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

2004－DBS－133 (1)
2004－ＦＩ－ 75 (1)
2004／5／13

FLWOR Arranging; 入れ子FLWORブロックを伴うXQueryの書き換えによる最適化手法

加藤弘之 日高宗一郎

国立情報学研究所

本稿ではXMLデータに対する展開された問合せを対象とした書き換えによる最適化手法を提案する．ビューに対する選択条件と要求する検索結果をビュー定義の問合せブロックに移動することで，選択条件の早期評価と冗長な計算を避けることができ結果として最適化された問合せとなる．このような最適化手法FLWOR Arrangingにおいて中心的な役割を果たしている式型木について記述する．

FLWOR Arranging; Optimizing nested FLWOR bloks in XQuery by rewriting

Hiroyuki KATO and Soichiro HIDAKA

National Institute of Informatics

In this paper, we propose a query optimization method, called "FLWOR Arranging", based on two techniques. One is from pushing selection and another is from constant folding. We introduce a ExprTypeTree for XQuery expression based on their functional feature.

1. はじめに

本稿ではXMLデータに対する展開された問合せを対象とした書き換えによる最適化手法を提案する．ビューに対する選択条件と要求する検索結果をビュー定義の問合せブロックに移動することで，選択条件の早期評価と冗長な計算を避けることができ結果として最適化された問合せとなる．

1.1 例による説明

この節では非常に簡単な例を用いて，本研究の成果を説明する．W3CによるXQueryのUseCase[11]の1.1.9.4のQ4について考える．この問合せは図1(i)の(a)の部分である．この問合せは，"http://a.b/bib.xml"上で評価され，著者ごとに論文のタイトルをまとめ"result"エレメントを付加し，ルートを"results"エレメントとしている問合せである．もしXQueryがビュー定義機能を有しているならば，入力データを再構成しているこの問合せは仮想ビュー定義のための問合せの典型とみなすことができる．しかしながら，XQuery1.0では様々な理由から問合せによるビュー定義機能をサポートしていない．理由の一つは，XQueryには閉包性がないことにある．そこで我々は結果がXMLデータとなるXQueryだけを対象とすることにする．これ以外の詳しい理由と回避法については3節で述べる．図1(i)の点線で囲まれた問合せを仮想ビュー定義とみなすと，このビューに対する問合せとして「タイトルに"TCP/IP"を含む"author"の検索」を考える．伝統的な仮想ビューに対する問合せ同様，この問合せはビュー定義の問合せを展開した問合せとして処理される．図1(i)はこの展開された問合せを示している．ビュー定義とビューに対する問合せからこの展開された問合せをどのように構築するかは3節で説明する．

図1(i)に示した展開された問合せは，そのまま処理されると，ビュー定義が評価された後で，ビューに対する選択条件が評価され該当する結果が返る．ここで，我々が開発中の最適化手法によると，この問合せに対して二種類の問合せ最適化手法を適用する余地がある．一つは伝統的なデータベース問合せ最適化手法の一種であるpushing selectionである．pushing selectionにより選択条件をビュー定義の内側へ移動することで問合せ処理中の中間結果の数を減らすことで効率的に処理できる．もう一つは定数畳み込み[1]に基づく手法である．定数畳み込みを適用することで，問合せが要求していない結果に関する冗長な計算をさけることができるので，この問合せが要求している結果"author"エレメントは，ビュー定義による"result"エレメントの導入などの再構成をすることなく求めることができる．

しかしながら，このpushing selectionと定数畳み込みを実現するためには次の問題を解決する必要がある．pushing selectionに関しては，選択条件を早期に評価するためには，ビュー上の"result"エレメントに含まれる"title"エレメント中に出現する文字列"TCP/IP"という選択条件を，"http://a.b/bib.xml"上の選択条件に変換する必要がある．しかしながら，"result"エレメントはビュー定義によって導入された新たなエレメントであり，"http://a.b/bib.xml"上には存在しない．よって，ビュー上の"title"に相当する"http://a.b/bib.xml"上の"title"エレメントを特定する必要がある．また，定数畳み込みに関しても，ビュー定義における冗長な計算を避けるためには，問合せが結果として要求している仮想ビュー上の"author"エレメントに対応する"http://a.b/bib.xml"上の"author"エレメントを特定する必要がある．

本稿ではこれらの問題を解決するために**式型木**を導入する．この式型木はXQuery式に対して構築され，入力式の型とサブ式へのエッジを有による木である．この木を構築し辿ることで上記の問題は解決されることを示す．このような木が構築できるのは，XQueryが関数型言語であるため式の組み合わせを用いて問合せ式が構築されているからである．この木をどのように構築するかについては??節で記述する．

我々が開発中の問合せ最適化手法**FLWOR Arranging**は次のようなステップで，この式型木を用いてFLWOR式をアレンジする．図1(ii)にこのFLWOR Arrangingを適用した結果の問合せを示す．

i) 展開された問合せにおけるビューに対する選択条件と問合せ結果式をそれぞれ基底XML文書に対するものに変換する．図1(i)に示した展開された問合せのビューに対する選択条件;

```
where contains(string($t), "TCP/IP")
```

と 結果式;

```
return $r/author
```

は，ビュー定義に関する式型木を用いることで;

```
for $b in doc("http://a.b/bib.xml")/bib/book,
    $t in $b/title
where contains(string($t), "TCP/IP")
```

と

```
for $a in
   distinct-values(doc("bib.xml")//author)
return $a
```

にそれぞれ展開される．

ii) 選択条件の適切な再配置

(1) 展開された選択条件は(e)で囲まれた式に対するものであることから，(e)の式にマージする．

(2) (e)式は(b)で束縛した変数$aと$ba との自己等結合であることが検出できれば，問合せの等価性を保持するために変数$aを束縛している(b)の変数束縛部の基底データをこの選択条件で

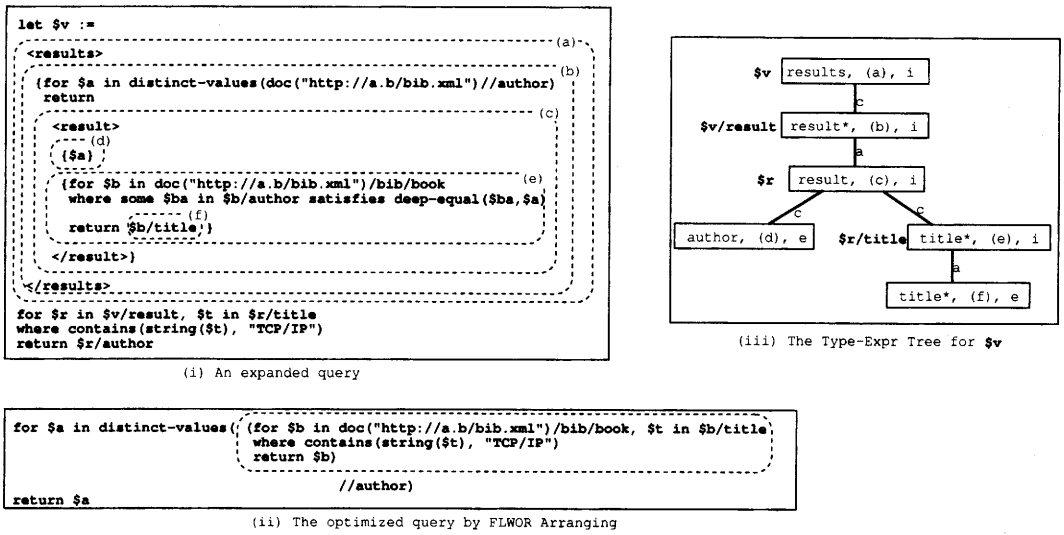

図 1 (i)an expanded query, (ii) the optimized query and (iii). the Type-Expr Tree for the view difinition.

絞り込んだものに置換する必要がある．これはXQueryの結合が外部結合であるからである．例えば，(e) 式にだけ選択条件をマージすると，ビューには" TCP/IP" の本を書いていない著者が束縛された$a に対して，(e) を評価した空値が出力されるからである．このとき変数$a を束縛しているエレメントauthor と選択条件で用いている変数束縛のエレメントbook の**蝶番ノード**を特定する必要がある．問合せ中の経路式から，book と類推できる．よって最も外側の for 変数束縛部において，基底 XML データdoc(''http://a.b/bib.xml'') は図 1(ii) の点線で囲まれた部分に置換される．

iii) return 式の再配置

上記ステップより return 式に該当するのが (d) 式であることがわかる．今 (d) 式の変数$a の束縛部は，選択条件を push した変数束縛部と同じであるので，(d) 式を return 式とすればよい．

1.2 Preliminary experiments

この書き換えられた問合せが入力問合せと比較してどの程度有効であるかを予備実験により検証してみた．この予備実験では人工データを用いて図 1(i) に示す入力問合せである展開された問合せと，これを最適化した図 1(ii) に示す問合せの実行時間を選択率 (図 2) とデータサイズ (図 3) とでそれぞれ比較してみた．XQuery エンジンとしては Galax[10] を用いた．但し，問合せの書き換えにかかるコストは考慮にいれていない点に注意されたい．この結果が示すように，問合せの書き換えコストが十分に低い限り，この書き換えは開発する価値があることが分かった．

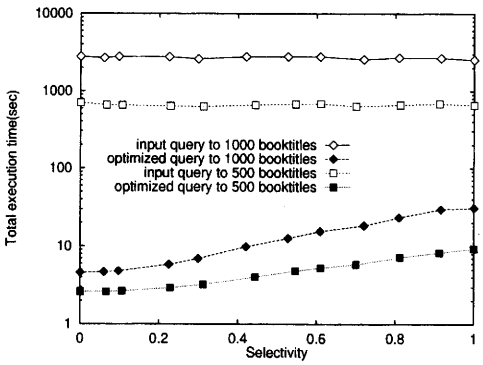

図 2 Execution time over selectivity

以下この論文の構成は次のとおりである．次の節では関連研究について述べる．Section ??では XQuery において中心的な役割を演じている FLWOR 式について説明する．また，XQuery による仮想ビュー定義についても述べる．4 節では，開発中の最適化手法の中心部分を構成する式型木について述べる．5 節はまとめと今後の課題である．

2. 関連研究

Paparizos ら [7] は，入力 XQuery を *TAX* と呼ばれる彼らが導入した代数に変換し，TAX 上で最適化し実行する

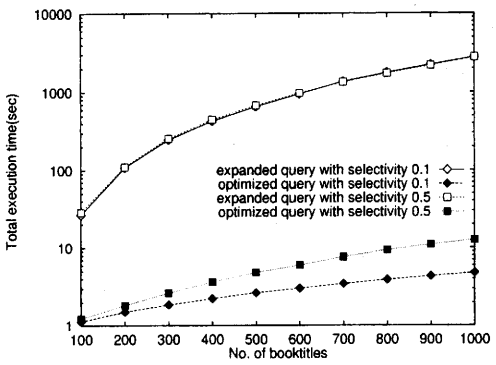

図 3 Execution time over data size

システムを，TIMBER と呼ばれる XML データベース [2] 上に開発した．XQuery の最適化に関する研究の多くは，このように XQuery を自分たちの代数に変換して自分たちの代数エンジンで実行するものである．これに対して，我々は XQuery を入力とし XQuery を出力する．これにより TIMBER のような最適化機構を有していない情報源に対しても最適化された問合せの実行が可能となる．

本稿で提案する最適化技術の一つは pushing selection に基づくものである．SQL ではマジックセット [5, 6, 9] が最も効果がある pushing selection の一つである．これにより多くの商用 DBMS ではマジックセットを採用している [8]．マジックセットは問合せ全体を書き換えて，マジック関係との結合による pushing selection の実現と，この結合の順番を決定するものである [5]．Predicate Move-Around[3] はマジックセットを補完するものであり，問合せ全体の構造を保ったまま選択条件の pulling と pushing をすることで最適な位置に述語を配置するものである．例えば，SQL における GROUPBY 節の外側の問合せブロックに述語 $Max(B) \geq c$ あるとする．但し，c は定数とする．このような場合，$Max(B)$ を計算するためには $B \geq c$ を満たすデータだけを用いれば良い．よって，Predicate Move-Around はこの述語を $B \geq c$ として GROUPBY 節の内側のブロックに移動する．

実際，前述したビューの定義は SQL における GROUPBY に基づいた問合せである．XQuery では明示的な GROUPBY 節をサポートしていない．SQL では GROUPBY という高次なコンストラクタを導入したことで，利用者にとって高度な問合せが記述しやすくなる一方で，GROUPBY 固有の最適化手法を開発する必要があった．これが関係マジックと Predicate Move-Around の果たした役割である．これに対して XQuery には明示的な GROUPBY 節が存在しないので，重複値を取り除く distinct-values() 関数と入れ子の FLWOR 式の組み合わせで記述する必要があるが，入れ子 FLWOR に対する uniform な最適化手法が適用できる．このように糖衣構文の導入にはトレードオフがある．

3. XQuery と仮想ビュー

XQuery は W3C で標準化中の問合せ言語であり、関数型言語に基づく言語である．この節では本稿で用いる XQuery の特性について述べる．また，現在の XQuery ではサポートされていない仮想ビュー定義機能について述べる．

3.1 FLWOR 式

XQuery において中心的な役割を果たしている FLWOR 式は，繰り返しと変数束縛のための For/Let 節部と，選択のための Where 節部，再構成のための OrderBy/Return 節部とから構成されている．XQuery の宣言性により，入れ子 FLWOR 式において外側と内側の両方から評価可能であるが，入力データの流れに関しては，入れ子 FLWOR 式に対する入力データは外側の FLWOR 式において処理されたデータが内側の FLWOR 式の入力となる．したがって，入れ子 FLWOR 式における pushing selection とは選択条件を外側の FLWOR 式に移動することである．

3.2 仮想ビューと展開された問合せ

XQuery1.0 では仮想ビューの定義はサポートされていない．その理由の一つに XQuery にエレメント同一性 (element identity) の問題がある [12]．この問題は具現ビューと仮想ビューの不一致を引き起こす．この問題を回避するためにビュー定義の XQuery 全体にエレメントコンストラクタを適用することにした．これにより，ビューに対する問合せと展開された問合せの等価性を保証できる．このコンストラクタの適用により，情報源のデータはコピーされるので，全てのノードがユニークなノード識別を持つ．

このような仮定の下，展開された問合せはビューに対する問合せから次のようにして構築できる．

- Let 節によりビュー定義の問合せ全体を新しい変数に束縛する．
- ビューに対する問合せにおけるビュー参照の出現を上記で導入した新しい変数参照に置換する．

図 1(i) に展開された問合せを示す．

3.3 式とそのゆるやかな型

XQuery は関数型言語であり，式の組み合わせから構成されている．本研究では以下の式に着目し，そのゆるやかな型を定義する．これらの式は XQuery の構文解析木の中間ノードに現れる．これ以外にも式はあるが，本稿ではこれらに制限して話を進めることにする．尚，各式のゆるやかな型を定義する際，スキーマは用いない．理由は WWW 上にある XML 文書の約半数がスキーマの使用を明示していない [4] ので，スキーマ情報から当該データの構造をあらかじめ知ることができない場合に対処するた

めである．スキーマを利用すればよりきつい型を定義できる．以下，式 $expr$ の型を $ltype(expr)$ で表わす．尚，本稿で用いる型とは，式を評価した際のエレメントの出現の順番をエレメント名の正規表現で記述したものである．逆に本稿で扱う式は，評価した場合にエレメントの正規表現となるような式に制限する．本稿で扱う型であるエレメント名の正規表現について，'*' 追加と削除に関して次のよう操作を定義する．

定義 3..1: 与えられた型 T について，

- $as(T) = T*$ 但し，$as(T*)* = T*$
- $rs(T*) = T$ 但し，$rs(T**) = T$

□

変数参照 (XQuery の構文木の中間ノード**VarRef** に相当) は式である．但し，本稿では変数に束縛されるオブジェクトは本稿で扱う式に限定することに注意されたい．変数参照の型は変数に束縛されている式の型に一致する．そこで変数に束縛されている式の型を特定するために，**変数参照はずし式 (variable dereferencing expression)** を定義する．

定義 3..2: 変数参照式$\$var$ に対して，その変数参照はずし式はその変数束縛部によって次のように定義される．

- Let 節で束縛されている場合 (**let** $\$var$:= $expr$)，$vb(\$var) = expr$
- For 節で束縛されている場合 (**for** $\$var$ **in** $expr$)，$vb(\$var) = each(expr)$

与えられた変数参照式$\$var$ に対して，$vb(\$var)$ は一意に決まり，その型 $ltype(vb(\$var))$ は

- $vb(\$var) = expr$ のとき，$ltype(expr)$ と一致する．
- $vb(\$var) = each(expr)$ のとき，$rs(ltype(expr))$ と一致する．

□

関数呼び出し (構文木の中間ノード**FunctionCall** に相当) は式であり，関数名と引数のリストから構成されており，引数のリストの要素は式である．以下本稿では関数呼び出し式を FunctionCall(関数名，引数) と記述する．関数呼び出しのゆるやかな型は関数名に依存するが，distinct-values() 関数のみその引数の型とし，それ以外の関数呼び出しは未定義とする．

FRWOR 式 (構文木の中間ノード**FLWORExpr** に相当) は式であり，その構成要素の一つである "return" の後につづく式 (構文木の中間ノード**ExprSingle** に相当) の型の列がそのゆるやかな型となる．定義より For 節で変数に束縛される値の数だけこの verb+ExprSingle+ 対応する式が評価されるからである．以下本稿では FLWOR 式は FLWOR(re) と記述することにする．但し，re は "return" の後につづく式を示している．よって $ltype(FLWOR(re)) = as(ltype(re))$ となる．

経路式 (構文木の中間ノード**RelativePathExpr** の特別な場合に相当) は式である．本稿ではこの経路式は "relative path expression" のみを扱い AxisStep に関しては AbbrevForwardStep のみを考える．また，AbbrevForwardStep に関しては NameTest のみを対象とする．NameTest に関しては QName のみを対象とする．経路式の型は経路式の / または // で区切られた最右辺のエレメント名の列型となる．

定義 3..3:
本稿で扱う経路式は，

QName (("/" | "//") **QName**)*

の形式に制限する．この経路式はまた，

ButLast (("/" | "//") **QName**

と表現することができるので，今後 RPE(ButLast,("/" | "//"), エレメント名) と記述する．□

$ltype$(RPE(ButLast,("/" | "//"), エレメント名))
$=as$(エレメント名) である．

エレメントコンストラクタ (構文木中の中間ノード **DirElemConstructor** もしくは**CompElemConstructor** に相当[1]) は式であり，新たに構築するエレメント名とその内容から構成されており，そのエレメント名がその型に相当する．今後エレメントコンストラクタは EC(en,c_1,...,c_n) と記述する．但し，en はエレメント名を c1,...,cn は式 (中間ノード ElementContent に相当) を示している．

括弧付けされた式 (中間ノード ParenthesizedExpr に相当) は式である．今後この括弧付けされた式は $(e_1, ..., e_n)$ と記述する．但し，e_i は式である．括弧付けされた式 $(e_1, ..., e_n)$ のゆるやかな型 $ltype(e_1, ..., e_n)$ は，$(ltype(e_1), ..., ltype(e_n))$ と一致する．

例 3..1: 図 1(i) の**$v** は展開された問合せを構築するために導入された新しい変数であり，$vd($**$v**$)$ は図 1(i) の (a) の式であり，EC("results",FLWOR(EC("result",$e_{(d)}$,$e_{(e)}$))) と記述できる．但し，$e_{(d)}$ と $e_{(e)}$ はそれぞれ図 1(i) の (d) と (e) で囲まれた式である．また，その型は "results"' である．□

4. 式型木

既に述べたように本研究の目的は，展開された問合せを入力とし，ビューに対する選択条件と要求する結果式の双方を，ビュー定義問合せブロックへ移動することによる，

[1] XQuery Core において**DirElemConstructor** は正規化されて**CompElemConstructor** に統合される．

選択条件の早期評価と定数畳み込みに基づく冗長な計算の削除による最適化された問合せへの書き換えにある．この選択条件と結果式の移動のためには，選択条件と結果式で参照されている　ビューに対する式に相当するビュー定義中の式を特定する必要がある．この節では，この特定のために用いられる**式型木**について記述する。まず式型木の構築法について述べ，その後で式型木を辿ることによるこの特定法について述べる．

4.1　ビュー定義に対する式型木の構築

XQuery は関数型言語であるので，問合せ式は式の組み合わせから成り立っている．3節で示した性質より，各式に対してその式を評価した場合のゆるやかな型を決めることができる．この節ではこの性質を利用した式型木の構築について記述する．

定義 4..1:　**式型木** et は式とそのゆるやかな型をノードとし，その式のサブ式へのノードをエッジとする木であり，与えられた式expr に対して，(I, T, E, N, A) の五つ組で表わす．但し，

- I はその式を評価した値が，外延であるか内包であるかを示す指示子であり，外延である場合"e" を，内包である場合"i" を値として持つ．式型木 et に対して I へのアクセスは $indicator(et)$ で表現する．
- T は与えられた式 $expr$ を評価したときのゆるやかな型 $ltype(expr)$ であり，式型木 et に対して T へのアクセスは $et.t$ で表現する．
- E は $expr$ であり，式型木 et に対して E へのアクセスは $et.e$ で表現する．
- N は式型木へのエッジのリストであり，式型木 et に対して N へのアクセスは $et.c[i]$ で i 番目のエッジへのアクセスを，$et.c$ でリスト全体へんおアクセスを表現する．リスト cl に対してその要素数は $|cl|$ で表現する．
- A は式型木へのエッジであり，式型木 et に対して A へのアクセスは $et.a$ で表現する．

□

定義 4..2:　式型木は，与えられた問合せ式 $expr$ に応じて次のように再帰的に構築される．但し，上記の五つ組に対して式型木のコンストラクタは，$C(I, T, E, N, A)$ とする．

式型木の構築 $et(expr)$

- 変数参照の場合，その変数参照はずし式に関する式型木を構築する．
- 関数呼び出しの場合，関数名が distinct-values() 関数の場合は，その引数に関する式型木を構築する．それ以外は，式型木を $C(\texttt{"e"}, \texttt{UNDEF}, expr, nil, nil)$ で構築する．
- FLWOR 式 (FLWOR(re)) の場合，$C(\texttt{"i"}, as(et(re).t), expr, nil, et(re))$ で構築する．
- 経路式 (RPE(bl,op,en)) の場合，$C(\texttt{"e"}, as(en), expr, nil, nil)$ で構築する．
- エレメントコンストラクタ (EC(name,c_1,...,c_n)) の場合，$C(\texttt{"i"}, name, expr, [et(c_1), ..., et(c_n)], nil)$ で構築する．
- 括弧付けされた式 (ParenthesizedExpr(e_1,...,e_n)) の場合，$C(\texttt{"i"}, (et(e_1).t, \cdots, et(e_n).t), expr, [et(e_1), \cdots, et(e_n)], nil)$ で構築する．

□

例 4..1:　図 1(iii) に図 1(i) におけるビュー定義の問合せ ($\$v$ に束縛されている式) の型式木を示す．□

性質 4..1:　式型木には以下のような性質があり，それぞれ式型木の構成に関する帰納法で証明できる．証明は自明なので本稿では省略する．

- 与えられた式に対してユニークである．
- 式型木からユニークな XQuery を生成できる．

4.2　式型木の辿り方

目標はビュー上の選択条件や検索結果式をビュー定義式中の該当するサブ式にマージすることである．今ビュー定義式の式型木が構築済みであると仮定すると，展開された問合せにおいて新たに導入した変数でビュー定義式を束縛している変数 (**$v** in this example) に該当する式型木は構築済みということである．前述した本稿で扱う XQuery の式のうち，FLWOR式とエレメントコンストラクタ式，括弧付けされた式は，式型木を成長させるのに対して，変数参照式，経路式はその対象が式型木である場合，木を辿り部分木もしくは，部分木を用いた新たな式型木を構築する．例えば，図 1(iii) に示す式型木に対して，経路式$\$v$/result に対応する式型木は $et(\$v).c[1]$ で辿った部分木となる．この節では，ビューに対する変数参照式と経路式によって，ビュー定義の式型木がどのように辿られ対応する部分木を構築するかについて記述する．この操作によりビューに対する問合せの選択条件と結果式に出現する式の，ビュー定義上の対応する式を特定することができる．尚，関数呼び出しに関しては関数の中身に依存するのでここでは扱わない．

定義 4..3:　与えられた式型木 et に対して，経路式 RPE(ButLast, op, en)，特に op が" /" である場合，の対応する et の部分木を構築する $find\text{-}child(et, en, \texttt{TRUE})$ を図 4に示す．□

定義 4..4:　与えられた式型木 et に対して，ある変数参照式の対応する変数参照はずし式が $each(expr)$ である

```
proc find-child(arg : ExprTypeTree, en : QName
                , flag : Boolean) result ExprTypeTree
{
  var cl : list of ExprTypeTree := [];
      t : ExprTypeTree;
  if arg.e is FLWORExpr{
    t := find-child(arg.a, en, flag);
    return C("i", as(en), expr(FLWOR(t.expr, arg)), nil, t);
  }
  elseif arg.e is ParenthesizedExpr
    for(i := 1 to number of child(arg))
      append find-child(arg.c[i], en, flag) to cl;
    if |cl| == 1
      return element of cl;
    elseif |cl| > 1
      return C("i", type(seq(n1.t, ..., nk.t))
               , expr(ParethesizedExpr(n1.e, ..., nk.e)), cl, nil);
    else /* not found */
      return nil;
  elseif arg.e is ElementConstructor
    if flag
      for ((i := 1 to number of child(arg))
        append find-child(arg.c[i], en, FALSE) to cl;
      if |cl| == 1
        return element of cl;
      elseif |cl| > 1
        return C("i", type(seq(n1.t, ..., nk.t))
                 , expr(ParethesizedExpr(n1.e, ..., nk.e)), cl, nil);
      else /* not found */
        return nil;
    else
      if arg.t == en
        return arg;
      else /* not found */
        return nil;
  elseif arg.e is RelativePathExpr
    if flag
      return C("e", as(en), expr(RPE(arg.e, "/", en), nil, nil);
    else
      if rs(arg.t) == en
        return arg;
      else /* not found */
        return nil;
  elseif arg.e is FunctionCall
    return nil;
  elseif arg.e is EachExpr
    return nil;
}
```

図 4 経路式による式型木の resolution

ような場合の，対応する et の部分木を構築する $resolve\text{-}each(et)$ を図 5に示す．□

例 4..2: 図 1(i) に示した展開された問合せにおけるビューに対する選択条件である `where contains(string($t),"TCP/IP")` 中の$t について，その変数参照はずし式 $vd(\$t) = each(each(vb(\$v)/result)/title)$ であり，図 1(iii) に示されている$v に関する式型木を上記で定義した手続きに基づいて辿ることで，式 $each(each(doc("bib.xml")/bib/book)/title)$ を得る．これを変数参照式に変換することで，図 1(ii) の点線で囲まれた式のうちの return 式を除く部分を得ることができる．□

最後に，これまで述べた式型木の構築と resolution を一つに統合したアルゴリズムを図6に示す．

```
proc resolve-each(arg : ExprTypeTree) result ExprTypeTree
{
  if arg.e is FLWORExpr
    resolve-each(arg.a);
  elseif arg.e is ParenthesizedExpr
    return nil;
  elseif arg.e is ElementConstructor
    return arg;
  else
    return nil; }
```

図 5 変数参照はずし式 $each(expr)$ の resolution

5. まとめと今後の課題

本稿では現在開発中の XQuery の書き換えによる最適化手法 FLWOR Arranging の概要を紹介し，その中心的な役割を果たす式型木の扱いについて述べた．今後の課題としては，この式型木の resolution の正しさを XQuery の形式的意味におけるジャッジメントを用いずに示すことである．そのためのラムダ計算に基づくモデルを定義中である．

参考文献

[1] D. F. Bacon, S. L. Graham, and O. J. Sharp. Compiler transformations for high-performance computing. *ACM Computing Surveys*, 26(4):345–420, December 1994.

[2] H. Jagadish, S. Al-Khalifa, A. Chapman, L. Lakshmanan, A. Nierman, S. Paparizos, J. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. TIMBER: A native xml database. *VLDB Journal*, 11(4):274–291, 2002.

[3] A. Y. Levy, I. S. Mumick, and Y. Sagiv. Query optimization by predicate move-around. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 96–107, Santiago, Chile, 1994.

[4] L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a First Study. In *Proceedings of the International World Wide Web Conference (WWW)*, 2003.

[5] I. S. Mumick, S. J. Finkelstein, H. Pirahesh, and R. Ramakrishnan. Magic is relevant. In *Proc. ACM SIGMOD Conf.*, page 247, Atlantic City, NJ, May 1990.

[6] I. S. Mumick and H. Pirahesh. Implementation of magic-sets in a relational database system. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 103–114, May 1994.

[7] S. Paparizos, S. Al-Khalifa, H. V. Jagadish, L. Lakshmanan, A. Nierman, D. Srivastava, and Y. Wu. Grouping in XML. In *Proceedings of the XML-Based Data Management Workshop(XMLDM2002, in conjunction with EDBT2002), LNCS, Vol.2490*, 2002.

[8] R. Ramakrishnan. *Database Management Systems*. McGraw-Hill, 1997.

[9] P. Seshadri, J. M. Hellerstein, H. Pirahesh, T. Y. C. Leung, R. Ramakrishnan, D. S. P. J. Stuckey, and S. Sudarshan. Cost-based optimization for magic: Algebra and implementation. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 435–446, June 1996.

[10] The Galax team. The Galax XQuery and XPath 2.0 interpretor, version 0.3.0. http://db.bell-labs.com/galax/.

[11] World Wide Web Consortium. XML Qery Use Cases. http://www.w3.org/TR/xmlquery-use-cases, November 2003.

[12] World Wide Web Consortium. XPath 2.0 and XQuery 1.0 isssues. http://www.w3.org/TR/2003/11/xpath-xquery-issues/, November 2003.

```
proc et(arg : VE) result ExprTypeTree
{
 if arg == FunctionCall(fn : QName, fa : VE)
  if fn =="distinct-values"
   return et(fa);
  else /* including doc()*/
   return C("e", UNDEF, arg, nil, nil);
 elseif arg == FLWORExpr(re : VE)
  return C("i", as(et(re).t), arg, nil, et(re));
 elseif arg == RPE(bl : RPE, op : OP, en : QName)
  if indicator(et(bl)) == "e" {
   return C("e", as(en), arg, nil, nil);
  else /* indicator(et(bl)) == "i" */
   if op == "/" {
    return find-child(et(bl), en, TRUE);
   else /* op == "//" */
    return find-descendant(et(bl), en);
   }
  }
 elseif arg == EC(name : QName, c1 : VE, ..., cn : VE)
  return C("i", name, arg, [et(c1), ..., et(cn)], nil);
 elseif arg == ParenthesizedExpr(e1 : VE, ··· , en : VE)
  return C("i", (et(e1).t, ··· , et(en).t), arg
                          , [et(e1), ··· , et(en)], nil);
 elseif arg == each(cont : VE)
  if indicator(et(cont)) == "e"
   return C("e", rs(et(cont).t), arg, nil, nil);
  else
   if resolve-each(et(cont)) == nil
    return C("i", rs(et(cont).t), arg, nil, nil);
   else
    return resolve-each(et(cont));
 elseif arg == VarRef
  return et(vd(arg));
 else /* other forms which do not deal in this paper*/
  return nil;
 }
}
```

図 6 式型木の構築と resolution を統合したアルゴリズム