DEWS2005 6B-o2

高速処理のための XPath 問い合わせ変換手法

阿部　将佳† 　原　忠司†† 　石原　靖哲† 　織田　美樹男††

† 大阪大学大学院情報科学研究科
〒 565–0871 吹田市山田丘 1–5
†† 株式会社メディアフュージョン
〒 530–0004 大阪市北区堂島浜 2–1–29 古河大阪ビル
E-mail: †{msys-abe,ishihara}@ist.osaka-u.ac.jp, ††{hara,oda}@mediafusion.co.jp

あらまし　筆者らは，XPath 問い合わせの処理を高速化する問題を，(1) 任意の XPath 問い合わせの，述語が単純な複数の XPath 問い合わせに集合和積演算を適用した形の式への変換，(2) 集合和積演算の処理高速化，(3) 述語が単純な XPath 問い合わせの処理高速化，の 3 つの部分問題に分割し，それぞれ個別に解決するというアプローチについて検討を行っている．本稿では，上記 (1) の XPath 問い合わせの変換手法について述べる．具体的には，まず変換の対象となる XPath 問い合わせの部分クラスを定義する．そして，そのクラスに属する問い合わせを 1 回の走査によりトップダウンに変換するアルゴリズムを提案し，その正しさを証明する．また，実験により，本変換手法が処理の高速化に有効であることを確かめ，上述の，処理の高速化問題を 3 つの部分問題に分割するアプローチについても，その有効性を確かめた．

キーワード　XML **データベース**, XPath, **問い合わせ最適化**

XPath Query Transformation for Time-Efficient Evaluation

Masayoshi ABE†, Tadashi HARA††, Yasunori ISHIHARA†, and Mikio ODA††

† Graduate School of Information Science and Technology, Osaka University
1–5 Yamadaoka, Suita, Osaka, 565–0871 Japan
†† Media Fusion Co.,Ltd.
Furukawa Osaka Bldg. 2–1–29 Dojimahama, Kita-ku, Osaka, 530–0004 Japan
E-mail: †{msys-abe,ishihara}@ist.osaka-u.ac.jp, ††{hara,oda}@mediafusion.co.jp

Abstract　The aim of this research is to develop a time-efficient evaluation method of XPath queries. In our approach, we divide the problem into the following three subproblems and then solve them independently: (1) to develop a transformation algorithm from XPath queries into queries which are in the form of set unions and/or intersections of XPath subqueries with simple predicates, (2) to develop a time-efficient evaluation method of set union and intersection operations, and (3) to develop a time-efficient evaluation method of XPath subqueries with simple predicates. In this paper, we present a transformation alogorithm stated in the above (1). Concretely, we first define a subclass of XPath queries, which include the input query class of the transformation algorithm. Then we present a one-pass top-down transformation algorithm, and prove the correctness of the proposed algorithm. Lastly, we empirically show that the transformation algorithm and also the whole of our approach contribute to time-efficient evaluation of XPath queries.

Key words　XML Database, XPath, query optimization

1. はじめに

XPath [12] は XML ドキュメント内の要素を指定する方法として広く使われており，XSLT [5] や XQuery [3] などのいくつかの実用的な XML 操作言語における重要な構成要素である．これら操作言語の処理効率向上のためには XPath 問い合わせを高速に処理できることが求められる．しかし，例えば XPath 問い合わせの処理系を XPath の意味定義通りに実装した場合，非常に大きな XML ドキュメントに対する XPath 問い合わせの処理に多くの時間を要することが知られている [6]．そのた

め，XPath 問い合わせを高速に処理するための手法の研究が，近年，盛んに行なわれはじめた．

筆者らは，XPath 問い合わせの処理を高速化する問題を，(1) 任意の XPath 問い合わせの，述語が単純な複数の XPath 問い合わせ (以降，原子式と呼ぶ) に集合和積演算を適用した形の式への変換，(2) 集合和積演算の処理の高速化，(3) 述語が単純な XPath 問い合わせの処理高速化，の 3 つの部分問題に分割し，それぞれ個別に解決するというアプローチについて検討している．ここで，述語が単純とは，述語がネストしておらず，かつ述語内部に演算子 “*and*”，“*or*”，“ | ” を含まないことを意味している．すなわち，述語内部に存在する演算子は，比較演算 “ = ” あるいは “! = ” だけである．

本稿では，上記 (1) の XPath 問い合わせの変換手法に関して，XPath 問い合わせの部分クラスを 1 回の走査によりトップダウンに変換する手法を提案する．本手法は，文献 [2], [9] と同様，等価変換規則に従って，与えられた問い合わせを変形する．それに際し，文献 [2], [9] で述べられている XPath の等価変換規則に，新たな規則を加えた．また，入力，ならびに出力される問い合わせ式を定義するために，新たに XPath の部分クラスに集合演算を追加した言語を定義し，本変換手法の正しさを証明した．そして，実験により，本変換手法が，述語が複雑な XPath 問い合わせの処理に有効であること，ならびに述語で指定される要素が少ないほど，その問い合わせ処理に有効であることを確かめた．

以降，まず 2 節で，XPath 問い合わせ処理の高速化に関する他の研究について述べ，3 節で変換手法の入出力を定義する．次に 4 節で変換手法を提案し，5 節でその変換の正しさを証明する．そして 6 節で，有効性評価のために実施した実験について述べ，最後に 7 節で現状の問題とその解決策について言及する．なお，本稿が対象としているのは XPath 1.0 [12] である．

2. 関 連 研 究

XPath 問い合わせ処理を効率良く実行する研究は最近になって，盛んに行なわれはじめた．この分野において初期の頃から行なわれている研究は，冗長なステップを消去することによって，XPath 問い合わせのサイズを小さくする手法 [1], [10], [13] である．また，よく似た手法として，問い合わせ中に現れるワイルドカードをできる限り消去することにより，処理効率を上げる手法 [4] が，最近提案されている．

他の最適化手法として，多項式時間で XPath 問い合わせを処理できる手法 [6], [7] が提案されている．この手法は，本手法と同様，集合演算を用いるが，与えられた XPath 問い合わせを全てノードテストの単位まで分解している点で，筆者らの提案手法と異なる．[6], [7] の手法では，全ての集合演算の被演算子に，あるノードテストに一致する全要素の全子要素が現れる．このため，集合演算の被演算子となる集合の要素数が非常に大きくなり，集合演算 1 つ 1 つの処理効率が悪くなるという問題がある．しかも，ノードテストの単位まで分解しているため，この問題は避けられない．一方，本稿の手法では，原子式 1 つ 1 つの処理法を自由に選択できるため，集合演算を用いる以外の効率よい処理を選択できる．筆者らは，1 節で述べた部分問題 (3) で十分な成果を上げることによってこの問題を解決し，[6], [7] の手法に比べて，より効率の良い処理法を実現した．

更に，問い合わせを等価変換規則に従って，より簡単な XPath 問い合わせに変換して処理する手法 [2] や，効率の良い SAX 的処理を実現するため，等価変換規則を用いて問い合わせ中の上方向軸を消去する手法 [9] が存在する．ここで上方向軸とは，*parent* や *ancestor* など，ドキュメントを上方向に走査する軸のことである．文献 [9] に示される手法は，上方向軸を消去する代わりに，述語を生成しているという点で，本提案手法とは対極に位置する．本手法は，ネストした述語を消去する代わりに，上方向軸を生成する．

また，リレーショナル DB の高速化手法の 1 つであるパイプライン処理を，XML データベースでも実現するため，問い合わせを代数式に変換する手法 [8] が知られている．

3. 問い合わせ言語クラスの定義

本節では，本稿で用いる問い合わせ言語の構文とその意味，そして変換への入力対象とする XPath 問い合わせのクラス，出力される問い合わせ (中間コードと呼ぶ) のクラスを順に定義する．

3.1 集合演算付き部分 XPath (ssXPath)

本変換手法は，XPath のある部分クラスに属する問い合わせを入力とし，中間コードを出力する．そこで，この入力対象の XPath のクラスと中間コードのクラスとを包含する，集合演算付き部分 XPath (以下 ssXPath と略記) を定義する．

3.1.1 構　　文

ssXPath の EBNF 記法による構文定義を図 1 に示す．図 1 に現れる $\frac{dom}{root}(P_{op})$ は，引数の P_{op} が指定する要素集合に根要素 *root* が含まれる場合は全要素集合 *dom* を返し，*root* が含まれない場合は空集合 $\emptyset$ を返す関数である．

ssXPath は，W3C 勧告として公開されている XPath [12] と以下の点で異なる．

- 集合演算 $\cup$ ならびに $\cap$ を含む．
- 関数 $\frac{dom}{root}(P_{op})$ を含む (8 式)．
- W3C が公開する XPath では軸 $AxisName$ に 13 の軸が定義されているのに対し，ssXPath では *self*，*child*，*parent*，*attribute* の 4 軸のみを扱う．また，3.2 節で述べるように，本提案手法は軸 *parent* ならびに親軸を辿るステップ “..” を含む XPath 問い合わせを入力として許さないので，*parent* ならびに “..” は他の軸やステップと区別し，これを別に定義している (13，14，15，20，21，22 式)．
- ノードテストは要素名だけで，*text*()，*comment*() などのノードテストは含まない (16 式)．
- 軸に名前空間を含まないため，名前空間接頭辞を用いた要素名の評価をすることができない (17 式)．
- 述語内は等式による比較のみで，不等式，加減乗除計算を含まない (27 式)．
- 等式演算 “ = ”，“! = ” の被演算子に boolean 型の式が現れない (27 式)．

$$
\begin{aligned}
(\text{集合演算付き部分 } XPath) &::= [\ Con_{or}\ ] && (1)\\
Con_{or} &::= Con_{an} \mid Con_{or}\ \text{“}\cup\text{”}\ Con_{an} && (2)\\
Con_{an} &::= Con_{pr} \mid Con_{an}\ \text{“}\cap\text{”}\ Con_{pr} && (3)\\
Con_{pr} &::= P_{op} \mid \text{“(”}\ Con_{or}\ \text{“)”} && (4)\\
P_{op} &::= AbsoluteP_{op} \mid RelativeP_{op} && (5)\\
AbsoluteP_{op} &::= \text{“/”}\ [\ RelativeP_{op}\ ] && (6)\\
RelativeP_{op} &::= P_{S.pr} \mid RelativeP_{op}\ \text{“/”}\ P_{S.pr} && (7)\\
P_{S.pr} &::= Step \mid Predicate\ \{Predicate\} \mid \text{“(”}\ P_{S.or}\ \text{“)”} \mid \frac{dom}{root}(P_{op}) && (8)\\
P_{S.or} &::= P_{S.an} \mid P_{S.or}\ \text{“}\cup\text{”}\ P_{S.an} && (9)\\
P_{S.an} &::= P_{op} \mid P_{S.an}\ \text{“}\cap\text{”}\ P_{op} && (10)\\
Step &::= AxisSpecifier\ NodeTest\ \{Predicate\} \mid AbbreviatedStep && (11)\\
AxisSpecifier &::= AxisName\ \text{“::”} \mid AbbreviatedAxisSpecifier && (12)\\
AxisName &::= A\text{-}Name1 \mid A\text{-}Name2 && (13)\\
A\text{-}Name1 &::= \text{“}self\text{”} \mid \text{“}child\text{”} \mid \text{“}attribute\text{”} && (14)\\
A\text{-}Name2 &::= \text{“}parent\text{”} && (15)\\
NodeTest &::= NameTest && (16)\\
NameTest &::= (\text{任意の名前}) && (17)\\
Predicate &::= \text{“[”}\ PredicateExpr\ \text{“]”} && (18)\\
PredicateExpr &::= Expr && (19)\\
AbbreviatedStep &::= AbbreviatedStep1 \mid AbbreviatedStep2 && (20)\\
AbbreviatedStep1 &::= \text{“.”} && (21)\\
AbbreviatedStep2 &::= \text{“..”} && (22)\\
AbbreviatedAxisSpecifier &::= \text{“@”} \mid \varepsilon && (23)\\
Expr &::= OrExpr && (24)\\
OrExpr &::= AndExpr \mid OrExpr\ \text{“}or\text{”}\ AndExpr && (25)\\
AndExpr &::= EqualityExpr \mid AndExpr\ \text{“}and\text{”}\ EqualityExpr && (26)\\
EqualityExpr &::= UnionExpr \mid UnionExpr\ \text{“}=\text{”}\ Literal \mid UnionExpr\ \text{“}!=\text{”}\ Literal && (27)\\
UnionExpr &::= PathExpr \mid UnionExpr\ \text{“|”}\ PathExpr && (28)\\
PathExpr &::= LocationPath \mid FilterExpr \mid FilterExpr\ \text{“/”}\ RelativeLocationPath && (29)\\
LocationPath &::= RelativeLocationPath \mid AbsoluteLocationPath && (30)\\
AbsoluteLocationPath &::= \text{“/”}\ [\ RelativeLocationPath\ ] && (31)\\
RelativeLocationPath &::= Step \mid Step\ \text{“/”}\ RelativeLocationPath && (32)\\
FilterExpr &::= PrimaryExpr \mid FilterExpr\ Predicate && (33)\\
PrimaryExpr &::= \text{“(”}\ Expr\ \text{“)”} \mid Literal && (34)\\
Literal &::= \text{‘”’\{(『”』以外の文字)\}‘”’} \mid \text{“’”\{(『’』以外の文字)\}“’”} && (35)
\end{aligned}
$$

図 1　集合演算付き部分 XPath (ssXPath) の構文定義

- XPath 問い合わせの式中に変数評価，数式演算，関数呼び出しを含まない (34 式).

本稿では紙面の都合上，構文定義で示した記号を表 1 のように略記する．さらに E，E' ならびに，eop，cmp という非終端記号を新しく導入し，これらを次のように定義する．

$$E ::= Predicate\ \{\ Predicate\ \}. \qquad E' ::= [E].$$

$$eop ::= \text{“}=\text{”} \mid \text{“}!=\text{”}. \qquad cmp ::= Path\ eop\ val.$$

3.1.2　意　　味

文献 [6] では，XPath 問い合わせの，集合演算を用いた意味定義が提案されている．この定義を拡張して，ssXPath の意味定義を図 2 のように与える．

コンテキストノード集合 N_0 における ssXPath 問い合わせ X の意味は，$S_{\rightarrow}[X](N_0)$ により与えられる．ここで，通常 N_0 は対象とする XML ドキュメント中の全要素集合 dom である．また χ^{-1} は軸 χ の逆軸を表す (表 2 参照)．関数 $T()$ は引数と

$$S_{\rightarrow}[Con_{or}\ \text{“}\cup\text{”}\ Con_{an}](N_0) := S_{\rightarrow}[Con_{or}](N_0) \cup S_{\rightarrow}[Con_{an}](N_0) \tag{36}$$

$$S_{\rightarrow}[Con_{an}\ \text{“}\cap\text{”}\ Con_{pr}](N_0) := S_{\rightarrow}[Con_{an}](N_0) \cap S_{\rightarrow}[Con_{pr}](N_0) \tag{37}$$

$$S_{\rightarrow}[\ \text{“(”}\ Con_{or}\ \text{“)”}\ ](N_0) := S_{\rightarrow}[Con_{or}](N_0) \tag{38}$$

$$S_{\rightarrow}[\ \text{“/”}\ ](N_0) := \chi(\{root\}) \tag{39}$$

$$S_{\rightarrow}[\ \text{“/”}\ P_{S.pr}](N_0) := S_{\rightarrow}[P_{S.pr}](\{root\}) \tag{40}$$

$$S_{\rightarrow}[P_{op}\ \text{“/”}\ P_{S.pr}](N_0) := S_{\rightarrow}[P_{S.pr}](S_{\rightarrow}[P_{op}](N_0)) \tag{41}$$

$$S_{\rightarrow}[\ \text{“(”}\ P_{S.or}\ \text{“)”}\ ](N_0) := S_{\rightarrow}[P_{S.or}](N_0) \tag{42}$$

$$S_{\rightarrow}[\frac{dom}{root}(P_{op})](N_0) := \frac{dom}{root}(S_{\rightarrow}[P_{op}](N_0)) \tag{43}$$

$$S_{\rightarrow}[P_{S.or}\ \text{“}\cup\text{”}\ P_{S.an}](N_0) := S_{\rightarrow}[P_{S.or}](N_0) \cup S_{\rightarrow}[P_{S.an}](N_0) \tag{44}$$

$$S_{\rightarrow}[P_{S.an}\ \text{“}\cap\text{”}\ P_{op}](N_0) := S_{\rightarrow}[P_{S.an}](N_0) \cap S_{\rightarrow}[P_{op}](N_0) \tag{45}$$

$$S_{\rightarrow}[\chi\ \text{“::”}\ t[e_1][e_2]\dots[e_k]](N_0) := \chi(N_0) \cap T(t) \cap S_{\leftarrow}[e_1\ and\ e_2\ and\ \cdots\ and\ e_k] \tag{46}$$

$$S_{\rightarrow}[\ \text{“@”}\ t[e_1][e_2]\dots[e_k]](N_0) ::= attribute(N_0) \cap T(t) \cap S_{\leftarrow}[e_1\ and\ e_2\ and\ \cdots\ and\ e_k] \tag{47}$$

$$S_{\rightarrow}[t[e_1][e_2]\dots[e_k]](N_0) ::= child(N_0) \cap T(t) \cap S_{\leftarrow}[e_1\ and\ e_2\ and\ \cdots\ and\ e_k] \tag{48}$$

$$S_{\rightarrow}[[e_1][e_2]\dots[e_k]](N_0) := S_{\leftarrow}[e_1\ and\ e_2\ and\ \cdots\ and\ e_k] \cap N_0 \tag{49}$$

$$S_{\rightarrow}[\ \text{“.”}\ ](N_0) ::= N_0 \tag{50}$$

$$S_{\rightarrow}[\ \text{“..”}\ ](N_0) ::= parent(N_0) \tag{51}$$

$$S_{\leftarrow}[e_{or}\ \text{“}or\text{”}\ e_{an}] := S_{\leftarrow}[e_{or}] \cup S_{\leftarrow}[e_{an}] \tag{52}$$

$$S_{\leftarrow}[e_{an}\ \text{“}and\text{”}\ e_{eq}] := S_{\leftarrow}[e_{an}] \cap S_{\leftarrow}[e_{eq}] \tag{53}$$

$$S_{\leftarrow}[e_{uni}\ \text{“}\mid\text{”}\ e_{pr}] := S_{\leftarrow}[e_{uni}] \cup S_{\leftarrow}[e_{pr}] \tag{54}$$

$$S_{\leftarrow}[\text{“(”}e_{pr}\text{“)”}E'] := S_{\leftarrow}[e_{pr}\ E'] \tag{55}$$

$$S_{\leftarrow}[\text{“(”}e_{uni} \mid e_{pr}\text{“)”}E'] := S_{\leftarrow}[\text{“(”}e_{uni}\text{“)”}\ E'] \cup S_{\leftarrow}[e_{pr}\ E'] \tag{56}$$

$$S_{\leftarrow}[\text{“(”}e_{pr}\text{“)”}\ E'\ \text{“/”}\pi_r] := S_{\leftarrow}[e_{pr}\ E'\ \text{“/”}\pi_r] \tag{57}$$

$$S_{\leftarrow}[\text{“(”}e_{uni} \mid e_{pr}\text{“)”}\ E'\ \text{“/”}\pi_r] := S_{\leftarrow}[\text{“(”}e_{uni}\text{“)”}\ E'\ \text{“/”}\pi_r] \cup S_{\leftarrow}[e_{pr}\ E'\ \text{“/”}\pi_r] \tag{58}$$

$$S_{\leftarrow}[e_{uni}\ \text{“}\mid\text{”}\ e_{pr}\ eop\ val] := S_{\leftarrow}[e_{uni}\ eop\ val] \cup S_{\leftarrow}[e_{pr}\ eop\ val] \tag{59}$$

$$S_{\leftarrow}[\text{“(”}e_{pr}\text{“)”}\ E'\ eop\ val] := S_{\leftarrow}[e_{pr}\ E'\ eop\ val] \tag{60}$$

$$S_{\leftarrow}[\text{“(”}e_{uni} \mid e_{pr}\text{“)”}\ E'\ eop\ val] := S_{\leftarrow}[\text{“(”}e_{uni}\text{“)”}\ E'\ eop\ val] \cup S_{\leftarrow}[e_{pr}\ E'\ eop\ val] \tag{61}$$

$$S_{\leftarrow}[\text{“(”}e_{pr}\text{“)”}\ E'\ \text{“/”}\pi_r\ eop\ val] := S_{\leftarrow}[e_{pr}\ E'\ \text{“/”}\pi_r\ eop\ val] \tag{62}$$

$$S_{\leftarrow}[\text{“(”}e_{uni} \mid e_{pr}\text{“)”}\ E'\ \text{“/”}\pi_r\ eop\ val] := S_{\leftarrow}[\text{“(”}e_{uni}\text{“)”}\ E'\ \text{“/”}\pi_r\ eop\ val] \cup S_{\leftarrow}[e_{pr}\ E'\ \text{“/”}\pi_r\ eop\ val] \tag{63}$$

$$S_{\leftarrow}[\ \text{“/”}\ ] := \frac{dom}{root}(\{root\}) \tag{64}$$

$$S_{\leftarrow}[\ \text{“/”}\ \pi_r] := \frac{dom}{root}(S_{\leftarrow}[\pi_r]) \tag{65}$$

$$S_{\leftarrow}[\chi\ \text{“::”}\ t[e_1][e_2]\dots[e_k]\ \text{“/”}\ \pi_r] := \chi^{-1}(S_{\leftarrow}[e_1\ and\ e_2\ and\ \cdots\ and\ e_k] \cap T(t) \cap S_{\leftarrow}[\pi_r]) \tag{66}$$

$$S_{\leftarrow}[t[e_1][e_2]\dots[e_k]\ \text{“/”}\ \pi_r] := parent(S_{\leftarrow}[e_1\ and\ e_2\ and\ \cdots\ and\ e_k] \cap T(t) \cap S_{\leftarrow}[\pi_r]) \tag{67}$$

$$S_{\leftarrow}[\chi\ \text{“::”}\ t[e_1][e_2]\dots[e_k]] := \chi^{-1}(T(t) \cap S_{\leftarrow}[e_1\ and\ e_2\ and\ \cdots\ and\ e_k]) \tag{68}$$

$$S_{\leftarrow}[\ \text{“@”}\ t[e_1][e_2]\dots[e_k]] := parent(T(t) \cap S_{\leftarrow}[e_1\ and\ e_2\ and\ \cdots\ and\ e_k]) \tag{69}$$

$$S_{\leftarrow}[t[e_1][e_2]\dots[e_k]] := parent(T(t) \cap S_{\leftarrow}[e_1\ and\ e_2\ and\ \cdots\ and\ e_k]) \tag{70}$$

$$S_{\leftarrow}[\chi\ \text{“::”}\ t[e_1][e_2]\dots[e_k]\ eop\ val] := S_{\leftarrow}[\chi :: t[e_1\ and\ e_2\ and\ \cdots\ and\ e_k\ and\ \text{“.”}\ eop\ val]] \tag{71}$$

$$S_{\leftarrow}[\ \text{“@”}\ t[e_1][e_2]\dots[e_k]\ eop\ val] := S_{\leftarrow}[attribute :: t[e_1\ and\ e_2\ and\ \cdots\ and\ e_k\ and\ \text{“.”}\ eop\ val]] \tag{72}$$

$$S_{\leftarrow}[t[e_1][e_2]\dots[e_k]\ eop\ val] := S_{\leftarrow}[child :: t[e_1\ and\ e_2\ and\ \cdots\ and\ e_k\ and\ \text{“.”}\ eop\ val]] \tag{73}$$

$$S_{\leftarrow}[\ \text{“.”}\ eop\ val] := T(.\ eop\ val) \tag{74}$$

図 2　集合演算付き部分 XPath (ssXPath) の意味定義

$$atom ::= Path\ [\ \text{“[”}\ Path\ [\ \text{“}=\text{”}\ val\ ]\ \text{“]”}\ ]\ Path' \tag{75}$$
$$Path ::= AbsolutePath \mid RelativePath \tag{76}$$
$$AbsolutePath ::= \text{“/”}\ [\ RelativePath\ ] \tag{77}$$
$$RelativePath ::= NoPredicateStep \mid RelativePath\ \text{“/”}\ NoPredicateStep \tag{78}$$
$$NoPredicateStep ::= AxisName\ \text{“::”}\ NodeTest \mid AbbreviatedStep \tag{79}$$
$$Path' ::= [\ Path\ ] \tag{80}$$

図 3　原子式の定義

$$(\text{中間コード}) ::= AtomUnion \mid \text{“(”}\ (\text{中間コード})\ \text{“)”}\ AbsolutePath \tag{81}$$
$$AtomUnion ::= AtomIntersection\ \{\ \text{“}\cup\text{”}\ AtomIntersection\} \tag{82}$$
$$AtomIntersection ::= atom\ \{\ \text{“}\cap\text{”}\ atom\} \tag{83}$$

図 4　中間コードの定義

して，要素名，属性名，または式 “$.=val$”，“$.!=val$” をとる．引数が要素名，属性名の場合は，その要素名または属性名に一致する要素集合を返し，引数が “$.=val$” の場合は，値 val を持つ要素集合を返す．引数が “$.!=val$” の場合は，値 val を持つ要素集合を dom から除いた集合を返す．最後に，述部の式の添字が k であるものに関しては $k=0$ となる場合があるが，この場合，この述部は無視する．

3.2　入力対象の XPath 問い合わせ

本手法が入力として許す XPath 問い合わせ π の軸には $A\text{-}Name2$ (すなわち軸 “$parent$”) が含まれてはならず，また，ステップ中に $AbbreviatedStep2$ (すなわち “..”) が含まれてもいけない．この π に関する制限は，変換の正しさを保証するために必要であるが，詳細に関しては 7 節で述べる．また，問い合わせ π は空問い合わせ ε であってもよい．

3.3　原子式と中間コード

原子式の構文を図 3 により定義する．直感的には，式中に述語を最大で 1 つ含み，かつその述語内部の式には “and”，“or”，“|”，述語のいずれも含まれないような XPath 問い合わせが原子式である．

また中間コードの構文を図 4 により定義する．直感的には，原子式と，∩ (集合積)，∪ (集合和)，/.. (親)，/ (子) の 4 演算子のみから成る式が中間コードである．

表 3 に集合演算付き部分 XPath (ssXPath)，入力対象の XPath 問い合わせ (Input)，中間コード (OptCode)，原子式 (atom) の例を示す．(Q2) は述語がネストしているため，中間コードではない．(Q3) は式中に “..” を含むため，入力 XPath ではない．(Q4) は左丸括弧よりも左側にパスが存在せず，括弧の中の式が全て原子式であるため，中間コードであるが，(Q5) は左丸括弧よりも左側にパス `/A` が存在するため，中間コードではない．(Q6) は等号が 2 つあり，最初の等号を評価した結果が boolean 型になるため，2 つ目の等号の被演算子が boolean 型となる．このため ssXPath ではない．

表 1　記号の略記

正式名	略記
$AxisName$	χ
$NodeTest$	t
$PredicateExpr$	e
$OrExpr$	e_{or}
$AndExpr$	e_{an}
$EqualityExpr$	e_{eq}
$UnionExpr$	e_{uni}
$PathExpr$	e_{pr}
$LocaionPath$	π
$RelativeLocationPath$	π_r
$Literal$	val

表 2　軸とその逆軸の関係

軸 (χ)	逆軸 (χ^{-1})
self	self
child	parent
attribute	parent

4. 変換手法

この節では任意の ssXPath 問い合わせを中間コードに変換する手法について述べる．

本稿が提案する変換手法は，与えられた XPath 問い合わせを先頭から順に読み込んでいき，トークン毎に順に処理していくことで，トップダウンに一回の走査だけで中間コードを得る．ここでトークンとは，次の記号列を指す．

$$Path,\ val,\ \text{“}=\text{”},\ \text{“}!=\text{”},\ \text{“[”},\ \text{“]”},$$
$$\text{“(”},\ \text{“)”},\ \text{“}or\text{”},\ \text{“}and\text{”},\ \text{“}\mid\text{”}$$

変換は，複雑な述語を複数の単純な述語に分けるステップと，そのステップにより得られる式を中間コードに変換するステップの 2 つに分けられる．

まず，述語の単純化のステップは図 5 に与えた変換アルゴリズムにより実現できる．図中に現れる $(/..)^{|Path|}$ は $Path$ 内に含まれる要素の数だけ，親ノードを辿ることを意味している．XPath 問い合わせ π を変換した式は $C^{\rightarrow}_{\pi}(\pi)$ で与えられる．この変換で得られる式は集合和ならびに集合積の被演算子が述語のみの場合があり，これは必ずしも中間コードとはならない．そのため，中間コードに変換するステップが必要となる．

中間コードに変換するステップでは，述語の単純化のステッ

表 3　ssXPath，入力 XPath (Input)，中間コード (OptCode)，原子式の例 (atom)

	問い合わせ式	ssXPath	Input	OptCode	atom
(Q1)	`/A/B/C[X="x"]/Y`	○	○	○	○
(Q2)	`/A/B[C[X="x" and Y="y"]]/D`	○	○	×	×
(Q3)	`/A/B[C[X="x" and Y="y"]]/../D`	○	×	×	×
(Q4)	`(/A/B/C[X="x"]∩/A/B/C[Y="y"])/../D`	○	×	○	×
(Q5)	`/A(B/C[X="x"]∩B/C[Y="y"])/../D`	○	×	×	×
(Q6)	`/A/B/C[X="x"= true]/Y`	×	×	×	×

プで得られた式中に，次の a–d 式 (補題 6–9) の左辺に一致する部分式が存在する限りそれを右辺の形に展開する．これによって中間コードが得られる．

a. $\pi_1(\pi_2 \cup \pi_3) \Longrightarrow \pi_1/\pi_2 \ \cup \ \pi_1/\pi_3$.
b. $\pi_1(\pi_2 \cap \pi_3) \Longrightarrow \pi_1/\pi_2 \ \cap \ \pi_1/\pi_3$.
c. $(\pi_1 \cup \pi_2)\pi_3 \Longrightarrow \pi_1/\pi_3 \ \cup \ \pi_2/\pi_3$.
d. $(\pi_1 \cap \pi_2)\pi_3 \Longrightarrow \pi_1/\pi_3 \ \cap \ \pi_2/\pi_3$．(ただし軸は $A\text{-}Name1$ に限る．)

例えば，表 3 中の (Q2) を入力として，本変換手法を用いる場合を考える．このとき，図 5 の変換手法を用いると

`/A/B((C((([X="x"])∩([Y="y"])))/..))/D`

が得られる．この式は左丸括弧の左側にパス `/A/B` が現れるため中間コードではない．そこで a–d 式によって，この式に存在する括弧を展開していくことで，中間コードである (Q4) が得られる．

今，わかりやすさのため，中間コードへの変換方法をこれら 2 つのステップに分けた．しかし，後者のステップで行なわれる処理は括弧の展開だけであるので，これらのステップは同時に行なうことができる．例えば，括弧の前にある式を括弧内に展開する a，b 式 (補題 6，7) の場合，括弧の前にある式をメモリに記憶させておき，括弧の中身の式を出力する際に，この括弧の前にあった式も同時に出力することで，実現できる．また，括弧の後ろにある式を括弧内に展開する c，d 式 (補題 8，9) の場合，括弧の後ろにある式を出力する段階で，既に出力しているそれぞれの式に，括弧の後ろにあった式を付け加えることで，実現できる．

以上のような考え方により，中間コードへの変換は 1 回の走査により実現できる．

5. 変換の正しさの証明

この節では，与えられた XPath 問い合わせの意味が変換により変わらないこと，ならびに得られた式が中間コードであることを証明する．

5.1 意味が変わらないことの証明

5.1.1 準　　備

まず，新しく表記法を導入し，いくつかの補題とその証明を与える (紙面の都合上，証明を一部省略する)．

［表記 1］ 要素集合 X 内の全ての要素から，軸 χ で辿ることができる全ての要素集合を $\chi(X)$ と書く．すなわち

$$\chi(X) = \bigcup_{x \in X} \{y \mid y \text{ は } x \text{ から軸} \chi \text{で辿ることができる要素 }\}.$$

また，軸 χ が，“*parent*”，“*child*” または “*attribute*” の場合に関しては，それぞれ $p(X)$，$c(X)$，$@(X)$ と略記する．

［補題 1］ $c(A \cup B) = c(A) \cup c(B)$.

［補題 2］ $@(A \cup B) = @(A) \cup @(B)$.

［補題 3］ $p(A \cup B) = p(A) \cup p(B)$.

［補題 4］ $p(A \cap B) \subseteqq p(A) \cap p(B)$.

証明 $p(A \cap B) \subseteqq p(A)$ かつ，$p(A \cap B) \subseteqq p(B)$ であるので，$p(A \cap B) \subseteqq p(A) \cap p(B)$ である．　□

なお，補題 4 の右辺に関して，$p(A) \cap p(B) = (p(A \cap \overline{B}) \ \cap \ p(\overline{A} \cap B)) \ \cup \ p(A \cap B)$ であるため，一般に左辺と右辺は等しくない．$p(A \cap \overline{B}) \ \cap \ p(\overline{A} \cap B) = \emptyset$ が成立するとき，かつそのときにのみ等号は成立する．

［補題 5］ $p(A \cap c(B)) = p(A) \cap B$.

証明 $p(\overline{A} \cap c(B))$ を考える．$p()$ の引数が $c(B)$ の部分集合であるので，$p(\overline{A} \cap c(B)) \subseteqq B$．一方，$p(A \cap \overline{c(B)})$ を考える．$p()$ の引数が $\overline{c(B)}$ の部分集合であるので，$p(A \cap \overline{c(B)}) \subseteqq \overline{B}$．よって，$p(\overline{A} \cap c(B)) \cap p(A \cap \overline{c(B)}) = \emptyset$.

ここで，補題 4 とその直後の議論において，B を $c(B)$ とすると，$p(A \cap c(B)) = p(A) \cap B$ が得られる．　□

［補題 6］ $S_{\rightarrow}[\pi_1(\pi_2 \cup \pi_3)](N_0) = S_{\rightarrow}[\pi_1/\pi_2 \cup \pi_1/\pi_3](N_0)$ (文献 [2] の left-distributivity).

［補題 7］ $S_{\rightarrow}[\pi_1(\pi_2 \cap \pi_3)](N_0) \ = \ S_{\rightarrow}[\pi_1/\pi_2 \cap \pi_1/\pi_3](N_0)$ (ssXPath の意味定義より示せる).

［補題 8］ $S_{\rightarrow}[(\pi_1 \cup \pi_2)\pi_3](N_0) = S_{\rightarrow}[\pi_1/\pi_3 \cup \pi_2/\pi_3](N_0)$ (文献 [2] の right-distributivity).

［補題 9］ $S_{\rightarrow}[(\pi_1 \cap \pi_2)\pi_3](N_0) \ \subseteqq \ S_{\rightarrow}[\pi_1/\pi_3 \cap \pi_2/\pi_3](N_0)$ (ssXPath の意味定義より示せる).

5.1.2 証　　明

［定理 1］ 本変換手法によって得られる問い合わせ式は，入力された問い合わせと等価である．

証明 前節で述べたように，本変換手法は複雑な述語を複数の単純な述語に分けるステップと，そのステップにより得られる式を中間コードに変換するステップに分けられる．このうち後者に関しては，本節の準備で述べたように，補題 6–9 が成立しているため，ステップの前後で式の意味が変わらない．そこで，以降は，前者に関して，図 5 の変換手法の式 (84)–(89) の意味がそれぞれ変換の前後で等しいことを帰納的に示す (紙面の都合により，(88) 式のみ記す)．

(88) 式

この変換規則が用いられるのは e_{eq} 中に等号演算 eop を含む場合と含まない場合の 2 つに分けられる．

$$C^{\rightarrow}_{\pi}(\pi) \longrightarrow [\ Path\ [\ C^{\rightarrow}_{E}(E_1)\ ]\ ]\ \{\ Path_a\ [\ C^{\rightarrow}_{E}(E_2)\ ]\ \} \quad (\text{where} \quad \pi ::= [\ Path\ [\ E_1\ ]\ ]\ \{\ Path_a\ [\ E_2\ ]\ \}). \tag{84}$$

$$C^{\rightarrow}_{E}(E) \longrightarrow \text{“(”}\ C^{\leftarrow}_{or}(e_{or_1})\ \text{“)”}\ \{\text{“}\cap\text{”}\ \text{“(”}\ C^{\leftarrow}_{or}(e_{or_2})\ \text{“)”}\}\ [\ \text{“}\cap\ [\ .\text{”}\ eop\ val\ \text{“]”}\ ]$$
$$(\text{where} \quad E ::= \text{“[”}\ e_{or_1}\ \text{“]”}\ \{\ \text{“[”}\ e_{or_2}\ \text{“]”}\ \}\ [\ eop\ val\ ]). \tag{85}$$

$$C^{\leftarrow}_{or}(e_{or}) \longrightarrow C^{\leftarrow}_{an}(e_{an_1})\ \{\ \text{“}\cup\text{”}\ C^{\leftarrow}_{an}(e_{an_2})\ \} \quad (\text{where} \quad e_{or} ::= e_{an_1}\ \{\ \text{“}or\text{”}\ e_{an_2}\ \}). \tag{86}$$

$$C^{\leftarrow}_{an}(e_{an}) \longrightarrow C^{\leftarrow}_{eq}(e_{eq_1})\ \{\ \text{“}\cap\text{”}\ C^{\leftarrow}_{eq}(e_{eq_2})\ \} \quad (\text{where} \quad e_{an} ::= e_{eq_1}\ \{\ \text{“}and\text{”}\ e_{eq_2}\ \}). \tag{87}$$

$$C^{\leftarrow}_{eq}(e_{eq}\ [\ E_0\ ]\ [\ \text{“/”}\ e_{pr_0}]\ [eop\ val_0\ ]\ )$$
$$\longrightarrow \text{“(”}\ C^{\leftarrow}_{pr}(e_{pr_1}\ [\ eop\ val\ ]\ [\ E_0\ ]\ [\ \text{“/”}\ e_{pr_0}]\ [\ eop\ val_0\ ]\ )$$
$$\{\ \text{“}\cup\text{”}\ C^{\leftarrow}_{pr}(e_{pr_2}\ [\ eop\ val\ ]\ [\ E_0\ ]\ [\ \text{“/”}\ e_{pr_0}]\ [\ eop\ val_0\ ]\ )\ \}\ \text{“)”}$$
$$(\text{where} \quad e_{eq} ::= e_{pr_1}\ \{\ \text{“|”}\ e_{pr_2}\ \}\ [\ eop\ val\ ]). \tag{88}$$

$$C^{\leftarrow}_{pr}(e_{pr}\ [\ eop\ val_0\ ]) \longrightarrow \begin{cases} \text{“[”}\ Path_r\ [\ eop\ val_0\ ]\ \text{“]”} \\ \qquad (\text{if} \quad e_{pr} ::= Path_r). \\ \frac{dom}{root}(\ \text{“[”}\ Path_r\ [\ eop\ val_0\ ]\ \text{“]”}\ ) \\ \qquad (\text{if} \quad e_{pr} ::= \text{“/”}\ Path_r). \\ \text{“(”}\ C^{\leftarrow}_{or}(e_{or})\ \text{“)”} \\ \qquad (\text{if} \quad e_{pr} ::= \text{“(”}\ e_{or}\ \text{“)”}). \\ \text{“(”}\ C^{\leftarrow}_{eq}(e_{eq}\ [\ eop\ val_0\ ])\ \text{“)”} \\ \qquad (\text{if} \quad e_{pr} ::= \text{“(”}\ e_{eq}\ \text{“)”}). \\ \text{“(”}\ C^{\leftarrow}_{eq}(e_{eq}\ [\ E\ ]\ \text{“/”}\ e_{pr_1}\ [\ eop\ val_0\ ])\ \text{“)”} \\ \qquad (\text{if} \quad e_{pr} ::= \text{“(”}\ e_{eq}\ \text{“)”}\ [\ E\ ]\ \text{“/”}\ e_{pr_1}). \\ Path_r\ \text{“(”}\ C^{\rightarrow}_{E}(E\ [\ eop\ val_0\ ]\ )\ \text{“)”}\ (/..)^{|Path_r|} \\ \qquad (\text{if} \quad e_{pr} ::= Path_r\ E). \\ Path_r\ \text{“(”}\ C^{\rightarrow}_{E}(E)\ \text{“}\cap\text{”}\ \text{“(”}\ C^{\leftarrow}_{pr}(e_{pr_1}\ [\ eop\ val_0\ ]\ )\ \text{“)”}\ \text{“)”}\ (/..)^{|Path_r|} \\ \qquad (\text{if} \quad e_{pr} ::= Path_r\ E\ \text{“/”}\ e_{pr_1}). \\ \frac{dom}{root}(\text{“[”}\ Path_r\ \text{“(”}\ C^{\rightarrow}_{E}(E\ [\ eop\ val_0\ ]\ )\ \text{“)”}\ (/..)^{|Path_r|}\ \text{“]”}\ ) \\ \qquad (\text{if} \quad e_{pr} ::= \text{“/”}\ Path_r\ E). \\ \frac{dom}{root}(\text{“[”}\ Path_r\ \text{“(”}\ C^{\rightarrow}_{E}(E)\ \text{“}\cap\text{”}\ \text{“(”}\ C^{\leftarrow}_{pr}(e_{pr_1}\ [\ eop\ val_0\ ]\ )\ \text{“)”}\ \text{“)”}\ (/..)^{|Path_r|}\ \text{“]”}) \\ \qquad (\text{if} \quad e_{pr} ::= \text{“/”}\ Path_r\ E\ \text{“/”}\ e_{pr_1}). \end{cases} \tag{89}$$

図 5　変換アルゴリズム

e_{eq} に等号が含まれる場合，構文の定義から等号の後ろに “E_0”，“$/e_{pr_0}$”，“$eop\ val_0$” のいずれも続くことはない．よってこの場合，変換前の式が返す頂点集合は，定義 $S_{\leftarrow}[e_{uni}$ “|” $e_{pr}\ eop\ val]$ から

$$S_{\leftarrow}[e_{pr_1}\ eop\ val] \cup \cdots \cup S_{\leftarrow}[e_{pr_k}\ eop\ val] \tag{90}$$

に等しい．一方変換後の式が返す頂点集合は

$$S_{\leftarrow}[C^{\leftarrow}_{pr}(e_{pr_1}\ eop\ val)] \cup \cdots \cup S_{\leftarrow}[C^{\leftarrow}_{pr}(e_{pr_k}\ eop\ val)] \tag{91}$$

であるので，帰納法の仮定 $S_{\leftarrow}[e_{pr}\ \ eop\ \ val] = S_{\leftarrow}[C^{\leftarrow}_{pr}(e_{pr}\ eop\ val)]$ から変換の前後で等しいことが証明された．

次に，e_{eq} に等号が含まれない場合，変換前の式が返す頂点集合は，定義 $S_{\leftarrow}[(e_{uni}$ “|” $e_{pr})E'$ “/” $\pi_r\ eop\ val]$ から

$$S_{\leftarrow}[e_{pr_1}[e]/e_0\ eop\ val] \cup \cdots \cup S_{\leftarrow}[e_{pr_k}[e]/e_0\ eop\ val] \tag{92}$$

に等しい．一方変換後の式が返す頂点集合は

$$S_{\leftarrow}[C^{\leftarrow}_{pr}(e_{pr_1}[e]/e_0\ eop\ val)] \cup \cdots \cup S_{\leftarrow}[C^{\leftarrow}_{pr}(e_{pr_k}[e]/e_0\ eop\ val)] \tag{93}$$

であるので，帰納法の仮定 $S_{\leftarrow}[e_{pr}[e]/e_0\ \ eop\ \ val] = S_{\leftarrow}[C^{\leftarrow}_{pr}(e_{pr_k}[e]/e_0\ eop\ val)]$ から，この場合も変換の前後で等しいことが証明された．

以上より，(88) 式による変換は正しい．　□

5.2　得られた式が中間コードであることの証明

［定理 2］　提案する変換手法により中間コードが得られる．

証明　与えられた問い合わせ中の述語に現れる全ての “*and*”，“*or*”，“|” は述語評価中，必ず 1 度は式 (86)–(88) を通るため，全て取り除かれる．また，述語がネストする場合，式 (89) の 6–9 番目の式により評価された後，式 (85) により評価されることで，角括弧が取り除かれ，これにより必ず述語内部の述語が取り払われる．これが再帰的に行なわれるので，最終的に述語のネストは残らない．以上により，図 5 による変換で，全ての述語内部は $Path$ または cmp のみとなる．

この後，補題 6–9 を用いて展開することで，左丸括弧の前に存在していた全てのパスが丸括弧内部に入れられる．よって，本変換手法により得られる式は中間コードである．　□

6. 評価実験

提案した XPath 問い合わせの変換手法と，XPath 問い合わせの処理を高速化する問題を 3 つの部分問題に分割する本アプローチの 2 つについて，評価実験を行なった．本実験では，OS Windows XP Professional, CPU Pentiun4 3.6GHz，メモリ 3GB の PC を用い，1 節で述べた部分問題 (2)，(3) の成果を組み込んだネイティブ XML データベースを使用した．この

データベースに，XMark [11] によって生成した XML 文書を格納し，84 の問い合わせを与え，その処理に要する時間を測定した．紙面の都合上，84 の問い合わせの内，特徴的な値を示した以下の 3 つの問い合わせについて述べる．

(Q16) /site/regions/africa/item[location=
“United States” and quantity=“1”]/name

(Q33) /site/people/person[profile/age=”33” and
profile/education=”Graduate School” and
address/zipcode=”26” and address/country=”Zaire”
and address/city=”Cleveland”]/name

(Q47) /site/open_auctions/open_auction[initial=
“323.41” and current=“446.41”]/type

まず，提案する変換手法の有効性を確かめるために，何の変換も施されていない問い合わせと，その問い合わせを中間コードに変換した式とをデータベースに与え，その処理に費された時間を測定した (実験 1)．実験 1 では，サイズ 5,750KB，要素数 83,798 の XML 文書 1 と，サイズ 11,679KB，要素数 167,864 の XML 文書 2 を用いた．この結果を，表 4 に掲げた．(Q16), (Q47) と (Q33) を比較すると，and の数が多い (Q33) の方が，変換の前後で実行時間の差が顕著に現れている．このように，本変換手法は述語が複雑なときほど効果的である．

次に，高速化問題に対する本アプローチの有効性を確かめるために, [6] の手法を実装し，問い合わせの処理時間を測定した (実験 2)．実験 2 では，サイズ 95KB，要素数 4,106 の XML 文書を用い，実験 1 と同じ 84 の問い合わせを与えた．本稿では，紙面の都合上，(Q16) と (Q47) の評価結果を表 5 に掲げる．この手法では，用いた XML 文書のサイズが，実験 1 で用いた XML 文書 2 の 100 分の 1 以下であるにも関わらず，処理に多くの時間を要している．これより，本手法は [6] の手法に比較して実用的であると考えられる．また，実験 1 において (Q16) と (Q47) は構造が似ているが，(Q16) は変換により処理が遅くなるのに対し，(Q47) は変換により処理が速くなる．この速度差は部分問題 (3) の原子式の高速化問題の解決方法に依存しており，本実験で利用した XML データベースでは，述語を満たす要素数が少ないときほど，変換による効果が大きい．

7. ま と め

本稿では XPath 問い合わせの処理を高速化するため，制限された XPath 問い合わせのクラスを入力とし，中間コードを出力する変換アルゴリズムを提案した．そして，この変換手法の正しさ，ならびに実施した評価実験について報告した．

本変換手法では入力 XPath 問い合わせの軸を $A\text{-}Name2$ (すなわち “*parent*”) が含まれないものに制限し，ステップも “..” が含まれないものに制限した．これは変換式 (89) の 6–9 番目の式において，軸 “*parent*” やステップ “..” が含まれている場合，正しく変換できないためである．しかし，これらの変換式は述語にネストがある場合にのみ実行されるので，入力 XPath の述語にネストがない場合であれば，軸やステップを制限する必要はない．一方，現在の変換手法には，W3C で定義されている XPath の全ての軸を扱えないことや，比較演算の被演算子として boolean 型の式を扱えないこと，数式を扱えないことといった制限が存在する．よって今後の課題として，より広い XPath のクラスに対する変換手法の考案が挙げられる．また，本変換手法が有効となる複雑な述語をもつ XPath 問い合わせのより厳密な考察，ならびに，XML ビューを生成する RDB に対する本手法の有効性に関する考察も今後の課題である．

表 4　Q16，Q33，Q47 に対する実験 1 の結果 (単位：ms)

	XML 文書 1		XML 文書 2	
式番	変換前	変換後	変換前	変換後
(Q16)	22	64	47	179
(Q33)	1354	187	2677	114
(Q47)	444	20	877	20

表 5　Q16，Q47 に対する実験 2 の結果 (単位：ms)

式番	時間	式番	時間
(Q16)	1188	(Q47)	1141

謝　　辞

常日頃より御指導を下さる大阪大学 藤原融教授に心から感謝致します．また，文献 [6] について御教示下さった奈良先端科学技術大学院大学 高田喜朗助手，そして本研究をご支援下さる株式会社メディアフュージョン 榊原淳社長に深く感謝致します．

文　　献

[1] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of tree pattern queries. In *SIGMOD Conference*, 2001.

[2] M. Benedikt, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. In *ICDT*, pages 79–95, 2003.

[3] S. Boag, D. D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. Xquery 1.0: An XML query language. *http://www.w3.org/TR/xquery/*, 2003.

[4] C. Y. Chan, W. Fan, and Y. Zeng. Taming XPath queries by minimizing wildcard steps. In *VLDB*, pages 156–167, 2004.

[5] J. Clark. XSL transformations (XSLT) 1.0. *http://www.w3.org/TR/xslt/*, 1999.

[6] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *VLDB*, pages 95–106, 2002.

[7] G. Gottlob, C. Koch, and R. Pichler. XPath query evaluation: Improving time and space efficiency. In *ICDE*, pages 379–390, 2003.

[8] S. Helmer, C.-C. Kanne, and G. Moerkotte. Optimized translation of XPath into algebraic expressions parameterized by programs containing navigational primitives. In *WISE*, pages 215–224, 2002.

[9] D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking forward. In *EDBT Workshops*, pages 109–127, 2002.

[10] P. Ramanan. Efficient algorithms for minimizing tree pattern queries. In *SIGMOD Conference*, pages 299–309, 2002.

[11] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proc. VLDB*, pages 974–985, 2002.

[12] W3C. XPath recommendation. *http://www.w3c.org/TR/xpath/*, 1999.

[13] P. T. Wood. Minimising simple XPath expressions. In *WebDB*, pages 13–18, 2001.