2005－DBS－137（Ⅱ）（48）
2005／7／14

インクリメンタルに更新可能なXPushマシン

武川　肇[†1]　片山　薫[†2,†3]　石川　博[†2,†3]

†1職業能力開発総合大学校情報工学科
†2首都大学東京システムデザイン学部
†3東京都立大学大学院・工学研究科

あらまし　XPath フィルタエンジンはフィルタ条件を XML データごとに数多く評価する必要がある．そのため Gupta らは複数のフィルタをボトムアップに結合し，フィルタ条件の数に影響しないオートマトン，XPush マシンを提案した．しかし XPush マシンは更新ができないという未解決問題を持つ．そのため 1 つのフィルタを削除することであっても XPush マシン全体の再計算が必要となる．初期状態の非決定性有限オートマトンに戻った XPush マシンは決定性有限オートマトンへの遅延変換を要求するため，システムのスループットを低下させる．この問題を解決するために，問合せごとに構築したサブ XPush マシンの集合から全体のオートマトンを構築する方式を提案する．Gupta らの方式は XPath 用に拡張した AFA（Alternating Finite Automaton）から XPush 状態を見つけ出す．一方，本方式はこの XPush 状態にキーを追加し，XPush 状態どうしのさらなる結合を実現する．またサブ XPush マシン単位の分割管理構造では状態間に世代関係が現れる．提案方式ではこの関係を使ってインクリメンタルな更新をさらに実現する．

An Incrementally Updatable XPush Machine

Hajime TAKEKAWA[†1]　Kaoru KATAYAMA[†2,†3]　and　Hiroshi ISHIKAWA[†2,†3]

†1Department of information and Computer Science , Polytechnic University, Kanagawa
†2Tokyo Metropolitan University, System Design
†3Tokyo Metropolitan University, Engineering

Abstract　XPath filter engine needs to evaluate many filter conditions for every XML data. Therefore, Gupta et. al. proposed the automaton called XPush machine which does not influence the number of filter conditions by combining two or more filters in a bottom up fusion. However, an XPush machine has the unsolved problem that it is impossible to update. Therefore, even if it is deleting only one filter, the re-calculation of the whole XPush machine is necessary. The XPush machine which returned to NFA of an initial state decreases the throughput of the engine in order to require the lazy translation to DFA. In order to solve this problem, we proposed a method which constructs the whole automaton from a set of sub-XPush machines for each query. Gupta et. al. 's method creates joint states from a set of AFAs (Alternating Finite Automaton) for XPath. On the other hand, we realize a further join function to combine two XPush states by adding a key for each state. Moreover, with the divided management structure of sub-XPush machines, generation relationships appear between states. We realize incremental updating further by using these relationships.

1. はじめに

XML データをコンテンツベースに処理するフィルタ型配信システムや XML ルータはフィルタ条件を XML データごとに数多く評価する必要がある[7]．これらシステムのスループット向上のため Gupta らは，遅延型の単純決定性プッシュダウンオートマトン（simple deterministic pushdown automaton；simple DPDA）に基づき，複数のフィルタをボトムアップに結合した XML フィルタエンジン用オートマトン，XPush マシンを提案した[8]．

XPush マシンではフィルタ条件を XPath[4]で記述し，この条件を XPath フィルタという．XPath フィルタはナビゲーション部とプリディケート部からなる．次に XPush マシンで利用する XPath フィルタの例を示す．

例 1 [Running example]
P1 : //a[@b<20]
P2 : //a[@b>=10 and @b<20]
P3 : //a[@b>=10]

“[ ]”で囲まれた部分がプリディケート部であり，そこには条件を記述する．プリディケート部より前の部分をナビゲーション部という．“//a”はナビゲーション部であり，上記 3 式ともナビゲーション部は同じである．“//”はワイルドカードであり，“//a”は a タグが XML データのどの位置に現れてもよいことを表す．“@”は属性を意味し，“@b”は属性 b を表す．数値の 10 と 20 は引用符で囲まれていないため，XML データの属性 b の値は数値型のデータとして評価される．

XPath の評価には DOM 型の XML プロセッサがよく利用される．ここで DOM 型とは XML データ全体を一時的にメモリに読込み処理する方式を指す．XML プロセッサによる XPath 評価は XML データサイズと XPath フィルタ数の双方に比例する．一方 XPush マシンの処理速度は文書数に比例するだけであり，フィルタ数にはほとんど影響を受けない．

例えば例 1 で XPush マシンを利用すると，

XPush マシンは 3 つのフィルタ条件を予めボトムアップに結合し 1 つの状態遷移表に変換するため，文書ごとに 1 回の逐次的なアクセスだけで 3 つ全てのフィルタ条件評価をカバーできる．

逐次的に処理するために XPush マシンは SAX 型の XML パーサを利用する．SAX 型のパーサは XML データをストリームとして扱い，XML データのタグや値ごとにイベントをアプリケーションに通知する．アプリケーションは各イベント時に渡される断片的な XML データによって処理を行う．一般に SAX 型は XML データサイズに制限されない利点を持つ．実際 XPush マシンのスタック制御部は XML データの深さ分のスタックしか利用しない．

しかし Gupta らの提案では XPush マシンの更新手段が未解決であるため，XPush マシン全体の再計算が更新処理のために必要となる．また XPush マシンは XML データの評価時に非決定性有限オートマトンから決定性有限オートマトンへ遅延変換を行う．そのために評価データの構造が変化しやすい環境ではオートマトンの状態数が増え続け，システムのメモリを使い果たす問題がある．

ここでメモリコストをコントロールする必要があるシステム，例えば利用者が多いデータベースシステム上の問い合わせエンジンとしての応用を考える．メモリコストを下げるため XPath フィルタ集合から使用頻度の低いものに対して削除処理が計画されたとする．XPush マシンは 1 つのフィルタ条件を削除することであっても XPush マシン全体の再計算が必要となる．初期状態の非決定性有限オートマトンに戻った XPush マシンは決定性有限オートマトンへの遅延変換を要求するため，システムのスループットを低下させる．

本稿ではこの未解決問題のために XPath フィルタグループごとに構築した部分状態遷移表（サブ XPush マシンと呼ぶ）の集合から全体のオートマトンを構築する方法を提案する．なお XPath フィルタグループには問合せ[1][6][10]やセッション単位を想定している．以降では，このサブ XPush マシン集合から構築した全体のオートマトンを統合型 XPush マシンと呼ぶ．

統合型 XPush マシンは XPush マシンにフィルタの結合と交換の 2 つの機能を追加する．その仕組みは次の通りである．

統合型 XPush マシンはオートマトン制御部とのインターフェースが従来どおりであり，従来方式と同じスループット性能を持つ．そしてコンパイル後のオートマトンはグループごとに分割管理された状態遷移表（サブ XPush マシン）となっている．サブ XPush マシンのコンパイラには Gupta らの XPush マシンのコンパイラをほぼそのまま利用する．このコンパイラは状態ごとにキーを追加する修正が施されるだけである．状態どうしのさらなる結合には状態ごとのキーを用いる．フィルタグループ単位で分解も可能であり，分解した XPath フィルタも結合前と同じ状態に再結合が可能である．

また統合型 XPush マシンを利用する XML システムの XPath フィルタ更新において，再結合の処理速度が低コストであれば，そのシステムのスループット維持を期待することができる．そのため追加および分解の更新アルゴリズムにはサブ XPush マシンが持つ分割された情報からの再集計を避けたインクリメンタルな結合および遅延変換を行う．この更新処理は状態間の世代関係を利用している．

本稿の構成は以下の通りである．この節の残りでは関連する研究について述べる．続く **2 節**で，提案方式の概要を述べ，**3 節**では再結合時間に関する実験概要とその結果を示し，最後に **4 節**で再結合に用いるキーについて議論する．

1.1. 関連研究

XML ストリームを利用した XML フィルタの研究の多くはナビゲーション部を利用する[2][5]．それらは XML 構造が木構造であることに基づき XPath フィルタ間の同じナビゲーション部評価の削減計画を行い効率化が図られる．しかしこの手法ではプリディケート部については未評価の状態で XML データの断片が抽出される．

一方，本方式は Gupta らの XPush マシンと同様に，ナビゲーション部だけではなくプリディケート部評価の効率化を行っている．

XML ストリームを利用しない研究として，Tian ら[9]は XML パブリッシング／サブスクライビングシステムのために関係データベースを利用して，100 万を超えるフィルタ数への対応を実現している．この手法はデータベースに XML データを一時保存するため，XML データの評価時の耐久性を持たすことも実現している．さらに，XPath フィルタのコンパイルをデータベース上で全て行うため，コンパイル時のメモリ制限をなくすことを実現している．この手法は XPath フィルタ更新が可能である．

一方，本方式を利用するシステムにはデータベースシステムは必要ない．しかし本方式ではある時点の必要なフィルタが判断できることを要求する．この制約と XPush マシンへの更新機

能の追加によって次のことがいえる．フィルタの分離と結合ができるため，同時に評価させるフィルタを適切に判断し制御すればデータ評価時のフィルタ数に対するメモリ制限をなくすことができる．

本研究の目的を以下にまとめると以下のようになる．

- XPush マシンに XPath フィルタの更新を可能にすること．
- 更新処理がスループットに影響を与えないため，インクリメンタルコンパイルを実現すること．

2. 提案方式

この節では本提案方式の統合型 XPush マシンについて述べる．本方式は XPush マシンを拡張している．そこでまず **2.1 節**で XPush マシンを説明してから，**2.2 節**で本提案方式の概要，**2.3.** で結合状態キー，**2.4 節**でインクリメンタル更新アルゴリズムについて述べる．なお，XPush マシンの説明にはボトムアップ結合を中心とした説明をするため，トップダウン結合に関することなど多くのことが省略されている．

2.1. XPush マシン（従来方式）の概要

XPush マシンを用いた XPath フィルタエンジンのアーキテクチャを**図 1** に示す．このエンジンはオートマトンモジュールとコントローラモジュールからなる．

オートマトンモジュールでは XPath フィルタから問合せ木（構文木）への変換を XPath パーサによって処理した後に，それをさらにコンパイラによって状態遷移表へと変換する．

XPush マシンのコンパイラは AFA（Alternating Finite Automaton）[3]に基づいている．Gupta らは XPath 用に AFA を拡張し，XPush マシンに利用する．以降でいう AFA は，この XPush マシンで利用する拡張された AFA を意味する．コンパイラは XPath フィルタごとの AFA を利用してフィルタ間の XPush 状態を探し，その状態を用いて状態遷移表を作成する．XPush 状態とは，全 AFA のそれぞれに同じ遷移をさせたときの，全 AFA の各遷移後の状態の和集合をいう．AFA は次の特徴を持つ．

- 必ず終端が存在し，葉はアトミックプリディケートである．アトミックプリディケートは文字列と数値型を区別し，＝,＜,＞,≧,＝,≠の比較操作を持つ．
- AND,OR,NOT の ε -動作（ε -transision）を持つ．
- ルートは XPath フィルタの受理状態を意味する．
- 自己再帰を持つ．
- 遷移には value,pop,add の 3 種類がある．
- pop による遷移後の状態数は 0 個以上．

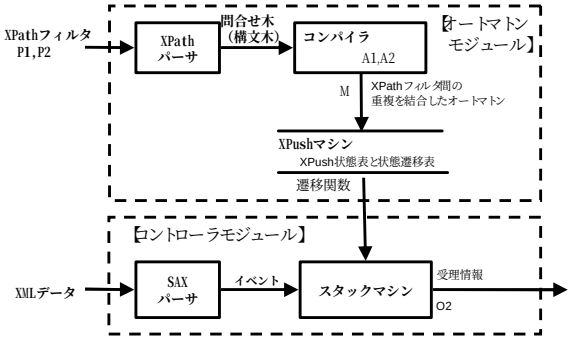
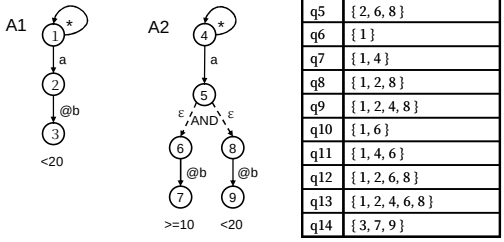


図 1 XPush マシンを用いた XPath フィルタエンジンのアーキテクチャ



q0	Φ
q1	{3, 9}
q2	{7}
q3	{2, 8}
q4	{6}
q5	{2, 6, 8}
q6	{1}
q7	{1, 4}
q8	{1, 2, 8}
q9	{1, 2, 4, 8}
q10	{1, 6}
q11	{1, 4, 6}
q12	{1, 2, 6, 8}
q13	{1, 2, 4, 6, 8}
q14	{3, 7, 9}

a. P1のAFA　b. P2のAFA　c. XPush状態のコレクション

図 2 例 1 の P1,P2 の AFA と XPush 状態のコレクション

AFA の例を**図 2** に示す．図 2a と図 2b は以前に示した例 1 の P1,P2 の AFA であり，図 2c はそれら AFA をボトムアップに結合した XPush 状態のコレクションである．以降ではこの XPush 状態のコレクションを XPush 状態表という．例えば図 2c の XPush 状態 q3 は，図 2a の A1 と図 2b の A2 それぞれに(<20, pop(@b))の遷移をさせたときの状態集合となっている．

コントローラモジュールはコンパイラよって作成した状態遷移表に従って XML データの評価を行う．状態遷移表の参照は遷移関数を利用する．このとき必要があれば遅延変換処理が発生する．図 2c の XPush 状態表には A1 と A2 から遷移可能なすべての状態集合が含まれている．しかし XPush 状態を探す計算量は状態数を n とすると $O(2^n)$のため難しい．そこで Gupta らはこの処理をコンパイル時には行わず，遷移関数が利用されるデータ評価時に必要とされた分だけ計算する遅延型方式を提案している．この処理によって XPush マシンは部分的に非決定性有限オートマトンから決定性有限オートマトン

になる．この変換結果は XPush 状態表と状態遷移表に保存される．なお遅延変換はスループット低下の原因となる．

XPush マシンは XPush 状態表以外にいくつかの状態遷移表の Tpop, Tadd, Tvalue などからなる．状態遷移表ではこの XPush 状態だけを状態値として用いる．AFA の状態を直接扱うことはない．**図 3** に例 1 の P1,P2 の状態遷移図を示す．

XPush マシンの制御部はデータを XML ストリームとして評価する．スタック制御部はタグや値ごとに発生するイベントから**図 4** のイベントプロシージャを実行する．

図 4 のイベントプロシージャが示すように各イベントでは遷移関数 value(str), pop(qb, a), add(q, qaux)のどれか 1 つを 1 回だけ利用する．XPush マシンの状態遷移表に対して決められた回数の表引が行われる．そのため評価時の計算量は文書サイズに比例するが，タグごとの処理時間は O(1)であり，フィルタ数には一定である．また XML ストリームを利用するため XML データサイズに対する評価時のメモリ制限はない。

2.2. 統合型 XPush マシンの概要

ここでは提案方式の統合型 XPush マシンについての概要について述べる．

統合型 XPush マシンを用いた XPath フィルタエンジンのアーキテクチャを**図 5** に示す．エンジンは従来方式と同じくコントローラモジュールとオートマトンモジュールからなる．

コンパイラとスタック制御部の間には統合処理部が存在する．このことは Gupta らの手法と大きく異なる点である．図 5 で M1～M5 はサブ XPush マシンを表す．これらは XPath パーサとコンパイラを利用した XPath フィルタのコンパイル結果である．サブ XPush マシンは XPath フィルタグループ単位に作成される．現在フィルタグループには問合せやセッション単位を想定している．コンパイラは Gupta らの XPush マシンのコンパイラとほぼ同じである．統合処理部はコンパイル結果の統合型 XPush マシンへの結合と，さらに外部記憶を利用したサブ XPush マシン単位の分離と再結合を処理する．

例えば図 5 では 2 つの XPath フィルタ P1,P2 を M3 にコンパイルした後，統合処理部によって統合 XPush マシンがインクリメンタルに更新される．そして M4 の id を利用して分離命令を統合処理部に伝えると，M4 を統合型 XPush マシンから分離後，外部記憶に保存する．さらに M5 の結合命令によってサブ XPush マシン M5 を復元し，そして統合 XPush マシンとの再結合が行われるという流れを示している．

	a	@b	*	@*
q0			q0	q0
q1		q3	q0	q0
q2		q4	q0	q0
q3	q6		q0	q0
q4			q0	q0
q5	q7		q0	q0
q6			q6	q0
q7			q7	q0
q8			q6	q0
q9			q7	q0
q10			q6	q0
q11			q7	q0
q12	q7		q6	q0
q13			q7	q0
q14		q5	q0	q0

a. Tpop状態遷移表

	q0	q3	q4	q5	q6	q7
q0	q0	q3	q4	q5	q6	q7
q1	q1					
q2	q2					
q3	q3	q3	q5	q5	q8	q9
q4	q4	q5	q4	q5	q10	q11
q5	q5	q5	q5	q5	q12	q13
q6	q6	q8	q10	q12	q6	q7
q7	q7	q9	q11	q13	q7	q7
q8	q8	q8	q12	q12	q8	q9
q9	q9	q9	q13	q13	q9	q9
q10	q10	q12	q10	q12	q10	q11
q11	q11	q13	q11	q13	q11	q11
q12	q12	q12	q12	q12	q12	q13
q13	q13	q13	q13	q13	q13	q13
q14	q14					

b. Tadd状態遷移表

(-∞,10)	[10,20)	[20, ∞)
	q14	q2

c. Tvalue状態遷移表

図 3 例 1 の P1,P2 の状態遷移表

```
// モジュール変数
s  // XPush状態を保持するスタック
q0 // XPush状態の初期値
q  // 現在のXPush状態

// イベントプロシージャ
void startElement(String tag-name)
    q ← q0
void text(String str)
    q ← value(str)
void endElement(String tag-name)
    qaux ← pop(qb, a)
    q ← s.pop()
    q ← add(q, qaux)
```

図 4 スタックマシンのコールバック関数

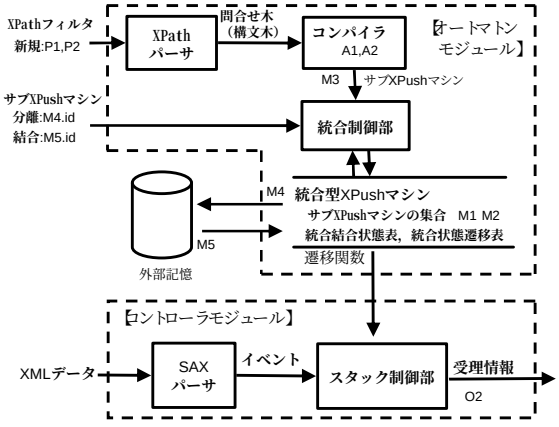


図 5 統合型 XPush マシンを用いた XPath フィルタエンジンのアーキテクチャ

統合型 XPush マシンに結合したサブ XPush マシンの XPush 状態の管理テーブルは統合結合状態のコレクションである．以降，統合結合状態表という．統合結合状態は統合中のサブ XPush マシンごとに 1 つの XPush 状態への参照を含む．

遷移関数は統合結合状態を参照する統合状態遷移表からの表引きに変更している．これによって，オートマトンモジュールとコントローラモジュールが従来方式と互換性を保ち，そのため XPath フィルタエンジンのスループットは同じである．

2.3. 結合状態キー

ここでは結合状態キーの作成方法について述べる．

図 6 に結合状態表と統合結合状態表を示す．図 6a,図 6b,図 6c はそれぞれ例 1 の P1,P2,P3 の AFA をボトムアップ結合した XPush マシン M1,M2,M3 の結合状態表 M1.Qs,M2.Qs,M3Qs である．結合状態表は，従来方式の XPush 状態表と異なり状態ごとにキーがある．以降ではこのキーを結合状態キー，そして結合キーを持つ XPush 状態を結合状態という．本方式では統合制御部が結合状態表の結合時に，この結合状態キーを利用する．なお本方式ではフィルタグループごとに AFA 中の状態番号を 1 から割当てる．そのため M1.Qs,M2.Qs,M3.Qs の番号に重複した値が存在している．例えば P1,P2 が同じフィルタグループの場合は，以前に示した図 2 での Gupta らの方式の場合と同じ AFA の状態番号が割当てられ，M1.Qs,M2.Qs 間の重複した AFA の状態番号はなくなる．

結合状態キーはサブ XPush マシンの XPush 状態を見つけ出す時に一緒に計算する．以前に XPush 状態はそれぞれの AFA に対して同じ遷移をさせたときの各 AFA の状態の和集合であることを述べた．結合状態キーはその遷移を記録して作成する．例外として，“{ }”で囲まれているキーがあるが，そのキーを持つ XPush 状態は Tvalue 状態遷移表の作成時に決定するため，そのときに一緒に計算する．結合状態キーの特徴として，サブ XPush マシン間をまたいだ結合状態キーは存在しない．

図 6d は M1,M2 をさらに結合した統合型 XPush マシンの統合結合状態表である．図 6d には 15 個の統合結合状態が存在し，それら各々がサブ XPush マシンごとに 1 つの XPush 状態への参照が存在している．同様に統合結合状態表にもキーが存在し，これを統合結合状態キーという．また統合 XPush マシンとサブ XPush マシン間をまたいだ統合結合状態キーは存在しない．このキーの作成については次項で詳しく述べる．

図 6e の統合結合状態表は，図 6d の統合型 XPush マシンの結合状態表に図 6c のサブ XPush マシンの結合状態表をインクリメンタルに結合した結果である．

M1.Qs		結合状態キー
q0	Φ	(null)
q1	{ 3 }	value <20
q2	{ 2 }	pop q1, "@b"
q3	{ 1 }	pop q2, "a"
q4	{ 1, 2 }	add q2, q3

a. サブXPushマシン M1の結合状態表

M2.Qs		結合状態キー
q0	Φ	(null)
q1	{ 4 }	value >=10
q2	{ 6 }	value <20
q3	{ 3 }	pop q1, "@b"
q4	{ 5 }	pop q2, "@b"
q5	{ 3, 5 }	add q3, q4
q6	{ 1 }	pop q5, "a"
q7	{ 1, 3 }	add q3, q6
q8	{ 1, 5 }	add q4, q6
q9	{ 1, 3, 5 }	add q5, q6
q10	{ 4, 6 }	{ >=10, <20 }

b. サブXPushマシン M2の結合状態表

M3.Qs		結合状態キー
q0	Φ	(null)
q1	{ 3 }	value >=10
q2	{ 2 }	pop q1, "@b"
q3	{ 1 }	pop q2, "a"
q4	{ 1, 2 }	add q2, q3

c. サブXPushマシン M3の結合状態表

Xs	M1	M2	統合結合状態キー
x0	q0	q0	(null)
x1	q1	q2	value <20
x2	q2	q4	pop x1, @b
x3	q3	q0	pop x2, "a"
x4	q4	q4	add x2, x3
x5	q0	q1	value >=10
x6	d0	d3	pop x5, "@b"
x7	q2	q5	add x2, x6
x8	q3	q3	add x3, x6
x9	q4	q5	add x4, x6
x10	q3	q6	pop x7, "a"
x11	q4	q8	add x2, x10
x12	q3	q7	add x6, x10
x13	q4	q9	add x7, x10
x14	q1	q10	{ <20, >=10 }

d. 統合結合状態表 更新前 Xs(=M1.Qs+M2.Qs)

Xs	M1	M2	M3	統合結合状態キー
x0	q0	q0	q0	(null)
x1	q1	q2	q0	value <20
x2	q2	q4	q0	pop x1, @b
x3	q3	q0	q0	pop x2, "a"
x4	q4	q4	q0	add x2, x3
x5	q0	q1	q1	value >=10
x6	q0	q3	q2	pop x5, "@b"
x7	q2	q5	q2	add x2, x6
x8	q3	q3	q2	add x3, x6
x9	q4	q5	q2	add x4, x6
x10	q3	q6	q3	pop x7, "a"
x11	q4	q8	q3	add x2, x10
x12	q3	q7	q4	add x6, x10
x13	q4	q9	q4	add x7, x10
x14	q0	q0	q3	pop x6, "a"
x15	q3	q0	q3	pop x8, "a"
x16	q2	q4	q3	add x2, x14
x17	q4	q4	q3	add x4, x14
x18	q0	q3	q4	add x6, x14
x19	q2	q5	q4	add x7, x14
x20	q3	q3	q4	add x8, x14
x21	q4	q5	q4	add x9, x14
x22	q1	q10	d1	{ <20, >=10 }

e. 統合結合状態表 更新後 Xs += M3.Qs

図 6 結合状態表と統合結合状態表

2.4. インクリメンタル更新アルゴリズム

ここでは統合型 XPush マシンの更新アルゴリズムについて述べる．

統合制御部は関係データベースにおける表結合のようにキーを利用して，統合結合状態表へサブ XPush マシンの結合状態表をインクリメンタルに結合する．

しかし，そのままでは統合結合状態キーと結合状態キーは結合できない．なぜならそれらのキーはそれが記録されている表に存在する状態を参照し，閉じた状態となっているためである．またキーの中で add は引数を 2 つ持っているが引数の順番には意味がなく対等である．たとえば“add q2, q3”と“add q3, q2”はキーとしては等しいものとして扱う．さらにアトミックプリディケートの集合は例外処理が必要であり，value 統合状態遷移表の計算時に処理をする必要がある．

次に統合制御部の結合アルゴリズムの概要を示す．

【結合アルゴリズムの概要】

入力：結合するサブ XPush マシン m

Step1：前処理

Step2：統合結合状態 x の遷移 q が m 上に存在するとき，q を x に追加する．

Step3：Step2 で利用されなかった m 上の遷移を統合結合状態として新規に登録する．

図 7 に統合制御部における結合アルゴリズムを示す．

Step2 では，統合結合状態 x を m の状態遷移表で遷移させ，その結果を x に追加する．このときサブ XPush マシンの状態遷移表を利用するために，予め統合結合状態の参照を結合状態の参照に変換する（15 行目）．

Step3 では，結合状態 q を統合遷移表で遷移させ，Step2 との重複を取り除く（24,25,26 行目）．その結果を新規の統合結合状態として統合結合状態表へ追加する（27 行目）．このとき統合状態遷移表を利用するために，予め結合状態の参照を統合結合状態の参照に変換する（22 行目）．

統合結合状態は Step3 で新規作成する（27 行目）．このとき統合結合状態キーも一緒に作成する．その方法は結合状態キーと同様に，遷移動作を記録する．

さらに統合結合状態の新規作成時に状態遷移表にも更新を行う．統合状態遷移表の更新は全てのサブ XPush マシンの状態遷移表からの集計によって計算することができる．しかしフィルタグループ数が増加すると計算量が増えるため好ましくない．そのため本方式ではこの計算を統合結合状態間の世代関係を利用してインクリメンタルに計算する．

図 8a に更新前の統合状態遷移表 Tadd と図 8b に結合中のサブ XPush マシンの状態遷移表 M3.Tadd を示す．図 6e にある更新後の統合状態遷移表より，新規に作成された統合結合状態の M1,M2 の参照値と同じ統合結合状態が存在する．これを統合結合状態の世代関係と呼んでいる．先に作成された統合結合状態が前世代となり，次世代の子孫は複数存在しうる．図 8c に統合結合状態の世代関係を示す．この関係を使い，図 8a の Tadd から図 8d の Tadd へ更新する．更新には図 8e の IncrementalTAdd 関数を利用する．Tpop 統合状態遷移表も同様にインクリメンタルに更新できる．

ただし更新コストを削減させるフィルタグループ間の世代関係を考慮したインクリメンタル更新は，状態遷移表を分割管理していない従来方式の構造には適用できない．

```
 1: // 統合XPushマシンのモジュール変数
 2: Ms  // サブXPushマシンの管理テーブル
 3: Xs  // 統合型XPushマシンの統合状態遷移表
 4:
 5: procedure joint(m) {
 6:   m // サブXPushマシン
 7: {
 8:   // Step1
 9:   Ms.add(m);
10:   unbindTvalue();
11:   x0.add(m.q0);
12:
13:   // Step2
14:   foreach x in Xs order by K {
15:     K k = x.getKey().X2Q(mapX2Q);
16:     Q q = m.δ(key);
17:     x.add(q);
18:   }
19:
20:   // Step3
21:   foreach q in m.Qs order by K {
22:     Ks ks = q.getKey().Q2Xs(mapQ2Xs);
23:     foreach k in ks order by K {
24:       if (Xs.containsK(k) continue;
25:       X x = δ(k);
26:       if (Xs.containsXQ(x,q))
27:         Xs.add(x,q);
27:     }
28:   }
29: }
```

図 7 統合制御部の結合アルゴリズム

Tadd	x0	x2	x3	x6	x10
x0	x0	x2	x3	x6	x10
x1	x1				
x2	x2	x2	x4	x7	x11
x3	x3	x4	x3	x8	x10
x4	x4	x4	x4	x9	x11
x5	x5				
x6	x6	x7	x8	x6	x12
x7	x7	x7	x9	x7	x13
x8	x8	x9	x8	x8	x12
x9	x9	x9	x9	x9	x13
x10	x10	x11	x10	x12	x10
x11	x11	x11	x11	x13	x11
x12	x12	x13	x12	x12	x12
x13	x13	x13	x13	x13	x13
x14	x14				

a. 統合状態遷移表Tadd 更新前

M3 .Tadd	q0	q2	q3	q4
q0	q0	q2	q3	q6
q1	q1			
q2	q2	q2	q4	q4
q3	q3	q4	q3	q4
q4	q4	q4	q4	q4

b. サブXPushマシンM3の状態遷移表M3.Tadd

X	pre	M1	M2	M3
x14	x0	q0	q0	q3
x15	x3	q3	q0	q3
x16	x2	q2	q4	q3
x17	x4	q4	q4	q3
x18	x6	q0	q3	q4
x19	x7	q2	q5	q4
x20	x8	q3	q3	q4
x21	x9	q4	q5	q4

c. 統合結合状態間の世代関係

Tadd	x0	x2	x3	x6	x10	x14	x15
x0	x0	x2	x3	x6	x10	x14	x15
x1	x1						
x2	x2	x2	x4	x7	x11	x16	x17
x3	x3	x4	x3	x8	x10	x15	x15
x4	x4	x4	x4	x9	x11	x17	x17
x5	x5						
x6	x6	x7	x8	x6	x12	x18	x20
x7	x7	x7	x9	x7	x13	x19	x21
x8	x8	x9	x8	x8	x12	x20	x20
x9	x9	x9	x9	x9	x13	x21	x21
x10	x10	x11	x10	x12	x10	x10	x10
x11	x11	x11	x11	x13	x11	x11	x11
x12	x12	x13	x12	x12	x12	x12	x12
x13	x13	x13	x13	x13	x13	x13	x13
x14	x14	x16	x15	x18	x10	x14	x15
x15	x15	x17	x15	x20	x10	x15	x15
x16	x16	x16	x17	x19	x11	x16	x17
x17	x17	x17	x17	x21	x11	x17	x17
x18	x18	x19	x20	x18	x12	x18	x20
x19	x19	x19	x21	x19	x13	x19	x21
x20	x20	x21	x20	x20	x12	x20	x20
x21	x21	x21	x21	x21	x13	x21	x21
x22	x22						

d. 統合状態遷移表Tadd 更新後
Tadd += M3.Tadd

```
function IncrementalTadd(x1, x2) : X
{
    X x = Tadd( pre(x1), pre(x2) );
    Q q = M.Tadd( q1, q2 );
    X rval = Xs.find(x, q) ;
    return rval;
}
```

e. 統合結合状態間の世代関係を利用した統合状態遷移表Taddの更新関数

図 8 統合結合状態遷移表のインクリメンタル更新

3. 実験

ここでは統合型 XPush マシンの再結合時間に関する実験概要と実験結果を示す．

3.1. 実験概要

実験は提案方式のフィルタエンジンを Java で実装し，これを Red Had Linux 8.0 上で使用した．使用したコンピュータの CPU は PentiumⅣ 3.3GHz，メモリ 2G である．Java の最大ヒープサイズは 30MB に設定し，XML パーサには xerces2.6.2（http://xml.apache.org）を利用した．

また比較のために従来方式の XPush マシンも Java で実装し，同じ環境で測定した．

実験データには Gupta らの実験でも用いられた Protein（http://pir.georgetown.edu）を利用した．Protein は 700MB あるデータセットであり，これを 9.12MB のサイズに分割したものを用いた．XPath フィルタは XML データからランダムに作成するプログラムを作成し，深さ 2〜4 の XPath を 100 件生成させ用いた．このフィルタ 100 件全てを従来方式のフィルタエンジンで再コンパイルさせデータ評価時間を測定した．結果は再コンパイル後の評価時間に約 40 秒必要であり，このときスループットが 1.5MB/s から 0.2MB/s に一時的に下がった．

同様に本方式のフィルタエンジンに 100 件のフィルタ条件を 1 から 20 のグループサイズでコンパイルし，1 つのフィルタをランダム交換したときのデータ評価時間を図 9 に示す．

3.2. 実験結果

再結合時間は再計算時間に比べて，フィルタグループサイズが 7 以下で 97%以上改善している．しかしグループサイズにはスループットを維持できない限界サイズがある．さらに別の実験ではフィルタ総数を増やすと限界サイズは小さくなる傾向があった．グループサイズが増加する程，再結合にコストがかかっている．この理由は以下のように考えられる．

統合型 XPush マシンは，サブ XPush マシンごとの部分的に確定した決定性有限オートマトンをシリアライズする．サブ XPush マシン間に確定した決定性有限オートマトンはサブ XPush マシン外に記憶され，シリアライズ対象外となる．このシリアライズ対象外となった情報は，統合型 XPush マシンで管理されているときだけ有効である．そのために再結合が必要なフィルタ数（フィルタグループサイズ）が増えると，決定性有限オートマトンへの再変換数が増加する．

現在の実装ではこの管理情報はサイズが大きく，まだその内部構造が把握できない問題があり，フィルタ分離時にはその世代への参照の切離し処理を行い，参照がなくなった管理情報は自動的に削除対象となる手段を取っている．

4. ディスカッション

ここでは Gupta の方式では扱えない本方式の結合状態キーについて議論する．

まず，Gupta らの方式の XPush 状態表と本方式の結合状態表に違いについて考える．XPush 状態とは全 AFA のそれぞれに同じ遷移をさせたときの，全 AFA の各遷移後の状態の和集合であることを以前に述べた．Gupta らの XPush マシンでは XPush 状態のそれに含まれる AFA 状態を昇順に並替え，XPush 状態表として管理している．Gupta らの方式では結合状態は遷移後の結果である．一方，本方式では AFA の状態リストをそのまま利用し，さらにそのときの遷移動作も結合状態キーとして利用する．従来方式では結合状態作成時の遷移情報は残らないため，キーを利用した結合アルゴリズムは適用できない．

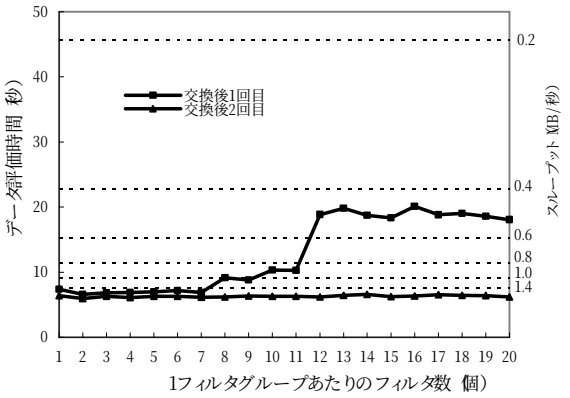


図 9 統合型 XPush マシンのフィルタ交換時間

次に，XPush マシンの AFA と本方式のキーを比較する．XPush マシンの AFA は XPath フィルタ用に拡張した有限オートマトンである．一方，本方式のキーは XPush マシンの AFA を展開した AFA と考えることができる．**図 10** にその比較を示す．

図 10a,図 10b は例 1 の P1,P2 の AFA である．一方，図 10c,図 10d は P1,P2 の AFA の展開後を表している．それぞれ図 6a,図 6c の結合状態表の結合状態キーの参照を結合した結果である．ただし図 10c は統合結合状態キーとなっている．

本方式の結合アルゴリズムは，メモリコストを考え，図 10c,図 10d のようにキーを展開することないが，キーのマッチングと同じことを実現している．図 10e に，図 10c の統合結合状態キーと図 10d の結合状態キーのマッチング結果を示す．このマッチング結果に基づいて，統合処理部は図 10d のキーから図 10f，また統合結合状態の新規作成時は図 10g のように統合結合状態キーへと更新される．x7 はサブ XPush マシン間を跨った x3 と x6 の参照を持つ．x9 も同様であるが，さらに x9 は x4 と x7 間の重複 x2 を除去した簡略化後の統合結合状態となっている．これらの追加されたキーは本方式では現在シリアライズ対象外となっている．

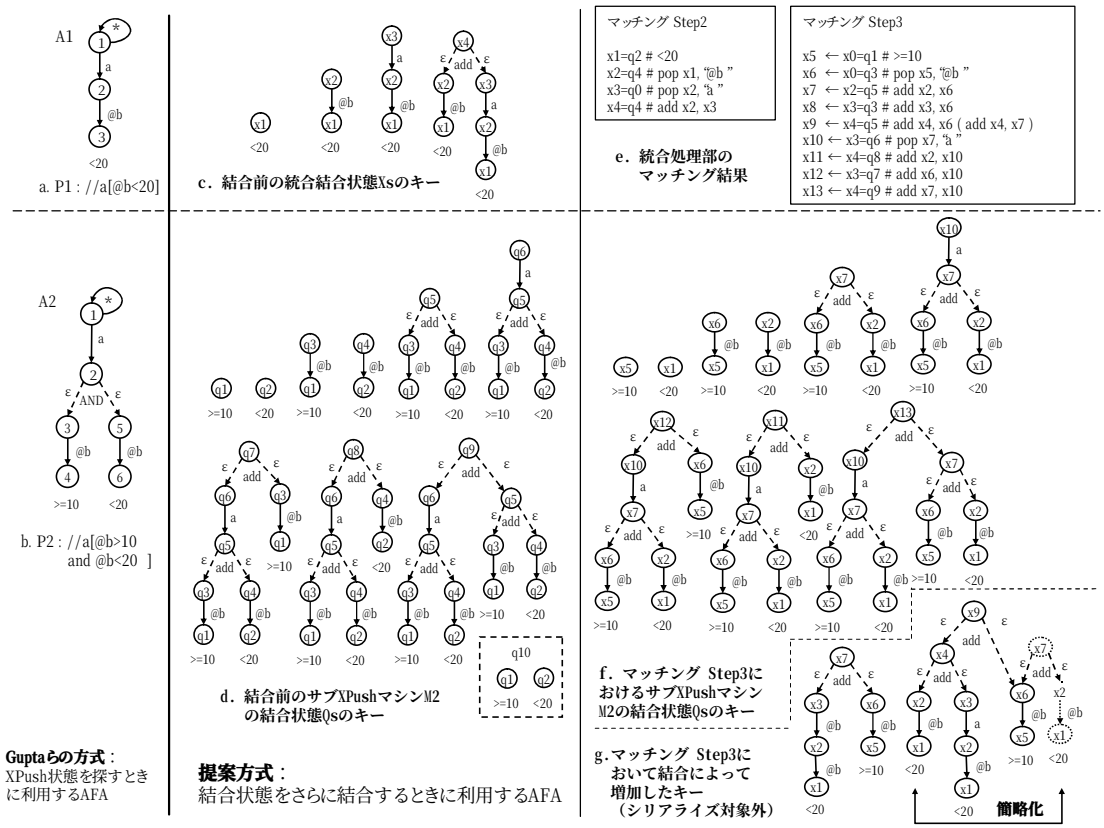


図 10 XPush マシンの AFA と本方式のキーとの比較

5. おわりに

本研究では XPush マシンの更新を可能にし，学習済みのフィルタとの再結合を可能にする統合型 XPush マシンを提案した．

そのために本稿では世代関係を利用して XPush マシンをインクリメンタルに更新を行う方法を述べ，フィルタグループサイズが小さいときに統合型 XPush マシンの更新時間が XPush マシンの再計算より計算コストが低く，従来方式よりスループットを維持することを実験結果で示した．

また統合型 XPush マシンはサブ XPush マシンの XPush 状態ごとに XPush マシンの AFA を展開して得る結合状態キーを持ち，この結合状態キーを結合時にキーとして用いることを述べた．

さらに従来方式では XPush 状態発見時の遷移情報は残らないこと，そして状態遷移表を分割管理していないため，統合型ではない従来型のアーキテクチャには本方式を適用できないことを述べた．

今後の課題として，フィルタグループサイズが大きくなったときの更新コストを削減するために統合結合状態表のキャッシュの実現を考えている．

参　考　文　献

[1] 澤井里枝,塚本昌彦,寺田努,西尾章治朗：“フィルタリングのためのユーザ要求記述言語 FilteringSQL について”, DBS 2003.

[2] 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”, DBWS 2003.

[3] Chandra, A. Kozen, D. Stockmeyer, L. : “Alternation”, In journal of the ACM, pp. 115–133, January 1981.

[4] Clark, J. : “XML path language(XPath), 1999. http://www.w3.org/Tr/xpath.

[5] Diao, Y. Fischer, P. Franklin, M. To, R. : “Yfilter: Efficient and scalable filtering of XML documents”. In Proc. ICDE, pp. 341–344, February 2002.

[6] Feng, Peng. S. C. Sudarshan. : “XPath Queries on Streaming Data”. In Proc. SIGMOD, pp431-442, 2003.

[7] Green, T. J. Miklau, G. Onizuka, M. Suciu. D. : “Processing XML streams with deterministic automata”. In Proc. ICDT, pp. 173–189, 2003.

[8] Gupta, A. Suciu, D. : “Stream Processing of XPath Queries with Predicates”, In Proc. SIGMOD, pp. 419–430 , 2003.

[9] Tian, Feng. “Implementing A Scalable XML Publish/Subscribe System Using Relational Database Systems”, In Proc. SIGMOD, pp. 479–490, 2004.

[10] XQuery 1.0, http://www.w3.org/TR/xquer/