IEICE TRANS. INF. & SYST., VOL.E92–D, NO.4 APRIL 2009

PAPER

An XML Transformation Algorithm Inferred from an Update Script between DTDs

Nobutaka SUZUKI[†a)], ***Member and*** **Yuji FUKUSHIMA**[†], ***Nonmember***

SUMMARY Finding an appropriate data transformation between two schemas has been an important problem. In this paper, assuming that an update script between original and updated DTDs is available, we consider inferring a transformation algorithm from the original DTD and the update script such that the algorithm transforms each document valid against the original DTD into a document valid against the updated DTD. We first show a transformation algorithm inferred from a DTD and an update script. We next show a sufficient condition under which the transformation algorithm inferred from a DTD d and an update script is unambiguous, i.e., for any document t valid against d, elements to be deleted/inserted can unambiguously be determined. Finally, we show a polynomial-time algorithm for testing the sufficient condition.

key words: *XML, data transformation, schema evolution*

1. Introduction

Suppose that we maintain XML documents valid against a DTD. If the DTD is updated, then we have to transform each of the documents into a valid one against the updated DTD. Transforming each document manually is surely impractical, so constructing an appropriate transformation algorithm between original and updated DTDs is a very important problem.

In this paper, we propose a novel transformation approach based on an update script between original and updated DTDs; assuming that the update script applied to a DTD is known, we construct a transformation algorithm "inferred" from the DTD and the update script. Here, an update script to a DTD is a sequence of update operations, where each update operation inserts/deletes an element or operator in a content model of the DTD.

For example, let us consider DTD d_1 shown in Fig. 1 (a). Suppose that d_1 is updated to a new DTD d_2 by an update script that (i) deletes "age" and (ii) aggregates a subsequence "(address, zip, country)" of the content model of "staff" into "addr_info" (Fig. 1 (b)). Then for any XML document t valid against d_1, the transformation algorithm inferred from d_1 and the update script

1. deletes the "age" element in t, and
2. inserts a new "addr_info" element into t as the parent of "address", "zip", and "country" elements.

For example, the XML document t_1 in Fig. 1 (c) (represented as a tree without text strings) is transformed into t_2 in Fig. 1 (d), which is valid against d_2.

Let d be a DTD and S be a set of XML documents valid against d. Suppose that a user updated d to a new DTD by applying some update script s to d. Since s concretely represents how the user intends to modify d, s strongly suggests how to transform each document in S. Therefore, if we can obtain a transformation algorithm T inferred from d and s as shown above, then we can say that T is a transformation algorithm that faithfully reflects the user's intention represented by s.

However, depending on a DTD d and an update script s to d, the transformation algorithm T inferred from d and s may become "ambiguous", that is, for some document t valid against d T cannot unambiguously determine which elements in t should be deleted/inserted (conversely, if there is no such tree, then T is called "unambiguous"). For example, let us consider DTD d_3 (Fig. 2 (a)). Suppose that d_3 is updated to a new DTD d_4 by an update script that aggregates subexpression "(section,section*,ack?)" of the content model of "book" into "chapter" (Fig. 2 (b)). For the tree t_3 in Fig. 2 (c), we have two alternatives t_4, t_5 according to the positions at which "chapter" elements should be inserted (Fig. 2 (d,e)). Thus T is ambiguous (T outputs one of t_4 and t_5 arbitrarily). In general, an ambiguous transformation algorithm is undesirable since it may delete elements that should not be deleted and may insert elements at unexpected positions. Therefore, for a DTD d and an update script s, we should be able to decide if the transformation algorithm inferred from d and s is unambiguous.

In this paper, we first define update operations to DTDs. Then, based on the update operations we show a (possibly ambiguous) transformation algorithm inferred from a DTD and an update script. Then we show sufficient conditions under which the transformation algorithm inferred from a DTD and an update script is unambiguous. Finally, we show a polynomial-time algorithm for determining if, given a DTD d and an update script s, the transformation algorithm inferred from d and s satisfies the sufficient conditions.

Related Work

Schema matching, query discovery, and other related problems have been extensively studied, e.g., [1], [2], [11], [13]–[16], [20]. These studies except [20] consider finding an ap-

Manuscript received February 26, 2008.
Manuscript revised October 2, 2008.

[†]The authors are with the Graduate School of Library, Information and Media Studies, University of Tsukuba, Tsukuba-shi, 305–8550 Japan.
a) E-mail: nsuzuki@slis.tsukuba.ac.jp

DOI: 10.1587/transinf.E92.D.594

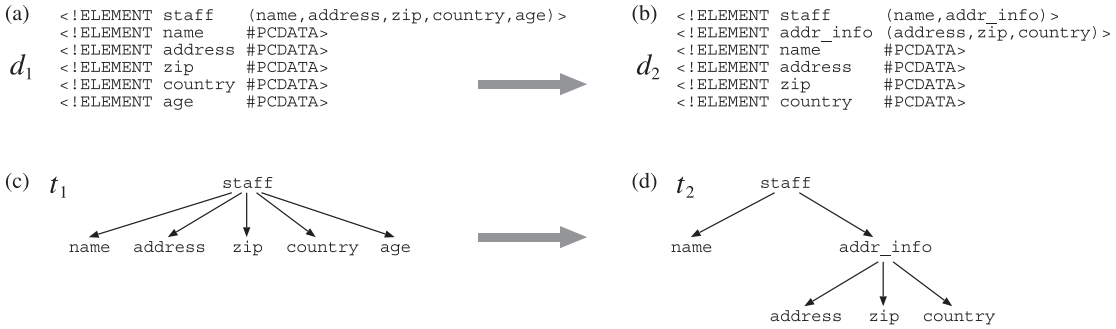

Fig. 1 DTDs d_1, d_2 and XML documents t_1, t_2.

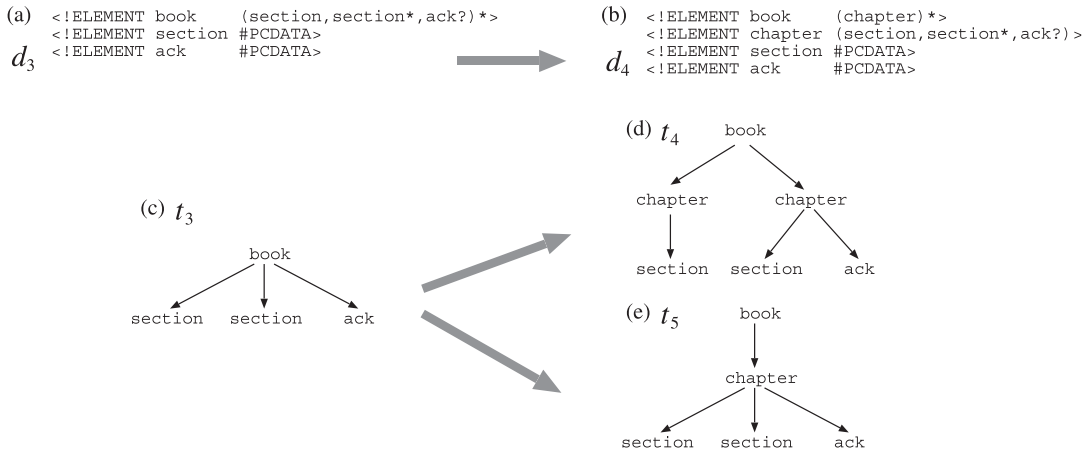

Fig. 2 DTDs d_3, d_4 and XML documents t_3, t_4, t_5.

propriate matching or transformation between schemas, assuming that no update script between the schemas is known. Ref. [20] considers the problem of adapting mappings to schema changes. The study assumes that a mapping between (nested relational) schemas are explicitly provided, thus the ambiguities discussed in this paper do not arise.

Several studies propose update operations to schemas. Ref. [12] proposes update operations to represent the "diff" between two DTDs. Ref. [10] proposes update operations to tree grammars to preserve schema's expressive power; any updated grammar admits only trees to which trees valid against its original grammar are embeddable. Refs. [9], [19] propose update operations assuring that any updated schema includes its original schema.

2. Definitions

An XML document is modeled as an ordered labeled tree (attributes are omitted). Each node in a tree represents an element. A text node is omitted, in other words, we assume that each leaf node has an implicit text node. By $l(n)$ we mean the label (element name) of node n. In what follows, we use the term tree when we mean ordered labeled tree.

Let Σ be a set of labels. In order to define update operations to a DTD concisely, each regular expression is represented as a term in prefix notation. Formally, a *regular expression* over Σ is recursively defined as follows.

- ϵ and a are regular expressions, where $a \in \Sigma$.
- If $r_1, \cdots, r_n$ are regular expressions, then $\cdot(r_1, \cdots, r_n)$ and $+(r_1, \cdots, r_n)$ are regular expressions ($n \geq 1$).
- If r_1 is a regular expression, then $*(r_1)$ is a regular expression.

For example, we write $\cdot(a, *(+(b, c)))$ instead of usual notation $a(b+c)^*$. The language specified by a regular expression r is denoted $L(r)$.

Let r be a regular expression. The set of *positions* of r, denoted $pos(r)$, is defined as follows.

- If $r = \epsilon$ or $r = a$ for some $a \in \Sigma$, then $pos(r) = \{\lambda\}$, where λ denotes an empty sequence.
- If $r = op(r_1, \cdots, r_n)$ with $op \in \{+, \cdot, *\}$, then $pos(r) = \{\lambda\} \cup \{u \mid u = iv, 1 \leq i \leq n, v \in pos(r_i)\}$, where $n = 1$ if $op =$ '$*$'.

For example, let $r = \cdot(+(a, b, c), *(d))$. Figure 3 shows the tree representation of r, in which each node is associated with its corresponding position. Thus $pos(r) = \{\lambda, 1, 2, 11, 12, 13, 21\}$.

Let $u \in pos(r)$. The label at u in r, denoted $l(r, u)$, and the subexpression at u in r, denoted $sub(r, u)$, are recursively defined as follows.

- If $r = \epsilon$ or $r = a$ for some $a \in \Sigma$, then $l(r, \lambda) = r$ and

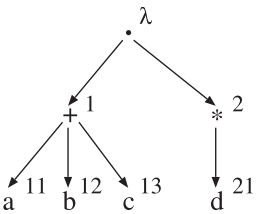

Fig. 3 Tree representation of $\cdot(+(a,b,c),*(d))$.

$sub(r,\lambda) = r$.

- If $r = op(r_1,\cdots,r_n)$ with $op \in \{+,\cdot,*\}$, and
  - if $u = \lambda$, then $l(r,u) = op$ and $sub(r,u) = r$,
  - if $u = jv$ for some $1 \le j \le n$ and some $v \in pos(r_j)$, then $l(r,u) = l(r_j,v)$ and $sub(r,u) = sub(r_j,v)$,

  where $n = 1$ if $op =$ '$*$'.

For example, in Fig. 3 $l(r,1) =$ '+', $l(r,21) = d$, and $sub(r,1) = +(a,b,c)$.

Let w be a word over Σ. By $|w|$ we mean the length of w, and by $w[i]$ we mean the ith label of w. We define that $w[i,j] = w[i]w[i+1]\cdots w[j]$ $(1 \le i \le j \le |w|)$. For example, if $w = tsukuba$, then $w[2,5] = suku$.

A *DTD* d is a (possibly partial) mapping from Σ to the set of regular expressions over Σ. For a label $a \in \Sigma$, $d(a)$ is the *content model* of a. A tree t is *valid* against d if for each node n in t the sequence of labels on the children of n is in $L(d(l(n)))$. For labels $a_i, a_j \in \Sigma$, a_j is *reachable* from a_i if (i) $a_i = a_j$ or (ii) a_j occurs in $d_1(a_k)$ for some label a_k reachable from a_i.

Let r be a regular expression. By r' we mean the *superscripted* regular expression resulting from r by superscripting each label in r by its corresponding position. By $sym(r')$ we mean the set of superscripted labels occurring in r'. For example, if $r = \cdot(+(a,b,c),*(+(d,b)))$, then $r' = \cdot(+(a^{11},b^{12},c^{13}),*(+(d^{211},b^{212})))$ and $sym(r') = \{a^{11},b^{12},c^{13},d^{211},b^{212}\}$. Let a^i be a superscripted label of a. Then by $(a^i)^\natural$ we mean the label resulting from a^i by dropping the superscript of a^i, that is, $(a^i)^\natural = a$. Let w' be a superscripted word (i.e., a sequence of superscripted labels). We define that $(w')^\natural = w'[1]^\natural\cdots w'[|w'|]^\natural$. For any regular expression r, it holds that $L(r) = L(r')^\natural$, where $L(r')^\natural = \{(w')^\natural \mid w' \in L(r')\}$.

3. Update Operations to DTD

In this section, we define update operations to DTD d. There are two types of update operations; the update operations of type 1 relate to modifying labels in a content model, and the update operations of type 2 relate to modifying operators in a content model. Let $a \in \Sigma$ be a label and $u \in pos(d(a))$ be a position in $d(a)$.

Type 1a: Inserting/deleting an element in a content model.

- *ins_elm*(a,b,vi): Inserts a new label b at vi in $d(a)$, where $vi \in pos(d(a))$ with i is a positive integer and $b \in \Sigma \cup \{\epsilon\}$ (Fig. 4 (b,c)). This is applicable to d only if $d(b)$ is defined, $l(d(a),v) \in \{+,\cdot\}$, and $v(i-1) \in pos(d(a))$ (i.e., the operator at v has at least $i-1$ operands).
- *del_elm*(a,vi): Deletes the label/ϵ at vi in $d(a)$. More formally, we have two cases according to the operator at v.
  - The case where $l(d(a),v) =$ '+': If $l(d(a),vi) = l(d(a),vk)$ for some $k \ne i$, then $l(d(a),vi)$ is deleted from $d(a)$ (deleting one of duplicated labels). Otherwise, $l(d(a),vi)$ is replaced by ϵ.
  - The case where $l(d(a),v) =$ '$\cdot$': In this case, $l(d(a),vi)$ is deleted from $d(a)$ (Fig. 4 (a,b)). This is applicable to d only if $vk \in pos(d(a))$ for some $k \ne i$ (i.e., $l(d(a),vi)$ has at least one siblings).

Type 1b: Extracting a label in $d(a)$ and aggregating a subexpression of $d(a)$ into a new label.

- *ext_elm*(a,u): Extracts a label $l(d(a),u)$ in $d(a)$. Formally, this operation replaces label $l(d(a),u)$ in $d(a)$ by regular expression $d(l(d(a),u))$ (Fig. 4 (e,f)). This is applicable to d only if $sub(d(a),u)$ is a label in Σ and $a \ne l(d(a),u)$.
- *agg_elm*(a,b,u): Aggregates subexpression $sub(d(a),u)$ into single label b. Formally, this operation (i) sets $d(b) = sub(d(a),u)$ and (ii) replaces $sub(d(a),u)$ by label b (Fig. 4 (d,e)). This is applicable to d only if $d(b)$ is undefined.

Type 2: Inserting/deleting an operator ('+', '$\cdot$', or '$*$') in $d(a)$.

- *ins_opr*(a,opr,vi,vj): Inserts a new operator opr as the parent of the sibling subexpressions at $vi,\cdots,vj$ in $d(a)$, where $opr \in \{+,\cdot,*\}$ (Fig. 4 (c,d)). This is applicable to d only if (i) $i = j$ (opr has only one operand) or (ii) $i < j$, $opr \in \{+,\cdot\}$, and $opr = l(d(a),v)$ (nesting the operator at v by opr).
- *del_opr*(a,vi): Deletes an operator at vi in $d(a)$ (Fig. 4 (f,g)). This is applicable to d only if (i) $l(d(a),v) = l(d(a),vi)$ (unnesting the operator at vi) or (ii) the operator at vi has only one operand.

Let op be an update operation to a DTD d. By $op(d)$ we mean the DTD obtained by applying op to d. Let $s = op_1op_2\cdots op_n$ be a sequence of update operations $(n \ge 0)$. s is *applicable* to d if op_i is applicable to $op_{i-1}(op_{i-2}(\cdots op_1(d)\cdots))$ for every $1 \le i \le n$. We say that s is an *update script* to d if s is applicable to d. For an update script $s = op_1op_2\cdots op_n$ to d, we define that $s(d) = op_n(op_{n-1}(\cdots op_1(d)\cdots))$. An update script of length zero is denoted ϵ, where ϵ is an *identity operator* such that $\epsilon(d) = d$ for any DTD d.

Example 1: Let d be a DTD, where d(staff) =

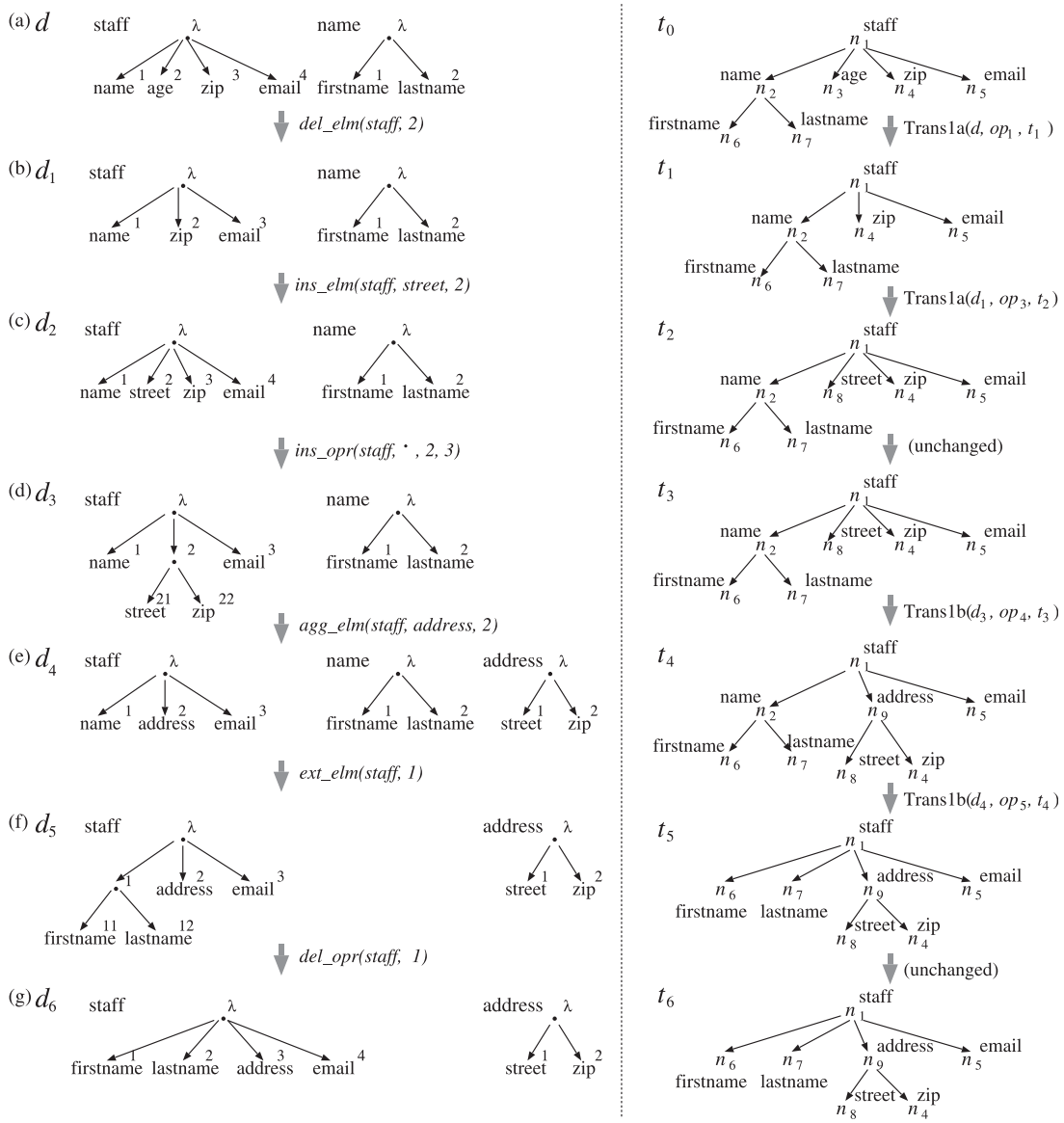

Fig. 4 An update script s to a DTD d and a transformation according to s and d.

·(name,age,zip,email), d(name) = ·(firstname,lastname), and $d(l) = \epsilon$ for any label l other than "staff" and "name". Let $s = op_1 \cdots op_6$, where $op_1 = del\_elm(\text{staff}, 2)$, $op_2 = ins\_elm(\text{staff}, \text{street}, 2)$, $op_3 = ins\_opr(\text{staff}, \cdot, 2, 3)$, $op_4 = agg\_elm(\text{staff}, \text{address}, 2)$, $op_5 = ext\_elm(\text{staff}, 1)$, and $op_6 = del\_opr(\text{staff}, 1)$. Then d and $d_i = op_i(op_{i-1}(\cdots(op_1(d))\cdots))$ for $1 \le i \le 6$ are illustrated in Fig. 4 (left). □

We say that a DTD d_2 *includes* a DTD d_1 if for any tree t, t is valid against d_2 whenever t is valid against d_1. We have the following lemma.

Lemma 1: Let d be a DTD and op be an update operation to d. Then $op(d)$ includes d if op satisfies one of the following conditions.

1. $op = ins\_elm(a, b, vi)$ and either (i) $l(d(a), v) =$ '+' or (ii) $l(d(a), v) =$ '·' and $b = \epsilon$.
2. $op = del\_elm(a, vi)$ and either (i) $l(d(a), v) =$ '·' and $l(d(a), vi) = \epsilon$ or (ii) $l(d(a), v) =$ '+' and $l(d(a), vi) \in L(sub(d(a), vk))$ for some $k \neq i$ (i.e., the element at vi is contained in some sibling).
3. $op = ext\_elm(a, u)$ and $l(d(a), u) \in L(d(l(d(a), u)))$.
4. $op = ins\_opr(a, opr, vi, vj)$, where $opr \in \{+, \cdot, *\}$.
5. $op = del\_opr(a, u)$ and either (i) $l(d(a), u) \in \{+, \cdot\}$ or (ii) $l(d(a), u) =$ '$*$' and $L(sub(d(a), u1)) = \{\epsilon\}$.

□

For example, in Fig. 4 $ins\_opr(\text{staff}, \cdot, 2, 3)$ satisfies Condition (4) and $del\_opr(\text{staff}, 1)$ satisfies Condition (5) of the above lemma.

4. Transformation Algorithm Inferred from DTD and Update Script

In this section, we show a (possibly ambiguous) transforma-

tion algorithm inferred from a DTD and an update script.

4.1 Outline

We first show an outline of our transformation algorithm. Let d be a DTD and op be an update operation to d. For a tree t valid against d, our transformation algorithm T inferred from d and op transforms t as follows.

1. If t is valid against $op(d)$, then T does nothing.
2. Otherwise, T modifies t according to the type of op.

Type 1a: (1) If $op = ins\_elm(a, b, u)$, then b is inserted at u in $d(a)$. Accordingly, for each position p in t at which the b-label should be inserted, T creates a new valid tree[†] whose root is labeled by b and insert the tree at position p in t. For example, if $d(a) = \cdot(a, b)$ and $op = ins\_elm(a, c, 3)$, then for each node n in t labeled by a, a new valid tree whose root is labeled by c is inserted as the third child of n.
(2) If $op = del\_elm(a, u)$, then the label, say b, at u in $d(a)$ is deleted or replaced by ϵ. Accordingly, T first identifies the subtrees in t whose roots match the label b, then T deletes the identifies subtrees.

Type 1b: (1) If $op = ext\_elm(a, u)$, then T identifies the internal nodes in t that match the extracted label in $d(a)$, and deletes the identified internal nodes from t. For example, if $d(a) = \cdot(b, c)$ and $op = ext\_elm(a, 1)$, then each internal node in t that matches b is deleted.
(2) If $op = agg\_elm(a, b, u)$, then T inserts new b-labeled nodes into t as the parents of sibling nodes that should be aggregated. For example, if $d(a) = +(\cdot(b, c), e)$ and $op = agg\_elm(a, f, 1)$, then for each pair of siblings labeled by b and c in t a new node labeled by f is inserted as the parent of the siblings.

Type 2: By Conditions (4) and (5) of Lemma 1, $op = del\_opr(a, u)$ and $sub(d(a), u) = *(q)$ for some regular expression q (op deletes the '$*$' from $*(q)$). Thus T identifies the nodes in t that match $*(q)$ and deletes "excess" subtrees from the subtrees rooted at the identified nodes (since $d(a)$ admits arbitrary repetitions of q but $op(d)(a)$ does not) and supplements "missing" trees to t (since $*(q)$ matches ϵ but q may not).

Let d be a DTD and $s = op_1 \cdots op_n$ be an update script to d. The transformation algorithm inferred from d and s applies $T_1, \cdots, T_n$ to t, where T_i is the transformation algorithm inferred from d_{i-1} and op_i and $d_{i-1} = op_{i-1}(\cdots(op_1(d))\cdots)$.

4.2 The Transformation Algorithm

We first show some definitions. Let r be a regular expression, $u \in pos(r)$ be a position, $q = sub(r, u)$ be a subexpression at u of r, w be a word such that $w \in L(r)$, and w' be a superscripted word such that $w' \in L(r')$ and that $(w')^\natural = w$. We say that $w'[i, j]$ *maximally matches* q' if $w'[i, j] \in L(q')$ and either (i) $i = 1$ and $j = |w'|$ or (ii) $w'[i', j'] \notin L(q')$ for any i', j' with $\{i, \cdots, j\} \subset \{i', \cdots, j'\}$. We define that

$$match(w', q') = \{(i, j) \mid w'[i, j] \text{ maximally matches } q'\}.$$

For example, let $r = *(\cdot(a, +(b, c)))$ and $q = sub(r, 12)$. Then $r' = *(\cdot(a^{11}, +(b^{121}, c^{122})))$ and $q' = sub(r, 12)' = +(b^{121}, c^{122})$. If $w' = a^{11}b^{121}a^{11}c^{122}$, then $match(w', q') = \{(2, 2), (4, 4)\}$.

Let w be a word and b^h be a superscripted label. We say that a superscripted word w' is a *superscripted supersequence* of w w.r.t. b^h if removing every b^h from w' yields a word w'' such that $(w'')^\natural = w$.

Let d be a DTD and op be an update operation to d. We show the transformation algorithm inferred from d and op, denoted $\textsc{TransOp}_{d,op}$ as follows (subroutine $\textsc{Trans1a}$, $\textsc{Trans1b}$, and $\textsc{Trans2}$ are shown later).

$\textsc{TransOp}_{d,op}(t)$
Input: a tree t valid against d.
Output: a tree valid against $op(d)$.

1. If t is valid against $op(d)$, then return t.
2. Otherwise, do the following.
   a. If op is of type 1a, then return $\textsc{Trans1a}_{d,op}(t)$.
   b. If op is of type 1b, then return $\textsc{Trans1b}_{d,op}(t)$.
   c. If op is of type 2, then return $\textsc{Trans2}_{d,op}(t)$.

Let us show three subroutines $\textsc{Trans1a}$, $\textsc{Trans1b}$, and $\textsc{Trans2}$. We first show $\textsc{Trans1a}$.

$\textsc{Trans1a}_{d,op}(t)$

1. If $op = ins\_elm(a, b, vi)$, then for each node n labeled by a in t, do the following. Note that by Condition (1) of Lemma 1, $l(d(a), v) =$ '$\cdot$'.
   a. Let $n_1, \cdots, n_m$ be the children of n in t. Find a superscripted supersequence w' of $l(n_1)\cdots l(n_m)$ w.r.t. b^h such that $w' \in L((op(d)(a))')$, where b^h is the superscripted label in $op(d)(a)$ inserted by op.
   b. For each $(j, j) \in match(w', b^h)$, create a new tree valid against $op(d)$ whose root is labeled by b and insert the tree into t as the jth child of n.
2. If $op = del\_elm(a, vi)$, then for each node n labeled by a in t, do the following.
   a. Let $n_1, \cdots, n_m$ be the children of n in t. Find a superscripted word w' such that $w' \in L(d(a)')$ and that $(w')^\natural = l(n_1)\cdots l(n_m)$.
   b. By definition $sub(d(a), vi)'$ is a single superscripted label, say b^h. For each $(j, j) \in match(w', b^h)$, delete the subtree rooted at n_j from t.

[†]We assume that the text values of such a new tree are empty since they can hardly be estimated.

3. Return t transformed above.

In step (1a) we have to find a superscripted supersequence w' of $l(n_1)\cdots l(n_m)$ w.r.t. b^h such that $w' \in L((op(d)(a))')$, which can be obtained in $O(|op(d)(a)|^2 + |w'|)$ time, where $|op(d)(a)| = |d(a)| + 1$ (details are shown in Appendix A). In step (1b), a new tree can be constructed in $O(|d|)$ time. In step (2a), a superscripted word w' such that $w' \in L(d(a)')$ and that $(w')^\natural = l(n_1)\cdots l(n_m)$ can easily be obtained by using the Glushkov automaton of $d(a)$ (defined in Sect. 6.2), which requires $O(|d(a)|^2 + |w'|)$ time. Thus $\textsc{Trans1a}_{d,op}(t)$ runs in $O(|t| \cdot |d(a)|^2 + |t| \cdot |d|)$ time.

Example 2: Figure 4 (right) illustrates how tree t_0 is transformed by the transformation algorithm inferred from d and s, where d and s are given in Example 1 and Fig. 4 (left). Let us consider the intermediate transformations from t_0 to t_1 and t_1 to t_2 in Fig. 4 (right).

($t_0 \Rightarrow t_1$) Let d be the DTD in Fig. 4 (a). Then $d(\text{staff}) = \cdot(\text{name}, \text{age}, \text{zip}, \text{email})$. Since $op_1 = del\_elm(\text{staff}, 2)$, t_0 is transformed by step 2 of Trans1a. Consider the node n_1 of t_0. Since $d(\text{staff})' = \cdot(\text{name}^1, \text{age}^2, \text{zip}^3, \text{email}^4)$, the superscripted word w' of $l(n_2)l(n_3)l(n_4)l(n_5)$ = "name age zip email" such that $w' \in d(\text{staff})'$ is "$\text{name}^1\,\text{age}^2\,\text{zip}^3\,\text{email}^4$". Thus $match(w', \text{age}^2) = \{(2, 2)\}$, thereby the second child n_3 of n_1 is deleted from t_0.

($t_1 \Rightarrow t_2$) Let d_1 be the DTD in Fig. 4 (b). Then $d_1(\text{staff}) = \cdot(\text{name}, \text{zip}, \text{email})$. Since $op_2 = ins\_elm(\text{staff}, \text{street}, 2)$, we have $d_2(\text{staff}) = \cdot(\text{name}, \text{street}, \text{zip}, \text{email})$ and t_1 is transformed by step 1 of Trans1a. Consider the node n_1 in t_1. Since $d_2(\text{staff})' = \cdot(\text{name}^1, \text{street}^2, \text{zip}^3, \text{email}^4)$, the superscripted supersequence w' of $l(n_2)l(n_4)l(n_5)$ = "name zip email" w.r.t. street^2 such that $w' \in L(d_2(\text{staff})')$ is "$\text{name}^1\,\text{street}^2\,\text{zip}^3\,\text{email}^4$". Thus $match(w', \text{street}^2) = \{(2, 2)\}$, and a new node n_8 labeled by "street" is inserted into t_1 as the second child of n_1. □

We next show Trans1b.

$\textsc{Trans1b}_{d,op}(t)$

1. If $op = ext\_elm(a, u)$, then for each node n labeled by a in t, do the following.
   a. Let $n_1, \cdots, n_m$ be the children of n in t. Find a superscripted word w' such that $w' \in L(d(a)')$ and that $(w')^\natural = l(n_1)\cdots l(n_m)$.
   b. By definition $sub(d(a), u)'$ is a single superscripted label, say b^h. For each $(j, j) \in match(w', b^h)$, delete the jth child n_j of n from t.
2. If $op = agg\_elm(a, b, u)$, then for each node n labeled by a, do the following.
   a. Let $n_1, \cdots, n_m$ be the children of n in t. Find a superscripted word w' such that $w' \in L(d(a)')$ and that $(w')^\natural = l(n_1)\cdots l(n_m)$.
   b. For each $(j, k) \in match(w', sub(d(a), u)')$, insert a new node labeled by b as the parent of $n_j, \cdots, n_k$ into t.
3. Return t transformed above.

$\textsc{Trans1b}_{d,op}(t)$ runs in $O(|t| \cdot |d(a)|^2)$ time.

Example 3: Let us consider the intermediate transformations from t_3 to t_4 and from t_4 to t_5 in Fig. 4 (right).

($t_3 \Rightarrow t_4$) Let d_3 be the DTD in Fig. 4 (d). Then $d_3(\text{staff}) = \cdot(\text{name}, \cdot(\text{street}, \text{zip}), \text{email})$. Since $op_4 = agg\_elm(\text{staff}, \text{address}, 2)$, t_3 is transformed by step 2 of Trans1b. Consider the node n_1 in t_3. Since $d_3(\text{staff})' = \cdot(\text{name}^1, \cdot(\text{street}^{21}, \text{zip}^{22}), \text{email}^3)$, the superscripted word w' of $l(n_2)l(n_8)l(n_4)l(n_5)$ = "name street zip email" such that $w' \in L(d_3(\text{staff})')$ is "$\text{name}^1\,\text{street}^{21}\,\text{zip}^{22}\,\text{email}^3$". Thus we have $match(w', \cdot(\text{street}^{21}, \text{zip}^{22})) = \{(2, 3)\}$. Therefore, a new node n_9 labeled by "address" is inserted as the parent of the second and third children n_8, n_4 of n_1.

($t_4 \Rightarrow t_5$) Let d_4 be the DTD in Fig. 4 (e). Then $d_4(\text{staff}) = \cdot(\text{name}, \text{address}, \text{email})$. Since $op_5 = ext\_elm(\text{staff}, 1)$, t_4 is transformed by step 1 of Trans1b. Consider the node n_1 in t_4. Since $d_4(\text{staff})' = \cdot(\text{name}^1, \text{address}^2, \text{email}^3)$, the superscripted word w' of $l(n_2)l(n_9)l(n_5)$ = "name address email" such that $w' \in L(d_4(\text{staff})')$ is "$\text{name}^1\,\text{address}^2\,\text{email}^3$", thereby $match(w', \text{name}^1) = \{(1, 1)\}$. Thus the first child n_2 of n_1 is deleted. □

Finally, we show Trans2. Note that for any DTD d and any tree t valid against d, by Lemma 1 t is always valid against $op(d)$ if $op = ins\_opr(a, opr, vi, vj)$. Thus we do not have to transform t if $op = ins\_opr(a, opr, vi, vj)$, and it suffices to consider the case of $op = del\_opr(a, u)$. We need a definition. Let w' be a superscripted word and b^h be a superscripted label. Then a *variant* of w' w.r.t. b^h is a superscripted word obtained by deleting some b^h's from w' and inserting b^h's into w' at arbitrary positions.

$\textsc{Trans2}_{d,op}(t)$

1. By Conditions (4) and (5) of Lemma 1, $op = del\_opr(a, u)$ and $l(d(a), u) = \text{`}*\text{'}$. Thus $sub(d(a), u) = *(q)$ for some regular expression q. For simplicity, we assume that q is a single label b (the other case can be handled similarly) and let $q' = b^h$. For each node n in t labeled by a, do the following.
   a. Let $n_1, \cdots, n_m$ be the children of n in t. Find a superscripted word w' such that $w' \in L(d(a)')$ and that $(w')^\natural = l(n_1)\cdots l(n_m)$.
   b. Find a variant w'' of w' w.r.t. b^h such that $w'' \in L((op(d)(a))')$. For each node n_j corresponding to an occurrence of b^h deleted from w', delete the subtree rooted at n_j from t. For each occurrence of b^h inserted into w', create a new tree t_b valid against $op(d)$ whose root is labeled by b and insert t_b as a child of n at the position corresponding to the occurrence of b^h.
2. Return t transformed above.

A variant w'' of w' in step (1b) can be obtained in $O(|d(a)|^2 + |w'|)$ time (details are shown in Appendix B), and tree t_b can be obtained in $O(|d|)$ time. Thus $\textsc{Trans2}_{d,op}(t)$ runs in $O(|t| \cdot |d(a)|^2 + |t| \cdot |d|)$ time.

Let us now define the transformation algorithm inferred from a DTD and an update script. Let d be a DTD and $s = op_1 \cdots op_n$ be an update script to d. The *transformation algorithm inferred from d and s*, denoted $\textsc{Transform}_{d,s}(t)$, is defined as follows.

$\textsc{Transform}_{d,s}(t)$
Input: a tree t valid against d.
Output: a tree valid against $s(d)$.

1. If $s = \epsilon$, then return t.
2. Otherwise, let $d_{i-1} = op_{i-1}(op_{i-2}(\cdots(op_1(d))\cdots))$ and $s_{i-1} = op_1 op_2 \cdots op_{i-1}$. Return $\textsc{TransOp}_{d_{i-1},op_i}(\textsc{Transform}_{d,s_{i-1}}(t))$.

Note that $\textsc{Transform}_{d,s}(t)$ outputs a single tree but it may not be unique. Let $TS_{d,s}(t) = \{t' \mid t'$ can be the result of $\textsc{Transform}_{d,s}(t)\}$. We say that the transformation algorithm inferred from d and s is *unambiguous* if for any tree t valid against d, $|TS_{d,s}(t)| = 1$. This unambiguity is discussed in the subsequent two sections.

It is clear that $\textsc{Transform}_{d,s}(t)$ is "correct".

Theorem 1: Let d be a DTD and s be an update script s to d. Then for any tree t valid against d and any $t' \in TS_{d,s}(t)$, t' is valid against $s(d)$. □

Finally, let us consider the running time of $\textsc{Transform}_{d,s}(t)$. By $|t|$ we mean the number of nodes in t and by $|d|$ we mean the size of d. We first have the following lemma.

Lemma 2: $\textsc{TransOp}_{d,op}(t)$ runs in $O(|t| \cdot |d|^2)$ time.

Proof (sketch): First, consider line 1 of TransOp. For an unranked tree automaton A and a tree t, whether $t \in L(A)$ can be determined in $O(|t| \cdot |A|^2)$ time [17]. Thus, whether t is valid against $op(d)$ can be checked in $O(|t| \cdot |op(d)|^2)$ time, where $|op(d)| \leq |d| + c$ for some constant c. Consider next line 2. Let R be a regular expression in d with the maximum size. $\textsc{Trans1A}_{d,op}(t)$ runs in $O(|t| \cdot |R|^2 + |t| \cdot |d|)$ time, $\textsc{Trans1B}_{d,op}(t)$ runs in $O(|t| \cdot |R|^2)$ time, and $\textsc{Trans2}_{d,op}(t)$ runs in $O(|t| \cdot |R|^2 + |t| \cdot |d|)$ time, where $O(|t| \cdot |R|^2 + |t| \cdot |d|) \subseteq O(|t| \cdot |d|^2)$. □

Let d be a DTD, $s = op_1 \cdots op_n$ be an update script to d, and let $d_{i-1} = op_{i-1}(\cdots(op_1(d))\cdots)$. If $op_i = ins\_elm(a, b, u)$ and for some ancestor position v of u $l(d_{i-1}(a), v) =$ '$*$', then op_i is *starred*. If for some i, j with $i \leq j$ $op_i = ins\_elm(a, b, u)$, $op_j = ins\_elm(c, e, v)$, and c occurs in $d_{j-1}(b)$, then op_j is *nesting*. We define that $D_{max} = \max\{|d|, |op_1(d)|, \cdots, |s(d)|\}$. We now have the following theorem.

Theorem 2: Let d be a DTD, $s = op_1 \cdots op_n$ be an update script to d, and t be a tree valid against d. If the following condition holds, then $\textsc{Transform}_{d,s}(t)$ runs in $O(n^3 \cdot |t| \cdot D_{max}^3)$ time.

C1) For every $1 \leq i \leq n$, if op_i is an $ins\_elm()$ operation, then op_i is neither starred nor nesting.

Proof (sketch): Assume that Condition (C1) holds. Let us consider how the size of t increases w.r.t. s. Among the six operations defined in Sect. 3, only $ins\_elm()$ and $agg\_elm()$ may increase the size of t. We have the following observations.

- For a tree t', the number of nodes inserted into t' by an $agg\_elm(a, b, u)$ operation is at most the number of nodes in t' labeled by a. Moreover, label b cannot occur in t' by definition.
- Let op be an $ins\_elm()$ operation that is neither nesting nor starred. For each node n in a tree t', at most one new tree, say t'', is inserted as the child of n by op, where $|t''| \in O(D_{max})$. Thus, the number of nodes inserted into t' by op is in $O(|t'| \cdot D_{max})$.

Therefore, if s contains k $ins\_elm()$'s and l $agg\_elm()$'s, the size of input tree t grows at most $k \cdot l \cdot D_{max} \cdot |t|$ by s. This and Lemma 2 imply that $\textsc{Transform}_{d,s}(t)$ runs in $O(n \cdot (k \cdot l \cdot D_{max} \cdot |t|) \cdot D_{max}^2)$ time, where $k \leq n$ and $l \leq n$. □
It is open whether $\textsc{Transform}_{d,s}(t)$ runs in polynomial time in the case where Condition (C1) does not hold.

5. Sufficient Conditions for Unambiguous Transformation

In this section, we first show that deciding whether the transformation algorithm inferred from a DTD and an update script is unambiguous is PSPACE-hard. We next show sufficient conditions for the decision problem.

5.1 PSPACE-hardness

The *unambiguity problem* is to decide, for a DTD d and an update script s, whether the transformation algorithm inferred from d and s is unambiguous. In this subsection, we show that the unambiguity problem is PSPACE-hard.

Theorem 3: The unambiguity problem is PSPACE-hard even if an update script consists only of one update operation.

Proof: We use the *inclusion problem* for regular expressions, which is to decide, for two regular expressions r_1 and r_2, whether r_1 includes r_2. This problem is PSPACE-complete [18]. For an instance of the inclusion problem, we construct an instance of the unambiguity problem, as follows. Let a, b, c be labels occurring in neither r_1 nor r_2.

- Let d be a DTD, where $d(a) = +(\cdot(*(b), *(b), r_1), \cdot(*(b), *(b), r_2))$ and $d(l) = \epsilon$ for any label l occurring in $d(a)$.
- $s = ins\_elm(a, c, 22)$.

We have $op(d)(a) = +(\cdot(*(b), *(b), r_1), \cdot(*(b), c, *(b), r_2))$. Let T be the transformation algorithm inferred from d and s. We show that r_1 includes r_2 *iff* T is unambiguous.

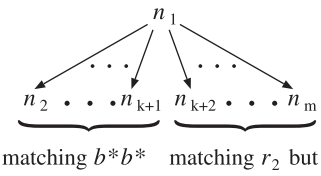

Fig. 5 A tree t valid against d but not $s(d)$.

Assume first that r_1 includes r_2. Then it is easy to show that for any tree t valid against d, t is also valid against $s(d)$, and thus t is not transformed by T. Thus T is unambiguous.

Assume next that r_1 does not include r_2. Consider the tree t shown in Fig. 5. Then t is valid against d but not valid against $s(d)$. Since $s = ins\_elm(a, c, 22)$, T tries to insert a node labeled by c into t, but there are $k + 1$ positions in t at which such a node can be inserted. Hence T is not unambiguous. □

Thus, it is unlikely that the unambiguity problem can be solved efficiently. In what follows, we consider efficiently testable sufficient conditions for the unambiguity problem.

5.2 Sufficient Conditions

In this subsection, we show sufficient conditions for the unambiguity problem. In Sect. 6, we will show a polynomial-time algorithm for testing the sufficient conditions.

For an input tree t, the result of $\textsc{TransOp}_{d,op}(t)$ may not be unique due to the following reasons.

U1) In step (1b) of Trans1a, $match(w', b^h)$ depends on the superscripted supersequence w' selected in step (1a).
U2) In step (1b) of Trans1a, there may be more than one trees valid against $op(d)$ whose root is labeled by b.
U3) In step (2b) of Trans1a, $match(w', b^h)$ depends on the superscripted word w' selected in step (2a). A similar argument also applies to steps (1b) and (2b) of Trans1b.
U4) In step (1b) of Trans2, there may be more than one variant w'' of w'.

Let us first show a simple sufficient condition related to (U2). We define a DTD that admits exactly one valid tree. For a DTD d and a label b, we say that $d(b)$ is *simple* if for any label a reachable from b, $|L(d(a))| = 1$. For example, let d be a DTD, where $d(a) = \cdot(b, c, e)$, $d(b) = \cdot(f, g)$, and $d(c) = d(e) = d(f) = d(g) = \epsilon$. Then $d(a)$ is simple. We have the following lemma.

Lemma 3: For any DTD d, there is exactly one tree valid against d whose root is labeled by b *iff* $d(b)$ is simple. □

A DTD d is *simple* if for every label b such that $d(b)$ is defined, $d(b)$ is simple. Assuming that only simple DTDs are available, none of the ambiguities of (U1) to (U4) arises. Thus we have the following.

Theorem 4: Let d be a DTD and $s = op_1 \cdots op_n$ be an update script to d. If $d, op_1(d), \cdots, s(d)$ are simple, then the transformation algorithm inferred from d and s is unambiguous. □

Since the above DTD is too restrictive, let us next consider more general ones. We show definitions related to (U1) and (U3). Consider first (U3). A regular expression r is *one-unambiguous* if the Glushkov automaton of r is deterministic [6] (Glushkov automaton is defined in Sect. 6.2). The XML specification [4] requires any content model in a DTD to be one-unambiguous (non-normatively). If r is one-unambiguous, then for any word $w \in L(r)$, there is exactly one superscripted word w' such that $w' \in L(r')$ and that $(w')^\natural = w$ [6]. Thus, if every content model is one-unambiguous, then the superscripted word w' in (U3) can uniquely be determined. Consider next (U1). We define a regular expression that admits only superscripted supersequences such that the positions at which b^h should be inserted can unambiguously determined. Let d be a DTD, $op = ins\_elm(a, b, vi)$, and b^h be the superscripted label in $(op(d)(a))'$ inserted by op. We say that $d(a)$ is *unambiguous* w.r.t. the insertion of b^h if for any word $w \in L(r)$ and for any superscripted supersequences w', w'' of w w.r.t. b^h such that $w', w'' \in L(op(d)(a))$, (i) $|w'| = |w''|$, and (ii) for any $1 \le k \le |w'|$, $w'[k] = b^h$ *iff* $w''[k] = b^h$.

Example 4: Let d be a DTD, $d(a) = \cdot(*(b), *(b))$, and $op = ins\_elm(a, c, 2)$. Then $(op(d)(a))' = \cdot(*(b^{11}), c^2, *(b^{31}))$. Consider a word $bb \in L(d(a))$. There are three superscripted supersequences of bb w.r.t. c^2 belonging to $L((op(d)(a))')$; $c^2b^{31}b^{31}$, $b^{11}c^2b^{31}$, and $b^{11}b^{11}c^2$. Thus $d(a)$ is not unambiguous w.r.t. the insertion of c^2. □

Since ϵ and '$*$' in a content model allow a label to occur "optionally" and "repeatedly", a content model $d(a)$ containing ϵ or '$*$' is not necessarily unambiguous w.r.t. the insertion of a superscripted label, even if $d(a)$ is one-unambiguous. We say that a DTD d is *limited* if $d(a)$ is one-unambiguous and contains neither ϵ nor '$*$' for every content model $d(a)$ of d. We have the following lemma (the proof is shown in Appendix C).

Lemma 4: Let d be a DTD and $op = ins\_elm(a, b, vi)$ be an update operation to d. If d and $op(d)$ are limited, then $d(a)$ is unambiguous w.r.t. the insertion of b^h, where b^h is the superscripted label inserted by op. □

If only limited DTDs are available, it suffices to check the ambiguity of (U2), and if only one-unambiguous DTDs are available, it suffices to check the ambiguities of (U1), (U2), and (U4). Thus we have the following.

Theorem 5: Let d be a DTD, $s = op_1 \cdots op_n$ be an update script to d, and $d_j = op_j(\cdots(op_1(d))\cdots)$.

1. Assume that $d_0, \cdots, d_n$ are limited. Then the transformation algorithm inferred from d and s is unambiguous if the following condition holds.
   R1) For every $1 \le j \le n$, if $op_j = ins\_elm(a, b, vi)$ and $l(d_{j-1}(a), v) =$ '$\cdot$' for some $a, b \in \Sigma$ and some $vi \in pos(d_{j-1}(a))$, then $d_j(b)$ is simple.
2. Assume that $d_0, \cdots, d_n$ are one-unambiguous. Then the

transformation algorithm inferred from d and s is unambiguous if the following condition holds as well as Condition (R1).

R2) For every $1 \leq j \leq n$,

- if $op_j = ins\_elm(a, b, vi)$ and $l(d_{j-1}(a), v) = \cdot$' for some $a, b \in \Sigma$ and some $vi \in pos(d_{j-1}(a))$, then $d_{j-1}(a)$ is unambiguous w.r.t. the insertion of b^h, where b^h is the superscripted label inserted by op_j, and
- if $op_j = del\_opr(a, u)$ for some $a \in \Sigma$ and some $u \in pos(d_{j-1}(a))$, then Condition (5) of Lemma 1 holds w.r.t. d_{j-1} and op_j. □

Finally, let us consider a sufficient condition for the unambiguity problem, without any assumption on DTD such as "limited" and "one-unambiguous". We give definitions related to (U3). We define a regular expression such that $match(w', sub(r, u)')$ can unambiguously determined. Let r be a regular expression, $u \in pos(r)$ be a position, and $w \in L(r)$ be a word. We say that r is *unambiguous w.r.t.* $sub(r, u)$ and w if for any superscripted words w', w'' such that $(w')^\natural = (w'')^\natural = w$ and that $w', w'' \in L(r')$, $match(w', sub(r, u)') = match(w'', sub(r, u)')$. We say that r is *unambiguous* w.r.t. $sub(r, u)$ if r is unambiguous w.r.t. $sub(r, u)$ and w for any $w \in L(r)$.

Example 5: Let d be the DTD in Fig. 2 (a) and let $d(b) = *(\cdot(s, *(s), +(a, \epsilon)))$, where labels b, s, and a stand for "book", "section", and "ack", respectively. We have $d(b)' = *(\cdot(s^{11}, *(s^{121}), +(a^{131}, \epsilon^{132})))$. Let $q' = sub(d(b), 1)' = \cdot(s^{11}, *(s^{121}), +(a^{131}, \epsilon^{132}))$. For a word $w = ssa \in L(d(b))$, we have two superscripted words $w' = s^{11}s^{11}a^{131}$ and $w'' = s^{11}s^{121}a^{131}$ of w such that $w', w'' \in L(d(b)')$. Since $match(w', q') = \{(1, 1), (2, 3)\}$ and $match(w'', q') = \{(1, 3)\}$, $d(b)$ is not unambiguous w.r.t. $q = sub(d(b), 1)$. □

We now obtain a sufficient condition for the unambiguity problem.

Theorem 6: Let d be a DTD, $s = op_1 \cdots op_n$ be an update script to d, and $d_j = op_j(\cdots(op_1(d))\cdots)$. Then the transformation algorithm inferred from d and s is unambiguous if for every $1 \leq j \leq n$, at least one of the following Conditions (S1) to (S5) holds.

S1) One of Conditions (1) to (5) of Lemma 1 holds w.r.t. d_{j-1} and op_j.
S2) $op_j = ins\_elm(a, b, vi)$ for some $a, b \in \Sigma$ and some $vi \in pos(d_{j-1}(a))$, $d_{j-1}(a)$ is unambiguous w.r.t. the insertion of b^h, and $d_j(b)$ is simple, where b^h is the superscripted label inserted by op_j.
S3) $op_j = del\_elm(a, vi)$ for some $a \in \Sigma$ and some $vi \in pos(d_{j-1}(a))$, and $d_{j-1}(a)$ is unambiguous w.r.t. $l(d_{j-1}(a), vi)$.
S4) $op_j = ext\_elm(a, u)$ for some $a \in \Sigma$ and some $u \in pos(d_{j-1}(a))$, and $d_{j-1}(a)$ is unambiguous w.r.t. $l(d_{j-1}(a), u)$.
S5) $op_j = agg\_elm(a, b, u)$ for some $a, b \in \Sigma$ and some $u \in pos(d_{j-1}(a))$, and $d_{j-1}(a)$ is unambiguous w.r.t. $sub(d_{j-1}(a), u)$.

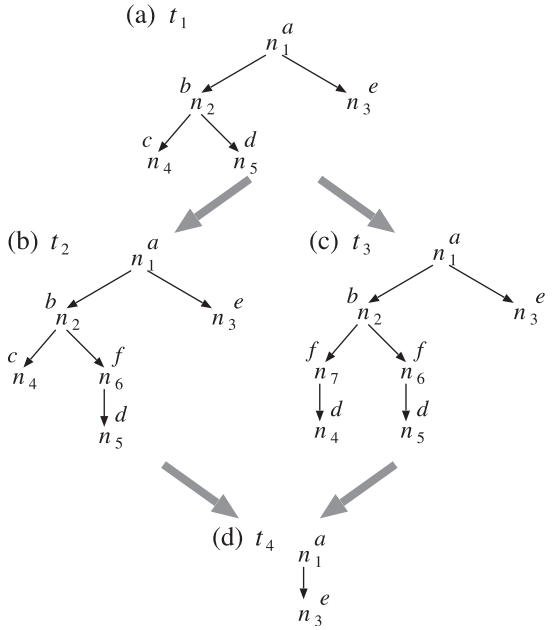

Fig. 6 Trees t_1, t_2, t_3, and t_4.

Proof (sketch): Assume that at least one of Conditions (S1) to (S5) holds for every $1 \leq j \leq n$. Then the ambiguities of (U1) and (U2) are avoided by Condition (S2), the ambiguity of (U3) is avoided by Conditions (S3), (S4), and (S5), and the ambiguity of (U4) is avoided by Condition (S1). □

The next example shows that the above condition is not necessary.

Example 6: Let d_1 be a DTD, where

$$d_1(a) = \cdot(b, e),$$
$$d_1(b) = \cdot(*(c), *(+(c, d))),$$
$$d_1(c) = d_1(d) = d_1(e) = \epsilon.$$

Let $s = op_1op_2$ be an update script to d_1, where $op_1 = agg\_elm(b, f, 21)$ and $op_2 = del\_elm(a, 1)$. Let $d_2 = op_1(d_1)$ and $d_3 = s(d_2)$. We have

$$d_2(a) = \cdot(b, e),$$
$$d_2(b) = \cdot(*(c), *(f)),$$
$$d_2(f) = +(c, d),$$
$$d_2(c) = d_1(d) = d_1(e) = \epsilon,$$

and

$$d_3(a) = \cdot(e),$$
$$d_2(e) = \epsilon.$$

Consider how the tree t_1 in Fig. 6 (a) is transformed according to s. It is easy to verify that $d_1(b)$ is not unambiguous w.r.t. $sub(d_1(b), 21)$, thereby Condition (S5) does not hold. Thus, according to op_1 t_1 can be transformed into two trees t_2 and t_3 (Fig. 6 (b,c)). Then, according to op_2, both t_2 and

t_3 are transformed into the same tree t_4 (Fig. 6 (d)). In general, it is easy to see that for any tree t valid against d_1, $|TS_{d_1,s}(t)| = 1$ whereas op_1 satisfies none of Conditions (S1) to (S5). □

6. Testing the Sufficient Conditions

In this section, we show a polynomial-time algorithm for testing the the sufficient conditions. We have to solve the following problems.

P1) In Condition (R1) of Theorem 5 and Condition (S2) of Theorem 6, we have to determine if a content model is simple.
P2) In Conditions (S3), (S4), and (S5) of Theorem 6, we have to determine if a regular expression $d(a)$ is unambiguous w.r.t. a subexpression of $d(a)$.
P3) In Condition (R2) of Theorem 5 and Condition (S2) of Theorem 6, we have to determine if a regular expression $d(a)$ is unambiguous w.r.t. the insertion of b^h.

In the subsequent subsections, we show how to solve the above problems.

6.1 Checking the Simplicity of a DTD

Consider first (P1). Recall that content model $d(b)$ is simple if for any label a reachable from b, $|L(d(a))| = 1$. It is easy to show that, given regular expression r, $|L(r)| = 1$ *iff* for an ϵ-free NFA M such that $L(M) = L(r)$, all the paths from the initial state to the final state in M have the same sequence of labels. This implies that testing if $|L(r)| = 1$ can be done in $O(|r|^2)$ time. In particular, if r is one-unambiguous, then whether $|L(r)| = 1$ can be checked in $O(|r|)$ time since the Glushkov automaton of r (defined below) can be constructed in linear time [6]. Thus, we have the following lemma.

Lemma 5: Let d be a DTD and b be a label. Then whether $d(b)$ is simple can be checked in $O(|d|^2)$ time. In particular, if d is one-unambiguous, then this check can be done in $O(|d|)$ time. □

6.2 Checking the Unambiguity of a Regular Expression w.r.t. a Subexpression

Consider next (P2). Let r be a regular expression. In order to determine whether r is unambiguous w.r.t. a subexpression of r, we use the Glushkov automaton of r [3], [6] defined as follows. First, we define the *initial set* I_r and the *final set* F_r.

1. If $r = \epsilon$, then $I_r = F_r = \{E\}$, where E is a new label (I_r and F_r contain E if $\epsilon \in L(r)$).
2. If $r = a$ for some $a \in \Sigma$, then $I_r = F_r = \{a^i\}$, where a^i is the superscripted label such that $r' = a^i$.
3. If $r = +(r_1, \cdots, r_n)$, then $I_r = I_{r_1} \cup \cdots \cup I_{r_n}$ and $F_r = F_{r_1} \cup \cdots \cup F_{r_n}$.
4. If $r = \cdot(r_1, \cdots, r_n)$, then
$$I_r = (I_{r_1} - \{E\}) \cup \cdots \cup (I_{r_{i-1}} - \{E\}) \cup I_{r_i},$$
$$F_r = F_{r_j} \cup (F_{r_{j+1}} - \{E\}) \cup \cdots \cup (F_{r_n} - \{E\}),$$
where
$$i = \begin{cases} n & \text{if } E \in I_{r_k} \text{ for every } 1 \le k \le n, \\ \min\{k \mid E \notin I_{r_k}, 1 \le k \le n\} & \text{otherwise,} \end{cases}$$
$$j = \begin{cases} 1 & \text{if } E \in F_{r_k} \text{ for every } 1 \le k \le n, \\ \max\{k \mid E \notin F_{r_k}, 1 \le k \le n\} & \text{otherwise.} \end{cases}$$
5. If $r = *(r_1)$, then $I_r = I_{r_1} \cup \{E\}$ and $F_r = F_{r_1} \cup \{E\}$.

Let a^i be a superscripted label occurring in r'. The *set of successors* of a^i in r', denoted $Succ(a^i, r')$, is defined as follows.

1. If $r' = a^i$, then $Succ(a^i, r') = \emptyset$.
2. If $r' = +(r'_1, \cdots, r'_n)$ and a^i occurs in r'_k $(1 \le k \le n)$, then $Succ(a^i, r') = Succ(a^i, r'_k)$.
3. If $r' = \cdot(r'_1, \cdots, r'_n)$ and a^i occurs in r'_k $(1 \le k \le n)$, then
$$Succ(a^i, r') = \begin{cases} Succ(a^i, r'_k) \\ \qquad \text{if } k = n \text{ or } a^i \notin F_{r_k}, \\ Succ(a^i, r'_k) \cup (I_{r_{k+1}} - \{E\}) \cup \\ \cdots \cup (I_{r_j} - \{E\}) \\ \qquad \text{if } k < n \text{ and } a^i \in F_{r_k}, \end{cases}$$
where
$$j = \begin{cases} n & \text{if } E \in I_{r_i} \text{ for every } k+1 \le i \le n, \\ \min\{i \mid E \notin I_{r_i}, k+1 \le i \le n\} & \text{otherwise.} \end{cases}$$
4. If $r' = *(r'_1)$, then
$$Succ(a^i, r') = \begin{cases} Succ(a^i, r'_1) & \text{if } a^i \notin F_{r_1}, \\ Succ(a^i, r'_1) \cup (I_{r_1} - \{E\}) & \text{otherwise.} \end{cases}$$

The *Glushkov automaton* of r is a 5-tuple $(Q, \Sigma, \delta, q^I, F)$, where Q is the set of *states*, δ is the *transition function*, q^I is the *initial state*, and F is the set of *final states* defined as follows.

- $Q = sym(r') \cup \{q^I\}$,
- $\delta(q^I, a) = \{a^j \mid a^j \in I_r, (a^j)^\natural = a\}$ for every $a \in \Sigma$, and $\delta(a^j, a) = \{a^k \mid a^k \in Succ(a^j, r'), (a^k)^\natural = a\}$,
- $F = \begin{cases} F_r \cup \{q^I\} & \text{if } \epsilon \in L(r), \\ F_r & \text{otherwise.} \end{cases}$

Example 7: Let $r = \cdot(a, *(+(a, b)))$. Then $r' = \cdot(a^1, *(+(a^{211}, b^{212})))$. We have $I_r = \{a^1\}$, $F_r = \{a^1, a^{211}, b^{212}\}$, and $Succ(a^1, r') = Succ(a^{211}, r') = Succ(b^{212}, r') = \{a^{211}, b^{212}\}$. The Glushkov automaton of r is illustrated in Fig. 7. □

It is easy to show by induction that for any regular expression r, $L(r) = L(G_r)$.

We test the unambiguity by using a graph called testing graph[†] of a Glushkov automaton. Let $G_r = (Q, \Sigma, \delta, q^I, F)$

[†]We use a modified version of testing graph, originally defined in [8].

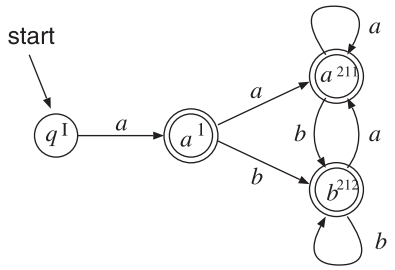

Fig. 7 The Glushkov automaton of $\cdot(a, *(+(a, b)))$.

be the Glushkov automaton of r. A pair (a^i, a^j) of states in Q is *compatible* if (i) $a^i = a^j = q^I$, or (ii) there is a compatible pair (a^k, a^l) such that, $a^i \in \delta(a^k, a)$, $a^j \in \delta(a^l, a)$, and that $(a^i)^\natural = (a^j)^\natural = a$. Then the *testing graph* of G_r is a graph $T(G_r) = (N, E)$, where

$$N = \{(a^i, a^j) \mid (a^i, a^j) \text{ is a compatible pair of } Q\},$$
$$E = \{(a^k, a^l) \xrightarrow{a} (a^i, a^j) \mid a^i \in \delta(a^k, a),\ a^j \in \delta(a^l, a)\}.$$

The following lemma holds by definition.

Lemma 6: Let r be a regular expression and $G_r = (Q, \Sigma, \delta, q^I, F)$ be the Glushkov automaton of r. There are superscripted words $w', w'' \in L(r')$ such that $(w')^\natural = (w'')^\natural$ and that $|w'| = |w''| = l$ *iff* there is a path $(q^I, q^I) \xrightarrow{(a^{i_1})^\natural} (a^{i_1}, a^{j_1}) \xrightarrow{(a^{i_2})^\natural} \cdots \xrightarrow{(a^{i_l})^\natural} (a^{i_l}, a^{j_l})$ in $T(G_r)$ such that $w'[k] = a^{i_k}$ and $w''[k] = a^{j_k}$ for every $1 \le k \le l$ and that $a^{i_l}, a^{j_l} \in F$. □

We say that a compatible pair (a^i, a^j) is *accepting* if $a^i, a^j \in F$. Now the unambiguity can be checked as follows.

Theorem 7: Let r be a regular expression, $u \in pos(r)$ be a position, $q = sub(r, u)$ be a subexpression of r, and G_r be the Glushkov automaton of r. Then r is unambiguous w.r.t. q *iff* the following two conditions hold.

1. For any node (a^i, a^j) in $T(G_r)$ from which some accepting node is reachable, either $a^i, a^j \in sym(q')$ or $a^i, a^j \in sym(r') \backslash sym(q')$.
2. For any edge $(a^i, a^j) \xrightarrow{a} (a^k, a^l)$ in $T(G_r)$ such that some accepting node is reachable from (a^k, a^l), if $a^i, a^j, a^k, a^l \in sym(q')$, then either
   a. $a^k \notin Succ(a^i, q')$ and $a^l \notin Succ(a^j, q')$, or
   b. $a^k \in Succ(a^i, q')$ and $a^l \in Succ(a^j, q')$.

Proof: *Only if part:* Assume that at least one of Conditions (1) and (2) does not hold. Then by Lemma 6 it is easy to show that there are words $w', w'' \in L(r')$ such that $(w')^\natural = (w'')^\natural$ and that for some i, j $(1 \le i \le j \le |w'|)$ $w'[i, j]$ maximally matches q' but $w''[i, j] \notin L(q')$. Thus, r is not unambiguous w.r.t. q by definition.

If part: Assume that r is not unambiguous w.r.t. q. Then there are words $w', w'' \in L(r')$ such that $(w')^\natural = (w'')^\natural$ and that for some i, j $(1 \le i \le j \le |w'|)$ $w'[i, j]$ maximally matches q' but $w''[i, j]$ does not maximally match q'. First, if $w''[k] \notin sym(q')$ for some $i \le k \le j$, then by Lemma 6 Condition (1) does not hold. Assume on the other hand that $w''[k] \in sym(q')$ for any $i \le k \le j$. This and the fact that $w''[i, j]$ does not maximally matches q' imply that (i) $w''[i-1] \in sym(q')$ and $w''[i] \in Succ(w''[i-1], q')$, (ii) $w''[j+1] \in sym(q')$ and $w''[j+1] \in Succ(w''[j], q')$, or (iii) $w''[k+1] \notin Succ(w''[k], q')$ for some $i \le k < j$. In the case of (iii), it is clear that Condition (2) does not hold. Consider the case of (i) (the case of (ii) can be shown similarly). Since $w'[i, j]$ maximally matches q', we have either (a) $w'[i-1] \notin sym(q')$ or (b) $w'[i-1] \in sym(q')$ but $w'[i] \notin Succ(w'[i-1], q')$. This and Lemma 6 imply that at least one of Conditions (1) of (2) does not hold. □

Lemma 7: Let r be a regular expression and q be a subexpression of r. Then whether r is unambiguous w.r.t. q can be checked in $O(|r|^4)$ time. In particular, if r is one-unambiguous, then this check can be done in $O(|r|^2)$ time.

Proof: Assume first that r is not one-unambiguous. Then the Glushkov automaton G_r of r can be constructed in $O(|r|^2)$ time [5], and the testing graph $T(G_r)$ can be constructed in $O(|r|^4)$ time. Moreover, the condition in Theorem 7 can be checked in linear time w.r.t. $T(G_r)$. Second, if r is one-unambiguous, then the lemma follows from the fact that G_r can be constructed in linear time. □

6.3 Checking the Unambiguity of a Regular Expression w.r.t. the Insertion of a Superscripted Label

Finally, let us consider (P3). We show how to decide if a regular expression is unambiguous w.r.t. the insertion of a superscripted label.

To check this unambiguity we slightly modify the testing graph of a Glushkov automaton. Let r be a regular expression, $G_r = (Q, \Sigma, \delta, q^I, F)$ be the Glushkov automaton of r, and $b^h \in Q$ be a superscripted label. We first define a *contracted transition function* δ' w.r.t. b^h, which is obtained by contracting each pair of transitions from a^i to b^h and from b^h to a^j into one transition from a^i to a^j. Formally, δ' is defined so that for any $a^i, a^j \in Q$, $a^j \in \delta'(a^i, a)$ *iff*

- $a^j \in \delta(a^i, a)$, $a^i \ne b^h$, and $a^j \ne b^h$, or
- $b^h \in \delta(a^i, b)$ and $a^j \in \delta(b^h, a)$,

where $a = (a^j)^\natural$ and $b = (b^h)^\natural$. A pair (a^i, a^j) of states is *compatible* (w.r.t. δ') if (i) $a^i = a^j = q^I$, or (ii) there is a compatible pair (a^k, a^l) such that, $a^i \in \delta'(a^k, a)$, $a_j \in \delta'(a^l, a)$, and that $(a^i)^\natural = (a^j)^\natural = a$. Then the *contracted testing graph* of G_r w.r.t. b^h, denoted $T_{b^h}(G_r)$, is a graph (N, E), where

$$N = \{(a^i, a^j) \mid (a^i, a^j) \text{ is a compatible pair of } Q\},$$
$$E = \{(a^k, a^l) \xrightarrow{a} (a^i, a^j) \mid a^i \in \delta'(a^k, a),\ a^j \in \delta'(a^l, a)\}.$$

For an edge $(a^k, a^l) \xrightarrow{a} (a^i, a^j) \in E$, if (i) $a^i \in \delta(a^k, a)$ but $b^h \in \delta(a^l, b)$ and $a^j \in \delta(b^h, a)$, or (ii) $a^j \in \delta(a^l, a)$ but $b^h \in \delta(a^k, b)$ and $a^i \in \delta(b^h, a)$, then we say that $(a^k, a^l) \xrightarrow{a} (a^i, a^j)$ is *odd*. If a node $(a^i, a^j) \in N$ satisfies one of the following conditions, then (a^i, a^j) is called *accepting*.

1. $a^i \in F$ and $a^j \in F$.

2. $b^h \in \delta(a^i, b)$, $b^h \in \delta(a^j, b)$, and $b^h \in F$.
3. (i) $a^i \in F$, $b^h \in \delta(a^j, b)$, and $b^h \in F$, or (ii) $a^j \in F$, $b^h \in \delta(a^i, b)$, and $b^h \in F$.

In particular, if (a^i, a^j) satisfies Condition (3), then (a^i, a^j) is *oddly* accepting. Now the unambiguity can be checked as follows.

Theorem 8: Let d be a DTD, $op = ins\_elm(a, b, vi)$ be an update operation to d such that $l(d(a), v) =$ '$\cdot$', b^h be the superscripted label inserted by op, and $G_{op(d)(a)}$ be the Glushkov automaton of $op(d)(a)$. Then $d(a)$ is *unambiguous* w.r.t. the insertion of b^h *iff* the following three conditions hold.

1. For any odd edge $(a^k, a^l) \xrightarrow{a} (a^i, a^j)$ in $T_{b^h}(G_{op(d)(a)})$, no accepting node is reachable from (a^i, a^j).
2. $T_{b^h}(G_{op(d)(a)})$ contains no oddly accepting node.
3. $b^h \notin \delta(b^h, b)$, where δ is the transition function of $G_{op(d)(a)}$.

Proof (sketch): Let w be a word. Since $l(d(a), u) =$ '$\cdot$', $w \in L(d(a))$ *iff* there is a superscripted supersequence w' of w w.r.t. b^h such that $w' \in L((op(d)(a))')$. Thus the theorem can be proved as follows.

Only if part: It is easy to show that if one of Conditions (1) to (3) does not hold, then $d(a)$ is not unambiguous w.r.t. the insertion of b^h.

If part: Assume that $d(a)$ is not unambiguous w.r.t. the insertion of b^h. Then for some word $w \in L(d(a))$, there are superscripted supersequences w', w'' of w w.r.t. b^h such that $w', w'' \in L((op(d)(a))')$ and that for some k $(1 \le k \le |w'|)$

- $w'[i] = w''[i]$ for every $1 \le i \le k-1$, but $w'[k] = b^h$ and either $w''[k] \ne b^h$ or $|w''| = k-1$.

Assume that $k > 1$ (the case where $k = 1$ can be shown similarly). We have two cases to be considered according to $|w''|$.

The case where $|w''| \ge k$*:* We have $w''[k] \ne b^h$. Suppose first that $w'[k-1] = w''[k-1] \ne b^h$. Since $w''[k] \ne b^h$, there is an index $k' > k$ such that $w'[k']$ matches $w''[k]$. If $k' = k+1$, then Condition (1) does not hold (since $(w'[k-1], w''[k-1]) \xrightarrow{a} (w'[k'], w''[k])$ is an odd edge). If $k' > k+1$, then $w'[k] = w'[k+1] = b^h$, thereby Condition (3) does not hold. Suppose next that $w'[k-1] = w''[k-1] = b^h$. Then $w'[k-1] = w'[k] = b^h$, thus Condition (3) does not hold.

The case where $|w''| = k-1$*:* If $w'[k-1] = w''[k-1] \ne b^h$, then $(w'[k-1], w''[k-1])$ must be an oddly accepting node and Condition (2) does not hold. If $w'[k-1] = w''[k-1] = b^h$, then $w'[k-1] = w'[k] = b^h$, thereby Condition (3) does not hold. □

The following lemma can be shown similarly to Lemma 7.

Lemma 8: Let d be a DTD and $op = ins\_elm(a, b, vi)$ be an update operation such that $l(d(a), v) =$ '$\cdot$'. Then whether $d(a)$ is unambiguous w.r.t. the insertion of b^h can be checked in $O(|d(a)|^4)$ time, where b^h is the superscripted label inserted by op. In particular, if $d(a)$ is one-unambiguous, then this check can be done in $O(|d(a)|^2)$ time. □

Finally, the complexities of checking the sufficient conditions are summarized as follows.

Theorem 9: Let d be a DTD, $s = op_1 \cdots op_n$ be an update script to d, k be the number of $ins\_elm()$ operations in s, l be the number of $del\_opr()$ operations in s, and R_{max} be a regular expression with maximum size occurring in $d, op_1(d), \cdots, s(d)$.

1. Whether $d, op_1(d), \cdots, s(d)$ are simple/limited/one-unambiguous can be checked in $O(|d|^2 + n \cdot |R_{max}|^2)$ time.
2. Assuming that $d, op_1(d), \cdots, s(d)$ are limited, Condition (R1) of Theorem 5 can be checked in $O(k \cdot D_{max})$ time.
3. Assuming that $d, op_1(d), \cdots, s(d)$ are one-unambiguous, Conditions (R1) and (R2) of Theorem 5 can be checked in $O(k \cdot |R_{max}|^2 + l \cdot |R_{max}| + k \cdot D_{max})$ time.
4. The condition in Theorem 6 can be checked in $O(n \cdot |R_{max}|^4 + k \cdot D_{max}^2)$ time.

Proof: Consider first (1). In order to check if $d, op_1(d), \cdots, s(d)$ are simple (limited, one-unambiguous), it suffices to check (i) whether d is simple (resp., limited, one-unambiguous) and (ii) whether r is simple (resp., limited, one-unambiguous) for every updated content model r in $op_1(d), \cdots, s(d)$. Whether a regular expression r is one-unambiguous can be determined in $O(|r|^2)$ time [6]. By this and the discussion about Lemma 5, it is easy to show that (i) can be checked in $O(|d|^2)$ time and (ii) can be checked in $O(n \cdot |R_{max}|^2)$ time. Second, (2) follows from Lemma 5. Consider (3). By Condition (5) of Lemma 1 we have to check if $L(q) = \{\epsilon\}$ for a subexpression $q = sub(d(a), u1)$ of $d(a)$, which can be determined in $O(|q|)$ time by using the Glushkov automaton of q. This and Lemmas 5 and 8 imply that (3) holds. Finally, (4) holds by Lemmas 5, 7, and 8. □

7. Conclusion

In this paper, we first proposed a transformation algorithm inferred from a DTD and an update script. Then we show sufficient conditions under which the transformation algorithm inferred from a DTD and an update script is unambiguous. Finally, we presented a polynomial-time algorithm for testing the sufficient conditions.

As a future work, we have to improve the algorithm so that the algorithm covers more effective application area. We need to make experiments in order to examine if the transformation algorithm can be applied to actual XML documents and DTD updates, and the result of Theorem 2 implies that we should improve the complexity of the algorithm and verify this experimentally.

We would like to investigate whether real DTDs tend to admit unambiguous transformation. The unambiguity of regular expression w.r.t. subexpression is a weaker condition than one-unambiguity of regular expression, and Ref. [7] states that only four of 60 real DTDs contain regular expressions that are not one-unambiguous. This might suggest that

real DTDs tend to permit unambiguous transformation.

We also have to make considerations on update operations further. First, there are some restrictions on our update operations to DTDs. For example, a $del\_opr(a, u)$ operation can be applied only if the operator at u is nesting or has just one operand. We have to consider relaxing such restrictions. Second, in this paper we can neither define a new content model nor undefine an existing content model. We should consider incorporating such update operations into our transformation algorithm.

Acknowledgement

The authors are thankful to Associate Professor Yasunori Ishihara of Osaka University and Research Associate Tetsuji Kuboyama of Tokyo University for their insightful comments.

References

[1] M. Arenas and L. Libkin, "Xml data exchange: Consistency and query answering," Proc. ACM PODS, pp.13–24, 2005.
[2] P. Bohannon, W. Fan, M. Flaster, and P.P.S. Narayan, "Information preserving xml schema embedding," Proc. VLDB, pp.85–96, 2005.
[3] R. Book, S. Even, S. Greibach, and G. Ott, "Ambiguity in graphs and expressions," IEEE Trans. Comput., vol.C-20, no.2, pp.149–153, 1971.
[4] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau, eds., Extensible Markup Language (XML) 1.0 (Fourth Edition), http://www.w3.org/TR/xml/, 2006.
[5] A. Brüggenmann-Klein, "Regular expressions into finite automata," Theor. Comput. Sci., vol.120, no.2, pp.197–213, 1993.
[6] A. Brüggenmann-Klein and D. Wood, "One-unambiguous regular languages," Inf. Comput., vol.142, no.2, pp.182–206, 1998.
[7] B. Choi, "What are real dtds like?," Proc. WebDB, pp.43–48, 2002.
[8] S. Even, "On information lossless automata of finite order," IEEE Trans. Electronic Computers, vol.EC-14, pp.561–569, 1965.
[9] G. Guerrini, M. Mesiti, and D. Rossi, "Impact of xml schema evolution on valid documents," Proc. WIDM (in conjunction with ACM CIKM), pp.39–44, 2005.
[10] K. Hashimoto, Y. Ishihara, and T. Fujiwara, "A proposal of update operations for schema evolution in xml databases and their properties on preservation of schema's expressive power," IEICE Trans. Inf. & Syst. (Japanese Edition), vol.J90-D, no.4, pp.990–1004, April 2007.
[11] E. Kuikka, P. Leinonen, and M. Penttonen, "Towards automating of document structure transformations," Proc. ACM DocEng, pp.103–110, 2002.
[12] E. Leonardi, T.T. Hoai, S.S. Bhowmick, and S. Madria, "Dtd-diff: A change detection algorithm for dtds," Proc. DASFAA, pp.817–827, 2006.
[13] R. Miller, M.A. Hernandez, L. Hass, L. Yan, C. Ho, R. Fagin, and L. Popa, "The clio project: Managing heterogeneity," SIGMOD Record, vol.30, no.1, pp.78–83, 2001.
[14] T. Milo and S. Zohar, "Using schema matching to simplify heterogeneous data translation," Proc. VLDB, pp.122–133, 1998.
[15] A. Morishima, T. Okawara, J. Tanaka, and K. Ishikawa, "Smart: A tool for semantic-driven creation of complex xml mappings," Proc. ACM SIGMOD, pp.909–911, 2005.
[16] E. Rahm and P.A. Bernstein, "A survey of approaches to automatic schema matching," VLDB Journal, vol.10, no.4, pp.334–350, 2001.
[17] T. Schwentick, "Automata for xml — A survey," J. Comput. Syst. Sci., vol.73, no.3, pp.289–315, 2007.
[18] L.J. Stockmeyer and A.R. Meyer, "Word problems requiring exponential time," Proc. ACM STOC, pp.1–9, 1973.
[19] N. Suzuki, "An edit operation-based approach to the inclusion problem for dtds," Proc. ACM SAC, pp.482–488, 2007.
[20] Y. Velegrakis, R.J. Miller, and L. Popa, "Mapping adaption under evolving schemas," Proc. VLDB, pp.584–595, 2003.

Appendix A: Finding a Superscripted Supersequence of a Word w.r.t. the Insertion of a Superscripted Label

Let d be a DTD, $op = ins\_elm(a, b, vi)$ with $l(d(a), v) = \cdot$, and b^h be the superscripted label in $op(d)(a)$ inserted by op. Let $w \in L(d(a))$ be a word and consider finding a superscripted supersequence w'' of w w.r.t. the insertion of b^h. We use a function γ that maps each superscripted label in $d(a)'$ to its corresponding superscripted label in $(op(d))(a)'$. Let $w' \in L(d(a)')$ be a superscripted word such that $(w')^\natural = w$ and let $w'' = \gamma(w'[1]) \cdots \gamma(w'[|w'|])$. Then it suffices to insert b^h's into w'' at appropriate positions so that the resulting superscripted word matches $(op(d)(a))'$. Thus a superscripted supersequence of w w.r.t. the insertion of b^h can be obtained as follows.

1. Construct the Glushkov automaton $G_{d(a)}$ of $d(a)$.
2. By using $G_{d(a)}$, find a superscripted word w' such that $w' \in L(d(a)')$ and that $(w')^\natural = w$.
3. Let $w'' = \gamma(w'[1]) \cdots \gamma(w'[|w'|])$.
4. Construct the Glushkov automaton $G_{op(d)(a)} = (Q, \Sigma, \delta, q^I, F)$ of $op(d)(a)$. Do the following.
   a. If $w''[|w''|] \notin F$, then append b^h to w''.
   b. For each $i = |w'|, |w'| - 1, \cdots, 1$, if $w''[i] \notin Succ(w''[i-1], (op(d)(a))')$, then insert b^h between $w''[i-1]$ and $w''[i]$.
   c. If $w''[1] \notin Succ(q^I, (op(d)(a))')$, then insert b^h before the head of w''.
5. Return w''.

Example 8: Let $d(a) = *(+(a, \cdot(b, c)))$ and $op = ins\_elm(a, d, 122)$. Then $d(a)' = *(+(a^{11}, \cdot(b^{121}, c^{122})))$, $op(d)(a) = *(+(a, \cdot(b, d, c)))$, and $(op(d)(a))' = *(+(a^{11}, \cdot(b^{121}, d^{122}, c^{123})))$. Let $w = bcabc \in L(d(a))$, and consider finding a superscripted supersequence of w w.r.t. the insertion of d^{122}. In step 2, we obtain $w' = b^{121}c^{122}a^{11}b^{121}c^{122}$, thus in step 3 we have $w'' = \gamma(b^{121})\gamma(c^{122})\gamma(a^{11})\gamma(b^{121})\gamma(c^{122}) = b^{121}c^{123}a^{11}b^{121}c^{123}$. Consider step 4. It is easy to see that $c^{123} \notin Succ(b^{121}, (op(d)(a))')$, thereby one d^{122} is inserted between $w''[4]$ and $w''[5]$ and another d^{122} is inserted between $w''[1]$ and $w''[2]$. Hence we obtain $w'' = b^{121}d^{122}c^{123}a^{11}b^{121}d^{122}c^{123}$. □

It is easy to show that a superscripted supersequence of w w.r.t. the insertion of b^h can be found in $O(|op(d)(a)|^2 + |w|)$ time.

Appendix B: Finding a Variant of a Word

Let d be a DTD and $op = del\_opr(a, u)$ with $l(d(a), u) = *$.

Then $sub(d(a), u) = *(q)$ for some subexpression q of $d(a)$. For simplicity, assume that q' is a single superscripted label, say b^h. Let $w \in L(d(a))$ be a word and w' be a superscripted word such that $w' \in L(d(a)')$ and that $(w')^\natural = w$. Then a variant w'' of w' w.r.t. b^h such that $w'' \in L((op(d)(a))')$ can be found similarly to Appendix A. The only difference is that we also have to delete "excess" b^h's. More concretely, let $w'' = \gamma(w'[1]) \cdots \gamma(w'[|w'|])$. To obtain a desirable variant of w' w.r.t. b^h, we have to

v1) insert "missing" b^h's into w'' and
v2) delete "excess" b^h's from w''

so that the resulting superscripted word matches $(op(d)(a))'$. (v1) can be done similarly to step 4 of the method in Appendix A, and (v2) can be done by contracting each block of consecutive b^h's in w'' to single b^h.

Appendix C: Proof of Lemma 4

Let d be a limited DTD and $op = ins\_elm(a, b, vi)$ be an update operation to d such that $op(d)$ is limited. We show that for any word $w \in L(d(a))$, there is exactly one superscripted supersequence w'' of w w.r.t. the insertion of b^h such that $w'' \in L((op(d)(a))')$, where b^h is the superscripted label inserted by op.

Let $w \in L(d(a))$ be a word. Since $d(a)$ is one-unambiguous, there is exactly one superscripted word w' such that $w' \in L(d(a)')$ and that $(w')^\natural = w$. Let $G_{d(a)}$ be the Glushkov automaton of $d(a)$. Then $q^I = w'[0] \to w'[1] \to \cdots \to w'[|w'|]$ is the unique simple path in $G_{d(a)}$ representing w, where q^I is the initial state and $w'[|w'|]$ is a final state. By applying op to $d(a)$, b^h is inserted into $G_{d(a)}$. If for any $1 \le i \le |w'|$ b^h is not inserted between $w'[i-1]$ and $w'[i]$, then w' is the unique superscripted supersequence of w w.r.t. the insertion of b^h such that $w' \in L((op(d)(a))')$. Assume on the other hand that b^h is inserted between $w'[i-1]$ and $w'[i]$ for some i. Let γ be a function that maps each superscripted label in $d(a)'$ to its corresponding label in $op(d)(a)'$. Then $w'' = \gamma(w'[1]) \cdots \gamma(w'[i-1]) b^h \gamma(w'[i]) \cdots \gamma(w'[|w'|])$ is the unique superscripted supersequence of w w.r.t. the insertion of b^h such that $w'' \in L((op(d)(a))')$. Actually, (i) b^h occurs neither "optionally" nor "repeatedly" since $op(d)$ is limited, and (ii) b^h cannot occur between $w'[j-1]$ and $w'[j]$ for any $1 \le j \le |w'|$ with $j \ne i$ since $op(d)$ is limited and thus $w''[0] \to w''[1] \to \cdots \to w''[|w''|]$ must be a simple path in the Glushkov automaton of $op(d)(a)$.

Nobutaka Suzuki received his B.E. degree in information and computer sciences from Osaka University in 1993, and his M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology in 1995 and 1998, respectively. He was with Okayama Prefectural University as a Research Associate in 1998–2004. In 2004, he joined University of Tsukuba as an Assistant Professor. His current research interests include database theory and structured documents.

Yuji Fukushima received his B.E. degree in library and information science from University of Tsukuba in 2007. Currently, he is an M.C. student of Graduate School of Library, Information and Media Studies, University of Tsukuba. His current research interests are data transformation and XPath satisfiability.