Homomorphism Resolving of XPath Trees Based on Automata*

Ming Fu and Yu Zhang[1,2]

[1] Department of Computer Science & Technology,
University of Science & Technology of China, Hefei, 230027, China
[2] Laboratory of Computer Science, Chinese Academy of Sciences, Beijing, 100080, China
brightfu@ustc.edu, yuzhang@ustc.edu.cn

Abstract. As a query language for navigating XML trees and selecting a set of element nodes, XPath is ubiquitous in XML applications. One important issue of XPath queries is containment checking, which is known as a co-NP complete. The homomorphism relationship between two XPath trees, which is a PTIME problem, is a sufficient but not necessary condition for the containment relationship. We propose a new tree structure to depict XPath based on the level of the tree node and adopt a method of sharing the prefixes of multi-trees to construct incrementally the most effective automata, named XTHC (XPath Trees Homomorphism Checker). XTHC takes an XPath tree and produces the result of checking homomorphism relationship between an arbitrary tree in multi-trees and the input tree, thereinto the input tree is transformed into events which force the automata to run. Moreover, we consider and narrow the discrepancy between homomorphism relationship and containment relationship as possible as we can.

Keywords: XPath tree, containment, homomorphism, automata.

1 Introduction

XML has become the standard of exchanging a wide variety of data on Web and elsewhere. XML is essentially a directed labeled tree. XPath[1] is a simple and popular query language to navigate XML trees and extract information from them.

XPath expression p is said to contain another XPath expression q, denoted by $q \subseteq p$, if and only if for any XML document D, if the resulting set of p returned by querying on D contains the resulting set of q. Containment checking becomes one of the most important issues in XPath queries. Query containment is crucial in many contexts, such as query optimization and reformulation, information integration, integrity checking, etc. However, [2] shows that containment in fragment $XP^{\{[\,],*,//\}}$ is co-NP complete. The authors proposed a complete algorithm for containment, whose complexity is EXPTIME. The authors also proposed a sound but incomplete PTIME

* This work is supported by the National Natural Science Foundation of China under Grant No. 60673126, and the Foundation of Laboratory of Computer Science, Chinese Academy of Science under Grant No. SYSKF0502.

algorithm based on homomorphism. This algorithm may return false negatives because the homomorphism relationship between two XPath trees is a sufficient but not necessary condition for the containment relationship. In many practical situations containment can be replaced by homomorphism.

The homomorphism algorithms proposed in [2][3] are mainly focused on how to resolve the containment problem between two XPath expressions. In [3] the authors proposed hidden-conditioned homomorphism to further narrow the discrepancy between homomorphism and containment based on [2]. However, the homomorphism relationship was considered in these works only between two XPath trees. In practice we may need to verify the homomorphism relationship between an arbitrary tree in a set of XPath trees and the input XPath tree, such as filtering redundant queries in a large query set. It is inefficient to check one by one using the homomorphism algorithm, because the same prefix and branch in multi-trees will cause redundant computing. Although a method handling this was discussed in [4], it will return false negatives for some XPath expressions which have containment relationship, such as XPath expressions $p = /a//*/b$, $q = /a/*//b$ etc.

In this paper, we propose an efficient method to check homomorphism from multi-trees to a single XPath tree based on automata.We also narrow the discrepancy between homomorphism and containment as possible as we can. Our major contributions are: 1)We propose the fixed tree and alterable tree to describe the XPath tree, and define homomorphism based on them. 2)We define XTHC machine, a kind of indexed incremental automata with prefix-sharing of multi-trees, and our method can give the optimal automata. 3)We propose an algorithm to check homomorphism from multi-trees to a single tree based on XTHC machine. 4)The experiment results demonstrate both the practicability and efficiency of our techniques.

The rest of this paper is organized as follows. Section 2 gives some basic notations and definitions. Section 3 is the major part of our work, that is, how to construct XTHC machine and how to use XTHC to resolve the homomorphism problem. The last two sections present the experimental evaluation and conclusions, respectively.

2 Preliminaries

Each XPath expression has a corresponding XPath tree. The XPath tree given in [2] uses each node test in the XPath expression as a node in the tree, and classifies its edges into child-edge and descendant-edge according to the type of axes in the XPath expression. This description is straightforward and easy to understand, however, difficult to expand. If there is any backward axis (parent-axis or ancestor-axis) in the XPath expression, this method is no longer applicable to describing the tree structure.

We now give a different description of XPath tree, in which the level information between the adjacent two node tests is abstracted from the type of the axis between the node tests, and recorded at the corresponding node in the XPath tree. Our work is limited to $XP^{\{[\,],*,//\}}$ expression only.

Definition 1: For a given $XP^{\{[\,],*,//\}}$ expression q, we construct an XPath tree T. The root of T is independent of q. Every node test n in q can be described by a non-root node v. The relationship between v and its parental node v' is denoted by $L(v)=[a, b]$, where *a* and *b* are the minimum and maximum numbers of levels between *v* and *v'* respectively. The relationship between nodes in tree *T* is given as:

1) If *n* is a root node test, i.e. */n* or *//n*, there exists an edge in *T* between the node *v* in *T* that corresponds to *n*, and the root *r*, *edge*(*r*, *v*), where *r* is the parental node of *v*. When */n*, $L(v)$=[1, 1]; and $L(v)$=[1, ∞] when *//n*.

2) If *n* is not a root node test, there is an adjacent node test *n'* in *q* that satisfies *n'/n*, *n'*[*n*], *n'//n* or *n'*[.//*n*], therefore, there exists an edge in *T* between *v* and *v'* (corresponding to *n* and *n'* respectively), where *v'* is the parental node of *v*. When *n'/n* or *n'*[*n*], $L(v)$=[1, 1]; and $L(v)$=[1, ∞] when *n'//n* or *n'*[.//*n*].

Definition 2: Given an XPath tree *T*, let *NODES*(*T*) be the set of nodes in *T*, *EDGES*(*T*) be the set of edges in *T*, *ROOT*(*T*) be the root node of *T*. If there exists $v \in$ *NODES*(*T*), and the outdegree of *v* is greater than 1, or the outdegree or the indegree of *v* is 0, node *v* is then called *key node* of the XPath tree *T*. $\forall$ *edge*(*x,y*)$\in$ *EDGES*(*T*), where *x,y*$\in$ *NODES*(*T*), and *edge*(*x,y*) implies *x* is the parental node of *y*. If *nid* is the unique idtentifier of node *y* and *ln* is the label of node *y*, we then denote node *y* by $nid^{[a,b]}$, where [*a*,*b*] equals to $L(y)$.

Informally, key nodes in an XPath tree are branching nodes (nodes with outdegree greater than 1), leaves, and root.

There are often some wildcard location steps without predicate used in an XPath expression, which are represented as non-branching nodes '*', such as the expression */a/*//*/b*. We can remove those wildcard nodes in the XPath tree for simplification, but have to revise the $L(v)$ value of some related non-wildcard node *v* which is the descendent node of the removed wildcard node. Fig. 1(a) illustrates the two XPath trees of the expression */a/*//*/b* before and after removing non-branching wildcard nodes, where $L(b)$ is revised. In the following context, all XPath trees are those trees from which the non-branching wildcard nodes are removed.

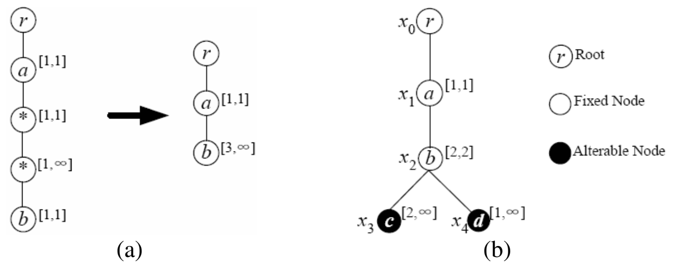


Fig. 1. (a)XPath Tree */a/*//*/b* ; (b)XPath Tree */a/*/b*[.//*/*c*]//*d*

Definition 3: Given an XPath tree *T*, let *CNODES*(*T*) be the set of alterable nodes, *FNODES*(*T*) be the set of fixed nodes, *NODES*(*T*)={*ROOT*(*T*)} $\cup$ *CNODES*(*T*) $\cup$ *FNODES*(*T*). $\forall$ $n \in$ *NODES*(*T*) and $n \neq$ *ROOT*(*T*), $L(n)$ = [*a*,*b*]. If *a*=*b*, then $n \in$ *FNODES*(*T*); if $b = \infty$, then $n \in$ *CNODES*(*T*). When *CNODES*(*T*) is not empty, the XPath tree *T* is an *alterable tree*, otherwise it is a *fixed tree*.

As an example, the XPath tree of the XPath expression */a/*/b*[.//*/*c*]//*d* is shown in Fig. 1(b). The set of level relationship between node x_2 and its parental node is $L(x_2)$=[2,2]. From definition 3, node x_2 is a fixed node. The set of level relationship between node x_3 and its parental node is $L(x_3)=[2,\infty]$, and node x_3 is an alterable node, so the corresponding XPath tree is an alterable tree.

Definition 4: Function *h*: *NODES*(*p*) $\longrightarrow$ *NODES*(*q*) describes the homomorphism relationship from XPath tree *p* to XPath tree *q*:

1)*h*(*ROOT*(*p*)) = *ROOT*(*q*);

2)For each $x\in$ *NODES*(*p*), *LABEL*(*x*)='*' or *LABEL*(*x*) = *LABEL*(*h*(*x*));

3)For each *edge*(*x*,*y*) $\in$ *EDGES*(*p*), where $x,y\in$ *NODES*(*p*), $L(x,y)\supseteq L(h(x),h(y))$;

Fig. 2 shows the homomorphism mapping *h* from XPath tree *p* to XPath tree *q* based on XPath expressions /*a*/*//*b* and /*a*[*c*]//*/*//*b*.

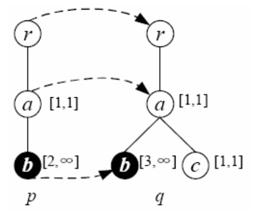


Fig. 2. Homomorphism mapping *h*:*p*→*q*

3 Homomorphism Resolution Based on XTHC Machine

3.1 Construction of Basic XTHC Machine

We will incrementally construct NFA with prefix-sharing on the set of XPath trees $P=\{p_1,p_2\ldots p_n\}$. Each node $nid^{[a,b]}$ in the XPath tree will be mapped to an automata fragment in NFA, and such a fragment has a unique start state and a unique end state. There are two cases while constructing the fragment from the node $nid^{[a,b]}$:

1. When $a=b$, $nid^{[a,b]}$ is a fixed node, the constructed automata fragment is shown in Fig.3(a). The states *s*-1 and *s*+*a*-1 are the start and end states of the fragment, respectively. Since *a* represents the minimum number of levels between node $nid^{[a,b]}$ and its parental node, starting from state *s*-1, we can construct in turn *a*-1 states along the arcs labeled '*', which are called *extended states*; we then construct state *s*+*a*-1 along the arc labeled *ln* from state *s*+*a*-2. Obviously there exist extended states in the automata fragment based on $nid^{[a,b]}$ when *a*>1.

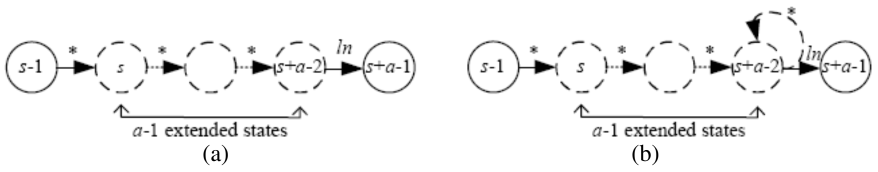


Fig. 3. (a) The automata fragment corresponding to the fixed node $nid^{[a,a]}$; (b) The automata fragment corresponding to the alterable node $nid^{[a,\infty]}$

2. When $b=\infty$, $nid^{[a,b]}$ is an alterable node, many kinds of automata fragment can be constructed, one example is shown in Fig.3(b). Similarly to that in case 1, we first construct *a*-1 extended states and the end state *s*+*a*-1, starting from state *s*-1. Since $b=\infty$, it is necessary to add self-looping arc, labeled by '*', in any one or more states from state *s*-1 or the following *a*-1 extended states. The chain consisting of the start state and the extended states, is denoted by *extended state-chain*. Fig.3(b) only shows one self-looping arc at last state of the extended state-chain. Obviously,

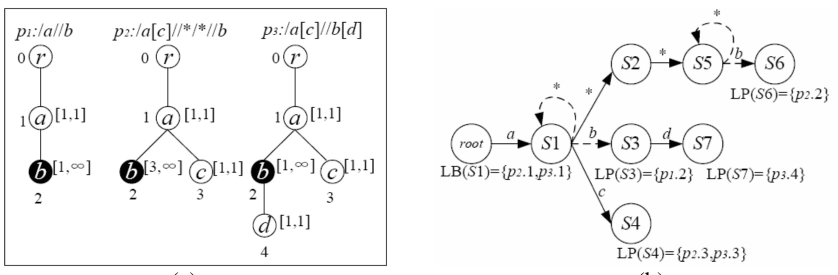
an automata fragment corresponding to an alterable node $nid^{[a,b]}$ (a>1) in an XPath tree p is optimal, if and only if there is only one state in the fragment that has a self-looping arc, and this state must be the last state along the extended state-chain.

Definition 5: Suppose the NFA constructed from set P of XPath trees is A, called the XHTC machine. We can create the following two index tables for each state s in A:

1) LP(s): list of leaf nodes. $\forall p\in P$, for each leaf node n_l in p, if s is the last state constructed from n_l, then $n_l\in$ LP(s). Only when s is a leaf state, LP(s) is non-empty.

2) LB(s): list of branching nodes. $\forall$ $p\in P$, for each branching node n_b in p, if s is the last state constructed from n_b, then $n_b\in$ LB(s). Only when s is a branching state, LB(s) is non-empty.

Fig. 4(b) is the XTHC machine constructed from XPath trees p_1, p_2, and p_3 which are shown in Fig.4(a), $p_i.x$ represents node x in XPath tree p_i, a state is denoted by a circle. An arc implies state transition, where dashed lines represents transition of descendant-axis type, and solid lines represents transition of child-axis type. A label on an arc is a node test. State $S1$ has an arc to itself since it has a transition of descendant-axis type.



(a) (b)

Fig. 4. (a) The XPath tree set P ; (b) The XTHC machine constructed from XPath tree set P

Definition 6: A basic non-deterministic XTHC machine A is defined as:

$$A = (Q^s, \Sigma, \delta, q^s_0, F, B, S_s)$$

where

- Q^s is the set of NFA states;
- Σ is the set of input symbols;
- q^s_0 is the *initial*(or *start*) NFA state of A, i.e. the root state;
- δ is the set of state transition functions, it contains at least the NFA state transition function, i.e. $t_{forward}$: $Q^s\times\Sigma\rightarrow 2^{Q^s}$;
- $F\subseteq Q^s$ is the set of *final* states, it is also the set of *leaf* states;
- $B\subseteq Q^s$ is the set of *branching* states;
- $\forall$ $q^s\in Q^s$, we call q^s an NFA state of A, LP(q^s) and LB(q^s) are two index tables of q^s (see definition 5); S_s is the stack for state transition, the stack frame of S_s is a subset of Q^s.

3.2 Running an XTHC Machine

In order to resolve the homomorphous relationship using an XTHC machine, a depth-first traverse on the input XPath tree is required to generate SAX events. These events will be used as input to the XTHC machine for the XTHC machine running.

Four types of events will be generated at depth-first traverse on the input XPath tree *p*: startXPathTree, startElement, endElement and endXPathTree. Time of these events being generated is:

1) send startXPathTree event when entering root of *p*;
2) send startElement event when entering non-root node of *p*;
3) send endElement event when tracing back to non-root node of *p*;
4) send endXPathTree event when tracing back to root of *p*.

Since *a* and *b* are not always 1 in a node $nid^{[a,b]}$ of an XPath tree, more than one events are sent at entering or tracing back to node $nid^{[a,b]}$:

1) the startElement event sequence sent when a=b is shown in Fig.5(a);
2) the startElement event sequence sent when b=∞ is shown in Fig.5(b).

In particular, there are some restrictions applied on a startElement('//') event:

1) it occurs only when node $nid^{[a,b]}$ is an alterable node;
2) state transition driven by this event occurs only at state *s* in the extended state-chain corresponding to the alterable node, and there is a unique state transition:

$$t_{forward}(s, \text{'//'}) \rightarrow s.$$

Similarly, more than one endElement event are sent when tracing back to node $nid^{[a,b]}$ in the tree, which are shown in Fig. 5(c) and 5(d).

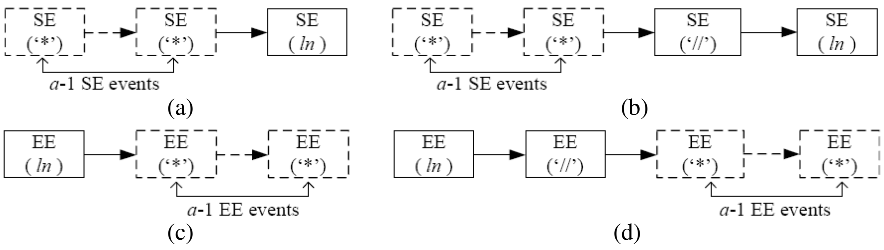


Fig. 5. (a) The startElement("SE" for short) event sequence of the fixed node $nid^{[a,a]}$; (b) The startElement event sequence of the alterable node $nid^{[a,\infty]}$; (c) The endElement("EE" for short) event sequence of the fixed node $nid^{[a,a]}$; (d) The endElement event sequence of the alterable node $nid^{[a,\infty]}$

Fig. 6 shows rules of processing SAX events in an XTHC machine. The homomorphism relationship between tree p_i in a set of XPath tree P={p_1,p_2,…,p_n} and an input tree *q* can be resolved by running the XTHC machine. When the XTHC machine is running, $\forall p \in P$, homomorphism information between each node *v* in *p* and nodes in the input tree *q* is recorded. Let $v \in p$, *a* be the label of node *u* in the input XPath tree *q*. We define the following three operations to mark, deliver and reset information about the mapping in the XPath tree *p*:

1) mark(v, u): when the XTHC machine is running at a leaf state $q^s(q^s \in F)$, $\forall\ v \in$ LP(q^s), mark on v the information about the mapping from the leaf node v to the node u in the input XPath tree q;
2) deliver(v): when the machine traces back to a key state $q^s(q^s \in F \cup B)$, $\forall\ v \in$ LB(q^s) $\cup$ LP(q^s), if information about the mapping was marked on node v, deliver the mapping information of v to the nearest ancestor key node to v in the XPath tree;
3) reset(v): when the machine traces back to a key state $q^s(q^s \in F \cup B)$, $\forall\ v \in$ LB(q^s) $\cup$ LP(q^s), reset the mapping information on node v.

```
startXPathTree()
  push($S_s$, {$q^s_0$}); other initialization
startElement($a$)
  $q^s_{set}$ ={ }; // current NFA state set
  $u$ = getCurrentInputNode( );
  for each $q^s$ in peek($S_s$)
    merge $t_{forward}(q^s, a)$ into $q^s_{set}$
  push($S_s$, $q^s_{set}$);
  for each $q^s$ in $q^s_{set}$
    if ($q^s \in F$)
      for each $v$ in LP($q^s$)

endElement($a$)
  $q^s_{set}$ = pop($S_s$);
  for each $q^s$ in $q^s_{set}$
    if ($q^s \in B$ or $q^s \in F$){
      for each $v$ in LB($q^s$) or LP($q^s$)
        if exsit mapping of $v${
            deliver($v$);
            reset($v$);
        }
    }
endXPathTree( )
```

Fig. 6. The processing rules of SAX events in XTHC

The time complexity of the algorithm resolving homomorphism from one XPath tree p to another XPath tree q is O(|p||q|^2)[2]. Therefore, the time complexity from each tree p in a set of XPath tree P={p_1,p_2,…,p_n} to q is O(n|p||q|^2) without using prefix-sharing automata. However, if prefix-sharing automata is used, the time complexity is O(m|q|^2), where m is the number of states in NFA. When XPath trees in P have common branches and prefixes, n|p| is much greater than m, therefore, it is much more efficient to resolve homomorphism from multi-XPath trees to one single XPath tree using prefix-sharing automata.

4 Experiments

An algorithm resolving homomorphism based on the XTHC machine (XHO) was implemented using Java. The experimental platform is on Windows XP operation system, Pentium 4 CPU, with frequency of 1.6GHz and memory of 512MB. We compared several algorithms: the homomorphism algorithm (HO)[2], the complete algorithm in a cononical model (CM), branch homomorphism algorithm(BHO)[4], and the proposed XHO algorithm. We checked the scope of each algorithm at resolving containment of XPath expressions (see table 1, where T/F represents p containing/not containing q), and the time complexity of these algorithms(see Fig. 7).

This experiment shows XHO is as capable as existing homomorphism algorithms. Furthermore, XHO supports containment calculation from multi-XPath expressions to one single XPath expression. Although BHO also supports such calculation, it may give incorrect results in some cases as shown in Table 1. BHO gives a result that is rather different from the correct result CM gives. Compared to BHO, XHO gives smaller discrepancy between containment and homomorphism.

Table 1. Some pairs of XPath trees for experiments and containment results

No	*p*	*q*	HO	BHO	XHO	CM
no.1	*/a//*[.//c]//d*	*/a//b[c]//d*	T	T	T	T
no.2	*/a/*/*/c*	*/a/b[c]/e/c*	T	T	T	T
no.3	*/a//b[*//c]/b/*c	*/a//b[*//c]/b[b/c]//c*	T	T	T	T
no.4	*/a//*/b*	*/a/*//b*	T	F	T	T
no.5	*/a/*[.//b]//c*	*/a//*/b/c*	F	F	T	T
no.6	*/a[a//b[c/*//d]/b/c/d]*	*/a[a//b[c/*//d]/b[c//d]/b/c/d]*	F	F	F	T
no.7	*/a/*/*/*/c*	*/a//*/b//b/c*	F	F	F	F
no.8	*/a//b[c]/d*	*/a/b[.//c]//d*	F	F	F	F

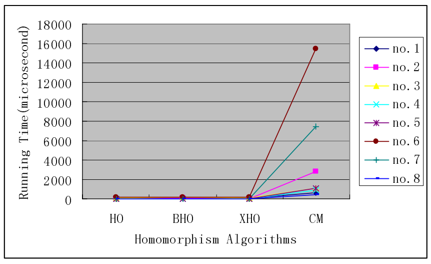


Fig. 7. The experimental results for some homomorphism algorithms

5 Conclusion

This paper considers an algorithm to resolve containment between multi-XPath expressions and one single XPath expression through homomorphism. While high efficiency is kept at calculating multi-containment relationships, we also consider discrepancy between containment and homomorphism. The algorithm works correctly on calculating containment of a special type of XPath expressions. Experiments showed that our algorithm is more complete than conventional homomorphism algorithms. Future research can be done on how to resolve homomorphism between one XPath tree and multi-XPath trees simultaneously.

References

[1] World Wide Web Consortium, XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath, W3C Recommendation, November 1999.

[2] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. Journal of the ACM, 51(1):2-45, 2004.

[3] Yuguo Liao, Jianhua Feng, Yong Zhang and Lizhu Zhou. Hidden conditioned homomorphism for XPath fragment containment. In DASFAA 2006, LNCS 3882, 454-467, 2006.

[4] Sanghyun Yoo, Jin Hyun Son and Myoung Ho Kim. Maintaining homomorphism information of XPath patterns. IASTED-DBA2005, 2005, 192-197.