

An XML Document Transformation Algorithm Inferred from an Edit Script between DTDs

Nobutaka Suzuki

Yuji Fukushima

Graduate School of Library, Information and Media Studies
University of Tsukuba
1-2, Kasuga, Tsukuba, Ibaraki 305-8550, Japan
Email: {nsuzuki@slis,s0721554@ipe}.tsukuba.ac.jp

Abstract

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

Keywords: XML, data transformation, edit operation, Glushkov automaton, 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 valid one against the updated DTD. Transforming each document manually is surely impractical, thereby constructing an appropriate transformation algorithm between original and updated DTDs is a greatly important problem.

In this paper, we propose a novel transformation approach based on an edit script between original and updated DTDs; assuming that the edit script applied to a DTD is known, we construct a transformation algorithm “inferred” from the DTD and the edit script. Here, an edit script to a DTD is a sequence of edit operations, where each edit operation inserts/deletes an element or an 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 edit 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 edit 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_1 be a DTD and S be a set of XML documents valid against D_1 . Suppose that a user updated D_1 to a new DTD D_2 by applying some edit script s to D_1 . Since s concretely represents the user’s intention how to modify D_1 , s strongly suggests how to transform each document in S . Therefore, if we can obtain a transformation algorithm T inferred from D_1 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 edit 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 a 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 edit script s that aggregates subexpression “(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 edit 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 edit operations to DTDs. Then, based on the edit operations we show a (possibly ambiguous) transformation algorithm inferred from a DTD and an edit script. Then we show a sufficient condition under which the transformation algorithm inferred from a DTD and an edit script is unambiguous. Finally, we show a polynomial-time algorithm for determining if, given a DTD D and an edit script s , the transformation algorithm inferred from D and s satisfies the sufficient condition.

Related Work

The problem of finding a mapping that identifies corresponding elements in two schemas (schema matching), the problem of finding a query that achieves actual data transformation between schemas (query discovery), and other related problems have greatly been studied, e.g., (Arenas & Libkin 2005, Bohannon et al. 2005, Kuikka et al. 2002, Miller et al. 2001, Milo

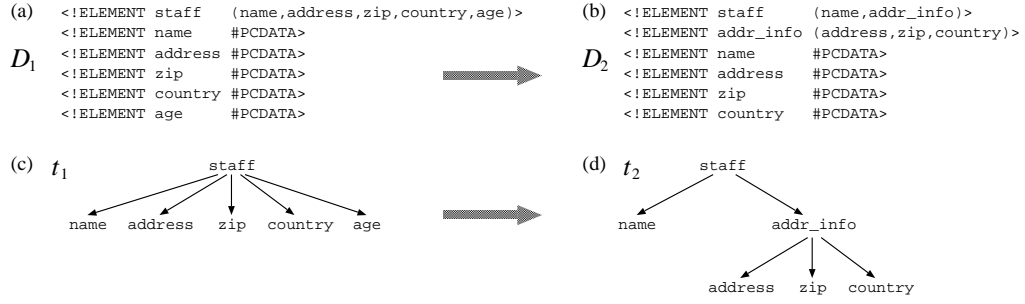


Figure 1: DTDs D_1, D_2 and XML documents t_1, t_2 .

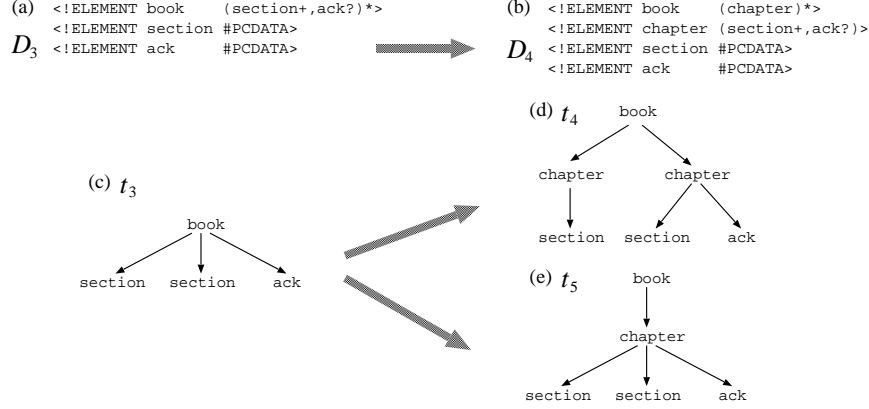


Figure 2: DTDs D_3, D_4 and XML documents t_3, t_4, t_5 .

& Zohar 1998, Morishima et al. 2005, Rahm & Bernstein 2001). These studies consider finding an appropriate matching or transformation between schemas, assuming that no edit script between the schemas is known. On the other hand, our aim is to construct a transformation algorithm inferred from an edit script between two schemas assuming that an edit script between the schemas is available.

Several studies propose edit operations to schemas. Ref. (Leonardi et al. 2006) proposes edit operations to represent the “diff” between two DTDs. Ref. (Hashimoto et al. 2007) proposes edit operations to tree grammars preserving schema’s expressive power; any updated grammar admits only trees to which trees valid against its original grammar are embeddable. Refs. (Guerrini et al. 2005, Suzuki 2007) propose edit operations such that any updated schema always includes its original schema.

To define a condition under which our transformation algorithm becomes unambiguous, we need some kind of unambiguity of regular expression. Previously, unambiguity and one-unambiguity of regular expression have been studied (Book et al. 1971, Brüggemann-Klein & Wood 1998). We use a weaker unambiguity w.r.t. subexpression of regular expression. In short, if a regular expression r is unambiguous w.r.t. a subexpression q of r , then for any word w matching r , the positions within w that match q can unambiguously be determined.

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, each leaf node has implicitly has a text node. By $l(n)$ we mean the label 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 edit 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, \dots, r_n are regular expressions, then $\cdot(r_1, \dots, r_n)$ and $+(r_1, \dots, r_n)$ 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 *occurrences (or positions)* of r , denoted $occ(r)$, is defined as follows.

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

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 occurrence. Thus $occ(r) = \{\lambda, 1, 2, 11, 12, 13, 21\}$.

Let $u \in occ(r)$. 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 $sub(r, \lambda) = r$.
- If $r = op(r_1, \dots, r_n)$ with $op \in \{+, \cdot\}$, and

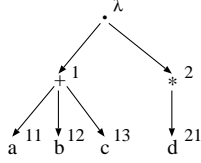


Figure 3: Tree representation of $\cdot+(a, b, c), *(d)$.

- if $u = \lambda$, then $l(r, u) = op$ and $sub(r, u) = r$,
- if $u = jv$ for some $1 \leq j \leq n$ and some $v \in occ(r_j)$, then $l(r, u) = l(r_j, v)$ and $sub(r, u) = sub(r_j, v)$.

- If $r = *(r_1)$, and

- if $u = \lambda$, then $l(r, u) = '*'$ and $sub(r, u) = r$,
- if $u = 1v$ for some $v \in occ(r_1)$, then $l(r, u) = l(r_1, v)$ and $sub(r, u) = sub(r_1, v)$.

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

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

A DTD is a tuple $D = (d_1, a_0)$, where d_1 is a (possibly partial) mapping from Σ to the union of the set of regular expressions over Σ and $\{\#PCDATA\}$, and $a_0 \in \Sigma$ is the *start label*. For a label $a \in \Sigma$, $d_1(a)$ is the *content model* of a . A tree t is *valid* against D if (v1) the root of t is labeled by a_0 , (v2) for each internal node n in t the sequence of labels on the children of n is in $L(d_1(l(n)))$, and (v3) for each leaf node n in t $d_1(l(n)) = \#PCDATA$. 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 . 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 . D is *cyclic* if for some labels a_i, a_j a_i is reachable from a_j and vice versa.

Let $D = (d_1, a_0)$ be a DTD. A *rootless* DTD is a pair (d_1, r) , where r is a regular expression over Σ . A forest t is *valid* against (d_1, r) if (v1') the sequence of the labels on the roots of t is in $L(r)$ and each internal node and each leaf node in t satisfy Conditions (v2) and (v3) above, respectively.

Let r be a regular expression. By r' we mean the *subscripted* regular expression resulting from r by subscripting each label in r by the corresponding occurrence. By $sym(r')$ we mean the set of subscripted 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 subscripted label of a . Then by a_i^\natural we mean the label resulting from a_i by dropping the subscript of a_i , that is, $a_i^\natural = a$. Let w_1 be a subscripted word (i.e., a sequence of subscripted labels). We define that $w_1^\natural = w_1[1]^\natural \cdots w_1[|w_1|]^\natural$. For any regular expression r , it holds that $L(r) = L(r')^\natural$, where $L(r')^\natural = \{w_1^\natural \mid w_1 \in L(r')\}$.

3 Edit Operations to DTD

In this section, we define edit operations to a DTD $D = (d_1, a_0)$. There are three types of edit operations; the edit operations of types 1 and 2 are to modify existing content models, and that of type 3 is to declarer a new content model.

Type 1a: Inserts/deletes an element in a regular expression $d_1(a)$.

- $ins_elm(a, b, vi)$: Inserts a new label b at vi in $d_1(a)$, where $v \in occ(d_1(a))$, i is an integer, and $b \in \Sigma \cup \{\epsilon\}$ (Fig. 4(c,d)). This is applicable to $d_1(a)$ only if $d_1(b)$ is defined, $l(d_1(a), v)$ is '+' or '.', and $v(i-1) \in occ(d_1(a))$ (i.e., the operator at v has at least $i-1$ operands).
- $del_elm(a, vi)$: Deletes label $l(d_1(a), vi)$ from $d_1(a)$ (Fig. 4(a,b)). This is applicable to $d_1(a)$ only if $sub(d_1(a), vi)$ is a label in Σ and $vk \in occ(d_1(a))$ for some $k \neq i$ (i.e., $l(d_1(a), vi)$ has at least one siblings).

Type 1b: (i) Aggregates a subexpression of $d_1(a)$ into a new label, or (ii) extract a label in $d_1(a)$, say b , by regular expression $d_1(b)$.

- $agg_elm(a, b, u)$: Aggregates subexpression $sub(d_1(a), u)$ into a single label b . Formally, this operation first replaces a subexpression $sub(d_1(a), u)$ by label b , then sets $d_1(b) = sub(d_1(a), u)$ (Fig. 4(d,e)). This is applicable to $d_1(a)$ only if $d_1(b)$ is undefined.
- $ext_elm(a, u)$: Extracts a label $l(d_1(a), u)$ in $d_1(a)$. Formally, this operation replaces a label $l(d_1(a), u)$ by a regular expression $d_1(l(d_1(a), u))$ (Fig. 4(e,f)). This is applicable to $d_1(a)$ only if $sub(d_1(a), u)$ is a label in Σ and $a \neq l(d_1(a), u)$.

Type 2: Inserts/deletes an operator ('+', '.', or '*') in $d_1(a)$.

- $ins_opr(a, opr, vi, vj)$: Inserts a new operator opr as the parent of the subexpressions at vi, \dots, vj in $d_1(a)$, where opr is '+', '.', or '*' and $i \leq j$ (Fig. 4(b,c)). Moreover, it must hold that if $i \neq j$ (i.e., opr has more than one operand), then $opr = l(d_1(a), v)$ (i.e., opr is "nested") and $opr \in \{'+', '\cdot'\}$.
- $del_opr(a, u)$: Deletes an operator at position u in $d_1(a)$ (Fig. 4(f,g)). This is applicable to $d_1(a)$ only if (i) $l(d_1(a), v) = l(d_1(a), u)$ with $u = vi$ (the operator at u is "nested") or (ii) $u1 \in occ(d_1(a))$ and $u2 \notin occ(d_1(a))$ (the operator at u has exactly one operand).

Type 3: Defines a new content model.

- $def_cm(a, r)$: Sets $d_1(a) = r$. This is applicable only if $d_1(a)$ is undefined.

Let op be an edit operation to a DTD D . By $op(D)$ we mean the DTD obtained by applying op to D . Let $s = op_1 op_2 \cdots op_n$ be a sequence of edit operations ($n \geq 0$). s is *applicable* to D if op_i is applicable to $op_{i-1}(op_{i-2}(\cdots op_1(D) \cdots))$ for every $1 \leq i \leq n$. s is an *edit script* to D if s is applicable to D . For an edit script $s = op_1 op_2 \cdots op_n$ to D , we define that $s(D) = op_n(op_{n-1}(\cdots op_1(D) \cdots))$. An edit script of length zero is denoted ϵ .

Example 1 Let $D = (d_0, staff)$ be a DTD, where $d_0(staff) = \cdot(name, age, zip, email)$, $d_0(name) = \cdot(firstname, lastname)$, and the other elements are of type $\#PCDATA$. Let $s = op_1 \cdots op_6$, where $op_1 = del_elm(staff, 2)$, $op_2 = ins_opr(staff, \cdot, 2, 2)$, $op_3 = ins_elm(staff, street, 21)$, $op_4 = agg_elm(staff, address, 2)$, $op_5 = ext_elm(staff, 1)$, and $op_6 = del_opr(staff, 1)$. Then D and $D_i = op_i(op_{i-1}(\cdots op_1(D) \cdots))$ for $1 \leq i \leq 6$ are illustrated in Fig. 4. \square

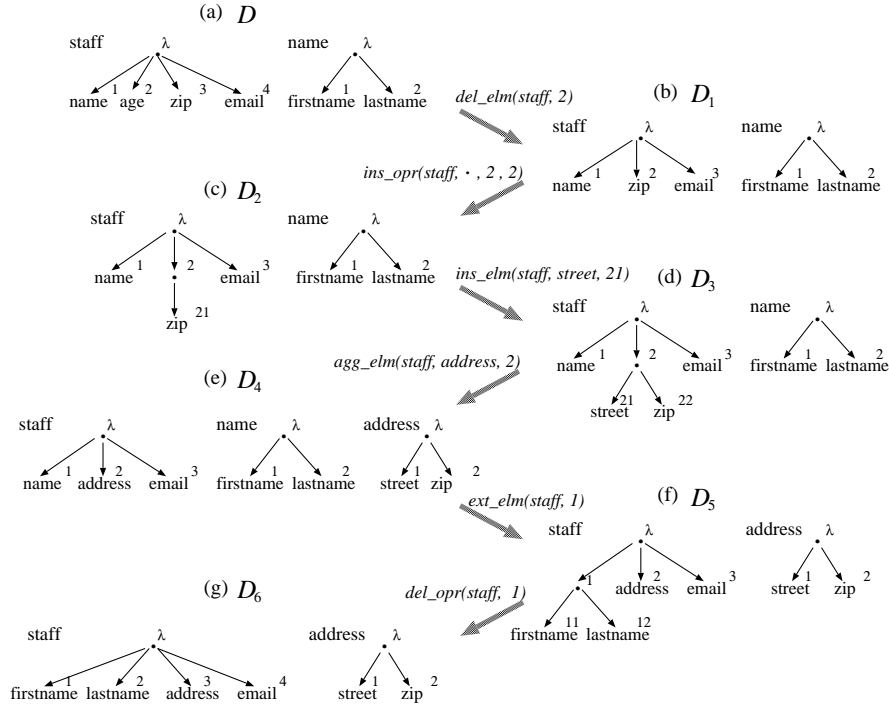


Figure 4: An edit script to a DTD D .

We have the following lemma.

Lemma 1 *Let $D = (d_1, a_0)$ be a DTD and op be an edit operation to D . Then $op(D)$ includes D iff op satisfies one of the following conditions.*

1. $op = ins_elm(a, b, vi)$ and either (i) $l(d_1(a), v) = '+'$ (inserting a label into a '+'-expression) or (ii) $l(d_1(a), v) = \cdot$ and $\epsilon \in L(d_1(b))$ (inserting an ϵ -able expression into a \cdot -expression).
2. $op = del_elm(a, vi)$ and either (i) $l(d_1(a), v) = '+'$ and $l(d_1(a), vi) \in L(sub(d_1(a), vk))$ for some $k \neq i$ (i.e., the element at vi is contained in some sibling subexpression) or (ii) $l(d_1(a), v) = \cdot$ and $l(d_1(a), vi) = \epsilon$.
3. $op = ext_elm(a, u)$ and $l(d_1(a), u) \in L(d_1(l(d_1(a), u)))$.
4. $op = ins_opr(a, opr, vi, vj)$, where opr is '+', \cdot , or '*'.
5. $op = del_opr(a, u)$ and either (i) $l(d_1(a), u)$ is '+' or \cdot , or (ii) $l(d_1(a), u) = *$ and $L(sub(d_1(a), u1)) = \{\epsilon\}$.
6. $op = def_cm(a, r)$. □

For example, $ins_opr(staff, 2, 2)$ and $del_opr(staff, 1)$ in Fig. 4 satisfy Condition (4) of the above lemma.

4 Transformation Algorithm Inferred from a DTD and an Edit Script

In this section, we show a (possibly ambiguous) transformation algorithm inferred from a DTD and an edit script.

4.1 Outline

Let us first show an outline of our transformation algorithm. Let $D = (d_1, a_0)$ be a DTD and op be an edit operation. For a tree t valid against D , our transformation algorithm T inferred from D and op transforms t as follows.

1. If op satisfies one of Conditions (1) to (6) of Lemma 1, then t is valid against $op(D)$. Thus 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 $d_1(a)$. Accordingly, for each position p in t at which the b -label should be inserted, T creates a new valid tree¹ whose root is labeled by b and insert the tree at position p in t . For example, if $d_1(a) = \cdot(a, b)$ 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 $d_1(a)$ is deleted. Accordingly, T first identifies the subtrees in t whose roots match the label b . If the parent of b is \cdot in $d_1(a)$, then T just deletes the identified subtrees. If the parent of b is '+' in $d_1(a)$, then T replaces each identified subtree by a new valid tree whose root is labeled by c , where c is a sibling of b in $d_1(a)$. For example, if $d_1(a) = +(b, c)$ and $op = del_elm(a, 1)$, then each subtree in t whose root is labeled by b is replaced by a new valid tree whose root is labeled by c .

Type 1b: (1) If $op = ext_elm(a, u)$, then T identifies the internal nodes in t that match the extracted label in $d_1(a)$, and deletes the identified internal nodes from t . For example, if $d_1(a) = \cdot(b, c)$ 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 internal nodes into t as the parents of sibling nodes that should be aggregated. For example, if $d_1(a) =$

¹We assume that the text values of such a new tree are empty since they can hardly be estimated.

$+(\cdot(b, c), d)$ and $op = agg_elm(a, e, 1)$, then for each siblings labeled by b and c in t a new node labeled by e is inserted as the parent of the siblings.

Type 2: By Conditions (4) and (5) of Lemma 1, $op = del_opr(a, vi)$ and $sub(d_1(a), vi) = *(q)$ for some regular expression q (i.e., op deletes the ‘ $*$ ’ from $*(q)$). Thus T identifies the nodes in t that match the subexpression $*(q)$ and deletes “excess” subtrees from the subtrees rooted at the identified nodes (since $d_1(a)$ admits arbitrary repetitions of q but $d_2(a)$ does not, where $op(D) = (d_2, a_0)$) and may add new trees to t (since $*(q)$ matches ϵ but q may not).

Let $s = op_1 \cdots op_n$ be an edit script to D and $D_{i-1} = op_{i-1}(\cdots(op_1(D))\cdots)$, where op_i is an edit operation. Then the transformation algorithm inferred from D and s is defined as a composition of T_1, \cdots, T_n , where T_i is the transformation algorithm inferred from D_{i-1} and op_i .

4.2 The Transformation Algorithm

We first show some definitions. Let r be a regular expression, $u \in occ(r)$, $q = sub(r, u)$ be a subexpression of r , w be a word such that $w \in L(r)$, and w_1 be a subscripted word such that $w_1 \in L(r')$ and that $w_1^{\sharp} = w$. We say that $w_1[i, j]$ *maximally matches* q' if $w_1[i, j] \in L(q')$ and either (i) $i = 1$ and $j = |w_1|$ or (ii) $w_1[i', j'] \notin L(q')$ for any i', j' ($1 \leq i' \leq i \leq j \leq j' \leq |w_1|$) with $i' < i$ or $j < j'$. We define that

$$match(w_1, q') = \{(i, j) \mid w_1[i, j] \text{ maximally matches } q'\}.$$

For example, let $r = *(.(a, +(b, c)))$ and $q = +(b, c)$. Then $r' = *(.(a_{11}, +(b_{121}, c_{122})))$ and $q' = +(b_{121}, c_{122})$. If $w_1 = a_{11}b_{121}a_{11}c_{122}$, then $match(w_1, q') = \{(2, 2), (4, 4)\}$.

Let w be a word and b_h be a subscripted label. We say that a subscripted word w_1 is a *subscripted supersequence* of w w.r.t. b_h if removing every b_h from w_1 yields a word w_2 such that $w_2^{\sharp} = w$.

Let $D = (d_1, a_0)$ be a DTD, op be an edit operation to D , and $op(D) = (d_2, a_0)$. We show the transformation algorithm inferred from D and op , denoted $TRANSOP_{D,op}$, as follows (subroutine $TRANS1A$, $TRANS1B$, and $TRANS2$ are shown later).

$TRANSOP_{D,op}(t)$

Input: a tree t valid against D .

Output: a tree valid against $op(D)$.

1. If op satisfies one of Conditions (1) to (6) of Lemma 1, then return t .
2. Otherwise, do the following.
 - (a) If op is of type 1a, then return $TRANS1A_{D,op}(t)$.
 - (b) If op is of type 1b, then return $TRANS1B_{D,op}(t)$.
 - (c) If op is of type 2, then return $TRANS2_{D,op}(t)$.

Let us show three subroutines $TRANS1A$, $TRANS1B$, and $TRANS2$. We first show $TRANS1A$.

$TRANS1A_{D,op}(t)$

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, $l(d_1(a), v) = \cdot$, i.e., b must be inserted into a ‘ \cdot ’-subexpression.

- (a) Let n_1, \cdots, n_m be the children of n in t . Find a subscripted supersequence w_1 of $l(n_1) \cdots l(n_m)$ w.r.t. b_h such that $w_1 \in L(d_2(a)'),$ where b_h is the subscripted label in $d_2(a)$ inserted by op .

- (b) For each $(j, j) \in match(w_1, b_h)$, create a new tree valid against a DTD (d_2, b) and insert the tree into t as the j th 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 subscripted word w_1 such that $w_1 \in L(d_1(a)'),$ and that $w_1^{\sharp} = l(n_1) \cdots l(n_m)$.

- (b) By definition $sub(d_1(a), vi)'$ is a single label, say b_h . Thus for each $(j, j) \in match(w_1, b_h)$, do the following.

- i. If $l(d_1(a), v) = \cdot$, then delete the subtree rooted at n_j from t .

- ii. If $l(d_1(a), v) = \cdot+$, then choose a “sibling” $q = sub(d_1(a), vk)$ of $l(d_1(a), vi)$ such that $k \neq i$. Then create a new forest valid against a (possibly rootless) DTD (d_2, q) , and replace the subtree rooted at n_j by the new forest.

3. Return t transformed above.

In step (1a) we have to find a subscripted supersequence w_1 such that $w_1 \in L(d_2(a)'),$ By using Glushkov automata (defined in Sect. 6.2), w_1 can be obtained in $O(|d_2(a)|^2 + |w_1|)$ time (details are omitted because of space limitation). A subscripted word w_1 such that $w_1 \in L(d_1(a)'),$ in step (2a) can also be obtained by using a Glushkov automaton.

Example 2 Figure 5 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. Here, let us consider the intermediate transformations from t_0 to t_1 and t_2 to t_3 in Fig. 5.

$(t_0 \Rightarrow t_1)$ Let $D = (d_0, staff)$ be the DTD in Fig. 4(a). Then $d_0(staff) = \cdot(\text{name, age, zip, email})$. Since $op_1 = del_elm(staff, 2)$, t_0 is transformed by step 2 of $TRANS1A$. Consider the node n_1 of t_0 . Since $d_0(staff)' = \cdot(\text{name}_1, \text{age}_2, \text{zip}_3, \text{email}_4)$, the subscripted word w_1 of $l(n_2)l(n_3)l(n_4)l(n_5)$ = “name age zip email” such that $w_1 \in d_0(staff)'$ is “name₁ age₂ zip₃ email₄”, thereby $match(w_1, \text{age}_2) = \{(2, 2)\}$. Thus the second child n_3 of n_1 is deleted from t_0 .

$(t_2 \Rightarrow t_3)$ Let $D_2 = (d_2, staff)$ be the DTD in Fig. 4(c). Then $d_2(staff) = \cdot(\text{name}, \cdot(\text{zip}), \text{email})$. Since $op_3 = ins_elm(staff, street, 21)$, we have $d_3(staff) = \cdot(\text{name}, \cdot(\text{street}, \text{zip}), \text{email})$ and t_2 is transformed by step 1 of $TRANS1A$. Consider the node n_1 in t_2 . Since $d_3(staff)' = \cdot(\text{name}_1, \cdot(\text{street}_{21}, \text{zip}_{22}), \text{email}_3)$, the subscripted supersequence w_1 of $l(n_2)l(n_4)l(n_5) = \text{“name zip email”}$ w.r.t. $street_{21}$ such that $w_1 \in L(d_3(staff)')$ is “name₁ street₂₁ zip₂₂ email₃”, thereby $match(w_1, street_{21}) = \{(2, 2)\}$. Thus a new node, say n_8 , labeled by “street” is inserted into t_2 as the second child of n_1 . \square

Let us next show $TRANS1B$ that is called if op is of type 1b.

$TRANS1B_{D,op}(t)$

1. If $op = ext_elm(a, u)$, then for each node n labeled by a in t , do the following.

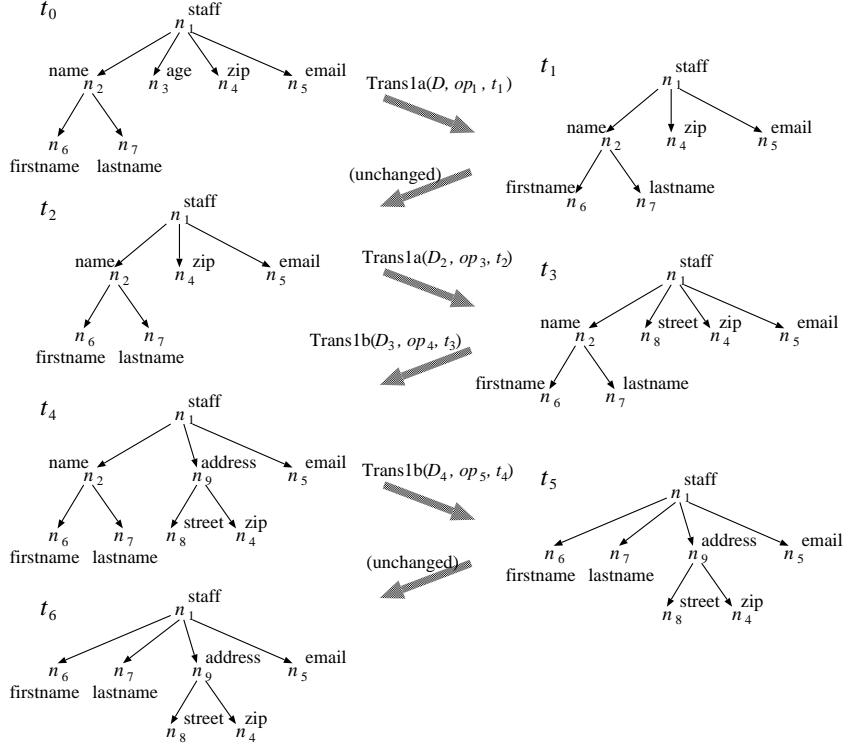


Figure 5: An example of transformation according to the edit script in Fig. 4.

- (a) Let n_1, \dots, n_m be the children of n in t . Find a subscripted word w_1 such that $w_1 \in L(d_1(a)')$ and that $w_1^{\natural} = l(n_1) \dots l(n_m)$.
 - (b) By definition $sub(d_1(a), u')$ is a single subscripted label, say b_h . For each $(j, j) \in match(w_1, b_h)$, delete the j th 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, \dots, n_m be the children of n in t . Find a subscripted word w_1 such that $w_1 \in L(d_1(a)')$ and that $w_1^{\natural} = l(n_1) \dots l(n_m)$.
 - (b) For each $(j, k) \in match(w_1, sub(d_1(a), u'))$, insert a new node labeled by b as the parent of n_j, \dots, n_k into t .
 3. Return t transformed above.

Example 3 Let us consider the intermediate transformations from t_3 to t_4 and from t_4 to t_5 in Fig. 5.

($t_3 \Rightarrow t_4$) Let $D_3 = (d_3, staff)$ be the DTD in Fig. 4(d). Then $d_3(staff) = \cdot(name, \cdot(street, zip), email)$. Since $op_3 = agg_elm(staff, address, 2)$, we have $d_4(staff) = \cdot(name, address, email)$ and $d_4(address) = \cdot(street, zip)$, and t_3 is transformed by step 2 of TRANS1B. Consider the node n_1 in t_3 . Since $d_3(staff)' = \cdot(name_1, \cdot(street_{21}, zip_{22}), email_3)$, the subscripted word w_1 of $l(n_2)l(n_8)l(n_4)l(n_5) = \text{“name street zip email”}$ such that $w_1 \in L(d_3(staff)')$ is $\text{“name}_1 \text{street}_{21} \text{zip}_{22} \text{email}_3$ ”. Thus $match(w_1, \cdot(street_{21}, zip_{22})) = \{(2, 3)\}$. Therefore, a new node, say 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 $D_4 = (d_4, staff)$ be the DTD in Fig. 4(e). Then $d_4(staff) = \cdot(name, address, email)$. Since $op_4 = ext_elm(staff, 1)$, t_4 is transformed by step 1 of TRANS1B. Consider the node n_1 in t_4 . Since $d_4(staff)' = \cdot(name_1, address_2, email_3)$, the subscripted word

w_1 of $l(n_2)l(n_9)l(n_5) = \text{“name address email”}$ such that $w_1 \in L(d_4(staff)')$ is $\text{“name}_1 \text{address}_2 \text{email}_3$ ”, thereby $match(w_1, name_1) = \{(1, 1)\}$. Thus the first child n_2 of n_1 is deleted. \square

Finally, we show TRANS2. We need a definition. Let w_1 be a subscripted word and b_h be a subscripted label. Then a *variant* of w_1 w.r.t. b_h is a subscripted word obtained by deleting some b_h 's from w_1 and inserting b_h 's into w_1 at arbitrary positions.

TRANS2 $_{D,op}(t)$

1. By Conditions (4) and (5) of Lemma 1, $op = del_opr(a, vi)$ and $l(d_1(a), vi) = \text{“*”}$. Thus $sub(d_1(a), vi) = \text{“*(q)”}$ for some subexpression q and this “*” is deleted by op . For simplicity, we assume that q is a single label b (the other cases 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, \dots, n_m be the children of n in t . Find a subscripted word w_1 such that $w_1 \in L(d_1(a)')$ and that $w_1^{\natural} = l(n_1) \dots l(n_m)$.
 - (b) Find a variant w_2 of w_1 w.r.t. b_h such that $w_2 \in L(d_2(a)')$. For each node n_j corresponding to an occurrence of b_h deleted from w_1 , delete the subtree rooted at n_j from t . For each occurrence of b_h inserted into w_1 , create a new tree t_b valid against DTD (d_2, 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.

Variant w_2 in step (1b) can be obtained in $O(|d_2(a)|^2 + |w_2|)$ time by using a Glushkov automaton (details are omitted because of space limitation).

Let us now define the transformation algorithm inferred from a DTD and an edit script. Let D be a DTD and $s = op_n \dots op_1$ be an edit script to D .

The transformation algorithm inferred from D and s , denoted $\text{TRANSFORM}_{D,s}(t)$, is defined as follows.

$\text{TRANSFORM}_{D,s}(t)$

Input: a tree t valid against D .

Output: a tree valid against $s(D)$.

1. If $s = \epsilon$, then return t .
2. Otherwise, let $D_{i-1} = \text{op}_{i-1}(\dots(\text{op}_1(D))\dots)$ and $s_{i-1} = \text{op}_{i-1} \dots \text{op}_1$. Return $\text{TRANSOP}_{D_{i-1}, \text{op}_i}(\text{TRANSFORM}_{D, s_{i-1}}(t))$.

Note that $\text{TRANSFORM}_{D,s}(t)$ outputs a single tree but it may not be unique. Let $TS_{D,s}(t) = \{t' \mid t' \text{ can be the result of } \text{TRANSFORM}_{D,s}(t)\}$. 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 sections).

It is clear that $\text{TRANSFORM}_{D,s}(t)$ is “correct”.

Theorem 1 *Let D be a DTD and s be an edit script s to D . Then for any tree t valid against D and any $t' \in TS_{D,s}(t)$, t' is valid against $s(D)$.* \square

Let $D = (d_1, a_0)$ be a DTD and $s = \text{op}_1 \dots \text{op}_n$ be an edit script to D . Finally, let us consider the running time of $\text{TRANSFORM}_{D,s}(t)$. If $\text{op}_i = \text{ins_elm}(a, b, u)$ and for some prefix v of u $l(d_1(a), v) = '*'$, then op_i is *starred*. If for some i, j with $i \leq j$ $\text{op}_i = \text{ins_elm}(a, b, u)$, $\text{op}_j = \text{ins_elm}(c, d, v)$, and c occurs in $d_{j-1}(b)$, then op_j is *nested*, where $(d_{j-1}, a_0) = \text{op}_{j-1}(\dots \text{op}_1(D) \dots)$. $\text{TRANSFORM}_{D,s}(t)$ runs in polynomial time with some exceptions.

Theorem 2 *Let D be a DTD, $s = \text{op}_1 \dots \text{op}_n$ be an edit script to D , and t be a tree valid against D . If the following conditions hold, then $\text{TRANSFORM}_{D,s}(t)$ runs in $O(n^3 \cdot |t| \cdot |D|^3)$ time, where $|t|$ is the number of nodes in t and $|D|$ is the size of D .*

- The number of $\text{ins_elm}()$ operations in s that are nested or starred is bounded by some constant.

\square

In short, for each $\text{ins_elm}()$ operation in s , if the above condition does not hold, then the size of t can be doubled. Thus the size of the resulting tree of $\text{TRANSFORM}_{D,s}(t)$ can be exponential without the above condition.

5 A Sufficient Condition for Unambiguous Transformation

In this section, we show a sufficient condition under which, for a DTD D and an edit script s , the transformation algorithm inferred from D and s is unambiguous. We will show in the next section how to check the sufficient condition.

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

- U1) In step (1b) of TRANS1A, $\text{match}(w_1, b_h)$ depends on the subscripted supersequence w_1 selected in step (1a).
- U2) In step (1b) of TRANS1A, there may be more than one trees valid against (d_2, b) . Similarly, in step (2b-ii) of TRANS1A there may be more than one forests valid against (d_2, q) .
- U3) In step (2b) of TRANS1A, $\text{match}(w_1, b_h)$ depends on subscripted word w_1 selected in step (2a). A similar argument also applies to steps (1b) and (2b) of TRANS1B.

U4) In step (2b-ii) of TRANS1A, there may be more than one siblings $\text{sub}(d_1(a), vk)$ of $l(d_1(a), vi)$ such that $vk \in \text{occ}(d_1(a))$ and that $k \neq i$.

U5) In step (1b) of TRANS2, there may be more than one variant w_2 of w_1 .

We give definitions related to (U1) to (U3). Consider first (U1). We define a regular expression that admits only subscripted supersequences such that the positions at which b_h should be inserted can unambiguously be determined. Let $D = (d_1, a_0)$ be a DTD, $\text{op} = \text{ins_elm}(a, b, vi)$ be an edit operation to D , $\text{op}(D) = (d_2, a_0)$, and b_h be the subscripted label in $d_2(a)'$ inserted by op . We say that $d_1(a)$ is *unambiguous* w.r.t. the insertion of b_h if for any word $w \in L(r)$, any subscripted supersequences w'_1, w'_2 of w w.r.t. b_h such that $w'_1, w'_2 \in L(d_2(a))$,

- $|w'_1| = |w'_2|$, and
- for any $1 \leq k \leq |w'_1|$, $w'_1[k] = b_h$ iff $w'_2[k] = b_h$.

Example 4 Let $d_1(a) = *(+(a, b, \cdot(b)))$, $\text{op} = \text{ins_elm}(a, a, 131)$, and $ab \in L(d_1(a))$. Then $d_2(a)' = *(+(a_{11}, b_{12}, \cdot(a_{131}, b_{132})))$. Both $a_{11}b_{12}$ and $a_{11}a_{131}b_{132}$ are subscripted supersequences of ab w.r.t. a_{131} belonging to $L(d_2(a)')$. Thus $d_1(a)$ is not unambiguous w.r.t. the insertion of a_{131} . \square

Consider next (U2). We define a DTD that admits exactly one valid tree. We say that a DTD $D = (d_1, a_0)$ is *simple* if D is acyclic and for any label a reachable from a_0 , $|L(d_1(a))| = 1$. For example, let $D = (d_1, a)$ be a DTD such that $d_1(a) = \cdot(b, c, d)$, $d_1(b) = \cdot(e, f)$, and that $d_1(c) = d_1(d) = d_1(e) = d_1(f) = \#\text{CDATA}$. Then D is simple. Similarly, a rootless DTD (d_2, r) is *simple* if $|L(r)| = 1$ and for any label a_1 appearing in r , (d_2, a_1) is simple. We have the following lemma.

Lemma 2 (1) *For any DTD D , there is exactly one tree valid against D iff D is simple.* (2) *For any rootless DTD D' , there is exactly one forest valid against D' iff D' is simple.* \square

Then consider (U3). We define a regular expression such that $\text{mach}(w_1, b_h)$ can unambiguously be determined. Let r be a regular expression, $u \in \text{occ}(r)$ be an occurrence, and $w \in L(r)$ be a word. We say that r is *unambiguous* w.r.t. $\text{sub}(r, u)$ and w if for any subscripted words w_1, w_2 such that $w_1^{\sharp} = w_2^{\sharp} = w$ and that $w_1, w_2 \in L(r')$, $\text{match}(w_1, \text{sub}(r, u)') = \text{match}(w_2, \text{sub}(r, u)')$. We say that r is *unambiguous* w.r.t. $\text{sub}(r, u)$ if r is *unambiguous* w.r.t. $\text{sub}(r, u)$ and w for any $w \in L(r)$.

Example 5 Let $d_1(a) = \cdot(*(b), *(+(b, c)))$. Then $d_1(a)' = \cdot(*(b_{11}), *(+(b_{211}, c_{212})))$. For a word $bc \in L(d_1(a))$, we have two subscripted words $w_1 = b_{11}c_{212}$ and $w_2 = b_{211}c_{212}$ such that $w_1^{\sharp} = w_2^{\sharp} = bc$ and that $w_1, w_2 \in L(d_1(a)')$. Since $\text{match}(w_1, *(+(b_{211}, c_{212}))) = \{(2, 2)\}$ and $\text{match}(w_2, *(+(b_{211}, c_{212}))) = \{(1, 2)\}$, $d_1(a)$ is not unambiguous w.r.t. $\text{sub}(d_1(a), 2) = *(+(b, c))$. \square

The following lemma shows a necessary and sufficient condition under which the transformation algorithm inferred from a DTD and an edit operation is unambiguous.

Lemma 3 *Let $D = (d_1, a_0)$ be a DTD, op be an edit operation to D , and $\text{op}(D) = (d_2, a_0)$. Then the transformation algorithm inferred from D and op is unambiguous iff one of the following five conditions holds.*

S1) One of Conditions (1) to (6) of Lemma 1 holds.

S2) $op = ins_elm(a, b, vi)$, $d_1(a)$ is unambiguous w.r.t. the insertion of b_h , and DTD (d_2, b) is simple, where b_h is the subscripted label in $d_2(a)$ inserted by op .

S3) $op = del_elm(a, vi)$, $d_1(a)$ is unambiguous w.r.t. $l(d_1(a), vi)$, and the following conditions hold in case of $l(d_1(a), v) = '+'$.

- The siblings of $l(d_1(a), vi)$ are equivalent, that is, for any $vj, vk \in occ(d_1(a))$ with $j \neq i$ and $k \neq i$, $L(sub(d_1(a), vj)) = L(sub(d_1(a), vk))$.
- For any $vj \in occ(d_1(a))$ with $j \neq i$, (possibly rootless) DTD $(d_2, sub(d_1(a), vj))$ is simple.

S4) $op = ext_elm(a, u)$ and $d_1(a)$ is unambiguous w.r.t. $l(d_1(a), u)$.

S5) $op = agg_elm(a, b, u)$ and $d_1(a)$ is unambiguous w.r.t. $sub(d_1(a), u)$.

Proof (sketch): It is easy to verify that if none of Conditions (S1) to (S5) holds, then $|TS_{D,op}(t)| > 1$ for some tree t valid against D . Let $op = ins_elm(a, b, vi)$ and assume that Condition (S2) holds (the other cases can be shown similarly). By Condition (S2) $d_1(a)$ is unambiguous w.r.t. the insertion of b_h , where b_h is the subscripted label inserted by op . This implies that in step (1b) of TRANS1A the positions of the occurrences of b_h in w_1 are independent of which subscripted word is selected in step (1a) of TRANS1A. Hence $|TS_{D,op}(t)| = 1$. \square

We now show a sufficient condition under which the transformation algorithm inferred from a DTD and an edit script is unambiguous (whether this sufficient condition is necessary is open).

Theorem 3 Let D be a DTD and $s = op_1 \cdots op_n$ be an edit script to D . Then the transformation algorithm inferred from D and s is unambiguous if for every $1 \leq i \leq n$ the transformation algorithm inferred from D_{i-1} and op_i satisfies one of Conditions (S1) to (S5), where $D_{i-1} = op_{i-1}(\cdots(op_1(D))\cdots)$. \square

6 Checking the Unambiguity of Transformation Algorithm

In this section, we show how to decide if, given a DTD D and an edit script s to D , the transformation algorithm inferred from D and s satisfies the condition in Theorem 3. We have to solve the following problems.

1. In Conditions (S2) and (S3), we have to determine if a DTD is simple.
2. In Conditions (S3), (S4), and (S5), we have to determine if a regular expression $d_1(a)$ is unambiguous w.r.t. a subexpression of $d_1(a)$.
3. In Condition (S2), we have to determine if a regular expression $d_1(a)$ is unambiguous w.r.t. the insertion of b_h .

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

6.1 Checking the Simplicity of a DTD

Recall that DTD $D = (d_1, a_0)$ is *simple* if D is acyclic and for any label a reachable from a_0 , $|L(d_1(a))| = 1$. First, whether D is acyclic can be determined in linear time. Second, 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. Thus, we have the following.

Lemma 4 For a DTD D , whether D is simple can be determined in $O(|D|^2)$ time. \square

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

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 (Book et al. 1971, Brüggemann-Klein & Wood 1998). First, the *initial set* I_r and the *final set* F_r are defined as follows.

1. If $r = \epsilon$, then $I_r = F_r = \{E\}$, where E is a new label.
2. If $r = a$ for some $a \in \Sigma$, then $I_r = F_r = \{a_i\}$, where a_i is the subscripted 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

$$\begin{aligned} I_r &= (I_{r_1} - \{E\}) \cup \cdots \cup (I_{r_{i-1}} - \{E\}) \cup I_{r_i}, \\ F_r &= F_{r_j} \cup (F_{r_{j+1}} - \{E\}) \cup \cdots \cup (F_{r_n} - \{E\}), \end{aligned}$$

where

$$\begin{aligned} i &= \begin{cases} n & \text{if } E \in I_{r_k} \text{ for every } 1 \leq k \leq n, \\ \min\{k \mid E \notin I_{r_k}, 1 \leq k \leq n\} & \text{otherwise,} \end{cases} \\ j &= \begin{cases} 1 & \text{if } E \in F_{r_k} \text{ for every } 1 \leq k \leq n, \\ \max\{k \mid E \notin F_{r_k}, 1 \leq k \leq n\} & \text{otherwise.} \end{cases} \end{aligned}$$

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 subscripted 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 ($1 \leq k \leq n$), 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 ($1 \leq k \leq n$), then

$$Succ(a_i, r') = \begin{cases} Succ(a_i, r'_k) & \text{if } k = n \text{ or } a_i \notin F_{r_k}, \\ Succ(a_i, r'_k) \cup (I_{r_{k+1}} - \{E\}) \cup \cdots \cup (I_{r_j} - \{E\}) & \text{if } k < n \text{ and } a_i \in F_{r_k}, \end{cases}$$

where

$$j = \begin{cases} n & \text{if } E \in I_{r_i} \text{ for every } k+1 \leq i \leq n, \\ \min\{i \mid E \notin I_{r_i}, k+1 \leq i \leq n\} & \text{otherwise.} \end{cases}$$

4. If $r' = *(r'_1)$, then

$$\begin{aligned} Succ(a_i, r') &= \begin{cases} Succ(a_i, r'_1) & \text{if } a_i \notin F_{r_1}, \\ Succ(a_i, r'_1) \cup (I_{r_1} - \{E\}) & \text{otherwise.} \end{cases} \end{aligned}$$

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.

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

We can show by an easy induction that $L(r) = L(G_r)$ for any regular expression r .

We test the unambiguity by using a graph called *testing graph*² of a Glushkov automaton. Let $G_r = (Q, \Sigma, \delta, q_I, F)$ 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

$$\begin{aligned} N &= \{(a_i, a_j) \mid (a_i, a_j) \text{ is a compatible pair of } Q\}, \\ E &= \{(a_k, a_l) \xrightarrow{a} (a_i, a_j) \mid a_i \in \delta(a_k, a), a_j \in \delta(a_l, a)\}. \end{aligned}$$

The following lemma holds by definition.

Lemma 5 *Let r be a regular expression and $G_r = (Q, \Sigma, \delta, q_I, F)$ be the Glushkov automaton of r . There are subscripted words $w_1, w_2 \in L(r')$ such that $w_1^{\natural} = w_2^{\natural}$ iff there is a path $(q_I, q_I) \xrightarrow{a_{i_1}^{\natural}} (a_{i_1}, a_{j_1}) \xrightarrow{a_{i_2}^{\natural}} \dots \xrightarrow{a_{i_l}^{\natural}} (a_{i_l}, a_{j_l})$ in $T(G_r)$ such that $w_1[k] = a_{i_k}$ and $w_2[k] = a_{j_k}$ for any $1 \leq k \leq l$, $a_{i_l}, a_{j_l} \in F$, and that $|w_1| = |w_2| = l$. \square*

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 4 *Let r be a regular expression, $u \in \text{occ}(r)$ be an occurrence, $q = \text{sub}(r, u)$ 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 $a_i, a_j \in \text{sym}(q')$ or $a_i, a_j \in \text{sym}(r') \setminus \text{sym}(q')$.
2. For any edge $(a_i, a_j) \xrightarrow{a} (a_k, a_l)$ in $T(G_r)$ from which some accepting node is reachable, if $a_i, a_j, a_k, a_l \in \text{sym}(q')$, then either

- (a) $a_k \notin \text{Succ}(a_i, q')$ and $a_l \notin \text{Succ}(a_j, q')$, or
- (b) $a_k \in \text{Succ}(a_i, q')$ and $a_l \in \text{Succ}(a_j, q')$.

Proof (sketch): *Only if part:* Assume that at least one of Conditions (1) and (2) does not hold. Then by Lemma 5 it is easy to show that there are words $w_1, w_2 \in L(r')$ such that $w_1^{\natural} = w_2^{\natural}$ and that for some i, j ($1 \leq i \leq j \leq |w_1|$) $w_1[i, j]$ maximally matches q' but $w_2[i, j] \notin L(q')$. 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 $w_1, w_2 \in L(r')$ such that $w_1^{\natural} = w_2^{\natural}$ and that for some i, j ($1 \leq i \leq j \leq |w_1|$) $w_1[i, j]$

maximally matches q' but $w_2[i, j]$ does not maximally match q' . We have two cases to be considered: (i) $w_2[i, j] \in L(q')$ but it is not maximal and (ii) $w_2[i, j] \notin L(q')$. Consider the case of (i) (the case of (ii) can be shown similarly). Assume that $w_2[i, j] \in L(q')$ but it is not maximal. Then there are indexes i', j' such that $w_2[i', j'] \in L(q')$ and that $\{i, \dots, j\} \subset \{i', \dots, j'\}$. Suppose $i' < i$ (the case of $j < j'$ can be shown similarly). Since $w_2[i', j'] \in L(q')$ with $i' < i$, $w_2[i-1] \in \text{sym}(q')$ and $w_2[i] \in \text{Succ}(w_2[i-1], q')$. On the other hand, since $w_1[i, j]$ maximally matches q' , either (i) $w_1[i-1] \notin \text{sym}(q')$ or (ii) $w_1[i-1] \in \text{sym}(q')$ but $w_1[i] \notin \text{Succ}(w_1[i], q')$. This implies that Condition (1) or (2) does not hold by Lemma 5. \square

Lemma 6 *For a regular expression r and a subexpression q of r , whether r is unambiguous w.r.t. q can be tested in $O(|r|^4)$ time.*

Proof: The Glushkov automaton G_r of r can be constructed in $O(|r|^2)$ time (Brüggenmann-Klein 1993), and the testing graph $T(G_r)$ can be constructed in $O(|r|^4)$ time. Moreover, the condition in Theorem 4 can be checked in linear time w.r.t. $T(G_r)$. \square

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

Finally, we show how to decide if a regular expression is unambiguous w.r.t. a label insertion.

To check this we slightly modify the testing graph of a Glushkov automaton. Let r be a regular expression and $G_r = (Q, \Sigma, \delta, q_I, F)$ be the Glushkov automaton of r . 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

- $a_j \in \delta(a_i, a)$, $a_i \neq b_h$, and $a_j \neq b_h$, 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 *c-compatible* if (i) $a_i = a_j = q_I$, or (ii) there is a c-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

$$\begin{aligned} N &= \{(a_i, a_j) \mid (a_i, a_j) \text{ is a c-compatible pair of } Q\}, \\ E &= \{(a_k, a_l) \xrightarrow{a} (a_i, a_j) \mid a_i \in \delta'(a_k, a), a_j \in \delta'(a_l, a)\}. \end{aligned}$$

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. $b_h \in F$ and (i) $a_i \in F$ and $b_h \in \delta(a_j, b)$ or (ii) $b_h \in \delta(a_i, b)$ and $a_j \in F$.

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.

²We use a modified version of the testing graph, originally defined in (Even 1965).

Theorem 5 Let $D = (d_1, a_0)$ be a DTD, $op = ins_elm(a, b, vi)$ be an edit operation to D such that $l(d_1(a), v) = \cdot$, b_h be the subscripted label inserted by op , and $G_{d_2(a)}$ be the Glushkov automaton of $d_2(a)$, where $op(D) = (d_2, a_0)$. Then $d_1(a)$ is unambiguous w.r.t. the insertion of b_h iff the following three conditions hold.

1. For any odd edge e in $T_{b_h}(G_{d_2(a)})$, no accepting node is reachable from e .
2. $T_{b_h}(G_{d_2(a)})$ contains no oddly accepting node.
3. $b_h \notin \delta(b_h, b)$, where δ is the transition function of $G_{d_2(a)}$.

Proof (sketch): Since $l(d_1(a), u) = \cdot$, $w \in L(d_1(a))$ iff there is a subscripted supersequence w_1 of w w.r.t. b_h such that $w_1 \in L((d_2(a))')$. Thus the theorem can be shown as follows.

Only if part: It is easy to show that if one of Conditions (1) to (3) does not hold, then $d_1(a)$ is not unambiguous w.r.t. the insertion of b_h .

If part: Assume that $d_1(a)$ is not unambiguous w.r.t. the insertion of b_h . Then for some word $w \in L(d_1(a))$, there are subscripted supersequences w_1, w_2 of w w.r.t. b_h such that $w_1, w_2 \in L(d_2(a))'$ and that for some k ($1 \leq k \leq |w_1|$)

- $w_1[i] = w_2[i]$ for every $1 \leq i < k$, but $w_1[k] = b_h$ and either $w_2[k] \neq b_h$ or $|w_2| < k$.

Consider the case where $w_2[k] \neq b_h$ with $k > 1$ (the other cases can be shown similarly). If $w_1[k-1] = w_2[k-1] = b_h$, then $w_1[k-1] = w_1[k] = b_h$, thus Condition (3) does not hold. Suppose that $w_1[k-1] = w_2[k-1] \neq b_h$. Since $w_2[k] \neq b_h$, there is an index $k' > k$ such that $w_1[k']$ matches $w_2[k]$. If $k' = k+1$, then Condition (1) does not hold (since $(w_1[k-1], w_2[k-1]) \xrightarrow{a} (w_1[k'], w_2[k])$ is an odd edge). If $k' > k+1$, then $w_1[k] = w_1[k+1] = b_h$, thereby Condition (3) does not hold. \square

Lemma 7 For a DTD (d_1, a_0) and an edit operation $op = ins_elm(a, b, vi)$ such that $l(d_1(a), v) = \cdot$, whether $d_1(a)$ is unambiguous w.r.t. the insertion of b_h can be determined in $O(|d_1(a)|^4)$ time, where b_h is the subscripted label inserted by op . \square

By Lemmas 4, 6, and 7, the complexity of checking the condition in Theorem 3 is summarized as follows.

Theorem 6 For a DTD D and an edit operation $s = op_1 \cdots op_n$ for D , whether the transformation algorithm inferred from D and s satisfies the condition in Theorem 3 can be determined in $O(n \cdot |R|^4 + k \cdot |D|^2)$ time, where R is the regular expression with maximum size in D and k is the number of operations of type 1a in s . \square

7 Conclusion

In this paper, we proposed a transformation algorithm inferred from a DTD and an edit script, then we presented a polynomial-time algorithm for determining if, for a DTD D and an edit script s , the transformation algorithm inferred from D and s is unambiguous.

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. (Choi 2002) states that only four out of 60 DTDs contains non-one-unambiguous regular expressions. This might suggest that real DTDs tend to admit unambiguous transformation.

Some schema languages, such as XML Schema and RELAX NG, admits more than one types against an element name. We would also like to take this feature into account.

Acknowledgement

This work is partially supported by the Grant-in-Aid for Young Scientists (B) #18700019 and Research Projects of Graduate School of Library, Information and Media Studies.

References

- Arenas, M. & Libkin, L. (2005), Xml data exchange: Consistency and query answering, in 'Proc. ACM PODS', pp. 13–24.
- Bohannon, P., Fan, W., Flaster, M. & Narayan, P. S. (2005), Information preserving xml schema embedding, in 'Proc. VLDB', pp. 85–96.
- Book, R., Even, S., Greibach, S. & Ott, G. (1971), 'Ambiguity in graphs and expressions', *IEEE Transactions on Computers* **C-20**(2), 149–153.
- Brüggenmann-Klein, A. (1993), 'Regular expressions into finite automata', *Theoretical Computer Science* **120**(2), 197–213.
- Brüggenmann-Klein, A. & Wood, D. (1998), 'One-unambiguous regular languages', *Information and Computation* **142**(2), 182–206.
- Choi, B. (2002), What are real dtlds like?, in 'Proc. WebDB', pp. 43–48.
- Even, S. (1965), 'On information lossless automata of finite order', *IEEE Transactions on Electronic Computers* **EC-14**, 561–569.
- Guerrini, G., Mesiti, M. & Rossi, D. (2005), Impact of xml schema evolution on valid documents, in 'Proc. WIDM (in conjunction with CIKM)', pp. 39–44.
- Hashimoto, K., Ishihara, Y. & Fujiwara, T. (2007), '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)* **J90-D**(4), 990–1004.
- Kuikka, E., Leinonen, P. & Penttonen, M. (2002), Towards automating of document structure transformations, in 'Proc. ACM DocEng', pp. 103–110.
- Leonardi, E., Hoai, T. T., Bhowmick, S. S. & Madria, S. (2006), Dtd-diff: A change detection algorithm for dtlds, in 'Proc. DASFAA', pp. 817–827.
- Miller, R., Hernandez, M. A., Hass, L., Yan, L., Ho, C., Fagin, R. & Popa, L. (2001), 'The clio project: Managing heterogeneity', *SIGMOD Record* **30**(1), 78–83.
- Milo, T. & Zohar, S. (1998), Using schema matching to simplify heterogeneous data translation, in 'Proc. VLDB', pp. 122–133.
- Morishima, A., Okawara, T., Tanaka, J. & Ishikawa, K. (2005), Smart: A tool for semantic-driven creation of complex xml mappings, in 'Proc. ACM SIGMOD', pp. 909–911.
- Rahm, E. & Bernstein, P. A. (2001), 'A survey of approaches to automatic schema matching', *VLDB Journal* **10**(4), 334–350.
- Suzuki, N. (2007), An edit operation-based approach to the inclusion problem for dtlds, in 'Proc. ACM SAC', pp. 482–488.