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

Nobutaka Suzuki Yuji Fukushima

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

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

Keywords: XML, data transformation, edit operation, Glushkov automaton, schema evolution

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

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

For example, let us consider DTD D1 shown in Fig. 1(a). Suppose that D1 is updated to a new DTD D2 by an edit script that (i) deletes “age” and (ii) aggregates a subsequence “(address, zip, country)” of the content model of “staff” into “addr_info” (Fig. 1(b)). Then for any XML document t valid against D1, the transformation algorithm inferred from D1 and the edit script

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

For example, the XML document t1 in Fig. 1(c) (represented as a tree without text strings) is transformed into t2 in Fig. 1(d), which is valid against D2.

Let D1 be a DTD and S be a set of XML documents valid against D1. Suppose that a user updated D1 to a new DTD D2 by applying some edit script s to D1. Since s concretely represents the user’s intention how to modify D1, s strongly suggests how to transform each document in S. Therefore, if we can obtain a transformation algorithm T inferred from D1 and s, then we can say that T is a transformation algorithm that faithfully reflects the user’s intention represented by s.

However, depending on a DTD D and an edit script s to D, the transformation algorithm T inferred from D and s may become “ambiguous”, that is, for some document t valid against D T cannot unambiguously determine which elements in t should be deleted/inserted (conversely, if there is no such a tree, then T is called “unambiguous”). For example, let us consider DTD D3 (Fig. 2(a)). Suppose that D3 is updated to a new DTD D4 by an edit script s that aggregates subexpression “(section+,ack?)” of the content model of “book” into “chapter” (Fig. 2(b)). For the tree t4 in Fig. 2(c), we have two alternatives t4, t5 according to the positions at which “chapter” elements should be inserted (Fig. 2(d,e)). Thus T is ambiguous (T outputs one of t4 and t5 arbitrarily). In general, an ambiguous transformation algorithm is undesirable since it may delete elements that should not be deleted and may insert elements at unexpected positions. Therefore, for a DTD D and an edit script s, we should be able to decide if the transformation algorithm inferred from D and s is unambiguous.

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

Related Work
The problem of finding a mapping that identifies corresponding elements in two schemas (schema matching), the problem of finding a query that achieves actual data transformation between schemas (query discovery), and other related problems have greatly been studied, e.g., (Arenas & Libkin 2005, Bohannon et al. 2005, Kuikka et al. 2002, Miller et al. 2001, Milo...
Let $\Sigma$ be a set of labels. In order to define edit operations to a DTD concisely, each regular expression is represented as a term in prefix notation. Formally, a regular expression over $\Sigma$ 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 $\cdot(r_1, \ldots, r_n)$ 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 occurrences (or positions) of $r$, denoted $occ(r)$, is defined as follows.

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

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

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

- If $r = \epsilon$ or $r = a$ for some $a \in \Sigma$, then $l(r, \lambda) = r$ and $sub(r, \lambda) = r$.
- If $r = op(r_1, \ldots, r_n)$ with $op \in \{+, \cdot\}$, and

2 Definitions

An XML document is modeled as an ordered labeled tree (attributes are omitted). Each node in a tree represents an element. A text node is omitted, in other words, each leaf node has implicitly has a text node. By $l(n)$ we mean the label of node $n$. In what follows, we use the term tree when we mean ordered labeled tree.
modify existing content models, and that of type 3 is an aggregation of type 1 and 2 operations; the edit operations of types 1 and 2 are applied to leaf nodes and a root node of type 1. Let \( \delta \) be an occurrence of \( \tau \). We define that \( v \in \delta \) if \( v \in \delta \). Then \( l(r, u) = l(r, v) \) and
\[
\text{sub}(r, u) = \text{sub}(r, v).
\]

- If \( r = r_1 \), and
  - if \( u = \lambda \), then \( l(r, u) = \text{op} \) and \( \text{sub}(r, u) = r \).
  - if \( u = v \) for some \( 1 \leq j \leq n \) and some \( v \in \delta \), then \( l(r, u) = l(r_j, v) \) and \( \text{sub}(r, u) = \text{sub}(r_j, v) \).

For example, in Fig. 3 l(r, 1) = ‘+’, l(r, 21) = d, and \( \text{sub}(r, 1) = + (a, b, c) \).

Let \( w \) be a word (or string) over \( \Sigma \). By \([w]\) we mean the length of \( w \), and by \([w][i] \) we mean the ith label of \( w \). We define that \( w[i, j] = [w][i+1] \cdots [w][j] \) (\( 1 \leq i \leq j \leq [w] \)). For example, if \( w = \text{sukubaka} \), then \([w][2, 5] = \text{sukub} \).

A DTD is a tuple \( D = (d_1, a_0) \), where \( d_1 \) is a DTD (a possibly partial) mapping from \( \Sigma \) to the union of the set of regular expressions over \( \Sigma \) and \{#PCDATA\}, and \( a_0 \) in \( \Sigma \) is the start label. For a label \( a \in \Sigma \), \( d_1(a) \) is the content model of \( a \). A tree \( t \) is valid against \( D \) if (v1) the root of \( t \) is labeled by \( a_0 \), (v2) for each internal node in \( t \) the sequence of labels on the children of \( n \) is in \( L(d_1((n))) \), and (v3) for each leaf node in \( n \) in \( d_1((n))) = \{\text{PCDATA}\} \). We say that a DTD \( D_2 \) includes a DTD \( D_1 \) if any tree \( t \), \( t \) is valid against \( D_2 \) whenever \( t \) is valid against \( D_1 \). For labels \( a_1, a_2 \in \Sigma \), \( a_1 \) is reachable from \( a_2 \) if (i) \( a_1 = a_2 \) or (ii) \( a_1 \) occurs in \( d_1(a_2) \) for some label \( a_2 \) reachable from \( a_1 \). A DTD is cyclic if for some labels \( a_1, a_2 \) \( a_1 \) is reachable from \( a_2 \) and vice versa.

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

Let \( r \) be a regular expression. By \( r' \) we mean the \( \text{subscripted} \) regular expression resulting from \( r \) by subscripting each label in \( r \) by the corresponding occurrence. By \( \text{sym}(r') \) we mean the set of subscripted 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 \( \text{sym}(r') = \{a_{11}, b_{12}, c_{13}, d_{211}, b_{212}\} \).

Let \( a_i \) be a subscripted label of \( a \). Then by \( a_i \) we mean the label resulting from \( a_i \) by dropping the subscript of \( a_i \), that is, \( a_i = a \).

Let \( w \) be a word (i.e., a sequence of subscripted labels). We define that
\[
w^i = w_1[1]^i \cdots w_1[w_1][i].
\]
For any regular expression \( r \), it holds that \( L(r) = L(r')^3 \), where \( L(r') = \{w_1 \mid w_1 \in L(r')\} \).

### 3 Edit Operations to DTD

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

#### Type 1a: Inserts/deletes an element in a regular expression \( d_1(a) \).
- \( \text{ins}_\text{elm}(a, b, v) \): Inserts a new label \( b \) at \( v \) in \( d_1(a) \), where \( v \in \text{occ}(d_1(a)) \). This is applicable to \( d_1(a) \) only if \( d_1(b) \) is defined, \( l(d_1(b), v) = v' \) for all \( v' \), and \( (v, v'+1) \in \text{occ}(d_1(a)) \).

- \( \text{del}_\text{elm}(a, vi) \): Deletes label \( l(d_1(a), v) \) from \( d_1(a) \). This is applicable to \( d_1(a) \) only if \( \text{sub}(d_1(a), v) \) is a label in \( \Sigma \) and \( \forall k \neq i \) \( l(d_1(a), v) \) has at least one sibling.

#### Type 1b: Aggregates a subexpression of \( d_1(a) \) into a new label, or (ii) extract a label in \( d_1(a) \), say \( b \), by regular expression \( d_1(b) \).
- \( \text{agg}_\text{elm}(a, b, u) \): Aggregates subexpression \( \text{sub}(d_1(a), u) \) into a single label \( b \). Formally, this operation first replaces a subexpression \( \text{sub}(d_1(a), u) \) by label \( b \), then sets \( d_1(b) = \text{sub}(d_1(a), u) \).

- \( \text{ext}_\text{elm}(a, u) \): Extracts a label \( l(d_1(a), u) \) in \( d_1(a) \). Formally, this operation replaces a label \( l(d_1(a), u) \) by a regular expression \( d_1((l(d_1(a), u))) \).

#### Type 2: Inserts/deletes an operator (‘+’, ‘•’ or ‘•’) in \( d_1(a) \).
- \( \text{ins}_\text{op}(a, op, vi, vj) \): Inserts a new operator \( op \) as the parent of the subexpressions at \( vi \) and \( vj \) in \( d_1(a) \). \( op \) is an operator (‘+’, ‘•’ or ‘•’).

- \( \text{del}_\text{op}(a, u) \): Deletes an operator at position \( u \) in \( d_1(a) \). This is applicable to \( d_1(a) \) only if \( l(d_1(a), u) = l(d_1(a), u') \) and \( u = v \) (the operator at \( u \) is "nested") or (ii) \( u \in \text{occ}(d_1(a)) \) and \( u \notin \text{occ}(d_1(a)) \)

#### Type 3: Defines a new content model.
- \( \text{def}_\text{elm}(a, r) \): Sets \( d_1(a) = r \). This is applicable only if \( d_1(a) \) is undefined.

Let \( op \) be an edit operation to a DTD \( D \). By \( op \) we mean the DTD obtained by applying \( op \) to \( D \).

Example 1 Let \( D = (d_0, staff) \) be a DTD, where \( D_0(\text{staff}) = \{\text{name, age, zip, email}\}, \) \( d_0(\text{name}) = \{\text{first name, last name}\}, \) and the other elements are of type #PCDATA. Let \( s = \text{op}_1 \cdots \text{op}_6 \), where \( \text{op}_i = \text{del}_\text{elm}(\text{staff}, 2) \). Then \( D = \text{op}_1(\cdots(\text{op}_6(D) \cdots)) \).

Let \( D \) and \( D_1 = \text{op}_1(\cdots(\text{op}_i(D) \cdots)) \) for \( 1 \leq i \leq 6 \) be illustrated in Fig. 4.
We have the following lemma.

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

1. \( op = \text{ins}_\text{elm}(a, b, vi) \) and either (i) \( l(d_1(a), v) = '+' \) (inserting a label into a '+'-expression) or (ii) \( l(d_1(a), v) = '-' \) and \( c \in L(d_1(b)) \) (inserting an \( \epsilon \)-able expression into a '-'-expression).

2. \( op = \text{del}_\text{elm}(a, vi) \) and either (i) \( l(d_1(a), v) = '+' \) and \( l(d_1(a), vi) \in L(\text{sub}(d_1(a), v_k)) \) for some \( k \neq i \) (i.e., the element at \( vi \) is contained in some sibling subexpression) or (ii) \( l(d_1(a), v) = '-' \) and \( l(d_1(a), vi) = \epsilon \).

3. \( op = \text{ext}_\text{elm}(a, u) \) and \( l(d_1(a), u) \in L(d_1(\text{sub}(d_1(a), u))) \).

4. \( op = \text{ins}_\text{opr}(a, opr, vi, vj) \), where \( opr \) is '+' or '-'.

5. \( op = \text{del}_\text{opr}(a, u) \) and either (i) \( l(d_1(a), u) = '+' \) or (ii) \( l(d_1(a), u) = '-' \) and \( L(\text{sub}(d_1(a), u)) = \{\epsilon\} \).

6. \( op = \text{def}_\text{elm}(a, r) \).

For example, \( \text{ins}_\text{opr}(\text{staff}, 2, 2) \) and \( \text{del}_\text{opr}(\text{staff}, 1) \) in Fig. 4 satisfy Condition (4) of the above lemma.

4 Transformation Algorithm Inferred from a DTD and an Edit Script

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

4.1 Outline

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

1. If \( op \) satisfies one of Conditions (1) to (6) of Lemma 1, then \( t \) is valid against \( op(D) \). Thus \( T \) does nothing.

2. Otherwise, \( T \) modifies \( t \) according to the type of \( op \).

Type 1a: (1) If \( op = \text{ins}_\text{elm}(a, b, u) \), then \( b \) is inserted at \( u \) in \( d_1(a) \). Accordingly, for each position \( p \) in \( t \) at which the \( b \)-label should be inserted, \( T \) creates a new valid tree whose root is labeled by \( b \) and insert the tree at position \( p \) in \( t \). For example, if \( d_1(a) = \langle\text{a}, b, c, \rangle \) 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 \( d_1(a) \) is deleted. Accordingly, \( T \) first identifies the subtrees in \( t \) whose roots match the label \( b \). If the parent of \( b \) is '+' in \( d_1(a) \), then \( T \) just deletes the identifies subtrees. If the parent of \( b \) is '-' in \( d_1(a) \), then \( T \) replaces each identified subtree by a new valid tree whose root is labeled by \( c \), where \( c \) is a sibling of \( b \) in \( d_1(a) \). For example, if \( d_1(a) = +(b, c) \) and \( op = \text{del}_\text{elm}(a, 1) \), then each subtree in \( t \) whose root is labeled by \( b \) is replaced by a new valid tree whose root is labeled by \( c \).

Type 1b: (1) If \( op = \text{ext}_\text{elm}(a, u) \), then \( T \) identifies the internal nodes in \( t \) that match the extracted label in \( d_1(a) \), and deletes the identified internal nodes from \( t \). For example, if \( d_1(a) = +(b, c) \) 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 internal nodes into \( t \) as the parents of sibling nodes that should be aggregated. For example, if \( d_1(a) = +\epsilon \),
Let \( s = o_1 \cdots o_n \) be an edit script to \( D \) and \( D_{n-1} = o_1 \cdots o_{n-2} (o_1(D)) \cdots \), where \( o_1 \) is an edit operation. Then the transformation algorithm inferred from \( D \) and \( s \) is defined as a composition of \( T_1, \cdots, T_n \), where \( T_i \) is the transformation algorithm inferred from \( D_{i-1} \) and \( o_i \).

4.2 The Transformation Algorithm

We first show some definitions. Let \( r \) be a regular expression, \( u \in \text{occ}(r) \), \( q = \text{sub}(r, u) \) be a subexpression of \( r \), \( w \) be a word such that \( w \in L(r) \), and \( w_1 \) be a subworded word such that \( w_1 \in L(r) \) and that \( w_1' = w \). We say that \( w_1[i,j] \) maximally matches \( q' \) if \( w_1[i,j] \subseteq L(q') \) and either \((i) i = 1 \) and \( j = |w_1| \) or \((ii) w_1[i',j'] \not\subseteq L(q') \) for any \( i', j' \) \((1 \leq i' \leq i \leq j \leq j' \leq |w_1|) \) with \( i' < i \) or \( j < j' \). We define that

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

For example, let \( r = *(a, +(b, c)) \) and \( q = +(b, c) \). Then \( r' = *(a_11, +(b_12, c_{12})) \) and \( q' = +(b_12, c_{12}) \). If \( w_1 = a_{11} b_{12} a_{11} c_{12} \), then

\[
\text{match}(w_1, q') = \{(2, 4), (4, 4)\}.
\]

Let \( w \) be a word and \( b_0 \) be a subordinated label. We say that a subworded word \( w_1 \) is a \textit{subordinated supersequence} of \( w \) w.r.t. \( b_0 \) if removing every \( b_0 \) from \( w_1 \) yields a word \( w_2 \) such that \( w_2 = w \).

Let \( D = (d_1, a_0) \) be a DTD; \( D \) be an edit operation to \( D \), and \( op(D) = (d_2, a_0) \). We show the transformation algorithm inferred from \( D \) and \( op \), denoted \text{TRANS}D, \text{TRANS}1b, and \text{TRANS}2 are shown later.

\text{TRANS}D(op(t))

Input: a tree \( t \) valid against \( D \).
Output: a tree valid against \( op(D) \).
1. If \( op \) satisfies one of Conditions (1) to (6) of Lemma 1, then return \( t \).
2. Otherwise, do the following.
   (a) If \( op \) is of type 1a, then return \text{TRANS}1A.D(op(t)).
   (b) If \( op \) is of type 1b, then return \text{TRANS}1B.D(op(t)).
   (c) If \( op \) is of type 2, then return \text{TRANS}2D(op(t)).

Let us show three subroutines \text{TRANS}A, \text{TRANS}1b, and \text{TRANS}2. We first show \text{TRANS}A.

\text{TRANS}A.D(op(t))

1. If \( op = \text{inselm}(a, b, v) \), then for each node \( n \) labeled by \( a \) in \( t \), do the following. Note that by Condition (1) of Lemma 1, \( l(d_1(a), v) = v \), i.e., \( b \) must be inserted into \( v \)’s subtree.

   (a) Let \( n_1, \cdots, n_m \) be the children of \( n \) in \( t \). Find a subordinated supersequence \( w_1 \) of \( l(d_1(a_1)), \cdots, l(n_m) \) w.r.t. \( b_0 \) such that \( w_1 \subseteq L(d_2(a')) \), where \( b_0 \) is the subordinated label in \( d_2(a) \) inserted by \( op \).

   (b) For each \((j, j) \in \text{match}(n_1, b_0)\), create a new tree valid against a DTD \((d_2, b)\) and insert the tree into \( t \) as the \( j \)th child of \( n \).

2. If \( op = \text{delelm}(a, v) \), then for each node \( n \) labeled by \( a \) in \( t \), do the following.
   (a) Let \( n_1, \cdots, n_m \) be the children of \( n \) in \( t \). Find a subordinated word \( w_1 \) such that \( w_1 \subseteq L(d_1(a')) \) and that \( w_2 = l(n_1) \cdots l(n_m) \).

   (b) By definition \( \text{sub}(d_1(a), v) \) is a single label, say \( b_0 \). Thus for each \((j, j) \in \text{match}(n_1, b_0)\), do the following.

   i. If \( l(d_1(a), v) = v' \), then delete the subtree rooted at \( n_j \) from \( t \).
   ii. If \( l(d_1(a), v) = v' \), then choose a “sibling” \( q = \text{sub}(d_1(a), v') \) of \( l(d_1(a), v) \) such that \( k \neq i \). Then create a new forest valid against a (possibly rootless) DTD \((d_2, q)\), and replace the subtree rooted at \( n_j \) by the new forest.

3. Return \( t \) transformed above.

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

Example 2 Figure 5 illustrates how tree \( t_0 \) is transformed by the transformation algorithm inferred from \( D \) and \( s \), where \( D \) and \( s \) are given in Example 1 and Fig. 4. Here, let us consider the intermediate transformations from \( t_0 \) to \( t_1 \) and \( t_2 \) to \( t_3 \) in Fig. 5.

\((t_0 \Rightarrow t_1)\) Let \( D = (d_0, \text{staff}) \) be the DTD in Fig. 4(a). Then \( d_0(\text{staff}) = (\text{name, age, zip, email}) \). Since \( op_1 = \text{delelm}(\text{staff}, 2, 2) \), \( t_0 \) is transformed by step 2 of \text{TRANS}1A. Consider the node \( n_1 \) of \( \text{staff} \). Since \( d_0(\text{staff}') = (\text{name1, age2, zip3, email4}) \), the subordinated word \( w_1 \) of \( l(n_2)(l(n_3))(l(n_4))(l(n_5)) = \text{name1, age2, zip3, email4} \) such that \( w_1 \subseteq d_0(\text{staff}') \) is \( \text{name1, age2, zip3, email4} \), thereby \( \text{match}(w_1, \text{age}0) = (2, 2) \). Thus the second child \( n_3 \) of \( n_1 \) is deleted from \( t_0 \).

\((t_2 \Rightarrow t_3)\) Let \( D_2 = (d_2, \text{staff}) \) be the DTD in Fig. 4(c). Then \( d_2(\text{staff}) = (\text{name, (street, zip, email}) \). Since \( op_3 = \text{inselm}(\text{staff}, \text{street}, 21) \), we have \( d_3(\text{staff}) = (\text{name, (street, zip, email}) \) and \( t_2 \) is transformed by step 1 of \text{TRANS}1A. Consider the node \( n_1 \) in \( t_2 \). Since \( d_3(\text{staff}') = (\text{name1, \text{street}21, \text{zip}22, \text{email}1}) \), the subordinated supersequence \( w_1 \) of \( l(n_2)(l(n_3))(l(n_4)) = \text{name1, street21, zip22, email1} \) w.r.t. \text{street}21 such that \( w_1 \subseteq d_3(\text{staff}') \) is \( \text{name1, street21, zip22, email1} \), thereby \( \text{match}(w_1, \text{street}21) = (2, 2) \). Thus a new node, say \( n_8 \), labeled by “street” is inserted into \( t_2 \) as the second child of \( n_1 \).

Let us next show \text{TRANS}1b that is called if \( op \) is of type 1b.

\text{TRANS}1b.D(op(t))

1. If \( op = \text{extelm}(a, u) \), then for each node \( n \) labeled by \( a \) in \( t \), do the following.
Example 3 Let us consider the intermediate transformations from $t_3$ to $t_4$ and from $t_4$ to $t_5$ in Fig. 5.

(t₃ ⇒ t₄) Let $D_3 = (d_3, \text{staff})$ be the DTD in Fig. 4(d). Then $d_3(\text{staff}) = \langle \text{name}, \langle \text{street}, \text{zip} \rangle, \text{email} \rangle$. Since $\text{op}_3 = \text{agg,elm}(\text{staff, address, 2})$, we have $d_3(\text{staff}) = \langle \text{name, address, email} \rangle$ and $d_3(\text{address}) = \langle \text{street, zip} \rangle$, and $t_3$ is transformed by step 2 of TRANS1B. Consider the node $n_1$ in $t_3$. Since $d_3(\text{staff}) = \langle \text{name}, \langle \text{street1, zip2} \rangle, \text{email1} \rangle$, the subscripted word $w_1$ of $l(n_2)l(n_3)l(n_5) = \text{name street1 zip2 email1}$ is “name street1 zip2 email1”. Therefore, a new node, say $n_6$, labeled by “address” is inserted as the parent of the second and third children $n_8, n_9$ of $n_1$.

(t₄ ⇒ t₅) Let $D_4 = (d_4, \text{staff})$ be the DTD in Fig. 4(c). Then $d_4(\text{staff}) = \langle \text{name, address, email} \rangle$. Since $\text{op}_4 = \text{ext,elm}(\text{staff, 1})$, $t_4$ is transformed by step 1 of TRANS1B. Consider the node $n_1$ in $t_4$. Since $d_4(\text{staff}) = \langle \text{name1, address2, email3} \rangle$, the subscripted word $w_1$ of $l(n_2)l(n_3)l(n_5) = \text{name address email}$ such that $w_1 = L(d_4(\text{staff}'))$, thereby $\text{match}(w_1, \text{name1}) = \{(1, 1)\}$. Thus the first child $n_2$ of $n_1$ is deleted.

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

TRANS2D,op(l)

(a) Let $n_1, \ldots, n_m$ be the children of $n$ in $t$. Find a subscripted word $w_1$ such that $w_1 = L(d_1(\text{a}))$ and that $w_1[j] = l(n_1) \cdots l(n_m)$.

(b) By definition, $\text{sub}(d_1(\text{a}), u)$ is a single subscripted label, say $b$. For each $(j, i) \in \text{match}(w_1, b_h)$, delete the $j$th child $n_j$ of $n$ from $t$.

2. If $\text{op} = \text{agg,elm}(a, b, u)$, 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 subscripted word $w_1$ such that $w_1 = L(d_1(\text{a}))$ and that $w_1[j] = l(n_1) \cdots l(n_m)$.

(b) For each $(j, k) \in \text{match}(w_1, \text{sub}(d_1(\text{a}), u'))$, insert a new node labeled by $b$ as the parent of $n_1, \ldots, n_k$ into $t$.

3. Return $t$ transformed above.

Figure 5: An example of transformation according to the edit script in Fig. 4.
The transformation algorithm inferred from $D$ and $s$, denoted $\text{TRANSFORM}_{D,s}(t)$, 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 $D_{i-1} = op_{i-1}(...(op_1(D))...) \quad \text{and} \quad s_{i-1} = op_{i-1}...op_1$. Return $\text{TRANSFORM}_{D_{i-1},op_i}(\text{TRANSFORM}_{D,s_{i-1}}(t))$.

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

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

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

Let $D = (d_1,a_0)$ be a DTD and $s = op_1...op_n$ be an edit script to $D$. Finally, let us consider the running time of $\text{TRANSFORM}_{D,s}(t)$. If $op_i = \text{ins}_elm(a,b,u)$ and for some prefix $v$ of $u$ $l(d_i(a),v) = \{s\}$, then $op_i$ is starred. If for some $i,j$ with $i \leq j$, $op_i = \text{ins}_elm(a,b,u)$, $op_p = \text{ins}_elm(c,d,v)$, and $c$ occurs in $d_{j-1}(b)$, then $op_j$ is nested, where $(d_{j-1},a_0) = op_{j-1}(...op_1(D))$. $\text{TRANSFORM}_{D,s}(t)$ runs in polynomial time with some exceptions.

**Theorem 2** Let $D$ be a DTD, $s = op_1...op_n$ be an edit script to $D$, and $t$ be a tree valid against $D$. If the following conditions hold, then $\text{TRANSFORM}_{D,s}(t)$ runs in $O(n^2 \cdot |t| \cdot |D|^2)$ time, where $|t|$ is the number of nodes in $t$ and $|D|$ is the size of $D$.

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

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

5 A Sufficient Condition for Unambiguous Transformation

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

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

**U1** In step (1b) of $\text{TRANS1A}$, $\text{match}(w_1,b_0)$ depends on the subscripted supersequence $w_1$ selected in step (1a).

**U2** In step (1b) of $\text{TRANS1A}$, there may be more than one trees valid against $(d_2,b)$. Similarly, in step (2b-ii) of $\text{TRANS1A}$ there may be more than one forests valid against $(d_2,q)$.

**U3** In step (2b) of $\text{TRANS1A}$, $\text{match}(w_1,b_0)$ depends on subscripted word $w_1$ selected in step (2a). A similar argument also applies to steps (1b) and (2b) of $\text{TRANS1B}$.

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

We give definitions related to (U1) to (U3). Consider first (U1). We define a regular expression that admits only subscripted supersequences such that the positions at which $b_0$ should be inserted can unambiguously determined. Let $D = (d_1,a_0)$ be a DTD, $op = \text{ins}_elm(a,b,v)$ be an edit operation to $D$, $op(D) = (d_2,a_0)$, and $b_0$ be the subscribed label in $d_2(a)$ inserted by $op$. We say that $d_1(a)$ is unambiguous w.r.t. the insertion of $b_0$ if for any word $w \in L(r)$, any subscripted supersequences $w_1',w_2'$ of $w$ w.r.t. $b_0$ such that $w_1',w_2' \in L(d_2(a))$.

- $|w_1'| = |w_2'|$, and
- for any $1 \leq k \leq |w_1'|$, $w_1'[k] = b_0$ and $w_2'[k] = b_0$.

**Example 4** Let $d_1(a) = *((a,b),(b,c))$, $op = \text{ins}_elm(a,a,131)$, and $ab \in L(d_1(a))$. Then $d_2(a)' = *((a_1b_1,b_2),(a_1b_1,b_2))$. Both $a_1b_1b_2$ and $a_1b_1b_2b_3$ are subscripted supersequences of $ab$ w.r.t. $a_1b_1$ belonging to $L(d_2(a))$. Thus $d_1(a)$ is not unambiguous w.r.t. the insertion of $a_1b_1$.

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

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

Then consider (U3). We define a regular expression such that $\text{match}(w_1,b_0)$ can unambiguously determined. Let $r$ be a regular expression, $u \in \text{occ}(r)$ be an occurrence, and $w \in L(r)$ be a word. We say that $r$ is unambiguous w.r.t. $\text{sub}(r,u)$ and $w$ if for any subscripted words $w_1,w_2$ such that $w_1 = w_2$ and that $w_1,w_2 \in L(r')$, $\text{match}(w_1,\text{sub}(r,u)) = \text{match}(w_2,\text{sub}(r,u'))$. We say that $r$ is unambiguous w.r.t. $\text{sub}(r,u)$ if $r$ is unambiguous w.r.t. $\text{sub}(r,u)$ and $w$ for any $w \in L(r)$.

**Example 5** Let $d_1(a) = *((b),(b,c))$. Then $d_1(a)' = *((b_1),(b_1,c_1))$. For a word $be \in L(d_1(a))$, we have two subscripted words $w_1 = b_1c_1212$ and $w_2 = b_1c_1212$ such that $w_1' = w_2' = bc$ and that $w_1,w_2 \in L(d_1(a))$. Since $\text{match}(w_1,+(b_1,c_1)) = \{(2,2) \}$ and $\text{match}(w_2,+(b_2,1,c_2)) = \{(1,2) \}$, $d_1(a)$ is not unambiguous w.r.t. $\text{sub}(d_1(a),2) = *((+b,c))$.

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

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

- The following lemma shows a necessary and sufficient condition under which the transformation algorithm inferred from a DTD and an edit operation is unambiguous.
6.2 Checking the Unambiguity of a Regular Expression w.r.t. a Subexpression

Let \( r \) be a regular expression. In order to determine whether \( r \) is unambiguous w.r.t. a subexpression of \( r \), we use the Glushkov automaton of \( r \) (Book et al. 1971, Brüggemann-Klein & Wood 1998). First, the initial set \( I_r \) and the final set \( F_r \) are defined as follows.

1. If \( r = \varepsilon \), then \( I_r = F_r = \{ E \} \), where \( E \) is a new label.
2. If \( r = a \) for some \( a \in \Sigma \), then \( I_r = F_r = \{ a \} \), where \( a \) is the subscripted label such that \( r' = a_i \).
3. If \( r = + (r_1, \ldots, r_n) \), then \( I_r = I_{r_1} \cup \cdots \cup I_{r_n} \) and \( F_r = F_{r_1} \cup \cdots \cup F_{r_n} \).
4. If \( r = (r_1, \ldots, r_n) \), then
   \[
   I_r = (I_{r_1} \setminus \{ E \}) \cup \cdots \cup (I_{r_n} \setminus \{ E \}) \cup I_r,
   \]
   \[
   F_r = F_{r_1} \cup (F_{r_1} \setminus \{ E \}) \cup \cdots \cup (F_{r_n} \setminus \{ E \}),
   \]
   where
   \[
   i = \begin{cases} 
   n & \text{if } E \in I_{r_k} \text{ for every } 1 \leq k \leq n, \\
   \min \{ k \mid E \notin F_{r_k}, 1 \leq k \leq n \} & \text{otherwise}
   \end{cases}
   \]
   \[
   j = \begin{cases} 
   1 & \text{if } E \notin F_{r_k} \text{ for every } 1 \leq k \leq n, \\
   \max \{ k \mid E \notin F_{r_k}, 1 \leq k \leq n \} & \text{otherwise}
   \end{cases}
   \]
5. If \( r = * (r_1) \), then \( I_r = I_{r_1} \cup \{ E \} \) and \( F_r = F_{r_1} \cup \{ E \} \).

Let \( a \) be a subscripted label occurring in \( r' \). The set of successors of \( a \) in \( r' \), denoted \( \text{Succ}(a, r') \), is defined as follows.

1. If \( r' = a_i \), then \( \text{Succ}(a, r') = \emptyset \).
2. If \( r' = + (r'_1, \ldots, r'_n) \) and \( a \) occurs in \( r'_k \) (1 \leq k \leq n), then \( \text{Succ}(a, r') = \text{Succ}(a, r'_k) \).
3. If \( r' = (r'_1, \ldots, r'_n) \) and \( a \) occurs in \( r'_k \) (1 \leq k \leq n), then
   \[
   \text{Succ}(a, r') = \begin{cases} 
   \text{Succ}(a, r'_k) & \text{if } k = n \text{ or } a \notin F_{r_k}, \\
   \text{Succ}(a, r'_k) \cup (I_{r_k+1} \setminus \{ E \}) & \text{if } k < n \text{ and } a \in F_{r_k},
   \end{cases}
   \]
   where
   \[
   j = \begin{cases} 
   n & \text{if } E \in I_r \text{ for every } k + 1 \leq i \leq n, \\
   \min \{ i \mid E \notin I_r, k + 1 \leq i \leq n \} & \text{otherwise}
   \end{cases}
   \]
4. If \( r' = * (r'_1) \), then
   \[
   \text{Succ}(a, r') = \begin{cases} 
   \text{Succ}(a, r'_1) & \text{if } a \notin F_{r_1}, \\
   \text{Succ}(a, r'_1) \cup (I_{r_1} \setminus \{ E \}) & \text{otherwise},
   \end{cases}
   \]
The Glushkov automaton of \( r \) is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where \( Q \) is the set of states, \( \delta \) is the transition function, \( q_0 \) is the initial state, and \( F \) is the set of final states defined as follows.

- \( Q = sgm(r') \cup \{q_t\} \),
- \( \delta(q_t, a) = \{a \mid a \in I_r, a_0^2 = a\} \) for every \( a \in \Sigma \), and \( \delta(a_j, a) = \{a_k \mid a_k \in \text{Succ}(a_j, r'), a_k^1 = a\} \),
- \( F = \{F_r \cup \{q_t\} \mid \epsilon \in L(r) \), otherwise.

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

We test the unambiguity by using a graph called testing graph\(^2\) of a Glushkov automaton. Let \( G_r = (Q, \Sigma, \delta, q_0, F) \) be the Glushkov automaton of \( r \). A pair \( (ai, aj) \) of states in \( Q \) is compatible if (i) \( ai = aj = q_t \), or (ii) there is a compatible pair \( (ak, al) \) such that, \( ai \in \delta(ak, a), aj \in \delta(al, a), \) and that \( a_k^2 = a_l^2 = a \). Then the testing graph of \( G_r \) is a graph \( T(G_r) = (N, E) \), where

\[
N = \{(ai, aj) \mid (ai, aj) \text{ is a compatible pair of } Q\}, \\
E = \{(ak, al) \rightarrow (ai, aj) \mid ai \in \delta(ak, a), aj \in \delta(al, a)\}.
\]

The following lemma holds by definition.

**Lemma 5** Let \( r \) be a regular expression and \( G_r = (Q, \Sigma, \delta, q_0, F) \) be the Glushkov automaton of \( r \). There are subscripted words \( w_1, w_2 \in L(r') \) such that \( w_1^2 = w_2^2 \) if there is a path \( (q_0, q_1) \rightarrow (ai, aj) \rightarrow \cdots \rightarrow (ak, al) \rightarrow \cdots \rightarrow (ai, aj) \) in \( T(G_r) \) such that \( w_1[k] = a_k \) and \( w_2[k] = a_l \) for any \( 1 \leq k \leq l \), \( ai, aj \in F \), and that \( |w_1| = |w_2| = l \).

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

**Theorem 4** Let \( r \) be a regular expression, \( u \in \text{occ}(r) \) be an occurrence, \( q = \text{sub}(r, u) \) be a subexpression of \( r \), and \( G_q \) be the Glushkov automaton of \( r \). Then \( r \) is unambiguous w.r.t. \( q \) if the following two conditions hold.

1. For any node \( (ai, aj) \) in \( T(G_q) \) from which some accepting node is reachable, either \( ai, aj \in \text{sgm}(q') \) or \( ai, aj \in \text{sgm}(r') \setminus \text{sgm}(q') \).
2. For any edge \( (ai, aj) \rightarrow (ak, al) \) in \( T(G_q) \) from which some accepting node is reachable, if \( ai, aj, ak, al \in \text{sgm}(q') \), then either
   - (a) \( ak \notin \text{Succ}(ai, q') \) and \( al \notin \text{Succ}(aj, q') \), or
   - (b) \( ak \in \text{Succ}(ai, q') \) and \( al \in \text{Succ}(aj, q') \).

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

If part: Assume that \( r \) is not unambiguous w.r.t. \( q \). Then there are words \( w_1, w_2 \in L(r') \) such that \( w_1^2 = w_2^2 \) and for some \( i, j \) (1 \( \leq i < j \leq |w_1| \)) \( w_1[i, j] \) maximally matches \( q' \) but \( w_2[i, j] \) does not maximally match \( q' \). We have two cases to be considered: (i) \( w_2[i, j] \in L(q') \) but it is not maximal and (ii) \( w_2[i, j] \notin L(q') \). Consider the case of (i) (the case of (ii) can be shown similarly). Assume that \( w_2[i, j] \notin L(q') \) but it is not maximal. Then there are indexes \( i', j' \) such that \( w_2[i', j'] \in L(q') \) and that \( \{i', \ldots, j'\} \subset \{i', \ldots, j'\} \). Suppose \( i' < i \) (the case of \( j < j' \) can be shown similarly). Since \( \forall i, j \in L(q') \), either \( w_2[i, j] \notin L(q') \) can be tested in \( O(|r|^3) \) time.

**Lemma 6** For a regular expression \( r \) and a subexpression \( q \) of \( r \), whether \( r \) is unambiguous w.r.t. \( q \) can be tested in \( O(|r|^3) \) time.

**Proof:** The Glushkov automaton \( G_r \) of \( r \) can be constructed in \( O(|r|^2) \) time (Brüggenmann-Klein 1993), and the testing graph \( T(G_r) \) can be constructed in \( O(|r|^3) \) time. Moreover, the condition in Theorem 4 can be checked in linear time w.r.t. \( T(G_r) \).

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

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

To check this we slightly modify the testing graph of a Glushkov automaton. Let \( r \) be a regular expression and \( G_r = (Q, \Sigma, \delta, q_0, F) \) be the Glushkov automaton of \( r \). We first define a contracted transition function \( \delta' \) w.r.t. \( q_0 \), which is obtained by contracting each pair of transitions from \( ai \) to \( bh \), and from \( bj \) to \( aj \) into one transition from \( ai \) to \( aj \). Formally, \( \delta' \) is defined so that for any \( ai, aj \in Q, aj \in \delta'(ai, a) \) iff

- \( ai \in \delta(ai, a), ai \neq bh, \) and \( aj \neq bh, \) or
- \( bh \in \delta(ai, b) \) and \( aj \in \delta(bh, a) \),

where \( a = a_i^2 = a_j^2 \) and \( b = b_i^2 \). A pair \( (ai, aj) \) of states is \( c \)-compatible if (i) \( ai = aj = q_i \), or (ii) there is a \( c \)-compatible pair \( (ai, aj) \) such that \( ai \in \delta'(ai, a), aj \in \delta'(aj, a), \) and that \( a_i^2 = a_j^2 = a \). Then the contracted testing graph of \( G_r \) w.r.t. \( bh \), denoted \( T(G_r, bh) \), is a graph \( (N, E) \), where

\[
N = \{(ai, aj) \mid (ai, aj) \text{ is a compatible pair of } Q\}, \\
E = \{(ai, aj) \rightarrow (ai, aj) \mid ai \in \delta'(ai, a), aj \in \delta'(ai, a)\}.
\]

For an edge \( (ai, aj) \rightarrow (ai, aj) \in E \), if (i) \( ai \in \delta(ai, a) \) but \( bh \in \delta(bh, a) \), or (ii) \( aj \in \delta(ai, a) \) but \( bh \in \delta(bh, a) \) and \( ai \in \delta(bh, a) \), then we say that \( (ai, aj) \rightarrow (ai, aj) \) is odd. If a node \( (ai, aj) \in N \) satisfies one of the following conditions, then \( (ai, aj) \) is called accepting.

1. \( ai \in F \) and \( aj \in F \).
2. \( bh \in \delta(ai, b) \), \( bh \in \delta(ai, b) \), and \( bh \in F \).
3. \( bh \in F \) and (i) \( ai \in F \) and \( bh \in \delta(bj, b) \) or (ii) \( bh \in \delta(ai, b) \) and \( aj \in F \).

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

\footnote{We use a modified version of the testing graph, originally defined in (Even 1965).}
Theorem 5 Let $D = (d_1, a_0)$ be a DTD, $op = \text{ins}_\text{elm}(a, b, v)$ be an edit operation to $D$ such that $l(d_1(a), v) = \cdot$, $b$ be the subscripted label inserted by $op$, and $G_{d_2(a)}$ be the Glushkov automaton of $d_2(a)$, where $op(D) = (d_2, a_0)$. Then $d_1(a)$ is unambiguous w.r.t. the insertion of $b$ if the following three conditions hold.

1. For any odd edge $e$ in $T_{b_0}(G_{d_2(a)})$, no accepting node is reachable from $e$.
2. $T_{b_0}(G_{d_2(a)})$ contains no oddly accepting node.
3. $b_0 \notin \delta(b_0, b)$, where $\delta$ is the transition function of $G_{d_2(a)}$.

Proof (sketch): If $l(d_1(a), v) = \cdot$, then there is a subscribed supersequence $w_1$ of $w$ w.r.t. $b_0$ such that $w_1 \in L(d_2(a))$. Thus the theorem can be shown as follows.

Only if part: It is easy to show that if one of Conditions (1) to (3) does not hold, then $d_1(a)$ is not unambiguous w.r.t. the insertion of $b_0$.

Part: Assume that $d_1(a)$ is not unambiguous w.r.t. the insertion of $b_0$. Then for some word $w \in L(d_1(a))$, there are subscribed subsequences $w_1, w_2$ of $w$ w.r.t. $b_0$ such that $w_1, w_2 \in L(d_2(a))$ and that for some $k$ ($1 \leq k \leq |w_1|$) $w_1[i] = w_2[i]$ for every $1 \leq i < k$, but $w_1[k] = b$ and either $w_2[k] \neq b_0$ or $w_2[k] < k$.

Consider the case where $w_2[k] \neq b_0$ with $k > 1$ (the other cases can be shown similarly). If $w_1[k - 1] = w_2[k - 1] = b_0$, then $w_1[k - 1] = w_2[k] = b_0$, thus Condition (3) does not hold. Suppose that $w_1[k - 1] = w_2[k - 1] \neq b_0$. Since $w_2[k] \neq b_0$, there is an index $k' > k$ such that $w_1[k']$ matches $w_2[k]$. If $k' = k + 1$, then Condition (1) does not hold (since $w_1[k - 1], w_2[k - 1]$ matches $w_1[k'], w_2[k']$ is an odd edge). If $k' > k + 1$, then $w_1[k] = w_2[k] = b_0$, thereby Condition (3) does not hold.

Lemma 7 For a DTD $(d_1, a_0)$ and an edit operation $op = \text{ins}_\text{elm}(a, b, v)$ such that $l(d_1(a), v) = \cdot$, $d_1(a)$ is unambiguous w.r.t. the insertion of $b_0$ can be determined in $O(|d_1(a)|^2)$ time, where $b_0$ is the subscripted label inserted by $op$.

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

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

7 Conclusion

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

We would like to investigate whether real DTDs tend to admit unambiguous transformation. The unambiguity of regular expression w.r.t. subexpression is a weaker condition than one-unambiguity of regular expression, and Ref. (Choi 2002) states that only four out of 60 DTDs contain non-one-unambiguous regular expressions. This might suggest that real DTDs tend to admit unambiguous transformation.

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

Acknowledgement

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

References


