Typing XPath Subexpressions With Respect to an XML Schema

Yasunori Ishihara*, Kenji Hashimoto†, Atsushi Ohno*, Takuji Morimoto* and Toru Fujiwara*
* *Graduate School of Information Science and Technology, Osaka University, Suita, Japan*
Email: ishihara@ist.osaka-u.ac.jp, fujiwara@ist.osaka-u.ac.jp
† *Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Japan*
Email: k-hasimt@is.naist.jp

***Abstract*—This paper discusses typing XPath subexpressions with respect to an XML schema, which is a new static analysis problem of XPath expressions. More formally, the typing problem is to decide whether there exists an XML document conforming to a given XML schema such that the nodes of the document matching to given subexpressions of a given XPath expression are of the given types. Deciding this problem is useful for query rewriting induced by schema evolution or integration. The contribution of this paper includes a decision algorithm for the typing problem, provided that XPath expressions include no path union operator. Moreover, it is shown that the typing problem is reducible to the XPath satisfiability problem in the presence of DTDs, for which many tractability results are known.**

***Keywords*-XPath; static analysis; type; XML schema**

I. INTRODUCTION

Static analysis of XPath expressions is one of the major theoretical topics in the field of XML databases. XPath is a query language for XML documents, where an XML document is often regarded as an unranked labeled ordered tree. An XPath expression specifies a pattern of (possibly branching) paths from the root of a given XML document. The answer to an XPath expression for an XML document t is a set of nodes v of t such that the specified path pattern matches the path from the root to v.

The most popular subtopic of static analysis may be *XPath satisfiability*, where a given XPath expression p is *satisfiable* under a given XML schema S if there is an XML document t conforming to S such that the answer to p for t is a nonempty set. Many tractable combinations of XPath classes and XML schema classes have been investigated so far [1], [2], [3]. Another popular one is *XPath containment* [4], [5], [6]. Moreover, *XPath validity*, which is a dual of XPath satisfiability, is investigated recently [7], [8].

This paper discusses a new problem of static analysis: *typing XPath subexpressions with respect to an XML schema*. We explain it by an example first.

Example 1: Consider the following fragment of an XML schema:

$$\begin{array}{ll} T_e \to \texttt{teachers}(M_t^*), & S \to \texttt{students}(M_s^*), \\ M_t \to \texttt{member}(N\,T_i), & M_s \to \texttt{member}(N\,G_e\,G_r), \\ N \to \texttt{name}(\cdots), & G_e \to \texttt{gender}(\cdots), \\ T_i \to \texttt{title}(\cdots), & G_r \to \texttt{grade}(\cdots). \end{array}$$

T_e, M_t, etc. are *types* defined in this XML schema, while `teachers`, `member`, etc. are *tag names* or *labels*. Note that the type of `member` is not unique in this schema.

Now, consider the following two XPath expressions:

$$p_1 = \downarrow^*\!::\texttt{member}/\downarrow::\texttt{name},\ p_2 = \downarrow^*\!::\texttt{member}/\downarrow::\texttt{gender}.$$

The types of the subexpression $\downarrow^*::\texttt{member}$ of p_1 are M_t and M_s, because in both cases `member` has `name` as its child. On the other hand, the type of $\downarrow^*::\texttt{member}$ of p_2 is only M_s, because `member` of type M_t has no `gender` as its child.

More formally, the typing problem is to decide whether for a given XML schema S, an XPath expression p, and a sequence $(\alpha_1, X_1), \ldots, (\alpha_k, X_k)$ of pairs of positions of p and types of S, there exists an XML document t conforming to S such that the node of t matching to the subexpression of p at α_i is of type X_i for each i ($1 \le i \le k$).

The typing problem can be viewed as a natural extension of the XPath satisfiability problem, and is useful especially for query rewriting and optimization induced by schema evolution or integration. For example, consider again the fragment of the schema in Example 1. Suppose that $M_t \to \texttt{member}(NT_i)$ has evolved into $M_t \to \texttt{t\_member}(NT_i)$. In order to keep the behavior of queries unchanged, p_1 must be rewritten to $(\downarrow^*::\texttt{member} \cup \downarrow^*::\texttt{t\_member})/\downarrow::\texttt{name}$ while p_2 is not necessarily rewritten. As another example, suppose that $M_t \to \texttt{member}(NT_i)$ has evolved into $M_t \to \texttt{member}(NT_iG_e)$. Then, the associated types of the subexpression $\downarrow^*::\texttt{member}$ of p_2 have been changed (i.e., the types are now both M_t and M_s). In this case, p_2 must be rewritten to an expression, say $\downarrow^*::\texttt{students}/\downarrow::\texttt{member}/\downarrow::\texttt{gender}$, so that the subexpression $\downarrow^*::\texttt{member}$ is associated with only M_s.

Under the assumption that XPath expressions include no path union operator, this paper adopts two approaches to developing decision algorithms for the typing problem. The first one is a direct approach. In this approach, a given XML schema and a given XPath expression are translated into finite tree automata, and the associated types are analyzed by computing their intersection automaton. The second one is a reduction-based approach. In this approach, it is shown that the typing problem is reducible to the XPath satisfiability problem in the presence of DTDs, for which many tractability results are known. Moreover, a part of this

 ISBN: 978-1-61208-247-9

result is extended so that all the possible combinations of types of subexpressions can be efficiently enumerated.

The rest of this paper is organized as follows. In Section II several preliminary definitions are provided. Sections III and IV present the direct and the reduction-based approaches, respectively. Section V summarizes the paper.

II. DEFINITIONS

A. Trees and XML documents

An XML document is represented by an unranked labeled ordered tree $t = (V_t, \lambda_t)$, where

- V_t is a prefix-closed, finite set of sequences of positive integers such that if $v \cdot i \in V_t$ and $i > 1$ then $v \cdot (i-1) \in V_t$; and
- λ_t is a mapping from V_t to a set Σ of labels.

Each element in V_t is called a *node* of t. The empty sequence $\epsilon \in V_t$ is called the *root* of t. The parent-child relation and sibling relation between nodes are defined in an ordinary way. We extend λ_t to a function on sequences, i.e., for a sequence $v_1 \cdots v_n$ of nodes, let $\lambda_t(v_1 \cdots v_n) = \lambda_t(v_1) \cdots \lambda_t(v_n)$. Attributes are not handled in this paper.

For any tree structure t (which is not necessarily representing an XML document) and its node v, let $t|_v$ denote the subtree of t rooted at v. For any trees $t_1, \ldots, t_n$ with the same node set V and $v \in V$, let $(t_1 \cdots t_n)|_v$ denote $t_1|_v \cdots t_n|_v$. Also, for any mapping f which returns a tree structure, let $f|_v$ denote a mapping such that $f|_v(x) = f(x)|_v$.

B. Regular expressions

A regular expression over an alphabet Σ consists of constants ϵ (empty sequence) and the symbols in Σ, and operators $\cdot$ (concatenation), $*$ (repetition), and $|$ (disjunction). We exclude $\emptyset$ (empty set) because we are interested in only nonempty regular languages. The concatenation operator is often omitted as usual. The string language represented by a regular expression e is denoted by $L(e)$. The size of a regular expression is the number of constants and operators appearing in the regular expression.

C. Finite tree automata and XML schemas

A *finite tree automaton* TA is a quadruple (N, Σ, B, P), where

- N is a finite set of states,
- Σ is a finite set of labels,
- $B \in N$ is the initial state, and
- P is a finite set of transition rules in the form of $X \to a(e)$ or $X \to Y$, where $X, Y \in N$, $a \in \Sigma$, and e is a regular expression over N called *content model*.

A finite tree automaton $TA = (N, \Sigma, B, P)$ is *local* if for any pair of rules $X \to a(e)$ and $X' \to a'(e')$ in P, $a \neq a'$ whenever $X \neq X'$.

An *interpretation* I_t^{TA} of a tree t for a finite tree automaton $TA = (N, \Sigma, B, P)$ is a mapping from V_t to the set of finite sequences over N satisfying the following conditions:

- $\mathit{first}(I_t^{TA}(\epsilon)) = B$ (note that ϵ is the root of t), where $\mathit{first}(x)$ denotes the first element of sequence x; and
- for any node v with n children ($n \geq 0$), there are transition rules $X_1 \to X_2$, $X_2 \to X_3$,..., $X_k \to a(e)$ in P such that
  - $I_t^{TA}(v) = X_1 X_2 \cdots X_k$,
  - $\lambda_t(v) = a$, and
  - $\mathit{first}(I_t^{TA}(v \cdot 1)) \cdots \mathit{first}(I_t^{TA}(v \cdot n)) \in L(e)$ (note that $v \cdot i$ is the i-th child of v).

A tree t is *accepted* by a finite tree automaton TA if there is an interpretation of t for TA. Let $TL(TA)$ denote the set of trees accepted by TA.

An *XML schema* S is a finite tree automaton such that

- S has no rule in the form of $X \to Y$, and
- S has no pair of rules $X \to a_1(e_1)$ and $X \to a_2(e_2)$ where $a_1 \neq a_2$.

A *DTD* is an XML schema which is also local. In a DTD there is a one-to-one correspondence between N and Σ, so we often use a triple (N, B, P) to mean a DTD. A tree t *conforms* to an XML schema S if t is accepted by S. In this paper, we assume that every XML schema $S = (N, \Sigma, B, P)$ contains no useless states. That is, for each $X \in N$, there are a tree $t \in TL(S)$ and its interpretation I_t^S such that $I_t^S(v) = X$ for some node v of t. Each state in an XML schema is referred to as a *type*. The *size* $|S|$ of an XML schema S is the sum of the size of all content models.

D. XPath expressions

The syntax of an XPath expression p is defined as follows:

$$\begin{aligned} p &::= \chi :: l \mid p/p \mid p \cup p \mid p[p], \\ \chi &::= \downarrow \mid \uparrow \mid \downarrow^* \mid \uparrow^* \mid \rightarrow^+ \mid \leftarrow^+, \end{aligned}$$

where $l \in \Sigma$. Each $\chi \in \{\downarrow, \uparrow, \downarrow^*, \uparrow^*, \rightarrow^+, \leftarrow^+\}$ is called an *axis*. A subexpression in the form of $\chi :: l$ is said to be *atomic*. The *size* $|p|$ of an XPath expression p is defined as the number of atomic subexpressions in p.

A *position* of an XPath expression p is a finite sequence of positive integers representing a node of a parse tree of p. Precisely, the subexpression $p|_\alpha$ of p at position α is defined as follows:

- $p|_\epsilon = p$.
- If $p|_\alpha = p_1/p_2$, then $p|_{\alpha \cdot 1} = p_1$ and $p|_{\alpha \cdot 2} = p_2$.
- If $p|_\alpha = p_1 \cup p_2$, then $p|_{\alpha \cdot 1} = p_1$ and $p|_{\alpha \cdot 2} = p_2$.
- If $p|_\alpha = p_1[p_2]$, then $p|_{\alpha \cdot 1} = p_1$ and $p|_{\alpha \cdot 2} = p_2$.

Next, we define the satisfaction relation of an XPath expression p by a tree t with a witness mapping w, which is a partial mapping from the set of positions of p to the set of pairs of nodes of t. Intuitively, $w(\alpha)$ is a pair of nodes that satisfies $p|_\alpha$. Note that $w(\alpha)|_1$ and $w(\alpha)|_2$ are the first and the second components of $w(\alpha)$, respectively:

- $t \models (\downarrow :: l)(w(\alpha))$ if $w(\alpha)|_2$ is a child of $w(\alpha)|_1$ and $\lambda_t(w(\alpha)|_2) = l$.

 ISBN: 978-1-61208-247-9

- $t \models (\uparrow :: l)(w(\alpha))$ if $w(\alpha)|_2$ is the parent of $w(\alpha)|_1$ and $\lambda_t(w(\alpha)|_2) = l$.
- $t \models (\downarrow^* :: l)(w(\alpha))$ if $w(\alpha)|_2$ is $w(\alpha)|_1$ or a descendant of $w(\alpha)|_1$, and $\lambda_t(w(\alpha)|_2) = l$.
- $t \models (\uparrow^* :: l)(w(\alpha))$ if $w(\alpha)|_2$ is $w(\alpha)|_1$ or an ancestor of $w(\alpha)|_1$, and $\lambda_t(w(\alpha)|_2) = l$.
- $t \models (\rightarrow^+ :: l)(w(\alpha))$ if $w(\alpha)|_2$ is a following sibling of $w(\alpha)|_1$ and $\lambda_t(w(\alpha)|_2) = l$.
- $t \models (\leftarrow^+ :: l)(w(\alpha))$ if $w(\alpha)|_2$ is a preceding sibling of $w(\alpha)|_1$ and $\lambda_t(w(\alpha)|_2) = l$.
- $t \models (p_1/p_2)(w(\alpha))$ if $t \models p_1(w(\alpha \cdot 1))$, $t \models p_2(w(\alpha \cdot 2))$, $w(\alpha \cdot 1)|_2 = w(\alpha \cdot 2)|_1$, and $w(\alpha) = (w(\alpha \cdot 1)|_1, w(\alpha \cdot 2)|_2)$.
- $t \models (p_1 \cup p_2)(w(\alpha))$ if $t \models p_i(w(\alpha \cdot i))$ and $w(\alpha) = w(\alpha \cdot i)$ for some $i \in \{1, 2\}$. Moreover, if $j \in \{1, 2\}$ does not satisfy $t \models p_j(w(\alpha \cdot j))$, then for any position α' whose prefix is $\alpha \cdot j$, $w(\alpha')$ is undefined.
- $t \models (p_1[p_2])(w(\alpha))$ if $t \models p_1(w(\alpha \cdot 1))$, $t \models p_2(w(\alpha \cdot 2))$, $w(\alpha \cdot 1)|_2 = w(\alpha \cdot 2)|_1$, and $w(\alpha) = w(\alpha \cdot 1)$.

If $t \models p(w(\epsilon))$, we say that t *satisfies* p *with witness* w and write $(t, w) \models p$. Note that if $(t, w) \models p$ and p does not include path union operator $\cup$, then w is a total mapping.

E. Typing problem

Suppose that an XML schema S, an XPath expression p without path union operator $\cup$, and a sequence $(\alpha_1, X_1), \ldots, (\alpha_k, X_k)$ of pairs of positions of p and types of S are given. The *typing problem* is to decide whether there exist $t \in TL(S)$, an interpretation I_t^S of t for S, and a mapping w such that

- $w(\epsilon)|_1 = \epsilon$ (i.e., the root node of t),
- $(t, w) \models p$, and
- $I_t^S(w(\alpha_i)|_2) = X_i$ for each i $(1 \leq i \leq k)$.

III. A Direct Approach

This section provides an algorithm which directly decides the typing problem. First, the algorithm translates a given XPath expression p into a finite tree automaton TA_p, maintaining the information on the structure of p as the states of TA_p. Then, the algorithm analyzes the correspondence between subexpressions of p and the types of a given schema S, by taking the intersection of TA_p and S.

A. Translating XPath expressions into finite tree automata

For a given XPath expression p without path union $\cup$, we construct a finite tree automaton TA_p with two distinguished sets NC_p and ND_p of states. Roughly speaking, TA_p accepts an arbitrary tree t satisfying p. States in NC_p and ND_p are associated with the "start" and "goal" nodes of t matching p.

First, we provide the definitions of $TA_p = (N_p, \Sigma, B_p, P_p, NC_p, ND_p)$ for $p = \chi :: l$, where $B_p = B$, $NC_p = \{C\}$, and $ND_p = \{D\}$.

- If $p = \downarrow :: l$, then P_p consists of
  - $A \to \sigma(A^*)$ for each $\sigma \in \Sigma$,
  - $B \to \sigma(A^* B A^*)$ for each $\sigma \in \Sigma$,
  - $B \to C$,
  - $C \to \sigma(A^* D A^*)$ for each $\sigma \in \Sigma$, and
  - $D \to l(A^*)$.
- If $p = \downarrow^* :: l$, then P_p consists of
  - $A \to \sigma(A^*)$ for each $\sigma \in \Sigma$,
  - $B \to \sigma(A^* B A^*)$ for each $\sigma \in \Sigma$,
  - $B \to C$,
  - $B' \to \sigma(A^* B' A^*)$ for each $\sigma \in \Sigma$,
  - $B' \to D$,
  - $C \to \sigma(A^* B' A^*)$ for each $\sigma \in \Sigma$,
  - $C \to D$, and
  - $D \to l(A^*)$.
- If $p = \rightarrow^+ :: l$, then P_p consists of
  - $A \to \sigma(A^*)$ for each $\sigma \in \Sigma$,
  - $B \to \sigma(A^* B A^*)$ for each $\sigma \in \Sigma$,
  - $B \to B'$,
  - $B' \to \sigma(A^* C A^* D A^*)$ for each $\sigma \in \Sigma$,
  - $C \to \sigma(A^*)$ for each $\sigma \in \Sigma$, and
  - $D \to l(A^*)$.

For axes $\uparrow$, $\uparrow^*$, and $\leftarrow^+$, the tree automata are defined by swapping states C and D (and their associated labels) for the ones of $\downarrow$, $\downarrow^*$, and $\rightarrow^+$, respectively.

Consider $TA_p = (N_p, \Sigma, B_p, P_p, NC_p, ND_p)$ where $p = p_1/p_2$. TA_p is defined as the intersection [9] of TA_{p_1} and TA_{p_2}, except that the states in ND_{p_1} overlaps only the states in NC_{p_2}, and vice versa. More precisely,

$$N_p = (ND_{p_1} \times NC_{p_2}) \cup ((N_{p_1} - ND_{p_1}) \times (N_{p_2} - NC_{p_2})).$$

Moreover, $B_p = (B_{p_1}, B_{p_2})$, $NC_p = NC_{p_1} \times N_{p_2}$, and $ND_p = N_{p_1} \times ND_{p_2}$. TA_p for $p = p_1[p_2]$ can be constructed in a similar way.

Lemma 1: TA_p satisfies Properties 1 and 2 below. Note that each state of TA_p has the same tree structure as p:

Property 1: For each mapping w such that $(t, w) \models p$, there is an interpretation $I_t^{TA_p}$ of t for TA_p such that for each position α of p, $I_t^{TA_p}(w(\alpha)|_1)|_\alpha$ contains a state in $NC_{p|_\alpha}$ and $I_t^{TA_p}(w(\alpha)|_2)|_\alpha$ contains a state in $ND_{p|_\alpha}$.

Property 2: Conversely, for each interpretation $I_t^{TA_p}$ of t for TA_p, there is a mapping w such that $(t, w) \models p$ and for each position α of p, $I_t^{TA_p}(w(\alpha)|_1)|_\alpha$ contains a state in $NC_{p|_\alpha}$ and $I_t^{TA_p}(w(\alpha)|_2)|_\alpha$ contains a state in $ND_{p|_\alpha}$.

Proof: The lemma is shown by the induction on the structure of p. The case where $p = \chi :: l$ is easy: If $(t, w) \models p$, then there is an interpretation $I_t^{TA_p}$ of t for TA_p such that $I_t^{TA_p}(w(\epsilon)|_1) = C$ and $I_t^{TA_p}(w(\epsilon)|_2) = D$. Conversely, if $I_t^{TA_p}$ is an interpretation of t for TA_p, then there are unique nodes v and v' of t such that $I_t^{TA_p}(v) = C$ and $I_t^{TA_p}(v') = D$. By letting $w(\epsilon) = (v, v')$, we have $(t, w) \models p$, $I_t^{TA_p}(w(\epsilon)|_1) = C$, and $I_t^{TA_p}(w(\epsilon)|_2) = D$.

 ISBN: 978-1-61208-247-9

Consider the case where $p = p_1/p_2$. Suppose that TA_{p_1} and TA_{p_2} satisfy the two properties and $(t, w) \models p_1/p_2$. Then, by the definition of $\models$, we have $t \models p_1(w(1))$, $t \models p_2(w(2))$, $w(1)|_2 = w(2)|_1$, and $w(\epsilon) = (w(1)|_1, w(2)|_2)$. Let $I_t^{TA_{p_1}}$ and $I_t^{TA_{p_2}}$ be interpretations of t for TA_{p_1} and TA_{p_2}, respectively, that satisfy Property 1. Define $I(v) = (I_t^{TA_{p_1}}(v), I_t^{TA_{p_2}}(v))$ for any node v of t. Then, by the definition of TA_p, I is an interpretation of t for TA_p and satisfies Property 1. Conversely, suppose that $I_t^{TA_p}$ is an interpretation of t for TA_p. Then, by the definition of TA_p, $I_t^{TA_p}|_1$ and $I_t^{TA_p}|_2$ are interpretations of t for TA_{p_1} and TA_{p_2}, respectively. Let w_1 and w_2 be the mappings determined by $I_t^{TA_p}|_1$ and $I_t^{TA_p}|_2$, respectively, which satisfy Property 2. Define $w(1 \cdot \alpha) = w_1(\alpha)$, $w(2 \cdot \alpha) = w_2(\alpha)$, and $w(\epsilon) = (w_1(\epsilon)|_1, w_2(\epsilon)|_2)$. Then $(t, w) \models p_1/p_2$ and Property 2 is satisfied.

The case where $p = p_1[p_2]$ can be shown similarly. ■

B. Analyzing correspondence between XPath expressions and schemas

Let $TA_{S \cap p}$ be an intersection automaton of an XML schema S and TA_p, except that the states in NC_p overlaps only the initial state of S, and vice versa. $TA_{S \cap p}$ accepts t if and only if t satisfies p at its root node and t conforms to S. Moreover, for each interpretation $I_t^{TA_{S \cap p}}$ of t for $TA_{S \cap p}$, $I_t^{TA_{S \cap p}}|_1$ is an interpretation of t for S and $I_t^{TA_{S \cap p}}|_2$ is an interpretation of t for TA_p. Since TA_p satisfies Property 2, it holds in turn that there is a mapping w such that $(t, w) \models p$ and for each position α of p, $I_t^{TA_{S \cap p}}|_2(w(\alpha)|_2)|_\alpha$ contains a state in $ND_{p|_\alpha}$. Conversely, by Property 1, if $(t, w) \models p$, then there is an interpretation $I_t^{TA_p}$ of t for TA_p such that for each position α of p, $I_t^{TA_p}(w(\alpha)|_2)|_\alpha$ contains a state in $ND_{p|_\alpha}$. It holds in turn that for any interpretation I_t^S of t for S , mapping I defined as $I(v) = (I_t^S(v), I_t^{TA_p}(v))$ is an interpretation of t for $TA_{S \cap p}$.

This observation naturally induces the following algorithm for deciding the typing problem. Suppose that an XML schema S, an XPath expression p, and a sequence $(\alpha_1, X_1), \ldots, (\alpha_k, X_k)$ of pairs of positions of p and types of S are given. For each i ($1 \leq i \leq k$), eliminate all the states (X, Y) (and associated rules) of $TA_{S \cap p}$ such that $X \neq X_i$ and $Y|_{\alpha_i} \in ND_{p|_{\alpha_i}}$. Then decide the emptiness of the tree language accepted by the resultant finite tree automaton. The answer of the typing problem is "yes" if and only if the tree language is not empty.

C. Discussion

It is easy to see that in the worst case, this algorithm runs at least in exponential time in the size of p because of the intersection operation on finite tree automata. However, this algorithm is expected to run reasonably fast in many cases. Indeed, finite tree automata can be translated into formulas in a variant of μ-calculus [10], and then the typing problem can be solved by fast decision procedures for μ-calculus formulas. An experimental analysis is left as future work.

IV. A Reduction-Based Approach

In this section, it is shown that the typing problem is reducible to XPath satisfiability in the presence of DTDs. Then, using this result, we provide a condition where all the possible combinations of types of atomic subexpressions can be efficiently enumerated.

Let p be an XPath expression without path union $\cup$. Then, for each subexpression $p|_\alpha$ of p, there is an atomic subexpression $p|_{\alpha'}$ of p such that for any t and w such that $(t, w) \models p$, we have $w(\alpha)|_2 = w(\alpha')|_2$. Therefore, in the rest of this section, we assume without loss of generality that all the given subexpressions of the typing problem are atomic.

A. Reduction to XPath satisfiability in the presence of DTDs

Let $S = (N, \Sigma, B, P)$ be an XML schema. Define a mapping φ as follows:

$$\begin{aligned} \varphi(S) &= (N, N \times \Sigma, B, \varphi(P)), \\ \varphi(P) &= \{X \to (X, a)(e) \mid X \to a(e) \in P\}. \end{aligned}$$

It is easy to show that $\varphi(S)$ is local.

For any $t \in TL(S)$ with an interpretation I_t^S, let $\varphi(t, I_t^S)$ be a tree such that $V_{\varphi(t, I_t^S)} = V_t$ and $\lambda_{\varphi(t, I_t^S)}(v) = (I_t^S(v), \lambda_t(v))$. It is easy to see that $\varphi(t, I_t^S) \in TL(\varphi(S))$. On the other hand, for any $t' \in TL(\varphi(S))$, let $\varphi^{-1}(t')$ denote the tree such that $V_{\varphi^{-1}(t')} = V_{t'}$ and $\lambda_{\varphi^{-1}(t')} = \lambda_{t'}|_2$. It is also easy to see that $\varphi^{-1}(t') \in TL(S)$ with interpretation $\lambda_{t'}|_1$.

Lemma 2: Let p be an XPath expression without path union $\cup$. Also, let p' be an XPath expression obtained by replacing each atomic subexpression $\chi_\alpha :: l_\alpha$ of p at α with $\chi_\alpha :: (X_\alpha, l_\alpha)$ for some $X_\alpha \in N$.

- Suppose that $t \in TL(S)$ with interpretation I_t^S and $(t, w) \models p$. Also suppose that for each position α of p such that $p|_\alpha$ is atomic, $I_t^S(w(\alpha)|_2) = X_\alpha$. Then, $(\varphi(t, I_t^S), w) \models p'$.
- Conversely, suppose that $t' \in TL(\varphi(S))$ and $(t', w') \models p'$. Then, $(\varphi^{-1}(t'), w') \models p$, and interpretation $\lambda_{t'}|_1$ of t for S satisfies that for each position α of p such that $p|_\alpha$ is atomic, $\lambda_{t'}|_1(w'(\alpha)|_2) = X_\alpha$.

Proof: The lemma is shown by induction on the structure of p. Consider the case where $p = \chi_\epsilon :: l_\epsilon$. Suppose that $t \in TL(S)$ with an interpretation I_t^S, $(t, w) \models p$, and $I_t^S(w(\epsilon)|_2) = X_\epsilon$. Since $p' = \chi_\epsilon :: (X_\epsilon, l_\epsilon)$, we have $\varphi(t, I_t^S) \models p'(w(\epsilon))$. Hence, the first condition holds. It is obvious that the second condition holds.

Consider the case where $p = p_1/p_2$. Suppose that $t \in TL(S)$ with an interpretation I_t^S and $(t, w) \models p_1/p_2$. Also suppose that for each position α of p such that $p|_\alpha$ is atomic, $I_t^S(w(\alpha)|_2) = X_\alpha$. Define two mappings w_1 and w_2 so that $w_1(\alpha) = w(1 \cdot \alpha)$ and $w_2(\alpha) = w(2 \cdot \alpha)$. Then, $(t, w_1) \models p_1$

and $(t, w_2) \models p_2$. By inductive hypothesis, $(\varphi(t), w_1) \models p'_1$ and $(\varphi(t), w_2) \models p'_2$, where $p' = p'_1/p'_2$. Hence $(\varphi(t), w) \models p'$. Conversely, suppose that $t' \in TL(\varphi(S))$ and $(t', w') \models p'_1/p'_2$. Define two mappings w'_1 and w'_2 so that $w'_1(\alpha) = w'(1 \cdot \alpha)$ and $w'_2(\alpha) = w'(2 \cdot \alpha)$. Then, $(t', w'_1) \models p'_1$ and $(t', w'_2) \models p'_2$. By inductive hypothesis, $(\varphi^{-1}(t'), w'_1) \models p_1$ and $(\varphi^{-1}(t'), w'_2) \models p_2$, and hence $(\varphi^{-1}(t'), w') \models p_1/p_2$. Moreover, interpretation $\lambda_{t'}|_1$ satisfies that for each position α of p_i such that $p_i|_\alpha$ is atomic, $\lambda_{t'}|_1(w'_i(\alpha)|_2) = X_\alpha$ ($i \in \{1, 2\}$). Hence the second condition holds.

The case where $p = p_1[p_2]$ can be shown similarly. ■

Theorem 1: The typing problem for an XPath class $\mathcal{X}$ with respect to an XML schema S is reducible in polynomial time to satisfiability for XPath class $\mathcal{X}$ plus path union in the presence of a DTD $\varphi(S)$.

Proof: Let $(\alpha_1, X_1), \ldots, (\alpha_k, X_k)$ be given pairs of positions of an XPath expression p and types of $S = (N, \Sigma, B, P)$, where $N = \{Y_1, \ldots, Y_n\}$. Let $\varphi(p)$ denote the XPath expression obtained by replacing each atomic subexpression $\chi_i :: l_i$ of p at α_i with $\chi_i :: (X_i, l_i)$, and other atomic ones $\chi :: l$ with $\chi :: (Y_1, l) \cup \cdots \cup \chi :: (Y_n, l)$.

Suppose that there exist $t \in TL(S)$, an interpretation I_t^S, and a mapping w such that $w(\epsilon)|_1 = \epsilon$, $(t, w) \models p$, and $I_t^S(w(\alpha_i)|_2) = X_i$ for each i ($1 \le i \le k$). Then, by Lemma 2, we have $(\varphi(t, I_t^S), w) \models \varphi(p)$.

Conversely, suppose that there exist $t' \in TL(\varphi(S))$ and a mapping w' such that $w'(\epsilon)|_1 = \epsilon$ and $(t', w') \models \varphi(p)$. Then, by Lemma 2 again, we have $(\varphi^{-1}(t'), w') \models p$ and the interpretation $\lambda_{t'}|_1$ of $\varphi^{-1}(t')$ for S satisfies that $\lambda_{t'}|_1(w'(\alpha_i)|_2) = X_i$ for each i ($1 \le i \le k$). ■

By Theorem 1, many known results on XPath satisfiability in the presence of DTDs can be used to realize tractable combinations of classes of XPath expressions and XML schemas. For example, from the result in [1], the typing problem for an XPath class consisting of downward axes with respect to an arbitrary XML schema S is tractable.

In what follows, we focus on the known results on *disjunction-capsuled DTDs* [3], or *DC-DTDs* for short, since satisfiability of wide XPath classes including path union $\cup$ is tractable under DC-DTDs. A regular expression e is *disjunction-capsuled*, or *DC* for short, if e is in the form of $e_1 e_2 \cdots e_n$ ($n \ge 1$), where each e_i ($1 \le i \le n$) is either

- a symbol in Σ, or
- in the form of $(e'_i)^*$ for a regular expression e'_i.

An XML schema $S = (N, \Sigma, B, P)$ is *disjunction-capsuled*, or *DC* for short, if for each transition rule $X \to a(e)$ in P, e is disjunction-capsuled. Immediately from the results in [3], we have the following corollary of Theorem 1:

Corollary 1: The typing problem for an XPath class $\mathcal{X}$ with respect to a DC XML schema S is tractable if

- $\mathcal{X}$ consists of $\downarrow$, $\downarrow^*$, $\rightarrow^+$, $\leftarrow^+$, and [ ]; or
- $\mathcal{X}$ consists of $\downarrow$, $\downarrow^*$, $\uparrow$, $\uparrow^*$, $\rightarrow^+$, and $\leftarrow^+$.

The known time complexities are $O(|p||S|^4)$ for the former case and $O(|p|^3|S|^3)$ for the latter case.

B. Efficient enumeration of types of subexpressions

The first case of Corollary 1 can be extended so that all the possible combinations of types of atomic subexpressions (precisely speaking, the witness mappings ξ introduced below) can be efficiently enumerated. To demonstrate this, we first briefly explain how satisfiability can be determined in this case. We introduce a *schema graph* of a given DC-DTD, which represents parent-child relationship as well as the possible positions of the children specified by the DC-DTD. We also define a satisfaction relation between schema graphs and XPath expressions. The satisfaction relation coincides the satisfiability under DC-DTDs and is decidable efficiently. In what follows, let $D = (N, B, P)$ be a DC-DTD and p be an XPath expression without upward axes. For each $A \in N$, let $P(A)$ denote the content model of the unique rule in P whose left-hand side is A. Moreover, for a DC regular expression $e = e_1 e_2 \cdots e_n$, let $len(e)$ denote the number n of subexpressions of the top-level concatenation.

Definition 1: The *schema graph* [3] $G = (U, E)$ of a DC-DTD $D = (N, B, P)$ is a directed graph defined as follows:

- A node $u \in U$ is either
  - $(\perp, 1, -, B)$, where $\perp$ is a new symbol not in N, or
  - (A, i, ω, A'), where A, $A' \in N$ and $1 \le i \le len(P(A))$ such that A' appears in the i-th subexpression e_i of $P(A)$, and $\omega =$ "$-$" if e_i is a single symbol in N and $\omega =$ "$*$" otherwise.

  The first, second, third and fourth components of u are denoted by $\lambda_{par}(u)$, $pos(u)$, $\omega(u)$, and $\lambda(u)$, respectively.
- An edge from u to u' exists in E if and only if $\lambda(u) = \lambda_{par}(u')$.

If $t \in TL(D)$, then each node of t can be associated with a node of the schema graph of D. More precisely, there exists a mapping θ, called an *SG mapping* of t, from the set of nodes of t to the set of nodes of the schema graph of D with the following properties:

- θ maps the root node of t to $(\perp, 1, -, B)$.
- Let v be a node of t with n children. Then, $\theta(v \cdot j) = (\lambda_t(v), i_j, \omega_{i_j}, \lambda_t(v \cdot j))$, where $1 \le i_j \le len(P(\lambda_t(v)))$, $\omega_{i_j} =$ "$-$" if the i_j-th subexpression of $P(\lambda_t(v))$ is a single symbol in N and $\omega_{i_j} =$ "$*$" otherwise, and $i_j \le i_{j'}$ if $j \le j'$. Moreover, for every maximum subsequence $(v \cdot j) \cdots (v \cdot j')$ such that $i_j = \cdots = i_{j'}$, the i_j-th subexpression of $P(\lambda_t(v))$ matches $\lambda_t((v \cdot j) \cdots (v \cdot j'))$.

A satisfaction relation $\models_{\mathrm{DC}}$ of an XPath expression p by a schema graph G with a witness mapping ξ, which is from the set of positions of p to the set of pairs of nodes of G, is defined as follows (some cases are omitted because of the space limitation):

- $G \models_{\mathrm{DC}} (\downarrow :: l)(\xi(\alpha))$ if there is an edge from $\xi(\alpha)|_1$ to $\xi(\alpha)|_2$ in G and $\lambda(\xi(\alpha)|_2) = l$.

- $G \models_{\mathrm{DC}} (\rightarrow^{+}:: l)(\xi(\alpha))$ if $\lambda_{par}(\xi(\alpha)|_1) = \lambda_{par}(\xi(\alpha)|_2)$, $\lambda(\xi(\alpha)|_2) = l$, and $pos(\xi(\alpha)|_1) < pos(\xi(\alpha)|_2)$ if $\omega(\xi(\alpha)|_1) =$ "$-$" and $pos(\xi(\alpha)|_1) \leq pos(\xi(\alpha)|_2)$ if $\omega(\xi(\alpha)|_1) =$ "$*$".
- $G \models_{\mathrm{DC}} (p_1/p_2)(\xi(\alpha))$ if $G \models_{\mathrm{DC}} p_1(\xi(\alpha \cdot 1))$, $G \models_{\mathrm{DC}} p_2(\xi(\alpha \cdot 2))$, $\xi(\alpha \cdot 1)|_2 = \xi(\alpha \cdot 2)|_1$, and $\xi(\alpha) = (\xi(\alpha \cdot 1)|_1, \xi(\alpha \cdot 2)|_2)$.
- $G \models_{\mathrm{DC}} (p_1 \cup p_2)(\xi(\alpha))$ if $G \models_{\mathrm{DC}} p_i(\xi(\alpha \cdot i))$ and $\xi(\alpha) = \xi(\alpha \cdot i)$ for some $i \in \{1, 2\}$. Moreover, if $j \in \{1, 2\}$ does not satisfy $G \models_{\mathrm{DC}} p_j(\xi(\alpha \cdot j))$, then for any position α' whose prefix is $\alpha \cdot j$, $\xi(\alpha')$ is undefined.

If $G \models_{\mathrm{DC}} p(\xi(\epsilon))$, we say that G *satisfies* p with witness ξ and write $(G, \xi) \models_{\mathrm{DC}} p$.

Theorems 3 and 4 in [3] imply the following theorem:

Theorem 2: If $(t, w) \models p$ with an SG mapping θ, then $(G, \theta \circ w) \models_{\mathrm{DC}} p$. Conversely, if $(G, \xi) \models_{\mathrm{DC}} p$, then there is an SG mapping θ such that $\xi = \theta \circ w$ and $(t, w) \models p$.

Hence, ξ such that $(G, \xi) \models_{\mathrm{DC}} p$ has enough information to give one possible combination of the types of atomic subexpressions of p.

Let S be a given DC XML schema and p be a given XPath expression without path union and upward axes. Let $\varphi'(p)$ denote the expression obtained by replacing each atomic subexpression $\chi :: l$ of p with $\chi :: (Y_1, l) \cup \cdots \cup \chi :: (Y_n, l)$, where $Y_1, \ldots, Y_n$ are all the states of S. Now, the enumeration algorithm is as follows. First, construct the schema graph G of DC-DTD $D = \varphi(S)$. Then, compute the set $\Xi(\alpha)$ of all the pairs (u, u') such that $G \models_{\mathrm{DC}} \varphi'(p)|_\alpha(u, u')$ for each position α of p, in a bottom-up manner with respect to the structure of p. Finally, by traversing Ξ in a top-down manner with respect to α, construct each ξ such that $(G, \xi) \models_{\mathrm{DC}} p$. Each ξ can be enumerated with worst-case delay $O(|p||S|^4)$ time.

V. Conclusion

This paper has discussed typing XPath subexpressions with respect to an XML schema. An algorithm which directly decides the typing problem has been proposed. Moreover, it has been shown that the typing problem is reducible to the XPath satisfiability problem in the presence of DTDs, for which many tractability results are known.

In the definition of the typing problem, we have excluded the path union operator $\cup$ from XPath expressions. Actually, we have found that handling path union is a challenging task. For example, consider an XPath expression $p_1 \cup p_2$. If (ϵ, X) is specified, then we have to check whether X is associated with p_1 *or* X is associated with p_2. On the other hand, if $(1, Y)$ and $(2, Z)$ are specified, then we have to check whether Y is associated with p_1 *and* Z is associated with p_2 simultaneously. In this sense, the meaning of $\cup$ changes depending on the specified pairs of positions and types. Now we are trying to incorporate path union operator and to find a wider condition where the typing problem is solvable efficiently. However, we are also conjecturing that polynomial-time reduction to XPath satisfiability in the presence of DTDs is possible only if atomic subexpressions are specified in the typing problem. It is also interesting to investigate whether efficient enumeration is possible in the second case of Corollary 1.

Acknowledgment

The authors thank the anonymous reviewers for their helpful comments to improve the paper. This research is supported in part by Grant-in-Aid for Scientific Research (C) 23500120 from Japan Society for the Promotion of Science.

References

[1] M. Benedikt, W. Fan, and F. Geerts, "XPath satisfiability in the presence of DTDs," *Journal of the ACM*, vol. 55, no. 2, 2008.

[2] M. Montazerian, P. T. Wood, and S. R. Mousavi, "XPath query satisfiability is in PTIME for real-world DTDs," in *Proceedings of the 5th International XML Database Symposium, LNCS 4704*, 2007, pp. 17–30.

[3] Y. Ishihara, T. Morimoto, S. Shimizu, K. Hashimoto, and T. Fujiwara, "A tractable subclass of DTDs for XPath satisfiability with sibling axes," in *Proceedings of the 12th International Symposium on Database Programming Languages*, 2009, pp. 68–83.

[4] P. T. Wood, "Containment for XPath fragments under DTD constraints," in *Proceedings of the 9th International Conference on Database Theory*, 2003, pp. 297–311.

[5] G. Miklau and D. Suciu, "Containment and equivalence for a fragment of XPath," *Journal of the ACM*, vol. 51, no. 1, pp. 2–45, 2004.

[6] F. Neven and T. Schwentick, "On the complexity of XPath containment in the presence of disjunction, DTDs, and variables," *Logical Methods in Computer Science*, vol. 2, no. 3, 2006.

[7] H. Björklund, W. Martens, and T. Schwentick, "Optimizing conjunctive queries over trees using schema information," in *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science*, 2008, pp. 132–143.

[8] K. Hashimoto, Y. Kusunoki, Y. Ishihara, and T. Fujiwara, "Validity of positive XPath queries with wildcard in the presence of DTDs," in *The 13th International Symposium on Database Programming Languages*, 2011. [Online]. Available: http://www.cs.cornell.edu/conferences/dbpl2011/papers/dbpl11-hashimoto.pdf

[9] M. Murata, D. Lee, M. Mani, and K. Kawaguchi, "Taxonomy of XML schema languages using formal language theory," *ACM Transactions on Internet Technology*, vol. 5, no. 4, pp. 660–704, 2005.

[10] P. Genevès, N. Layaïda, and A. Schmitt, "Efficient static analysis of XML paths and types," in *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation*, 2007, pp. 342–351.

 ISBN: 978-1-61208-247-9