Detecting XSLT Rules Affected by DTD Updates

Yang WU∗ Nobutaka SUZUKI○

Schemas of XML documents may be updated for various reasons. If a schema is updated, 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 users do not always fully understand the dependencies between XSLT stylesheets and DTDs. In this paper, we consider three classes of tree transducers as subsets of XSLT, and investigate whether XSLT rules affected by DTD updates can be detected efficiently for these classes.

1. Introduction

Schemas of XML documents may be updated for various reasons, e.g., changes to real world events, user's requirement changes, and initial design mistakes [6]. 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 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 users do not always fully understand the dependencies between XSLT stylesheets and DTDs. In this paper, we consider detecting XSLT rules affected by a DTD update.

Let us give a small example of XSLT rules affected by a DTD update (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?”, as shown in 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. We say that these two XSLT rules are affected by the DTD update.

It is not trivial to detect whether XSLT rules are affected by DTD updates since the behavior of an XSLT rule depends on other XSLT rules as well as the definition of a DTD. In this paper, we investigate whether the problem can be solved efficiently or not. Specifically, we introduce three classes of tree transducers modeling subclasses of XSLT: UTT, UTT′, and UTT′′, and investigate whether the problem can be solved efficiently or not for these classes. Here, UTT coincides with the standard unranked tree transducer [7], and UTT′ and UTT′′ are extensions of UTT, where put denotes XSLT pattern and sel denotes select of apply-templates. We firstly give a polynomial-time algorithm for detecting XSLT rules affected by a DTD update assuming UTT as XSLT. We next show that the problem becomes NP-hard for UTT′′, and even under very restricted DTDs.

Related Work

The study mostly related to this paper is [3]. Their algorithm transforms XPath expressions according to schema updates. Although XPath expressions are used as patterns in XSLT stylesheets, their algorithm cannot be applied to our problem. This is because XSLT rules affected by schema updates cannot be detected by checking each XSLT pattern independently since XSLT rules depend on each other. To the best of our knowledge, there is no study on detecting XSLT rules affected by schema updates.

On the other hand, there are studies dealing with XML schema updates. For example, Leonardi et al. propose algorithms for extracting “diff” between two DTDs [6], and Horie et al. propose algorithms for extracting “diff” between two tree grammars [5]. Guerrini et al. propose update operations such that any updated schema “contains” its original schema [2]. This ensures that documents under an original schema remains valid under its updated schema. Hashimoto et al. propose update operations to tree grammars so that schema’s expressive power is preserved [4]; a tree valid to an original grammar is embeddable to a tree valid to its updated grammar. Oliveira et al. introduce a taxonomy of possible problems for XQuery induced by schema updates, and gives an algorithm to detect such problems [10].

The XML typechecking problem is a problem related to XSLT and schema. This problem is to decide, for a tree transducer Tr and schemas S1, S2, whether the tree obtained by transforming by Tr is valid to S2, for any tree t valid to S1. The problem is shown to be intractable for a number of cases (e.g., [7, 8]), and is also intractable for UTT [7].

2. Preliminaries

In this section, we firstly give some definitions related to DTD. Then we define three classes of tree transducers.

2.1 DTD and Update Operations to DTD

Let Σ be a set of labels. For a node v in a tree t, by lv(v) we mean the label of v. The language specified by a regular expression r is denoted L(r). A DTD is a triple D = (d, sl, Σ), where d is a mapping from Σ to the set of regular expressions over Σ, and sl ∈ Σ is the start label. For a label a ∈ Σ, d(a) is the content model of a. A tree t is valid to D if lv(v) = sl for the root v of t and for any node v in t, lv(v)lv(v2)⋯lv(vn) ∈ L(lv(v)), where v1, v2, ⋯, vn are the child nodes of v.

Example 1 Consider the following DTD, where article is the start label.

<!ELEMENT article (title, section+)>
sert/delete operators (Fig. 2 shows an example). The first four operations in-
sert/delete/(un)nest labels and the rest two operations in-
sert and delete an operator at
position of the sibling subexpressions at
position u
label at position r
associated with its position. For a regular expression
tree structure of
of each node by Dewey order. For example, Fig. 2(a) shows

[A xml]
<!ELEMENT section (title, para+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT para (#PCDATA)>

Then the DTD is denoted (d,article,Σ), where d(article) =
title section, d(section) = title para, d(title) = d(para) = ϵ, and
Σ = {article, title, section, para}.

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(a) shows
the tree structure of (ab(c)a) and each node in the tree is associated with its position. For a regular expression r, the label at position u in r is denoted label(r, u) and the subexpression at position u of r is denoted sub(r, u). For example, consider the regular expression r = (ab)(ca) as follows: for each
node v in matches pat if there is a path going to v that matches pat.

A hedge is a finite sequence of trees. The set of hedges is denoted by H. For a set Q, by H2(Q) we mean the set of
hedges such that leaf nodes can be labeled with elements from Q.
A tree transducer is a quadruple
Tr = (Q,Σ,q0,R), where Q is a finite set of states, q0 ∈ Q is the initial state, and
R is a finite set of rules of the form (q, pat) → h, where q ∈ Q, pat is a pattern, and h ∈ H2(Q).

Let v be a node in a tree t. The translation defined by Tr
at v in state q, denoted by Tr(t,v), is inductively defined as follows.
R1: If there is a rule (q, pat) → h ∈ R such that v matches
pat, then Tr(t,v) is obtained from h as follows: for each
leaf node u in h, if (u) is a state, say p, then replace u
with a label u in t matches pat if there is a path going to v

[A xml]
<xsl:template match="/"/>
</xsl:template>

2.2 Classes UTT and UTT\textsuperscript{out}
A pattern is defined as pat = l_{s_1} \ldots l_{s_n}, where l_{s_i} is either
the restricted class is denoted UTT, which coincides with that

[A xml]
<xs: elemental name="a/b/c" name="q"/>
</xs:template>

We first define location paths used in select attributes. A
relative location path is defined as same as a pattern. That

2.3 Classes UTT\textsuperscript{Col} and UTT\textsuperscript{pat,sel}
We first define location paths used in select attributes. A
relative location path is defined as same as a pattern. That
is, it is of the form $ls_1/\cdots/ls_n$, where $ls_j$ is a label or ‘*’. An absolute location path consists of ‘*’ optionally followed by a relative location path. The set of relative location paths and absolute location paths is denoted by $SEL$. By $H_2(Q \times SEL)$ we mean the set of hedges such that leaf nodes can be labeled with elements from $Q \times SEL$.

A tree transducer in $UTT_{\text{out},sel}$ is also defined as a quadruple $(Q, \Sigma, q_0, R)$. The rule is extended as follows: for every rule $(q, pat) \rightarrow h$ in $R$, $h$ belongs to $H_2(Q \times SEL)$. The other syntactical definitions remain the same as defined in $UTT_{\text{out}}$. For example, $(q, a/b/c) \rightarrow c(p, l/d)$ corresponds to the following XSLT template rule.

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

For a relative location path $sel$, by $N(t, v, sel)$ we mean the set of nodes reachable from $v$ via $sel$ in $t$. Similarly, for an absolute location path $sel$, by $N(t, sel)$ 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.

- **If there is a rule $(q, pat) \rightarrow h$ in $R$ such that $v$ matches $pat$, then $Tr^q(t, v)$ is obtained from $h$ as follows.** For each leaf node $w$ in $h$ such that $w = (p, sl) \in Q \times SEL$,
  - if $sel$ is a relative location path, then replace $u$ with $Tr^q(t, v_1) \cdots Tr^q(t, v_n)$, where $N(t, v, sel) = \{v_1, \ldots, v_n\}$,
  - if $sel$ is an absolute location path, then replace $u$ with $Tr^q(t, v_1) \cdots Tr^q(t, v_n)$, where $N(t, sel) = \{v_1, \ldots, v_n\}$.

- **Otherwise, $Tr^q(t, v) = \varepsilon$.**

The translation of $t$ by $Tr$, denoted by $Tr(t)$, is defined as $Tr^q(t, v_0)$, where $v_0$ is the root node of $t$.

By $UTT_{\text{out}}$ we mean the subclass of $UTT_{\text{out},sel}$ such that $pat$ is a single label for any rule $(q, pat) \rightarrow h$.

### 3. Rules Affected by DTD Updates

In this section, we first introduce the notion of “correspondence” between elements of two DTDs. Based on the correspondence, we define rules affected by DTD updates.

#### 3.1 Correspondence between Elements

The same element name may be referenced multiple times and from multiple content model definitions, and we have to distinguish such elements when detecting the rules affected by DTD updates. By DTD updates. By $D^b_s$ we mean the element $a$ at position $u$ in $d(b)$. We say that $D^b_s$ is a superscripted label. If $a$ is the start label, then the corresponding superscripted label is $D'^{s,b}_e$. For a DTD $D = (d, sl, \Sigma)$, by $D_s = (d_s, sl^{out}, \Sigma)$ we mean the superscripted DTD of $D$ defined as follows.

- For each label $a \in \Sigma$, $D_s(a)$ is obtained by replacing each label in $d(a)$ with its superscripted label.
- $\Sigma_s$ is the set of superscripted labels.

**Example 2** Let $D = (d, article, \Sigma)$ be the DTD in Example 1. Then $D_s = (d_s, article^{out,1}, \Sigma, \Sigma_s)$, where

- $d_s(article) = title^{article1}(section^{article1,1}, par^{article1,2})^*$,
- $d_s(section) = title^{section1}(para^{section1,1}, par^{section1,2})^*$,
- $d_s(title) = \epsilon$,
- $d_s(para) = \epsilon$,

and

$$\Sigma_s = \{\text{article}^{out,1}, \text{section}^{article1,1}, \text{section}^{section1,2}, \text{para}^{section1,2}\}.$$  

For a tree $t$ valid to $D$, $t_s$ is a superscripted tree of $t$ if $t_s$ is obtained by replacing each label in $t$ with its superscripted label so that $t_s$ is valid to $D_s$ (see Fig. 3).

Let $D = (d, sl, \Sigma)$ be a DTD and $s$ be an update script to $D$. For a superscripted label $e'^{s,w}$ in $D_s$, if $e'^{s,w}$ is not deleted by $s$, then $e'^{s,w}$ also appears in $s(D)_s$ as $e''^{s,w}$ for some label $f'$ and some position $w'$.

We say that $e'^{s,w}$ corresponds to $e''^{s,w}$. As shown below, more than one superscripted label in $s(D)_s$ may correspond to $e''^{s,w}$. For a superscripted label $e''^{s,w}$ in $D_s$, by $C_p(e''^{s,w}, s)$ we mean the set of superscripted labels in $s(D)_s$ corresponding to $e''^{s,w}$ (the formal definition of $C_p(e''^{s,w}, s)$ is omitted because of space limitation).

---

2 If no update script between old and new DTDs is given, the algorithms in [6, 5] can generate update scripts between DTDs.
Here, illustrates to illustrate a tree transducer in UTG. To solve the problem, we firstly define a notation. Consider rule R1 in the definition of a node v with l(v) = d^a_b. This means that rule R1 holds for a node v in state q, in other words, state q is assigned to d^a_b and thus we have a node (d^a_b). Then consider the consequence of rule R1. Each state p in h is replaced by Tr^b(t_u,v) and Tr^a(t_u,v). Let c^a_b = (l(v)). Then Tr^a(t_u,v) means that state p is assigned to c^a_b, thus we obtain a node (c^a_b, p). Since (c^a_b, q) is obtained by (d^a_b, q), we denote this dependency by an edge (c^a_b, p) → (d^a_b, q). A dependency graph is a graph consisting of such nodes and edges.

4.1 Dependency Graph

To obtain R^b(d^a_b, d^a_b) and R^a(d^a_b, d^a_b), we have to find the rules applicable to each superscripted label. To do this, we define dependency graph. In short, a node in a dependency graph is a pair (d^a_b, q), which means that state q is assigned to d^a_b. Consider rule R1 in the definition of tree transducer (Section 2.2), and suppose that the antecedent of rule R1 holds for a node v with l(v) = d^a_b. This means that rule (q,a) → h is applied to v in state q, in other words, state q is assigned to d^a_b and thus we have a node (d^a_b). Then consider the consequence of rule R1. Each state p in h is replaced by Tr^b(t_u,v) and Tr^a(t_u,v). Let c^a_b = (l(v)). Then Tr^a(t_u,v) means that state p is assigned to c^a_b, thus we obtain a node (c^a_b, p). Since (c^a_b, q) is obtained by (d^a_b, q), we denote this dependency by an edge (c^a_b, p) → (d^a_b, q). A dependency graph is a graph consisting of such nodes and edges.

Example 4 Let D_k = (d^a_b, a^c^d, Σ, h) be a superscripted DTD, where d^a_b = b^a_q, d^a_b = c^a_q, d^a_b = e^a_q, c^a_q = d^a_q = e^a_q = h. Let Tr = (Q, Σ, q_0, R) be a tree transducer, where Q = {p,q}, Σ = {a,b,c,e}, and R = (q,a) → (a,q),(q,c) → (c,q),(b) → (b,q). Since the start label is a^c^d and the initial state is q, we obtain a superscripted DTD. Since d^a_b = b^a_q, c^a_q is a node that can be applied to a^c^d in state q, we obtain nodes (b^q,c^q) and edges (b^q,c^q) → (a^c^d) and (c^q,a^c^d) → (a^c^d, q) and (c^q, a^c^d) → (a^c^d, q). By applying rules in R similarly, we obtain the dependency graph in Fig. 5.

To define the dependency graph formally, we need to define the notion of “dependency”. Let D = (d,a,Σ) be a DTD, Tr = (Q,Σ,q_0,R) be a tree transducer, and rl = (q,a) → h. By S(h) we mean the set of states in h. For example, if h = a(p), then S(h) = {p,q}. Let

p' = (a^c^d,q)_0 = (a^c^d,q)_1 = ··· = (a^c^d,q)_n

be a path, where a_0 = sl. We say that an edge (d^a_b, q) ← (c^a_q, p) depends on rl and p' under D_k if

- (d^a_b,q)_k = (d^a_b,q),
- c^a_q appears in d^a_b, and
- p ∈ S(h).

The dependency graph of D and Tr is a graph G_D = (V_D, E_D) satisfying the following conditions.

- (s^c^d,q), V_D, (s^c^d,q), is called the root of G_D.
- If the following conditions hold, then (c^a_q, p) ∈ V_D and (d^a_b,q) ← (c^a_q, p) ∈ E_D.
  - For a rule rl = (q,a) → h ∈ R, G_D contains a path p' from (d^a_b,q) to the root such that (d^a_b,q) ← (c^a_q, p) depends on rl and p'.

4.2 Rules Affected by DTD Updates

Based on the above correspondence between superscripted labels, we define rules affected by DTD updates. Let D = (d,a,Σ) be a DTD, Tr = (Q,Σ,q_0,R) be a tree transducer. For a superscripted label d^a_b and a rule rl ∈ R, rl is applicable to d^a_b under D_k if for some tree t valid to D and some node v in t, (1) rl is applied to v (i.e., the antecedent of rule R1 holds for rl and v) during a translation of Tr(t), and (2) for some superscripted tree of r, l is labeled by d^a_b in t.

Let a^c^d be a superscripted label in D_k and d^a_b ∈ C_D(a^c^d). We define two sets of rules affected by s between a^c^d and d^a_b, denoted R^b(a^c^d, d^a_b) and R^a(a^c^d, d^a_b), as follows.

- R^b(a^c^d, d^a_b) is the set of rules rl ∈ R such that rl is not applicable to a^c^d under D_k but becomes applicable to d^a_b under S(D_k).
- R^a(a^c^d, d^a_b) is the set of rules rl ∈ R such that rl is applicable to a^c^d under D_k but not applicable to d^a_b under S(D_k).

In the subsequent sections, we consider solving the following problem, called affected rule detection problem.

- Input: DTD D = (d,a,Σ), update script s to D, tree transducer Tr = (Q,Σ,q_0,R), rule rl ∈ R, pair (a^c^d, d^a_b) of corresponding superscripted labels, and flag f ∈ {+,-}.

- Problem: Determine whether rl ∈ R^b(a^c^d, d^a_b) if f = +,' and determine whether rl ∈ R^a(a^c^d, d^a_b) if f = -.
• $G_D$ does not contain any other nodes and edges stated above.

We say that rule $rl = (q, a) \rightarrow h$ is applicable to a node $(a^{n+1}_{l-1}, q_n)$ in $G_D$ if $G_D$ contains a path $p'$ from $(a^{n+1}_{l-1}, q_n)$ to the root such that $a_0 = a$ and that $q_l = q$. The following lemma shows the important property of dependency graph.

**Lemma 1** Let $D = (d, s, \Sigma)$ be a DTD, $Tr = (Q, \Sigma, q_0, R)$ be a tree transducer, $G_D = (V_D, E_D)$ be the dependency graph of $D$ and $Tr$, and $rl = (q, a) \rightarrow h \in R$ be a rule. Then $rl$ is applicable to $(a^{n+1}, q)$ in $G_D$ iff $rl$ is applicable to $(a^{n}, q)$ in $G_D$.

**Proof:** Assume that $rl$ is applicable to $(a^{n}, q)$ in $G_D$. Then $G_D$ contains a path $p = (q_0, q_1, \ldots, q_n)$ such that $q_0$ is the root with $l(q_0) = a_0$ and that $l(q_i) = a^{n+1}_i$ for $1 \leq i \leq n$, and that

there is a sequence $r_0, r_1, \ldots, r_{n-1} = r_l$ of rules such that

$q_i \in S(t_r(r_{i-1}))$ for $1 \leq i \leq n$, and

$q_l = (q, a) \rightarrow h$.

Since $rl = (q, a) \rightarrow h$, $q_0 = q$, and $a_0 = a$, $rl$ is applied to $v_0$ with $l(v_0) = a^{n+1}$ during a translation of $Tr$.

Assume that $rl$ is applicable to $(a^{n+1}, q)$ in $G_D$. Then $G_D$ contains a superscripted tree $t_\ell$ such that $t_\ell$ contains a node $v$ with $l(v) = a^{n+1}$ and that $rl$ is applied to $v$ during a translation of $Tr$. Then we have the following.

a) $t_\ell$ contains path

$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_n$

such that $v_0$ is the root and that $v_n = v$ with $l(v) = a^{n+1}$. Moreover, let $l(v_2) = a^{n+1}_2$ and $l(v) = a^{n+1}$ for $1 \leq i \leq n$. Then we have that $d_\ell(d_{t_\ell}(v_1))$ contains $a^{n+1}_1$ for $1 \leq i \leq n$.

b) There is a sequence $r_0, r_1, \ldots, r_{n} = r_l$ of rules such that

$q_i \in S(t_r(r_{i-1}))$ for $1 \leq i \leq n$, and

$q_l = (q, a) \rightarrow h$.

By a) and b) above, we can inductively show that $G_D$ contains an edge $(a^{n+1}_0, q_0) \rightarrow (a^{n+1}_1, q_1)$ and $(a^{n+1}_n, q_n) \rightarrow (a^{n+1}_n, q_{n+1})$ for $1 \leq i \leq n - 1$. Thus $G_D$ contains a path

$\ell' = (a^{n+1}_0, q_0) \rightarrow (a^{n+1}_1, q_1) \rightarrow (a^{n+1}_2, q_2) \rightarrow (a^{n+1}_n, q_n),$

where $(a^{n+1}_0, q_0)$ is the root and $(a^{n+1}_n, q_n) = (a^{n+1}, q)$. Hence $rl$ is applicable to $(a^{n+1}, q)$ in $G_D$. \qed

### 4.2 Constructing Dependency Graph

We present algorithm CONSTDG for constructing a dependency graph $G_D = (V_D, E_D)$ (Fig. 6). In lines 3 to 11, the following is repeated until $E_D$ converges.

- For each node $(a^{n}, q)$ obtained in the previous iteration and each rule $rl = (q, a) \rightarrow h$ applicable to $(a^{n}, q)$ in $G_D$, the nodes and the edges obtained by applying $rl$ to $(a^{n}, q)$ are added to $V_D$ and $E_D$, respectively.

We show the correctness of the algorithm. Firstly, the following lemma holds immediately.

**Lemma 2** Let $G_D$ be the dependency graph of $D$ and $Tr$, and let

$\ell' = (a^{n+1}_0, q_0) \rightarrow (a^{n+1}_1, q_1) \rightarrow (a^{n+1}_2, q_2) \rightarrow (a^{n+1}_n, q_n)$

be a path in $G_D$. Then there is a simple (i.e., containing no cycle) path $\ell''$ from $(a^{n-1+1}_{l-1}, q_{l-1})$ to $(a^{n+1}_0, q_0)$.

### Input

| Input: DTD $D = (d, s, \Sigma)$, tree transducer $Tr = (Q, \Sigma, q_0, R)$ in UTT. |

### Output

| Output: Dependency graph $G_D = (V_D, E_D)$. |

1. Let $D_T = (d_T, s^{\text{root}}, \Sigma, \Sigma)$ be the superscripted DTD of $D$.
2. $V_D \leftarrow \{(s^{\text{root}}, q_0)\}; E_D \leftarrow \emptyset; V_{old} \leftarrow \emptyset; V_{old} \leftarrow \emptyset$
3. do
   4. $E_{old} \leftarrow E_D; V_{old} \leftarrow V_D$
   5. for each $(a^{n+1}, q) \in V_D \setminus V_{old}$ do
      6. for each rule $rl = (q, a) \rightarrow h \in R$ do
          7. if $rl$ is applicable to $(a^{n+1}, q)$ in $G_D = (V_D, E_D)$ then
              8. for each superscripted label $e^{\ell'}$ appearing in $d_\ell(a)$ and each state $p \in S(t_r(h))$
                  9. $E_D \leftarrow E_D \cup \{(a^{n+1}, q, (e^{\ell'}, p))\}$
              10. $V_D \leftarrow V_D \cup \{(e^{\ell'}, p)\}$
            11. while $E_D \neq E_{old}$
            12. return $G_D = (V_D, E_D)$

**Figure 6: Algorithm CONSTDG**

**Proof:** $p''$ can be obtained by bypassing every cycle in $p'$. \qed

The following lemma shows the correctness of the algorithm.

**Lemma 3** Let $G_D$ be the result of CONSTDG for DTD $D$ and tree transducer $Tr$. Then $G_D$ is the dependency graph of $D$ and $Tr$.

**Proof:** Let $H_D$ be the dependency graph of $D$ and $Tr$. Since line 7 applies a rule $rl$ applicable to $(a^{n+1}, q)$, it is clear that $G_D$ is a subgraph of $H_D$.

Consider the opposite direction and let us show that every node and every edge in $H_D$ is in $G_D$. For a path $\ell'$ from $(a^{n+1}, q)$ to the root in $H_D$ and a rule $rl$, if an edge $e = (a^{n+1}, q) \rightarrow (e^{\ell'}, p)$ depends on $rl$ and $\ell'$ is a simple path, then we say that $\ell'$ is a simple dependency path of $e$ and $rl$. We show that every edge in $H_D$ is in $G_D$ by induction on the length of the simple dependency path of the edge. Then this lemma holds by Lemma 2.

**Basis:** In this case, the length of the simple dependency path of an edge and a rule is zero. It is clear that every edge incident to the root of $H_D$ is obtained by CONSTDG.

**Induction:** Assume as the induction hypothesis that if the length of the simple dependency path of an edge and a rule in $H_D$ is less than $k$, then the edge is obtained by CONSTDG. Consider an edge $e = (a^{n+1}, q) \rightarrow (e^{\ell'}, p)$ and a rule $rl$ whose simple dependency path is of length $k$, and let $\ell'$ be the simple dependency path. Let

$e'' = (a^{n+1}, q) \rightarrow (e^{\ell'}, p) \rightarrow (a^{n+1}, q),$

where $(a^{n+1}, q) = (a^{n+1}, q)$. Since the simple dependency path $e''$ is of length $k - 1$, $e''$ is obtained in $G_D$. Thus $e'$ is also obtained in $G_D$ by CONSTDG. Therefore, $e'$ is in $G_D$, and thus $e$ is obtained in lines 7 to 10 of CONSTDG. \qed

Consider the time complexity of CONSTDG. The numbers of nodes of a dependency graph is in $O(|D||\Sigma|)$, where $|D|$ is the sum of the sizes of content models of $D$. The total number of nodes examined in line 5 over the iterations of the while
Input: DTD $D = \langle d, s, \Sigma \rangle$, update script $s$ to $D$, tree transducer $Tr = \langle Q, \Sigma, q_0, R \rangle$ in UTI.
Output: $R' \langle a^{\alpha}, a^{\beta} \rangle$ for every pair $\langle a^{\alpha}, a^{\beta} \rangle$ of superscripted labels such that $a^{\alpha} \neq C_0(a^{\beta}, s)$.

1. $G_0 \leftarrow \text{CONST\!DG}(D, Tr)$.
2. $G'_0 \leftarrow \text{CONST\!DG}(s(D), Tr)$.
3. Compute $C_0(a^{\alpha}, s)$ for each $a^{\alpha}$ in $D_a$.
4. For each pair $\langle a^{\alpha}, a^{\beta} \rangle$ such that $a^{\alpha} \in C_0(a^{\beta}, s)$ do
   5. $M \leftarrow \{rl \in R | rl \text{ is applicable to } \langle a^{\alpha}, q \rangle \text{ in } G_0 \}$
   6. $M' \leftarrow \{rl \in R | rl \text{ is applicable to } \langle a^{\beta}, q \rangle \text{ in } G'_0 \}$
   7. $R' \langle a^{\alpha}, a^{\beta} \rangle \leftarrow M \cup M'$
8. Return $\{R' \langle a^{\alpha}, a^{\beta} \rangle | a^{\alpha} \in C_0(a^{\beta}, s)\}$

Figure 7: Algorithm FIND\!AFFECTED\!RULES

loop is in $O(|D(Q)|)$. For each node $\langle a^{\alpha}, q \rangle$ selected in line 5 and a rule $rl$ selected in line 6, $O(|D(Q)|)$ edges are added to $E_D$. Hence $\text{CONST\!DG}$ runs in

$$O(|D(Q)| \cdot |D(Q)| \cdot |R| \cdot |D(Q)|) = O(|R||D(Q)|^3).$$

(1)

4.3 Detecting Rules Affected by DTD Updates

Algorithm FIND\!AFFECTED\!RULES (Fig. 7) computes $R' \langle a^{\alpha}, a^{\beta} \rangle$ for every pair $\langle a^{\alpha}, a^{\beta} \rangle$ of corresponding superscripted labels $a^{\alpha} \neq C_0(a^{\beta}, s)$ can be obtained similarly. This algorithm constructs dependency graphs $G_D$ and $G'_D$ for old/new DTDs, and then calculates the diff between $G_D$ and $G'_D$ to obtain $R' \langle a^{\alpha}, a^{\beta} \rangle$. The following theorem immediately follows from Lemmas 1 and 3.

**Theorem 1** FIND\!AFFECTED\!RULES correctly computes $R' \langle a^{\alpha}, a^{\beta} \rangle$ for any pair $\langle a^{\alpha}, a^{\beta} \rangle$ such that $a^{\alpha} \neq C_0(a^{\beta}, s)$.

Consider the running time of FIND\!AFFECTED\!RULES. Let $D_{\max} = \max_{s \subseteq a} \{s(D)\}$, where $s = op_1op_2 \cdots op_p$, and $s_i = op_iop_i \cdots op_p$. Lines 1 and 2 require $O(R(|D_{\max}||Q)|)$ by (1).

In line 3, $C_0(a^{\alpha}, s)$ can be obtained by computing $C_0(a^{\alpha}, s)$ for $i = 1, 2, \ldots, n$, which can be done in $O(s(D_{\max}))$ time. Since the for loop in line 4 iterates $O(|D_{\max}^f|)$ times and lines 5 to 6 can be done in $O(|R|)$ time, lines 4 to 7 requires $O(|R||D_{\max}^f|)$. Thus FIND\!AFFECTED\!RULES runs in

$$O(R||D_{\max}||Q|^3 + s||D_{\max}^f||Q) + |R||D_{\max}^f|^3 = O(|R||D_{\max}||Q|^3 + |D_{\max}^f|^3).$$

5. Intractability of the Problem

In the previous section, the affected rule detection problem is shown to be in PTIME for UTI. In this section, we show that the problem is intractable for UTI and UTI under restricted DTDs.

5.1 Intractability of UTI

We first consider the intractability of the problem for UTI. To show the intractability, we consider the following simpler problem, called rule applicability problem.

- Input: DTD $D$, superscripted label $a^{\alpha}$ in $D_a$, tree transducer $Tr = \langle Q, \Sigma, q_0, R \rangle$, and a rule $rl \in R$.
- Problem: Determine whether $rl$ is applicable to $a^{\alpha}$ under $D_a$.

As we will see later, the rule applicability problem can be easily reduced to the affected rule detection problem.

We consider two kinds of restricted DTDs: flat and duplicate-free DTDs. Let us first consider flat DTDs. We say that a DTD $D = \langle d, s, \Sigma \rangle$ is flat if for any label $a$ appearing in $d(s), d(a) = e$. We first show that the rule applicability problem is intractable for UTI under such simple DTDs.

**Lemma 4** The rule applicability problem for UTI is NP-hard under flat DTDs.

**Proof:** We show this lemma by reducing the 3SAT problem to the rule applicability problem. Here, the 3SAT problem is described as follows.

- Input: Boolean formula $\phi = C_1 \land C_2 \land \cdots \land C_n$, where $C_i$ is a clause with three literals. Let $x_1, x_2, \ldots, x_n$ be the variables appearing in $\phi$.
- Problem: Determine whether there is a truth assignment of $x_1, x_2, \ldots, x_n$ such that $\phi = true$.

Firstly, DTD $D = \langle d, s, \Sigma \rangle$ is constructed from $\phi$ as follows.

$$d(s) = (T_1(F_1) \cdots T_m(F_m),$$

$$d(c_i) = \epsilon \quad (1 \leq i \leq n),$$

and

$$\Sigma = \{s \mid (c_1, c_2, \ldots, c_n) \cup (c'_1, c'_2, \ldots, c'_{m+1})\},$$

where,

- $c_i$ is a label representing clause $C_i$,
- $c'_j$ is a label used in tree transducer $Tr$ defined below,
- $T_i (1 \leq i \leq m)$ lists "the labels representing clauses which contain positive literal $x_i$",
- $F_i (1 \leq i \leq m)$ lists "the labels representing clauses which contain negative literal $\neg x_i$".

For example, if $\phi = C_1 \land C_2 \land C_3$, $C_1 = x_1 \lor \neg x_2 \lor x_3$, $C_2 = x_2 \lor x_3 \lor x_4$, $C_3 = \neg x_1 \lor \neg x_2 \lor x_3$, $C_4 = x_1 \lor x_2 \lor \neg x_4$, then $T_1 = c_1c_4$, $F_1 = c_3$, $T_2 = c_2c_4$, $F_2 = c_1c_3$, and so on.

Secondly, we define a tree transducer $Tr = \langle Q, \Sigma, q_0, R \rangle$ as follows.

$$Q = \{q_0, q_1, \ldots, q_{n+1}\},$$

$$R = \{(q_0, s) \rightarrow (q_1, f(s/c_1), q_1, c_1) \rightarrow (q_2, f(s/c_2), q_2, c_2) \rightarrow (q_3, f(s/c_3), q_3, c_3) \rightarrow \cdots \rightarrow (q_{n+1}, q_{n+1}) \rightarrow (q_{n+1}, f(s/c_n), q_{n+1}, c_{n+1}) \rightarrow \cdots \}.$$
Then it is obvious that the truth assignment makes $\phi = true$.

($\Leftarrow$) Assume that $\phi$ is satisfiable. Then, there is a truth assignment of $x_1, x_2, \ldots, x_n$ which makes $\phi = true$. With this assignment, consider a tree $t$ constructed as follows.

- For every disjunction $T_{i}[F_i]$ in $d(s)$, if $x_i = true$, then $T_i$ is selected, and if $x_i = false$, then $F_i$ is selected.
- Since $\phi = true$ (i.e., all $C_i$s are true) with this assignment, $t$ has all of $c_1, c_2, \ldots, c_n$ as child elements of $s$. Thus it is easy to show that $(q_{n+1}, s) \rightarrow c'_{n+1}$ is applicable to $s^{root}$ under $D_s$. □

By using this lemma we show the NP-harness of the affected rule detection problem.

**Theorem 2** The affected rule detection problem for UTT$^{aw}$ is NP-hard even if $D$ is flat and $|s| = 1$.

**Proof (sketch):** The rule applicability problem can easily be reduced to the affected rule detection problem. Let $s$ be a nest$_{elm}$ operation that inserts a new label $l$ (unused in $D$) at some ancestor of $d^{aw}$ in $D_s$, and let $d^{aw}$ be a superscripted label in $s(D_s)$ corresponding to $d^{aw}$. Since $l$ is a new label, no rule is applicable to $d^{aw}$ under $s(D_s)$. Thus $rl$ is applicable to $d^{aw}$ under $D_s$ iff $rl \in R^{(d^{aw}, d^{aw})}$.

Let us next consider another restricted DTD, called duplicate-free DTD. Let $r$ be a regular expression and $\Sigma(r)$ be the set of labels appearing in $r$. Then $r$ is duplicate free if each label in $\Sigma(r)$ occurs exactly once in $r$. A DTD $D$ is duplicate free if for each content model $d(a)$ of $D$, $d(a)$ is duplicate free. It is shown that the complexity of the XPath satisfiability problem becomes lower under duplicate-free DTDs[9, 12]. On the other hand, the duplicate-freeness does not reduce the complexity of our problem, as shown below.

**Theorem 3** The affected rule detection problem for UTT$^{aw}$ is NP-hard under duplicate-free DTDs.

**Proof:** Again we reduce the 3SAT problem to the rule applicability problem. Without loss of generality, we assume that for any variable $x_i$, its positive literal $x_i$ and its negative literal $\sim x_i$ do not appear in the same clause.

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

$$d(s) = X_1X_2\cdots X_n$$

$$d(X_i) = T|F_i, (1 \leq i \leq m)$$

$$d(c_i) = \epsilon, (1 \leq i \leq n)$$

and

$$\Sigma = \{s\} \cup \{X_1, X_2, \cdots, X_m\} \cup \{c_1, c_2, \cdots, c_n\} \cup \{c'_1, c'_2, \cdots, c'_{n+1}\}$$

where $T_i$ and $F_i$ are defined as same as Lemma 4. Note that $D$ is duplicate free since any $c_i$ does not appear in both $T_i$ and $F_i$ by the above assumption.

Secondly, $Tr = (Q, \Sigma, q_0, R)$ is defined as follows.

$$Q = \{(q_0, s) \rightarrow s(q_1), (q_1, s) \rightarrow s(q_2), \cdots, (q_{n-1}, s) \rightarrow s(q_n)\}$$

$$R = \{(q_0, s) \rightarrow s(q_1), (q_1, s) \rightarrow s(q_2), \cdots, (q_{n-1}, s) \rightarrow s(q_n)\}$$

Similarly, to Lemma 4, we can show that the last rule $(q_{n+1}, s) \rightarrow c'_{n+1}$ in $R$ is applicable to $s^{root}$ under $D_s$ if and only if $\phi$ is satisfiable. Thus the rule applicability problem is NP-hard under duplicate-free DTDs.

By the above result and the reduction of Theorem 2, the affected rule detection problem becomes intractable for UTT$^{aw}$.

**Theorem 4** The affected rule detection problem for UTT$^{aw}$ is NP-hard under duplicate-free DTDs.

**Proof:** We reduce the 3DNF non-tautology problem to the rule applicability problem. The 3DNF non-tautology problem is described as follows.

- Input: Boolean formula $\phi = c_1 \lor c_2 \lor \cdots \lor c_s$, where $C_i$ is a conjunctive clause with three literals. Let $x_1, x_2, \cdots, x_n$ be the variables appearing in $\phi$.
- Problem: Determine whether $\phi$ is a non-tautology, that is, there is a truth assignment of $x_1, x_2, \cdots, x_n$ such that $\phi = false$.

Since the 3DNF non-tautology problem is equivalent to the 3SAT problem, 3DNF non-tautology problem is NP-complete. Firstly, DTD $D = (d, s, \Sigma)$ is defined as follows.

$$d(s) = T_i|F_1$$

$$d(T_1) = d(F_1) = T_2|F_2$$

$$d(T_2) = d(F_2) = T_3|F_3$$

$$\vdots$$

$$d(T_{m-1}) = d(F_{m-1}) = T_m|F_m$$

$$d(T_m) = d(F_m) = b_r$$

$$d(b) = c$$

$$d(c) = \epsilon$$

and

$$\Sigma = \{s\} \cup \{T_1, T_2, \cdots, T_m\} \cup \{F_1, F_2, \cdots, F_m\} \cup \{c_1, c_2, \cdots, c_{n+1}\}$$

Here, $T_i$ stands for "$x_i$ is true" and $F_i$ stands for "$x_i$ is false". Thus each tree valid to $D$ is representing a truth assignment for $\phi$.

Secondly, we define a tree transducer $Tr = (Q, \Sigma, q_0, R)$ as follows.

$$Q = \{(q_0, q_1, \cdots, q_{m+1})\}$$

$$R = R_1 \cup R_2 \cup R'_2$$

where

$$R_1 = \{(q_0, s) \rightarrow s(q_1), (q_1, s) \rightarrow s(q_2), \cdots, (q_{m-1}, s) \rightarrow s(q_m)\}$$

$$R_2 = \{(q_0, T_m) \rightarrow c_w(q_{m+1}), (q_m, F_m) \rightarrow c_w(q_{m+1})\}$$

$$\{q_{m+1}, b) \rightarrow (b(q_{m+2}),)\}$$
Here, the rules in $R_1$ are applied to the nodes from the root to the $b$ node (the parent of the leaf $c$), and either a rule in $R_1$ or the rule in $R_2'$ is applied to the leaf $c$. And $pat_i$ used in $R_2$ is a pattern defined as follows. Let $C_i = (l_{i1} \land l_{i2} \land \ldots \land l_{in})$ be the $i$-th conjunctive clause of $\phi$, and let $x_{ij}$ be the variable of literal $l_{ij}$ ($1 \leq j \leq 3$). Without loss of generality, we assume that $i < i_1 < i_2$. Then $pat_i$ is defined as follows.

$$pat_i = s_i \cdot \ldots \cdot s_1 \cdot /L_{i1}/ /s_i \cdot \ldots \cdot s_1 \cdot /L_{i2}/ /s_i \cdot \ldots \cdot s_1 \cdot /L_{i3}/ /s_i \cdot \ldots \cdot s_1 \cdot /b/c, \quad (1 \leq j \leq 3)$$

where

$$L_{ij} = \begin{cases} T_i, & \text{if } l_{ij} = x_{ij}, \\ F_i, & \text{if } l_{ij} = \neg x_{ij}. \end{cases} \quad (1 \leq j \leq 3)$$

We show that $\phi$ is a non-tautology iff the rule $(q_{m+2}, C) \rightarrow c$ of $R_2'$ is applied to $c$. 

$(\Rightarrow)$ Assume that $\phi$ is a non-tautology. Then there is a true assignment of $x_1, x_2, \ldots, x_m$ such that $\phi = false$. Let $t$ be the tree representing this truth assignment. Since $C_i = false$ for every $1 \leq i \leq n$, none of $pat_1, pat_2, \ldots, pat_n$ matches the $c$ node in $t$, thus no rule in $R_2$ is applied to the $c$ node. Therefore, the rule in $R_2'$ is applied to the $c$ node.

$(\Leftarrow)$ Assume that $\phi$ is a tautology. Then for any truth assignment of $x_1, x_2, \ldots, x_m$, $\phi = true$. Therefore, for any truth assignment of $x_1, x_2, \ldots, x_m$, $C_i = true$ for some $i$. Thus, for any tree $t$ valid to $D$, some rule in $R_2$ is applied to the $c$ node in $t$ (since $|pat_i| \geq 2$ for any $i$, the rule in $R_2'$ is not applied to $c$).

By the above result and Theorem 2, the affected rule detection problem is NP-hard under duplicate-free DTDs. □

6. Conclusion

In this paper, we consider the problem of detecting XSLT rules affected by DTD updates. We firstly propose an algorithm for detecting XSLT rules affected by DTD updates, assuming that UTT as XSLT. Then we showed several cases in which the problem becomes intractable.

In this paper, we considered three classes of tree transducers. However, some functions of XSLT, e.g., conditional branch, are missing in these classes. As a future work, we would like to consider more powerful classes of tree transducers. We also have to consider extending our algorithm so that the algorithm can handle more powerful schema languages such as XML Schema and RELAX NG.

[Acknowledgments]

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

[Bibliography]


---

Yang WU received B.E. degree in software from Dalian Jiaotong University in 2013. Currently, she is a master course student of Graduate School of Library, Information and Media Studies, University of Tsukuba. Her current research interests includes XML document management and tree transducers.

Nobutaka SUZUKI received his B.E. degree in information and computer sciences from Osaka University in 1995, 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. Currently, he is an Associate Professor of Faculty of Library, Information and Media Science, University of Tsukuba. His current research interests include database theory and structured documents.