D-022

制約論理プログラミングに基づくXMLデータベースの試作
An Implementation of XML Database based on Constraint Logic Programming

駅 昌弥† 大囿 忠親† 新谷 虎松†
Masaya Eki, Tadachika Ozono, Toramatsu Shintani

1 まえがき

XMLは，その柔軟性や豊かな表情力およびプラットフォーム中立という特徴から，データ交換の標準に役立つ．広範囲でXMLが利用されるようになり，コモンツールの開発や，アプリケーションを構築するインフラストラクチャが提供されてきた．これらにより，次世代のビジネスコンピューティングインフラストラクチャの重要なコンポーネントであるSOAPやXMLベースのwebサービスなどの普及に拍車がかかった．しかしながら，XMLが次世代のビジネスコンピューティングインフラストラクチャとして用いられるが，広く用いられているXMLの解析技術は，XMLに基づいたコンピューティングインフラストラクチャの実行要求を満たすことができない．

XMLデータベースは，XML形式の情報を格納できるデータベースであり，例えばIBM社のDB2やOracle社のOracle XML DB，Microsoft社のSQL Server XMLなどがある．XMLのスキーマ言語の1つとしてDTDが使われているが，データ型が定義できない点や，文法がXMLと無関係である点など様々な欠点が指摘されている．W3CではXML Schemaの開発を進めているが，仕様が複雑で標準化は難航している．他の主なスキーマ言語として，XDR，SOX，Schematron，DSD，RELAX NGなどがある．これらのスキーマ言語を，使いやすさの観点で比較すると，DTDが独自構文であるにもかかわらず，スキーマ言語習得が最も簡単であるとされている．Schematronの言語仕様は非常に単純で記述性が高いが，XPathを知っていることが前提になる．RelaxingはXML Schema並みの記述性を兼ね備えていながら，シンプルな構造の言語である．スキーマ言語の比較について，詳しくは文献[3]に記されている．XMLデータベースへの問い合わせ言語は，いくつか提案されているが，広く普及されていて機能性の高いものに，W3Cから勧告を受けているXQueryがある．XQueryは，表現式という単位から構成され，表現式の組み合わせによって複雑な問い合わせを記述できる関数型言語である．XQuery 1.0はXPath 2.0をサブセットとして含んでおり，XMLの各要素を指定するにはXPathを使用する．しかしながら，一般的にXMLの検索の用いられる問い合わせ言語のXQueryは，検索に関する問い合わせしか行えず，更新や削除などに対してパスの指定を行えるのみである．W3CによりXQuery Update Facilityが考案されているが，勧告が難航しており，XQuery Update Facilityを実装しているシステムがほとんど無いのが現状である．また，検索の問い合わせなどを行う際に，ユーザは詳しい内部構造を知っている必要があり，既存の検索は，タグ構造のみに着目した検索である．

本稿では，制約論理プログラミングに基づくXMLデータベースの作成について論じる．本システムは，XML文書に出現するタグの意味的な情報，例えばタグの構造パターンや要素のデータの種類のような情報からXMLデータを格納や更新などを行う際に学習し，制約論理プログラミングの制約として格納するシステムである．Semantic Webのオントロジーのように，あらかじめ意味を持った共通の物差しを仕様して意味を解釈するのではなく，制約論理プログラミングのスキーマ構造をXMLデータから生成し，そのスキーマ構造がXMLデータの意味的な定義となっているため，異なるXML構造の文書に対して同じ意味のデータを認識することができる．また，本システムはタグやタグ要素に対する数値計算の制約を処理することも可能とする．

† 名古屋工業大学大学院 工学研究科 情報工学専攻

2 制約論理プログラミング

ある問題を解決しようとするとき，その問題自身を厳密に記述する必要がある．そのために，問題の領域とその領域において問題を構成する対象及びこれら対象間の関係を明確にしなければならない．制約とは，このような対象間の関係を宣言的に記述したものである．制約論理プログラミングは，論理プログラミングに制約の考えを付加したものである．

本システムの制約論理プログラミングは，大囿らが提案しているリフレクティブな制約論理プログラミングの提案を参考に実装されている[2]．

論理プログラミングにおけるユニフィケーションは，等式の左辺はすべて変数であり，左辺に現れる変数は互いに異なり，左辺に現れる変数は右辺に現れないという制約を持つとき，すべての充足可能な制約に対してまったく同じ解を持つ標準形の制約がただ1つ存在することを保証している．本システムでは，線形方程式，線形不等式を扱うために制約解消系としてシンプレックス法を用いており，制約に対して制約充足可能かどうかを判定し，充足可能であればすべての解を見つけ出す．また，解が有限個の場合には，生成手続きが必ず停止するという条件も持つ．

3 データベースの作成

XMLデータから知的情報を取得する試みとして，浅井らが木のグラフのマイニングに関する研究を行っている[1]．著者らは，半構造データストリームマイニングの手法を提案している．また，姚らがsuffix arrayを用いたXMLデータの検索に関する研究を行っている[6]．本システムは，suffix arrayにてXMLデータを格納し，パターンなどの発見を行っている．

格納されるXMLデータは，あらかじめスキーマが定義されている必要は無く，XML文書を格納してから制約によるスキーマを生成する．このとき生成されるスキーマは，タグとタグの関係をタイトに表現するものではなく，意味的なタグの集合を制約で表している．

本システムは，最初にXMLデータのタグの親子関係やタグの要素，属性などの制約を格納する．次に，部分的なタグ構造に関する制約を格納し，最後にそれらの制約の重複を削除したり，意味的な対応関係などを判別する．

3.1 タグに関する情報の格納

XMLデータのタグの親子関係やタグの要素，属性などの制約を格納する流れについて論じる．最初に，XMLデータをタグごとに分解した配列を作成する．その配列の1番目の要素は，XMLデータを木構造で表現したときのルートタグであるので，配列の1番目の要素を下記のように格納する．また，XMLデータには同じタグが存在するので，特定のタグを指定するために配列の要素の入っている番号を，そのタグのidとして格納する．

```
root(配列の1番目の要素のタグ).
root(1).
```

制約 root(X) は，ルートタグを表している．配列の 2 番目の要素から最後の要素は，すべて同じ処理を行う．n 番目の要素のタグに関して，もしそのタグが葉であるとき，下記のように制約を格納する．

```
leaf(配列の n 番目の要素のタグ).
leaf(n).
```

制約 leaf(X) は，そのタグが葉であることを表している．さらに，タグが葉であったとき，そのタグが要素を持つときにさらに下記の制約を格納する．

```
element(配列の n 番目の要素のタグ, タグの要素).
element(n, タグの要素).
```

制約 element(X, Y) は，X の要素が Y であることを表している．また，葉でなかったとき，そのタグの親子関係を表す制約を下記のように格納する．

```
parent(配列の n 番目の要素のタグ,n の子タグ).
parent(n,n の子タグの格納されている配列の番号).
```

制約 parent(X, Y) は，X が Y の親であることを表している．最後に，n 番目の要素のタグが葉であるかどうかに関わらず，そのタグが属性を持てば下記の制約を格納する．

```
attribute(配列の n 番目の要素のタグ,[属性名, 値]).
attribute(n,[属性名, 値]).
```

制約 attribute(X, Y) は，X の属性に関するリストが Y であることを表している．リスト Y は，属性名と値の繰り返しである．XML データのタグの親子関係やタグの要素，属性などの制約を格納するアルゴリズムを Algorithm 1 に示す．Algorithm 1 の make_F は，制約 F を格納する手続きである．例えば make_root は，制約 root を格納する手続きである．

3.2 部分的なタグ構造の格納

部分的なタグ構造に関する制約を格納する流れについて論じる．最初に，タグ構造を作り始めるタグの子タグを取得する．本システムでは，作り始めるタグの初期設定には，ルートタグが設定されている．そして，子タグを 1 つずつ取得し，子タグがなくなるまで繰り返す．

取得した子タグに関する操作において，まずそのタグの親子関係を表す制約を下記のように作る．

```
parent(作り始めるタグ, 現在の子タグ).
parent(作り始めるタグの id, 現在の子タグの id).
```

次に，現在の子タグが葉であるとき，下記のように制約を作る．

```
leaf(現在の子タグ).
leaf(現在の子タグの id).
```

さらに，現在の子タグが葉であったとき，そのタグが要素を持つときにさらに下記の制約を作る．

```
element(現在の子タグ, タグの要素).
element(現在の子タグの id, タグの要素).
```

また，現在の子タグが葉でなかったとき，その子タグを新たにタグ構造を作り始めるタグとして，再帰的に部分的なタグ構造の格納を行う．最後に，現在の子タグが葉であるかどうかに関わらず，そのタグが属性を持てば下記の制約を作る．

Algorithm 1. Storing XML Data algorithm

```
StoringTagData(Tag, tagSize)
1 : Integer i ← 0;
2 : while i − 1 ≠ tagSize do
3 :   if i = 0 then
4 :     make_root(Tag[i]);
5 :     make_root(an id of Tag[i]);
6 :   else
7 :     if Tag[i] ∈ a set of leaf tags then
8 :       make_leaf(Tag[i]);
9 :       make_leaf(Tag[i] id);
10 :      if Tag[i] has an element then
11 :        make_element(Tag[i]);
12 :        make_element(Tag[i] id);
13 :      end if
14 :    else
15 :      make_parent(Tag[i], child Tag[i]);
16 :      make_parent(Tag[i] id, child Tag[i] id);
17 :    end if
18 :    if Tag[i] has an attribute then
19 :      make_attribute(Tag[i]);
20 :      make_attribute(Tag[i] id);
21 :    end if
22 :  end if
23 :  i ← i + 1;
24 : end while
```

```
attribute(現在の子タグ,[属性名, 値]).
attribute(現在の子タグの id,[属性名, 値]).
```

全ての制約を作り終えたら，それらすべての制約を持つ新たな制約を下記のようにデータベースに格納する．

```
タグ構造を作り始めるタグ :- 全ての制約.
```

部分的なタグ構造に関する制約を格納するアルゴリズムを Algorithm 2 に示す．Algorithm 2 の l12 では，このアルゴリズムを再帰的に呼び出している．l17 の make(constraint) は，制約の集合 constraint をすべて含む制約を生成する手続きである．

3.3 意味的なタグ構造の判断

3.2 節で生成された制約に関して，タグ構造を作り始めるタグには同じタグが存在することがあるので，同じ制約が作られる可能性がある．それらの制約に関して，本節では重複を削除したり，意味的な対応関係などを判別する流れについて論じる．本システムでは，タグ構造を意味的に考え，大まかには形式の異なったタグ構造が同じ意味である，ある程度の関係がある，全く関係がないの 3 つに分類する．

一般に，ユーザやアプリケーションの違いによって，意味的に同じタグ構造が，異なった形式のタグ構造として作られる場合がある．シングルユーザのみで用いられるシステムなら問題はあまり無いが，一般にはデータベースシステムは多人数が作ったデータを多人数で操作するシステムであることが多々あるので，自分で作ったタグ構造を操作することは可能だが，他のユーザが作ったタグ構造に対して操作を行うことが難しくなる．

意味的なタグ構造の判断の流れについて論じる．最初に，3.2 節で生成された制約を取得する．そして，制約を 1 つずつ取得し，制約がなくなるまで繰り返す．取得した制約に関

Algorithm 2. Storing XML structure algorithm

```
StoringXMLStructure(start_id)
1 : List child_id ← parent(start_id, X)
2 : List constraint ← {};
3 : while child_id ≠ φ do
4 :   Integer current_id ← child.pop();
5 :   constraint ← parent(start_id tag, current_id tag);
6 :   if leaf(current_id) then
7 :     constraint ← leaf(current_id tag);
8 :     if element(current_id) then
9 :       constraint ← element(current_id);
10 :    end if
11 :  else
12 :    StoringXMLStructure(current_id)
13 :  end if
14 :  if attribute(current_id) then
15 :    constraint ← attribute(current_id);
16 :  end if
17 :  make(constraint);
18 :  constraint.clear();
19 : end while
```

する操作は，それまで確認された制約を格納しているクローズドリストのすべての制約と比較を行うことである．

制約との比較は，最初にその制約とクローズドリストの制約が完璧に等しい，つまり形式的なタグ構造が等しいかどうかを判定し，もし等しいならば全く同じ制約が複数存在することになるで，その制約を破棄する．

次に，現在取得している制約が，クローズドリストのある制約の部分集合であるか判定する．クローズドリストのある制約の部分集合となるとき，さらにその部分集合の大きさから，強い関連性があるか，そこまで関連性が無いかを判定する．強い関連性があるならば，クローズドリストのある制約を，現在取得している制約を含むように書き換える．もしその制約が，現在取得している制約を含む表現であるときは，制約は書き換える必要はない．そして，現在取得している制約のデータを破棄する．

それから，クローズドリストのある制約が，現在取得している制約の部分集合であるか判定する．現在取得している制約の部分集合となるとき，先程と同様に，さらにその部分集合の大きさから，強い関連性があるか，そこまで関連性が無いかを判定する．強い関連性があるならば，現在取得している制約を，クローズドリストのある制約を含むように書き換える．もしその制約が，クローズドリストのある制約含む表現であるときは，制約は書き換える必要はない．そして，現在取得している制約とクローズドリストのある制約を交換し，交換された状態の現在取得している制約のデータを破棄する．

最後に，タグ構造が異なっているが意味的には等しい制約の有無を判定する．この判定は，タグの数や，深さ，かたち(谷が 1 つであるかなど)，タグの要素を用いて行う．

また，現在取得している制約が，上記の 4 つのどれにも当てはまらなかったとき，その制約をクローズドリストに格納する．

意味的なタグ構造の判断に関するアルゴリズムを Algorithm 3 に示す．Algorithm 3 の l13 及び l21 の if 文は，部分集合の関連性が強いか，そこまで関連性が無いかを判定するものである．l14 及び l22 は，強い関連性があったときの処理で，制約を書き換える処理，制約を交換する処理，制約を破棄する処理のことである．l16 及び l25 は，関連性があまりなかった処理で，部分集合となっている制約から，母集

Algorithm 3. Decision semantics algorithm

```
Decision semantics(structure_constraint)
1 : List closed_list ← {};
2 : Boolean relation_flag ← false;
3 : while structure_constraint ≠ φ do
4 :   String current ← structure_constraint.pop();
5 :   List temp_list ← closed_list;
6 :   while temp_list ≠ φ do
7 :     String temp ← temp_list.pop();
8 :     if current ≡ temp then
9 :       current ← null;
10 :      relation_flag ← false;
11 :    else
12 :      if current ⊂ temp then
13 :        if regarded as same meaning then
14 :          refer to temp for current;
15 :        else
16 :          refer to temp for current weakly;
17 :        end if
18 :        relation_flag ← false;
19 :      else
20 :        if current ⊃ temp then
21 :          if regarded as same meaning then
22 :            refer to current for temp;
23 :            swap_status(temp, current);
24 :          else
25 :            refer to temp current temp weakly;
26 :          end if
27 :        else
28 :          if current = temp then
29 :            make_samemeans(current, temp);
30 :          end if
31 :        end if
32 :        relation_flag ← false;
33 :      end if
34 :    end if
35 :  end while
36 :  if relation_flag then
37 :    closed_list ← temp;
38 :  end if
39 :  temp_list.clear()
40 : end while
```

合の制約を参照する処理のことである．l29 は，タグ構造が異なっているが意味的には等しい制約に関する処理で，タグの数や，深さ，かたち，タグの要素を用いて処理を行う．

3.4 制約の更新

ユーザの問い合わせに関する情報は，リアルタイムで制約に反映させるのではなく，一定量溜まったらバッチ処理を行う．問い合わせや構造には，よく用いられる問い合わせや参照される構造，ほとんど参照されない問い合わせや構造などがある．システムとして同じ意味の構造であると認識していない複数の構造に対して，ユーザが何度もそれらの構造に問い合わせを行っている場合，それらの構造が同じ意味であるか関連がある可能性が高いので，パターンの格納を条件付きで行う．この条件付きとは，同じ意味と見なす条件を緩くし，その際に生成される制約を，通常のパターンの格納において生成された制約と違うという認識をすることである．また，よく参照される構造に対しては，評価する順番の優先度を上

表 1: 実験結果

Number of nodes	Document size (Kbytes)	time (ms)	
		time	reasoning
10,000	1,500	78	51
50,000	7,520	164	100
100,000	16,800	251	184
150,000	25,100	420	275
200,000	33,500	662	320
250,000	41,000	990	412
300,000	52,300	1335	573

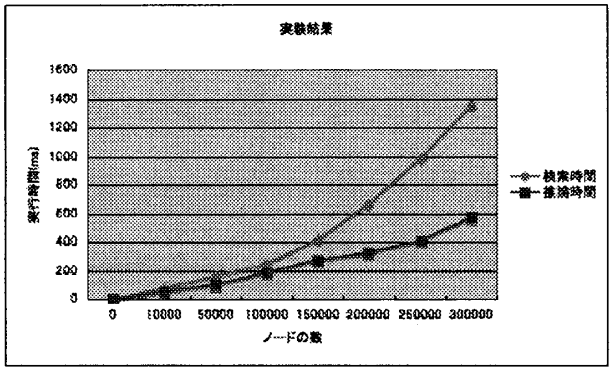

図 1: 実験結果のグラフ

げたり，逆に参照されない構造に対しては優先度を下げたりする．

4 実験と評価

本節では，本システムの検索速度及び制約からの推論の実行速度の実験について論じる．

実験環境は，OS は WINDOWS XP Professional，CPU は 1.8 GHz Pentium4 Processor，メモリは 1GB(SDRAM) である．実験データは，ウェブ上に落ちていた様々な XML 文書を貼り繋いだりしてつくった XML データである．この XML データを，本システムではノードの数に実行速度が比例するので，ノード (XML タグの開きと閉じを組とし，空タグはそれ 1 つでノード 1 つと数える) のサイズ毎に実験を行った．また，そのときに XML データのドキュメントサイズも記述してある．実験では，ルートタグから木構造としてすべてのタグを辿ったときの検索速度と，意味的なタグ構造の制約に問い合わせを行った知識推論の実行時間を，それぞれ 100 回の実験を行った検索速度の平均値として記録した．表 1 は，実験の結果を表にしたものである．ここでは，10,000, 50,000, 100,000, 150,000, 200,000, 250,000, 300,000 ノードの XML データに対して実験を行った．そのときの実行時間を ms で記述してある．また，図 1 は，表 1 をグラフで表現したものである．グラフよりわかることは，通常の検索に関してはノード数 n にたいして O(n) となっているが，推論にかかる時間が $O(n^2)$ で増加している．規模が中程度ならば，これらの実行速度において，支障はあまり無い．

5 おわりに

本稿では，制約論理プログラミングに基づく XML データベースの作成について述べた．本システムは，XML のタグの構造パターンや要素のデータの種類のような情報から，XML データを格納や更新などを行う際に学習し，制約論理プログラミングの制約として格納するシステムである．本システムは Java 言語で実装されており，論文 [4]，論文 [5] を参考に実装されている．また，Java 言語で実装されているので，本システムはマルチプラットフォームに対応している．本システムは，Semantic Web のオントロジーとは異なり，制約論理プログラミングのスキーマ構造を XML データの意味的な構造として定義しているため，異なる XML 構造の文書に対して同じ意味のデータを認識することができる．

格納する XML データがスキーマとして定義されている，もしくはそれに準ずるようなタグ構造であるならば，意味的なタグ構造としてほぼ正しく認識することが可能だが，その XML データの意味的なタグ構造が判断し辛かったり，判断が全くできない場合には，意味的な制約はほとんど作られず，検索パフォーマンスが悪くなったり，ユーザにとって欲しいデータをすべて得ることができなくなる．従って今後の展望として，意味的なタグ構造が判断が全くできない場合は，その XML データを作ったユーザ及びアプリケーションの責任だと割り切るが，タグ構造が判断し辛いときの判断精度を，様々なマイニング手法を参考にし，上げることなどが挙げられる．

参考文献

[1] 浅井達哉，有村博紀，“半構造データマイニングにおけるパターン発見技法”，電子情報通信学会論文誌，Vol.87, No.2, pp.79-96(2004).

[2] 大囿忠親，新谷虎松，“マルチエージェントシステムのための制約論理型言語 RXF の実現”，情報処理学会論文誌，Vol. 37, N0. 10, pp.1765-1772(1996).

[3] Dongwon Lee, Wesley W. Chu, “Comparative Analysis of Six XML Schema Languages”, Internet Document.

[4] Margaret G. Kostoulas, Morris Matsa, Noah Mendelsohn, Eric Perkins, Abraham Heifets, “XML Screamer: An Integrated Approach to High Performance XML Parsing, Validation and Deserialization”, pp.93-102, WWW2006.

[5] Morris Matsa, Eric Perkins, Abraham Heifets, Margaret Gaitatzes Kostoulas, Daniel Silva, Noah Mendelsohn, Michelle Leger, “A High-Performance Interpretive Approach to Schema-Directed Parsing”, pp.1063-1072, WWW2007.

[6] 姚 全珠，都司 達夫，“Suffix Array に基づく大規模 XML 文書のための高速サーチエンジン”，福井大学工学部研究報告，Vol.52, No.2, pp.153-160(2004).