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

Nobutaka SUZUKI†, Member

SUMMARY

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

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

1. Introduction

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

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

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

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

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

- In general, the problem of inferring \( K \) optimum transformations of \( t \) from \( s \) is intractable due to combinatorial explosion. In fact, we show that the problem is NP-hard even if \( K = 1 \).
- If \( s \) is restricted to be of length one, i.e., \( s \) consists only of one update operation, the problem can be solved relatively efficiently. In fact, we construct an algorithm...
for solving this problem, which runs in time polynomial of \(|D|, |t|, \text{ and } K\).

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

Related Work

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

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

2. Definitions

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

Let \(\Sigma\) be a set of labels. A regular expression over \(\Sigma\) is recursively defined as follows.

- \(e\) and \(a\) are regular expressions, where \(a \in \Sigma\).
- If \(r_1, \ldots, r_n\) are regular expressions, then \(r_1 \cdot \cdots \cdot r_n\) are regular expressions \((n \geq 1)\).
- If \(r\) is a regular expression, then \(r^*, r^?\), and \(r^\dagger\) are regular expressions.

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

In order to define update operations to a DTD, we sometimes represent a regular expression as a term in prefix notation. For example, we may write \(\langle a, ^\dagger (\langle b, c \rangle) \rangle\) instead of \(a(b|c)^\dagger\), where ‘\(\cdot\)’ denotes a concatenation operator. Let \(r\) be a regular expression in prefix notation. The set of positions of \(r\), denoted \(pos(r)\), is defined as follows.

- If \(r = e\) 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 \(op \in \{\cdot, ^\dagger, ^\dagger, ^?\}\), then \(pos(r) = \{\lambda\} \cup \{u \mid u = iv, 1 \leq i \leq n, v \in pos(r_i)\}\). Note that \(n = 1\) if \(op \in \{^?, ^\dagger\}\).

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

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

- If \(r = e\) 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 \(op \in \{\cdot, ^\dagger, ^\dagger, ^?\}\), and
  - if \(u = \lambda\), then \(l(r, u) = op\) and \(sub(r, u) = r\),
  - if \(u = jv\) for some \(1 \leq j \leq n\) and some \(v \in pos(r_j)\), then \(l(r, u) = l(r_j, v)\) and \(sub(r, u) = sub(r_j, v)\).

For example, in Fig. 2 \(l(r, 1) = \langle\rangle\), \(l(r, 11) = a\), and \(sub(r, 21) = \langle c, d \rangle\).

Let \(w\) be a word over \(\Sigma\). By \(|w|\) we mean the length of \(w\), and by \(w[i]\) we mean the \(i\)th label of \(w\). We define that \(w[i, j] = w[i]w[i+1]\cdots w[j]\) \((1 \leq i \leq j \leq |w|)\). For example, if \(w = \text{kasuga}\), then \(w[3, 5] = \text{sug}\).

Let \(r\) be a regular expression. By \(r^\#\) we mean the superscripted regular expression resulting from \(r\) by superscripting each label in \(r\) by its corresponding position. By sym\((r^\#)\) we mean the set of superscripted labels occurring in \(r^\#\). For example, if \(r = \langle ab(c)db \rangle^\dagger\), then \(r^\# = \langle a1|b12|c15|(d211)b212\rangle^\dagger\) and sym\((r^\#)\) =
Let \( d' \) be a superscripted label of \( a \). Then by \((a')^d\) we mean the label resulting from \( a' \) by dropping the superscript of \( a' \), that is, \((a')^d = a \). Let \( w' \) be a superscripted word (i.e., a sequence of superscripted labels). We define that \(((w')^d)^{\epsilon} = w'[1]^d \cdots w'[|w'|]^d\). For any regular expression \( r \), it holds that \( L(r)^{\epsilon} = \{[(w')^d] | w' \in L(r)^d\}\).

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

\[
\begin{align*}
    d(b) &= \text{chapter}^+ \\
    d(c) &= \text{section}^+ \text{bib} \\
    d(s) &= \epsilon \\
    d(bib) &= \epsilon
\end{align*}
\]

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

3. Update Operations to DTD

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

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

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

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

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

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

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

- \textit{ins_elm}(a, b, vi): Inserts a new label \( b \) at position \( vi \) in \( d(a) \), where \( vi \in \text{post}(d(a)), i \) is a positive integer, and \( b \in \Sigma \) (Fig. 4 (b,c)). This is applicable to \( D \) only if \( d(b) \) is defined, \( l(d(a), vi) \in \{\cdot,\} \), and the operator at \( v \) has at least \( i - 1 \) children.
- \textit{del_elm}(a, vi): Deletes the label (possibly \( \epsilon \)) at \( vi \) in \( d(a) \). More formally, we have two cases according to the operator at \( v \).

  - The case where \( l(d(a), vi) = \cdot \cdot \cdot \cdot \): The label at \( vi \) in \( d(a) \) is deleted from \( d(a) \) (Fig. 4 (a,b)). This is applicable to \( D \) only if the operator at \( v \) has more than one child.
  - The case where \( l(d(a), vi) = \cdot \cdot \cdot \cdot \): If \( l(d(a), vi) = l(d(a), vk) \) for some \( k \neq i \), then the label at \( vi \) in \( d(a) \) is deleted from \( d(a) \) (deleting one of duplicated labels). Otherwise, the label at \( vi \) in \( d(a) \) is replaced by \( \epsilon \).
tion/aggregation of an element.

- ext Elm\((a, u)\): Extracts the label at \(u\) in \(d(a)\). Formally, this operation replaces the label at \(u\) in \(d(a)\) by regular expression \(d(l(d(a), u))\) (Fig. 4(e,f)). This is applicable to \(D\) only if \(l(d(a), u) \in \Sigma, l(d(a), u) \neq a\), and \(d(l(d(a), u))\) is defined.

- agg\_elm\((a, b, u)\): Aggregates the subexpression at \(u\) in \(d(a)\) into single label \(b\). Formally, this operation (i) sets \(d(b) = \text{sub}(d(a), u)\) and (ii) replaces the subexpression at \(u\) in \(d(a)\) by \(b\) (Fig. 4(d,e)). This is applicable to \(D\) only if \(d(b)\) is undefined.

The following three operations relate to handling an operator \((\_, \_, ^{+}, \_?)\) in a content model.

- ins\_opr\((a, opr, u, v)\): Inserts a new operator \(opr\) as the parent of the siblings at \(u, \ldots, v\) in \(d(a)\), where \(opr \in \{\_, ^{+}, \_?\}\) (Fig. 4(c,d)). This is applicable to \(D\) only if (i) \(u = v\) (\(opr\) has only one child) or (ii) \(opr \in \{\_, \_?\}\) and \(opr = l(d(a), w)\), where \(u = wi\) and \(v = wj\) for some \(i < j\) (nesting the operator at \(w\) by \(opr\)).

- del\_opr\((a, u)\): Deletes an operator at \(u\) in \(d(a)\) (Fig. 4(f,g)). This is applicable to \(D\) only if (i) the operator at \(u\) has only one child or (ii) \(l(d(a), u) = l(d(a), v)\), where \(u = vi\) for some \(i\) (unnesting the operators at \(u\) and \(v\)).

- change\_opr\((a, opr, u)\): Replaces the operator at \(u\) in \(d(a)\) by \(opr\), where \(l(d(a), u), opr \in \{\_, ^{+}, \_?\}\).

Let \(op\) be an update operation to a DTD \(D\). By \(op(D)\) we mean the DTD obtained by applying \(op\) to \(D\). Let \(s = op\_1op\_2\cdots op\_n\) be a sequence of update operations. We say that \(s\) is an update script to \(D\) if \(op\_i\) is applicable to \(op\_{i-1}(op\_{i-2}(\cdots op\_1(D)\cdots))\) for every \(1 \leq i \leq n\). By \(|s|\) we mean the length of \(s\), that is, \(|s| = n\). We say that a DTD \(D_2\) includes a DTD \(D_1\) if for any tree \(t\), \(t\) is valid against \(D_2\) whenever \(t\) is valid against \(D_1\). We have the following lemma.

**Lemma 1:** Let \(D = (d, sl)\) be a DTD and \(op\) be an update operation applicable to \(D\). Then \(op(D)\) includes \(D\) if

\[
\begin{align*}
& op = \text{ins\_elm}(a, b, vi) \text{ and } l(d(a), v) = \{\_\}, \\
& op = \text{ins\_opr}(a, opr, u, v), \\
& op = \text{del\_opr}(a, u) \text{ and } l(d(a), u) \in \{\_, \_?\}, \text{ or} \\
& op = \text{change\_opr}(a, opr, u), \text{ and (i) } opr = \{\_, ^{+}, \_?\} \text{ or (ii) } l(d(a), u) = opr.
\end{align*}
\]

4. **Transformation Algorithm**

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

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

**TransOp**(\(D, t, op\))

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

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

1. If \(t\) is valid against \(op(D)\), return \(t\).
2. Else
   \[
   \text{if } op = \text{ins\_elm}(a, b, vi), \text{ return Trans1}(D, t, op),
   \]
   \[
   \text{else if } op = \text{del\_elm}(a, vi), \text{ return Trans2}(D, t, op),
   \]
   \[
   \text{else if } op = \text{agg\_elm}(a, b, vi), \text{ return Trans3}(D, t, op),
   \]
   \[
   \text{else if } op = \text{del\_opr}(a, u), \text{ return Trans4}(D, t, op),
   \]
   \[
   \text{else if } op = \text{agg\_opr}(a, opr, u), \text{ return Trans5}(D, t, op),
   \]
   \[
   \text{else if } op = \text{change\_opr}(a, opr, u), \text{ return Trans6}(D, t, op).
   \]
if \( op = \text{del}_\text{elm}(a, v) \), return \( \text{Trans2}(D, t, op) \).
if \( op = \text{ext}_\text{elm}(a, u) \), return \( \text{Trans3}(D, t, op) \).
if \( op = \text{agg}_\text{elm}(a, b, u) \), return \( \text{Trans4}(D, t, op) \).
if \( op = \text{ins}_\text{op}(a, opr, u, v) \), return \( t \).
if \( op = \text{del}_\text{opr}(a, u) \), return \( \text{Trans5}(D, t, op) \).
if \( op = \text{change}_\text{op}(a, opr, u) \), return \( \text{Trans6}(D, t, op) \).

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

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

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

Let us first show \text{Trans1}. \( \text{Trans1}(D, t, op) \) transforms \( t \) according to \( op \). In this case, \( op = \text{ins}_\text{elm}(a, b, v) \), and by Lemma 1 \( l(d_1(a), v) = \cdots \). Thus, it suffices to insert new \( b \) elements at appropriate positions in \( t \). We need a definition. Let \( w \) be a word and \( b^\# \) be a superscripted label. We say that a superscripted word \( w' \) is a superscripted supersequence of \( w \) w.r.t. \( b^\# \) if removing every \( b^\# \) from \( w' \) yields a word \( w'' \) such that \( (w'')^\# = w \). In the following, we denote \( D = (d_1, sl) \) and \( \text{op}(D) = (d_2, sl) \), and assume that each transformation is done in bottom-up manner.

\text{Trans1}(D, t, op)

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

For example, the transformation from \( t_1 \) to \( t_2 \) in Fig. 4 is done by \text{Trans1}.

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

Let us next show \text{Trans2}. In this case, \( op = \text{del}_\text{elm}(a, v) \). Thus, it suffices to delete the elements in \( t \) that match the label in \( d_1(a) \) deleted by \( op \).

\text{Trans2}(D, t, op)

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

The transformation from \( t_0 \) to \( t_1 \) in Fig. 4 is an example of \text{Trans2}.

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

\text{Trans3}(D, t, op)

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

The transformation from \( t_4 \) to \( t_5 \) in Fig. 4 is an example of \text{Trans3}.

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

\text{Trans4}(D, t, op)

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

The transformation from \( t_3 \) to \( t_4 \) in Fig. 4 is an example of \text{Trans4}.

Let us show \text{Trans5}. We have \( op = \text{del}_\text{opr}(a, u) \) and

\[\text{We assume that the text values of such a new element are empty since they can hardly be estimated.}\]
SUZUKI: AN ALGORITHM FOR INFERRING K OPTIMUM TRANSFORMATIONS OF XML DOCUMENT FROM UPDATE SCRIPT TO DTD

\( l(d_1(a), u) \in \{?, +, \} \). Thus we have three cases to be considered: (i) \( l(d_1(a), u) = ? \), (ii) \( l(d_1(a), u) = + \), and (iii) \( l(d_1(a), u) = +? \). Let \( sub(d_1(a), u) = q \). Consider first the case of (i). In this case, \( sub(d_1(a), u) = q \) and this is changed to \( q \) by \( op \). Thus for each sequence of nodes matching \( q \), if the sequence is \( \epsilon \), we have to insert a sequence of elements matching \( q \). This can be done similarly to the case of (iv) of Trans6 shown later. Let us next consider the case of (ii). Since \( q \) is changed to \( q \) by \( op \), for each sequence \( seq \) matching \( q \), (a) if \( seq = \epsilon \), we have to insert a sequence of elements matching \( q \) and (b) otherwise, \( seq \) must be “shrunk” so that \( seq \) matches \( q \) instead of \( q^\prime \). These can be handled by a combination of similar ideas shown later; (c) (a) can be handled similarly to the case of (iv) of Trans6 and (b) can be done similarly to the case of (iii) (since \( q^\prime = q^\prime \)).

In the following, we consider the case of (iii). In this case, \( sub(d_1(a), u) = q^\prime \). Since \( q^\prime \) is changed by \( q \) by \( op \), we have to “shrunk” each sequence of nodes in \( t \) that matches \( q^\prime \) so that the resulting sequence matches \( q \) instead of \( q^\prime \).

The extraction \( d_1^t(a) \) of \( d_1(a) \) is obtained from \( d_1(a) \) by replacing \( q^\prime \) with \( q^\prime \). Clearly, \( d_1^t(a) \) is equivalent to \( d_1(a) \). Let \( w^\prime \) be a superscripted word such that \( (w^\prime)^\# \in L(d_1^t(a)) \). A shrink \( w^\prime \) of \( w^\prime \) w.r.t. \( (q^\prime)^\# \) is obtained by deleting every sequence matching \( sub(d_1(a), u) \) or \( sub(d_1(a), u) \).

Trans5(D, t, op)

1. 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 \( t \). If \( l(n_1) \cdots l(n_m) \notin L(d_2(a)) \), do the following.
      i. Find a superscripted word \( w^\prime \) such that \( w^\prime \in L(d_1(a)) \) and that \( (w^\prime)^\# = l(n_1) \cdots l(n_m) \).
      ii. Find a shrink \( w^\prime \) of \( w^\prime \) w.r.t. \( (q^\prime)^\# \), where \( q^\prime = sub(d_1(a), u) \). For each \( 1 \leq j \leq |w^\prime| \) such that \( w^\prime[j] \) disappears in \( w^\prime \), delete the subtree rooted at \( n_j \) from \( t \).

2. Return \( t \).

Finally, let us show Trans6. We have \( op = change_opr(a, opr, u) \), and by Lemma 1 we have four cases to be considered: (i) \( l(d_1(a), u) = \) “?” and \( opr = \) “?”?, (ii) \( l(d_1(a), u) = \) “?” and \( opr = \) “?”?, (iii) \( l(d_1(a), u) = \) “?” and \( opr = \) “?”?, and (iv) \( l(d_1(a), u) = \) “?” and \( opr = \) “?”?. Let \( sub(d_1(a), u) = q \). In the cases of (i) and (ii), for each sequence \( seq \) of nodes matching \( q^\prime \) or \( q^\prime \), \( seq \) must be “shrunk” so that \( seq \) matches \( q \) instead of \( q^\prime \) or \( q^\prime \). This can be treated similarly to the case of (iii) of Trans5. The case of (iii) can be handled similarly to the case of (iv). In the following, we consider the case of (iv). Then \( sub(d_1(a), u) = q^\prime \) and \( q^\prime \) is changed to \( q^\prime \) by \( op \). Thus, for each position in \( t \) matching \( q^\prime \), if the matched sequence is \( \epsilon \), then we have to insert a sequence of elements matching \( q \). Let \( w^\prime \in L(d_1(a)^\#) \) be a superscripted word. For an index \( 0 \leq i \leq |w^\prime| \), \( i \) is a potential gap w.r.t. \( q^\prime \) if neither \( w^\prime[i] \) nor \( w^\prime[i+1] \) is in \( sym(q^\prime) \) (assuming that \( w^\prime[0], w^\prime[|w^\prime|+1] \notin sym(q^\prime) \)). An extension of \( w^\prime \) w.r.t. \( q^\prime \) is a superscripted word obtained by inserting zero or more \( w^\prime[i] \) and \( w^\prime[i+1] \) for every potential gap \( i \) w.r.t. \( q^\prime \), where \( w^\prime_q \) is a word such that \( w^\prime_q \in L(q^\prime) \).

Trans6(D, t, op)

1. 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 \( t \). If \( l(n_1) \cdots l(n_m) \notin L(d_2(a)) \), do the following.
      i. Find a superscripted word \( w^\prime \) such that \( w^\prime \in L(d_1(a)^\#) \) and that \( (w^\prime)^\# = l(n_1) \cdots l(n_m) \).
      ii. Find an extension \( w^\prime \) of \( w^\prime \) w.r.t. \( q^\prime \) such that \( w^\prime \in L(d_2(a)^\#) \). For each superscripted label \( w^\prime[i] \) inserted into \( w^\prime \), create a tree \( t_i \) valid against \( d_2(a) \), \( (w^\prime[i])^\# \) and insert \( t_i \) as the \( i \)th child of \( n \).

2. Return \( t \).

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

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

5. NP-Hardness

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

5.1 Formal Definition of the Problem

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

- \( df(t_1, t_2) \) is defined as the set of root nodes of the subtrees inserted into \( t_1 \) if
  - \( op = ins_elm(a, b, vi) \), or
  - \( op = change_opr(a, opr, u), l(d_1(a), u) = \) “?” and \( opr = \) “?”?

- \( df(t_1, t_2) \) is defined as the set of root nodes of the subtrees deleted from \( t_1 \) if
  - \( op = del_elm(a, vi) \), or
  - \( op = del_opr(a, u) \) and \( l(d_1(a), u) = \) “?”

- \( df(t_1, t_2) \) is defined as the set of nodes deleted from \( t_1 \) if \( op = \) “?”?

- \( df(t_1, t_2) \) is defined as the set of nodes deleted from \( t_1 \) if \( op = ext_elm(a, u) \).

- \( df(t_1, t_2) \) is defined as the set of nodes inserted into \( t_1 \) if \( op = agg_elm(a, b, u) \).

Otherwise, \( df(t_1, t_2) = \emptyset \).
Let $D$ be a DTD, $s = op_1 \cdots op_n$ be an update script to $D$, and $t$ be a tree valid against $D$. A sequence $TS = t_0, t_1, \cdots, t_n$ of trees is called a transformation sequence w.r.t. $(t, D, s)$ if $t_0 = t$ and $t_i \in \text{TransOp}(D_{i-1}, t_{i-1}, op_i)$ for every $1 \leq i \leq n$, where $D_{i-1} = op_{i-1}(\cdots(op_1(D))\cdots)$. The cost of a transformation sequence $TS$, denoted $\gamma(TS)$, is defined as $\gamma(TS) = \sum_{1 \leq i \leq n} \lvert df(t_{i-1}, t_i) \rvert$. For a positive integer $K$, we say that $K$ optimum transformation sequences $TS_1, \cdots, TS_K$ w.r.t. $(t, D, s)$ are $K$ optimum transformation sequences $TS_1, \cdots, TS_K$ w.r.t. $(t, D, s)$ if $\gamma(TS_i) \leq \gamma(TS_{i+1})$ for any $1 \leq i \leq K-1$ and $\gamma(TS_K) \leq \gamma(TS)$ for any transformation sequence $TS$ w.r.t. $(t, D, s)$ such that $TS \notin \{TS_1, \cdots, TS_K\}$. Now our problem is formulated as follows.

Instance: A DTD $D$, a tree $t$ valid against $D$, an update script $s$ to $D$, and a positive integer $K$.

Question: Find $K$ optimum transformation sequences w.r.t. $(t, D, s)$.

5.2 NP-Hardness of the Problem

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

Instance: A DTD $D$, a tree $t$ valid against $D$, an update script $s = op_1 op_2 \cdots op_n$ to $D$, and a positive integer $B$.

Question: Is there a transformation sequence $TS = t_0, t_1, \cdots, t_n$ w.r.t. $(t, D, s)$ such that $\gamma(TS) \leq B$?

We have the following theorem.

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

**Proof:** We use the following SAT problem.

Instance: A set $X = \{x_1, \cdots, x_n\}$ of variables and a collection $C = \{C_1, \cdots, C_m\}$ of clauses over $X$.

Question: Is there a satisfying truth assignment for $C$?

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

- Tree $t = t_0$ is constructed as shown in Fig. 5 (top), where $T_i$ and $F_i$ stand for sequences of labels defined as follows ($1 \leq i \leq n$).
  - Let $C_{i_1}, \cdots, C_{i_k}$ be the clauses in $C$ that contain positive literal $x_i$. Then $T_i = c_{i_1} \cdots c_{i_k}$, where $c_i$ is a label corresponding to clause $C_i$. That is, $T_i$ consists of the clauses that are satisfied by setting $x_i = \text{true}$.
  - Let $C_{i_1'}, \cdots, C_{i_k'}$ be the clauses in $C$ that contain negative literal $\neg x_i$. Then $F_i = c_{i_1'} \cdots c_{i_k'}$. That is, $F_i$ consists of the clauses that are satisfied by setting $x_i = \text{false}$.
- $D = (d, r)$, where $d(r) = a^+$, $d(a) = b^+$, $d(b) = T_1[F_1] \cdots [T_n][F_n]$, and $d(c_i) = \varepsilon$ ($1 \leq i \leq m$).
- $s = s_1 s_2 s_3$, where $s_1 = \text{del}_{\text{op}}(a, \lambda) \text{ext}_{\text{elm}}(a, \lambda) \text{ext}_{\text{elm}}(r, 1)$, $s_2 = \text{ins}_{\text{op}}(r, |, \lambda) \text{ins}_{\text{subexpr}}(r, q, 2)$ $\text{del}_{\text{subexpr}}(r, 1) \text{del}_{\text{op}}(r, \lambda)$, $s_3 = \text{ins}_{\text{elm}}(r, c_1, 2) \text{del}_{\text{elm}}(r, 2)$ $\cdots$ $\text{ins}_{\text{elm}}(r, c_m, 2) \text{del}_{\text{elm}}(r, 2)$, and $q = (c_1 \cdots |c_m|)(c_1 \cdots |c_m|)^*$. $\gamma(TS)$ is greater or equal to the tree edit distance between $t_0$ and $t_0$, assuming that a subtree insertion/deletion can be done by one edit operation.$^1$

In $s_2$, (i) $\text{ins}_{\text{subexpr}}(r, q, 2)$ stands for a “macro” that inserts $q$ into $d(r)$ at position 2 and (ii) $\text{del}_{\text{subexpr}}(r, 1)$ is a macro that deletes the subexpression $d(r)$ at position 1. Thus $s_2$ updates regular expression $(T_1[F_1] \cdots [T_n][F_n])^* \rightarrow (c_1 \cdots |c_m|)(c_1 \cdots |c_m|)^*$.

- $B = 3n$.

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

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

1. First, $d(r) = (c_1 \cdots |c_m|)(c_1 \cdots |c_m|)^*$ is updated to $(c_1 \cdots |c_m|)(c_1 \cdots |c_m|)^* \rightarrow \text{ins}_{\text{elm}}(r, c_1, 2)$.
2. Then $(c_1 \cdots |c_m|)(c_1 \cdots |c_m|)^*$ is updated to $(c_1 \cdots |c_m|)^* \rightarrow \text{del}_{\text{elm}}(r, 2)$. Since $t_{|s_3|}$ is not changed by $s_3$, $t_{|s_3|}$ must have a leaf node labeled by $c_i$ for every $1 \leq i \leq m$. Now consider the following truth assignment $\alpha$ ($1 \leq i \leq n$).

\[ \alpha(x_i) = \begin{cases} \text{true} & \text{if } T_r \text{ is deleted by } \text{del}_{\text{op}}(a, \lambda) \text{ of } s_1, \\ \text{false} & \text{if } T_r \text{ is deleted by } \text{del}_{\text{op}}(a, \lambda) \text{ of } s_1. \end{cases} \]

Since $t_{|s_3|}$ has a leaf node labeled by $c_i$ for every $1 \leq i \leq m$, by the definitions of $T_i$ and $F_i$, it is easy to see that $\alpha$ is a satisfying truth assignment for $C$.

**Only if part:** Assume that there is a satisfying truth assignment for $C$. We show that there is a satisfying truth assignment for $C$ if and only if there is a transformation sequence $TS = t_0, t_1, \cdots, t_{|s_3|}$ w.r.t. $(t, D, s)$ such that $\gamma(TS) \leq B$. Therefore, we can conclude that $\text{del}_{\text{op}}(a, \lambda)$ is NP-hard even if $K = 1$. Thus, the transformation decision problem is NP-hard.
SUZUKI: AN ALGORITHM FOR INFERRING $K$ OPTIMUM TRANSFORMATIONS OF XML DOCUMENT FROM UPDATE SCRIPT TO DTD

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

6. Algorithm for Finding $K$ Optimum Transformation Sequences

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

The main difference between Glushkov automaton and usual NFA is that for any regular expression $r$, there is a one to one correspondence between the superscripted labels in $r^a$ and the states of the Glushkov automaton of $r$ (except the initial state), but a usual NFA does not have this property. For example, let $r = d((c' b) (c b'))$ be a regular expression. Then $r^a = d^2(((c^{211})^c (b^{212}))((c^{221}) (b^{222}))')$. The Glushkov automaton of $r$ is shown in Fig. 6 (c) ($a^g$ is the initial state). Except the initial state $a^g$, each superscripted label in $r^a$ occurs exactly once in the Glushkov automaton, and vice versa. For a DTD $D$ and a tree $t$ valid against $D$, when $D$ is updated, we have to identify the nodes in $t$ that should be deleted and/or the positions in $t$ that new nodes should be inserted into. The above property is useful to obtaining such nodes and positions. For example, let $D = (d, s)$ be a DTD, $n$ be a node with $l(n) = a$ in a tree, $ch(n)$ be the children of $n$, and $G_{d(a)}$ be the Glushkov automaton of $d(a)$. If $del_elm(a, u)$ is applied to $D$, we have to find the nodes in $ch(n)$ that should be deleted according to $del_elm(a, u)$. This can be done by finding the nodes to which $b^n$ is assigned under a matching between $ch(n)$ and $d(a)^a$, where $b^n$ is the state in $G_{d(a)}$ corresponding to the label at $u$ in $d(a)$.

6.1 Glushkov Automaton

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

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

\[
I_r = (I_{r_1} - \{E\}) \cup \cdots \cup (I_{r_n} - \{E\}) \cup I_r,
\]
\[
F_r = F_{r_1} \cup (F_{r_1} - \{E\}) \cup \cdots \cup (F_{r_n} - \{E\}),
\]

where

\[
i = \begin{cases} n & \text{if } E \in I_r \text{ for every } 1 \leq k \leq n, \\
\min\{k \mid E \notin I_r, 1 \leq k \leq n\} & \text{otherwise},
\end{cases}
\]
\[
j = \begin{cases} n & \text{if } E \in F_r \text{ for every } 1 \leq k \leq n, \\
\max\{k \mid E \notin F_r, 1 \leq k \leq n\} & \text{otherwise}.
\end{cases}
\]

- If $r = r_1^* \text{ or } r = r_1\#$, then $I_r = I_{r_1} \cup \{E\}$ and $F_r = F_{r_1} \cup \{E\}$.
- If $r = r_1^* \cdots r_n^\#$ and $a^\epsilon$ occurs in $r_k^\#$ $(1 \leq k \leq n)$, then

\[
\text{Succ}(a^\epsilon, r^\#) = \begin{cases} \text{Succ}(a^\epsilon, r_k^\#) & \text{if } k \text{ is an } \epsilon \text{ or } a^\epsilon \notin F_{r_k}, \\
\text{Succ}(a^\epsilon, r_k^\#) \cup (I_{r_k} - \{E\}) \cup \\
\cdots \cup (I_{r_n} - \{E\}) & \text{if } k < n \text{ and } a^\epsilon \in F_{r_k},
\end{cases}
\]

where

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

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

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

- If $r^\# = (r_1^\#)^\#$, then \( \text{Succ}(a^\epsilon, r^\#) = \text{Succ}(a^\epsilon, r_1^\#) \).

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

- $Q = \text{sym}(r^\#) \cup \{a^\epsilon\}$,
- $\delta(a^\epsilon, a) = \{a^\epsilon \mid a^\epsilon \in I_r, (a^\epsilon)^\# = a\}$ for every $a \in \Sigma$, and $\delta(a^\epsilon, a) = \{a^\epsilon \mid a^\epsilon \in \text{Succ}(a^\epsilon, r^\#), (a^\epsilon)^\# = a\}$,
- $F = \{F_r \cup \{a^\epsilon\} - \{E\} \mid \epsilon \in L(r)\}$, otherwise.

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

6.2 Algorithm

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

Main Algorithm

The algorithm consists of the “main” algorithm and some subroutines. Let us first show the “main” algorithm. Let $D = (d, sl)$ be a DTD, $t$ be a tree valid against $D$, $n$ be a node in $t$, and $op$ be an update operation to $D$. By $t_0$ we mean the subtree of $t$ rooted at $n$, and let $D(n) = (d, l(n))$ be a DTD. We say that $d_f(n), \cdots, d_f(K)(n)$ are $K$ optimum diffs w.r.t. $(t_0, D(n), op)$ if for some $K$ optimum transformation sequences $TS_1, \cdots, TS_K$ w.r.t. $(t_0, D(n), op)$, $d_f(n) = \gamma(TS_i)$ for every $1 \leq i \leq K$.

The following algorithm Main computes $K$ optimum diffs $d_f(n), \cdots, d_f(K)(n)$ w.r.t. $(t_0, D(n), op)$ for each node $n$ in bottom-up manner. For each node $n$ in $t$, the algorithm does the following.

- If $n$ is a leaf and no child needs to be added to $n$ by $op$, then $d_f(n), \cdots, d_f(K)(n)$ are obtained in steps 2 and 3. In step 2, we have $(d, s_l) = op(D)$.
- Otherwise, $d_f(n), \cdots, d_f(K)(n)$ are computed in steps 4 to 21. The subroutines in these steps are shown later.

- In steps 5 to 19, a graph $G(N, E)$ and a weight function $w$ are obtained, where $G(N, E)$ represents the “product” of $d_1(a)$ and the children of $n$, and $w$ assigns a diff to each edge on $G(N, E)$.
- In step 20, $K$ optimum diffs $d_f(n), \cdots, d_f(K)(n)$ are computed by finding $K$ “shortest” paths on $G(N, E)$.

Main($D, t, op, K$)

Input: A DTD $D = (d, sl)$, a tree $t$ valid against $D$, an update operation $op$ to $D$, and a positive integer $K$.

Output: $K$ optimum diffs w.r.t. $(t, D, op)$.

begin
1. for each node $n$ in $t$ (in bottom-up order) do
2. if $n$ is a leaf and $(l(n) \neq a \text{ or } \epsilon \in L(d_2(a)))$ then
3. $d_f(n) \leftarrow 0$ and $d_f(n) \leftarrow n$ for each $2 \leq i \leq K$;
4. else begin
5. if $(l(n) = a$ and $l(n_1) \cdots l(n_m) \notin L(d_2(a)))$ then
6. $d_f(n) \leftarrow \text{ins}(l(a), b, v)$ if $\text{ins}(l(a), b, v)$ then
7. $(G(N, E), w) \leftarrow \text{MaxGraph1}(D, t, n, op, K)$;
8. $d_f(n) \leftarrow \text{del}(l(a), v)$ if $\text{del}(l(a), v)$ then
9. $(G(N, E), w) \leftarrow \text{MaxGraph2}(D, t, n, op, K)$;
10. if $op = \text{ext}(l(a), v)$ then
11. $(G(N, E), w) \leftarrow \text{MaxGraph3}(D, t, n, op, K)$;
12. if $op = \text{agg}(l(a), b, w)$ then
13. $(G(N, E), w) \leftarrow \text{MaxGraph4}(D, t, n, op, K)$;
end
end
Outline of Subroutines

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

Let \( n \) be a node in \( t \) labeled by \( a \), and let us consider finding \( K \) optimum diffs \( d_f(n_1), \cdots, d_f(n) \). Assuming that \( d_f(n_1), \cdots, d_f(n) \) have been obtained for each child \( n_i \) of \( n \), we find \( d_f(n_1), \cdots, d_f(n) \) as follows. Suppose that \( op = \text{del}_\text{elm}(a, v_i) \).

1. We first make a “child list graph” \( CL(N', E') \) of \( n \). Figure 6 (b) is an example assuming that \( K = 2 \). As shown later, each edge \( n_i' \rightarrow n_j' \) is associated with the \( l \)th diff \( d_f(n_l) \) of \( n_i \).

2. We make the Glushkov automaton \( G_{d_i(a)} \) of \( d_i(a) \). For example, Fig. 6 (c) shows the Glushkov automaton of \( d_i(a) = d(c^*b)(c^*b^*) \).

3. We make the “product graph” \( G(N, E) \) of \( G_{d_i(a)} \) and \( CL(N', E') \) as shown in Fig. 6 (d), then associate a “weight” (actually, a diff) to each edge in \( E \). \( G(N, E) \) has the following properties.

   a. Any path in \( G(N, E) \) from the source to a destination represents the sequence of children that matches \( d_i(a)^q \). For example, path

   \[
   (a', n'_0) \rightarrow (d_1, n'_1) \rightarrow (c_221, n'_2) \rightarrow (b_2221, n'_3)
   \]

   in Fig. 6 (d) represents the sequence of children \( n_0, n_2, n_3 \) that matches \( d_i c_221 b_2221 \in L(d_i(a)^q) \), for any \( l_1, l_2, l_3 \in \{1, 2, 3\} \).

   b. Each edge \( e = (a', n'_0) \rightarrow (a', n'_n) \) in \( E \) is associated with the \( l \)th diff \( d_f(n_l) \) of \( n_i \), but we have one exception; if \( a' \) is the superscripted label deleted from \( d_i(a) \) by \( op \), then \( e \) is associated with \( (n_i) \) instead of \( d_f(n_i) \), where \( (n_i) \) represents the diff when the subtree rooted at \( n_i \) is deleted.

4. Find \( K \) “shortest” paths from the source to the destinations. By (a) and (b) above, the diffs on these paths are precisely \( K \) optimum diffs \( d_f(n_1), \cdots, d_f(n) \).

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

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

\[
N' = \{n'_0, \cdots, n'_m\},
\]

\[
E' = \{n'_{i-1} \rightarrow n'_i | 1 \leq i \leq m, 1 \leq l \leq K\},
\]

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

\[
N = \{(a', n'_0), a_1 \in Q, n'_j \in N', (a')^q = l(n'_j)\},
\]

\[
E = \{(a', n'_j-1) \rightarrow (a', n'_j) | d_a' \in \delta(a', (a')^q), n'_j-1 \rightarrow n'_j \in E'\}.
\]

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

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

Input: A DTD \( D = (d_1, sl) \), a tree \( t \) valid against \( D \), a node \( \text{root}(t) \), and an update operation \( op = \text{del}_\text{elm}(a, v_i) \), and a positive integer \( K \).

Output: A graph \( G(N, E) \) and a function \( w \).

begin
1. Construct the child list graph \( CL(N', E') \) of \( n \). \n2. Construct the Glushkov automaton \( G_{d_i(a)} \) of \( d_i(a) \). \n3. Construct the product graph \( G(N, E) \) of \( G_{d_i(a)} \) and \( CL(N', E') \). \n4. for each \( e = (a', n'_0) \rightarrow (a', n'_n) \in E \) let \( \text{w}(e) \leftarrow \{ \text{nil} \} \) if \( a' = b^* \) and \( l = 1 \), \( \{ d_f(n_i) \} \) if \( a' = b^* \) and \( l > 1 \), \( \{ \} \) if \( a' \neq b^* \).

   where \( b^* \) is the superscripted label deleted from \( d_i(a)^q \) by \( op \).
5. \end
6. \return \((G(N, E), w)\);
end

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

KShortestPaths\((H(N_H, E_H))\)

Input: A weighted acyclic graph \( H(N_H, E_H) \).

Output: A set of weights of \( K \) shortest paths.

begin
1. Let \( \Delta_n \) be the multiset of \( K \) \( \infty \)’s for each \( n_i \in N_H \);
2. \( \Delta_n[1] \leftarrow 0 \);
3. Sort the nodes in \( N_H \) w.r.t. ‘<’ topologically.
Let \( n_1, \ldots, n_{m_0} \) be the result.
4. for \( h = 1 \) to \( |N_f| \) do
5. for each edge \( e \in E_h \) leaving \( n_0 \) with \( w(e) \neq \# \) do
6. Let \( e = n_0 \rightarrow n_j \).
7. for \( k = 1 \) to \( K \) do
8. \( df \leftarrow \Delta_{K}[k] + w(e); \)
9. if \( df < \Delta_{K}[K] \) then
10. Replace \( \Delta_{K}[K] \) by \( df \) in \( \Delta_{K} \).
11. \( \Delta \leftarrow \bigcup_{k} \) is a destination \( \Delta_{K} \);
12. return \( \{ \Delta[1], \ldots, \Delta[K] \} \);
end

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

\[
p = (a^i, n'_j) \xrightarrow{p_1} \cdots \xrightarrow{p_l} (a^i, n'_k) \xrightarrow{p_{l+1}} \cdots \xrightarrow{p_m} (a^i, n'_n)
\]

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

\[
w(p_x) = w((a^i, n'_j) \rightarrow (a^i, n'_k)) \cup \cdots \cup w((a^i, n'_{k-1}) \rightarrow (a^i, n'_k)).
\]

Then \( w(p_x) \) represents a diff for \( t_o \) assuming that
1. diffs for \( t_{n_1}, \ldots, t_{n_0} \) are ignored,
2. \( n'_j \) is associated with \( a^i \) for every \( 1 \leq j \leq g \), i.e., we have \( w'([j] = a^i \) due to step (1-a-i) of Trans2, and that
3. under Condition (2) above, \( t_o \) is transformed by the \( l_j \) optimum diff w.r.t. \( (t_{n_j}, D(n_j), op) \) \((1 \leq j \leq g)\).

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

\[
C_{(a^i, n'_j)} = \{ w(p_x) \mid p_x \) is a path from \( (a^i, n'_j) \) to \((a^i, n'_k) \) in \( G(N, E) \}.\]

FindKDiffS shown below computes \( \Delta_{(a^i, n'_j)} \) for every \((a^i, n'_j) \in N \). Similarly to KShortestPaths, we write

\((a^i, n'_j) < (a^i, n'_k) \) if \((a^i, n'_j) \rightarrow (a^i, n'_k) \in E \). Thus, the nodes in \( N \) are visited in a bottom-up manner due to lines 3 and 4. Note that \( G(N, E) \) is acyclic since \( CL(N', E') \) is acyclic. In lines 8 to 10, \( \Delta_{(a^i, n'_j)}[k] \) denotes the \( k \)th optimum diff in \( \Delta_{(a^i, n'_j)} \), and we assume that if \( \Delta_{(a^i, n'_j)}[k] = nil \), then \( |\Delta_{(a^i, n'_j)}[k]| \) is \( \infty \). In line 9, a condition to check if \( df \neq \Delta_{(a^i, n'_j)} \) is added since there may be more than one paths having the same diff, i.e., paths \( p, p' \) from \((a^i, n'_j) \) to \((a^i, n'_k) \) such that \( w(p) = w(p') \). Without this condition \( \Delta_{(a^i, n'_j)} \) might contain duplicated diffs.

FindKDiffS(G(N, E), w)
positive integer $K$.

Output: A graph $G(N, E)$ and a function $w$.

\begin{itemize}
\item begin
\item 1. Append a dummy node $n_{m+1}$ labeled by $x$ as the last child of $n$. Let $df_j(n_{m+1}) \leftarrow \emptyset$ for $1 \leq l \leq K$.
\item 2. Construct the child list graph $CL(N', E')$ of $n$.
\item 3. Construct the Glushkov automaton $G_{\text{glushkov}} = (Q, \Sigma, \delta, \alpha', F)$, where $(d_1, sl) = op(D)$.
\item 4. Construct the product $G(N, E)$ of $G_{\text{glushkov}}$ and $CL(N', E')$.
\item 5. for each edge $(a', n_j') \rightarrow (a, n_j') \in E$ do
\item 6. $\Delta_1 \leftarrow \emptyset$, $\Delta_2 \leftarrow \emptyset$.
\item 7. if $a' \in \delta(a', (a_1, \theta))$ then
\item 8. $\Delta_1 \leftarrow \{df_j(n_j) \mid 1 \leq l \leq K\}$.
\item 9. if $b^h \in \delta(a', b)$ and $a^c \in \delta(b', (a', \theta))$ then
\item 10. if $b^h \in \delta(b', b)$ then
\item 11. $\Delta_2 \leftarrow \{df_j(n_j) \cup S(b, l) \mid 1 \leq l \leq K\}$.
\item 12. else
\item 13. $\Delta \leftarrow \Delta_1 \cup \Delta_2$.
\item 14. $\Delta \leftarrow \Delta_1 \cup \Delta_2$.
\item 15. for $l = 1$ to $K$ do
\item 16. $w((a', n_j'), \rightarrow (a, n_j')) \leftarrow \Delta[l]$
\item 17. return $(G(N, E), w)$.
\item end
\end{itemize}

We show the correctness of the algorithm.

**Theorem 3:** For a DTD $D$, a tree $t$ valid against $D$, an update operation $op$ to $D$, and a positive integer $K$, $\text{MAIN}(D, t, op, K)$ returns $K$ optimum diffs w.r.t. $(t, D, op)$.

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

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

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

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

**Induction:** Let $n$ be a node in $t$ with children $n_1, \ldots, n_m$. As an induction hypothesis, assume that $df_1(n_1), \ldots, df_k(n_1)$ are $K$ optimum diffs w.r.t. $(t_{n_1}, D(n_1), op)$ for every child $n_1$ of $n$. In the following, we consider the case where $l(n) = a$, $l(n_1) \ldots l(n_m) \notin L(d_2(a))$, and $op = \text{ins}_elm(a, b, vi)$ (the other cases can be shown similarly). We have $df_j(n_{m+1}) = 0$ for every $1 \leq l \leq K$ by line 1 of $\text{McGraph1}$. Let $b^h$ be the superscripted label in $d_2(a)^h$ inserted by $op$, and let $(b^{h'}(0)) = \epsilon$ and $(b^{h'}(k)) = (b^{h'})^{(k-1)}b^h$. Moreover, we define that $\Delta_k(l, 1) = S(b, k) \cup df_j(n_1) (1 \leq i \leq m + 1)$. Let $t'_n \in \text{TransOpt}(D(n), t_n, op)$ be a tree such that $\delta(t_n, t'_n)$ is $i$th optimum with $i \leq K$. Then we have

\[
\delta(t_n, t'_n) = A_k(k, l_1) \cup \cdots \cup A_{m+1}(k_{m+1}, l_{m+1})
\]
for some $1 \leq l_1, \cdots, l_{m+1} \leq K$ and some $0 \leq k_1, \cdots, k_{m+1} \leq K$ such that
\[(b^k(a^i_1 \cdots a^j_i))(b^k(a^i_{m+1})) \in L(d_2(a))^k,\]
where $a^i_j$ is a superscripted label such that $(a^i_j)^k \leq l(n_j)$ $(1 \leq j \leq m)$. Let $G(N,E)$ be the product and $w$ be the weight function obtained by MkGraph1. Since $s(t_n, t'_n)$ is $i$th optimum with $i \leq K$, by lines 5 to 16 of MkGraph1 it is easy to show that there is a path
\[(a^i_1, n'_0) \rightarrow (a^i_2, n'_1) \rightarrow \cdots \rightarrow (a^i_{m+1}, n'_{m+1})\]
in $G(N,E)$ such that $(a^i_1, n'_0)$ is the source, $(a^i_{m+1}, n'_{m+1})$ is a destination, and $w((a^i_{j-1}, n'_{j-1}) \rightarrow (a^i_j, n'_j)) = \Delta_j(k_j, l_j)$ for every $1 \leq j \leq m + 1$. Hence $G(N,E)$ covers any paths having desirable diffs. Now it is easy to show that $d_{f_1, \cdots, d_{f_K}}$ are $K$ optimum diffs w.r.t. $(t_n, D(n), op)$ iff there is a path $p_i$ from the source to a destination in $G(N,E)$ such that $w(p_i) = d_{f_i}$ and $p_i$ is the $i$th “shortest” path for every $i \leq K$. Thus, $\text{FindKDiffs}(G(N,E), w)$ in line 20 of $\text{Main}$ correctly returns $K$ optimum diffs w.r.t. $(t_n, D(n), op)$. $\square$

7. Conclusion

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

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

Acknowledgement

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

References


Appendix A: MkGraph Subroutines

Let us first consider $\text{ MkGraph4.}$ We have $op = agg \_ elem(a,b,u)$. Let $G(N,E)$ be the product of $G(d_1(a))$ and $CL(N', E')$, as defined in $\text{ MkGraph2}$, and let $q = sub(d_1(a), u)$. By $op$, for each sequence of nodes in $t$ that maximally match $q$, a node labeled by $b$ is inserted into $t$ as the parent of the nodes. To represent such a node insertion, for each path $(a^i_0, n'_0) \rightarrow \cdots \rightarrow (a^i_j, n'_j) \rightarrow (a^i_{j+1}, n'_{j+1})$ in $G(N,E)$ that “maximally matches” $q$, we add new edges from $(a^i_0, n'_0)$ to $(a^i_j, n'_j)$ to $G(N,E)$ that represent a newly inserted node.

Formally, we say that a path $(a^i_0, n'_0) \rightarrow \cdots \rightarrow (a^i_{m+1}, n'_{m+1})$ in $G(N,E)$ maximally matches $q$ if
\[
(a^i_0, n'_0) \rightarrow \cdots \rightarrow (a^i_{m+1}, n'_{m+1})
\]
is the source of $G(N,E)$ or $a^i_j \notin S \cup c(a^i_k, q^i_k)$, $a^{i+1} \in S \cup c(a^i_k, q^i_k)$ for $1 \leq k \leq n - 1$, and $(a^i_{m+1}, n'_{m+1})$ is a destination of $G(N,E)$ or there is an edge $(a^i_{m+1}, n'_{m+1}) \rightarrow (a^i_{m+1}, n'_{m+1})$ such that $a^i \notin S \cup c(a^i_k, q^i_k)$. 
Suppose that there is a path from \((a^b, n_j')\) to \((a^e, n_j')\) maximally matching \(q\). By \(G((a^b, n_j'), (a^e, n_j'), q)\) we mean the subgraph of \(G(N, E)\) consisting of the paths from \((a^b, n_j')\) to \((a^e, n_j')\) maximally matching \(q\) \((a^e, n_j')\) is the source and \((a^e, n_j')\) is the destination of this subgraph. We create a new edge \(e = (a^e, n_i') \rightarrow (a^e, n_j')\) representing a newly inserted node, say \(v\), and compute \(w(e)\) by taking the union of (i) \([v]\) and (ii) the diff on the \(l\)th shortest path from \((a^b, n_j')\) to \((a^e, n_j')\) in \(G((a^b, n_j'), (a^e, n_j'), q)\). Now \(\text{MkGraph}^4\) is defined as follows. Lines 4 and 5 compute the weight of edges that are not on any paths maximally matching \(q\). Lines 6 to 15 treat the edges representing newly inserted nodes; for each pair \(((a^e, n_i'), (a^e, n_j'))\) of nodes in \(G(N, E)\) such that there is a path maximally matching \(q\) between the nodes, a graph \(G((a^e, n_i'), (a^e, n_j'), q)\) is constructed, then for each 

1 \(\leq l \leq K\), the weight of edge \((a^e, n_i') \rightarrow (a^e, n_j')\) is obtained as shown above and the edge is added to \(G(N, E)\).

\(\text{MkGraph}^4(D, t, n, op, K)\)

Input: A DTD \(D = (d_1, sl)\), a tree \(t\) valid against \(D\), a node \(n\) in \(t\), an update operation \(op = \text{del}_\text{opr}(a, u)\), and a positive integer \(K\).

Output: A graph \(G(N, E)\) and a function \(w\).

begin
1. \(\text{Construct the child list graph } CL(N', E')\) of \(n\).
2. \(\text{Construct the Glushkov automaton } G_\text{opr}(a)\) of \(d_1(a)\).
3. \(\text{Construct the product } G(N,E) \times G_\text{opr}(a)\) and \(CL(N', E')\).
4. \(\text{For each } (a^e, n_i') \rightarrow (a^e, n_j') \in E\) \(\text{let}\)
5. \(w(e) \leftarrow\begin{cases} nil & \text{if } a^e \in \text{sym}(q)\text{'}, \\
\{df(n)\} & \text{otherwise}\end{cases}\)
6. \(P \leftarrow \{(a^e, n_i'), (a^e, n_j')\} \text{ there is a path from } (a^e, n_i') \text{ to } (a^e, n_j') \text{ in } G(N, E) \text{ that maximally matches } q;\)
7. \(\text{for each } ((a^e, n_i'), (a^e, n_j')) \in P \text{ do}\)
8. \(\text{Construct a graph } G' = G((a^e, n_i'), (a^e, n_j'), q).\)
9. \(\text{for each edge } e = (a^e, n_j') \rightarrow (a^e, n_i') \text{ in } G' \text{ let}\)
10. \(w'(e) \leftarrow \{ df(n)\};\)
11. \((df_1, \cdots, df_k) \leftarrow \text{FindKDiff}(G', w').\)
12. \(\text{for each } l = 1 \text{ to } K \text{ do}\)
13. \(\text{Create a new node } v \text{ labeled by } b.\)
14. \(w((a^e, n_i') \rightarrow (a^e, n_j')) \leftarrow\{ v \} \cup \{ df_l\};\)
15. \(\text{Add } (a^e, n_i') \rightarrow (a^e, n_j') \text{ to } E.\)
16. \(\text{return } G(N, E), w;\)
end

Finally, let us show \(\text{MkGraph}^6\). In this case, \(op = \text{change}_\text{opr}(a, opr, u)\). We have four cases to be considered: (i) \(l(d_1(a), u) = "\star\"\) and \(opr = "\star\"\), (ii) \(l(d_1(a), u) = "\star\"\) and \(opr = "?\"\), (iii) \(l(d_1(a), u) = "?\"\) and \(opr = "\star\"\), and (iv) \(l(d_1(a), u) = "?\"\) and \(opr = "?\"\). The cases of (i) and (ii) can be treated similarly to the case of (iii) of \(\text{MkGraph}^5\). The case of (iiii) can be handled similarly to the case of (iv) below. In the following, we consider the case of (iv). Then \(\text{sub}(d_1(a), u) = q^*\) for some regular expression \(q\). Since \(q^*\) is changed to \(q^*\) by \(op\), for each sequence matching \(q^*\), if the sequence is \(e\), we have to insert a sequence of elements matching \(q\). Let \(G_{d_1(a)} = (Q, \Sigma, \delta_1, a^1, F)\) be the Glushkov automaton of \(d_1(a)\) and \(G_{d_2(a)} = (Q, \Sigma, \delta_2, a^2, F)\) be the Glushkov automaton of \(d_2(a)\), where \((d_2, sl) = \text{op}(D)\). For states \(a^i, a^i \notin \text{sym}(q^*)\), we say that the transition from \(a^i\) to \(a^j\) is missing if \(a^i \notin \delta_1(a^i, a^j)\) but \(a^i \notin \delta_2(a^i, a^j)\). If nodes \(n_i, n_i+1\) match \(a^i\) and \(a^j\), respectively, and the transition from \(a^i\) to \(a^j\) is missing, then for a word \(w \in L(q)\), it suffices to insert \([w]\) elements matching \(w\) between \(n_i\) and \(n_i+1\) of \(t\). Thus \(\text{MkGraph}^6\) is defined as follows.

\(\text{MkGraph}^6(t, n, op, K)\)

Input: A DTD \(D = (d_1, sl)\), a tree \(t\) valid against \(D\), a node \(n\) in \(t\), an update operation \(op = \text{change}_\text{opr}(a, opr, u)\), and a positive integer \(K\).

Output: A graph \(G(N, E)\) and a function \(w\).

begin
1. \(\text{Construct the child list graph } CL(N', E')\) of \(n\).
2. \(\text{Construct the Glushkov automaton } G_{d_1(a)}\) of \(d_1(a)\), and the Glushkov automaton \(G_{d_2(a)}\) of \(d_2(a)\).
3. \(\text{Construct the product } G(N,E) \times G_{d_1(a)}\) and \(G(N,E')\).
4. \(\text{Let } w \text{ be a word in } L(\text{sub}(d_1(a), u)).\)
5. \(\text{for each } e = (a^e, n_i') \rightarrow (a^e, n_j') \in E \text{ do}\)
6. if the transition from \( a' \) to \( a \) is missing then
7. Create new nodes \( v_1, \ldots, v_{|w|} \) labeled by
   \( w[1], \ldots, w[|w|] \), respectively.
8. \( w(e) \leftarrow [v_1, \ldots, v_{|w|}] \cup def(n_i) \);
9. else
10. \( w(e) \leftarrow def(n_i) \);
11. return \( (G(N, E), w) \);
end

Appendix B: The Sketch of Proof of Theorem 4

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

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

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

Nobutaka Suzuki

received his B.E. degree in information and computer sciences from Osaka University in 1993, and his M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology in 1995 and 1998, respectively. He was with Okayama Prefectural University as a Research Associate in 1998–2004. In 2004, he joined University of Tsukuba as an Assistant Professor. Since 2009, he has been an Associate Professor of Graduate School of Library, Information and Media Studies, University of Tsukuba. His current research interests include database theory and structured documents.