corasick 機械) $M(\Pi) = (R, \Sigma, goto, fail, output)$ である．ここに，

- $R = \{\, \overline{s} \mid s \in Prefix(\pi), \pi \in \Pi \,\}$.
- $goto : R \times \Sigma \to R$ は $str(\Pi)$ のトライを定義する.

つまり，$goto(\overline{p}, a) = \overline{pa}$ $(\forall \overline{p} \in R)$.

- $fail : R \to R$ は，空遷移を定義する．ここで，$fail(\overline{p})$ は，p の最長の空でない接尾辞を $q \in str(\Pi)$ を与える.
- $output(\overline{p})$ は，p の接尾辞となるパス $q \in \Pi$ が表す状態 $t_\pi \in Q$ 全体の集合を返す.

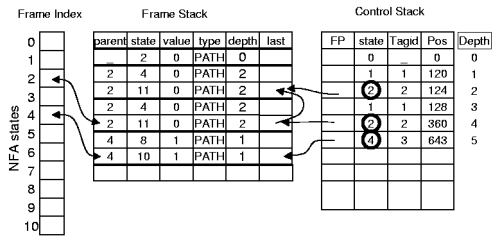


図 6　XPath 照合機械の実行時環境

AC_P は，与えられた記号列上を左から右に一度だけ走るあいだに，$str(\Pi)$ のすべての文字列の出現を $O(n+k)$ 時間で検出する．ここに，n は記号列の長さであり，k は出力数である．

5.2　実行時環境とシステムの実装

XMatch は，問合わせ木とパス AC 機械を組み合わせて，イベント列上のパターン NFA の動きを模倣する．XMatch の特徴は，実行時環境において，巨大な 2 値の静的状態表を使わずに，プログラミング言語における関数呼び出しと入れ子状の変数束縛環境を導入したことである．これにより，複雑かつ大量の問合わせを記憶領域を節約しながら，同時に高速に処理できる．もう一つの特徴は，これにより状態の概念を一般化し，状態を整数カウンターや一般の変数として使うことができる．(このアイディアは，Arikawa 等の SIGMA システム [5] に遡れる．)

これにより，以下のようなさまざまな演算を自然に，XPath パターン照合に取り込むことができる：

- ブール値 $\{0,1\}$ を値域とし，論理積を上昇時演算とすることで，通常のパターン NFA を模倣できる．
- 自然数 $\mathbf{N} = \{0,1,2,\ldots\}$ を値域とし，count(P) 等の**計数演算**(counting) が，データの逐次走査で自然に実現できる．
- さらに，加法半群 $(\mathbf{N},+)$ などの結合束を満たす値域をとれば，SUM, AVR, MAX, MIN, VAR 等のデータベースの**集約演算**(aggregations) が実現できる．
- 一般に大きな問合わせおよびデータ計算量をもつと考えられている XPath での位置制約演算 position=p [6] を，パターン自身の計数演算に帰着できるため，効率良い実装が可能である．

図 6 に XMatch の実行時環境を示す．紙数の制限から，今回は予備的な記述にとどめる．実行時環境と，AC 機械による模倣については，別の機会に詳細な説明を与えたい．

Control Stack には，開始タグイベント (e.g., stag，satt) が到着する度に組 (現在のパス AC 機械の状態，タグ ID，入力テキスト上の読み込み位置) が一段ずつ押し込まれる．パス AC 機械が，仮想的なパターン NFA のパスを検出する度に，その子パスに関する状態や値変数等の情報が，6 組 $(parent, state, value, type, depth, last)$ として，**Frame Stack** 上に続けて，割り当てられる．

終了タグイベント (e.g., (etag，eatt)) が到着すると，Control Stack 上の対応する開始タグと，そのときのパス AC 機械の状態が復元される．この情報を

表 2　XPath パターン

問	DB	パターン
Q1	DBLP	/dblp/article
Q2	DBLP	/dblp/article[author,title,year]
Q3	DBLP	/dblp/inproceeding[@mdate="2002", author,title,year,ee['pdf']]
Q4	DBLP	/dblp/article[@mdate="2002", year]
Q5	PSD	/ProteinDatabase/ProteinEntry[sequence, protein/name]
Q6	PSD	//sequence

もとに，Frame Stack 中の仮想的なデータ木の節点で照合した部分パターンとその子パターンの情報が取り出され，必要な検査が行なわれる．前小節の計数演算や集約演算のための変数（状態）への演算は，このときに行なわれる．

子孫軸を含むパターン照合を実現するためには，指定されたパターン P に対して，現在の文脈節点の上方のすべての出現を効率よく列挙する機構が必要である．これは，プログラム言語処理系における**ディスプレイ**を用いた値参照を用いる [3]．Frame Index を用いて，指定された状態の最新の出現フレームに定数時間でアクセスし，そこからフレームの last ポインタをたどって，すべての上方の出現を列挙する．

6.　実　　験

本節では，アルゴリズムの評価実験について述べる．

6.1　データと実験手順

実験データには次の 2 種類の XML データを用いた．

- DBLP：オンライン論文データベース (160MB)(注1)．
- PSD：アミノ酸配列データベース (801MB)(注2)．

DBLP は平均深さが 3 前後と浅く単純なデータであり，PSD は平均深さ 6 前後とやや深いデータである．

次の 3 つのアルゴリズムの計算時間と使用領域量を比較した．

- XMatch: 5. 節のアルゴリズムを Java1.4.1 で実装したもの．XML 走査器には 3. 節の A-SAX を用いた．
- XSQ+SAX: Peng と Chawathe 等 [8] のスタック付き有限状態機械によるアルゴリズム XSQ を彼等が Java で実装したもの．GPL の下で公開されている(注3)．
- Xalan+JAXP: Apache による XPath1.0 と XSLT1.0 の Java を用いた実装．入力と前処理に JAXP (Apache Xerces) の DOM を用いて，主記憶上にデータ木を構築する．

実験環境は，PC (Athlon4, 1.2GHz, RAM 1GB, WindowsXP) 上で, Java SDK1.4.1+JAXP を用いて実験を行った．表 6.1 に，実験に用いた XPath 問合せを示す．二つのデータベースにおいて，これらの質問を 1 回ずつ実行し，計算時間と使用領域を計測した．このうち実験では，問合せ 4 に関する結果のみ示す．他の質問についても，実験結果の傾向は変わらな

(注1)：http://dblp.uni-trier.de/
(注2)：PIRhttp://pir.georgetown.edu/
(注3)：http://www.cs.umd.edu/projects/xsq/

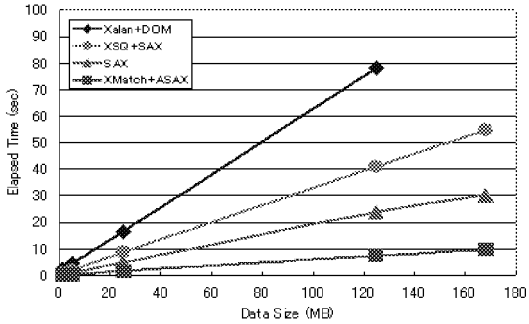


図 7　外部記憶上での計算時間の比較 (DBLP)

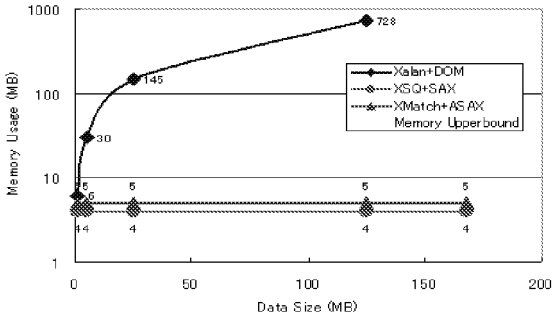


図 8　記憶領域量の比較 (DBLP)

いことを確認した.

6.2 実験結果

DBLP データ上で，データサイズを 1(MB) から 168(MB) まで増加させながら，質問 4 に対する 3 つのアルゴリズムの計算時間と使用領域を計った．Java Virtual Machine には，800MB の主記憶領域を割り当てた.

6.2.1 計算時間

図 7 に，外部記憶上にデータをおいた場合に，ファイル入力から検索までにかかる総計算時間のグラフを示す．横軸にデータサイズ (MB) を示し，縦軸に総計算時間 (sec) を示す．また例として，次の表に，DBLP と PSD における固定したサイズに対する計算時間を示す.

表 3　計算時間の比較

Elapsed time	SAX	XMatch	XSQ	Xalan
DBLP (125MB)	23.7	7.35	41.0	78.1
PSD (801MB)	157.4	45.9	325.0	NA

まず，上のグラフから，どのアルゴリズムの計算時間もデータサイズに線形に増加していることがわかる．この意味でどれも時間に関して規模耐性が良いといえる.

しかし，上の図 7 のグラフと表 3 の時間から，Peng 等の XSQ+SAX は，Xalan+DOM の 2 倍弱高速であり，さらに，われわれの XMatch+ASAX は XSQ+SAX のさらに 4 倍程度 (Xalan+DOM の 10 倍) 高速であることがわかる．この結果は，XSQ+SAX や XMatch+ASAX のようなストリーム指向のスタック付き有限状態機械アプローチの優位性を示す.

6.2.2 使用領域量

図 8 に上と同じ実験での 3 つのアルゴリズムの使用領域量の比較のグラフを示す．横軸にデータサイズ (MB) を示し，縦軸に実行環境である Java Virtual Machine が使用した領域量 (MB) の対数を示す．データストリーム指向アルゴリズムである XSQ+SAX と XMatch+ASAX は，入力データサイズと無関係に，それぞれ，4(MB) と 5(MB) 程度の記憶領域しか使っていない．これに対して，主記憶指向アルゴリズムである Xalan+DOM はデータサイズに比例する記憶領域を必

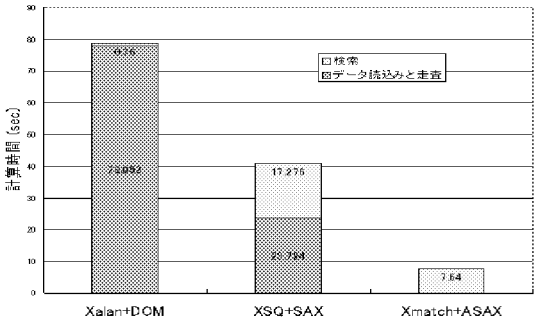


図 9　計算時間の内訳 (DBLP)

要とし，データサイズ $N = 125$(MB) では 800(MB) のメモリをほとんど使いきり，$N = 168$(MB) では計算を継続できなかった

図 9 に 3 つのアルゴリズムの計算時間の詳細を示す．このグラフから，Xalan+DOM では，簡単な問合わせでは，前計算時間の 99%をデータ入力と DOM 木構築が占めていることがわかる．また，XSQ+SAX では前計算時間の 57.5% を XML 走査器の SAX が占めていることがわかる.

これに対して，われわれの XMatch+ASAX では，全計算時間を合わせても，SAX だけによるデータ走査時間の 1/3 以下にしかならないことがわかる.

以上の結果より，アルゴリズム XMatch は，高速に流れ続ける XML ストリームに対して，効率よく働き続ける効率よいアルゴリズムであるといえる.

7. 終わりに

本稿では，最近重要性を増している XML ストリーム処理に取り組み，効率よく XML データ走査を行なう ASAX と，イベント列の一方向逐次走査を用いて，スタック有限状態機械を用いて効率良い XPath パターン照合を実現するアルゴリズム XMatch を提案した.

実際の XML データを用いた実験では，標準的な XML データ処理パッケージである Apache Xalan (with Java 1.4.1 JAXP) や，同種のスタック付き有限状態機械による XPath パターン照合アルゴリズム XSQ [8] に対して，高速であることを確認した.

XMatch は，Chan 等の XTrie [9] と基本設計は同じであるが，ASAX との結合や，動的な束縛環境の利用などことなる点も多い．今後はこの点を生かして，高度な DB 問合わせの実現や，出力や再構成機構の実装，プログラミング言語との融合など挑戦したいと考える．また，XPushMachine [7] や XTrie [9] との実証比較も今後の課題である．

Takeda, Miyamoto 等 [10] は，Prefix 符号上の複数パターン照合アルゴリズムを開発し，XML データ処理に応用している．このような技法は，3. 節の ASAX のような効率良い文書走査器の実現に有効な技術である．

文　　献

[1] S. Abiteboul, P. Buneman, D. Suciu, Data on the Web, Morgan Kaufmann, 2000.

[2] A. V. Aho, M. J. Corasick, Efficient String Matching: An Aid to Bibliographic Search, *CACM* 18(6), 333–340, 1975.

[3] A. V. Aho and J. D. Ullman, *Principles of Compiler Design*, Addison-Wesley, 1977.

[4] A. V. Aho, J. E. Hopcroft, J. D. Ullman, Data Structures and Algorithms, Addison-Wesley, 1983.

[5] S. Arikawa, T. Shinohara, A Run-Time Efficient Realization of Aho-Corasick Pattern Matching Machines, New Generation Computing, 2(2), 171–186, 1984.

[6] G. Gottlob, C. Koch, and R. Pichler, Efficient Algorithms for Processing XPath Queries, Proc. 28th VLDB'02, 2002.

[7] A. Gupta and D. Suciu, Stream Processing of XPath Queries with Predicates, In *Proc. ACM SIGMOD Conference on Management of Data* (SIGMOD'03), 2003.

[8] F. Peng and S. S. Chawathe. XPath Queries on Streaming Data, In *Proc.. ACM SIGMOD International Conference on Management of Data* (SIGMOD'03). 2003.

[9] C. Y. Chan, P. Felber, M. N. Garofalakis, R. Rastogi, Efficient Filtering of XML Documents with XPath Expressions, In *Proc. ICDE'02*, 2002.

[10] M. Takeda, S. Miyamoto, T. Kida, A. Shinohara, S. Fukamachi, T. Shinohara, S. Arikawa, Processing Text Files as Is: Pattern Matching over Compressed Texts, Multi-byte Character Texts, and Semi-structured Texts. In *Proc. SPIRE 2002*, LNCS, Springer, 170–186, 2002.