Vol.2015-DBS-161 No.6
Vol.2015-IFAT-119 No.6
2015/8/5

An Algorithm for Detecting XSLT Rules Affected by Schema Updates

Yang Wu[1,a)] Nobutaka Suzuki[2,b)]

Abstract: Schemas of XML documents are continuously updated according to changes in the real world. Moreover, XSLT stylesheets are also affected by schema updates if its underlying schema is updated. In order to maintain the consistencies of XSLT stylesheets with updated schemas, we have to detect the XSLT rules affected by schema updates. However, detecting such XSLT rules manually is a difficult and time-consuming task, since recent DTDs and XSLT stylesheets are becoming more complex and users do not always fully understand the dependencies between XSLT stylesheets and DTDs. In this paper, we consider three subclasses of XSLT based on unranked tree transducer, and consider an algorithm and complexity for detecting XSLT rules affected by a DTD update for the classes.

Keywords: XML, XSLT, schema evolution

1. Introduction

In general, schemas of XML documents are continuously updated according to changes in the real world. If a schema is updated, then XSLT stylesheets are also affected by the schema update. To maintain the consistencies of XSLT stylesheets with updated schemas, we have to detect the XSLT rules affected by schema updates in order to determine whether the XSLT rules need to be updated accordingly. However, detecting such XSLT rules manually is a difficult and time-consuming task, since recent DTDs and XSLT stylesheets are becoming larger and more complex and users do not always fully understand the dependencies between XSLT stylesheets and old/updated schemas. In this paper, we consider an algorithm for detecting XSLT rules affected by a DTD update automatically.

Let us give a small example of XSLT rules affected by a DTD update. Consider the fragments of old/new DTDs and an XSLT stylesheet shown in Fig. 1. `DTD_old` has two `meta` elements: one is a child of `book` and the other is a child of `info`. The first XSLT rule is applied to the former `meta` element, while the second rule is applied to the latter `meta` element. Here, suppose that the `info` element is unnested, i.e., `info` in the content model of `music` is replaced by "`meta,description?`" (`DTD_new`). Then the first XSLT rule is now applied to the latter `meta` element as well as the former, and we have no `meta` element to which the second XSLT rule is applied. These two XSLT rules are *affected* by the DTD update.

In this paper, we propose an algorithm for detecting XSLT rules affected by a DTD update. We consider three subclasses

```
DTD_old:
<!ELEMENT items (book*,music*)>
<!ELEMENT book  (meta,title,authors)>
<!ELEMENT music (info,title,artist)>
<!ELEMENT meta  (id,date,(pages|length),format?)>
<!ELEMENT info  (meta,description?)>

DTD_new:
<!ELEMENT items (book*,music*)>
<!ELEMENT book  (meta,title,authors)>
<!ELEMENT music (meta,description?,title,artist)>
<!ELEMENT meta  (id,date,(pages|length),format?)>

XSLT:
<xsl:template match="meta">
  ...
</xsl:template>

<xsl:template match="info/meta">
  ...
</xsl:template>
```

Fig. 1 Fragments of DTDs and XSLT

of XSLT: UTT, UTT^{pat}, and UTT^{pat,sel}. UTT coincides with the standard unranked tree transducer [7], and UTT^{pat} and UTT^{pat,sel} are extensions of UTT, where *pat* denotes XSLT pattern and *sel* denotes `select` of `apply-templates`. We first give a polynomial-time algorithm for detecting XSLT rules affected by a DTD update assuming UTT/UTT^{pat} as XSLT. We next show that the problem becomes undecidable if UTT^{pat,sel} is assumed as XSLT. We also made an experiment on the algorithm.

Related Work

[3] proposes an algorithm for transforming XPath expressions according to a schema update. Although XPath expressions are used as XSLT patterns, their algorithm cannot be applied to our problem since XSLT rules affected by a schema update cannot be detected by checking each XSLT pattern independently. To

[1] Graduate School of Library Information and Media Studies, University of Tsukuba, 1-2 Kasuga, Tsukuba 305-8850, Japan
[2] Faculty of Library, Information and Media Science, University of Tsukuba, 1-2 Kasuga, Tsukuba 305-8850, Japan
[a)] wuyang65432@yahoo.co.jp
[b)] nsuzuki@slis.tsukuba.ac.jp

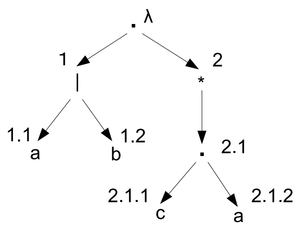

Fig. 2 Tree structure of r

the best of our knowledge, there is no study on detecting XSLT rules affected by a schema update. On the other hand, there are several studies dealing with XML schema updates. For example, [4], [6] propose algorithms for extracting "diff" between two schemas. [2], [9] propose update operations that assures any updated schema contains its original schema. [8] introduces a taxonomy of possible problems for XQuery induced by a schema update, and gives an algorithm to detect such problems. [5] studies query-update independence analysis, and shows that the performance of [1] can be drastically enhanced in the use of μ-calculus.

This paper is organized as follows. Section 2 defines DTD, tree transducer, and some related notions. Section 3 presents an algorithm for detecting XSLT rules affected by schema updates. Section 4 shows the undecidability of the problem. Section 5 briefly gives the result of an experimentation. Section 6 summarizes this paper.

2. Preliminaries

In this section, we give some definitions related to DTD and tree transducer.

2.1 DTD and Update Operations to DTDs

Let Σ be a set of labels. For a node v in a tree t, by $l(v)$ we mean the label of v. The language specified by a regular expression r is denoted $L(r)$. A DTD is a tuple $D = (d, sl)$, where d is a mapping from Σ to the set of regular expressions over Σ, and $sl \in \Sigma$ is the start label. For a label $a \in \Sigma$, $d(a)$ is the *content model* of a. A tree t is *valid* against $D = (d, sl)$ if $l(v) = sl$ for the root v of t and for any node n in t, $l(v_1)\cdots l(v_n) \in L(d(l(v)))$, where $v_1, \cdots, v_n$ are the child nodes of v.

Example 1 Consider the following DTD, where book is the start label. Then this DTD is denoted $(d, book)$, where $d(\text{book}) = \text{title chapter}^+$, $d(\text{chapter}) = \text{title section}^+ \text{bib?}$, $d(\text{title}) = d(\text{section}) = d(\text{bib}) = \epsilon$.

```
<!ELEMENT book    (title,chapter+)>
<!ELEMENT chapter (title, section+, bib?)>
<!ELEMENT title   (#PCDATA)>
<!ELEMENT section (#PCDATA)>
<!ELEMENT bib     (#PCDATA)>
```

To define update operations to DTDs, we need to define the positions of elements/operators in a content model. Thus, we represent a content model as a tree and specify the position of each node by Dewey order. For example, Fig. 2 shows the tree structure of $r = (a|b)(ca)^*$, where each node is associated with its position. For a regular expression r, the label at position u in r is denoted $l(r, u)$ and the subexpression at position u of r is denoted $sub(r, u)$. For example, in Fig. 2, $l(r, 1) =$ '|', $l(r, 1.1) = a$, $sub(r, 2.1) = ca$.

Let $D = (d, sl)$ be a DTD. *Update operations* to D are defined as follows.

- $ins\_elm(a, b, u)$: inserts a label b at position u in $d(a)$.
- $del\_elm(a, u)$: deletes the label at position u in $d(a)$.
- $nest(a, b, u)$: nests the subexpression at u in $d(a)$ by b. This operation replaces the subexpression at u in $d(a)$ by b and sets $d(b) = sub(d(a), u)$.
- $unnest(a, u)$: this is the inverse operation of $nest$, and replaces the label $l' = l(d(a), u)$ at u in $d(a)$ by regular expression $d(l')$.

By $op(D)$ we mean the DTD obtained by applying an update operation op to D. An *update script* is a sequence of update operations. For an update script $s = op_1 op_2 \cdots op_n$, we define $s(D) = op_n(\cdots(op_2(op_1(D))))$.

2.2 Classes UTT and UTT^{pat} of Tree Transducers

A *pattern* is defined as $pat = ls_1/\cdots/ls_n$, where $ls_i = ax_i :: l_i$, $ax_i \in \{\downarrow, \downarrow^*\}$, and $l_i \in \Sigma$. $\downarrow$ and $\downarrow^*$ denote child and descendant-or-self axes, respectively. Let t be a tree and v be a node of t. We say that v *matches* pat if there is a sequence $v_1, \cdots, v_n$ of nodes in t such that $v_n = v$, $l(v_i) = l_i$ $(1 \le i \le n)$, and that for any $2 \le i \le n$, if $ax_i = \downarrow$, then t has edge $v_{i-1} \to v_i$, otherwise (i.e., $ax_i = \downarrow^*$) there is a path from v_{i-1} to v_i in t.

A *hedge* is a finite sequence of trees. The set of hedges is denoted by H_Σ. For a set Q, by $H_\Sigma(Q)$ we mean the set of Σ-hedges such that leaf nodes can be labeled with elements from Q. A *tree transducer* is a quadruple (Q, Σ, q_0, R), where Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, and R is a finite set of *rules* of the form $(q, pat) \to h$, where $q \in Q$, pat is a pattern, and $h \in H_\Sigma(Q)$. For example, $(q, a/b/c) \to c(p)$ corresponds to the following XSLT template.

```
<xsl:template match="a/b/c" mode="q">
  <c>
    <xsl:apply-templates mode="p" />
  </c>
</xsl:template>
```

Let v be a node in a tree t. The translation defined by a tree transducer $Tr = (Q, \Sigma, q_0, R)$ at v in state q, denoted by $Tr^q(t, v)$, is inductively defined as follows.

R1: If there is a rule $(q, pat) \to h \in R$ such that v matches pat, then $Tr^q(t, v)$ is obtained from h as follows: for each leaf node u in h, if $l(u)$ is a state, say p, then replace u with hedge $Tr^p(t, v_1)\cdots Tr^p(t, v_n)$, where $v_1, \cdots, v_n$ are the children of v.

R2: Otherwise, $Tr^q(t, v) = \epsilon$.

The transformation of t by Tr, denoted by $Tr(t)$, is defined as $Tr(t) = Tr^{q_0}(t, v_0)$, where v_0 is the root node of t. The class of the tree transducers defined above is denoted UTT^{pat}. In particular, if for every rule $(q, pat) \to h \in R$ pat is a single label, then the restricted class is denoted UTT, which coincides with that of the standard unranked tree transducer[7].

2.3 Class UTT^{pat,sel}

We first show the definitions of XPath location paths used in select. A *relative location path* is defined as $ls_1/\cdots/ls_n$, where $ls_i = ax_i :: l_i, ax_i \in \{\downarrow, \uparrow, \downarrow^*\}$, l_i is a label, and $\uparrow$ is a parent axis. An *absolute location path* consists of '/' optionally followed by a relative location path. The set of relative location paths and

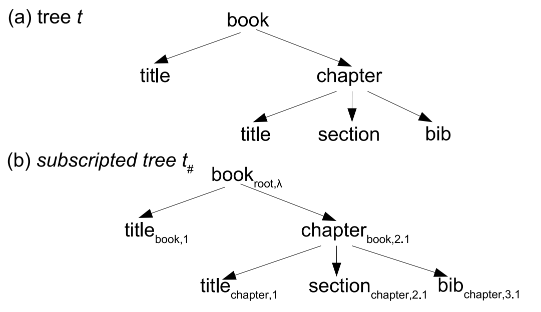

Fig. 3 Tree t and its subscripted tree $t_\#$

absolute location paths is denoted by SEL. By $H_\Sigma(Q \times SEL)$ we mean the set of hedges such that leaf nodes can be labeled with elements from $(q, sel) \in Q \times SEL$.

A tree transducer in UTT^{pat,sel} can also be defined as a quadruple (Q, Σ, q_0, R'). The rules of transformation are extended as follows: for every transformation rule $(q, pat) \to h$ in R', h belongs to $H_\Sigma(Q\times SEL)$. The other definitions remain the same as defined in UTT^{pat}. For a relative location path sel, by $S(t, v, sel)$ we mean the set of nodes reachable from v via sel in t. In the same way, for an absolute location path sel, by $S(t, sel)$ we mean the set of nodes reachable from the root of t via sel.

Let t be a tree and v be a node of t. The translation defined by a tree transducer $Tr = (Q, \Sigma, q_0, R')$ on node v of tree t in state q, denoted by $Tr^q(t, v)$, is defined as follows.

- The case where there is a rule $(q, pat)\to h\in R'$ such that $M_{pat}(t, v, pat) \neq \emptyset$:
  - If sel is a relative location path, then $Tr^q(t, v)$ is obtained from h as follows:
    - for each leaf node u in h, if $l(u) = (p, sel) \in Q \times SEL$, then replace u with hedge $Tr^p(t, v_1) \cdots Tr^p(t, v_n)$, where $S(t, v, sel) = \{v_1, \cdots, v_n\}$.
  - If sel is an absolute location path, then $Tr^q(t, v)$ is obtained from h as follows:
    - for each leaf node u in h, if $l(u) = (p, sel) \in Q \times SEL$, then replace u with hedge $Tr^p(t, v_1) \cdots Tr^p(t, v_n)$, where $S(t, sel) = \{v_1, \cdots, v_n\}$.
- The case where there is no rule $(q, pat)\to h\in R'$ such that $M_{pat}(t, v, pat) \neq \emptyset$:
  - $Tr^q(t, v) = \epsilon$.

The transformation of t by Tr, denoted by $Tr(t)$, is defined as $Tr^{q_0}(t, v_0)$, where v_0 is the root node of t. The class of the tree transducers defined above is denoted UTT^{pat,sel}.

2.4 Rules Affected by DTD Updates

A DTD may contain more than one element having the same name, and we have to distinguish such elements when detecting the rules affected by DTD updates. By $a_{b,u}$ we mean the element a at position u in $d(b)$. We say that $a_{b,u}$ is a *subscripted* label. If a is the root element, then the corresponding subscripted labels is $a_{root,\lambda}$. By $D_\#$ we mean the DTD obtained from D by replacing each label in a content model with its corresponding subscripted label. For a tree t valid against D, $t_\#$ is a *subscripted tree* of t if $t_\#$ is obtained by replacing each label in t with its corresponding subscripted label so that $t_\#$ is valid against $D_\#$ (see Fig. 3).

Let $D = (d, sl)$ be a DTD, s be an update script to D, and

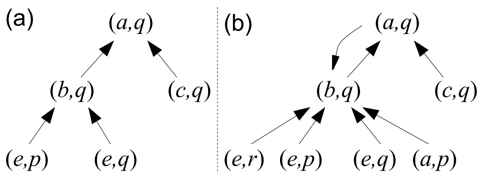

Fig. 4 Dependency graphs

$s(D) = (d', sl)$. For a subscripted label $a_{b,u}$ in $D_\#$, if $a_{b,u}$ is not deleted by s, then $a_{b,u}$ also appears in $s(D)_\#$ (its parent b and position u may change). Thus, this element in $s(D)_\#$ can be denoted by $a_{b',u'}$ for some element b' and position u', and we say that $a_{b,u}$ *corresponds* to $a_{b',u'}$ [*1].

Let $Tr = (Q, \Sigma, q_0, R)$ be a tree transducer. For a subscripted label $a_{b,u}$ and a rule $rl \in R$, rl is *applicable* to $a_{b,u}$ in a tree t if for some node v in t, (1) rl is applied to v during the transformation of $Tr(t)$, and, (2) for some subscripted tree $t_\#$ of t, the label of v is $a_{b,u}$ in $t_\#$.

Let $a_{b',u'}$ be a subscripted label in $s(D)_\#$ and $a_{b,u}$ be its corresponding element in $D_\#$. We define two sets of rules *affected* by s at $a_{b',u'}$, denoted $R^+(a_{b',u'})$ and $R^-(a_{b',u'})$, as follows.

- $R^+(a_{b',u'})$ is the set of rules $rl \in R$ such that rl is not applicable to $a_{b,u}$ in $D_\#$ but becomes applicable to $a_{b',u'}$ in $s(D)_\#$.
- $R^-(a_{b',u'})$ is the set of rules $rl \in R$ such that rl is applicable to $a_{b,u}$ in $D_\#$ but not applicable to $a_{b',u'}$ in $s(D)_\#$. In particular, if $D_\#$ has no subscripted label corresponding to $a_{b',u'}$, then $R^-(a_{b',u'}) = \emptyset$.

3. Algorithm

In this section, we present an algorithm for computing $R^+(a_{b',u'})$ and $R^-(a_{b',u'})$ for every $a_{b',u'}$, assuming a tree transducer belongs to UTT/UTT^{pat}.

To compute these sets, we have to find the rules applicable to each element. To do this, we use label-state pairs and find dependencies of such pairs. In short, pair (a, q) means that a rule can be applied to a in state q. Consider the rule R1 in the definition of UTT/UTT^{pat}, and suppose that the antecedent of the rule R1 holds. This means that a rule $(q, pat) \to h$ is applied to a node v in state q, and thus we have a pair (a, q) with $l(v) = a$. Then consider the consequence of the rule R1. Each state p in h is replaced by $Tr^p(t, v_1) \cdots Tr^p(t, v_n)$. Let $b = l(v_i)$. Then $Tr^p(t, v_i)$ means that a rule is (possibly) applied to b in state p, thus we obtain (b, p). Since (b, p) is obtained by (a, q), we denote this dependency by an edge $(b, p) \to (a, q)$. A *dependency graph* is a graph $G_D = (V_D, E_D)$ consisting of such nodes and edges.

Example 2 Let $D = (d, a)$ be a DTD, where $d(a) = bc$, $d(b) = e$, $d(c) = d(e) = \epsilon$. Let $Tr = (Q, \Sigma, q, R)$ be a tree transducer, where $Q = \{p, q, r\}$, $\Sigma = \{a, b, c, e\}$, and $R = \{(q, a) \to a(q), (q, c) \to c, (q, b) \to b(pq), (q, b/a/b) \to b(r)\}$. Since the root element is a and the initial state is q, we obtain (a, q). Since $d(a) = bc$ and $(q, a) \to a(q)$ can be applied to a in state q, we obtain nodes (b, q), (c, q) and edges $(b, q) \to (a, q)$ and $(c, q) \to (a, q)$. By applying rules in R similarly, we obtain the dependency graph in Fig. 4(a).

[*1] In some cases the update script between old and new DTDs is not given explicitly. In such cases, the algorithms in [4], [6] can generate update scripts between DTDs.

To define the algorithm formally, we need some definitions. By $St(h)$ we mean the set of states in a hedge h. For example, if $h = a(pq)$, then $St(h) = \{p, q\}$. Let $pat = ax_1 :: l_1/\cdots/ax_n :: l_n$ be a pattern. A rule $(q, pat) \to h$ is *applicable* to (a, q) in G_D if there is a sequence $(a_1, q_1), \cdots, (a_n, q_n)$ such that (M1) $(a, q) = (a_n, q_n)$, (M2) $a_i = l_i$ $(1 \le i \le n)$, and that (M3) for every $2 \le i \le n$, if $axis_i = \downarrow$, then $(a_i, q_i) \to (a_{i-1}, q_{i-1}) \in E_D$, otherwise (i.e., $axis_i = \downarrow^*$) there is a path from (a_i, q_i) to (a_{i-1}, q_{i-1}) in G_D.

We present algorithm FindDep for constructing a dependency graph. $R(a, q)$ denotes the set of rules applied to (a, q) so far. S maintains a set of label-state pairs that should be examined, which is initially $\{(sl, q_0)\}$ (line 3). Then the following is repeated until S becomes empty. First, the algorithm chooses an arbitrary pair (a, q) from S (line 5), then finds a rule $rl = (q, pat) \to h$ such that rl is applicable to (a, q) in G_D and that rl is not applied to (a, q) so far (line 6). According to rl and the consequence of the rule R1, for every $q' \in St(h)$ and every child element b of a, if (b, q') is a newly found pair, then (b, q') is added to V_D (and S) and $(b, q') \to (a, q)$ is added to E_D (lines 9 to 13). On the other hand, if (b, q') is already in V_D, only $(b, q') \to (a, q)$ is added to E_D (lines 14 to 15). Due to this edge addition, some rule may become applicable to a pair (c, q'') if (a, q) becomes reachable from (c, q'') via $(b, q') \to (a, q)$ (see Example 3 below). Thus we find such pairs and add to S (lines 16 to 18).

Algorithm FindDep

Input: DTD $D = (d, sl)$, tree transducer $Tr = (Q, \Sigma, q_0, R)$.
Output: Dependency graph $G_D = (V_D, E_D)$.

```
( 1 ) $V_D \leftarrow \{(sl, q_0)\}$; $E_D \leftarrow \emptyset$
( 2 ) $R(sl, q_0) \leftarrow \emptyset$
( 3 ) $S \leftarrow \{(sl, q_0)\}$;
( 4 ) while $S \neq \emptyset$ do
( 5 )     Choose a pair $(a, q) \in S$. Delete $(a, q)$ from $S$.
( 6 )     if there is a rule $rl = (q, pat) \to h \in R$ such that $rl$ is
          applicable to $(a, q)$ in $G_D$ and that $rl \notin R(a, q)$ then
( 7 )         $R(a, q) \leftarrow R(a, q) \cup \{rl\}$
( 8 )         for each element $b$ appearing in $d(a)$ and each state
              $q' \in St(h)$ do
( 9 )             if $(b, q') \notin V_D$ then
( 10 )                $V_D \leftarrow V_D \cup \{(b, q')\}$
( 11 )                $E_D \leftarrow E_D \cup \{(b, q') \to (a, q)\}$
( 12 )                $R(b, q') \leftarrow \emptyset$
( 13 )                $S \leftarrow S \cup \{(b, q')\}$
( 14 )            else if $(b, q') \to (a, q) \notin E_D$ then
( 15 )                $E_D \leftarrow E_D \cup \{(b, q') \to (a, q)\}$
( 16 )                for each $(c, q'') \in V_D$ such that $(a, q)$ becomes
                      reachable from $(c, q'')$ via $(b, q') \to (a, q)$ do
( 17 )                    if there is a rule $rl \in R$ such that $rl$ is applicable
                          to $(c, q'')$ in $G_D$ and that $rl \notin R(b, q')$ then
( 18 )                        $S \leftarrow S \cup \{(c, q'')\}$;
( 19 ) return $G_D = (V_D, E_D)$
```

Example 3 Consider the DTD and the tree transducer in Example 2. Then suppose that D is slightly modified so that $d(b) = ea$ instead of $d(b) = e$. Since element a is now a child of b, by rule $(q, b) \to b(pq)$ we obtain a new node (a, p) and new edges $(a, p) \to (b, q)$ and $(a, q) \to (b, q)$ (Fig. 4(b)). Moreover, since rule $(q, b/a/b) \to b(r)$ is now applicable to (b, q) due to path $(b, q) \to (a, q) \to (b, q)$, we obtain new node (e, r) and new edge $(e, r) \to (b, q)$.

To present an algorithm for computing $R^+(a_{b',u'})$ and $R^-(a_{b',u'})$, we need a definition. A rule $(q, pat) \to h$ is *applicable* to (a, q) *with parent* b in G_D if there is a sequence $(a_1, q_1), \cdots, (a_n, q_n)$ that satisfies the following condition as well as the conditions M1 to M3.

M4) If $axis_n = \downarrow$, then $a_{n-1} = b$. Otherwise (i.e., $axis_n = \downarrow^*$), there is a path from (a_n, q_n) to (a_{n-1}, q_{n-1}) whose first edge is $(a_n, q_n) \to (b, q')$ for some $q' \in Q$.

The following algorithm computes $R^-(a_{b',u'})$ for every $a_{b',u'}$ ($R^+(a_{b',u'})$ can be obtained similarly). This algorithm constructs dependency graphs G_D and G'_D for old/new DTDs, and then takes "diff" between G_D and G'_D to obtain $R^-(a_{b',u'})$.

Algorithm Main

Input: DTD $D = (d, sl)$, update script s to D, tree transducer $Tr = (Q, \Sigma, q_0, R)$.
Output: $R^-(a_{b',u'})$ for every $a_{b',u'}$ in $s(D)$.

```
( 1 ) $G_D \leftarrow$ FindDep($D, Tr$)
( 2 ) $G'_D \leftarrow$ FindDep($s(D), Tr$)
( 3 ) for each subscripted element $a_{b',u'}$ in $s(D)$ do
( 4 )     $R^-(a_{b',u'}) \leftarrow \emptyset$
( 5 )     if $D$ has a subscripted element $a_{b,u}$ corresponding
          to $a_{b',u'}$ then
( 6 )         $M \leftarrow \{rl \in R \mid rl$ is applicable to $(a, q)$ with parent $b$
              in $G_D, q \in Q\}$
( 7 )         $M' \leftarrow \{rl \in R \mid rl$ is applicable to $(a,q)$ with parent $b'$
              in $G'_D, q \in Q\}$
( 8 )         $R^-(a_{b',u'}) \leftarrow M \setminus M'$
( 9 ) return $\{R^-(a_{b',u'}) \mid a_{b',u'}$ is a subscripted element in $s(D)\}$
```

Let $M_c = \max_{a \in \Sigma} |d(a)|$ and $M_s = \max_{(q,pat) \to h \in R} |St(h)|$, where $|d(a)|$ denotes the number of labels appearing in $d(a)$. Then algorithm Main runs in $O(|R|^5 (M_c M_s)^3)$. In particular, assuming a tree transducer belongs to UTT, the algorithm runs in $O(|R|^2 (M_c M_s))$. We also have the following.

Theorem 1 Let D be a DTD, s be an update script to D, $T_r = (Q, \Sigma, q_0, R)$ be a tree transducer belonging to UTT/UTT^{pat}, and $a_{b,u}$ be a subscripted element in $s(D)$. Then $rl \in R^-(a_{b',u'})$ iff $rl \in R'^-(a_{b',u'})$, where R'^- is the result of Main.

Proof (sketch): Let G_D be the dependency graph obtained by FindDep(D, T_R). It suffices to show that rl is applicable to (a, q) with parent b in G_D if and only if rl is applicable to $a_{b,u}$ in D.

If part: Suppose that rl is applicable to $a_{b,u}$ in D. Then there is a tree valid against D containing a path $n_0 \to n_1 \to \cdots \to n_n$ and a sequence of $n + 1$ states $q_0, q_1, \cdots, q_n$ such that

- $l(n_0) = s$, $l(n_i)$ is contained in $d(l(n_{i-1}))$ $(1 \le i \le n)$, $l(n_{n-1}) = b$, and $l(n_n) = a$, and that
- there is a rule $(q_i, pat_i) \to h_i \in R$ such that n_i matches pat_i and that h_i contains q_{i+1} $(0 \le i \le n)$.

This implies that G_D contains a path $(a_0, q_0) \leftarrow (a_1, q_1) \leftarrow \cdots \leftarrow (a_n, q_n)$ such that $a_0 = s$, $a_{n-1} = b$, and that $a_n = a$. Thus the if part holds.

Only if part: Suppose that rl is applicable to (a, q) with parent b in G_D. Then it is easy to show that we can construct a tree t valid against D such that rl is applicable to $a_{b,u}$ in D. □

4. Undecidability

In this section, we show that if UTT^{pat,sel} is assumed as XSLT, then deciding whether there is a rule affected by a schema update is undecidable.

Theorem 2 The following problem is undecidable even if s is of length one.

- Instance: DTD D, Tree transducer T_r in UTT^{pat,sel}, edit script s to D, subscripted element $a_{b,u}$ in $s(D)$
- Problem: Determine whether there is a rule of T_r affected by s at $a_{b,u}$.

Proof: It is known that the halting problem of Turing machine is undecidable. We reduce this halting problem to the above problem. Without loss of generality, we assume that a Turing machine has a one-way infinite tape.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a Turing machine, where Q is a set of states, Σ is a set of input symbols, Γ is a set of tape symbols ($\Sigma \subseteq \Gamma$), δ is a transition function, $q_0 \in Q$ is the initial state, B is the blank symbol, and F is a set of accepting states ($F \subseteq Q$). $\delta(q, a)$ is either defined or undefined. If $\delta(q, a)$ is defined, then the value is denoted (p, b, D), where

- p is the state in which M enters after the transition,
- $b \in \Gamma$ is the symbol written into the current cell, and
- $D \in \{L, R\}$ is the move direction (L and R stand for "left" and "right", respectively).

That is, if $\delta(q, a) = (p, b, D)$, then the state of M changes from q to p, the content of the current cell changes from a to b, and the head moves in direction D.

We define a tree transducer T_r so that T_r simulates M. In addition, we add "extra states" to M so that a rule of T_r is affected by an update to DTD D if and only if M halts. Let $a[1], a[2], \cdots, a[n]$ be the initial content of the tape of M. From the tape and $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, we define a DTD D, a tree transducer T_r, and an edit operation op, as follows.

Firstly, DTD $D = (d, s)$ is defined as follows.

$$\begin{aligned} d(s) &= b_1 \\ d(b_i) &= b_{i+1} \quad (1 \le i \le n-1) \\ d(b_n) &= B \\ d(B) &= B|\epsilon \end{aligned}$$

where $s, b_1, \cdots, b_n, B \notin \Sigma$. B is a label representing the blank symbol. Figure 5(a) shows a tree valid against D.

We next define a tree transducer $T_r = (Q \cup Q' \cup \{q_0', r, u\}, \Sigma \cup B', q_0', R' \cup R_M)$. First, $Q' = \{q_1, \cdots, q_{n+1}\}$ and $B' = \{b_1, \cdots, b_n, B\}$. R' consists of the following rules.

$$\begin{aligned} (q_0', s) &\rightarrow s(q_1, b_1) \\ (q_1, b_1) &\rightarrow a[1](q_2, b_2) \\ &\vdots \\ (q_{n-1}, b_{n-1}) &\rightarrow a[n-1](q_n, b_n) \\ (q_n, b_n) &\rightarrow a[n](q_{n+1}, B) \\ (q_{n+1}, B) &\rightarrow B(q_0, /s/a[1]) \end{aligned}$$

Intuitively, R' constructs the "input tape" $a[1], \cdots, a[n]$ from $b_1, \cdots, b_n$ and then moves to the child $a[1]$ of the root s. On the other hand, R_M is a set of rules to simulate M. For any state $q \in Q$ and any symbol $a \in \Sigma$, R_M contains the following rule corresponding to $\delta(q, a)$.

(1) The case where $\delta(q, a) = (p, b, L)$: $(q, a) \rightarrow b((p, \uparrow\!:: *)) \in R_M$[*2]. That is, if a node labeled by a is matched in state q, a node b is created and the current node is moved to its parent in state p.

(2) The case where $\delta(q, a) = (p, b, R)$: $(q, a) \rightarrow b((p, \downarrow\!:: *)) \in R_M$. That is, if a node labeled by a is matched in state q, a node labeled by b is created and the current node is moved to its child in state p.

(3) The case where $\delta(q, a)$ is undefined: $(q, a) \rightarrow a(r, /)) \in R_M$. Since $\delta(q, a)$ is undefined, M halts. In this case, T_r moves to the root in state r, where r is a new state not in Q.

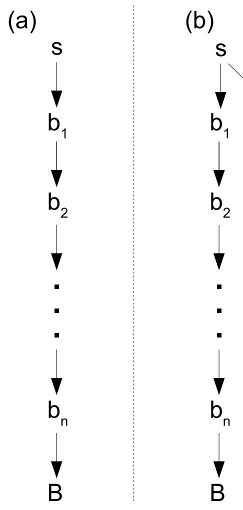

Fig. 5 Trees valid against D and $op(D)$.

Besides the above rules, R_M has the following two rules. These rules mean that if T_r moves to the root s in state r, then child c of s is replaced by e.

(4) $(r, s) \rightarrow s((u, \downarrow\!:: c))$

(5) $(u, c) \rightarrow e$

Finally, let $op = ins\_elm(d(s), 2, c)$. Figure 5(b) shows a tree valid against $op(D)$.

By the definition of T_r, we have the following.

- If M halts, then rules 4 and 5 are affected by op by the following (1) and (2).
  (1) In the update tree, child c of s is replaced by e by rules 4 and 5.
  (2) On the other hand, the rules are not applied anywhere in the original tree since s does not have a child labeled by c.
- If M does not halt, then rule 3 is never applied. Thus, T_r does not enter state r and thus rule 4 is not applied either. Therefore, no rule is affected by op.

Consequently, M halts if and only if there is a rule affected by op at $c_{s,2}$. □

Table 1 shows the summary of the result obtained in this paper.

Table 1 Summary of the Result

Class	Complexity
UTT	$O(\lvert R\rvert^2(M_C M_S))$
UTT^{pat}	$O(\lvert R\rvert^5(M_C M_S)^3)$
UTT^{pat,sel}	undecidable

[*2] $(q, a) \rightarrow b((p, \uparrow\!:: *))$ denotes k rules $(q, a) \rightarrow b((p, \uparrow\!:: a_1)), \cdots, (q, a) \rightarrow b((p, \uparrow\!:: a_k))$, where $\Sigma = \{a_1, \cdots, a_k\}$.

Table 2 Update operations between D and $s(D)$

ins_elm	*del_elm*	*nest*	*unnest*	Total
68	11	27	0	106

5. Evaluation Experiment

We implemented our algorithm in Java and conducted a preliminary experiment. The DTDs used in our experiment are version 2.1.1 and version 2.2.2 of MSRMEDOC*3. We denote the former as D and the later as $s(D)$, respectively. MSRMEDOC is a format for information interchange in the development process of production and supply. The numbers of elements in D and $s(D)$ are 183 and 204, respectively. Table 2 shows the number of update operations between D and $s(D)$.

Since we didn't find any XSLT stylesheet for the DTDs, we made an XSLT stylesheet for XML to HTML transformation. The stylesheet has 10 rules and the average length of the patterns is 4. In the experiment, we have two examinees who are both graduate students and are familiar with DTD and XSLT. We explained the definitions of the DTDs, R^+ rule, and R^- rule, and related examples to the examinees in advance. Then, we presented the stylesheet to the examinees and asked them to find R^+/R^- rules manually. For both examinees 18 minutes was required to find all the R^+/R^- rules. On the other hand, the execution time of our algorithm is 2135ms under a mobile PC with Intel Core i3 2.60GHz. This result suggests that our algorithm can save much time to maintain the consistencies of XSLT stylesheets.

In this preliminary experiment, we used a relatively small XSLT stylesheet. However, we expect that it will take even more time to detect the R^+/R^- rules manually in larger XSLT stylesheets. We will investigate this further.

6. Conclusion

In this paper, we considered three subclasses of XSLT, UTT, UTT^{pat}, and UTT^{pat,sel}. We proposed a polynomial-time algorithm for detecting XSLT rules affected by a DTD update for UTT and UTT^{pat}. We also showed that the problem is undecidable assuming UTT^{pat,sel} as XSLT.

As a future work, we would like to conduct more experiments by using more DTDs and XSLT stylesheets.

Acknowledgement

This work is partially supportted by Grants-in-aid for Scientific Research (# 26330126).

References

[1] Benedikt, M. and Cheney, J.: Destabilizers and independence of XML updates, *Proc. VLDB Endow.*, Vol. 3, No. 1-2, pp. 906–917 (2010).
[2] Guerrini, G., Mesiti, M. and Rossi, D.: Impact of XML Schema Evolution on Valid Documents, *Proc. WIDM*, pp. 39–44 (2005).
[3] Hasegawa, K., Ikeda, K. and Suzuki, N.: An Algorithm for Transforming XPath Expressions According to Schema Evolution, *Proc. DChanges 2013* (2013).
[4] Horie, K. and Suzuki, N.: Extracting differences between regular tree grammars, *Proc. ACM SAC*, pp. 859–864 (2013).
[5] Junedi, M., Genevès, P. and Layaïda, N.: XML query-update independence analysis revisited, *Proceedings of the 2012 ACM symposium on Document engineering*, DocEng '12, pp. 95–98 (2012).
[6] Leonardi, E., Hoai, T. T., Bhowmick, S. S. and Madria, S.: DTD-Diff: A Change Detection Algorithm for DTDs, *Data & Knowledge Engineering*, Vol. 61, pp. 384–402 (2007).
[7] Martens, W. and Neven, F.: Typechecking Top-Down Uni- form Unranked Tree Transducers, *Proc. ICDT*, pp. 64–78 (2002).
[8] Oliveira, R., Genevès, P. and Layaïda, N.: Toward automated schema-directed code revision, *Proceedings of the 2012 ACM symposium on Document engineering*, DocEng '12, pp. 103–106 (2012).
[9] Suzuki, N.: An Edit Operation-Based Approach to the Inclusion Problem for DTDs, *Proc. ACM SAC*, pp. 482–488 (2007).

*3 http://www.msr-wg.de/medoc/downlo.html