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

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　高速ネットワーク技術と大容量記憶装置の発達により，この数年，ウェブページや XML データ等の膨大な半構造データが流通・蓄積されるようになってきた．そのため，パターン出現検出や統計情報収集，データ変換等のストリーム情報処理を，データの流れを止めることなく効率良く行うことが必要となっている．本研究では，ストリームデータから，有用な情報を効率よく獲得する高速なオンライン型半構造情報変換システムの研究開発を行なう．具体的には，先に開発したストーム指向 XPath 処理系 XMatch に，抽象命令を処理する抽象機械を導入することで，検索・統計・変換が統一的に行なえることを示す．

キーワード　XML 検索, XPath, XSLT，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, XSLT, Semi-structured Data Streams, Pattern Matching, Online algorithms

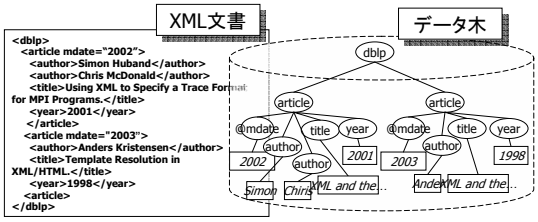

図 1　XML 文書と対応したデータ木

1. はじめに

インターネットの発展を背景として，XML や HTML 等の**半構造データ** [1], [17] の利用が盛んになっている．図 1 に，XML 文書と対応するデータ木 (DOM) の例を示す．最近では，データの交換や蓄積だけでなく，ウェブサービスの隆盛に伴って，SOAP や WSDL, RDF などさまざまなインターネット技術の基盤として XML が用いられている．そのため，XML データに対する効率よい検索・変換・蓄積技術が盛んに研究されている．

XPath は，XML 文書中の階層的情報へのアクセス手段として，1999 年に W3C によって勧告された検索言語である [18]．図 2 に，XPath 問い合わせの例を示す．XPath は，XML 文書の木構造を利用し，パスや部分木に関する条件を組み合わせて，必要な情報を選択する．XPath は, XSLT や XQuery 等のさまざまな検索言語で部分仕様として採用されている．現在，XML に対する最も基本的な検索言語である．

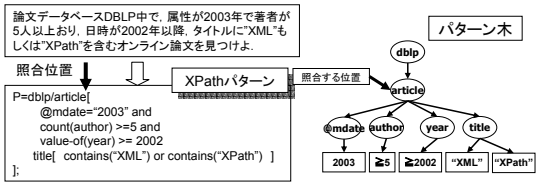

図 2　XPath 問い合わせの例

一方，新しい大規模データ応用として，さまざまなデータが無制限に流れる高速なデータストリームにおいて検索・変換・蓄積を行うためのストリーム情報処理が注目されている [10]．特に，XML 等の**半構造データストリーム**に関しては，2000 年頃から入力 XML データをいったん主記憶に展開することなく検索を

行うオンライン XML 検索の研究が行われている [6], [14], [16]. これらのオンライン XML 検索技術は，XPath 言語の部分クラスを対象にしており，単純なパス照合や木パターン照合といった基本的な XML 検索を行うには十分が，集約や変換など，より複雑な XPath 処理の効率よい実現法はまだ明らかでない．

そこで，われわれは半構造データストリームに対して，検索・統計・再構成などの複雑な**半構造データストリーム処理**を実現するための研究を行っている [11]．本稿では，先に開発した単純な XPath を対象とするオンライン XPath 処理系 XMatch [11] に，従来のオンライン XPath 処理系では扱いが難しいパターン定義と，値抽出，計数・集約演算，テキスト照合，複数問合せ，再構成などの機能を導入する．

これらの機能を実現する上での困難さは，出力ストリームや記憶域等の環境への副作用を伴う点である．改訂した XMatch の基本的なアイディアは，有限状態機械による高速なテキスト走査と，イベント駆動の抽象機械による問合せ実行の結合である．これを，副作用を遅延実行するための機構と組み合わせて，上記の機能をもつ複雑な問合せを高速に実行する．

このような XML ストリームに対するオンライン情報処理技術は，ストリーム処理だけでなく，即時性と多様性が求められるさまざまな種類の半構造データ処理において，今後重要性を増すと思われる．例えば，オンデマンドのビデオサービス等の PUSH 型コンテンツ配信サービスでは，(1) 複数のユーザが発信する (2) 多様で複雑な問合せを (3) 毎時更新されるデータに対して，(4) 短い反応時間で処理することが要求されるため，この種の技術が有効だと思われる．

XMatch のようなオンライン XML 検索では，Semi-join や，ancestor 軸への検索等，読み戻しを要する演算は本質的に苦手であり，この意味で W3C による XPath の全仕様を厳密に満たすことは難しい．そのため，従来の XPath 応用をそのまま置き換える可能性は少ないと思われる．しかし XMatch の潜在的な応用として，例えば，従来の主記憶型の検索／変換エンジンのバックエンドや，XML を直接格納するデータベース (Native XML DB) のアクセス手段として用いるような応用が考えられる．XMatch のようなオンライン XML 検索エンジンがどのような機能と機構をもつべきかは，今後，このような応用を通じて考えていくべきだろう．

本稿の構成は次のとおりである 2 節では，準備として XML テキストと XML 木，パス文脈等の定義を行う．3 節では，XMatch が扱う言語の定義を行う．4 節では，XMatch システムの詳細について述べる．5 節では，実験結果を示す．6 節では，まとめ今後の課題について述べる．

2. 準　　備

2.1 XML テキスト

自然数の集合を $\mathbf{N}$ で表す．Σ を文字のアルファベットとする．$\mathcal{T} = \Sigma^*$ をテキストの集合，$\mathcal{N} \subseteqq \Sigma$ を名前の符号化の集合とし，$\mathcal{V} \subseteqq \Sigma$ を数の符号化の集合とする．ただし，名前 $n \in \mathcal{N}$ は，英字または英数字と文字 {-, _} からなる文字列である．値 $v \in \mathcal{V}$ としては整数値だけを扱い，通常の 10 進記法を用いて符号化する．値の全体集合を $\mathcal{D} = \mathcal{V} \cup \mathcal{T}$ で表す．

このとき，**XML テキスト**とは，次の生成規則をもつ文法で，非終端記号 X から生成される文字列である．

$$
\begin{array}{lcll}
X & := & E \cdots E \mid v \mid t & (v \in \mathcal{V}, t \in \mathcal{T}) \\
E & := & \texttt{<}n\ A\texttt{>}\ X\ \texttt{</}n\texttt{>} & (n \in \mathcal{N}) \quad \% \text{ 要素} \\
A & := & \varepsilon \mid \sqcup a\texttt{="}t\texttt{"}\ A & (a \in \mathcal{A}, s \in \mathcal{T}) \quad \% \text{ 属性}
\end{array}
$$

ここに，E から生成される文字列を**要素**（element）といい，$v \in \mathcal{V}$ を**値**(value) と，$t \in \mathcal{T}$ を**テキスト**(text) という．この定義は，最上位として要素列を許している点と，混合要素を許さない点を除き，通常の XML にしたがう．読みやすさのため，空白 $\sqcup$ や改行，タブなどの空白記号を，XML テキストに適当に挿入してよい．

2.2 XML 木，パス文脈，アドレス

本稿の定義では XML テキストは通常と異なり，根を複数もつ**ラベル付き順序森** (labeld ordered forest, Hedge [12]) に対応する．

XML テキストに対応する**ラベル付き順序森**は，順序木の列である．各節点はラベルとして対応するタグ名 $n \in \mathcal{N}$ と，兄弟中の順番を示す子番号 $c \in \mathbf{N}$ をもつ．属性$@n$ $(n \in \mathcal{N})$ は，名前$@n$ をもつ節点として考える．一対のタグで囲まれたテキストや属性値は，そのまま節点に関連付けられた値とみなす．ただし，長兄（最左の子）が子番号 $c = 1$ をもつとする．節点全体の集合を dom と書く．

XPath では，問合せの評価をおこなう現在の節点 $v \in dom$ を**文脈節点**(context node)　という．節点 v の**パス文脈**と**アドレス**は，それぞれ，根から v までのパス上の名前の列および子番号の列と定義する．アドレスが現在の節点を一意に決定するのに対して，パス文脈はそうではないことに注意されたい．

XML テキスト $X = x_1 \cdots x_n$ $(n \geqq 0)$ に対して，その任意の前綴り $X = x_1 \cdots x_i$ $(1 \leqq i \leqq n)$ を**テキスト文脈**と呼ぶ．これは，XML 木 T 中の節点 v と一意に対応する．このとき，X パス文脈を，v のパス文脈と定義する．

3. 言　　語

本節では，XPath プログラムと呼ぶ問合せ言語を導入する．表 2 にわれわれの問合せ言語の構文を与える．

XPath プログラムは，軸として子供軸と子孫軸のみをもつ XPath をもとに，パターン定義文と出力指定文を加えて拡張したものである．XPath プログラムでは，複数の主問合せと，さらに複数の補助問合せを一つのプログラムに記述可能である．

- パス表現は，子供軸 (child axis) /または先祖軸からなる**基本パス**(elementary path) だけに制限する(注1)．さらに，先祖軸はパスの先頭にしか現れない．例えば，`dblp/article` や `//www/author` は基本パスである．(注2)

(注1)：これは XQuery で用いられている XPath に近い
(注2)：パス途中に出現する先祖軸は，コンパイル時に後述のパターン定義を用いて二つのパターンに分割する．

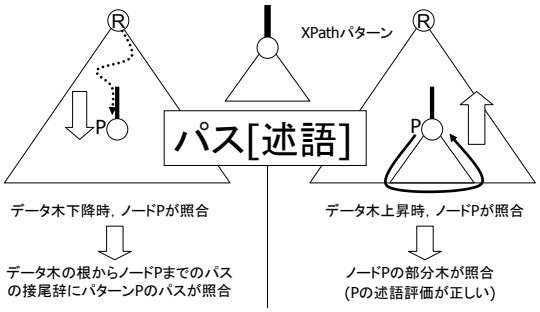

図 3　XPath パターン照合

- **パターン**P は，$Path$ [$Expr$] => $PrintExpr$ の形の表現である．ここに，$Path$ は基本パスであり，$Expr$ は，パターンと論理式，数式等から再帰的に構成される評価式である．表現 => $PrintExpr$ は省略可能な出力命令であり，パターンが成功すると現在の要素または値をバッファに出力する．ただし，上位のパターンがどこかで失敗すると，出力は取り消される．
- 現在の文脈節点 v における**パターン** P **の評価**は，パス文脈評価と部分木評価の二つからなる（図 3）．

すなわち，文脈節点を x として P が**ある節点** y **を選択する**のは，y の上方のパス文脈のある後綴りに $Path$ が照合し，かつ y を根とする部分木が述語 $Expr$ を満足することである．このとき，$Expr$ の部分式として出現する副パターン Q_i $(i = 1, \ldots, n)$ が部分木において評価され，その結果を用いて $Expr$ が y で評価される．

- プログラムは，文のならびである．**呼び出し文** match P は，パターン P を最上位の問合せとして，XML 文書の根を文脈節点として評価する．各呼び出し文は，互いに透過的に実行される．**定義文**N = P は，パターン名 N にパターン P を結びつける．一つのパターン名に複数のパターンが結び付いても良い．このとき，それぞれは互いに透過的に N に作用する．

［例 1］　次の表 1 に，XPath プログラムの例を示す．

表 1　XPath プログラムの例

```
 1  match dblp/inproceedings
 2  match dblp/inproceedings[year and author]
 3  match dblp/article[value-of(year) >= "1965"]
 4  match //inproceeding[@mdate="2002"]
 5  match //article["Web" or "XML"]
 6  match dblp[$P]
 7  $P = article[$Y and count($A)>1] => element-of()
 8  $P = www[$Y and count($A)>1] => element-of()
 9  $Y = year[value-of() >= 1995]
10  $A = author
```

上の 1 行目から 5 行目までは，それぞれ独立した問合せであり，順に単純なパス，述語による分岐，値切り出し，子孫軸と属性，子孫軸と（複数）文字列パターン照合を用いた例である．ここに，5 行目の問合せで，定数文字列"XML"は部分文字列照合 contains("XML") の略記である．6 行目から 10 行目までは，複合問合せ「1995 年以降で 2 人以上の著者をもつ雑誌論文またはオンライン論文を出力せよ」を表す一つの主問合せと複数の副問合せからなる組である．7 行目と 8 行目の二つのパターンが一つのパターン名$P を定義していることに注意．

表 2　XPath 問合せの構文．

%%*XPath Patterns*

Pattern	:=	*Path* [*Expr*]	
	|	*Path* [*Expr*] => *PrintExpr*	% 出力文
	|	*Name*	% 呼び出し
Path	:=	*ElemPath* | //*ElemPath*	
ElemPath	:=	ε | *Name* / *Elem*	

%%*Expressions*

Expr	:=	*Pattern*	
	|	*Expr* and *Expr* | not *Expr*	
	|	*Expr* or *Expr*	
	|	*Arith Rop Arith*	
		$(Rop \in \{=, \neq, \leqq, \geqq, <, >\})$	
	|	contains(*Str*)	$(Str \in \Sigma^*)$
Arith	:=	*Integer* | *Float*	
	|	*Arith Aop Arith* $(Aop \in \{+, -, *, \mathtt{div}, \mathtt{mod}\})$	
	|	value-of(*Pattern*)	
	|	count(*Pattern*)	
	|	sum(*Pattern*) | avr(*Pattern*)	
	|	position()	
PrintExpr	:=	text-of() | element-of()	% 出力式
	|	*Name* [*PrintExpr*]	

%%*XPath Programs*

Program	:=	ε | *Sentence Program*	% プログラム
Sentence	:=	*PatDef* | *PatCall*	% 文
PatDef	:=	*Name* = *Pattern*;	% 定義
PatCall	:=	match *Pattern*;	% 呼び出し

4. XMatch システム

前節の XPath パターン照合を実現するために，我々は XMatch システムを開発した．図 4 に構成図を示す．

システムは，XPath パターンを入力としてもらうと，XPath パターンを解析し，照合機械と抽象機械を生成する．その後，この 2 つの機械を実行しながら，XPath 処理を行う．

4.1　XML 走査器

XML 走査器とは，XML 文書を順次読み込みながら，解析を行いイベントを検出するものである．ここでイベント e とは，組 ($type$，$name$) からなる．図 5 に XML 文書走査器による解析の様子を示す．

- type:イベントタイプ．type ∈ {stag,etag,satt,eatt,cstr,aval}
- name：イベント名

ここで，stag,etag,satt,eatt,cstr,aval をそれぞれ，開始タグイベント，終了タグイベント，開始属性名イベント，終了属性名イベント，属性値イベント (@attr=”aval”)，定数文字列パターンイベント (contains(“Str”)) とする．

われわれはこの XML 走査器として，**ASAX(Active SAX，Alternative SAX)** を開発した．ASAX は文字列パターン照

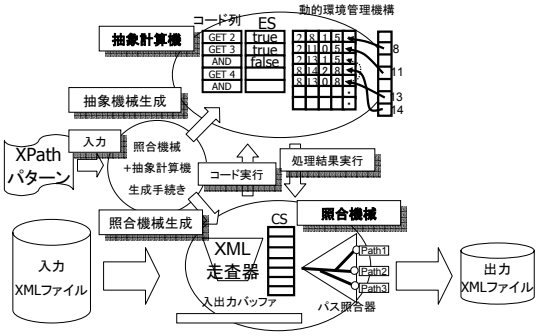

図 4　XMatch システムの構成図

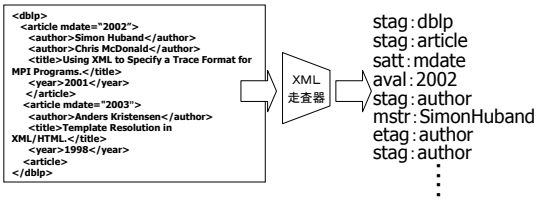

図 5　XML 走査器

合技術 [3] を用いたプログラム可能な XML 走査器であり，入力 XML 文書を左から右へ一回走査する間に，パターンが含むすべてのタグ，属性名，属性値，定数文字列パターンのすべての出現を検出する．

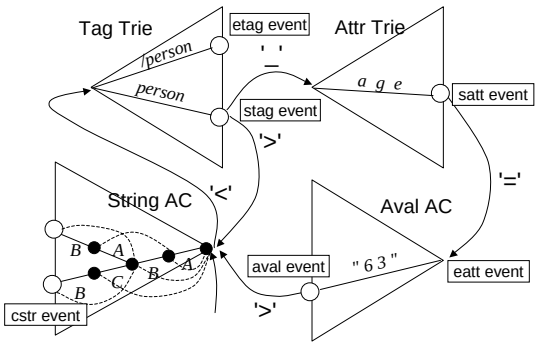

図 6　ASAX の構成図

ASAX では図 6 のように 4 つクラスの有限状態機械 (パターン照合機械) を切り替え器と組み合わせて，プログラム可能な XML 走査器を実現している．タグと属性名はトライ (前綴り木) を用いて照合し，属性値と定数文字列パターンは，AC 機械 [2], [3] を用いて照合する．ASAX は開始タグや終了タグ，属性名，属性値，定数文字列パターンのはじまりや終わり，区切りを表す特別な記号'>'，'<'，'='，'@'，'"' によって，これら 4 つの機械を切りかえながら逐次的に走る．

4.2　イベント管理スタック (Control Stack) とパス照合器 (パス AC 機械)

パス照合器はスタック付き有限状態機械であり，XML 走査器で検出されたイベントを管理スタックで次のように管理しながら，状態遷移を行い，XPath パターンに含まれる基本パスの検出を行う．(図 7)

- if type ∈ {stag,satt} then 管理スタックへイベントをプッシュする．
- if type ∈ {etag,eatt} then 管理スタックからイベントをポップする．

つまり，パス照合器は，入力 XML 文書を左から右へ一回の走査する間に，現在まで読んだ XML 文書 (の前綴り) に対応する仮想的な文脈節点を y に対して，その y のパス文脈の末尾に後綴りとして出現するようなパスをすべて検出するものである．

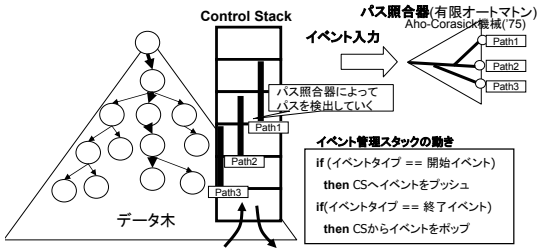

図 7　イベント管理スタックとパス照合器

4.3　問い合わせ木 (Code Tree)

パス照合器で検出する連続パスの XPath パターン上の呼び出し関係 (親子関係) を表した木である．図 8 は，パターン $Path[Q_1 ... Q_n]$ に対する問い合わせ木である．各ノードには，その子パターンに対応する変数列と命令列がついている．ここで命令列 $Code$ は次の 2 組からなる列である．

- type：命令の種類
- data：命令によって処理されるデータ．

抽象機械はこの変数列を参照しながら，命令列を実行して評価することにより，XMatch はさまざまな XPath 処理を行うことが出来る．命令列は次の 4 種類からなる．

- pre：パスが照合した時に実行する命令列．
- post：子パターンの評価が全て終わった時に，親の述語をすべて評価するために実行する命令列．
- succ：post 命令列の実行後，述語評価が正しければ実行を行う命令列．
- fail：post 命令列の実行後，述語評価が正しくなければ実行を行う命令列．

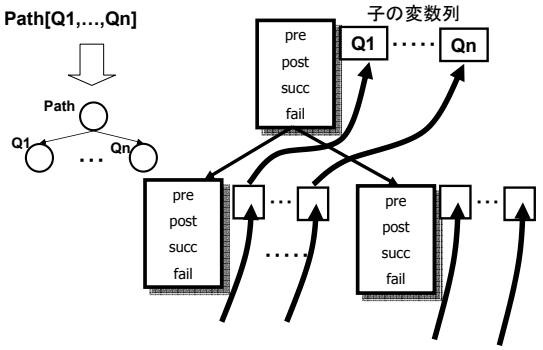

図 8　問い合わせ木

また，値の評価のために，抽象機械は評価スタックを持つ．問い合わせ木の変数列からスタックに値を積み込んだり，ス

タックから値を変数列にストアしながら，命令列の評価を行う．評価スタックには，整数値だけでなく，実数値や文字列値を積むことが出来る．

4.4 動的な変数管理機構：Frame Stack

問い合わせ木の各ノードがもつ子変数列を動的に管理するために，Frame Stack を用いる．問い合わせ木の各ノードは Frame Stack に自分の子変数列をもつことができ，これを自分の**変数フレーム (Activation Record)** 呼ぶ．変数フレームは，フレームの子変数に対応した局所変数列 (**変数セル**) を持つ．変数セルは次の 5 組からなる．

- frame：変数フレーム番号
- cell：変数セル番号
- value：変数セルの値．整数値，実数値，文字列値が入る．
- depth：データ木中の深さ．
- last：変数セルのひとつ前のエントリへのポインタ．

変数インデックス (Variavble Index) は，Frame Stack 中の最新の変数セルのエントリを指している．これにより，最新の変数セルへ定数時間でアクセス出来る．last は，一つ前の変数セルの出現へのポインタである．(図 9)

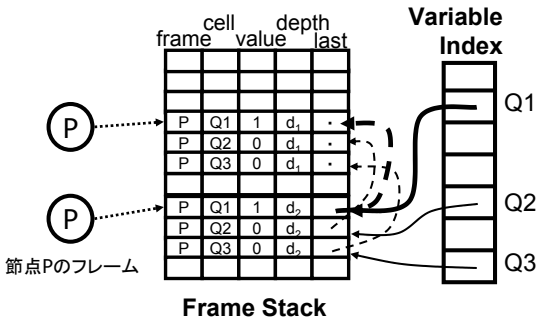

図 9　Frame Stack

4.5 抽象機械の命令

抽象機械は，XPath パターン照合を行うために次の命令をもつ．

4.5.1 メモリ操作命令

メモリ操作命令は Frame Stack 上の変数を管理するための命令である．メモリ操作命令は次の命令になる．ここで変数フレーム P の変数セルを Q_1, Q_2, Q_3 とする．(図 10)

- F_ALOC Q_1：Frame Stack へ変数セル Q_1 を割り付ける．
- GETL Q_1：変数セル Q_1 の値を評価スタックへロードする．
- PUTL Q_1：評価スタックから Frame Stack の変数セル Q_1 へ値をストアする．
- F_DALOC P：変数フレーム P を解放する．

4.5.2 式評価命令

次の命令は，パターンの述語部において式を評価するための命令である．ここで，評価スタックの頂上の値を$1，その下の値を$2 と書く．戻り値がある場合は，戻り値はスタックの頂上にプッシュするものとする．

- AND：$1 と$2 の論理積を計算する．

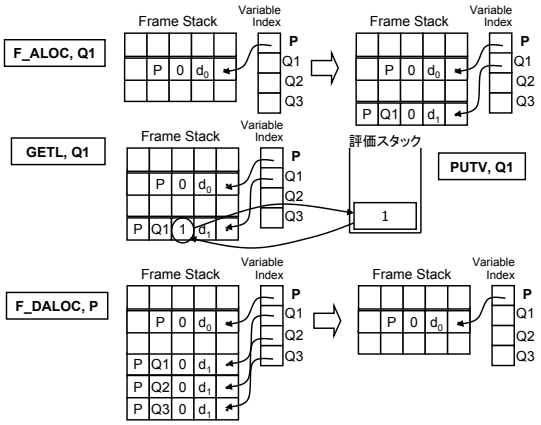

図 10　メモリ操作命令

- OR：$1 と$2 の論理和を計算する．
- NOT：$1 の NOT 演算を行う．
- ADD(+)：数値計算$1+$2(加法演算) を行う．
- SUB(-)：数値計算$1-$2(減法演算) を行う．
- MUL(*)：数値計算$1*$2(乗法演算) を行う．
- DIV(/)：数値計算$1/$2(除法演算) を行う．
- MOD(%)：数値演算$1%$2(剰余演算) を行う．
- EQ(==)：式$1==$2 を行い，真偽を返す．
- NEQ(!=)：式$1!=$2 を行い，真偽を返す．
- LT(<)：式$1<$2 を行い，真偽を返す．
- GT(>)：式$1>$2 を行い，真偽を返す．
- LEQ(<=)：式$1<=$2 を行い，真偽を返す．
- GEQ(>=)：式$1>=$2 を行い，真偽を返す．

4.6 XPath パターン照合のための抽象命令へのコンパイル

XMatch システムは，与えられた XPath パターンを解析し，前節で述べた抽象命令を用いることにより，XPath パターン照合を行う命令列をコンパイルする．例えば，XPath パターン P=article[author and title and year] は，図 11 の問い合わせ木 (左) と，命令列 (右) にコンパイルされる．命令列は article に対する抽象命令列である．

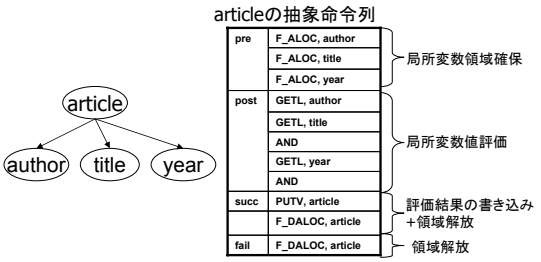

図 11　問い合わせ木 T(左) と命令列 C(右)

この命令列は，次のように動作し，パターン P の照合を行う．

(1)　データ木下降時にパス照合器が基本パス article を検出 (article の stag を検出) した時，pre 命令列を実行する．pre 命令列では，article の子パターンである author，title，year の変数領域を F_ALOC 命令で，Frame Stack 上に割り付ける．

(2)　データ木上昇時にパス照合器が article を検出 (article

の etag を検出) した時 (この時点で子パターンである author, title, year の評価が終わっている), post 命令列を実行し，述語評価を行う．まず 2 つの GETL 命令で author, title の値を評価スタックに積み，AND 演算を行う．次に 3 番目の変数である year を評価スタックに積み AND 演算を行う．

（a） もし post 命令列の評価結果が正しければ，succ 命令を実行する．この場合は article の変数領域に PUTL 命令で値を書き込み，F_DALOC 命令で article の変数フレームを解放する．

（b） もし post 命令列の評価結果が正しくなければ，fail 命令を実行する．この場合は article の値の更新は行わず，F_DALOC 命令で article の変数フレームを解放するのみである．

この処理を再帰的にパターンに適応していくことにより，XPath パターン照合を実現できる．

4.7 出力機構

出力は，XML 処理を行う上で重要な問題となる．XPath ではパターンがある節点に照合した時，その部分木全体またはその直下のテキスト値を出力する．XMatch ではこの出力を**出力スタック**と**出力バッファ**を用いて管理する．

出力スタックは，データ木下降時，出力対象ノードに到達すると，出力バッファの現在のカーソル位置をプッシュする．データ木上昇時に出力対象ノードに到達すると，出力スタックから出力カーソルをポップする．

出力命令には次の抽象命令がある．

- O_PUSH：出力スタックへ出力バッファのカーソル位置をプッシュする．
- O_POP：出力スタックから出力バッファのカーソル位置をポップする．
- WRITE：バッファにストリームを書き込む．
- CANCEL：バッファをキャンセルする．
- WRITE：バッファをフラッシュする．

WRITE 命令について，出力形式は次の 3 種類の指定ができる．

- XPath 式
- XML 文書
- ストリーム中の出力対象となる開始位置と終了位置の組

図．12 に，XPath パターン
$dblp/article/author[contains(\text{“}Claus - Rainer\text{''})]$
に対する実際の出力の様子を示す．ここでは，すべて同じ対象ノードに対する出力を示している．

図．13 は，出力の動きを表した図である．

右上はパターン木，右下はデータ木 (上は失敗例，下は成功例)，左はバッファの様子を表している．パターン木について，出力対象ノードは paper，title，author である．paper ノードを**照合ノード**といい，出力バッファをフラッシュするかキャンセルするかを判断するノードである．title，author ノードは**出力ノード**といい，ノードに到達するとバッファにその部分木の書き込みを始める．

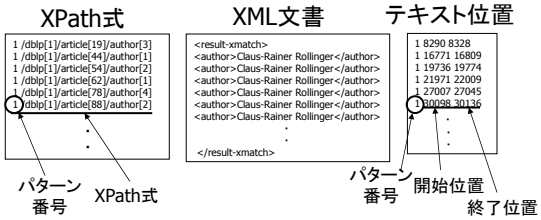

図 12　実際の出力の様子

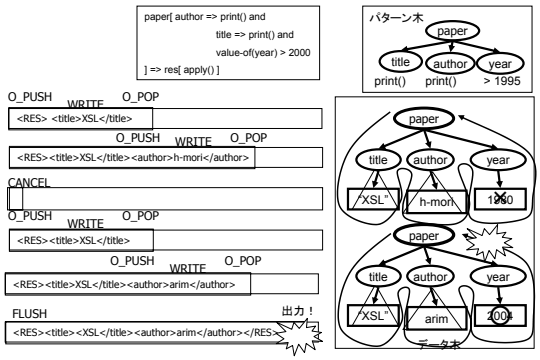

図 13　出力処理の様子

データ木上の失敗例について，まず title，author ノードでそれぞれの部分木をバッファに保存する．次に year ノードでは評価が正しくない．照合ノード paper に戻ってきた時に評価を行うが，この時 post 命令列は失敗する．post 命令列が失敗したら，バッファをキャンセルしなければならないので，CANCEL 命令を実行しバッファ内をキャンセルする．一方，データ木上の成功例について，同様に title，author ノードでそれぞれの部分木をバッファに書き込む．照合ノード paper に戻ってきた時，今度は year 評価は成功しているので，paper ノードでの post 評価は正しいことになる．この時，バッファをフラッシュし，出力を行う．フラッシュには，FLUSH 命令を用いる．

このように，出力スタックと出力バッファを用いることにより，オンライン処理を行いながら，効率よく遅延処理を組み込み，出力処理を行っている．

4.8 パターン変数機構

XMatch ではパターン変数機構を用いて，一つの問い合わせを一つのトップレベル問い合わせといくつかの部分問い合わせに分割して，記述することができる．このため，match 宣言によりトップレベル問い合わせを宣言する．match 宣言を行わなかったパターンは部分問い合わせである．また，パターン定義 $P=Pattern によって，パターン同士の呼び出し関係を記述できる (図 14)．これにより，プログラムのモジュラリティが高くなり，段階的なプログラム開発が行える．さらに共通する部分問い合わせをまとめることができ，効率よい実行が可能となる．

4.9 その他の XPath 処理

現在 XMatch では上の 4.1 から 4.8 で述べた機能を用いて，次の XPath 処理を実装している．

- 軸：子供 (child, /) 軸，子孫軸 (descendant, //)
- ラベル：タグ，ラベル変数，...
- パス：tag/tag/tag

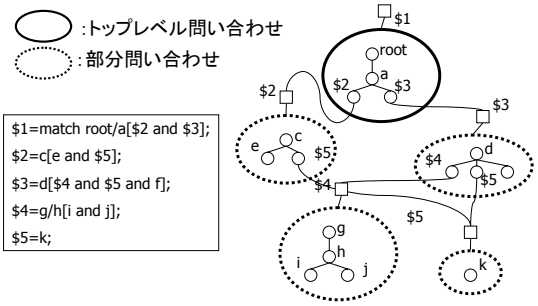

図 14　パターン変数機構を用いたパターン

- 属性：@attr
- 述語：[式]
- – and，or，not
- – >，<，<=，>=，+，−，∗，div, mod
- パターン定義とパターン変数：$p
- 文字列照合：contains("Str")
- 計数と集約：count()，sum()，avr()
- 位置演算：potision()
- 値参照：value-of()，text-of()
- 任意の位置での出力：print()

今回触れることの出来なかった文字列照合，計数と集約，位置演算，値参照等については別の機会に述べたい．次は現在未実装であるが，上記の機構を用いて実装可能なものである．

- 軸：親 (parent, ../) 軸
- 途中に述語を含むパス：tag[式]/tag[式]/tag[式]/...
- 位置定義：first()，last()

現在の XMatch の実装では，XPath における先祖軸 (ancestor) や位置指定の last()，XSLT における自由な再出力機構，XQuery における for 文，let 文，where 文等の機能の実現は難しい．

5. 実　　験

5.1 データと実験手順

XMatch システムを Java1.4.2 で実装し，評価実験を行った．XML データには，オンライン論文データベース DBLP(170MB) (注3) を用いた．実験では XMatch を，公開されている Peng 等 [?] のオンライン XPath 検索システム XSQ (注4) と比較した．実装は Java で，XML 走査器には SAX (Xerces) が使われている．実験環境は，PC (Pentium4, 2.4GHz, RAM 768MB，Red Hat Linux,Java SDK1.4.2) 上である．同じ問合せを 3 回ずつ実行し，平均時間を測定した．

5.2 実 験 結 果

規模耐性．図 15 に，規模耐性のグラフを示す．問合せは，表 3 の Q5 を用いた．計算時間はすべて線形に増加しており規模耐性が良い．図には，オフライン主記憶型の Apache Xalan+DOM (Java2 JAXP) による結果も示す．XMatch

(注3)：http://dblp.uni-trier.de/

(注4)：http://www.cs.umd.edu/projects/xsq/

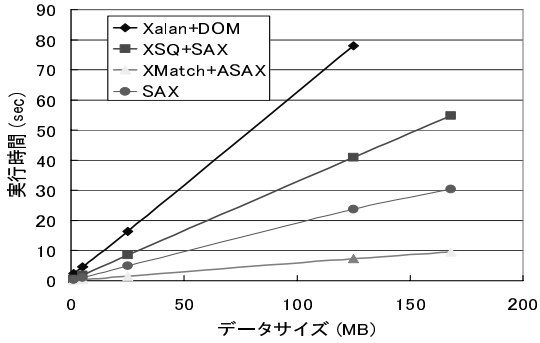

図 15　データサイズに対する計算時間の比較

+ASAX は，Xalan+DOM に比べて 10 倍，XSQ+SAX に比べて 4 倍程度高速である．

データサイズ N に対する主記憶領域サイズ $S(N)$ は，Xalan+DOM では実験的に $6N$ 必要で，$N = 125$MB に対して $S(N) = 728$MB だった．XMatch と XSQ では，測定した全サイズでそれぞれ 5MB と 4MB 前後と著しく小さかった．以上は，XMatch や XSQ のようなオンライン XML 検索システムの優位性を示す．

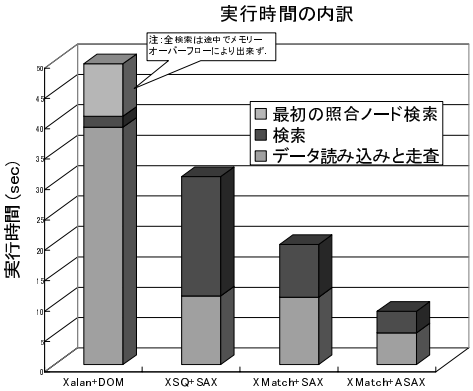

図 16　実行時間の内訳

表 3　実験で用いた XPath パターン

```
Q1  match dblp/inproceedings
Q2  match //article
Q3  match dblp/inproceeding[@mdate="2002"]
Q4  match count(dblp/inproceedings)
Q5  match dblp/inproceedings[year and author]
Q6  match dblp/article/publish[contains("XML")]
Q7  match dblp/inproceedings[title and author => print()]
Q8  match dblp/article[$1 and count($2) > 5 and $3]
    match dblp/inproceedings[$1 and count($2) > 5 and $3]
    match dblp/www[$1 and count($2) > 5 and $3]
    match $1=title[contains("XML")]
    match $2=author
    match $3=year[value-of() >= 2002]
```

計算時間の内訳．次に，図 16 に，サイズ 125(MB) のときの計算時間の内訳を示す．問い合わせは表 3 を用いた．ここでは，ASAX の効果を検証するために，XMatch と SAX の組み合わせも計測した．図から第一に，オフラインの Xalan+DOM は DOM の構築だけで，オンラインの XSQ と XMatch 全体

より大きな時間を費やしていることがわかる．第二に，XML 走査器を SAX に変えても，XMatch は XSQ の 1.5 倍程度高速である．XMatch+SAX と XMatch+ASAX を比較すると，XMatch の本体部分の速度が 2 倍以上向上していることがわかる．これより ASAX は，オンライン XPath 検索に有効な設計であるといえる．さらに ASAX の単機能性が，XMatch の高速性の唯一の理由ではないといえる．

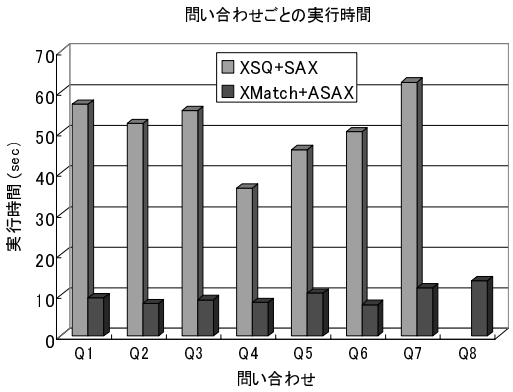

図 17　異なるタイプの問合せに対する計算時間

最後に図 17 に, タイプの異なる問合せに対する計算時間を示す．これより，XMatch+ASAX が簡単なものから複雑な問合せまで，従来の手法よりも，高速に安定して処理できることが分かる．

6. 終わりに

本稿では，単純な XPath プログラムを対象としたオンライン XPath 処理系 XMatch について報告した．特に，半構造データストリームに対して，検索・統計・再構成などの複雑な**半構造データストリーム処理**を実現するための言語系と XML 走査系，抽象計算機系について解説した．実験では，オフラインおよびオンラインの XPath 処理系 Xalan および XSQ とその効率を比較した．実験では，ASAX 単体で SAX の 2 倍程度，XMatch+ASAX で XSQ+SAX の 4 倍程度高速であった．

XMatch の高速性は ASAX の機能の制限にだけ起因するかにみえるがそうではない．例えば，ASAX は SAX と異なり，名前空間を実装しておらず，開始タグと終了タグの対応程度の最小限の検証しかしていない．しかし，XMatch と組み合わせた時の ASAX の高速性は，これらの機能の省略によるものだけではない．ASAX では，プログラム可能な有限状態機械を用いることで，パターン中に出現するタグかトークンに制限した検出を行うことができる．さらに，広いクラスの属性テスト (@attr=”aval”) や，定数文字列照合 (contains(“Str”)) を有限状態機械を用いて下位のレベルで実行できるため，SAX のように上位レベルの処理のために，すべての属性とテキストを入力データから切り出して，対応する Java オブジェクトを生成することを回避できる．これらの工夫は，XMatch の高速化に貢献していると考えられる．

XMatch に対する拡張としては，XML 名前空間への対応や連想記憶 (ハッシュ配列) などの組み込みによる複数データストリームの結合などである．これらに関しては，今後の課題としたい．

文　献

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

[2] A. V. Aho, Algorithms for Finding Patterns in Strings, Handbook of Theoretical Computer Science, 1990.

[3] S. Arikawa et al., SIGMA: A text database management system, Proc. Berliner Informatik-Tage, 1989

[4] S. Chandrasekaran, M. J. Franklin, Streaming Queries over Streaming Data, Proc. VLDB ’02, 2002

[5] G. Gottlob, C. Koch, Monadic Datalog and the Expressive Power of Languages for Web Information Extraction, Proc. PODS’02, 2002.

[6] A. Gupta, D. Suciu, Stream Processing of XPath Queries with Predicates, Proc. SIGMOD ’03, 2003

[7] C. M. Hoffmann and M. J. O’donnell, Pattern Matching in Trees, Journal of the ACM, 1982

[8] Java 2 Platform, Standard Edition (J2SE). http://java.sun.com/

[9] 喜田拓也、宮本哲、竹田正幸, 文字列照合技術に基づく XML データ処理, Proc. FIT ’02, 2002

[10] N. Koudas, D. Srivastava, Data Stream Query Processing: A Tutorial, VLDB 2003, 1149, 2003.

[11] 森川裕章，浅井達哉，有村博紀, データストリーム処理のための効率良い XPath 問合せ機構, 夏のデータベースワークショップ (DBWS2003)，28, 情報処理学会，網走湖畔温泉，July 2003.

[12] M. Murata, Extended Path Expressions for XML, PODS’01, 126–137, 2001

[13] F. Neven and T. Schwentick, Expressive and efficient pattern languages for tree-structured data, Proc. PODS 2000

[14] F. Peng, S. S. Chawathe, XPath Queries on Streaming Data, Proc. SIGMOD ’03,2003. (システム http://www.cs.umd.edu/projects/xsq/ で公開．)

[15] M. Takeda, S. Miyamoto, T. Kida, A. Shinohara, S. Fukamachi, T. Shinohara, S. Arikawa, Processing Text Files as Is：Pattern Matching over Compressed Texts, Multibyte Character Texts, and Semi-structured Texts, Proc. SPIRE ’02, 2002.

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

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

[18] World Wide Web Consortium. XML Path Language (XPath) version 1.0, W3C Recommendation 16 November 1999. http://www.w3.org/TR/XPath