An XML Transformation Algorithm Inferred from an Update Script between DTDs

Nobutaka SUZUKI†, Member and Yuji FUKUSHIMA†, Nonmember

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

key words: XML, data transformation, schema evolution

1. Introduction

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

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

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

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

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

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

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

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

Related Work

Schema matching, query discovery, and other related problems have been extensively studied, e.g., [1], [2], [11], [13]–[16], [20]. These studies except [20] consider finding an ap-
appropriate matching or transformation between schemas, assuming that no update script between the schemas is known. Ref. [20] considers the problem of adapting mappings to schema changes. The study assumes that a mapping between (nested relational) schemas are explicitly provided, thus the ambiguities discussed in this paper do not arise.

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

2. Definitions

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

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

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

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

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

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

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

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

- If \( r = \epsilon \) or \( r = a \) for some \( a \in \Sigma \), then \( l(r, \lambda) = r \) and
3. Update Operations to DTD

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

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

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

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

- $\text{ext}_{\text{elm}}(a, u)$: Extracts a label $l(d(a), u)$ in $d(a)$. Formally, this operation replaces label $l(d(a), u)$ in $d(a)$ by regular expression $d(l(d(a), u))$ (Fig. 4(e,f)). This is applicable to $d$ only if $\text{sub}(d(u), a)$ is a label in $d(a)$ and $a \neq l(d(a), u)$.
- $\text{agg}_{\text{elm}}(a, b, u)$: Aggregates subexpression $\text{sub}(d(a), u)$ into single label $b$. Formally, this operation (i) sets $d(b) = \text{sub}(d(a), u)$ and (ii) replaces $\text{sub}(d(u), a)$ by label $b$ (Fig. 4(d,e)). This is applicable to $d$ only if $d(b)$ is undefined.

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

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

Let $opr$ be an update operation to a DTD $d$. By $opr(d)$ we mean the DTD obtained by applying $opr$ to $d$. Let $s = opr_1 \circ opr_2 \cdots \circ opr_n$ be a sequence of update operations $(n \geq 0)$, $s$ is applicable to $d$ if $opr_i$ is applicable to $opr_{i-1}(opr_{i-2}(\cdots (opr_1(d) \cdots)\))$ for every $1 \leq i \leq n$. We say that $s$ is an update script to $d$ if $s$ is applicable to $d$. For an update script $s = opr_1 \circ opr_2 \cdots \circ opr_n$ to $d$, we define that $s(d) = opr_n(pr_{n-1}(\cdots opr_1(d)) \cdots)$. An update script of length zero is denoted $\epsilon$, where $\epsilon$ is an identity operator such that $\epsilon(d) = d$ for any DTD $d$.

**Example 1:** Let $d$ be a DTD, where $d(\text{staff}) =$
We say that a DTD $d_2$ includes a DTD $d_1$ if for any tree $t$, $t$ is valid against $d_2$ whenever $t$ is valid against $d_1$. We have the following lemma.

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

1. $op = ins_{elm}(a, b, vi)$ and either (i) $l(d(a), v) = '+'$ and $l(d(a), vi) = e$ or (ii) $l(d(a), v) = '-'$ and $l(d(a), vi) \in L(sub(d(a), vk))$ for some $k \neq i$ (i.e., the element at $vi$ is contained in some sibling).
2. $op = del_{elm}(a, vi)$ and either (i) $l(d(a), v) = '+'$ and $l(d(a), vi) = e$ or (ii) $l(d(a), v) = '-'$ and $l(d(a), vi) \in L(sub(d(a), vk))$ for some $k \neq i$ (i.e., the element at $vi$ is contained in some sibling).
3. $op = ext_{elm}(a, u)$ and $l(d(a), u) \in L(d(d(a), u)))$.
4. $op = ins_{opr}(a, opr, vi, vj)$, where $opr \in \{+,-,\}$. $\Box$

For example, in Fig. 4 $ins_{opr}(staff, -, 2, 3)$ satisfies Condition (4) and $del_{opr}(staff, 1)$ satisfies Condition (5) of the above lemma.

**4. Transformation Algorithm Inferred from DTD and Update Script**

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

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

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

Type 1a: (1) If \( op = \text{ins}_\text{elm}(a, b, u) \), then \( b \) is inserted at \( u \) in \( d(a) \). Accordingly, for each position \( p \) in \( t \) at which the \( b \)-label should be inserted, \( T \) creates a new valid tree whose root is labeled by \( b \) and insert the tree at position \( p \) in \( t \). For example, if \( d(a) = (\langle a, b \rangle) \) and \( op = \text{ins}_\text{elm}(a, c, 3) \), 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 \( op = \text{del}_\text{elm}(a, u) \), then the label, say \( b \), at \( u \) in \( d(a) \) is deleted or replaced by \( e \). Accordingly, \( T \) first identifies the subtrees in \( t \) whose roots match the label \( b \), then \( T \) deletes the identified subtrees.

Type 1b: (1) If \( op = \text{ext}_\text{elm}(a, u) \), then \( T \) identifies the internal nodes in \( t \) that match the extracted label in \( d(a) \), and deletes the identified internal nodes from \( t \). For example, if \( d(a) = (\langle b, c \rangle) \) and \( op = \text{ext}_\text{elm}(a, 1) \), then each internal node in \( t \) that matches \( b \) is deleted.

(2) If \( op = \text{agg}_\text{elm}(a, b, u) \), then \( T \) inserts new \( b \)-labeled nodes into \( t \) as the parents of sibling nodes that should be aggregated. For example, if \( d(a) = (\langle b, c, e \rangle) \) and \( op = \text{agg}_\text{elm}(a, f, 1) \), then for each pair of siblings labeled by \( b \) and \( c \) in \( t \) a new node labeled by \( f \) is inserted as the parent of the siblings.

Type 2: By Conditions (4) and (5) of Lemma 1, \( op = \text{del}_\text{op}(a, u) \) and \( \text{sub}(d(a), u) = \star(q) \) for some regular expression \( q \) (\( op \) deletes the ‘\( * \)’ from \( \star(q) \)). Thus \( T \) identifies the nodes in \( t \) that match \( \star(q) \) and deletes “excess” subtrees from the subtrees rooted at the identified nodes (since \( d(a) \) admits arbitrary repetitions of \( q \) but \( op(d(a)) \) does not) and supplements “missing” trees to \( t \) (since \( \star(q) \) matches \( e \) but \( q \) may not).

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

4.2 The Transformation Algorithm

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

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

For example, let \( r = \ast((a, \langle b, c \rangle)) \) and \( q = \text{sub}(r, 12) \). Then \( r' = \ast((a'^{121}, \langle b^{121}, c^{122} \rangle)) \) and \( q' = \text{sub}(r, 12') = \ast((b^{121}, c^{122})). \) If \( w' = a'^{112}_1 b^{121}_1 c^{122}_1 \), then \( \text{match}(w', q') \) is \( \{(2, 2), (4, 4)\} \).

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

Let \( d \) be a DTD and \( op \) be an update operation to \( d \).

We show the transformation algorithm inferred from \( d \) and \( op \), denoted \( \text{Trans}_\text{op} \), as follows (subroutine \( \text{Trans}_1 \), \( \text{Trans}_2 \), and \( \text{Trans}_3 \) are shown later).

\( \text{Trans}_\text{dp} \) of \( t \)

Input: a tree \( t \) valid against \( d \).

Output: a tree valid against \( op(d) \).

1. If \( t \) is valid against \( op(d) \), then return \( t \).
2. Otherwise, do the following.
   a. If \( op \) is of type 1a, then return \( \text{Trans}_1 \) of \( t \).
   b. If \( op \) is of type 1b, then return \( \text{Trans}_2 \) of \( t \).
   c. If \( op \) is of type 2, then return \( \text{Trans}_3 \) of \( t \).

Let us show three subroutines \( \text{Trans}_1 \), \( \text{Trans}_2 \), and \( \text{Trans}_3 \). We first show \( \text{Trans}_1 \).

\( \text{Trans}_1 \) of \( t \)

1. If \( op = \text{ins}_\text{elm}(a, b, vi) \), then for each node \( n \) labeled by \( a \) in \( t \), do the following. Note that by Condition (1) of Lemma 1, \( l(d(a), vi) = \ast^\prime \).
   a. Let \( n_1, \ldots, n_m \) be the children of \( n \) in \( t \). Find a superscripted supersequence \( w' \) of \( l(n_1) \cdots l(n_m) \) w.r.t. \( b^k \) such that \( w' \in L((op(d)(a))^\prime) \), where \( b^k \) is the superscripted label in \( op(d(a)) \) inserted by \( op \).
   b. For each \( (j, j) \in \text{match}(w', b^k) \), create a new tree valid against \( op(d) \) whose root is labeled by \( b \) and insert the tree into \( t \) as the \( j \)-th child of \( n \).

2. If \( op = \text{del}_\text{elm}(a, vi) \), then for each node \( n \) labeled by \( a \) in \( t \), do the following.
   a. Let \( n_1, \ldots, n_m \) be the children of \( n \) in \( t \). Find a superscripted word \( w' \) such that \( w' \in L(d(a)) \) and that \( (w')^k = l(n_1) \cdots l(n_m) \).
   b. By definition \( \text{sub}(d(a), vi) \) is a single superscripted label, say \( b^k \). For each \( (j, j) \in \text{match}(w', b^k) \), delete the subtree rooted at \( n_1 \) from \( t \).

\( \text{We assume that the text values of such a new tree are empty since they can hardly be estimated.} \)
3. Return $t$ transformed above.

In step (1a) we have to find a superscripted supersequence $w'$ of $\{(l(n_1) \cdots l(n_m))\}$ such that $w' \in L((op(d(a))^t))$, which can be obtained in $O(|op(d(a)|^2 + |w'|)$ time, where $|op(d(a)| = |d(a)| + 1$ (details are shown in Appendix A).

In step (1b), a new tree can be constructed in $O(|d||t|)$ time. In step (2a), a superscripted word $w'$ such that $w' \in L(d(a)^t)$ and that $|w'| = l(n_1) \cdots l(n_m)$ can easily be obtained by using the Glushkov automaton of $d(a)$ (defined in Sect. 6.2), which requires $O(|d(a)|^2 + |w'|)$ time. Thus $\text{Trans}1_{d,op}(t)$ runs in $O(|t| \cdot |d(a)|^2)$ time.

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

$\bigtriangledown$ (t_0 \Rightarrow t_1)$ Let $d$ be the DTD in Fig. 4 (a). Then $d_{(\text{staff})} = \text{(name, age, zip, email)}$. Since $op_1 = \text{del_elm(staff, 2)}$, $t_0$ is transformed by step 2 of $\text{Trans}1_a$. Consider the node $n_1$ of $t_0$. Since $d(\text{staff})' = \text{(name1, age2, zip3, email1)}$, the superscripted word $w'$ of $l(n_2)(l(n_2))(l(n_2))(l(n_2))(l(n_2))$ is “name age zip email” such that $w' \in d(\text{staff})'$ is “name_1 age_2 zip_3 email_1”. Thus $\text{match}(w', age) = \{(2, 2)\}$, thereby the second child $n_3$ of $n_1$ is deleted from $t_0$.

(t_1 \Rightarrow t_2)$ Let $d_1$ be the DTD in Fig. 4 (b). Then $d_{1(\text{staff})} = \text{(name, zip, email)}$. Since $op_2 = \text{ins_elm(staff, street, 2)}$, we have $d_{2(\text{staff})} = \text{(name, street, zip, email)}$ and $t_1$ is transformed by step 1 of $\text{Trans}1_a$. Consider the node $n_1$ in $t_1$. Since $d_1(\text{street})' = \text{(name1, street2, zip3, email4)}$, the superscripted supersequence $w'$ of $l(n_2)(l(n_2))(l(n_2))$ is “name zip email” w.r.t. street such that $w' \in L(d_2(\text{street})')$ is “name_1 street_2 zip_3 email_4”. Thus $\text{match}(w', \text{street}) = \{(2, 2)\}$, and a new node $n_8$ labeled by “street” is inserted into $t_1$ as the second child of $n_1$.

We next show $\text{Trans}1_{b}$.

$\text{Trans}1_{d,op}(t)$

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

   a. Let $n_1, \ldots, n_m$ be the children of $n$ in $t$. Find a superscripted word $w'$ such that $w' \in L(d(a)^t)$ and that $|w'| = l(n_1) \cdots l(n_m)$.

   b. By definition $\text{sub}(d(a), u)^t$ is a single superscripted label, say $b^h$. For each $(j, k) \in \text{match}(w', b^h)$, delete the $j$th child $n_j$ of $n$ from $t$.

2. If $op = \text{agg_elm}(a, b, u)$, then for each node $n$ labeled by $a$, do the following.

   a. Let $n_1, \ldots, n_m$ be the children of $n$ in $t$. Find a superscripted word $w'$ such that $w' \in L(d(a)^t)$ and that $|w'| = l(n_1) \cdots l(n_m)$.

   b. For each $(j, k) \in \text{match}(w', \text{sub}(d(a), u)^t)$, insert a new node labeled by $b$ as the parent of $n_j, \ldots, n_k$ into $t$.

3. Return $t$ transformed above.

$\text{Trans}1_{d,op}(t)$ runs in $O(|t| \cdot |d(a)|^2)$ time.

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

$(t_3 \Rightarrow t_4)$ Let $d_3$ be the DTD in Fig. 4 (d). Then $d_3(\text{staff}) = \text{(name, street, zip, email)}$. Since $op_3 = \text{agg_elm(staff, address, 2)}$, $t_3$ is transformed by step 2 of $\text{Trans}1_a$. Consider the node $n_1$ in $t_3$. Since $d_3(\text{staff})' = \text{(name1, street2, zip2, email3)}$, the superscripted word $w'$ of $l(n_3)(l(n_4))(l(n_5))$ is “name street zip email” such that $w' \in L(d_3(\text{staff})')$ is “name_1 street_2 zip_2 email_3”. Thus we have $\text{match}(w', \text{street}2, \text{zip}2, \text{email}3) = \{(2, 3)\}$. Therefore, a new node $n_9$ labeled by “address” is inserted as the parent of the second and third children $n_8, n_3$ of $n_1$.

$(t_4 \Rightarrow t_5)$ Let $d_4$ be the DTD in Fig. 4 (e). Then $d_4(\text{staff}) = \text{(name, address, email)}$. Since $op_3 = \text{ext_elm(staff, 1)}$, $t_4$ is transformed by step 1 of $\text{Trans}1_a$. Consider the node $n_1$ in $t_4$. Since $d_4(\text{address})' = \text{(name1, address2, email4)}$, the superscripted word $w'$ of $l(n_3)(l(n_4))(l(n_5))$ is “name address email” such that $w' \in L(d_4(\text{address})')$ is “name_1 address_2 email_4”. Therefore, $\text{match}(w', \text{name}) = \{(1, 1)\}$. Thus the first child $n_2$ of $n_1$ is deleted.

Finally, we show $\text{Trans}2_a$. Note that for any DTD $d$ and any tree $t$ valid against $d$, by Lemma 1 $t$ is always valid against $op(d)$ if $op = \text{ins_opr}(a, opr, vi, vj)$. Thus we do not have to transform $t$ if $op = \text{ins_opr}(a, opr, vi, vj)$, and it suffices to consider the case of $op = \text{del_opr}(a, u)$. We need a definition. Let $w'$ be a superscripted word and $b^h$ be a superscripted label. Then a variant of $w'$ w.r.t. $b^h$ is a superscripted word obtained by deleting some $b^h$'s from $w'$ and inserting $b^h$'s into $w'$ at arbitrary positions.

$\text{Trans}_2_{d,op}(t)$

1. By Conditions (4) and (5) of Lemma 1, $op = \text{del_opr}(a, u)$ and $l(d(a), u) = '*'$. Thus $\text{sub}(d(a), u) = s(q) \text{ for some regular expression } q$. For simplicity, we assume that $q$ is a single label $b$ (the other case can be handled similarly) and let $q = b^h$. For each node $n$ in $t$ labeled by $a$, do the following.

   a. Let $n_1, \ldots, n_m$ be the children of $n$ in $t$. Find a superscripted word $w'$ such that $w' \in L(d(a)^t)$ and that $|w'| = l(n_1) \cdots l(n_m)$.

   b. Find a variant $w''$ of $w'$ w.r.t. $b^h$ such that $w'' \in L((op(d(a))^t)$.

2. Return $t$ transformed above.
A variant $w''$ of $w'$ in step (1b) can be obtained in $O(d(a)^2 + |w'|)$ time (details are shown in Appendix B), and tree $t_0$ can be obtained in $O(|d|)$ time. Thus Trans$_{2,d,op}(t)$ runs in $O(|d| \cdot d(a)^2 + |d| \cdot |d|)$ time.

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

**Transform$_{d,s}(t)$**

Input: a tree $t$ valid against $d$.

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

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

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

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

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

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

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

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

Let $d$ be a DTD, $s = op_1 \cdots op_n$ be an update script to $d$, and let $d_{i-1} = op_{i-1}(op_{i-2}(\cdots (op_1(d)) \cdots))$. If $op_i = ins_elm(a,b,u)$ and for some ancestor position $v$ of $u \{d_{i-1}(a),v\} = \{'s'\}$, then $op_i$ is starred. If for some $i,j$ with $i \leq j$, $op_i = ins_elm(a,b,u)$, $op_j = ins_elm(c,e,v)$, and $c$ occurs in $d_{j-1}(b)$, then $op_j$ is nesting. We define that $D_{\max} = \max(|d|,|op_1(d)|,\cdots,|s(d)|)$. We now have the following theorem.

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

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

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

- For a tree $t'$, the number of nodes inserted into $t'$ by an agg_elm($a,b,u$) operation is at most the number of nodes in $t'$ labeled by $a$. Moreover, label $b$ cannot occur in $t'$ by definition.
- Let $op$ be an ins_elm() operation that is neither nesting nor starred. For each node $n$ in a tree $t'$, at most one new tree, say $t''$, is inserted as the child of $n$ by $op$, where $|t''| \in O(D_{\max})$. Thus, the number of nodes inserted into $t'$ by $op$ is in $O(t' \cdot D_{\max})$. Therefore, if $s$ contains $k$ ins_elm()'s and $l$ agg_elm()'s, the size of input tree $t$ grows at most $k \cdot l \cdot D_{\max} \cdot |t|$ by $s$. This and Lemma 2 imply that Transform$_{d,s}(t)$ runs in $O(n \cdot (k \cdot l \cdot D_{\max} \cdot |t| \cdot D_{\max}^2)$ time, where $k \leq n$ and $l \leq n$. $\Box$

It is open whether Transform$_{d,s}(t)$ runs in polynomial time in the case where Condition (C1) does not hold.

5. **Sufficient Conditions for Unambiguous Transformation**

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

5.1 **PSPACE-hardness**

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

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

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

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

We have $op(d)(a) = +(+(\cdot b),+(b),r_1)$, $+(\cdot b),+(b),s(b),r_2)$). Let $T$ be the transformation algorithm inferred from $d$ and $s$. We show that $r_1$ includes $r_2$ iff $T$ is unambiguous.
Assume first that $r_1$ includes $r_2$. Then it is easy to show that for any tree $t$ valid against $d$, $t$ is also valid against $s(d)$, and thus $t$ is not transformed by $T$. Thus $T$ is unambiguous.

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

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

### 5.2 Sufficient Conditions

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

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

**U1** In step (1b) of $Trans1A$, $match(w', b^h)$ depends on the superscripted supersequence $w'$ selected in step (1a).

**U2** In step (1b) of $Trans1A$, there may be more than one trees valid against $op(d)$ whose root is labeled by $b$.

**U3** In step (2b) of $Trans1A$, $match(w', b^h)$ depends on the superscripted word $w'$ selected in step (2a). A similar argument also applies to steps (1b) and (2b) of $Trans1B$.

**U4** In step (1b) of $Trans2$, there may be more than one variant $w''$ of $w'$.

Let us first show a simple sufficient condition related to (U2). We define a DTD that admits exactly one valid tree.

For a DTD $d$ and a label $b$, we say that $d(b)$ is simple if for any label $a$ reachable from $b$, $|L(d(a))| = 1$. For example, let $d$ be a DTD, where $d(a) = (b, e, c, d(b) = (f, g)$, and $d(c) = d(e) = d(f) = d(g) = e$. Then $d(a)$ is simple. We have the following lemma.

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

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

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

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

**Example 4:** Let $d$ be a DTD, $d(a) = (c(b), c(b))$, and $op = ins_{elm}(a, c, 2)$. Then $(op(d)(a))^h = (c(b 1), c^2, c(b^3))$. Consider a word $bb \in L(d(a))$. There are three superscripted supersequences of $bb$ w.r.t. $c^2$ belonging to $L((op(d)(a))^h)$: $c^2 b 31, b^3 c 31, b^3 b^3 1$ and $b^3 b^3 1 c$. Thus $d(a)$ is not unambiguous w.r.t. the insertion of $c^3$.

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

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

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

**Theorem 5:** Let $d$ be a DTD, $s = op_1 \cdots op_n$ be an update script to $d$, and $d_j = op_j \cdot \cdots (op_1(d)) \cdot \cdots$. Then the transformation algorithm inferred from $d$ and $s$ is unambiguous if the following condition holds.

**R1** For every $1 \leq j \leq n$, if $op_j = ins_{elm}(a, b, vi)$ and $d_{j-1}(a, v) = \ast \ast$ for some $a, b \in \Sigma$ and some $vi \in pos(d_{j-1}(a))$, then $d_j(b)$ is simple.

2. Assume that $d_0, \cdots, d_n$ are one-unambiguous. Then the
transformation algorithm inferred from \(d\) and \(s\) is unambiguous if the following condition holds as well as Condition (R1).

R2) For every \(1 \leq j \leq n\),
- if \(op_j = \text{ins}_\text{elm}(a, b, vi)\) and \(l(d_{j-1}(a), v) = \cdot\) for some \(a, b \in \Sigma\) and some \(vi \in \text{pos}(d_{j-1}(a))\), then \(d_{j-1}(a)\) is unambiguous w.r.t. the insertion of \(b^h\), where \(b^h\) is the superscripted label inserted by \(op_j\), and
- if \(op_j = \text{del}_\text{op}(a, u)\) for some \(a \in \Sigma\) and some \(u \in \text{pos}(d_{j-1}(a))\), then Condition (5) of Lemma 1 holds w.r.t. \(d_{j-1}\) and \(op_j\). \(\square\)

Finally, let us consider a sufficient condition for the unambiguity problem, without any assumption on DTD such as “limited” and “one-unambiguous”. We give definitions related to (U3). We define a regular expression such that \(\text{match}(w', \text{sub}(r, u'))\) can unambiguously determined. Let \(r\) be a regular expression, \(u \in \text{pos}(r)\) be a position, 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 superscripted words \(w', \omega\) such that \((w')^h = (\omega)^h = w\) and that \(w', \omega' \in L(r')\), \(\text{match}(w', \text{sub}(r, u')) = \text{match}(\omega', \text{sub}(r, u'))\). 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 \(d\) be the DTD in Fig. 2 (a) and let \(d(b) = sp(\cdot(s, s, + (a, e)))\), where labels \(b, s, a\) and \(e\) stand for “book”, “section”, and “ack”, respectively. We have \(d(b') = sp(\cdot(s, s, + (a^{131}, e^{132})))\). Let \(q' = \text{sub}(d(b), 1)' = sp(\cdot(s, s, + (a^{131}, e^{132})))\). For a word \(w = sxa \in L(d(b))\), we have two superscripted words \(w' = s^1s^1a^{131}\) and \(w'' = s^1s^1a^{131}\) of \(w\) such that \(w', w'' \in L(d(b'))\). Since \(\text{match}(w', q') = \{(1, 1), (2, 3)\}\) and \(\text{match}(w'', q') = \{(1, 3)\}\), \(d(b)\) is not unambiguous w.r.t. \(q = \text{sub}(d(b), 1)\). \(\square\)

We now obtain a sufficient condition for the unambiguity problem.

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

S1) One of Conditions (1) to (5) of Lemma 1 holds w.r.t. \(d_{j-1}\) and \(op_j\).

S2) \(op_j = \text{ins}_\text{elm}(a, b, vi)\) for some \(a, b \in \Sigma\) and some \(vi \in \text{pos}(d_{j-1}(a))\), \(d_{j-1}(a)\) is unambiguous w.r.t. the insertion of \(b^h\), and \(d_j(b)\) is simple, where \(b^h\) is the superscripted label inserted by \(op_j\).

S3) \(op_j = \text{del}_\text{elm}(a, vi)\) for some \(a \in \Sigma\) and some \(vi \in \text{pos}(d_{j-1}(a))\), and \(d_{j-1}(a)\) is unambiguous w.r.t. \(l(d_{j-1}(a), vi)\).

S4) \(op_j = \text{ext}_\text{elm}(a, u)\) for some \(a \in \Sigma\) and some \(u \in \text{pos}(d_{j-1}(a))\), and \(d_{j-1}(a)\) is unambiguous w.r.t. \(l(d_{j-1}(a), u)\).

S5) \(op_j = \text{agg}_\text{elm}(a, b, u)\) for some \(a, b \in \Sigma\) and some\( u \in \text{pos}(d_{j-1}(a))\), and \(d_{j-1}(a)\) is unambiguous w.r.t. \(\text{sub}(d_{j-1}(a), u)\).

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

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

**Example 6:** Let \(d_1\) be a DTD, where
\[
\begin{align*}
  d_1(a) &= \langle b, e \rangle, \\
  d_1(b) &= \langle s(c), + (c, d) \rangle, \\
  d_1(c) &= d_1(d) = d_1(e) = e.
\end{align*}
\]

Let \(s = op_1 \cdot op_2\) be an update script to \(d_1\), where \(op_1 = \text{agg}_\text{elm}(b, f, 21)\) and \(op_2 = \text{del}_\text{elm}(a, 1)\). Let \(d_2 = op_1(d_1)\) and \(d_3 = s(d_2)\). We have
\[
\begin{align*}
  d_2(a) &= \langle b, e \rangle, \\
  d_2(b) &= \langle s(c), + (f) \rangle, \\
  d_2(f) &= + (c, d), \\
  d_2(c) &= d_1(d) = d_1(e) = e,
\end{align*}
\]
and
\[
\begin{align*}
  d_3(a) &= \langle e \rangle, \\
  d_3(e) &= e.
\end{align*}
\]

Consider how the tree \(t_1\) in Fig. 6 (a) is transformed according to \(s\). It is easy to verify that \(d_1(b)\) is not unambiguous w.r.t. \(\text{sub}(d_1(b), 21)\), thereby Condition (S5) does not hold. Thus, according to \(op_1\) \(t_1\) can be transformed into two trees \(t_2\) and \(t_3\) (Fig. 6 (b,c)). Then, according to \(op_2\), both \(t_2\) and
t₁ are transformed into the same tree t₁ (Fig. 6 (d)). In general, it is easy to see that for any tree t valid against d₁, |TSₜ₁(ₜ)| = 1 whereas op₁ satisfies none of Conditions (S1) to (S5).

6. Testing the Sufficient Conditions

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

P1) In Condition (R1) of Theorem 5 and Condition (S2) of Theorem 6, we have to determine if a content model is simple.

P2) In Conditions (S3), (S4), and (S5) of Theorem 6, we have to determine if a regular expression e(a) is unambiguous w.r.t. a subexpression of e(a).

P3) In Condition (R2) of Theorem 5 and Condition (S2) of Theorem 6, we have to determine if a regular expression e(a) is unambiguous w.r.t. the insertion of bₙ.

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

6.1 Checking the Simplicity of a DTD

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

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

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

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

1. If is ε, then I_r = F_r = {E}, where ε is a new label (I_r and F_r contain ε if ε ∈ L(r).
2. If r = a for some a ∈ Σ, then I_r = F_r = |a|, where a is the superscripted label such that r = a.
3. If r = +(r₁, ⋯, rₙ), then I_r = I_r₁ ∪ ⋯ ∪ I_rₙ and F_r = F_r₁ ∪ ⋯ ∪ F_rₙ.
4. If r = -(r₁, ⋯, rₙ), then
   
   \[ I_r = (I_r₁ - \{E\}) \cup \cdots \cup (I_rₙ - \{E\}) \cup I_r, \]
   
   \[ F_r = F_r₁ \cup (F_rₙ - \{E\}) \cup \cdots \cup (F_rₙ - \{E\}), \]
   
   where
   
   \[ i = \begin{cases} n & \text{if } E \in I_r \text{ for every } 1 \leq k \leq n, \\ \min\{k | E \notin I_r, 1 \leq k \leq n\} & \text{otherwise}. \end{cases} \]
   
   \[ j = \begin{cases} 1 & \text{if } E \in F_r \text{ for every } 1 \leq k \leq n, \\ \max\{k | E \notin F_r, 1 \leq k \leq n\} & \text{otherwise}. \end{cases} \]
5. If r = *r₁, then I_r = I_r₁ ∪ {E} and F_r = F_r₁ ∪ {E}.

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

1. If r' = aᵢ, then Succ(aᵢ, r') = ∅.
2. If r' = +(r'₁, ⋯, r'ₙ) and aᵢ occurs in r'ₖ (1 ≤ k ≤ n), then Succ(aᵢ, r') = Succ(aᵢ, r'ₖ).
3. If r' = -(r'₁, ⋯, r'ₙ) and aᵢ occurs in r'ₖ (1 ≤ k ≤ n), then
   
   \[ \text{Succ}(aᵢ, r') = \begin{cases} \text{Succ}(aᵢ, r'ₖ) & \text{if } aᵢ \neq F_rₖ, \\ \text{Succ}(aᵢ, r'ₖ) \cup (I_rₖ - \{E\}) \cup \cdots \cup (I_rₙ - \{E\}) & \text{if } k < n \text{ and } aᵢ \neq F_rₙ, \end{cases} \]
   
   where
   
   \[ j = \begin{cases} n & \text{if } E \in I_r \text{ for every } k + 1 \leq i \leq n, \\ \min\{i | E \notin I_r, k + 1 \leq i \leq n\} & \text{otherwise}. \end{cases} \]
4. If r' = *r₁, then
   
   \[ \text{Succ}(aᵢ, r') = \begin{cases} \text{Succ}(aᵢ, r'₁) & \text{if } aᵢ \neq F_r₁, \\ \text{Succ}(aᵢ, r'₁) \cup (I_r₁ - \{E\}) & \text{otherwise}. \end{cases} \]

The Glushkov automaton of r is a 5-tuple (Q, Σ, δ, qᵢ, F), where Q is the set of states, δ is the transition function, qᵢ is the initial state, and F is the set of final states defined as follows.

- Q = sym(r') ∪ \{qᵢ\},
- δ(qᵢ, a) = |a| \cup a if a ∈ I_r, (a)^k = a for every a ∈ Σ, and δ(aᵢ, a) = |a|^k if aᵢ ∈ Succ(aᵢ, r'), (a)^k = a,
- F = \{ F_r ∪ \{qᵢ\} \} if a ∈ L(r),
- otherwise.

Example 7: Let r = -(a, +(a, b))). Then r' = -(a', +(a(b₁, b₂))]. We have I_r = |a'|, F_r = |a|^b₂, |b₂|, and Succ(aᵢ, r') = Succ(b₂, r') = |b|^b₂. The Glushkov automaton of r is illustrated in Fig. 7.

□

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

We test the unambiguity by using a graph called testing graph\(^*\) of a Glushkov automaton. Let G_r = (Q, Σ, δ, qᵢ, F)

\(^*\)We use a modified version of testing graph, originally defined in [8].
be the Glushkov automaton of \( r \). A pair \((a^i, a^j)\) of states in \( Q \) is compatible if (i) \( a^i = a^j = q^j \), 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 \((a^k)^{\prime} = (a^l)^{\prime} = a \). Then the testing graph of \( G_r \) is a graph \( T(G_r) = (N, E) \), where

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

The following lemma can be checked as follows.

**Lemma 6:** Let \( r \) be a regular expression and \( G_r = (Q, \Sigma, \delta, q^j, F) \) be the Glushkov automaton of \( r \). There are superscripted words \( w^i, w'' \in L(r^\prime) \) such that \((w^i)^{\prime} = (w'')^{\prime}\) and that \( |w^i| = |w''| = 1 \) if there is a path \((q^i, q^j) \rightarrow (a^{i^k_1}, a^{j^k_1}) \rightarrow \cdots \rightarrow (a^{i^k_4}, a^{j^k_4}) \) in \( T(G_r) \) such that \( w''[k] = a^i \) and \( w''[k] = a^j \) for every \( 1 \leq k \leq l \) and that \( a^i, a^j \in F \).

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

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

1. For any node \((a^i, a^j)\) in \( T(G_r) \) from which some accepting node is reachable, either \( a^i, a^j \in \text{sym}(q^j) \) or \( a^i, a^j \in \text{sym}(r^\prime) \setminus \text{sym}(q^j) \).
2. For any edge \((a^i, a^j) \rightarrow (a^k, a^l) \) in \( T(G_r) \) such that some accepting node is reachable from \((a^i, a^j)\), if \( a^i, a^k, a^l \in \text{sym}(q^j) \), then either
   a. \( a^i \notin \text{Succ}(a^k, q^j) \) and \( a^j \notin \text{Succ}(a^l, q^j) \), or
   b. \( a^k \in \text{Succ}(a^i, q^j) \) and \( a^l \in \text{Succ}(a^j, q^j) \).

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

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

**Lemma 7:** Let \( r \) be a regular expression and \( q \) be a subexpression of \( r \). Then whether \( r \) is unambiguous w.r.t. \( q \) can be checked in \( O(|r|^3) \) time. In particular, if \( r \) is one-unambiguous, then this check can be done in \( O(|r|^2) \) time.

**Proof:** Assume first that \( r \) is not one-unambiguous. Then the Glushkov automaton \( G_r \) of \( r \) can be constructed in \( O(|r|^3) \) time [5], and the testing graph \( T(G_r) \) can be constructed in \( O(|r|^2) \) time. Moreover, the condition in Theorem 7 can be checked in linear time w.r.t. \( T(G_r) \). Second, if \( r \) is one-unambiguous, then the lemma follows from the fact that \( G_r \) can be constructed in linear time.

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

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

To check this unambiguity we slightly modify the testing graph of a Glushkov automaton. Let \( r \) be a regular expression, \( G_r = (Q, \Sigma, \delta, q^j, F) \) be the Glushkov automaton of \( r \), and \( b^h \in Q \) be a superscripted label. We first define a contracted transition function \( \delta^r \) 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 \( a^i \) into one transition from \( a^i \) to \( a^i \). Formally, \( \delta^r \) is defined so that for any \( a^i, a^j \in Q, a^i \in \delta(a^i, a) \) iff

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

where \( a = (a^i)^{\prime} \) and \( b = (b^h)^{\prime} \). A pair \((a^i, a^j)\) of states is compatible (w.r.t. \( \delta^r \)) if (i) \((a^i = a^j = q^j) \), or (ii) there is a compatible pair \((a^i, a^j)\) such that, \( a^i \in \delta^r(a^i, a), a^j \in \delta^r(a^i, a) \), and that \((a^i)^{\prime} = (a^j)^{\prime} = a \). Then the contracted testing graph of \( G_r \) w.r.t. \( b^h \), denoted \( T_{b^h}(G_r) \), is a graph \((N, E) \), where

\[
N = \{(a^i, a^j) \mid (a^i, a^j) \text{ is a compatible pair of } Q\},
\]

\[
E = \{(a^k, a^l) \rightarrow (a^i, a^j) \mid a^i \in \delta^r(a^i, a), a^j \in \delta^r(a^i, a)\}.
\]

For an edge \((a^i, a^j) \rightarrow (a^i, a^j) \in E \), if (i) \( a^i \in \delta(a^i, a) \) but \( b^h \in \delta(a^i, b) \) and \( b^h \in \delta(b^h, a) \), or (ii) \( a^i \in \delta(a^i, a) \) but \( b^h \in \delta(a^i, b) \) and \( b^h \in \delta(b^h, a) \), then we say that \((a^i, a^j) \rightarrow (a^i, a^j) \) 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. \( b^h \in \delta(a', b), b^h \in \delta(a', b), \) and \( b^h \in F \).
3. (i) \( a' \in F, b^h \in \delta(a', b), \) and \( b^h \in F, \) or (ii) \( a' \in F, b^h \in \delta(a', b), \) and \( b^h \in F \).

In particular, if \( (a', a') \) satisfies Condition (3), then \( (a', a') \) is oddly accepting. Now the unambiguity can be checked as follows.

**Theorem 8**: Let \( a \) be a DTD, \( op = \text{ins}_\text{elm}(a, b, vi) \) be an update operation to \( a \) such that \( l(d(a), v) = \cdot \cdot, b^h \) be the superscripted label inserted by \( op \), and \( G_{\text{opt}(d(a))} \) be the Glushkov automaton of \( \text{opt}(d(a)) \). Then \( d(a) \) is unambiguous w.r.t. the insertion of \( b^h \) iff the following three conditions hold.

1. For any odd edge \( (a^k, a') \in T_{b^h}(G_{\text{opt}(d(a))}), \) no accepting node is reachable from \((a', a')\).
2. \( T_{b^h}(G_{\text{opt}(d(a))}) \) contains no oddly accepting node.
3. \( b^h \notin \delta(b^h, b), \) where \( \delta \) is the transition function of \( G_{\text{opt}(d(a))}. \)

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

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

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

\[ w'[i] = w''[i] \text{ for every } 1 \leq i \leq k - 1, \text{ but } w'[k] = b^h \text{ and either } w''[k] \neq b^h \text{ or } |w''| = k - 1. \]

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

*The case where \( |w''| \geq k \):* We have \( w''[k] \neq b^h \). Suppose first that \( w'[k - 1] = w''[k - 1] \neq b^h \). Since \( w'[k] \neq b^h \), there is an index \( k' > k \) such that \( w'[k'] = w''[k] \).

If \( k' = k + 1 \), then Condition (1) does not hold (since \( (w'[k - 1], w''[k - 1]) \rightarrow (w'[k'], w''[k]) \) is an odd edge). If \( k' > k + 1 \), then \( w'[k] = w''[k + 1] = b^h \), therefore Condition (3) does not hold. Suppose next that \( w'[k - 1] = w''[k - 1] = b^h \).

Then \( w'[k - 1] = w''[k] = b^h \); thus Condition (3) does not hold.

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

The following lemma can be shown similarly to Lemma 7.

**Lemma 8**: Let \( a \) be a DTD and \( op = \text{ins}_\text{elm}(a, b, vi) \) be an update operation such that \( l(d(a), v) = \cdot \cdot \). Then whether \( d(a) \) is unambiguous w.r.t. the insertion of \( b^h \) can be checked in \( O(|d(a)|^2) \) time, where \( b^h \) is the superscripted label inserted by \( op \). In particular, if \( d(a) \) is one-unambiguous, then this check can be done in \( O(|d(a)|^2) \) time. \( \Box \)

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

**Theorem 9**: Let \( d \) be a DTD, \( s = op_1 \cdots op_n \) be an update script to \( d \), \( k \) be the number of \( \text{ins}_\text{elm}() \) operations in \( s \), \( l \) be the number of \( \text{del}_\text{op}() \) operations in \( s \), and \( R \) be a regular expression with maximum width occurring in \( d, op_1(d), \ldots, s(d) \).

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

**Proof**: Consider first (1). In order to check if \( d, op_1(d), \ldots, s(d) \) are simple (limited, one-unambiguous), it suffices to check (i) whether \( d \) is simple (resp., limited, one-unambiguous) and (ii) whether \( s \) is simple (resp., limited, one-unambiguous) for every updated content model \( r \) in \( op_1(d), \ldots, s(d) \). Whether a regular expression \( r \) is one-unambiguous can be determined in \( O(|r|^2) \) time \([6]\). By this and the discussion about Lemma 5, it is easy to show that (i) can be checked in \( O(|d|^2) \) time and (ii) can be checked in \( O(n \cdot |R|_{\text{max}}^5) \) time. Second, (2) follows from Lemma 5. Consider (3). By Condition (5) of Lemma 1 we have to check if \( L(q) = \{e\} \) for a subexpression \( q = \text{sub}(d(a), u_1) \) of \( d(a) \), which can be determined in \( O(|q|) \) time by using the Glushkov automaton of \( q \). This and Lemmas 5 and 8 imply that (3) holds. Finally, (4) holds by Lemmas 5, 7, and 8. \( \Box \)

### 7. Conclusion

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

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

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

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

Acknowledgement

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

References


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

Let \( d \) be a DTD, \( op = \text{ins} \_ \text{elm} \,(a,b,v) \) with \( l(d,a), v = \{., \} \), and \( b^k \) be the superscripted label in \( op(d)(a) \) inserted by \( op \).

Let \( w \in L(d(a)) \) be a word and consider finding a superscripted supersequence \( w'' \) of \( w \) w.r.t. the insertion of \( b^k \).

We use a function \( \gamma \) that maps each superscripted label in \( d(a) \) to its corresponding superscripted label in \( op(d)(a) \). Let \( w' \in L(d(a)) \) be a superscripted word such that \( w'^k = w \) and let \( w'' = \gamma(w'[1]) \cdots \gamma(w'[|w'|]) \). Then it suffices to insert \( b^k \)'s into \( w'' \) at appropriate positions so that the resulting superscripted word matches \( op(d)(a) \). Thus a superscripted supersequence of \( w \) w.r.t. the insertion of \( b^k \) can be obtained as follows.

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

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

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

Appendix B: Finding a Variant of a Word

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

1) insert “missing” $b^h$’s into $w''$ and
2) delete “excess” $b^h$’s from $w''$

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

Appendix C: Proof of Lemma 4

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

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