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

2005－FI－78（10）
2005－DD－49（10）
2005／3／25

ストリーム指向のXQuery処理システムについて

石野 明* 竹田 正幸†‡

要旨. XML データストリームに対して一方向逐字処理技術に基づき XQuery の部分族を高速に処理するシステムについて述べる．Aho-Corasick のパターン照合機械によって XML データストリームを効率的に走査し，ストリーム走査器の出力系列を入力とした NFA を用いることで XQuery の部分族を高速に処理することが可能である．実装を行ったところ実際の XML データに対して 6.6MB/s という処理能力を得た.

A Proposal for Stream-Oriented XQuery Processing System

Akira Ishino* Masayuki Takeda†‡

Abstract. This paper describes an efficient XQuery processor for XML data streams. Aho-Corasic pattern-matching DFAs and NFAs for evaluating a subset of XQueries process XML data streams very fast: in our experiments we achieve a throughput of about 6.6MB/s for real XML data.

1　はじめに

XML データの様々な分野への応用に伴い，ネットワークの速度で無尽蔵に生み出され続ける XML ストリームデータを対象にし，同じネットワークの速度で処理することが現実に求められるようになってきた.

現在，広く用いられている XML データベースの多くは関係データベースを用い，タグの親子関係や，タグとその内容などの関係をデータベースに格納することで XQuery などの質問式による検索を可能としている [7, 9]．これらの手法では索引構造を作成することによって高速な検索を可能であるが，XML データは本来，データ中にその構造を内包していることが特徴であり，構造が安定していない XML ストリームデータに対して索引構造を保守し続けるのは困難である.

そこで，本稿では XML データストリームに対して一方向逐次処理によって XQuery を処理する手法について述べる．この手法は，XPath の部分族に対する [14] を拡張したものである.

XML データストリームを一方向逐次処理する際に，SAX(Simple API for XML) パーサーが用いられることが多い．しかし，SAX パーサーによるストリーム走査にかかる時間が質問処理時間の大半となることが報告されている [10]．XML データストリームを効率よく走査するために著者らは Aho-Corasick のパターン照合機械 [1] に基づいたストリーム走査器を提案してきた [11, 15, 16].

ストリーム走査器の出力系列を入力とする XPath の質問処理方式としては，XPath に対する有限オートマトンを構築しそれを用いる方式が提案されている [2, 4, 6, 14]．本稿では，XPath を含み，より複雑な質問が可能な XQuery に対して非決定性有限オートマトン (Nondeterministic Finite Automaton; NFA) を構成し，XQuery に関しても効率的に処理が可能であることを示す.

本稿の構成は以下の通りである．2 節では，本稿で扱う XQuery の部分族を定義し，その意味を与える．3 節では，XML データストリームに対して XQuery

*九州大学大学評価情報室・Office for Information of University Evaluation, Kyushu University
†九州大学大学院システム情報科学研究院・Graduate School of Information Science and Electrical Engineering, Kyushu University
‡科学技術振興機構 戦略的創造研究推進事業・SORST, JST

質問式を処理する手法を与え，4 節では，実装し実験を行った結果について述べる．5 節では，本稿で扱う XQuery の部分族と XQuery 1.0 [12] との差異について述べる．

2 XQuery の構文と意味

本稿では XQuery の部分族を扱う．そこで，まずその部分族の構文を定義し，その意味を与える．

2.1 XQuery の構文

本稿で扱う XQuery の部分族を図 1 に示す．XQuery は XPath の定義を内包しており，定義のほとんどは XPath に関するものである．図の定義における *Query* を**質問式**とよび，*Pattern* を**パスパターン**とよぶ．また，図の *Path* を単純パスとよぶ．

ここでは，子軸 (child axis) と子孫軸 (descendant axis) のみを扱い，親軸 (parent axis) と先祖軸 (ancestor axis) は扱っていない．属性軸 (attribute axis) と名前空間軸 (namespace axis) についてはここでは省略するが，子軸と同様に扱うことができる．

Qualifier は単純パスの最後につけることができる条件であり，*Expr* は数や文字列などの値を持つ式である．*Expr* としては他に多くの関数を考えることができるが，本稿では代表的なものとして *count* と *sum* について述べる．

XQuery と主な違いとして，単純パスに対して条件式を最後にしかつけることができないということがある．また，for 文を入れ子にすることもできない．これらの制限については 5 節で考察する．

2.2 XQuery の意味

本稿で扱う XQuery の部分族の意味を，XPath に対する意味を定義した [13, 14] を参考に拡張し定義したものを図 2 に示す．ここでは，XML 文書をランクを持たないラベル付き順序木と考える．質問式 *Query* は，節点 x を受け取り，条件を満たす節点 y の集合を返す関数とみなせる．すなわち，*dom* を木の節点全体の集合とするとき，質問式は *dom* から 2^{dom} への写像を与える．

図 2 において，*Query* は質問式のクラスを，*Qualifier* は条件式のクラスを，*Expr* は式のクラスを，*Boolean* は真偽値のクラスを，*Number* は自然数のクラスを，*String* は文字列のクラスを，それぞれ表す．また，$children(x)$, $descend(x)$ はそれぞれ節点 x の子全体の集合と子孫全体の集合を表す．$name(x)$ は節点 x の名前を，$number(x)$ は節点 x に含まれる数を，それぞれ表す．

3 ストリーム指向の XQuery 処理

本稿ではストリームとして与えられる XML データを一方向に走査しながら，開始タグ，終了タグ，キーワードの検出を行い，それぞれに設定された動作を実行することによって XQuery を処理するイベント駆動による手法について述べる．

3.1 DFA によるストリーム走査

XML データストリームを高速に走査し，タグ文字列の検出と同時に，与えられたキーワードのパターン照合も行いたい．

このような目的のために，複数文字列照合アルゴリズムのひとつである Aho-Corasick 法 [1] を用いる．入力として与えられたタグ文字列やキーワードに対してパターン照合機械 (Pattern Matching Machine; PMM) とよばれる一種の有限オートマトンを構成し，これに XML データストリームを走査させることによって，すべてのタグの出現とキーワードの出現を求める．

しかしながら，タグ文字列は，不定長の空白文字列を含んだり，属性に関する記述を含んでおり，単純な文字列ではない．そのため，Aho-Corasick 法をそのまま適用することができない．

文献 [11] において，著者らは，拡張語頭符号法のもとで表現された文字列に対しても，Aho-Corasick 法が拡張できることを示した．

詳細は省くが，この手法を適用すれば，単一の PMM によって，キーワードとタグの検出を効率的に行うことができる．PMM の状態数および構築時間は，いずれも入力である質問式のサイズに関して線形である．また，属性名や属性値を検出するための拡張も容易である．

図 3 に PMM の状態遷移グラフを示す．図中の点線は failure 遷移を表し，`tag name` の部分は質問式に現れるタグ名によるトライ (Trie) となる．実際の

Query	::=	*Pattern* | *Expr* | `for $`*v* `in` *Pattern* `where` *Qualifier* `return` *Expr*
Pattern	::=	*Path* | *Path*[*Qualifier*]
Path	::=	*Name* | * | *Names* | */Path* | *//Path* | *Path/Path* | *Path//Path*
Names	::=	*Name* | *Names* | *Name*
Qualifier	::=	*Pattern* | *Expr* = *Expr* | *Qualifier* and *Qualifier* | *Qualifier* or *Qualifier* | not Qualifier
Expr	::=	*i* | "*str*" | *Pattern* | *count*(*Pattern*) | *sum*(*Pattern*) | Expr Expr

図 1: 本稿で扱う XQuery の部分族の構文

PMM はキーワードの検出および属性などの拡張と多バイトコードへの対応が行われている．

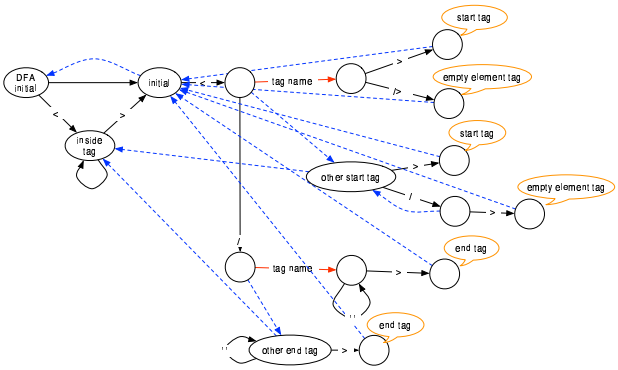

図 3: XML データストリームに対する PMM

3.2 パスパターンのマッチング

前節で述べた DFA によるストリーム走査の結果として出力されるタグの系列を逐次走査することによって，XQuery の質問式を高速処理したい．

そのために，基本となる次の形をしたパスパターンを考える．

$$p_1[p_2]$$

ここで，p_1, p_2 は，いずれも単純パスである．このパスパターンは根からの経路が p_1 である節点 x のうち，x からの経路が p_2 となるものが存在するものを指定している．パス p_1 のことを**親パス**といい，パス p_2 のことを**子パス**とよぶことにする．

このような節点を，DOM 木を構築することなく，ストリーム走査の出力系列を逐次処理することによって，仮想的に DOM 木の節点を巡回する．まず，空スタックを用意する．ストリーム走査によって開始タグが検出されるたびに，そのタグをスタックにプッシュし，終了タグが検出されるたびにポップする．このようにするとスタックの内容は，根から現在いる節点までの経路中の節点の系列となる．この節点の系列を文字列に見立ててパターン照合を行えばよい．

さて，上記のパスパターン $p_1[p_2]$ について考えよう．このパスパターンにおいては 2 つのパターン照合機械が構成される．ひとつは，親パスと子パスを合わせた p_1/p_2 に対するものであり，もうひとつは子パス p_2 の反転 p_2^R に対するものである．前者のパターン照合機械によって p_1/p_2 にある節点 n_2 がマッチしたときに，スタックに積まれた節点の系列を後者のパターン照合機械によって逆に辿ることによって節点 n_2 から p_2^R となる節点 n_1 を見つける．そのような節点 n_1 はすべて根からのパスが p_1 であり，さらにパス p_2 を持つものである．すなわち，n_1 は求めるパスパターン $p_1[p_2]$ にマッチする節点であり，n_2 はパスパターン p_1 からさらに p_2 だけ辿った節点である（図 4）．

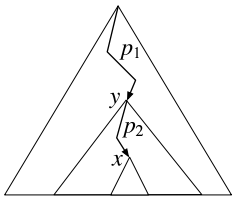

図 4: パスパターン $p_1[p_2]$

2 つのパターン照合機械をまとめて**パスパターン照合機械**とよぶことにする．このとき節点 n_2 に対応したパスパターン照合機械の状態に対して，節点 n_1 に値として真を保存するという処理を設定する．そして，節点 n_1 に対応したパスパターン照合機械の状態に対しては，その節点 n_1 に保存された値が真

$$
\begin{array}{lcl}
\mathcal{S} & : & \mathit{Query} \to \mathit{dom} \to 2^{\mathit{dom}} \\
\mathcal{S}[\![n]\!](x) & = & \{\, y \in \mathit{dom} \mid y \in \mathit{children}(x),\ \mathit{name}(y) = n \,\} \\
\mathcal{S}[\![*]\!](x) & = & \{\, y \in \mathit{dom} \mid y \in \mathit{children}(x) \,\} \\
\mathcal{S}[\![n_1 | n_2 | \ldots | n_t]\!](x) & = & \mathcal{S}[\![n_1]\!](x) \cup \mathcal{S}[\![n_2]\!](x) \cup \cdots \cup \mathcal{S}[\![n_t]\!](x) \\
\mathcal{S}[\![/p]\!](x) & = & \mathcal{S}[\![p]\!](\mathbf{root}) \\
\mathcal{S}[\![//p]\!](x) & = & \bigcup_{y \in \mathit{dom}} \mathcal{S}[\![p]\!](y) \\
\mathcal{S}[\![p_1/p_2]\!](x) & = & \{\, z \in \mathit{dom} \mid y \in \mathcal{S}[\![p_1]\!](x),\ z \in \mathcal{S}[\![p_2]\!](y) \,\} \\
\mathcal{S}[\![p_1//p_2]\!](x) & = & \{\, z \in \mathit{dom} \mid y \in \mathcal{S}[\![p_1]\!](x),\ w \in \mathit{descend}(y),\ z \in \mathcal{S}[\![p_2]\!](w) \,\} \\
\mathcal{S}[\![p[q]]\!](x) & = & \{\, y \in \mathit{dom} \mid y \in \mathcal{S}[\![p]\!](x),\ \mathcal{Q}[\![q]\!](y) \,\} \\
\mathcal{S}[\![e]\!](x) & = & \{\, \mathcal{E}[\![e]\!](\mathbf{root}) \,\} \\
\mathcal{S}[\![\text{for } v \text{ in } p \text{ where } q \text{ return } e]\!](x) & = & \{\, \mathcal{E}[\![e]\!](y) \mid y \in \mathcal{S}[\![p]\!](x),\ \mathcal{Q}[\![q]\!](x) \,\} \\
 & & \\
\mathcal{Q} & : & \mathit{Qualifier} \to \mathit{dom} \to \mathit{Boolean} \\
\mathcal{Q}[\![p]\!](x) & = & \mathcal{S}[\![p]\!](x) \neq \phi \\
\mathcal{Q}[\![e_1 = e_2]\!](x) & = & \mathcal{E}[\![e_1]\!](x) = \mathcal{E}[\![e_2]\!](x) \\
\mathcal{Q}[\![q_1 \text{ and } q_2]\!](x) & = & \mathcal{Q}[\![q_1]\!](x) \wedge \mathcal{Q}[\![q_2]\!](x) \\
\mathcal{Q}[\![q_1 \text{ or } q_2]\!](x) & = & \mathcal{Q}[\![q_1]\!](x) \vee \mathcal{Q}[\![q_2]\!](x) \\
\mathcal{Q}[\![\text{not } q]\!](x) & = & \neg \mathcal{Q}[\![q]\!](x) \\
 & & \\
\mathcal{E} & : & \mathit{Expr} \to \mathit{dom} \to \mathit{Number} \cup \mathit{String} \\
\mathcal{E}[\![i]\!](x) & = & i \in \mathit{Number} \\
\mathcal{E}[\![e_1 + e_2]\!](x) & = & \mathcal{E}[\![e_1]\!](x) + \mathcal{E}[\![e_2]\!](x) \\
\mathcal{E}[\![\mathit{count}(p)]\!](x) & = & |\, \mathcal{S}[\![p]\!](x) \,| \\
\mathcal{E}[\![\mathit{sum}(p)]\!](x) & = & \Sigma_{y \in \mathcal{S}[\![p]\!](x)} \mathit{number}(y) \\
\mathcal{E}[\![\text{"}\mathit{str}\text{"}]\!](x) & = & \mathit{str} \in \mathit{String} \\
\mathcal{E}[\![p]\!](x) & = & y \in \mathcal{S}[\![p]\!](x) \\
\mathcal{E}[\![e_1 e_2]\!](x) & = & \mathcal{E}[\![e_1]\!](x) \cdot \mathcal{E}[\![e_2]\!](x)
\end{array}
$$

図 2: 本稿で扱う XQuery の部分族の意味

であれば節点 n_1 を出力するという処理を設定する．これによって，質問式 $p_1[p_2]$ に対するパスパターン照合機械は，XML データストリームを入力として受け取ると，子パス p_2 を持つパス p_1 の節点を出力とする．これは図 2 で与えた意味と一致する．

同様にして，2 つの節点 n_1, n_2 に対応したパスパターン照合機械の状態に対して，質問式に応じた処理を設定し，XML データストリームを入力した際に各処理が実行されることによって質問式の回答を得ることができる．

そこで，節点 n_1 と節点 n_2 をそれぞれパスパターン $p_1[p_2]$ の**親節点**と**子節点**と呼ぶことにし，それぞれに対応した状態にどのような処理を割り当てればよいかを議論する．

また，各処理が実行される際には親節点に値を保存するということがある．これは，節点が積まれたスタックの対応する場所に値を保存するという操作であり，保存された値はスタックから節点がポップされる際に同時に破棄される．

3.3 質問式と処理

先の例でみたように，質問式に応じて適切な処理をパスパターン照合機械に設定することによって，他の質問式の処理も同様に可能となる．

質問式中で問題となるのは，パスパターンに関し

てである．パスパターン以外の部分については，数や文字列は定数であり，四則演算，論理演算，比較演算のいずれの演算結果もその部分式の値によって定まる値である．パスパターンを含む部分の値が求まれば質問式全体の値が定まる．

パスパターンを含む部分は，条件式としての *Pattern* と，式としての *Pattern*, *count*(*Pattern*), *sum*(*Pattern*)，そして for 文のいずれかである．

条件式としての *Pattern* については 3.2 節でみたように，子節点 n_2 では親節点 n_1 に値として真を設定し，親節点では保存された値に応じて出力を得る処理を設定すればよい．これを，次のように記述することにする．

n_2 : $v(n_1) := true$,

n_1 : if $v(n_1)$ then ...

パスパターンが入れ子になっている場合，例えば，$p[q[r]]$ といった場合は，親パスを p/q，子パスを r として考え，r の子節点 n_2 で p/q の親節点 n_1 に値として真を設定し，親節点では保存された値が真であったとき，さらに元の親パス p の親節点 n_0 に値として真を設定するという処理を行う．このように再帰的に処理を設定することによって同様に扱うことができる．

n_2 : $v(n_1) := true$,

n_1 : if $v(n_1)$ then $v(n_0) := true$,

n_0 : if $v(n_0)$ then ...

次に，式としての *Pattern* については，親節点に保存された値が真であったとき，すなわち条件部が真であったときに，親節点そのものを値とすることによって処理される．

n_2 : $v(n_1) := true$,

n_1 : if $v(n_1)$ then n_1

count(*Pattern*) は，子節点 n_2 においては親節点 n_1 に保存された値を 1 ずつ増加させ，親節点では保存された値を式の値とする．

n_2 : $v(n_1) := v(n_1) + 1$,

n_1 : $v(n_1)$.

sum(*Pattern*) の場合は，子節点において子節点の値を親節点に加算し，親節点では保存された値を式の値とする．

n_2 : $v(n_1) := v(n_1) + number(n_1)$,

n_1 : $v(n_1)$.

代表的な関数として *count* と *sum* を取り上げたが，最大値，最小値をそれぞれ返す *max*, *min* や平均値を返す *avg* なども同様に扱うことができる．

最後に for 文 `for $`v `in` p `return` e は，式 e の値をパス p の親節点 n_1 へと保存し，親節点では保存された値を式の値することによって処理される．

以上のことから，図 1 で示した XQuery の部分族は，パスパターン照合機械を用いることで XML データストリームを入力とし 1 回の読み込みだけで処理される．

4 実装

上記の手法を実際に計算機上に実装を行い実験を行った．本実装の基となった汎用テキストデータベース管理システム SIGMA[3] は C で実装されているが，本実装では Java (J2SE1.4.2) を用い，SIGMA の Java への移植を行った上でさらに拡張を行った．実際の実装では，本稿で述べた以外にも多くの関数に対応し，さらに XQuery 1.0 で定められている XML 要素の直接生成に関しても処理が可能となっている．

実験環境として Gentoo Gnu/Linux 2.6.9, 1.8GHz Pentium 4, メモリ 1GB のマシンを用い，255MB の DBLP[8] の文献データに対して,「2004 年の論文のタイトルを出力せよ」という次の質問式

```
for $p in /dblp/*[year=2004]
return $p/title
```

を実行した結果，55,180 件の論文タイトルが出力され，実行時間は 38.5 秒であった．単位時間あたりの処理能力すなわちスループットは 6.6MB/s となった．この結果は XPath よりも複雑な XQuery を扱うにも関わらず，従来の手法 [6] と比較しても十分に高速であるといえる．

図 5 に入力サイズを変化させたときの処理時間の変化を示す．処理時間は入力サイズに対して線形であり，ストリーム処理を行うための条件を満たしていることが分かる．

しかし，残念なことにスループットは質問式のサイズに依存する．上記の例で`$p/title` を n 繰り返

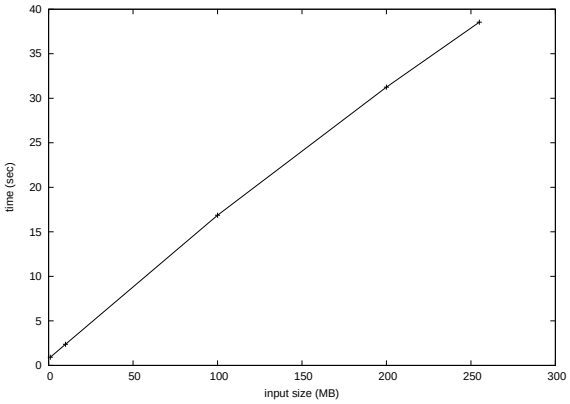

図 5: 入力サイズと処理時間

すことで質問式のサイズを変化させたときの処理時間の変化を図 6 に示す.

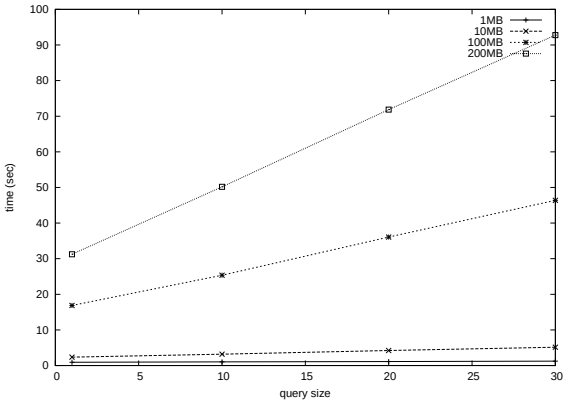

図 6: 質問式のサイズと処理時間

グラフから処理時間は質問式のサイズに比例していることが分かる．これは，質問式に含まれるパスパターンに比例した数だけの NFA が作成され，各タグが出力されるたびにすべての NFA が状態遷移を行うためである．

5 XQuery 1.0 との差異

本稿で扱ったのは XML データストリームを 1 度だけ逐次走査することによって処理することが可能な XQuery の部分族である．その制限にもかかわらず，データの各部を抜き出すことや，統計処理などを行うことができ，十分に実用的な能力を有する部分族である．

しかし，1 度だけの逐次走査では実現することが不可能であると考えられる質問式は取り扱うことができない．ここでは，そういった質問式について考察する．

5.1 入れ子になった for 文

本稿で扱った XQuery の部分族では入れ子になった for 文は扱わなかった．

入れ子になった for 文のうち外側の for 文で指定されているパスの子パスとなるパスが内側の for 文で指定されるパスとなるものに関しては 1 度の操作で処理を行えるため，拡張は可能だと考えられる．

それ以外の場合として，著者と書籍名が文献ごとに記述されているデータから，著者ごとの書籍リストを作成するといった操作や，2 つのデータベースに対して同一の著者のデータを集めるといった操作が考えられる．このような，操作はいずれも結合操作とよばれ，この操作はデータに終わりがある通常のデータベースであれば 1 度目の走査では著者のリストを作成し，2 度目の走査で著者ごとの for 文を実行する必要ことによって容易に処理することができる．しかし，対象がデータストリームである場合は走査が終わるということがないため実装は困難である．

5.2 親軸

親軸を許した場合，注目している親パスが示す節点よりも前の部分を子パスが指定する可能性がある．そのような場合は，入力される XML データストリームを前方から一方向に逐次処理する本手法では取り扱うことができない．また，親軸を含む質問式においては，一般的な手法においても質問式のサイズに対して指数関数的な計算時間がかかる場合があることが指摘されている [5].

5.3 単純ではないパスパターン

本稿ではパスに対する条件式は最後にだけ付加することができた．すなわち，p, r を単純パス，q を条件式としたとき $p[q]r$ といった形のパスパターンは対象としなかった．

しかし，例えば，図 7 の XML データに対して，2001 年の文献のタイトルを指定するにはパス

/dblp/www[year=2001]/title と指定する必要がある．

```
<?xml version="1.0"?>
<dblp>
  <www>
    <editor>Donald D. Chamberlin</editor>
    <editor>Daniela Florescu</editor>
    <editor>Jonathan Robie</editor>
    <editor>J&eacute;r&ocirc;me
              Sim&eacute;on</editor>
    <editor>Mugur Stefanescu</editor>
    <title>XQuery:
             A Query Language for XML</title>
    <year>2001</year>
    <url>http://www.w3.org/TR/xquery</url>
  </www>
</dblp>
```

図 7: XML データの例

ここで，このパスを一方向に逐次処理をすることを考えると，DFA によるストリーム走査の結果得られるタグの系列だけをみると，<dblp>, <www>, <editor>，</editor>，...，<title>，</title>, <year>, </year>, ... となる．このとき，先ほどのパスは title を出力として得るものであるので，</title>の時点であらかじめ設定してあった処理が実行されることになる．しかし，この時点では year に関するタグはまだ現れておらず，year=2001 という条件式は真にはならない．結果として，この title は出力されない．

このように，$p[q]r$ という形のパスパターンでは，条件式 q に子パス r の兄弟でかつ r よりも後に出てくるものが含まれることがある．

この問題に対しては，for 文を用いると解決が可能な場合がある．この例の場合は for $p in /dblp/*[year=2001] return $p/title という質問式によって期待する結果を得ることができる．

しかし，残念ながら，この解決策は常に可能なわけではなく，この問題を根本的に解決するためには，条件式の判定を</title>の時点ではなく，条件式がおかれている</www>の時点まで遅らせ，その時点まで title に関する出力を保留するという処理が必要である．

5.4 FLOWR 形式の Let と Order

本稿では XQuery の FLOWR (For-Let-Order-Where-Return) 形式のうち，For, Where, Return だけを扱った．Let に関してはその中に出てくるパスが for 文のパスを親パスとするものである限りは，syntax sugar にすぎず，容易に扱うことができる．それ以外の場合は 5.1 節の入れ子になった for 文と本質的に同じ問題となる．

また，Order すなわち並び替えに関しては扱わなかった．しかし，対象がストリームデータではなく，特定の大きさのデータであるなら，出力結果をソートすることによって容易に対応が可能である．

6 まとめ

本稿では XQuery がパターン照合機械と NFA を用いて高速に処理することが可能であることを示した．これまでに知られていた XPath に対する有限オートマトンを用いた手法よりも，複雑な質問を記述できる XQuery による質問式を取り扱えるにも関わらず，その処理能力は 6.6MB/s というスループットを達成した．

本稿で扱った XQuery は W3C の定めた XQuery 1.0 の部分族であるが，ストリームデータを対象に処理が可能なよい部分族となっていることをみた．

しかし，大規模な質問式を取り扱うには処理時間が質問式の大きさに依存するという問題を残した．この問題については Green らによる手法 [6] と同様に複数の NFA をまとめることによって解決できる．しかし，状態数が多くなるため，本手法で用いているビット並列化技術による効率的な NFA の実装法を用いることができなくなり，小さな質問式においてはスループットが低下することかもしれない．

今後は，本稿で述べたシステムを基に，大規模なデータベースや XML データストリームからのデータマイニングおよび XML フィルタリングへの応用を行う予定である．

参考文献

[1] Aho, A. V. and Corasick, M. J.: Efficient string matching: an aid to bibliographic search, *Commun. ACM*, Vol. 18, No. 6, pp. 333–340 (1975).

[2] Altinel, M. and Franklin, M. J.: Efficient Filtering of XML Documents for Selective Dissemination of Information, *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., pp. 53–64 (2000).

[3] Arikawa, S., Takeya, S., Miyano, S., Kawasaki, Y. and Inoue, H.: SIGMA: a text database management system, *Information modelling and knowledge bases*, IOS Press, pp. 455–468 (1990).

[4] Chan, C.-Y., Felber, P., Garofalakis, M. and Rastogi, R.: Efficient filtering of XML documents with XPath expressions, *VLDB Journal: Very Large Data Bases*, Vol. 11, No. 4, pp. 354–379 (2002).

[5] Gottlob, G., Koch, C. and Pichler, R.: Efficient Algorithms for Processing XPath Queries, *VLDB 2002*, pp. 95–106 (2002).

[6] Green, T. J., Miklau, G., Onizuka, M. and Suciu, D.: Processing XML Streams with Deterministic Automata, *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, LNCS, Vol. 2572, pp. 173–189 (2003).

[7] Jagadish, H. V., Al-Khalifa, S., Chapman, A., Lakshmanan, L. V. S., Nierman, A., Paparizos, S., Patel, J. M., Srivastava, D., Wiwatwattana, N., Wu, Y. and Wu, C. Y.: TIMBER: A native XML database, *VLDB J.*, Vol. 11, No. 4, pp. 274–291 (2002).

[8] Ley, M.: DBLP Computer Science Bibliography, http://dblp.uni-trier.de/.

[9] Meier, W.: eXist: An Open Source Native XML Database, *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, LNCS, No. 2593, pp. 169–183 (2003).

[10] Suciu, D.: From Searching Text to Querying XML Streams, *International Symposium on String Processing and Information Retrieval (SPIRE '02)*, LNCS, Vol. 2476 (2002).

[11] Takeda, M., Miyamoto, S., Kida, T., Shinohara, A., Fukamachi, S., Shinohara, T. and Arikawa, S.: Processing Text Files as Is: Pattern Matching over Compressed Texts, Multi-byte Character Texts, and Semi-structured Texts, *SPIRE 2002*, LNCS, No. 2476, pp. 170–186 (2002).

[12] W3C: XQuery 1.0: An XML Query Language, W3C Working Draft (2004). http://www.w3.org/TR/xquery/.

[13] Wadler, P.: Two Semantics for XPath (1999). http://homepages.inf.ed.ac.uk/wadler/topics/xml.html.

[14] 竹田正幸, 宮本哲, 石野明, 辻寿嗣: 高速一方向逐字処理技術に基づく XML 文書の検索と変換, 情報処理学会「デジタル・ドキュメント」第 41 回研究会資料 (2003).

[15] 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, 情報科学技術フォーラム 2002 (FIT2002), pp. 55–56 (2002).

[16] 喜田拓也, 貴福友晴, 竹田正幸: 半構造化テキストに対する文字列照合アルゴリズム, 2001 年度冬の LA シンポジウム予稿集 (2002).