IEICE TRANS. INF. & SYST., VOL.E93–D, NO.8 AUGUST 2010

PAPER

An Algorithm for Inferring *K* Optimum Transformations of XML Document from Update Script to DTD

Nobutaka SUZUKI[†a)], ***Member***

SUMMARY DTDs are continuously updated according to changes in the real world. Let t be an XML document valid against a DTD D, and suppose that D is updated by an update script s. In general, we cannot uniquely "infer" a transformation of t from s, i.e., we cannot uniquely determine the elements in t that should be deleted and/or the positions in t that new elements should be inserted into. In this paper, we consider inferring K optimum transformations of t from s so that a user finds the most desirable transformation more easily. We first show that the problem of inferring K optimum transformations of an XML document from an update script is NP-hard even if $K = 1$. Then, assuming that an update script is of length one, we show an algorithm for solving the problem, which runs in time polynomial of $|D|$, $|t|$, and K.

key words: *XML, DTD, schema update, document transformation*

1. Introduction

DTDs are continuously updated according to changes in the real world. Suppose that we maintain XML documents valid against a DTD, and that the DTD is updated by some update script. Then the documents may no longer be valid against the DTD, and thus we have to transform each document into a valid one. However, it is indeed a hard task to find an appropriate transformation of each document manually. In this paper, we consider an algorithm that is helpful for finding appropriate transformations of XML documents when a DTD is updated.

Let t be an XML document valid against a DTD D, and suppose that D is updated by applying an update script s. In general, there is more than one (possibly infinite) way to transform t. In other words, we cannot uniquely "infer" from s (i) the elements in t that should be deleted and/or (ii) the positions in t into which new elements should be inserted. Thus, we need to select an appropriate transformation from such transformations. In such a situation, it is useful to compute the list of top-K (or K optimum) transformations of t inferred from s so that we can easily select the most appropriate transformation from the list. In this paper, we consider inferring such K optimum transformations from an update script.

For example, let us consider DTD D_1 (Fig. 1 (a)). Suppose that D_1 is updated to D_2 by an update script, which "aggregates" subexpression "(section$^+$,bib?)" of the content model of "book" into a single label "chapter" (Fig. 1 (b)). For tree t_1 in Fig. 1 (c), we have two alternatives t_2, t_3 according to the positions at which "chapter" elements should be inserted (Fig. 1 (d,e)). Our algorithm can infer such a list of transformations from a given update script, where the listed trees are ordered by the "amount of changes" (the number of insertions/deletions applied to the input tree).

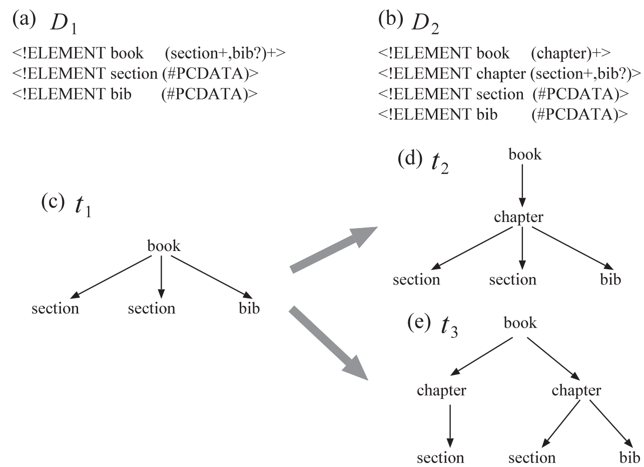

Fig. 1 DTDs D_1, D_2 and XML documents t_1, t_2, t_3.

As shown above, when a DTD is updated by an update script, more than one transformation of an XML document may be inferred from the update script, and we have to select an appropriate transformation from them. Clearly, listing such transformations in random order is very confusing to users. Although there is no universally agreed criterion for ordering such transformations, such a list can be readable and helpful to users if its transformations are ordered by the amount of changes, i.e., a transformation with less changes is ranked higher. Therefore, in this paper the transformation with the least amount of changes is treated as the optimum one.

Let s be an update script to a DTD D, t be an XML document valid against D, and K be a positive integer. The main results of this paper are the following twofold:

- In general, the problem of inferring K optimum transformations of t from s is intractable due to combinatorial explosion. In fact, we show that the problem is NP-hard even if $K = 1$.
- If s is restricted to be of length one, i.e., s consists only of one update operation, the problem can be solved relatively efficiently. In fact, we construct an algorithm

Manuscript received October 1, 2009.
Manuscript revised February 15, 2010.

[†]The author is 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.E93.D.2198

for solving this problem, which runs in time polynomial of $|D|$, $|t|$, and K.

In this paper, we first define update operations to a DTD. We next show a nondeterministic algorithm that transforms a tree according to a given update operation. Then, based on this algorithm, we show that the problem of inferring K optimum transformations of a tree from an update script is NP-hard even if $K = 1$. Finally, assuming that an update script s to a DTD D is of length one, we show an algorithm for inferring K optimum transformations of a tree t from s, which runs in time polynomial of $|D|$, $|t|$, and K.

Related Work

Schema matching and other related problems have been extensively studied, e.g., [1]–[8]. These studies considered finding an appropriate matching or transformation between schemas, assuming that no update script between schemas is known.

Several studies proposed update operations to schemas and discussed related problems. Leonardi et al. proposed update operations in order to represent the "diff" between two DTDs [9]. Hashimoto et al. proposed update operations to tree grammars so that no structural information of XML documents is lost when the documents are transformed according to a schema update [10]. Guerrini et al. proposed update operations for inclusion problem of schemas; any schema updated by their update operations includes its original schema [11]. Prashant et al. proposed three update operations and constructed an algorithm for generating XSLT scripts from a given update operation [12]. Suzuki et al. proposed an algorithm for deciding if, for a DTD D and an update script s, a transformation of t inferred from s is unique for any tree t valid against D [13]. To the best of the author's knowledge, no study considers inferring K optimum transformations of an XML document from an update script. Finally, this paper is a revised version of Ref. [14]. This paper provides (i) a revised estimation of the running time of the algorithm for inferring K optimum transformations of a tree from an update operation and (ii) a correctness proof of the algorithm, as well as excluding two insignificant update operations from those of Ref. [14]. The reason why the two update operations are excluded is that no transformation is required when a schema is updated by these operations, i.e., excluding these operations does not affect our transformation algorithm.

2. Definitions

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

Let Σ be a set of labels. 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 $r_1 \cdots r_n$ and $r_1|\cdots|r_n$ are regular expressions ($n \geq 1$).
- If r is a regular expression, then r^*, $r?$, and r^+ are regular expressions.

The language specified by a regular expression r is denoted $L(r)$.

In order to define update operations to a DTD, we sometimes represent a regular expression as a term in prefix notation. For example, we may write $\cdot(a, {}^*(|(b, c)))$ instead of $a(b|c)^*$, where '$\cdot$' denotes a concatenation operator. Let r be a regular expression in prefix notation. 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 $pos(r) = \{\lambda\} \cup \{u \mid u = iv, 1 \leq i \leq n, v \in pos(r_i)\}$. Note that $n = 1$ if $op \in \{{}^*, {}^+, ?\}$.

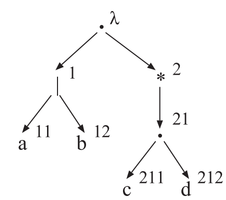

Fig. 2 Tree representation of r.

For example, let $r = (a|b)(cd)^*$. The prefix notation of r is $\cdot(|(a, b), {}^*(\cdot(c, d)))$. Figure 2 shows the tree representation of r, in which each node is associated with its corresponding position. Thus $pos(r) = \{\lambda, 1, 11, 12, 2, 21, 211, 212\}$.

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 $sub(r, \lambda) = r$.
- 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 $1 \leq j \leq n$ and some $v \in pos(r_j)$, then $l(r, u) = l(r_j, v)$ and $sub(r, u) = sub(r_j, v)$.

For example, in Fig. 2 $l(r, 1) =$ '|', $l(r, 11) = a$, and $sub(r, 21) = \cdot(c, d)$.

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 $w[i, j] = w[i]w[i+1] \cdots w[j]$ ($1 \leq i \leq j \leq |w|$). For example, if $w = kasuga$, then $w[3, 5] = sug$.

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 $r = (a|b|c)(d|b)^*$, then $r^\# = (a^{11}|b^{12}|c^{13})(d^{211}|b^{212})^*$ 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 $L(r^\#)^\natural = \{(w')^\natural \mid w' \in L(r^\#)\}$.

A *DTD* is a tuple $D = (d, sl)$, where d is a (possibly partial) mapping from Σ to the set of regular expressions over Σ, and $sl \in \Sigma$ is the *start label*. For example, the DTD in Fig. 1 (b) is denoted $(d, book)$, where d is a mapping defined as follows.

$$\begin{aligned} d(book) &= chapter^+ \\ d(chapter) &= section^+ bib \\ d(section) &= \epsilon \\ d(bib) &= \epsilon \end{aligned}$$

For a label $a \in \Sigma$, $d(a)$ is the *content model* of a. A tree t is *valid* against D if (i) the root of t is labeled by sl and (ii) for each node n in t the sequence of labels on the children of n is in $L(d(l(n)))$.

3. Update Operations to DTD

In this section, we define seven update operations to a DTD. Let us first consider desirable properties that our update operations should satisfy. First of all, the following property should clearly be satisfied.

P1) Any content model (regular expression) in a DTD can be updated to an arbitrary content model by using our update operations.

Update operations to insert/delete elements and those to insert/delete operators in a content model suffice to assure (P1). However, since a DTD also specifies ancestor-descendant relationships among elements, we often need update operations to insert/delete elements with such relationships preserved. Thus the following property should also be satisfied.

P2) Elements can be inserted/deleted, preserving ancestor-descendant relationships between elements specified in a DTD.

More concretely, let us consider how tree t_1 (Fig. 3 (d)) is transformed according to the DTD update shown in Fig. 3 (A). In this update, *contact* in $d(student)$ is "extracted", i.e., *contact* is deleted from $d(student)$ and *tel* and *email* are moved to $d(student)$ by a single update operation (Fig. 3 (b)), preserving ancestor-descendant relationships between *student* and *tel*/*email*. Thus, according to this update, the *contact* node in t_1 should be deleted and the *tel* and *email* nodes should be made as children of the *student* node (Fig. 3 (e)). Here, the above DTD update could seemingly be mimicked by using three distinct update operations; (i) a deletion of *contact* from $d(student)$ (Fig. 3 (B)) and (ii) insertions of *tel* and *email* into $d(student)$ (Fig. 3 (C)). However, this update is inappropriate since the update ignores the ancestor-descendant relationships between *student* and *tel*/*email* and thus the text values of *tel* and *email* elements in t_1 are not preserved (Fig. 3 (g)). Therefore, our update operations consist of the following two kinds of operations so that (P1) and (P2) are satisfied.

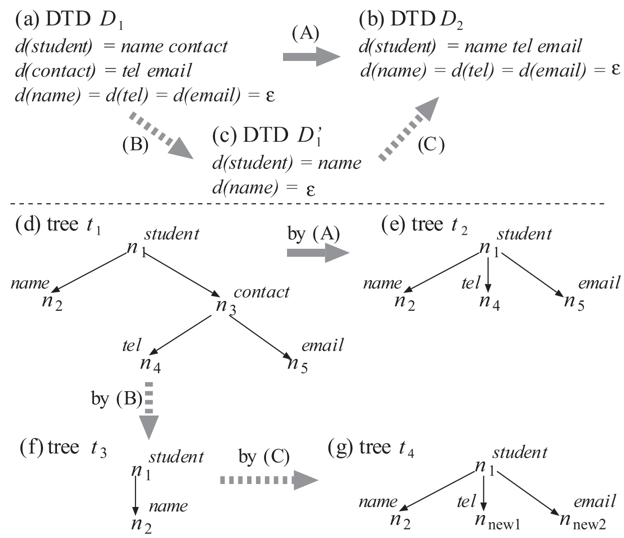

Fig. 3 Updating DTD D_1 to D_2 by extracting *contact* in $d(student)$.

- Update operations to insert/delete elements and to insert/delete operators ($\cdot$, $|$, *, $^+$, ?) in a content model. These are operations for assuring (P1).
- Update operations to change operators (*, $^+$, ?) and to insert/delete elements with ancestor-descendant relationships preserved. These are operations for assuring (P2).

Let us now show our update operations. Let $D = (d, sl)$ be a DTD. First, the following two operations relate to insertion/deletion of an element in a content model.

- $ins\_elm(a, b, vi)$: Inserts a new label b at position vi in $d(a)$, where $vi \in pos(d(a))$, i is a positive integer, and $b \in \Sigma$ (Fig. 4 (b,c)). This is applicable to D only if $d(b)$ is defined, $l(d(a), v) \in \{\cdot, |\}$, and the operator at v has at least $i - 1$ children.
- $del\_elm(a, vi)$: Deletes the label (possibly ϵ) at vi in $d(a)$. More formally, we have two cases according to the operator at v.
  - The case where $l(d(a), v) =$ '$\cdot$': The label at vi in $d(a)$ is deleted from $d(a)$ (Fig. 4 (a,b)). This is applicable to D only if the operator at v has more than one child.
  - The case where $l(d(a), v) =$ '$|$': If $l(d(a), vi) = l(d(a), vk)$ for some $k \neq i$, then the label at vi in $d(a)$ is deleted from $d(a)$ (deleting one of duplicated labels). Otherwise, the label at vi in $d(a)$ is replaced by ϵ.

Second, the following two operations relate to extraction/aggregation of an element.

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

The following three operations relate to handling an operator $(|, \cdot, ^*, ^+, ?)$ in a content model.

- *ins_opr*(a, opr, u, v): Inserts a new operator opr as the parent of the siblings at $u, \cdots, v$ in $d(a)$, where $opr \in \{\cdot, |, ^*, ^+, ?\}$ (Fig. 4 (c,d)). This is applicable to D only if (i) $u = v$ (opr has only one child) or (ii) $opr \in \{\cdot, |\}$ and $opr = l(d(a), w)$, where $u = wi$ and $v = wj$ for some $i < j$ (nesting the operator at w by opr).
- *del_opr*(a, u): Deletes an operator at u in $d(a)$ (Fig. 4 (f,g)). This is applicable to D only if (i) the operator at u has only one child or (ii) $l(d(a), u) = l(d(a), v)$, where $u = vi$ for some i (unnesting the operators at u and v).
- *change_opr*(a, opr, u): Replaces the operator at u in $d(a)$ by opr, where $l(d(a), u), opr \in \{?, ^*, ^+\}$.

Let op be an update operation to a DTD D. By $op(D)$ we mean the DTD obtained by applying op to D. Let $s = op_1 op_2 \cdots op_n$ be a sequence of update operations. We say that s is an *update script* to D if op_i is applicable to $op_{i-1}(op_{i-2}(\cdots op_1(D)\cdots))$ for every $1 \leq i \leq n$. By $|s|$ we mean the *length* of s, that is, $|s| = n$. 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 $D = (d, sl)$ be a DTD and op be an update operation applicable to D. Then $op(D)$ includes D if

- $op = ins\_elm(a, b, vi)$ and $l(d(a), v) =$ '|',
- $op = ins\_opr(a, opr, u, v)$,
- $op = del\_opr(a, u)$ and $l(d(a), u) \in \{\cdot, |\}$, or
- $op = change\_opr(a, opr, u)$, and (i) $opr =$ '*' or (ii) $l(d(a), u) = opr$.

□

Let D be a DTD and op be an update operation applicable to D. If $op(D)$ includes D, then any tree valid against D is also valid against $op(D)$. Thus, if the condition of the lemma holds, then without validating any trees we can find out that no transformation needs to be performed. Accordingly, the transformation algorithm defined in the next section uses the lemma in order to avoid unnecessary validations.

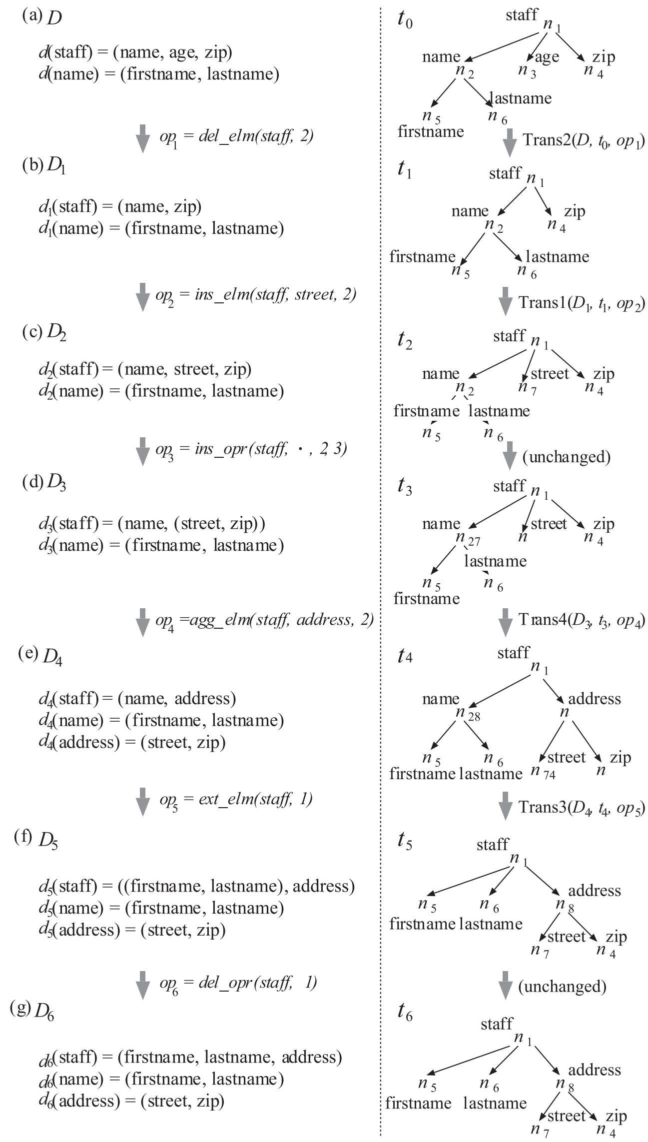

Fig. 4 An update script to D (left) and a transformation inferred from the update script (right).

4. Transformation Algorithm

Let t be a tree valid against a DTD D. If D is updated by an update operation op, we need to transform t according to op. In this section, we define an algorithm that nondeterministically transforms t according to op.

The following TransOp is the main part of the algorithm (Trans1 to Trans6 are shown later).

TransOp(D, t, op)

Input: a DTD D, a tree t valid against D, and an update operation op to D.
Output: a tree valid against $op(D)$.

1. If t is valid against $op(D)$, return t.
2. Else
   if $op = ins\_elm(a, b, vi)$, return Trans1(D, t, op),

if $op = del\_elm(a, vi)$, return Trans2(D, t, op),
if $op = ext\_elm(a, u)$, return Trans3(D, t, op),
if $op = agg\_elm(a, b, u)$, return Trans4(D, t, op),
if $op = ins\_opr(a, opr, u, v)$, return t,
if $op = del\_opr(a, u)$, return Trans5(D, t, op),
if $op = change\_opr(a,opr,u)$, return Trans6(D,t,op).

Note that if $op = ins\_opr(a, opr, u, v)$, then we do not have to transform t, since t is valid against $op(D)$ by Lemma 1.

Let us show six Trans subroutines. We need some definitions. Let r be a regular expression, $u \in pos(r)$ be a position in r, and $q = sub(r, u)$ be a subexpression of r. Moreover, let w' be a superscripted word such that $w' \in L(r^{\#})$. 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

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

For example, let $r = (a(b|c)^{+})^{*}$ and $q = sub(r, 12)$. Then $r^{\#} = (a^{11}(b^{1211}|c^{1212})^{+})^{*}$ and $q^{\#} = (b^{1211}|c^{1212})^{+}$. If $w' = a^{11}b^{1211}a^{11}c^{1212}b^{1211}$, then $match(w', q^{\#}) = \{(2, 2), (4, 5)\}$.

Let us first show Trans1. Trans1(D, t, op) transforms t according to op. In this case, $op = ins\_elm(a, b, vi)$, and by Lemma 1 $l(d_1(a), v) =$ '·'. Thus, it suffices to insert new b elements at appropriate positions in t.[†] We need a definition. 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 $(w'')^{\natural} = w$. In the following, we denote $D = (d_1, sl)$ and $op(D) = (d_2, sl)$, and assume that each transformation is done in bottom-up manner.

Trans1(D, t, op)

1. 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. If $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a superscripted supersequence w' of $l(n_1) \cdots l(n_m)$ w.r.t. b^h such that $w' \in L(d_2(a)^{\#})$, where b^h is the superscripted label in $d_2(a)^{\#}$ inserted by op.
      ii. For each $(j, j) \in match(w', b^h)$, create a new tree t_j valid against DTD (d_2, b) and insert t_j into t as the jth child of n.
2. Return t.

For example, the transformation from t_1 to t_2 in Fig. 4 is done by Trans1.

Note that in step (1-a-i) above, there may be more than one superscripted supersequence of $l(n_1) \cdots l(n_m)$ w.r.t. b^h matching $d_2(a)^{\#}$, and w' is selected nondeterministically. Similar behaviors can be found in the other Trans subroutines.

Let us next show Trans2. In this case, $op = del\_elm(a, vi)$. Thus, it suffices to delete the elements in t that match the label in $d_1(a)$ deleted by op.

Trans2(D, t, op)

1. 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. If $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a superscripted word w' such that $w' \in L(d_1(a)^{\#})$ and that $(w')^{\natural} = l(n_1) \cdots l(n_m)$.
      ii. By definition $(sub(d_1(a), vi))^{\#}$ is a single superscripted label, say b^{vi}. For each $(j, j) \in match(w', b^{vi})$, delete the subtree rooted at n_j from t.
2. Return t.

The transformation from t_0 to t_1 in Fig. 4 is an example of Trans2.

Let us show Trans3. In this case, $op = ext\_elm(a, u)$. Thus, it suffices to delete the nodes in t that match the label extracted by op.

Trans3(D, t, op)

1. 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. If $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a superscripted word w' such that $w' \in L(d_1(a)^{\#})$ and that $(w')^{\natural} = l(n_1) \cdots l(n_m)$.
      ii. By definition $(sub(d_1(a), u))^{\#}$ is a single superscripted label, say b^u. For each $(j, j) \in match(w', b^u)$, extract the jth child n_j of n from t, i.e., remove n_j from t and connect the children of n_j to the parent of n_j.
2. Return t.

The transformation from t_4 to t_5 in Fig. 4 is an example of Trans3.

Let us show Trans4. In this case, $op = agg\_elm(a, b, u)$. Thus, it suffices to insert a new parent node labeled by b into t for each sequence of nodes that matches $sub(d_1(a), u)$.

Trans4(D, t, op)

1. 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. If $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a superscripted word w' such that $w' \in L(d_1(a)^{\#})$ and that $(w')^{\natural} = l(n_1) \cdots l(n_m)$.
      ii. For each $(j,k) \in match(w', (sub(d_1(a),u))^{\#})$, insert a new node labeled by b as the parent of $n_j, \cdots, n_k$ into t.
2. Return t.

The transformation from t_3 to t_4 in Fig. 4 is an example of Trans4.

Let us show Trans5. We have $op = del\_opr(a, u)$ and

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

$l(d_1(a), u) \in \{?, ^*, ^+\}$. Thus we have three cases to be considered: (i) $l(d_1(a), u)$ = '?', (ii) $l(d_1(a), u)$ = '*', and (iii) $l(d_1(a), u)$ = '+'. Let $sub(d_1(a), u1) = q$. Consider first the case of (i). In this case, $sub(d_1(a), u) = q?$ and this is changed to q by op. Thus for each sequence of nodes matching $q?$, if the sequence is ϵ, we have to insert a sequence of elements matching q. This can be done similarly to the case of (iv) of Trans6 shown later. Let us next consider the case of (ii). Since q^* is changed to q by op, for each sequence seq matching q^*, (a) if $seq = \epsilon$, we have to insert a sequence of elements matching q and (b) otherwise, seq must be "shrunk" so that seq matches q instead of q^*. These can be handled by a combination of similar ideas shown later; (a) can be handled similarly to the case of (iv) of Trans6 and (b) can be done similarly to the case of (iii) (since $q^* = q^+|\epsilon$). In the following, we consider the case of (iii). In this case, $sub(d_1(a), u) = q^+$. Since q^+ is changed to q by op, we have to "shrink" each sequence of nodes in t that matches q^+ so that the resulting sequence matches q instead of q^+. The *q-extraction* $d_1^e(a)$ of $d_1(a)$ is obtained from $d_1(a)$ by replacing q^+ with q^*qq^*. Clearly, $d_1^e(a)$ is equivalent to $d_1(a)$. Let w' be a superscripted word such that $(w')^\natural \in L(d_1^e(a))$. A shrink w'' of w' w.r.t. $(q^+)^\#$ is obtained by deleting every sequence matching $sub(d_1^e(a), u1)$ or $sub(d_1^e(a), u3)$.

Trans5(D, t, op)

1. 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. If $l(n_1)\cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a superscripted word w' such that $w' \in L(d_1(a)^\#)$ and that $(w')^\natural = l(n_1)\cdots l(n_m)$.
      ii. Find a shrink w'' of w' w.r.t. $(q^+)^\#$, where $q^+ = sub(d_1(a), u)$. For each $1 \le j \le |w'|$ such that $w'[j]$ disappears in w'', delete the subtree rooted at n_j from t.
2. Return t.

Finally, let us show Trans6. We have op = $change\_opr(a, opr, u)$, and by Lemma 1 we have four cases to be considered: (i) $l(d_1(a), u)$ = '*' and opr = '?', (ii) $l(d_1(a), u)$ = '+' and opr = '?', (iii) $l(d_1(a), u)$ = '?' and opr = '+', and (iv) $l(d_1(a), u)$ = '*' and opr = '+'. Let $sub(d_1(a), u1) = q$. In the cases of (i) and (ii), for each sequence seq of nodes matching q^* or q^+, seq must be "shrunk" so that seq matches q instead of q^* or q^+. This can be treated similarly to the case of (iii) of Trans5. The case of (iii) can be handled similarly to the case of (iv). In the following, we consider the case of (iv). Then $sub(d_1(a), u) = q^*$ and q^* is changed to q^+ by op. Thus, for each position in t matching q^*, if the matched sequence is ϵ, then we have to insert a sequence of elements matching q. Let $w' \in L(d_1(a)^\#)$ be a superscripted word. For an index $0 \le i \le |w'|$, i is a *potential gap* w.r.t. $q^\#$ if neither $w'[i]$ nor $w'[i+1]$ is in $sym(q^\#)$ (assuming that $w'[0], w'[|w'|+1] \notin sym(q^\#)$). An *extension* of w' w.r.t. $q^\#$ is a superscripted word obtained by inserting zero or more w'_q's between $w'[i]$ and $w'[i+1]$ for every potential gap i w.r.t. $q^\#$, where w'_q is a word such that $w'_q \in L(q^\#)$.

Trans6(D, t, op)

1. 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. If $l(n_1)\cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a superscripted word w' such that $w' \in L(d_1(a)^\#)$ and that $(w')^\natural = l(n_1)\cdots l(n_m)$.
      ii. Find an extension w'' of w' w.r.t. $q^\#$ such that $w'' \in L(d_2(a)^\#)$. For each superscripted label $w''[i]$ inserted into w', create a tree t_i valid against $(d_2(a), (w''[i])^\natural)$ and insert t_i as the ith child of n.
2. Return t.

We write $t_2 \in$ TransOp(D, t_1, op) if t_2 can be the result of TransOp(D, t_1, op). It is clear that TransOp is correct.

Theorem 1: Let D be a DTD and op be an update operation to D. For any tree t_1 valid against D, every $t_2 \in$ TransOp(D, t_1, op) is valid against $op(D)$. □

5. NP-Hardness

In this section, we first define the problem of inferring K optimum transformations of an XML document from an update script. Then we show the NP-hardness of the problem.

5.1 Formal Definition of the Problem

Let D be a DTD, t_1 be a tree valid against D, and op be an update operation to D. For a tree $t_2 \in$ TransOp(D, t_1, op), the *difference* (or *diff*) between t_1 and t_2, denoted $df(t_1, t_2)$, is defined as follows. We have five cases according to op.

- $df(t_1, t_2)$ is defined as the set of root nodes of the subtrees inserted into t_1 if
  - $op = ins\_elm(a, b, vi)$, or
  - $op = change\_opr(a, opr, u)$, $l(d_1(a), u)$ = '*', and opr = '+'.
- $df(t_1, t_2)$ is defined as the set of root nodes of the subtrees deleted from t_1 if
  - $op = del\_elm(a, vi)$,
  - $op = del\_opr(a, u)$ and $l(d_1(a), u)$ = '+', or
  - $op = change\_opr(a, opr, u)$, $l(d_1(a), u)$ = '*', and opr = '?'.
- $df(t_1, t_2)$ is defined as the set of nodes deleted from t_1 if $op = ext\_elm(a, u)$.
- $df(t_1, t_2)$ is defined as the set of nodes inserted into t_1 if $op = agg\_elm(a, b, u)$.
- Otherwise, $df(t_1, t_2) = \emptyset$.

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. A sequence $TS = t_0, t_1, \cdots, t_n$ of trees is called *transformation sequence* w.r.t. (t, D, s) if $t_0 = t$ and $t_i \in \text{TransOp}(D_{i-1}, t_{i-1}, op_i)$ for every $1 \leq i \leq n$, where $D_{i-1} = op_{i-1}(\cdots op_1(D) \cdots)$. The cost of a transformation sequence TS, denoted $\gamma(TS)$, is defined as $\gamma(TS) = \sum_{1 \leq i \leq n} |df(t_{i-1}, t_i)|^{\dagger}$. For a positive integer K, we say that K transformation sequences $TS_1, \cdots, TS_K$ w.r.t. (t, D, s) are K *optimum* transformation sequences w.r.t. (t, D, s) if $\gamma(TS_i) \leq \gamma(TS_{i+1})$ for any $1 \leq i \leq K-1$ and $\gamma(TS_K) \leq \gamma(TS)$ for any transformation sequence TS w.r.t. (t, D, s) such that $TS \notin \{TS_1, \cdots, TS_K\}$. Now our problem is formulated as follows.

Instance: A DTD D, a tree t valid against D, an update script s to D, and a positive integer K.
Question: Find K optimum transformation sequences w.r.t. (t, D, s).

5.2 NP-Hardness of the Problem

In this subsection, we show that finding K optimum transformation sequences w.r.t. (t, D, s) is NP-hard even if $K = 1$. We consider the following decision problem, called *transformation decision problem*.

Instance: A DTD D, a tree t valid against D, an update script $s = op_1 op_2 \cdots op_n$ to D, and a positive integer B.
Question: Is there a transformation sequence $TS = t_0, t_1, \cdots, t_n$ w.r.t. (t, D, s) such that $\gamma(TS) \leq B$?

We have the following theorem.

Theorem 2: The transformation decision problem is NP-hard.

Proof: We use the following SAT problem.

Instance: A set $X = \{x_1, \cdots, x_n\}$ of variables and a collection $C = \{C_1, \cdots, C_m\}$ of clauses over X.
Question: Is there a satisfying truth assignment for C?

For an instance of the SAT problem, we construct an instance of the transformation decision problem, as follows.

- Tree $t = t_0$ is constructed as shown in Fig. 5 (top), where T_i and F_i stand for sequences of labels defined as follows $(1 \leq i \leq n)$.
  - Let $C_{i_1}, \cdots, C_{i_k}$ be the clauses in C that contain positive literal x_i. Then $T_i = c_{i_1} \cdots c_{i_k}$, where c_{i_j} is a label corresponding to clause C_{i_j}. That is, T_i consists of the clauses that are satisfied by setting x_i = true.
  - Let $C_{i_1}, \cdots, C_{i_l}$ be the clauses in C that contain negative literal $\neg x_i$. Then $F_i = c_{i_1} \cdots c_{i_l}$. That is, F_i consists of the clauses that are satisfied by setting x_i = false.
- $D = (d, r)$, where $d(r) = a^+$, $d(a) = b^+$, $d(b) = T_1|F_1|\cdots|T_n|F_n$, and $d(c_i) = \epsilon$ $(1 \leq i \leq m)$.
- $s = s_1 s_2 s_3$, where

$$\begin{aligned} s_1 &= del\_opr(a, \lambda) ext\_elm(a, \lambda) ext\_elm(r, 1), \\ s_2 &= ins\_opr(r, |, \lambda) ins\_subexpr(r, q, 2) \\ &\quad\; del\_subexpr(r, 1) del\_opr(r, \lambda), \\ s_3 &= ins\_elm(r, c_1, 2) del\_elm(r, 2) \\ &\quad\; \vdots \\ &\quad\; ins\_elm(r, c_m, 2) del\_elm(r, 2), \end{aligned}$$

  and

$$q = (c_1|\cdots|c_m)^*(c_1|\cdots|c_m)^*.$$

  In s_2, (i) $ins\_subexpr(r, q, 2)$ stands for a "macro" that inserts q into $d(r)$ at position 2 and (ii) $del\_subexpr(r, 1)$ is a macro that deletes the subexpression of $d(r)$ at position 1. Thus s_2 updates regular expression $(T_1|F_1|\cdots|T_n|F_n)^+$ into $(c_1|\cdots|c_m)^*(c_1|\cdots|c_m)^*$.
- $B = 3n$.

As shown below, s_1 corresponds to a truth assignment for $x_1, \cdots, x_n$, s_2 is the preliminary of s_3, and s_3 checks if the truth assignment chosen by s_1 satisfies C.

We show that there is a satisfying truth assignment for C *iff* there is a transformation sequence $TS = t_0, t_1, \cdots, t_{|s|}$ w.r.t. (t, D, s) such that $\gamma(TS) \leq B$.

If part: Assume that there is a transformation sequence $TS = t_0, t_1, \cdots, t_{|s|}$ w.r.t. (t, D, s) such that

$$\gamma(TS) \leq B. \tag{1}$$

Consider first s_1 of s. By $del\_opr(a, \lambda)$ one of t_{T_i} and t_{F_i} is deleted from t_0 for every $1 \leq i \leq n$, then by $ext\_elm(a, \lambda)$ n nodes labeled by b are deleted from t_1, and by $ext\_elm(r, 1)$ n nodes labeled by a are deleted from t_2 (Fig. 5). It is easy to see that t_3 is not changed by s_2, i.e., $t_3 = t_4 = \cdots = t_{|s_1 s_2|}$. Thus for transformation sequence $TS' = t_0, t_1, \cdots, t_{|s_1 s_2|}$ w.r.t. $(t, D, s_1 s_2)$, $\gamma(TS') = 3n = B$. This and (1) imply that by s_3 no node is inserted into $t_{|s_1 s_2|}$ and no node is deleted from $t_{|s_1 s_2|}$. For each $1 \leq i \leq m$, s_3 repeatedly updates $d(r)$ as follows.

1. First, $d(r) = (c_1|\cdots|c_m)^*(c_1|\cdots|c_m)^*$ is updated to $(c_1|\cdots|c_m)^* c_i (c_1|\cdots|c_m)^*$ by $ins\_elm(r, c_i, 2)$,
2. Then $(c_1|\cdots|c_m)^* c_i (c_1|\cdots|c_m)^*$ is updated to $(c_1|\cdots|c_m)^*(c_1|\cdots|c_m)^*$ by $del\_elm(r, 2)$.

Since $t_{|s_1 s_2|}$ is not changed by s_3, $t_{|s_1 s_2|}$ must have a leaf node labeled by c_i for every $1 \leq i \leq m$. Now consider the following truth assignment α $(1 \leq i \leq n)$.

$$\alpha(x_i) = \begin{cases} \text{true} & \text{if } t_{F_i} \text{ is deleted by } del\_opr(a, \lambda) \text{ of } s_1, \\ \text{false} & \text{if } t_{T_i} \text{ is deleted by } del\_opr(a, \lambda) \text{ of } s_1. \end{cases}$$

Since $t_{|s_1 s_2|}$ has a leaf node labeled by c_i for every $1 \leq i \leq m$, by the definitions of T_i and F_i it is easy to see that α is a satisfying truth assignment for C.

Only if part: Assume that there is a satisfying truth

† $\gamma(TS)$ is greater or equal to the tree edit distance between t_0 and t_n, assuming that a subtree insertion/deletion can be done by one edit operation.

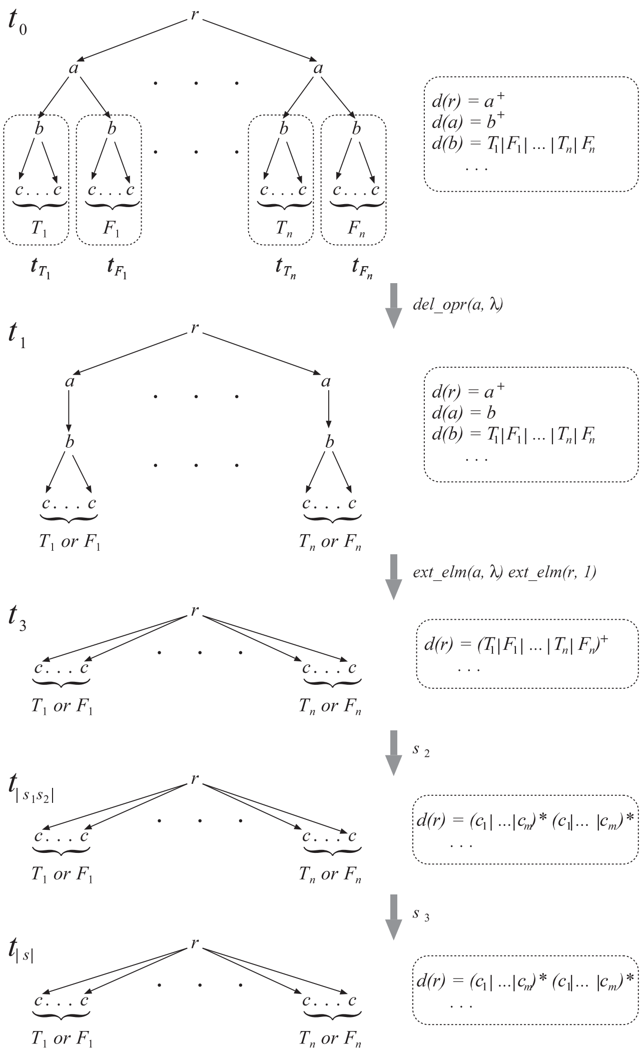

Fig. 5 Transformation sequence $t_0, t_1, \cdots, t_{|s|}$.

assignment α for C. Recall that by *del_opr*(a, λ) of s_1, one of t_{T_i} and t_{F_i} is deleted from t_0 for every $1 \leq i \leq n$. Along with the truth assignment α, t_0 can be transformed into t_1 so that for every $1 \leq i \leq n$,

- if $\alpha(x_i)$ = true, then t_{F_i} is deleted, and
- if $\alpha(x_i)$ = false, then t_{T_i} is deleted.

Since α is a satisfying truth assignment for C, it is easy to verify that for every $1 \leq i \leq m$, $t_{|s_1 s_2|}$ has at least one leaf node labeled by c_i. This implies that $t_{|s_1 s_2|}$ is not changed by s_3, i.e., $t_{|s_1 s_2|} = t_{|s_1 s_2|+1} = \cdots = t_{|s|}$. Here, let $TS = TS_1 TS_2$, where $TS_1 = t_0, t_1, \cdots, t_{|s_1 s_2|}$ and $TS_2 = t_{|s_1 s_2|+1}, \cdots, t_{|s|}$. Then we have $\gamma(TS_1) = 3n$ and $\gamma(TS_2) = 0$. Hence $\gamma(TS) = 3n \leq B$. □

Since an *ins_subexpr* operation consists of *ins_elm* and *ins_opr* operations and a *del_subexpr* operation consists of *del_elm* and *del_opr* operations, the above proof depends on neither *agg_elm* nor *change_opr* operation. Thus *ins_elm*, *del_elm*, *ext_elm*, *ins_opr*, and *del_opr* operations suffice to prove the NP-hardness.

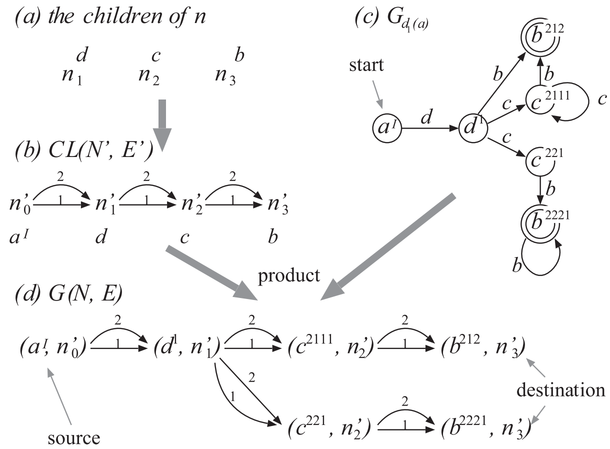

Fig. 6 The product $G(N, E)$ of $G_{d_1(a)}$ and $CL(N', E')$.

By Theorem 2, in general it is unlikely that we can find K optimum transformation sequences efficiently, even if $K = 1$. In the following, we consider finding K optimum transformation sequences assuming that an update script is of length one.

6. Algorithm for Finding *K* Optimum Transformation Sequences

In this section, we first define the Glushkov automaton [15] of a regular expression, which is required to describe our algorithm. We next show an algorithm for finding K optimum transformation sequences w.r.t. (t, D, s), assuming that $|s| = 1$.

The main difference between Glushkov automaton and usual NFA is that for any regular expression r, there is a one to one correspondence between the superscripted labels in $r^\#$ and the states of the Glushkov automaton of r (except the initial state), but a usual NFA does not have this property. For example, let $r = d((c^* b)|(cb^*))$ be a regular expression. Then $r^\# = d^1(((c^{2111})^* b^{212})|(c^{221}(b^{2221})^*))$. The Glushkov automaton of r is shown in Fig. 6 (c) (a^I is the initial state). Except the initial state a^I, each superscripted label in $r^\#$ occurs exactly once in the Glushkov automaton, and vice versa. For a DTD D and a tree t valid against D, when D is updated, we have to identify the nodes in t that should be deleted and/or the positions in t that new nodes should be inserted into. The above property is useful to obtaining such nodes and positions. For example, let $D = (d, sl)$ be a DTD, n be a node with $l(n) = a$ in a tree, $ch(n)$ be the children of n, and $G_{d(a)}$ be the Glushkov automaton of $d(a)$. If *del_elm*(a, u) is applied to D, we have to find the nodes in $ch(n)$ that should be deleted according to *del_elm*(a, u). This can be done by finding the nodes to which b^u is assigned under a matching between $ch(n)$ and $d(a)^\#$, where b^u is the state in $G_{d(a)}$ corresponding to the label at u in $d(a)$.

6.1 Glushkov Automaton

In this subsection, we define the Glushkov automaton of a regular expression. Let r be a regular expression. We first define the *initial set* I_r and the *final set* F_r, as follows.

- If $r = \epsilon$, then $I_r = F_r = \{E\}$, where E is a label not occurring in r (I_r and F_r contain E if $\epsilon \in L(r)$).
- If $r = a$ for some $a \in \Sigma$, then $I_r = F_r = \{a^i\}$, where a^i is the superscripted label such that $r^\# = a^i$.
- If $r = r_1|\cdots|r_n$, then $I_r = I_{r_1} \cup \cdots \cup I_{r_n}$ and $F_r = F_{r_1} \cup \cdots \cup F_{r_n}$.
- If $r = r_1 \cdots r_n$, 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

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

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

- If $r = r_1^*$ or $r = r_1?$, then $I_r = I_{r_1} \cup \{E\}$ and $F_r = F_{r_1} \cup \{E\}$.
- If $r = r_1^+$, then $I_r = I_{r_1}$ and $F_r = F_{r_1}$.

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.

- If $r^\# = a^i$, then $Succ(a^i, r^\#) = \emptyset$.
- If $r^\# = r_1^\#|\cdots|r_n^\#$ and a^i occurs in $r_k^\#$ ($1 \leq k \leq n$), then $Succ(a^i, r^\#) = Succ(a^i, r_k^\#)$.
- If $r^\# = r_1^\# \cdots r_n^\#$ and a^i occurs in $r_k^\#$ ($1 \leq k \leq n$), then

$$Succ(a^i, r^\#) = \begin{cases} Succ(a^i, r_k^\#) \\ \qquad \text{if } k = n \text{ or } a^i \notin F_{r_k}, \\ Succ(a^i, r_k^\#) \cup (I_{r_{k+1}} - \{E\}) \cup \\ \cdots \cup (I_{r_j} - \{E\}) \\ \qquad \text{if } k < n \text{ and } a^i \in F_{r_k}, \end{cases}$$

where

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

- If $r^\# = (r_1^\#)^*$ or $r^\# = (r_1^\#)^+$, then

$$Succ(a^i, r^\#) = \begin{cases} Succ(a^i, r_1^\#) & \text{if } a^i \notin F_{r_1}, \\ Succ(a^i, r_1^\#) \cup (I_{r_1} - \{E\}) & \text{otherwise.} \end{cases}$$

- If $r^\# = (r_1^\#)?$, then $Succ(a^i, r^\#) = Succ(a^i, r_1^\#)$.

The *Glushkov automaton* of r is a 5-tuple $G_r = (Q, \Sigma, \delta, a^I, F)$, where Q is the set of *states*, δ is the *transition function*, $a^I \notin sym(r^\#)$ is a new symbol denoting the *initial (or start) state* of G_r, and F is the set of *final states* defined as follows.

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

It is easy to show that for any regular expression r, $L(r) = L(G_r)$, where G_r is the Glushkov automaton of r. Figure 6 (c) shows the Glushkov automaton of regular expression $d((c^*b)|(cb^*))$.

6.2 Algorithm

In this subsection, we show an algorithm for finding K optimum transformation sequences $TS_1, \cdots, TS_K$ w.r.t. (t, D, op).

Main Algorithm

The algorithm consists of the "main" algorithm and some subroutines. Let us first show the "main" algorithm. Let $D = (d, sl)$ be a DTD, t be a tree valid against D, n be a node in t, and op be an update operation to D. By t_n we mean the subtree of t rooted at n, and let $D(n) = (d, l(n))$ be a DTD. We say that $df_1(n), \cdots, df_K(n)$ are K *optimum diffs* w.r.t. $(t_n, D(n), op)$ if for some K optimum transformation sequences $TS_1, \cdots, TS_K$ w.r.t. $(t_n, D(n), op)$, $df_i(n) = \gamma(TS_i)$ for every $1 \leq i \leq K$.

The following algorithm Main computes K optimum diffs $df_1(n), \cdots, df_K(n)$ w.r.t. $(t_n, D(n), op)$ for each node n in bottom-up manner. For each node n in t, the algorithm does the following.

- If n is a leaf and no child needs to be added to n by op, then $df_1(n), \cdots, df_K(n)$ are obtained in steps 2 and 3. In step 2, we have $(d_2, sl) = op(D)$.
- Otherwise, $df_1(n), \cdots, df_K(n)$ are computed in steps 4 to 21. The subroutines in these steps are shown later.
  - In steps 5 to 19, a graph $G(N, E)$ and a weight function w are obtained, where $G(N, E)$ represents the "product" of $d_1(a)$ and the children of n, and w assigns a diff to each edge on $G(N, E)$.
  - In step 20, K optimum diffs $df_1(n), \cdots, df_K(n)$ are computed by finding K "shortest" paths on $G(N, E)$.

Main(D, t, op, K)

Input: A DTD $D = (d_1, sl)$, a tree t valid against D, an update operation op to D, and a positive integer K.
Output: K optimum diffs w.r.t. (t, D, op).

```
    begin
1.  for each node n in t (in bottom-up order) do
2.    if n is a leaf and (l(n) ≠ a or ϵ ∈ L(d2(a))) then
3.      df1(n) ← ∅ and dfi(n) ← nil for each 2≤i≤ K;
4.    else begin
5.      if l(n) = a and l(n1)···l(nm) ∉ L(d2(a)) then
6.        if op = ins_elm(a, b, vi) then
7.          (G(N, E), w) ← MkGraph1(D, t, n, op, K);
8.        if op = del_elm(a, vi) then
9.          (G(N, E), w) ← MkGraph2(D, t, n, op, K);
10.       if op = ext_elm(a, u) then
11.         (G(N, E), w) ← MkGraph3(D, t, n, op, K);
12.       if op = agg_elm(a, b, u) then
13.         (G(N, E), w) ← MkGraph4(D, t, n, op, K);
```

```
            if op = del_opr(a, u) then
                (G(N, E), w) ← MkGraph5(D, t, n, op, K);
            if op = change_opr(a, opr, u) then
                (G(N, E), w) ← MkGraph6(D, t, n, op, K);
        else // none of the children of n is changed
            (G(N, E), w) ← MkGraph7(D, t, n, op, K);
        (df_1(n), ··· , df_K(n)) ← FindKDiffs(G(N, E), w);
    end
    Let n be the root of t. return df_1(n), ··· , df_K(n);
end
```

Outline of Subroutines

Among the subroutines in Main, we here explain MkGraph2 and FindKDiffs (the others are shown later). We first show outlines of MkGraph2 and FindKDiffs, then show their formal definitions.

Let n be a node in t labeled by a, and let us consider finding K optimum diffs $df_1(n), \cdots, df_K(n)$. Assuming that $df_1(n_i), \cdots, df_K(n_i)$ have been obtained for each child n_i of n, we find $df_1(n), \cdots, df_K(n)$ as follows. Suppose that $op = del\_elm(a, vi)$.

1. We first make a "child list graph" $CL(N', E')$ of n. Figure 6 (b) is an example assuming that $K = 2$. As shown later, each edge $n'_{i-1} \xrightarrow{l} n'_i$ is associated with the lth diff $df_l(n_i)$ of n_i.
2. We make the Glushkov automaton $G_{d_1(a)}$ of $d_1(a)$. For example, Fig. 6 (c) shows the Glushkov automaton of $d_1(a) = d((c^*b)|(cb^*))$.
3. We make the "product graph" $G(N, E)$ of $G_{d_1(a)}$ and $CL(N', E')$ as shown in Fig. 6 (d), then associate a "weight" (actually, a diff) to each edge in E. $G(N, E)$ has the following properties.
   a. Any path in $G(N, E)$ from the source to a destination represents the sequence of children that matches $d_1(a)^\#$. For example, path
   $$(a^I, n'_0) \xrightarrow{l_1} (d^1, n'_1) \xrightarrow{l_2} (c^{221}, n'_2) \xrightarrow{l_3} (b^{2221}, n'_3)$$
   in Fig. 6 (d) represents the sequence of children n_1, n_2, n_3 that matches $d^1 c^{221} b^{2221} \in L(d_1(a)^\#)$, for any $l_1, l_2, l_3 \in \{1, 2\}$.
   b. Each edge $e = (a^{i-1}, n'_{j-1}) \xrightarrow{l} (a^i, n'_j) \in E$ is associated with the lth diff $df_l(n_j)$ of n_j, but we have one exception; if a^i is the superscripted label deleted from $d_1(a)$ by op, then e is associated with $\{n_j\}$ instead of $df_l(n_j)$, where $\{n_j\}$ represents the diff when the subtree rooted at n_j is deleted.
4. Find K "shortest" paths from the source to the destinations. By (a) and (b) above, the diffs on these paths are precisely K optimum diffs $df_1(n), \cdots, df_K(n)$.

Steps 1 to 3 above are done by MkGraph2 and step 4 is done by FindKDiffs.

Let us show the formal definitions related to steps 1 to 3. Let n be a node in t with children $n_1, \cdots, n_m$ and K be a positive integer. Then the *child list graph* of n (w.r.t. K) is a graph $CL(N', E')$, where

$$N' = \{n'_0, \cdots, n'_m\},$$
$$E' = \{n'_{i-1} \xrightarrow{l} n'_i \mid 1 \leq i \leq m, 1 \leq l \leq K\},$$

and $l(n'_0) = a^I$ and $l(n'_i) = l(n_i)$ for $1 \leq i \leq m$. Let $G_r = (Q, \Sigma, \delta, a^I, F)$ be the Glushkov automaton of r. Then the *product* of G_r and $CL(N', E')$ is defined as a graph $G(N, E)$, where

$$N = \{(a^i, n'_j) \mid a_i \in Q, n'_j \in N', (a^i)^\natural = l(n'_j)\},$$
$$E = \{(a^i, n'_{j-1}) \xrightarrow{l} (a^k, n'_j) \mid$$
$$a^k \in \delta(a^i, (a^k)^\natural), n'_{j-1} \xrightarrow{l} n'_j \in E'\}.$$

We say that (a^I, n'_0) is the *source* of $G(N, E)$ and (a^h, n'_m) is a *destination* of $G(N, E)$ if $a^h \in F$. Now MkGraph2 is defined as follows.

MkGraph2(D, t, n, op, K)

```
Input: A DTD D = (d_1, sl), a tree t valid against D,
   a node n in t, an update operation op = del_elm(a, vi),
   and a positive integer K.
Output: A graph G(N, E) and a function w.
  begin
    Construct the child list graph CL(N', E') of n.
    Construct the Glushkov automaton G_{d_1(a)} of d_1(a).
    Construct the product G(N,E) of G_{d_1(a)} and CL(N',E').
    for each e = (a^i, n'_{j-1}) --l--> (a^k, n'_j) ∈ E let
        w(e) ← { {n_j}       if a^k = b^{vi} and l = 1,
               { nil         if a^k = b^{vi} and l > 1,
               { df_l(n_j)   if a^k ≠ b^{vi},
        where b^{vi} is the superscripted label deleted from
           d_1(a)^# by op.
    return (G(N, E), w);
  end
```

We next define FindKDiffs. This algorithm can be defined similarly to usual algorithms for finding K shortest paths (e.g. [16]) with a slight modification. Thus we first show an algorithm for solving the K shortest paths problem before showing FindKDiffs. Let $H(N_H, E_H)$ be a weighted acyclic graph having one source n_0 and one or more destinations, where a source is a node that no edge enters and a destination is a node that no edge leaves. By $w_H(e)$ we mean the weight (nonnegative real number) of edge $e \in E_H$. We show an algorithm for computing the weights of K shortest paths from the source to the destinations in $H(N_H, E_H)$. In the algorithm shown below, Δ_{n_i} denotes the multiset of weights of K shortest paths from n_0 to n_i, and the algorithm computes Δ_{n_i} for each $n_i \in N_H$. In line 3, we write $n_j \prec n_k$ if $n_j \to n_k \in E_H$. Thus the nodes in N_H are visited in a bottom-up manner due to lines 3 and 4. By $\Delta_{n_i}[k]$ we mean the kth least weight in Δ_{n_i}.

KShortestPaths$(H(N_H, E_H))$

```
Input: A weighted acyclic graph H(N_H, E_H).
Output: A set of weights of K shortest paths.
  begin
1.  Let Δ_{n_i} be the multiset of K ∞'s for each n_i ∈ N_H;
2.  Δ_{n_0}[1] ← 0;
3.  Sort the nodes in N_H w.r.t. '≺' topologically.
```

Let $n_{i_1}, \cdots, n_{i_{|N_H|}}$ be the result.
4. **for** $h = 1$ **to** $|N_H|$ **do**
5. **for** each edge $e \in E_H$ leaving n_{i_h} with $w_H(e) \neq \infty$ **do**
6. Let $e = n_{i_h} \to n_j$.
7. **for** $k = 1$ **to** K **do**
8. $df \leftarrow \Delta_{n_{i_h}}[k] + w_H(e)$;
9. **if** $df < \Delta_{n_j}[K]$ **then**
10. Replace $\Delta_{n_j}[K]$ by df in Δ_{n_j}.
11. $\Delta \leftarrow \bigcup_{n_i \text{ is a destination}} \Delta_{n_i}$;
12. **return** $\{\Delta[1], \cdots, \Delta[K]\}$;
end

Let n be a node in t with children $n_1, \cdots, n_m$, $CL(N', E')$ be the child list graph of n, $G_{d_1(l(n))}$ be the Glushkov automaton of $d_1(l(n))$, and $G(N, E)$ be the product of $G_{d_1(l(n))}$ and $CL(N', E')$. Since FindKDiffs have to find K optimum diffs instead of K weight values, we have to modify KShortestPaths so that the diff on a path in $G(N, E)$ is handled appropriately. Let

$$p = \underbrace{(a^I, n'_0) \xrightarrow{l_1} \cdots \xrightarrow{l_g} (a^{i_g}, n'_g)}_{p_g} \xrightarrow{l_{g+1}} \cdots \xrightarrow{l_m} (a^{i_m}, n'_m)$$

be a path from the source to a destination in $G(N, E)$ and let p_g be the prefix of p as shown above. Let $w(p_g)$ be the weight (diff) on p_g, that is,

$$w(p_g) = w((a^I, n'_0) \xrightarrow{l_1} (a^{i_1}, n'_1)) \cup \cdots \cup w((a^{i_{g-1}}, n'_{g-1}) \xrightarrow{l_g} (a^{i_g}, n'_g)).$$

Then $w(p_g)$ represents a diff for t_n assuming that

1. diffs for $t_{n_{g+1}}, \cdots, t_{n_m}$ are ignored,
2. n'_j is associated with a^{i_j} for every $1 \leq j \leq g$, i.e., we have $w'[j] = a^{i_j}$ due to step (1-a-i) of Trans2, and that
3. under Condition (2) above, t_{n_j} is transformed by the l_jth optimum diff w.r.t. $(t_{n_j}, D(n_j), op)$ $(1 \leq j \leq g)$.

Let $\Delta_{(a^{i_g}, n'_g)}$ be the collection of K optimum diffs of $C_{(a^{i_g}, n'_g)}$, where

$$C_{(a^{i_g}, n'_g)} = \{w(p_g) \mid p_g \text{ is a path from } (a^I, n'_0) \text{ to } (a^{i_g}, n'_g) \text{ in } G(N, E).\}.$$

FindKDiffs shown below computes $\Delta_{(a^{i_g}, n'_g)}$ for every $(a^{i_g}, n'_g) \in N$. Similarly to KShortestPaths, we write $(a^i, n'_j) \prec (a^h, n'_k)$ if $(a^i, n'_j) \xrightarrow{l} (a^h, n'_k) \in E$. Thus, the nodes in N are visited in a bottom-up manner due to lines 3 and 4. Note that $G(N, E)$ is acyclic since $CL(N', E')$ is acyclic. In lines 8 to 10, $\Delta_{(a^i, n'_j)}[k]$ denotes the kth optimum diff in $\Delta_{(a^i, n'_j)}$, and we assume that if $\Delta_{(a^i, n'_j)}[k] = nil$, then $|\Delta_{(a^i, n'_j)}[k]| = \infty$. In line 9, a condition to check if $df \notin \Delta_{(a^i, n'_j)}$ is added since there may be more than one paths having the same diff, i.e., paths p, p' from (a^I, n'_0) to (a^i, n'_j) such that $w(p) = w(p')$. Without this condition $\Delta_{(a^i, n'_j)}$ might contain duplicated diffs.

FindKDiffs$(G(N, E), w)$

Input: A product $G(N, E)$ and a weight function w.
Output: K optimum diffs of n.
begin
1. $\Delta_{(a^i, n'_j)} \leftarrow \{nil, \cdots, nil\}$ (K nil's) for each $(a^i, n'_j) \in N$;
2. $\Delta_{(a^I, n'_0)}[1] \leftarrow \emptyset$;
3. Sort the nodes in N w.r.t. '$\prec$' topologically. Let $(a^{i_1}, n'_{j_1}), \cdots, (a^{i_{|N|}}, n'_{j_{|N|}})$ be the result.
4. **for** $h = 1$ **to** $|N|$ **do**
5. **for** each edge $e \in E$ leaving (a^{i_h}, n'_{j_h}) with $w(e) \neq nil$ **do**
6. Let $e = (a^{i_h}, n'_{j_h}) \xrightarrow{l} (a^i, n'_j)$.
7. **for** each $k = 1$ to K with $\Delta_{(a^{i_h}, n'_{j_h})}[k] \neq nil$ **do**
8. $df \leftarrow \Delta_{(a^{i_h}, n'_{j_h})}[k] \cup w(e)$;
9. **if** $|df| < |\Delta_{(a^i, n'_j)}[K]|$ and $df \notin \Delta_{(a^i, n'_j)}$ **then**
10. Delete $\Delta_{(a^i, n'_j)}[K]$ from $\Delta_{(a^i, n'_j)}$ and add df to $\Delta_{(a^i, n'_j)}$.
11. $\Delta \leftarrow \bigcup_{(a^i, n'_m) \text{ is a destination}} \Delta_{(a^i, n'_m)}$;
12. **return** K optimum distinct diffs in Δ;
end

Comparing FindKDiffs to KShortestPaths, FindKDiffs maintains a collection of K diffs instead of a set of K weight values for each node in a graph, but it is easy to see that FindKDiffs still runs in time polynomial of $|D|$, $|t|$, and K.

Other Subroutines

First, MkGraph3 is defined exactly same as MkGraph2. In the following, we show MkGraph1 and MkGraph7. The rest MkGraph's are shown in Appendix A.

First, MkGraph7 can be defined easily. Let $D = (d_1, sl)$ be a DTD, n be a node in t, and $G(N, E)$ be the product of the Glushkov automaton of $d_1(a)$ and the child list graph $CL(N', E')$ of n. According to step 18 of Main, none of the children of n is changed, thus it suffices to set $w(e) = df_l(n_j)$ for each edge $e = (a^i, n'_{j-1}) \xrightarrow{l} (a^k, n'_j)$ in $G(N, E)$. Therefore, MkGraph7 is defined similarly to MkGraph2 except step 5.

MkGraph7(D, t, n, op, K)

Input: A DTD $D = (d_1, sl)$, a tree t valid against D, a node n in t, an update operation op, and a positive integer K.
Output: A graph $G(N, E)$ and a function w.
begin
1. Construct the child list graph $CL(N', E')$ of n.
2. Construct the Glushkov automaton $G_{d_1(a)}$ of $d_1(a)$.
3. Construct the product $G(N,E)$ of $G_{d_1(a)}$ and $CL(N',E')$.
4. **for** each $e = (a^i, n'_{j-1}) \xrightarrow{l} (a^k, n'_j) \in E$ **let**
5. $w(e) \leftarrow df_l(n_j)$;
6. **return** $(G(N, E), w)$;
end

We next show MkGraph1. We have $op = ins\_elm(a, b, vi)$. Let $D = (d_1, sl)$ be a DTD and n be a node labeled by a with children $n_1, \cdots, n_m$. Since $op = ins\_elm(a, b, vi)$, nodes labeled by b may be inserted into $n_1, \cdots, n_m$. Accordingly, we have to modify some definitions. First, to handle node insertion after n_m, we append a dummy node n_{m+1} labeled by x as the last child of n, where

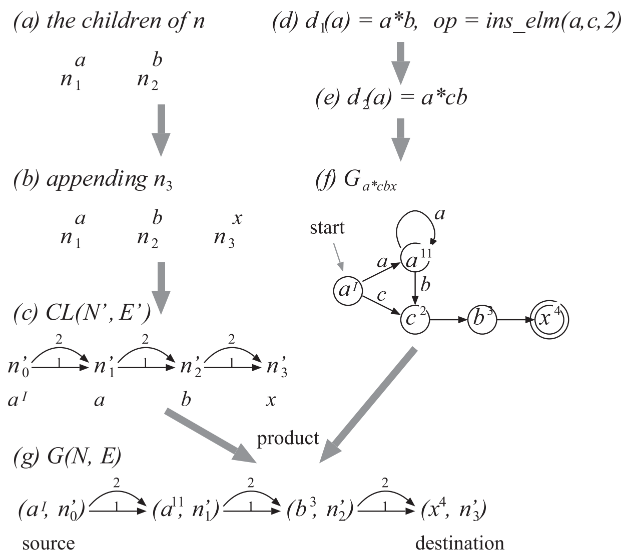

Fig. 7 The product $G(N, E)$ of G_{a*cbq} and $CL(N', E')$.

x is a new label not appearing in D (Fig. 7 (b)). We also modify the product of a Glushkov automaton and a child list graph. Let $(d_2, sl) = op(D)$. Since n_{m+1} is appended, we use the Glushkov automaton of $d_2(a)x$ instead of $d_2(a)$ (Fig. 7 (d–f)). Let $G_{d_2(a)x} = (Q, \Sigma, \delta, a^I, F)$ be the Glushkov automaton of $d_2(a)x$, $CL(N', E')$ be the child list graph of n modified as above, and b^h be the superscripted label inserted into $d_1(a)^\#$ by op. Taking the insertion of b^h into account, the *product* of $G_{d_2(a)x}$ and $CL(N', E')$ is defined as a graph $G(N, E)$, where

$$\begin{aligned} N &= \{(a^i, n'_j) \mid a^i \in Q, n'_j \in N', (a^i)^\natural = l(n'_j)\}, \\ E &= \{(a^i, n'_{j-1}) \xrightarrow{l} (a^k, n'_j) \mid n'_{j-1} \xrightarrow{l} n'_j \in E', \text{ and,} \\ &\qquad \text{(i) } b^h \in \delta(a^i, b) \text{ and } a^k \in \delta(b^h, (a^k)^\natural) \text{ or} \\ &\qquad \text{(ii) } a^k \in \delta(a^i, (a^k)^\natural)\}. \end{aligned}$$

This is defined similarly to the product graph in MkGraph2, except Condition (i) of E. This condition handles the case where a node matching b^h is inserted between n_{j-1} and n_j. Figure 7 (g) is an example with $K = 2$. We have two edges between (a^{11}, n'_1) and (b^3, n'_2) due to Condition (i), which implies that a new node labeled by c is inserted between n_1 and n_2.

Now let us show MkGraph1. The weight (diff) of each edge in E is computed in steps 5 to 16. Steps 7 and 8 compute a collection of diffs for the edges satisfying Condition (ii), and steps 9 to 13 compute a collection of diffs for the edges satisfying Condition (i). Lines 10 and 11 handle the case where one ore more b^h's can be inserted between n_{j-1} and n_j, while lines 12 and 13 handle the case only one b^h is inserted between n_{j-1} and n_j. Here, $S(b, k)$ denotes a set of k new nodes labeled by b, inserted between n_{j-1} and n_j. $\Delta[l]$ in step 16 denotes the lth optimum diff in Δ.

MkGraph1(D, t, n, op, K)

Input: A DTD D, a tree t valid against D, a node n in t, an update operation $op = ins\_elm(a, b, vi)$, and a positive integer K.
Output: A graph $G(N, E)$ and a function w.

begin
1. Append a dummy node n_{m+1} labeled by x as the last child of n. Let $df_l(n_{m+1}) \leftarrow \emptyset$ for $1 \le l \le K$.
2. Construct the child list graph $CL'(N', E')$ of n.
3. Construct the Glushkov automaton $G_{d_2(a)x} = (Q, \Sigma, \delta, a^I, F)$, where $(d_2, sl) = op(D)$.
4. Construct the product $G(N, E)$ of $G_{d_2(a)x}$ and $CL(N', E')$.
5. **for** each edge $(a^i, n'_{j-1}) \xrightarrow{1} (a^k, n'_j) \in E$ **do**
6. $\quad \Delta_1 \leftarrow \emptyset, \Delta_2 \leftarrow \emptyset$;
7. $\quad$ **if** $a^k \in \delta(a^i, (a_k)^\natural)$ **then**
8. $\quad\quad \Delta_1 \leftarrow \{df_l(n_j) \mid 1 \le l \le K\}$;
9. $\quad$ **if** $b^h \in \delta(a^i, b)$ and $a^k \in \delta(b^h, (a^k)^\natural)$ **then**
10. $\quad\quad$ **if** $b^h \in \delta(b^h, b)$ **then**
11. $\quad\quad\quad \Delta_2 \leftarrow \{df_l(n_j) \cup S(b, k) \mid 1 \le l \le K, 1 \le k \le K\}$;
12. $\quad\quad$ **else**
13. $\quad\quad\quad \Delta_2 \leftarrow \{df_l(n_j) \cup S(b, 1) \mid 1 \le l \le K\}$;
14. $\quad \Delta \leftarrow \Delta_1 \cup \Delta_2$;
15. $\quad$ **for** $l = 1$ to K **do**
16. $\quad\quad w((a^i, n'_{j-1}) \xrightarrow{l} (a^k, n'_j)) \leftarrow \Delta[l]$
17. **return** $(G(N, E), w)$;

end

We show the correctness of the algorithm.

Theorem 3: For a DTD D, a tree t valid against D, an update operation op to D, and a positive integer K, Main(D, t, op, K) returns K optimum diffs w.r.t. (t, D, op).

Proof (sketch): Let a be the label specified as the first argument of op. We first define the *level* of a node n in t, denoted $lv(n)$, as follows.

- If n is a leaf, and, $l(n) \neq a$ or $\epsilon \in L(d_2(a))$, then $lv(n) = 0$.
- If n is a leaf, $l(n) = a$, and $\epsilon \notin L(d_2(a))$, then $lv(n) = 1$.
- If n is an internal node with children $n_1, \cdots, n_m$, then $lv(n) = 1 + \max_{1 \le i \le m} lv(n_i)$.

Let $D = (d_1, sl)$ and $D(n) = (d_1, l(n))$. We show that for every node n in t $df_1(n), \cdots, df_K(n)$ are K optimum diffs w.r.t. $(t_n, D(n), op)$, by induction on $lv(n)$.

Basis: Let n be a leaf in t such that $lv(n) = 0$. Since $l(n) \neq a$ or $\epsilon \in L(d_2(a))$, we do not have to add any child to n. Thus, by steps 2 and 3 of Main $df_1(n) = \emptyset$ and $df_i(n) = nil$ for $2 \le i \le K$, which are K optimum diffs w.r.t. $(t_n, D(n), op)$.

Induction: Let n be a node in t with children $n_1, \cdots, n_m$. As an induction hypothesis, assume that $df_1(n_j), \cdots, df_K(n_j)$ are K optimum diffs w.r.t. $(t_{n_j}, D(n_j), op)$ for every child n_j of n. In the following, we consider the case where $l(n) = a$, $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, and $op = ins\_elm(a, b, vi)$ (the other cases can be shown similarly). We have $df_l(n_{m+1}) = \emptyset$ for every $1 \le l \le K$ by line 1 of MkGraph1. Let b^h be the superscripted label in $d_2(a)^\#$ inserted by op, and let $(b^h)^{(0)} = \epsilon$ and $(b^h)^{(k)} = (b^h)^{(k-1)} b^h$. Moreover, we define that $\Delta_i(k, l) = S(b, k) \cup df_l(n_i)$ $(1 \le i \le m + 1)$. Let $t'_n \in$ TransOp$(D(n), t_n, op)$ be a tree such that $\delta(t_n, t'_n)$ is ith optimum with $i \le K$. Then we have

$$\delta(t_n, t'_n) = \Delta_1(k_1, l_1) \cup \cdots \cup \Delta_{m+1}(k_{m+1}, l_{m+1})$$

for some $1 \le l_1, \cdots, l_{m+1} \le K$ and some $0 \le k_1, \cdots, k_{m+1} \le K$ such that

$$(b^h)^{(k_1)} a^{i_1} \cdots (b^h)^{(k_m)} a^{i_m} (b^h)^{(k_{m+1})} \in L(d_2(a)^\#),$$

where a^{i_j} is a superscripted label such that $(a^{i_j})^\natural = l(n_j)$ $(1 \le j \le m)$. Let $G(N, E)$ be the product and w be the weight function obtained by MkGraph1. Since $\delta(t_n, t'_n)$ is ith optimum with $i \le K$, by lines 5 to 16 of MkGraph1 it is easy to show that there is a path

$$(a^I, n'_0) \xrightarrow{l'_1} (a^{i_1}, n'_1) \xrightarrow{l'_2} \cdots \xrightarrow{l'_{m+1}} (a^{i_{m+1}}, n'_{m+1})$$

in $G(N, E)$ such that (a^I, n'_0) is the source, $(a^{i_{m+1}}, n'_{m+1})$ is a destination, and that $w((a^{i_{j-1}}, n'_{j-1}) \xrightarrow{l'_j} (a^{i_j}, n'_j)) = \Delta_j(k_j, l_j)$ for every $1 \le j \le m+1$. Hence $G(N, E)$ covers any paths having desirable diffs. Now it is easy to show that $df_1, \cdots, df_K$ are K optimum diffs w.r.t. $(t_n, D(n), op)$ *iff* there is a path p_i from the source to a destination in $G(N, E)$ such that $w(p_i) = df_i$ and that p_i is the ith "shortest" path for every $1 \le i \le K$. Thus, FindKDiffs$(G(N, E), w)$ in line 20 of Main correctly returns K optimum diffs w.r.t. $(t_n, D(n), op)$. □

It is easy to see that Main runs in time polynomial of $|t|$, $|D|$, and K. The proof of the following theorem is sketched in Appendix B.

Theorem 4: Let $D = (d_1, sl)$ be a DTD, t be a tree valid against D, K be a positive integer, and a be the label such that $|d_1(a)| \ge |d_1(b)|$ for any label b. Then Main(D, t, op, K) runs in $O(|t|^2 \cdot od(t)^2 \cdot |d_1(a)|^3 \cdot K^2)$ time, where $od(t)$ is the maximum outdegree in t. □

7. Conclusion

In this paper, we first showed that the problem of finding K optimum transformation sequences w.r.t. (t, D, s) is NP-hard even if $K = 1$. Then, assuming that $|s| = 1$, we proposed an algorithm for finding K optimum transformation sequences w.r.t. (t, D, s), which runs in time polynomial of $|D|$, $|t|$, and K.

We used a diff between trees as the criterion of optimality of transformation. We have to further investigate whether this criterion is appropriate. Moreover, this paper presented no experimental result. As a future work, we need to examine (i) by experiment if our algorithm can present appropriate transformations and (ii) the efficiency of our algorithm.

Acknowledgement

The author is deeply grateful for reviewers' insightful comments. This work is partially supported by the Grant-in-Aid for Young Scientists (B) #20700077.

References

[1] M. Arenas and L. Libkin, "XML data exchange: Consistency and query answering," Proc. ACM PODS, pp.13–24, 2005.
[2] S. Amano, L. Libkin, and F. Murlak, "XML schema mappings," Proc. ACM PODS, pp.33–42, 2009.
[3] P. Bohannon, W. Fan, M. Flaster, and P.P.S. Narayan, "Information preserving XML schema embedding," Proc. VLDB, pp.85–96, 2005.
[4] E. Kuikka, P. Leinonen, and M. Penttonen, "Towards automating of document structure transformations," Proc. ACM DocEng, pp.103–110, 2002.
[5] 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.
[6] T. Milo and S. Zohar, "Using schema matching to simplify heterogeneous data translation," Proc. VLDB, pp.122–133, 1998.
[7] 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.
[8] E. Rahm and P.A. Bernstein, "A survey of approaches to automatic schema matching," VLDB Journal, vol.10, no.4, pp.334–350, 2001.
[9] 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.
[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] 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.
[12] B.V.N. Prashant and P.S. Kumar, "Managing XML data with evolving schema," Proc. COMAD, 2006.
[13] N. Suzuki and Y. Fukushima, "An XML transformation algorithm inferred from an update script between DTDs," IEICE Trans. Inf. & Syst., vol.E92-D, no.4, pp.594–607, April 2009.
[14] N. Suzuki, "On inferring k optimum transformations of XML document from update script to DTD," Proc. COMAD, pp.210–221, April 2008.
[15] A. Brüggenmann-Klein and D. Wood, "One-unambiguous regular languages," Inf. Comput., vol.142, no.2, pp.182–206, 1998.
[16] E. Martins, "K-th shortest path problem," http://www.mat.uc.pt/~eqvm/OPP/KSPP/KSPP.html

Appendix A: MkGraph Subroutines

Let us first consider MkGraph4. We have $op = agg\_elm(a, b, u)$. Let $G(N, E)$ be the product of $G_{d_1(a)}$ and $CL(N', E')$, as defined in MkGraph2, and let $q = sub(d_1(a), u)$. By op, for each sequence of nodes in t that maximally match q, a node labeled by b is inserted into t as the parent of the nodes. To represent such a node insertion, for each path $(a^{i_0}, n'_{j_0}) \xrightarrow{l_1} \cdots \xrightarrow{l_n} (a^{i_n}, n'_{j_n})$ in $G(N, E)$ that "maximally matches" q, we add new edges from (a^{i_0}, n'_{j_0}) to (a^{i_n}, n'_{j_n}) to $G(N, E)$ that represent a newly inserted node. Formally, we say that a path $(a^{i_0}, n'_{j_0}) \xrightarrow{l_1} \cdots \xrightarrow{l_n} (a^{i_n}, n'_{j_n})$ in $G(N, E)$ *maximally matches* q if

- (a^{i_0}, n'_{j_0}) is the source of $G(N, E)$ or $a^{i_1} \notin Succ(a^{i_0}, q^\#)$,
- $a^{i_{k+1}} \in Succ(a^{i_k}, q^\#)$ for $1 \le k \le n-1$, and
- (a^{i_n}, n'_{j_n}) is a destination of $G(N, E)$ or there is an edge $(a^{i_n}, n'_{j_n}) \xrightarrow{l} (a^h, n'_k)$ such that $a^h \notin Succ(a^{i_n}, q^\#)$.

Suppose that there is a path from (a^{i_0}, n'_{j_0}) to (a^{i_n}, n'_{j_n}) maximally matching q. By $G((a^{i_0}, n'_{j_0}), (a^{i_n}, n'_{j_n}), q)$ we mean the subgraph of $G(N, E)$ consisting of the paths from (a^{i_0}, n'_{j_0}) to (a^{i_n}, n'_{j_n}) maximally matching q ((a^{i_0}, n'_{j_0}) is the source and (a^{i_n}, n'_{j_n}) is the destination of this subgraph). We create a new edge $e = (a^{i_0}, n'_{j_0}) \xrightarrow{l} (a^{i_n}, n'_{j_n})$ representing a newly inserted node, say v, and compute $w(e)$ by taking the union of (i) $\{v\}$ and (ii) the diff on the lth shortest path from (a^{i_0}, n'_{j_0}) to (a^{i_n}, n'_{j_n}) in $G((a^{i_0}, n'_{j_0}), (a^{i_n}, n'_{j_n}), q)$. Now MkGraph4 is defined as follows. Lines 4 and 5 compute the weight of edges that are not on any paths maximally matching q. Lines 6 to 15 treat the edges representing newly inserted nodes; for each pair $((a^i, n'_j), (a^h, n'_k))$ of nodes in $G(N, E)$ such that there is a path maximally matching q between the nodes, a graph $G((a^i, n'_j), (a^h, n'_k), q)$ is constructed, then for each $1 \le l \le K$, the weight of edge $(a^i, n'_j) \xrightarrow{l} (a^h, n'_k)$ is obtained as shown above and the edge is added to $G(N, E)$.

MkGraph4(D, t, n, op, K)

Input: A DTD $D = (d_1, sl)$, a tree t valid against D,
a node n in t, an update operation $op = agg\_elm(a,b,u)$,
and a positive integer K.
Output: A graph $G(N, E)$ and a function w.

begin
1. Construct the child list graph $CL(N', E')$ of n.
2. Construct the Glushkov automaton $G_{d_1(a)}$ of $d_1(a)$.
3. Construct the product $G(N,E)$ of $G_{d_1(a)}$ and $CL(N',E')$.
4. **for** each $e = (a^i, n'_{j-1}) \xrightarrow{l} (a^k, n'_j) \in E$ **let**
5. $\quad w(e) \leftarrow \begin{cases} nil & \text{if } a^k \in sym(q^\#), \\ df_l(n_j) & \text{otherwise.} \end{cases}$
6. $P \leftarrow \{((a^i,n'_j), (a^h,n'_k)) \mid$ there is a path from (a^i,n'_j) to (a^h, n'_k) in $G(N, E)$ that maximally matches $q\}$;
7. **for** each $((a^i, n'_j), (a^h, n'_k)) \in P$ **do**
8. $\quad$ Construct a graph $G' = G((a^i, n'_j), (a^h, n'_k), q)$.
9. $\quad$ **for** each edge $e = (a^i, n'_{j-1}) \xrightarrow{l} (a^k, n'_j)$ in G' **let**
10. $\quad\quad w'(e) \leftarrow df_l(n_j)$;
11. $\quad (df_1, \cdots, df_K) \leftarrow$ FindKDiffs(G', w').
12. $\quad$ **for** each $l = 1$ to K **do**
13. $\quad\quad$ Create a new node v labeled by b.
14. $\quad\quad w((a^i, n'_j) \xrightarrow{l} (a^h, n'_k)) \leftarrow \{v\} \cup df_l$;
15. $\quad\quad$ Add $(a^i, n'_j) \xrightarrow{l} (a^h, n'_k)$ to E.
16. **return** $(G(N, E), w)$;

end

Let us next show MkGraph5. We have $op = del\_opr(a, u)$ and $l(d_1(a), u) \in \{?, {}^*, {}^+\}$. We have three cases to be considered: (i) $l(d_1(a), u) =$ '?', (ii) $l(d_1(a), u) =$ '*', and (iii) $l(d_1(a), u) =$ '+'. Let $sub(d_1(a), u1) = q$. Consider first the case of (i). In this case, we have $sub(d_1(a), u) = q?$ and this is changed to q by op. Thus for each sequence of nodes matching $q?$, if the sequence is ϵ, we have to insert a sequence of elements matching q. This can be done similarly to the case of (iv) of MkGraph6 shown later. Let us next consider the case of (ii). Since q^* is changed to q by op, for each sequence seq matching q^*, (a) if $seq = \epsilon$, we have to insert a sequence of elements matching q and (b) otherwise, seq must be "shrunk" so that seq matches q instead of q^*. These can be handled by a combination of similar ideas shown later; (a) can be handled similarly to the case of (iv) of MkGraph6 and (b) can be done similarly to the case of (iii). In the following, let us consider the case of (iii). In this case, $sub(d_1(a), u) = q^+$ for some regular expression q, and q^+ is changed to q by op. Thus, for each sequence seq of nodes that maximally matches q^+, seq must be "shrunk". Recall that the *q-extraction* $d_1^e(a)$ of $d_1(a)$ is obtained from $d_1(a)$ by replacing q^+ with q^*qq^*. MkGraph5 is defined so that the nodes matching the first/second q^* in q^*qq^* are deleted. In step 5, $sub(d_1^e(a), u1)$ ($sub(d_1^e(a), u3)$) is the first (resp., second) q^* in q^*qq^*. $\{n_j\}$ in step 7 denotes the diff to delete the subtree rooted at n_j.

MkGraph5(D, t, n, op, K)

Input: A DTD $D = (d_1, sl)$, a tree t valid against D,
a node n in t, an update operation $op = del\_opr(a, u)$,
and a positive integer K.
Output: A graph $G(N, E)$ and a function w.

begin
1. Construct the child list graph $CL(N', E')$ of n.
2. Construct the $sub(d_1(a), u)$-extraction $d_1^e(a)$ of $d_1(a)$.
3. Construct the Glushkov automaton $G_{d_1^e(a)}$ of $d_1^e(a)$.
4. Construct the product $G(N,E)$ of $G_{d_1^e(a)}$ and $CL(N',E')$.
5. $LR \leftarrow sym(sub(d_1^e(a), u1)) \cup sym(sub(d_1^e(a), u3))$;
6. **for** each $e = (a^i, n'_{j-1}) \xrightarrow{l} (a^k, n'_j) \in E$ **let**
7. $\quad w(e) \leftarrow \begin{cases} \{n_j\} & \text{if } a^k \in LR \text{ and } l = 1, \\ nil & \text{if } a^k \in LR \text{ and } l > 1, \\ df_l(n_j) & \text{otherwise.} \end{cases}$
8. **return** $(G(N, E), w)$;

end

Finally, let us show MkGraph6. In this case, $op = change\_opr(a, opr, u)$. We have four cases to be considered: (i) $l(d_1(a), u) =$ '*' and $opr =$ '?', (ii) $l(d_1(a), u) =$ '+' and $opr =$ '?', (iii) $l(d_1(a), u) =$ '?' and $opr =$ '+', and (iv) $l(d_1(a), u) =$ '*' and $opr =$ '+'. The cases of (i) and (ii) can be treated similarly to the case of (iii) of MkGraph5. The case of (iii) can be handled similarly to the case of (iv) below. In the following, we consider the case of (iv). Then $sub(d_1(a), u) = q^*$ for some regular expression q. Since q^* is changed to q^+ by op, for each sequence matching q^*, if the sequence is ϵ, we have to insert a sequence of elements matching q. Let $G_{d_1(a)} = (Q, \Sigma, \delta_1, a^I, F)$ be the Glushkov automaton of $d_1(a)$ and $G_{d_2(a)} = (Q, \Sigma, \delta_2, a^I, F)$ be the Glushkov automaton of $d_2(a)$, where $(d_2, sl) = op(D)$. For states $a^i, a^k \notin sym(q^\#)$, we say that the transition from a^i to a^k is *missing* if $a^i \in \delta_1(a^k, (a^k)^\natural)$ but $a^i \notin \delta_2(a^k, (a^k)^\natural)$. If nodes n_i, n_{i+1} match a^i and a^k, respectively, and the transition from a^i to a^k is missing, then for a word $w \in L(q)$, it suffices to insert $|w|$ elements matching w between n_i and n_{i+1} of t. Thus MkGraph6 is defined as follows.

MkGraph6(t, n, D, op, K)

Input: A DTD $D = (d_1, sl)$, a tree t valid against D,
a node n in t, an update operation $op =$
$change\_opr(a, opr, u)$, and a positive integer K.
Output: A graph $G(N, E)$ and a function w.

begin
1. Construct the child list graph $CL(N', E')$ of n.
2. Construct the Glushkov automaton $G_{d_1(a)}$ of $d_1(a)$, and construct the Glushkov automaton $G_{d_2(a)}$ of $d_2(a)$.
3. Construct the product $G(N,E)$ of $G_{d_1(a)}$ and $CL(N',E')$.
4. Let w be a word in $L(sub(d_1(a), u1))$.
5. **for** each $e = (a^i, n'_{j-1}) \xrightarrow{l} (a^k, n'_j) \in E$ **do**

6. **if** the transition from a^i to a^k is missing **then**
7. Create new nodes $v_1, \cdots, v_{|w|}$ labeled by $w[1], \cdots, w[|w|]$, respectively.
8. $w(e) \leftarrow \{v_1, \cdots, v_{|w|}\} \cup df_l(n_j)$;
9. **else**
10. $w(e) \leftarrow df_l(n_j)$;
11. **return** $(G(N, E), w)$;
end

Appendix B: The Sketch of Proof of Theorem 4

Proof (sketch): Let us first consider the running time of FindKDiffs. In line 4, $|N| \in O(od(t) \cdot |d_1(a)|)$, where $od(t)$ denotes the maximum outdegree of the nodes in t. In line 5, there are at most $|d_1(a)|$ edges leaving (a^{i_h}, n'_{j_h}). For each k in line 7, lines 8 to 10 run in $O(K \cdot |t|)$. Thus, FindKDiffs$(G(N, E), w)$ runs in $O(od(t) \cdot |d_1(a)|^2 \cdot K^2 \cdot |t|)$.

Among subroutines MkGraph1 to MkGraph7, MkGraph4 is the most time consuming. Lines 7 to 11 of MkGraph4 are the most time consuming part of the subroutine. In line 7, the number of pairs in P is in $O((od(t) \cdot |d_1(a)|)^2)$ time. In line 8, G' can be obtained in $O(od(t) \cdot |d_1(a)| \cdot K)$ time. In line 11, FindKDiffs(G', w') runs in $O(od(t) \cdot |d_1(a)|^2 \cdot K^2 \cdot |t|)$. Thus, MkGraph4$(G(N, E), w)$ runs in $O(od(t)^3 \cdot |d_1(a)|^4 \cdot K^2 \cdot |t|)$ time.

Consequently, Main(D, t, op, K) runs in $O(od(t)^3 \cdot |d_1(a)|^4 \cdot K^2 \cdot |t|^2)$ time. □

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. Since 2009, he has been an Associate Professor of Graduate School of Library, Information and Media Studies, University of Tsukuba. His current research interests include database theory and structured documents.