例示操作に基づくXQuery問合せ学習システムの実装

松本　明†　森嶋　厚行††　中溝　昌佳††　北川　博之†††

† 筑波大学 システム情報工学研究科　〒305–8573 茨城県つくば市天王台 1–1–1
†† 芝浦工業大学 工学部情報工学科　〒330–8570 埼玉県さいたま市深作 307
††† 筑波大学 電子・情報工学系　〒305–8573 茨城県つくば市天王台 1–1–1
E-mail: †akey@kde.is.tsukuba.ac.jp, ††{amori,l99086}@sic.shibaura-it.ac.jp,
†††kitagawa@is.tsukuba.ac.jp

あらまし　既存の XML 問合せ言語や操作系では，一般に，ユーザはテキストエディタや視覚的操作系を用いて問合せを記述する．XLearner は全く異なるアプローチの XML 操作系であり，ユーザの例示操作に基づき XQuery 問合せを学習する．完全に規則的な構造を持つ RDB とは異なり，半構造データの一種である XML を対象とした問合せの学習は自明ではない．本稿では特に XLearner プロトタイプシステムの実装について説明する．

キーワード　XML，XQuery，学習システム，プロトタイプ実装

Implementation of a Prototype System to Learn XQuery Queries Based on Example Operations

Akira MATSUMOTO†, Atsuyuki MORISHIMA††, Akiyoshi NAKAMIZO††, and Hiroyuki KITAGAWA†††

† Doctoral Program in Sys. and Info. Eng., Univ. of Tsukuba,　Tsukuba, Ibaraki, 305–8573 Japan
†† Dept. of Info. Sci. and Eng., Shibaura Inst. of Tech.,　Saitama-City, Saitama, 330–8570 Japan
††† Institute of Info. Sci. and Elec., Univ. of Tsukuba,　Tsukuba, Ibaraki, 305–8573 Japan
E-mail: †akey@kde.is.tsukuba.ac.jp, ††{amori,l99086}@sic.shibaura-it.ac.jp,
†††kitagawa@is.tsukuba.ac.jp

Abstract　Existing XML query languages are textual or graphical languages in which we can specify queries for XML manipulation. XLearner, a different kind of manipulation framework for XML, learns XQuery queries based on operations of sample XML elements. Inferring queries for XML is a non-trivial task, because XML is a kind of semistructured data, in contrast to relational databases whose data structure is completely regular. The paper focuses on the implementation of a prototype system of XLearner.

Key words　XML, XQuery, learning systems, prototype implementation

1. はじめに

これまでに XML を対象とした問合せ言語や操作系が数多く提案されてきた [4]．これらでは一般に，ユーザはテキストエディタや視覚的操作系を用いて問合せを記述する．我々はこれらとは全く異なるアプローチの XML 操作系である XLearner の研究開発を行なっている．XLearner は，サンプルの XML 要素に対するユーザの例示操作から，XQuery 問合せを学習するシステムである．本稿では，XLearner プロトタイプシステムの実装について述べる．

XLearner プロトタイプシステムの開発にあたっては次の 2 点に留意した．第 1 に，ユーザと XLearner の間で必要なインタラクション数を減らす工夫を行なうことである．第 2 に，XQuery の学習処理におけるコストの削減を行なうことである．

本稿では，まず XLearner の概要を説明し，次にプロトタイプシステムの実装について記述する．

2. XLearner

まず XLearner の利用例を説明する．図 1(a) の DTD で表される XML データに対する問合せを学習させたいとする．このデータは，本屋における本のリストと分野情報を表している．`book` は本を表す．`category` は `book` が属する分野を表す．各 `book` がどの分野に属するかは，`book` の子要素である `category` 属性に，`category` 要素への参照を持つことで表す．この時，学

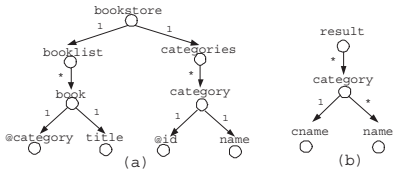

図 1　DTD

```
<result>{
for $c in /bookstore/categories/category
return <category>
         <cname>{$c/name}</cname>{
         for $b in /bookstore/booklist/book
         where $b/@category = $c/@id
         return <name>{$b/title}</name>
       }</category>
}</result>
```

図 2　XQuery 問合せ (document 関数と text 関数は省略)

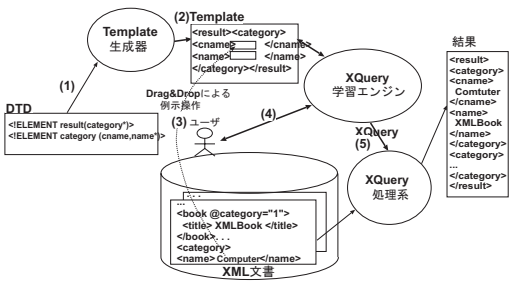

図 3　XLearner による問合せ学習の流れ

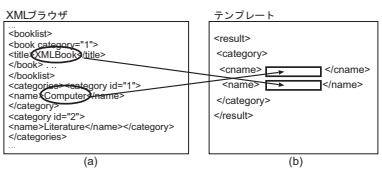

図 4　XML ブラウザとテンプレート

習させたい XQuery 問合せの例を図 2 に示す．これは，各分野毎に，分野の名前と本の名前を返す問合せである．

次に XLearner の使い方を説明する (図 3)．まず，結果として欲しい XML データの DTD(図 1(b)) をテンプレート生成器に入力する．次に，ユーザは操作対象となる XML 文書の中から，いくつか要素もしくは値（**XML ノード**と総称する）を選び，それらをテンプレートにドラッグアンドドロップ（以下 D&D) し，欲しい XML 問合せ結果の一部分を示す．テンプレートに D&D された XML ノードを**例示ノード**と呼ぶ．

与えられた XML 問合せ結果の一部分を手がかりに，学習エンジンは，ユーザが意図する問合せを学習するための質問をユーザに対して行う．このように，システムが能動的に質問を行ないながら学習を行なう枠組みを能動学習 [2] という．最後に，システムは XQuery 問合せを出力し，XQuery 処理系に渡す．

最初にあげた問合せ例の学習過程は次のようになる．まず，結果の DTD をテンプレートに入力すると，XLearner は図 4(b) のテンプレートを表示する．ユーザは XML ブラウザ中の XML の要素をいくつか選び，D&D し，図 4 のようにテンプレートを埋める．矢印は D&D 操作を表している．

```
N1:-     return <result>N1.1</result>
N1.1:-   for $c in /bookstore/categories/category
         return <category>N1.1.1 N1.1.2</category>
N1.1.1:- for $cn in $c/name return <cname>$cn</cname>
N1.1.2:- for $b in /bookstore/booklist/book, $n in $b/name
         where some $bc in $b/@category satisfies
               (some $ci in $c/@id satisfies $bc=$ci)
         return <name>$n</name>
```

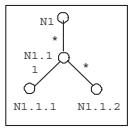

図 5　XQ-Tree 表現

次に XLearner は，ユーザの意図した XQuery 問合せを求めるために，ユーザに対して質問を行なう．これらは，各例示ノード e の **extent**(以下 EXT_e) を決定するための質問である．EXT_e は，例示ノード e が与えられた**文脈** (以下 $context_e$) において e が代表しているノードの集合である．ここで $context_e$ とは，EXT_e より前に extent が決定された例示ノードの集合である．例えば，“Computer”，“XMLBook” の順に extent が決まるとすると，$context_{\texttt{Computer}}$ は空であり，$context_{\texttt{XMLBook}}$ は {Computer} である．その結果，$EXT_{\texttt{Computer}}$ は全ての分野の名前の値の集合となり，$EXT_{\texttt{XMLBook}}$ は，その文脈における，`book` の集合 (すなわち，コンピュータ分野の本の集合) となる．

XLearner が行なう質問には，**Membership Query**(**MQ**) と **Equivalence Query**(**EQ**) がある．MQ では，XLearner がある XML ノードを選び，そのノードが EXT_e に含まれるかどうかをユーザに質問する．画面上では，対象ノードの上で「Yes」「No」のポップアップメニューが表示されるので，ユーザはそのノードが EXT_e に含まれる場合は「Yes」を，含まれない場合は「No」を選ぶ．EQ は，XLearner が生成した仮説 extent(以下，$E\hat{X}T_e$) に関して，$E\hat{X}T_e = EXT_e$ が成立するかを質問する．画面上で，$E\hat{X}T_e$ であるノード群がハイライト表示されるので，ユーザは，$E\hat{X}T_e = EXT_e$ ならば「OK」ボタンを，間違っている場合，反例を与える．反例には正の反例 ($\in EXT_e - E\hat{X}T_e$) と負の反例 ($\in E\hat{X}T_e - EXT_e$) がある．反例のノードの上で右クリックし，そのノードが正の反例ならば「Yes」を，負の反例ならば「No」を選ぶ．

3.　XQ-Tree とテンプレート

3.1　XQ-Tree

内部的には，XLearner は XQuery 問合せを直接学習するのではなく，XLearner における XQuery 問合せの内部表現である XQ-Tree の学習を行う．XQ-Tree の各ノード n は $\texttt{N}i\texttt{:-}q(n)$ の形式をしている．ここで，$\texttt{N}i$ はノード識別子であり，$q(n)$ は，問合せ断片（`flwr` 式 ([`for` $expr_f$[`where` $expr_w$]] `return` $expr_r$, ここで [...] は省略可）である．図 5 は，図 2 の問合せを表す XQ-Tree である．

XQuery 問合せ Q が与えられた時，それに対応する XQ-Tree t は次の手順で計算される：(1)`let` 式を除去し，変数をそれに割り当てられた式で置き換える．(2) 全ての XML ノードを返す式 (例えば，`$c/name`) を，等価な `flwr` 式（`for` $cn `in` `$c/name` `return` $cn）に書き換える．これにより入れ子の `flwr` 式ができる．(3) 以上で作成した `flwr` 式間の入れ子関係を木構造で表す．

t 中のあるノード n に対して，$cq_t(n)$ を n の**完全問合せ**と呼ぶ．$cq_t(n)$ は $q_t(n)$ と $\{q_t(m)|m \in depends_t(n)\}$ 中の問

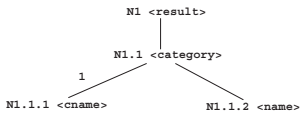

図 6　テンプレート

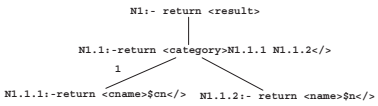

図 7　XQ-Tree スケルトン

合せ断片から構成される．ここで，$depends_t(n)$ は，t において n が依存する可能性があるノードの集合である．完全問合せ $q(n)$ と $\{q(m)|m \in depends_t(n)\}$ 中の問合せ断片に関数 $compose(q(m), q(m'))$ を繰り返し適用することで求められる．ここで，$compose(q(m), q(m'))$ は $q(m)$ 中で m' を参照するノード識別子を $q(m')$ で置き換えた問合せ断片を返す関数である．例えば，図 5 の XQ-Tree を t_1 とすると，cq_{t_1}(N1.1.1) = $compose(q$(N1), $compose(q$(N1.1), q(N1.1.1))) =“return (for $c in /bookstore/categories/category return (for $cn in $c/name return $cn))”=“for $c in /site/categories/category, $cn in $c/name return $cn” である．このように，通常 $cq_t(n)$ は “for $expr'_f, expr_f$ where $expr'_w \wedge expr_w$ return v” と記述することができる．ここで，$expr'_f$ と $expr'_w$ は $\{q(n)|n \in depends(n)\}$ 中に含まれる式である．

XQ-Tree 中において，問合せ断片は let 節を持たないため，変数 v は常に for 節または，some 式によって定義される．v が “for v in p” または “some v in p” で定義される時，“v in p” を $expr_t(v)$ で表す．

3.2　テンプレート

2 章で述べたように，XLearner は与えられた DTD に基づき，テンプレートを生成する．図 6 は図 1(b) の DTD から生成されるテンプレートである．

テンプレートは木構造をしており，ノードは DTD の各要素型に対して生成される．ある 2 つのノードについて，DTD 中の対応する要素タイプが親子関係を持つならば，その 2 ノードはその間に辺を持つ．例えば，要素定義が A=(B,C) の時，ノード n_A は子ノードとして n_B と n_C を持ち，n_A-n_B 間，n_A-n_C 間に辺を持つ．

もし，あるノードとその親ノードが 1 対 1 の関係ならば，それらをつないでいる辺は ‘1’ でラベル付けされる．このラベルは問合せの学習の際に利用する．

簡単のために以下の説明では，各ノードは ‘1’ でラベル付けされた辺でつながった子供は最大一つしか持たず，かつ，ルートから葉へのいかなるパスでも ‘1’ でラベル付けされた辺は連続しては存在しないものとする (図 6 のテンプレートはこの制約を満たす)．

4.　問合せ学習機構の概要

4.1　問合せの学習

問合せの学習は，2 つのステップから成る．

（ステップ 1）XQ-Tree スケルトンの作成: テンプレートと D&D で得られた例示ノードから，学習結果の XQ-Tree の基となる XQ-Tree スケルトンを生成する．図 7 は図 1(b) の DTD と図 4 の操作によって生成された XQ-Tree スケルトンである．各ノード n は，$q(n)$ = “return <$elem$>$expr$</>” の形式をしている．ここで，$elem$ はタグ名であり，$expr$ は子ノードへの参照を持つ．さらに，例示ノード e がドロップされた場所が XQ-Tree ノード n に対応する場所であったならば，$q(n)$ の $expr$ は e に対応する変数名 v_e を持つ（この例では$cn と$n）．

（ステップ 2）各問合せ断片の学習: XQ-Tree スケルトンを走査しながら，例示要素 e をドロップされた XQ-Tree ノード n_e のそれぞれに対して，各問合せ断片 $q(n_e)$ = for $expr_f$ where $expr_w$ return $expr_r$ の学習をしていく．実際に XLearner が学習可能なクラスでは，これらの形式は限定されている．詳細は 5 章で説明する．特に，$expr_r$ は XQ-Tree スケルトンで与えられた式そのものに限定されているので，実際に学習するのは，$expr_f$ と $expr_w$ である．XQ-Tree の走査にあたっては，n の前に必ず $depends(n)$ 中のノードの走査が行われるようにする．

4.2　context と extent

$q(n_e)$ の学習について述べる前に，2 章でふれた context と extent を定義する．

context:$context_e$ は，例示ノード e が与えられた時の文脈を表している．これは，既に問合せの学習が行われたノードにドロップされた例示ノードの集合として与えられる．例示ノード e の context を次のように定義する．まず，$assignment$ を (v, o) の組の集合とする．ここで，v は変数であり，o は XML ノードである．この時，$context_e$ は次のように記述できる．$\{(v_{e_i}, e_i)|n_{e_i} \in depends(n_e)\}$ ここで e_i はドロップされた例示ノードである．例えば，“Computer”，“XMLBook” の順に学習が行われるとすると，$context_{\text{Computer}} = \{\}$ である．$context_{\text{XMLBook}}$=$\{(v_{\text{Computer}}$,“Computer”)$\}$ であり，分野がコンピュータであるという文脈である．

extent:例示ノード e の extent EXT_e は，例示ノード e が与えられた文脈 ($context_e$) において e が代表しているノードの集合を表している．EXT_e の $context_e$ を，$context_e = \{(v_{e_1}, e_1), \ldots, (v_{e_m}, e_m)\}$ とすると，extent EXT_e は，$cq(n_e)$ と $context_e$ を用いて次のように定義される．EXT_e = {| for $expr'_f$, $expr_f$ where $expr'_w \wedge expr_w \wedge$ (v_{e_1} is e_1) $\wedge \ldots \wedge$ (v_{e_m} is e_m) return v_e|}．ここで {|$expr$|} は $expr$ の問合せ結果に含まれる XML ノードの集合である．例えば，$context_{\text{Computer}}$=$\{\}$ であるので，EXT_{Computer} は全ての分野の集合を表しており，$context_{\text{XMLBook}}$=$\{(v_{\text{Computer}}$,“Computer”)$\}$ であるので，EXT_{XMLBook} は分野がコンピュータであるという文脈における book 要素の集合（すなわち，コンピュータ分野に属する全ての本の名前）を表している．

4.3　学習の流れ

各問合せ断片 $q(n_e)$ の学習の基本的な流れは次のようになる．

(1)XLearner は EXT_e を学習するために MQ を発行する．そしてユーザからの答えと，$\{q(n)|n \in depends(n_e)\}$，$context_e$ に基づき，仮説 extent $E\hat{X}T_e$ を生成する．$E\hat{X}T_e$ は次のように表される．$E\hat{X}T_e = \{|$`for` $expr'_f, ex\hat{p}r_f$ `where` $expr'_w \wedge ex\hat{p}r_w \wedge (v_{e_1}$ `is` $e_1), \ldots, (v_{e_m}$ `is` $e_m)$ `return` $v_e|\}$．これは $ex\hat{p}r_f$ と $ex\hat{p}r_w$ がそれぞれ，$expr_f$ と $expr_w$ の仮説である以外は，EXT_e と同等である．(2)XLearner は EQ を発行し，$E\hat{X}T_e = EXT_e$ であるか質問する．(3) もし，$E\hat{X}T_e = EXT_e$ であれば終了する．このときの，学習結果は $q(n_e)$=“`for` $ex\hat{p}r_f$ `where` $ex\hat{p}r_w$ `return` v_e” となる．そうでない場合には (1) に戻る．ただし，$ex\hat{p}r_f$ と $ex\hat{p}r_w$ は，$E\hat{X}T_e \neq EXT_e$ を示す反例により修正される．

5. XQ-Tree の学習アルゴリズム

5.1 準　　備

$Expr^*_t(v)$ は v を計算するために必要なパス式の集合であり，次のように定義する．

（1） $expr_t(v) \in Expr^*_t(v)$.

（2） もし変数 w が存在し，“x `in` w/q”$\in Expr^*_t(v)$ ならば，$expr_t(w) \in Expr^*_t(v)$.

例えば，$Expr^*_{t_1}(\$cn) = \{$“`$c in /bookstore/categories/category/`”, “`$cn in $c/name`”$\}$ である．

$Expr^*_t(v)$ 中の全ての式を連結したものを $expr^*_t(v)$ で表す．例えば，$expr^*_{t_1}(\$cn)$=“`$cn in /bookstore/categories/category/name`” である．

$associated(v)$ を $Expr^*(v)$ 中に現れる変数の集合とする．また，`for` 節で v を定義するノードを n_v としたとき，$associatable(v)$ を n_v の祖先ノード（n_v 自身を含む）に現れる変数の集合とする．XQuery の変数のスコープから次の 2 点が保証される．(1)$associated(v) \subseteqq associatable(v)$ である．(2) もし，$v' \notin associatable(v)$ ならば，$q(n_v)$ は v' と v の関係を特定できない．

5.2 学習可能なクラス

まず 1-$Learnable(n_e)$ を定義する．直感的には，1-$Learnable(n_e)$ は $q(n_e)$ が，$\{q(n)|n \in depends(n_e)\}$ と $context_e$ が与えられたときに学習可能なある特定の形式を `where` 節の条件に持つ時に成立する．具体的には，1-$Learnbale(n)$ は，$q(n)$ = “`for` v `in` p `where` $expr_w$ `return` v” が次の条件を満たすときに成立する．

(1) $expr^*(v)$ のパス式は `document` 関数から始まるパス正規表現である．

(2) $expr_w$ は $\bigwedge_{v' \in (associatable(v) - associated(v))} RS(\{v, v'\})$ の形式である．

ここで $RS(\{v_1, v_2\})$ は $True$ もしくは $RS'(\{v_1, v_2\})$ である．$RS'(\{v_1, v_2\})$ は v_1 と v_2 の関係を表す次の式のいずれかである．

(Rel1) $data(v_1) = data(v_2)$

(Rel2) `some` w `in` v_1/cp `satisfies` $RS'(\{w, v_2\})$

(Rel3) `some` w `in document()`$/cp$ `satisfies` $RS'(\{v_1, w\}) \wedge RS'(\{w, v_2\})$

ここで cp は child 軸だけからなるパス式 (a/b/c など) である．

```
1. XQTree learnXQTree(XQTree t) {
2.   for each node n in t { // C^1(n)のノードを優先した
3.                          // 重み付き深さ優先探索
4.       if (n.VarInReturn()) {
5.           n.query = learnQuery1(n);
6.       } else {
7.           if there exists C^1(n) {
8.              collapse(n, C^1(n));
9.              n.query = learnQuery1(n);
10.          } else do nothing;
11.      }
12.  }
13.}
```

図 8　XQT1*+の学習アルゴリズム

この定義は `where` 節の条件が n で定義された変数 v と $v' \in (associatable(n) - associated(n))$ との結合の関係でなければならないことを示している．2 つの変数は間接的な関係を持つことも可能である．(Rel2) と (Rel3) は w を用いた間接的な関係のパターンである．ここで w を $RS'(\{v_1, v_2\})$ のリレーノードと呼ぶ．図 13 は 2 つのリレーノードを用いた$b と$c の間の間接的な関連を表している．“`some $bc in $b/@category satisfies (some $ci in $c/@id satisfies ($bc=$ci))`” である．

次に 1-$Learnable'(n_e)$ を定義する．まず，n が ‘1’ でラベル付けされた辺でつながった子ノードを持つとき $C^1(n)$ と表現する．また，$collapse(n, n')$ はノードを表し，その問合せ断片 $q(collapse(n, n'))$ は関数 $compose(q(n), q(n'))$ によって得られる問合せ断片となる．すると 1-$Learnable'(n_e)$ は次の 2 つの条件を満たすときに成立する．

(A1) $C^1(n_e)$ exists
$\Rightarrow$ 1-$Learnable(collapse(n_e, C^1(n_e)))$

(A2) $C^1(n_e)$ does not exist
$\Rightarrow q(n_e)$ =`return` “n の子ノードのノード識別子”

XQ-Tree t 中の全てのノード n が 1-$Learnable(n)$ もしくは 1-$Learnable'(n)$ を満たす XQ-Tree をクラス XQT1*+と呼ぶことにする．

5.3 XQT1 +の学習アルゴリズム

XQT1*+に属する問合せの学習アルゴリズムを図 8 に示す．図中の `learnQuery1(n)` は XQT1*+の学習アルゴリズムを実装している関数である．このアルゴリズムは，$C^1(n)$ のノードを優先した重み付き深さ優先探索で XQ-Tree スケルトンの走査を行いながら，`learnQuery1(n)` を用いて各問合せ断片を学習する．

5.3.1 learnQuery1(n)

図 8 中の $learnQuery1(n)$(5 行目) は 1-$Learnable(n)$ が成立するときの $q(n)$ を学習する関数である．説明に入る前に，XLearner における問合せ断片の標準形について説明する．次のように求めることができる．まず各 $expr_f$ を $expr^*(v)$ で置き換える．$expr^*(v)$ は，1-$Learnable(n)$ の定義によって document のルートから始まることが保証されている．次に，変数間の**パス共有条件**を指定するために，$expr^*(v)$ を拡張した表現を用いる．具体的な例を挙げると，$q_{t_1}(N1.1.1)$ は次のように記述される．`for $cn in /bookstore{$i1}/categories{$i2}/category{$i3}/name where {$i3 is $c} return $cn`．ここで “`/bookstore{$i1}/categories{$i2}/category{$i3}`

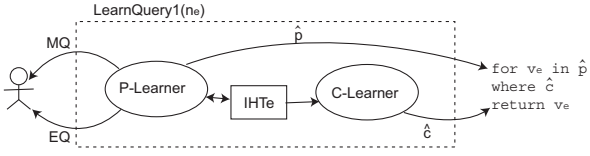

図 9　$q(n_e)$ の学習のデータの流れ

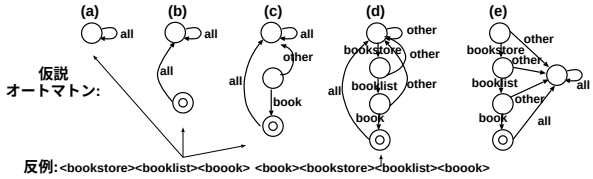

図 10　オートマトン導出過程

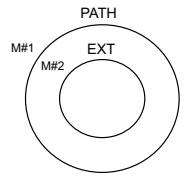

図 11　$PATH_e$ と EXT_e のミスマッチ

/name" は, "$i1 in /bookstore, $i2 in $i1/categories, $i3 in $i2/category,$cn in $i3/name" の省略形である. この表現により, $cn は document のルートから始まる $expr^*(v)$ で定義され, 同じ category 要素を共有しているという$cn と $c の間のパス共有条件は条件$i3 is $c で指定される. 一方, 図 5 の本来の記述法では, パス共有条件は $i で始まるバインディング式のパス表現 "$cn in $c/name" によって指定されている. これにより, $q(n_e)$ は "for v_e in p where c return v_e" の形式であり, p は document のルートから始まるパス表現であると考えることができる.

我々は, where 節の条件 c を述語の集合とし, c を c に含まれる述語の連言と見なす. もし, $c \supseteq c'$ ならば, {|for v in p where c return v|} は {|for v in p where c' return v|} に含まれるという意味で, c は c' より**厳しい**と呼ぶことにする.

learnQuery1(n) の概要:$learnQuery1(n)$ では, $q(n_e)$ は where 節に条件 c を持ち $depends(n_e) \neq \phi$ である. $EXT_e = \{|\text{for } expr'_f, v_e \text{ in } p \text{ where } expr'_r \wedge c \wedge (v_{e_1} \text{ is } e_1), \ldots, (v_{e_m} \text{ is } e_m) \text{ return } v_e|\}$ と表すことができる.

EXT_e が与えられたとき, $PATH_e$ を {|for v_e in p return v_e|} とする. XQT1 では, $PATH_e$ と EXT_e の間にミスマッチが生じる (図 11(a)). EXT_e と $PATH_e$ の間には $EXT_e \subseteq PATH_e$ が常に成り立つ. $q(n_e)$ を学習するためには $PATH_e$ と EXT_e の両方が必要である.

図 9 は $learnQuery1(n_e)$ と関連するモジュールとのデータの流れを示している. これにより $q(n_e) =$ for v_e in $\hat{p}$ where $\hat{c}$ return v_e を計算する. P-Learner は $\hat{p}$ を計算するための学習アルゴリズムを実装している. また, ユーザとのやりとりを IHT_e(Interaction Histry Table) に記録する. IHT_e は EXT_e を学習するために用いる. C-Learner は $\hat{c}$ を計算する.

5.3.2　P-Learner

XML 要素のパスを文字列と見なすと, パス正規表現を学習することは, 本質的にはパスの集合が表す言語を受理する有限オートマトンを求めることに対応する. ある言語が与えられた時, それを受理する最小の有限オートマトンを見つけることは NP 完全 [5] である. 一方, 能動学習を利用した Angluin のオートマトン導出アルゴリズム [1] を利用すれば, 多項式時間で発見することが可能である. XLearner では, Angluin のオートマトン導出アルゴリズムを利用して EXT_e に関する MQ と EQ からパス式の学習を行う [6].

図 10 は/bookstore/booklist/book に対応するオートマトンの導出過程を表している.

5.3.3　Interaction Histry Table

$IHT_e(Node, Ans, P, C)$ は, XLearner の質問とそれに対するユーザの答えを記録するリレーションである. IHT_e 中の各タプル t に XLearner の質問に対するユーザの答え一つが記録されている. 属性 $Node$ には XML ノードが格納されている. 属性 Ans には XLearner の質問に対する答えが格納されている. 例えば, XML ノード m が MQ として質問されたとき, その答えが Y であるとする. このとき, $t.Node = m$ であり, $t.Ans = Y$ である. IHT_e には答えに対する理由も記録する. もし, パス表現 p が $t.Node$ を受理しないという理由で $t.Ans = N$ ならば, $t.P = N$ となる. もし, $t.Node$ が条件 c を満たさないという理由で $t.Ans = N$ ならば, $t.C = N$ となる. もし, $t.Ans = Y$ であれば, $t.P$ と $t.C$ のどちらも Y である. IHT_e はドロップされた例示ノード e が正の例であるという記録 (e, Y, Y, Y) を常に含んでいる.

5.3.4　Extent の学習におけるミスマッチの問題

あるタグの並びを s とすると, P-Learner は $tags(path(m)) = s$ であるような XML ノード m を選び, ユーザに $m \in PATH_e$ であるか質問する. ここで $path(e)$ は XML インスタンスのルートから e までの XML ノードの並びであり, $tags(e_p)$ は e_p にマッチするタグの並びである. P-Learner は $m \in EXT_e$ であるかを質問する MQ を発行していないが, ユーザは $m \in EXT_e$ かどうかを質問されていると考えている. そのため, ユーザの答えと P-Learner が p を学習するために必要としている答えにはミスマッチが生じる図 11(a). ユーザの答えが Y の時には, $m \in EXT_e \Rightarrow m \in PATH_e$ であるため問題ない. しかし, もしユーザの答えが N であった場合には次の 2 つの可能性がある.

(Case M#1)　$m \notin PATH_e$ である.

(Case M#2)　$m \in PATH_e$ であるが, $c \wedge (v_{e_1} \text{ is } e_1) \wedge \ldots \wedge (v_{e_m} \text{ is } e_m)$ が間違っている.

ユーザの答えとして N が与えられたとき, XLearner は 2 つの Case のうちどちらであるのか区別することができない. そこで, アルゴリズムはまず最初に Case M#1 であると仮定する. よって, 答えが N の時にはタプル $t = (m, N, N, null)$ が IHT_e に挿入される. ここで $t.C = null$ は, c に対しては何も与えないことを意味する. 仮定が間違っていることを発見した時にはそのタプルを訂正し, その場所からやり直す [6].

```
A = {1, ..., k}
while (true) {
    Let ĉ = ∧_{i∈A} x_i.
    Make an EQ with ĉ.
    If the answer of the EQ is "OK" (i.e., ĉ = c) {
        halt
    } else {
        let y_1, ..., y_k be the counterexample.
        A = A ∩ {i | y_i = true}
    }
}
```

図 12　k 変数単調単項式を EQ により学習するアルゴリズム

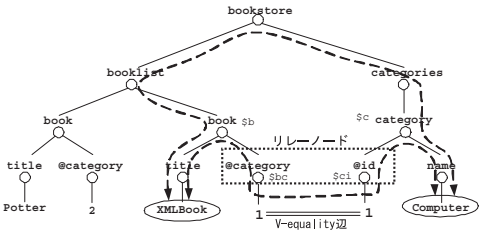

図 13　XML グラフ

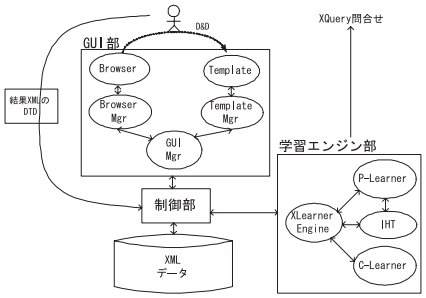

図 14　プロトタイプシステムのアーキテクチャ

5.3.5　C-Learner

C-Learner はそれまでに与えられた例示ノードの集合に無矛盾な最も厳しい条件 $\hat{c}$ を出力する．条件 $\hat{c}$ は述語の連言であるので，$\hat{c}$ を k 変数の単調単項式 $x_1 \wedge \ldots \wedge x_k$ と見なすことができる．ここで $\hat{c}$ 中の各述語は単項式中の各変数に対応する．k 変数の単調単項式の学習を，最大 k 回の EQ で学習するアルゴリズムが存在する．図 12 に k 変数の単調単項式を EQ により学習するアルゴリズムを示す.

このアルゴリズムの各変数に述語を対応させることにより，条件の学習を行う．assignment a_0 を $context_e \cup \{(v_e, e)\}$ とする．C-Learner は，ドロップされた例示ノード e と $context_e$ の間に成立する全ての述語の集合を計算することによって述語の候補の集合を計算する．1-$Learnable(n)$ に含まれる述語は全て候補として導出される．この述語の集合 ($cond(context_e, (v_e, e))$ で表す) を最初の推測とする．これは C-Learner が出力できる最も厳しい条件となる．通常，$cond(context_e, (v_e, e))$ は a_0 においてのみ成立する不必要な述語を含んでいる．そこで C-Learner は反例を用いてそのような述語を除去する．

C-Learner は $\hat{c}$ を計算するために $context_e$ と IHT_e を用いている．各 assignment $a_i \in \{context_e \cup \{(v_e, t.Node)\} | t \in \sigma_{Ans=Y}(IHT_e)\}$ は EXT_e に対する正の例示ノードが受理される状態を表す．我々は反例として各状態で成立する述語の集合を用いる．形式的には，各 $t \in \sigma_{Ans=Y}(IHT_e)$ に対して，$cond(context_e, (v_e, t.Node))$ が反例となる．この時，アルゴリズムは正の例が与えられた全ての状態で無矛盾な最も厳しい条件を出力する．

$cond(context_e, (v_e, e_i))$ **の計算**: $cond(context_e, (v_e, e_i))$ の計算は XML グラフ (図 13) を用いて行う．XML グラフは，XQuery and XPath data model [11] 等と同様に，XML インスタンスの構造を木構造で表したものである．加えて，我々の XML グラフでは，v-equality 辺を持つ．これは，同じ値を持つ XML ノード間に存在する辺である．

このアルゴリズムは 3 つのステップから成る．(1)XML において，$e_i \cup context_e$ に含まれる任意の 2 ノード間のパスを全て求める．図 13 は，e_i=XMLBook，$context_{\texttt{XMLBook}}$={Computer} の例を表している．(2) 各パスにおけるリレーノードのインスタンスを発見する．パス p 上のリレーノードのインスタンスとは，p 上において他のノードと v-equality 辺でつながっているノードである．(3) 各パスに関して成立する全ての述語を数え上げる．

6.　プロトタイプシステムの実装

本プロトタイプシステムの実装には Java(Java2 SDK 1.4) を用いた．GUI 部の実装には Swing および Drag and Drop API を利用した．システム構成図を図 14 に示す．プロトタイプシステムは制御部を中心に GUI 部と学習エンジン部によって構成される．XLearner は，能動学習を行なうため，学習エンジンが能動的に動く必要がある．したがって，GUI 部と学習エンジン部はそれぞれ別スレッドで動作する．

現段階のシステムには次の制約がある．まず，XML ブラウザからテンプレートに D&D 可能なものは，末端の XML ノードに限定される．また，再帰的定義を持つ DTD は，結果の XML が従う DTD として受け付けることができない．

GUI 部: `Browser` は，XML ブラウザを実装したモジュールである．操作対象の XML を表示する役割や，MQ，EQ の質問をユーザに示し，質問に対するユーザの答えを受け取る役割を果たす．`Template` は，テンプレートを実装したモジュールである．テンプレートを表示し，ユーザの D&D 操作によって例示ノードを受け取る．`BrowserMgr` は XML ブラウザを管理する．`Browser` に対して，MQ，EQ の表示の指示を行い，答えを `GUIMgr` に渡す．`TemplateMgr` はテンプレートを管理する．問合せ結果が従うべき DTD からテンプレートを生成し `Tempalte` に表示させたり，D&D された例示ノードの情報を `GUIMgr` に渡したりする．`GUIMgr` は GUI 全体を管理する．`TemplateMgr` や `BrowserMgr` の生成を行い，制御部とのデータのやりとりを行う．また，D&D された例示ノードとテンプレートに基づき，XQ-Tree スケルトンを作成する役割を果たす.

学習エンジン部: 学習エンジン部の役割は，`GUIMgr` より作成された XQ-Tree スケルトンを走査しながら，各ノードの問合せ断片を学習することである．学習エンジン部は大きくわけて，`XLearnerEngine`，`P-Learner`，`C-Learner`，`IHT` の 4 つで構成される．`XLearnerEngine` は，図 8 の XQ-Tree スケルトンを

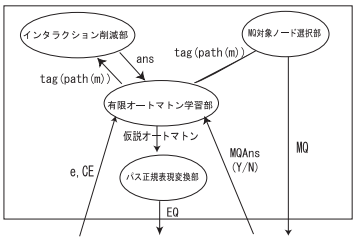

図 15　P-Learner の詳細

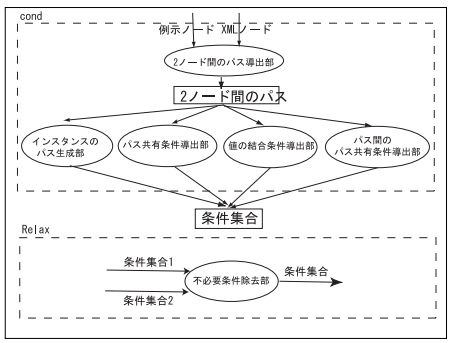

図 16　C-Learner の詳細

走査するアルゴリズムを実装している．各ノードの問合せ断片の学習は，P-Learner と C-Learner が行う．

P-Learner は，Angluin のオートマトン導出アルゴリズムを実装したモジュールであり，for 節のパス式 p を学習する．図 15 に P-Learner の詳細について示す．P-Learner は有限オートマトン学習部，インタラクション削減部，MQ 対象ノード選択部，パス正規表現変換部から構成されている．まず，XLearnerEngine から例示ノード e が有限オートマトン学習部に渡される．有限オートマトン学習部では e を基に仮説オートマトンを学習する．その際に MQ を生成する．詳細は後で説明するが，ユーザに質問せずに棄却できる MQ が存在する．質問すべきかどうかの判断はインタラクション削減部で行う．ユーザに質問すべき MQ はユーザに質問するのだが，MQ の対象となる XML ノードは一般に複数あるため，MQ 対象ノード選択部において，どの XML ノードを MQ として質問するのが効率的か決定する．そしてその XML ノードを MQ としてユーザに質問する．生成した仮説オートマトンはパス正規表現変換部に渡され，パス式 p を EQ として XLearnerEngine に返す．反例 ce が与えられた際には，有限オートマトン学習部はその ce を基に再びオートマトンの学習を行う．

C-Learner は where 節の条件 c の学習を担当する．これは XML グラフにおける例示ノード間のパスで成立する述語を数え上げることにより行われる．C-Learner は例示ノード e と，その仮説パス式 $\hat{p}$ と，$context_e$ と IHT_e を受け取り，5.3.5 節で述べた cond 関数を用いて，各 assignment a_i の条件集合を求め，それらに全て成立している条件を出力する．C-Learner の詳細を図 16 に示す．

cond 関数はまず最初に，2 ノード間のパス導出部において，XML グラフを用いて 2 ノード間のパスを全て求める．次に，各パスに対して，リレーノードのインスタンスを求め，変数名をつける．そして，リレーノードのインスタンスのパス式，リレーノードのインスタンスと例示ノードの各ノードに成立するパス共有条件，同じ値を持つノード間の値の結合条件を条件集合として全て導出する．各パス間のパス共有条件も求める．これらはそれぞれ，インスタンスのパス生成部，パス共有条件導出部，値の結合条件導出部，パス間のパス共有条件導出部で行う．

各 cond 関数で導出された条件集合のうち，不必要な条件は Relax 関数を用いて除去される．Relax 関数は 2 種類の条件集合を与えると，両方に成立する条件のみを返す関数である．その結果として得られた条件集合が $\hat{c}$ となる．

XLearnerEngine から渡された仮説パス式 $\hat{p}$ と，学習した $\hat{c}$ より eq' を生成する．生成された eq' は実際に XQuery 処理系で実行させることで，その結果に含まれる XML ノードの集合を $E\hat{X}T$ としている．XLearner では XQuery 処理系として IPSI-XQ [9] を用いている．$E\hat{X}T$ は XLearnerEngine に渡し，Browser でハイライト表示することでユーザに示す．

$E\hat{X}T$ 中に含まれる XML ノードの集合をハイライト表示する際，XLearner はどの XML ノードをハイライト表示すべきなのか一意に決定できない．理由は，XQuery では全てのノードにデフォルトで付く ID という概念がないからである．そこで，XML ノードに対して一意に ID を付加する必要がある．本プロトタイプシステムでは内部的に各 XML ノードに対して ID を明示的に付加しておくことで，XML ノードを一意に決定し，ハイライト表示を可能にしている．

IHT は Interaction Histry Table を実装したモジュールであり，XLearner の質問とそれに対するユーザの答えの情報を記録する．

6.1　処理の流れ

処理の流れは次のようになる．システムを立ち上げると，Browser が操作対象となる XML データを表示する．ユーザが，問合せ結果が従うべき DTD をシステムに与えると，TemplateMgr がテンプレートを作成し，Template に表示させる．この状態で，システムはユーザの D&D 操作を待つ．ユーザは Browser に表示されている XML ノードを Template に D&D し，Template を埋める．D&D 操作が完了したら．GUIMgr が XQ-Tree スケルトンを作成する．XLearnerEngine は XQ-Tree スケルトンを受け取り，学習を始める．各問合せ断片の学習で行なわれるユーザとのインタラクション (MQ や EQ) の結果は，P-Learner および C-Learner，IHT に渡される．システムが MQ や EQ を行なう際には，学習エンジンが Browser にその質問を表示させるよう指示する．

図 17 はプロトタイプシステムの実行画面を示している．左側が XML ブラウザ，右側がテンプレートである．

6.2　実装における問題と我々のアプローチ

XLearner のナイーブな実装には，次のような問題点がある．

(1) 学習に必要なインタラクション数の問題: n, k, m をそれぞれ，学習する有限オートマトンの状態数，アルファベットの数 (すなわちタグの種類の数)，反例の最大のサイズ (XML インスタンスにおけるルートノードから XML ノードへの長さ) とす

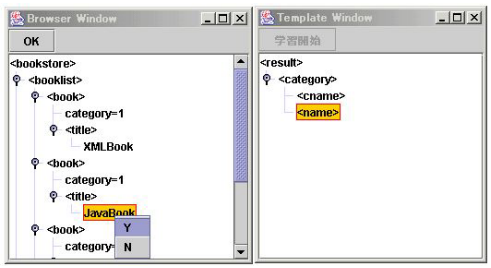

図 17　プロトタイプシステムの実行画面

る．このとき，インタラクションの数は $O(kmn^2)$ である．すなわち，簡単な有限オートマトンの学習でさえ，数百回のインタラクションが必要な可能性がある．

XLearner は，次に示す 2 つのルールで MQ を削減する [6]．

R1. DTD に矛盾するタグの並びの MQ に対し自動的に「No」を与える．

R2. 例示ノード e が与えられたときに e のパス $path(e)$ の末端のタグが t1 だとする．この時，タグの並びの末端が $t1$ でない MQ に対し自動的に「No」を与える．これは，パターンマッチングにおいて，末端のタグが重要であるというヒューリスティクスに基づいている．

さらに本プロトタイプシステムでは，次のルールを組み込む．

R3. MQ 対象となる XML ノードの候補が複数ある場合は，それらのうち，XML グラフ上で例示ノード e に最も近いノードを優先して MQ を行なう．これは，e に近いノードは EXT_e に含まれる (すなわち「Yes」が返される) 可能性が高いというヒューリスティクスに基づく．XLearner の学習アルゴリズムは，MQ に対して「Yes」が返された時はバックトラックを起こさないという性質がある [6] ため，インタラクション数を減らす効果がある．

これらのルールにより，例えば，例示ノード “Computer” のパス式を求める際に実際には 309 回質問される MQ が，一度も聞かれることなく求めることができる．

(2)$expr_w$ の学習コストの問題: 2 つの問題がある．第 1 に，全ての v-equality 辺を保持しておくには多くの余分な記憶容量が必要となる．第 2 に，例示ノード間の全てのパスを数え上げることは探索コストが高い．本プロトタイプシステムはヒューリスティックスと既存技術を応用し，これらに対処する．

v-equality 辺の問題. v-equality 辺は問合せに含まれる結合条件を導出するために利用されるが，実際に結合条件に用いられる値の組合せは限られている．例えば，Xmark [8] と XML Use Case [10] では，27 の等結合のうち 3 つは明示的な idref-id 参照である．残りは要素の値を用いているが，これらはすべて外部キーの役割を果たす値である．このような値に対してはインデックスが存在することが一般的であると考えられる．本プロトタイプシステムではこのようなインデックスの存在を仮定し，これを利用して (論理的な)XML グラフの探索を行なう．特別に v-equality 辺を表すためのデータを持つことはしない．

パスの計算コスト. 一般的な XQuery 問合せにおいては，結合に利用する XML ノード間の (XML グラフにおける) 距離はそれほど長くない．例えば， Xmark [8] と XML Use Cases [10] における最大の長さは 12 であるため，XML ノード間のパスを計算するためには，それぞれの XML ノードから 6 個ずつ探索すればよい．それに加え，Graph Schemas [3] のようなグラフ構造のメタデータを利用し，探索空間をあらかじめ制限する．

(3)EQ 処理を行なう際のパフォーマンスの問題. XLearner は EQ を質問するために，仮説 extent $E\hat{X}T$ を計算する必要がある．そのために処理時間が長くなってしまうという問題がある．しかし，`Browser` 中に一度に表示できる XML データの範囲は限定されるので，$E\hat{X}T$ の計算を XML データ全体に対して行うのではなく，`Browser` で見えている範囲のデータに対してのみ行なえば，処理時間の短縮が期待できる．これは, [7] と同様の手法である．ただし今回のプロトタイプシステムでは実装は行わず，XML データ全体に対して行っている．

7. ま と め

本稿では，XQuery 問合せの学習システム XLearner の概要および，プロトタイプシステムの実装について述べた．今後の課題としては，ユーザビリティーの検証が挙げられる．

謝　　辞

本研究の一部は，日本学術振興会科学研究費基盤研究 (B)(12480067) による．

文　　献

[1] D. Angluin. Learning regular sets from queries and counterexamples. Information and Conputation, 75(2):87-106, 1987

[2] D. Angluin. Computational Learning Theory: Survey and Selected Bibliography. *Proc. 24th Annual ACM Symposium on Theory of Computing*, pp. 351-369, 1992.

[3] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. *Proc. ICDT*, pp.336-350, 1997.

[4] M. Fernandez, J. Siméon and P. Wadler. XML Query Languages: Experiences and Exemplars. http://www.w3.org/1999/09/ql/docs/xquery.html.

[5] E.M. Gold. Complexity of automaton identification from given data. Information and Control 37:302-320, 1978

[6] A. Morishima, A. Matsumoto, H. Kitagawa. XLearner: A System to Learn XML Queries through Examples (submitted).

[7] A. Morishima, T. Mouri, H. Kitagawa. An Exampled-Based Web-Site Cnstruction Tool anad Its Implementation. *Information Systems and Databases*, pp.136-141, 2002

[8] A. Schmidt, F. Waas, M. Kersten, D. Florescu, M. Carey, I. Manolescu, and R. Busse. Why And How To Benchmark XML Databases. *SIGMOD Record*, 30(3):27–32, 2001.

[9] IPSI-XQ - The XQuery Demonstrator http://ipsi.fhg.de/oasys/projects/ipsi-xq/download_e.html

[10] XML Query Use Cases. http://www.w3.org/TR/xmlquery-use-cases.

[11] W3C. XQuery 1.0 and XPath 2.0 Data Model. http://www.w3.org/TR/query-datamodel/.