Regular Paper

DBSJ Journal
Vol. 16, Article No. 1 March 2018

Detecting XSLT Rules Affected by DTD Updates[1]

Yang WU[♡] Nobutaka SUZUKI[◇]

Schemas of XML documents may be updated for various reasons. 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 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 old/updated schemas. 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^{pat}, and UTT^{sel}, 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^{pat} and UTT^{sel} are extensions of UTT, where *pat* 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^{sel} and UTT^{pat}, 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 S_{in}, S_{out}, whether the tree obtained by transforming t by Tr is valid to S_{out} for any tree t valid to S_{in}. 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 $l(v)$ we mean the label of v. The language specified by a regular expression r is denoted $L(r)$. A DTD is a triple $D = (d, sl, \Sigma)$, 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* to D if $l(v) = sl$ for the root v of t and for any node n in t, $l(v_1)l(v_2)\cdots l(v_n) \in L(d(l(v)))$, where $v_1, v_2, \cdots, v_n$ are the child nodes of v.

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

```
<!ELEMENT article (title, section+)>
```

[1]This paper is partly based on [13, 14].

♡ Student Member Graduate School of Library, Information and Media Studies, University of Tsukuba
wuyang65432@yahoo.co.jp

◇ Member Faculty of Library, Information and Media Science, University of Tsukuba
nsuzuki@slis.tsukuba.ac.jp

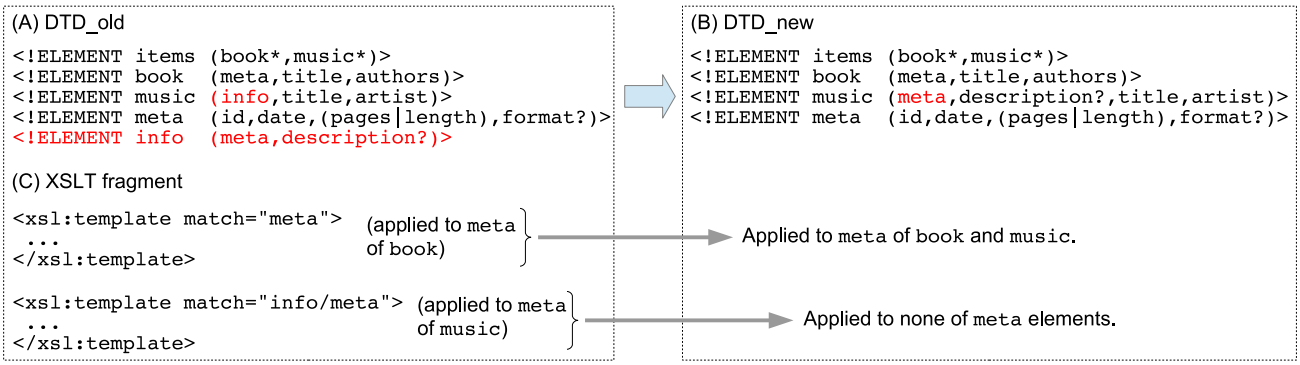

Figure 1: Old/new DTDs and an XSLT fragment

```
<!ELEMENT section (title, para+)>
<!ELEMENT title   (#PCDATA)>
<!ELEMENT para    (#PCDATA)>
```

Then the DTD is denoted $(d, \text{article}, \Sigma)$*, where* $d(\text{article}) = \text{title section}^+$, $d(\text{section}) = \text{title para}^+$, $d(\text{title}) = d(\text{para}) = \epsilon$*, and* $\Sigma = \{\text{article}, \text{title}, \text{section}, \text{para}\}$.

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 $(a|b)(ca)^*$ 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 $l(r, u)$ and the subexpression at position u of r is denoted $sub(r, u)$. For example, consider the regular expression $r = (a|b)(ca)^*$ in Fig. 2(a). Then $l(r, 1) =$ '|', $l(r, 1.1) = a$, $sub(r, 2.1) = ca$.

Let $D = (d, sl, \Sigma)$ be a DTD. Following [11], we define six *update operations* to D. The first four operations insert/delete/(un)nest labels and the rest two operations insert/delete operators (Fig. 2 shows an example).

- $ins\_elm(a, b, u)$: 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 $d(l')$.
- $ins\_opr(a, opr, u.i, u.j)$: inserts an operator opr as the parent of the sibling subexpressions at $u.i, u.i + 1, \cdots, u.j$ in $d(a)$, where $opr \in \{|, \cdot, *, +, ?\}$. We assume that opr has exactly one operand (child) whenever $opr \in \{*, +, ?\}$. $ins\_opr(a, opr, u.i, u.j)$ is applicable to $d(a)$ only if (i) $i = j$ (opr has only one operand) or (ii) $i < j$ and $opr \in \{|, \cdot\}$.
- $del\_opr(a, u)$: deletes an operator at u in $d(a)$. This is applicable to $d(a)$ only if the operator at u has only one operand whenever $l(d(a), v) \in \{*, +, ?\}$, where v is the parent position of u.

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 $s(D) = op_n(\cdots(op_2(op_1(D))))$.

It is easy to see that for any pair (D, D') of DTDs, there is an update script s such that $s(D)$ is equivalent to D'.

2.2 Classes UTT and UTT^{pat}

A *pattern* is defined as $pat = ls_1/\cdots/ls_n$, where ls_i is either a label $l_i \in \Sigma$ or a wildcard '$*$'. By $|pat|$, we mean the length of pat, that is, $|pat| = n$. Let t be a tree. We say that a path $v_1 \to v_2 \to \cdots \to v_m$ in t *matches* pat if $m \geq n$ and for any $1 \leq i \leq n$ $l(v_{m-n+i}) = ls_i$ whenever ls_i is a label. We say that a node v in t *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_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 $Tr = (Q, \Sigma, q_0, R)$, 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 rule.

```
<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 Tr 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 translation 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 $(q, pat) \to h$ pat is a single label, then the restricted class is denoted UTT, which coincides with that of the standard unranked tree transducer [7].

In R1, there may be more than one rule whose pattern is matched by v. Following the XSLT specification [1], we assume that if v matches both pat and a for two rules $(q, pat) \to h$ and $(q, a) \to h'$, then $(q, pat) \to h$ is applied to v whenever $|pat| \geq 2$.

2.3 Classes UTT^{sel} and UTT^{pat,sel}

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

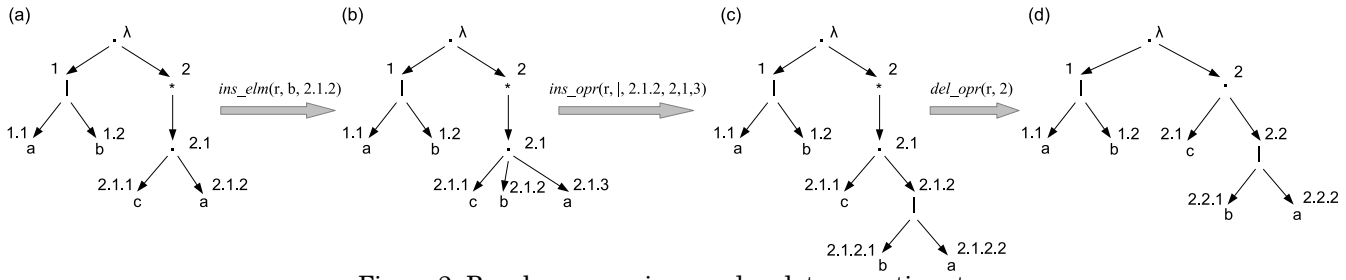

Figure 2: Regular expression r and update operations to r

is, it is of the form $ls_1/\cdots/ls_n$, where ls_i 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 $H_\Sigma(Q \times SEL)$ we mean the set of hedges such that leaf nodes can be labeled with elements from $Q \times SEL$.

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

```
<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 $(q, pat) \to h \in R$ such that v matches pat, then $Tr^q(t, v)$ is obtained from h as follows. For each leaf node u in h such that $l(u) = (p, sel) \in Q \times SEL$,
  - if sel is a relative location path, then replace u with hedge $Tr^p(t, v_1) \cdots Tr^p(t, v_n)$, where $N(t, v, sel) = \{v_1, \cdots, v_n\}$, or
  - if sel is an absolute location path, then replace u with hedge $Tr^p(t, v_1) \cdots Tr^p(t, v_n)$, where $N(t, sel) = \{v_1, \cdots, v_n\}$.
- Otherwise, $Tr^q(t, v) = \epsilon$.

The translation of t by Tr, denoted by $Tr(t)$, is defined as $Tr^{q0}(t, v_0)$, where v_0 is the root node of t.

By UTT^{sel} we mean the subclass of UTT^{pat,sel} such that pat is a single label for any rule $(q, pat) \to h$.

3. Rules Affected by DTD Updates

In this section, we firstly 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 $a^{b,u}$ we mean the element a at position u in $d(b)$. We say that $a^{b,u}$ is a *superscripted label*. If a is the start label, then the corresponding superscripted label is $a^{root,\lambda}$. For a DTD $D = (d, sl, \Sigma)$, by $D_\# = (d_\#, sl^{root,\lambda}, \Sigma, \Sigma_\#)$ we mean the *superscripted DTD* of D defined as follows.

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

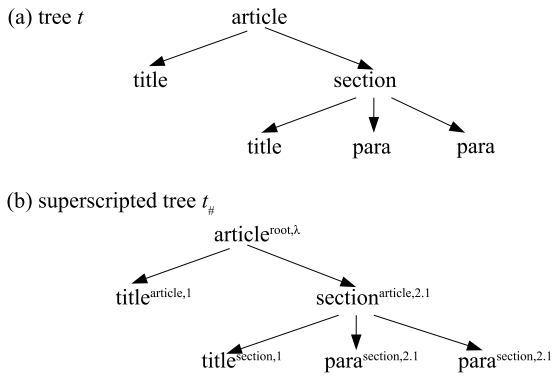

Figure 3: Tree t and its superscripted tree $t_\#$

Example 2 *Let $D = (d, \text{article}, \Sigma)$ be the DTD in Example 1. Then $D_\# = (d_\#, \text{article}^{root,\lambda}, \Sigma, \Sigma_\#)$, where*

$$\begin{aligned} d_\#(\text{article}) &= \text{title}^{\text{article},1}(\text{section}^{\text{article},2.1})^+, \\ d_\#(\text{section}) &= \text{title}^{\text{section},1}(\text{para}^{\text{section},2.1})^+, \\ d_\#(\text{title}) &= \epsilon, \\ d_\#(\text{para}) &= \epsilon, \end{aligned}$$

and

$$\begin{aligned} \Sigma_\# = \{&\text{article}^{root,\lambda}, \text{title}^{\text{article},1}, \text{section}^{\text{article},2.1}, \\ &\text{title}^{\text{section},1}, \text{para}^{\text{section},2.1}\}. \end{aligned}$$

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

Let $D = (d, sl, \Sigma)$ be a DTD and s be an update script to D. For a superscripted label $e^{f,w}$ in $D_\#$, if $e^{f,w}$ is not deleted by s, then $e^{f,w}$ also appears in $s(D)_\#$ as $e^{f',w'}$ for some label f' and some position w'.[2] We say that $e^{f',w'}$ *corresponds* to $e^{f,w}$ and that $e^{f,w}$ *corresponds* to $e^{f',w'}$. As shown below, more than one superscripted label in $s(D)_\#$ may correspond to $e^{f,w}$. For a superscripted label $e^{f,w}$ in $D_\#$, by $C_D(e^{f,w}, s)$ we mean the set of superscripted labels in $s(D)_\#$ corresponding to $e^{f,w}$ (the formal definition of $C_D(e^{f,w}, s)$ 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.

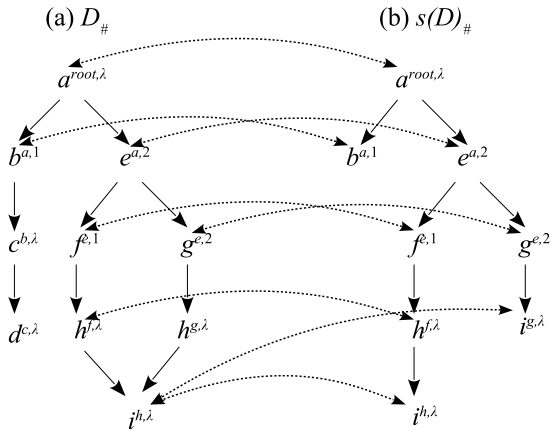

Figure 4: The correspondence between elements in $D_\#$ and $s(D)_\#$.

Example 3 *Let $D = (d, a, \Sigma)$ be a DTD, where $d(a) = be$, $d(b) = c$, $d(c) = d$, $d(e) = fg$, $d(f) = d(g) = h$, $d(h) = i$, $d(d) = d(i) = \epsilon$, and $\Sigma = \{a, b, c, d, e, f, g, h, i\}$. Figure 4(a) illustrates $D_\#$. Each solid edge in the figure represents a parent-child relationship between superscripted labels. Let $s = del\_elm(b, \lambda)unnest(g, \lambda)$. Then $s(D) = (d', a, \Sigma)$, where $d'(a) = be$, $d'(e) = fg$, $d'(f) = h$, $d'(g) = d'(h) = i$, $d'(b) = d'(i) = \epsilon$. Figure 4(b) illustrates $s(D)_\#$, and each dashed arc between two superscripted labels denotes the correspondence between the labels, e.g., $C_D(a^{root,\lambda}, s) = \{a^{root,\lambda}\}$, $C_D(b^{a,1}, s) = \{b^{a,1}\}$, and so on. Here, $C_D(c^{b,\lambda}, s) = C_D(d^{c,\lambda}, s) = \emptyset$ due to $del\_elm(b, \lambda)$. On the other hand, $C_D(i^{h,\lambda}, s)$ contains two labels due to $unnest(g, \lambda)$, i.e., $C_D(i^{h,\lambda}, s) = \{i^{h,\lambda}, i^{g,\lambda}\}$.*

3.2 Rules Affected by DTD Updates

Based on the above correspondence between superscripted labels, we define rules affected by DTD updates. Let $D = (d, sl, \Sigma)$ be a DTD and $Tr = (Q, \Sigma, q_0, R)$ be a tree transducer. For a superscripted label $a^{b,u}$ and a rule $rl \in R$, rl is *applicable* to $a^{b,u}$ under $D_\#$ 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 $t_\#$ of t, v is labeled by $a^{b,u}$ in $t_\#$.

Let $a^{b,u}$ be a superscripted label in $D_\#$ and $a^{b',u'} \in C_D(a^{b,u}, s)$. We define two sets of rules *affected* by s between $a^{b,u}$ and $a^{b',u'}$, denoted $R^+(a^{b,u}, a^{b',u'})$ and $R^-(a^{b,u}, a^{b',u'})$, as follows.

- $R^+(a^{b,u}, a^{b',u'})$ is the set of rules $rl \in R$ such that rl is not applicable to $a^{b,u}$ under $D_\#$ but becomes applicable to $a^{b',u'}$ under $s(D)_\#$.
- $R^-(a^{b,u}, a^{b',u'})$ is the set of rules $rl \in R$ such that rl is applicable to $a^{b,u}$ under $D_\#$ but not applicable to $a^{b',u'}$ under $s(D)_\#$.

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

- Input: DTD $D = (d, a, \Sigma)$, update script s to D, tree transducer $Tr = (Q, \Sigma, q_0, R)$, rule $rl \in R$, pair $(a^{b,u}, a^{b',u'})$ of corresponding superscripted labels, and flag $f \in \{+, -\}$.
- Problem: Determine whether $rl \in R^-(a^{b,u}, a^{b',u'})$ if $f =$ '–', and determine whether $rl \in R^+(a^{b,u}, a^{b',u'})$ if $f =$ '+'.

4. Detecting Rules Affected by DTD Updates

In this section, we consider solving the affected rule detection problem for UTT. To solve the problem, we firstly define "dependency graph" and propose an algorithm for constructing dependency graphs. Then we propose an algorithm for computing $R^+(a^{b,u}, a^{b',u'})$ and $R^-(a^{b,u}, a^{b',u'})$ by using the algorithm. Throughout this section, we assume that tree transducers are in UTT.

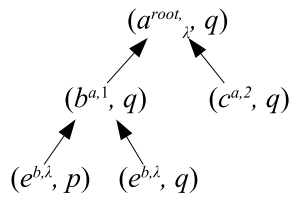

Figure 5: An example of dependency graph

4.1 Dependency Graph

To obtain $R^+(a^{b,u}, a^{b',u'})$ and $R^-(a^{b,u}, a^{b',u'})$, 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 $(a^{b,u}, q)$, which means that state q is assigned to $a^{b,u}$. 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 $l(v) = a^{b,u}$. This means that rule $(q, a) \to h$ is applied to v in state q, in other words, state q is assigned to $a^{b,u}$ and thus we have a node $(a^{b,u}, q)$. Then consider the consequence of rule R1. Each state p in h is replaced by $Tr^p(t, v_1) \cdots Tr^p(t, v_n)$. Let $c^{a,v} = l(v_i)$. Then $Tr^p(t, v_i)$ means that state p is assigned to $c^{a,v}$, thus we obtain a node $(c^{a,v}, p)$. Since $(c^{a,v}, p)$ is obtained by $(a^{b,u}, q)$, we denote this dependency by an edge $(c^{a,v}, p) \to (a^{b,u}, q)$. A dependency graph is a graph consisting of such nodes and edges.

Example 4 *Let $D_\# = (d_\#, a^{root,\lambda}, \Sigma, \Sigma_\#)$ be a superscripted DTD, where $d_\#(a) = b^{a,1}c^{a,2}$, $d_\#(b) = e^{b,\lambda}$, $d_\#(c) = d_\#(e) = \epsilon$. Let $Tr = (Q, \Sigma, q, R)$ be a tree transducer, where $Q = \{p, q\}$, $\Sigma = \{a, b, c, e\}$, and $R = \{(q, a) \to a(q), (q, c) \to c, (q, b) \to b(pq)\}$. Since the start label is $a^{root,\lambda}$ and the initial state is q, we obtain $(a^{root,\lambda}, q)$. Since $d_\#(a) = b^{a,1}c^{a,2}$ and $(q, a) \to a(q)$ can be applied to $a^{root,\lambda}$ in state q, we obtain nodes $(b^{a,1}, q)$, $(c^{a,2}, q)$ and edges $(b^{a,1}, q) \to (a^{root,\lambda}, q)$ and $(c^{a,2}, q) \to (a^{root,\lambda}, q)$. 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 $D = (d, sl, \Sigma)$ be a DTD, $Tr = (Q, \Sigma, q_0, R)$ be a tree transducer, and $rl = (q, a) \to h \in R$. By $St(h)$ we mean the set of states in h. For example, if $h = a(pq)$, then $St(h) = \{p, q\}$. Let

$$p' = (a_0^{root,\lambda}, q_0) \leftarrow (a_1^{a_0,u_1}, q_1) \leftarrow \cdots \leftarrow (a_n^{a_{n-1},u_n}, q_n)$$

be a path, where $a_0 = sl$. We say that an edge $(a^{b,u}, q) \leftarrow (c^{a,v}, p)$ *depends* on rl and p' under $D_\#$ if

- $(a_n^{a_{n-1},u_n}, q_n) = (a^{b,u}, q)$,
- $c^{a,v}$ appears in $d_\#(a)$, and
- $p \in St(h)$.

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

- $(sl^{root,\lambda}, q_0) \in V_D$. $(sl^{root,\lambda}, q_0)$ is called the *root* of G_D.
- If the following conditions hold, then $(c^{a,v}, p) \in V_D$ and $(a^{b,u}, q) \leftarrow (c^{a,v}, p) \in E_D$.
  - For a rule $rl = (q, a) \to h \in R$, G_D contains a path p' from $(a^{b,u}, q)$ to the root such that $(a^{b,u}, q) \leftarrow (c^{a,v}, p)$ depends on rl and p'.

- G_D does not contain any other nodes and edges stated above.

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

Lemma 1 *Let $D = (d, sl, \Sigma)$ 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 $rl = (q, a) \to h \in R$ be a rule. Then rl is applicable to $(a^{b,u}, q)$ in G_D iff rl is applicable to $a^{b,u}$ under $D_\#$.*

Proof: $\Rightarrow$) Assume that rl is applicable to $(a^{b,u}, q)$ in G_D. Then G_D contains a path

$$(a_0^{root,\lambda}, q_0) \leftarrow (a_1^{a_0,u_1}, q_1) \leftarrow \cdots \leftarrow (a_n^{a_{n-1},u_n}, q_n),$$

where $a_0 = sl$ and $(a_n^{a_{n-1},u_n}, q_n) = (a^{b,u}, q)$. This implies that we can construct a superscripted tree $t_\#$ valid to $D_\#$ such that

- $t_\#$ contains a path $v_0 \to v_1 \to \cdots \to v_n$ such that v_0 is the root with $l(v_0) = a_0^{root,\lambda}$ and that $l(v_i) = a_i^{a_{i-1},u_i}$ for $1 \le i \le n$, and that
- there is a sequence $rl_0, rl_1, \cdots, rl_n = rl$ of rules such that $q_i \in St(h_{i-1})$ for $1 \le i \le n$, where $rl_i = (q_i, a_i) \to h_i$.

Since $rl = (q, a) \to h$, $q_n = q$, and $a_n = a$, rl is applied to v_n with $l(v_n) = a_n^{a_{n-1},u_n} = a^{b,u}$ during a translation of Tr.

$\Leftarrow$) Assume that rl is applicable to $a^{b,u}$ under $D_\#$. Then there is a superscripted tree $t_\#$ valid to $D_\#$ such that $t_\#$ contains a node v with $l(v) = a^{b,u}$ and that rl is applied to v during a translation of Tr. Then we have the following.

a) $t_\#$ contains path

$$v_0 \to v_1 \to \cdots \to v_n$$

such that v_0 is the root and that $v_n = v$ with $l(v) = a^{b,u}$. Moreover, let $l(v_0) = a_0^{root,\lambda}$ and $l(v_i) = a_i^{a_{i-1},u_i}$ for $1 \le i \le n$. Then we have that $d_\#(a_{i-1})$ contains $a_i^{a_{i-1},u_i}$ for $1 \le i \le n$.

b) There is a sequence $rl_0, rl_1, \cdots, rl_n = rl$ of rules such that $q_i \in St(h_{i-1})$ for $1 \le i \le n$, where $rl_i = (q_i, a_i) \to h_i$ and $q_n = q$.

By a) and b) above, we can inductively show that G_D contains an edge $(a_0^{root,\lambda}, q_0) \leftarrow (a_1^{a_0,u_1}, q_1)$ and $(a_i^{a_{i-1},u_i}, q_i) \leftarrow (a_{i+1}^{a_i,u_{i+1}}, q_{i+1})$ for $1 \le i \le n-1$. Thus G_D contains a path

$$p' = (a_0^{root,\lambda}, q_0) \leftarrow (a_1^{a_0,u_1}, q_1) \leftarrow \cdots \leftarrow (a_n^{a_{n-1},u_n}, q_n),$$

where $(a_0^{root,\lambda}, q_0)$ is the root and $(a_n^{a_{n-1},u_n}, q_n) = (a^{b,u}, q)$. Hence rl is applicable to $(a^{b,u}, q)$ in G_D. □

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 $(a^{b,u}, q)$ obtained in the previous iteration and each rule $rl = (q, pat) \to h$ applicable to $(a^{b,u}, q)$ in G_D, the nodes and the edges obtained by applying rl to $(a^{b,u}, q)$ 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*

$$p' = (a_0^{root,\lambda}, q_0) \leftarrow (a_1^{a_0,u_1}, q_1) \leftarrow \cdots \leftarrow (a_n^{a_{n-1},u_n}, q_n)$$

be a path in G_D. Then there is a simple (i.e., containing no cycle) path p'' from $(a_n^{a_{n-1},u_n}, q_n)$ to $(a_0^{root,\lambda}, q_0)$.

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

1. Let $D_\# = (d_\#, sl^{root,\lambda}, \Sigma, \Sigma_\#)$ be the superscripted DTD of D.
2. $V_D \leftarrow \{(sl^{root,\lambda}, q_0)\}$; $E_D \leftarrow \emptyset$; $E_{old} \leftarrow \emptyset$;$V_{old} \leftarrow \emptyset$
3. **do**
4. $\quad E_{old} \leftarrow E_D$; $V_{old} \leftarrow V_D$;
5. $\quad$ **for each** $(a^{b,u}, q) \in V_D \setminus V_{old}$ **do**
6. $\quad\quad$ **for each** rule $rl = (q, pat) \to h \in R$ **do**
7. $\quad\quad\quad$ **if** rl is applicable to $(a^{b,u}, q)$ in $G_D = (V_D, E_D)$ **then**
8. $\quad\quad\quad\quad$ **for** each superscripted label $c^{e,v}$ appearing in $d_\#(a)$ and each state $p \in St(h)$ **do**
9. $\quad\quad\quad\quad\quad E_D \leftarrow E_D \cup \{(a^{b,u}, q) \leftarrow (c^{e,v}, p)\}$
10. $\quad\quad\quad\quad\quad V_D \leftarrow V_D \cup \{(c^{e,v}, p)\}$
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'. □

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 $(a^{b,u}, q)$, 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 p' from $(a^{b,u}, q)$ to the root in H_D and a rule rl, if an edge $e = (a^{b,u}, q) \leftarrow (c^{a,v}, p)$ depends on rl and p' and p' is a simple path, then we say that p' 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 $e = (a^{b,u}, q) \leftarrow (c^{a,v}, p)$ and a rule rl whose simple dependency path is of length k, and let p' be the simple dependency path. Let

$$p' = \overbrace{(a_0^{root,\lambda}, q_0) \rightsquigarrow \underbrace{(a_{k-1}^{a_{k-2},u_{k-1}}, q_{k-1}) \to (a_k^{a_{k-1},u_k}, q_k)}_{e'}}^{p'' \text{ (length } k-1)},$$

where $(a_k^{a_{k-1},u_k}, q_k) = (a^{b,u}, q)$. Since the simple dependency path p'' of e' is of length $k-1$, p'' is obtained in G_D. Thus e' is also obtained in G_D by CONSTDG. Therefore, p' is in G_D and thus e is obtained in lines 7 to 10 of CONSTDG. □

Consider the time complexity of CONSTDG. The numbers of nodes of a dependency graph is in $O(|D||Q|)$, 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 $D = (d, sl, \Sigma)$, update script s to D, tree transducer $Tr = (Q, \Sigma, q_0, R)$ in UTT.
Output: $R^-(a^{b,u}, a^{b',u'})$ for every pair $(a^{b,u}, a^{b',u'})$ of superscripted labels such that $a^{b',u'} \in C_D(a^{b,u}, s)$.

1. $G_D \leftarrow$ CONSTDG(D, Tr).
2. $G'_D \leftarrow$ CONSTDG$(s(D), Tr)$.
3. Compute $C_D(a^{b,u}, s)$ for each $a^{b,u}$ in $D_\#$.
4. **for** each pair $(a^{b,u}, a^{b',u'})$ such that $a^{b',u'} \in C_D(a^{b,u}, s)$ **do**
5. $\quad M \leftarrow \{rl \in R \mid rl$ is applicable to $(a^{b,u}, q)$ in $G_D\}$
6. $\quad M' \leftarrow \{rl \in R \mid rl$ is applicable to $(a^{b',u'}, q)$ in $G'_D\}$
7. $\quad R^-(a^{b,u}, a^{b',u'}) \leftarrow M \setminus M'$
8. **return** $\{R^-(a^{b,u}, a^{b',u'}) \mid a^{b',u'} \in C_D(a^{b,u}, s)\}$

Figure 7: Algorithm FINDAFFECTEDRULES

loop is in $O(|D||Q|)$. For each node $(a^{b,u}, q)$ selected in line 5 and a rule rl selected in line 6, $O(|D||Q|)$ edges are added to E_D. Hence CONSTDG runs in

$$O(|D||Q| \cdot |D||Q| \cdot |R| \cdot |D||Q|) = O(|R|(|D||Q|)^3). \quad (1)$$

4.3 Detecting Rules Affected by DTD Updates

Algorithm FINDAFFECTEDRULES (Fig. 7) computes $R^-(a^{b,u}, a^{b',u'})$ for every pair $(a^{b,u}, a^{b',u'})$ of corresponding superscripted labels $a^{b,u}$ and $a^{b',u'}$ ($R^+(a^{b,u}, a^{b',u'})$ 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 $R^-(a^{b,u}, a^{b',u'})$.

The following theorem immediately follows from Lemmas 1 and 3.

Theorem 1 FINDAFFECTEDRULES *correctly computes* $R^-(a^{b,u}, a^{b',u'})$ *for any pair* $(a^{b,u}, a^{b',u'})$ *such that* $a^{b',u'} \in C_D(a^{b,u}, s)$.

Consider the running time of FINDAFFECTEDRULES. Let $D_{max} = \max_{1 \le i \le n} |s_i(D)|$, where $s = op_1 op_2 \cdots op_n$ and $s_i = op_1 op_2 \cdots op_i$. Lines 1 and 2 require $O(|R|(|D_{max}||Q|)^3)$ by (1). In line 3, $C_D(a^{b,u}, s)$ can be obtained by computing $C_D(a^{b,u}, s_i)$ for $i = 1, 2, \cdots, n$, which can be done in $O(|s||D_{max}|^2)$ time. Since the for loop in line 4 iterates $O(|D_{max}|^2)$ times and lines 5 to 6 can be done in $O(|R|)$ time, lines 4 to 7 requires $O(|R||D_{max}|^2)$. Thus FINDAFFECTEDRULES runs in

$$\begin{aligned} & O(|R|(|D_{max}||Q|)^3 + |s||D_{max}|^2 + |R||D_{max}|^2) \\ = \quad & O(|R|(|D_{max}||Q|)^3 + |s||D_{max}|^2). \end{aligned}$$

5. Intractability of the Problem

In the previous section, the affected rule detection problem is shown to be in PTIME for UTT. In this section, we show that the problem is intractable for UTT^{pat} and UTT^{sel} under restricted DTDs.

5.1 Intractability for UTT^{sel}

We first consider the intractability of the problem for UTT^{sel}. To show the intractability, we consider the following simpler problem, called *rule applicability problem*.

- Input: DTD D, superscripted label $a^{b,u}$ in $D_\#$, tree transducer $Tr = (Q, \Sigma, q_0, R)$, and a rule $rl \in R$.
- Problem: Determine whether rl is applicable to $a_{b,u}$ under $D_\#$.

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 $D = (d, sl, \Sigma)$ is *flat* if for any label a appearing in $d(sl)$, $d(a) = \epsilon$. We first show that the rule applicability problem is intractable for UTT^{sel} under such simple DTDs.

Lemma 4 *The rule applicability problem for UTT^{sel} 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 $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, where C_i is a clause with three literals. Let $x_1, x_2, \cdots, x_m$ be the variables appearing in ϕ.
- Problem: Determine whether there is a truth assignment of $x_1, x_2, \cdots, x_m$ such that $\phi = true$.

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

$$\begin{aligned} d(s) &= (T_1|F_1)(T_2|F_2)\cdots(T_m|F_m), \\ d(c_i) &= \epsilon \quad (1 \le i \le n), \end{aligned}$$

and

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

where,

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

For example, if $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$, $C_1 = x_1 \vee \neg x_2 \vee x_3$, $C_2 = x_2 \vee x_3 \vee x_4$, $C_3 = \neg x_1 \vee \neg x_2 \vee x_3$, $C_4 = x_1 \vee x_2 \vee \neg x_4$, then $T_1 = c_1 c_4$, $F_1 = c_3$, $T_2 = c_2 c_4$, $F_2 = c_1 c_3$, and so on.

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

$$\begin{aligned} Q &= \{q_0, q_1, \cdots, q_{n+1}\}, \\ R &= \{(q_0, s) \to s(q_1, /s/c_1), \\ & \quad (q_1, c_1) \to c'_1(q_2, /s/c_2), \\ & \quad (q_2, c_2) \to c'_2(q_3, /s/c_3), \\ & \quad \vdots \\ & \quad (q_{n-1}, c_{n-1}) \to c'_{n-1}(q_n, /s/c_n), \\ & \quad (q_n, c_n) \to c'_n(q_{n+1}, /s), \\ & \quad (q_{n+1}, s) \to c'_{n+1}\}. \end{aligned}$$

In the following, we show that the last rule $(q_{n+1}, s) \to c'_{n+1}$ is applicable to $s^{root,\lambda}$ under $D_\#$ if and only if ϕ is satisfiable.

($\Rightarrow$) Assume that $(q_{n+1}, s) \to c'_{n+1}$ is applicable to $s^{root,\lambda}$ under $D_\#$. From the left-hand side of the rule, s is assigned with state q_{n+1}, therefore rule $(q_n, c_n) \to c'_n(q_{n+1}, /s)$ needs to be applied to c_n. Similarly, for each $i = n, n-1, \cdots, 1$, c_i needs to be assigned with state q_i, which implies that rule $(q_{i-1}, c_{i-1}) \to c'_{i-1}(q_i, /s/c_i)$ needs to be applied to c_{i-1} for $2 \le i \le n$. This implies that there is a tree t valid to D such that s has all of $c_1, c_2, \cdots, c_n$ as its children. Let $cls \in L(d(s))$ be the sequence of child labels of s in t. Consider the following truth assignment of $x_1, x_2, \cdots, x_m$.

- For every $1 \le i \le m$, if cls contains T_i, $x_i = true$, if cls contains F_i, $x_i = false$.

Then it is obvious that the truth assignment makes $\phi = true$.

($\Leftarrow$) Assume that ϕ is satisfiable. Then, there is a truth assignment of $x_1, x_2, \cdots, x_m$ 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 $c_1, c_2, \cdots, c_n$ as child elements of s. Thus it is easy to show that $(q_{n+1}, s) \to c'_{n+1}$ is applicable to $s^{root,\lambda}$ under $D_\#$. □

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

Theorem 2 *The affected rule detection problem for UTT^{sel} 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 $a^{b,u}$ in $D_\#$, and let $a^{b',u'}$ be a superscripted label in $s(D)_\#$ corresponding to $a^{b,u}$. Since l is a new label, no rule is applicable to $a^{b',u'}$ under $s(D)_\#$. Thus rl is applicable to $a^{b,u}$ under $D_\#$ iff $rl \in R^-(a^{b,u}, a^{b',u'})$. □

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^{sel} 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 $\neg x_i$ do not appear in the same clause.

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

$$\begin{aligned} d(s) &= X_1 X_2 \cdots X_m, \\ d(X_i) &= T_i|F_i, (1 \le i \le m) \\ d(c_i) &= \epsilon, \quad (1 \le i \le n) \end{aligned}$$

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_j 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.

$$\begin{aligned} Q &= \{q_0, q_1, \cdots, q_{n+1}\}, \\ R &= \{(q_0, s) \to s(q_1, /s/*/c_1), \\ &\quad (q_1, c_1) \to c'_1(q_2, /s/*/c_2), \\ &\quad (q_2, c_2) \to c'_2(q_3, /s/*/c_3), \\ &\quad \vdots \\ &\quad (q_{n-1}, c_{n-1}) \to c'_{n-1}(q_n, /s/*/c_n), \\ &\quad (q_n, c_n) \to c'_n(q_{n+1}, /s), \\ &\quad (q_{n+1}, s) \to c'_{n+1}\}. \end{aligned}$$

Similarly to Lemma 4, we can show that the last rule $(q_{n+1}, s) \to c'_{n+1}$ in R is applicable to $s^{root,\lambda}$ under $D_\#$ if and only if ϕ 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 is NP-hard under duplicate-free DTDs. □

5.2 Intractability for UTT^{pat}

Pattern is a simple and most likely extension to UTT. However, the following theorem shows that the affected rule detection problem becomes intractable for UTT^{pat}.

Theorem 4 *The affected rule detection problem for UTT^{pat} 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 $\phi = C_1 \vee C_2 \vee \cdots \vee C_n$, where C_i is a conjunctive clause with three literals. Let $x_1, x_2, \cdots, x_m$ be the variables appearing ϕ.
- Problem: Determine whether ϕ is a non-tautology, that is, there is a truth assignment of $x_1, x_2, \cdots, x_m$ 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.

$$\begin{aligned} d(s) &= T_1|F_1, \\ d(T_1) = d(F_1) &= T_2|F_2, \\ d(T_2) = d(F_2) &= T_3|F_3, \\ &\vdots \\ d(T_{m-1}) = d(F_{m-1}) &= T_m|F_m, \\ d(T_m) = d(F_m) &= b, \\ d(b) &= c, \\ d(c) &= \epsilon, \end{aligned}$$

and

$$\begin{aligned} \Sigma &= \{s, b, c\} \cup \{T_1, T_2, \cdots, T_m\} \cup \{F_1, F_2, \cdots, F_m\} \\ &\quad \cup \{c_1, c_2, \cdots, c_{n+1}\}. \end{aligned}$$

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 a path representing a truth assignment for ϕ.

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

$$\begin{aligned} Q &= \{q_0, q_1, \cdots, q_{m+2}\}, \\ R &= R_1 \cup R_2 \cup R'_2, \end{aligned}$$

where

$$\begin{aligned} R_1 &= \{(q_0, s) \to s(q_1), \\ &\quad (q_1, T_1) \to c_1(q_2), (q_1, F_1) \to c_1(q_2), \\ &\quad (q_2, T_2) \to c_2(q_3), (q_2, F_2) \to c_2(q_3), \\ &\quad \vdots \\ &\quad (q_m, T_m) \to c_m(q_{m+1}), (q_m, F_m) \to c_m(q_{m+1}), \\ &\quad (q_{m+1}, b) \to b(q_{m+2})\}, \end{aligned}$$

$$
\begin{aligned}
R_2 &= \{(q_{m+2}, pat_1) \to c, \\
&\quad (q_{m+2}, pat_2) \to c, \\
&\qquad \vdots \\
&\quad (q_{m+2}, pat_n) \to c\}, \\
R'_2 &= \{(q_{m+2}, c) \to c\}.
\end{aligned}
$$

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_2 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 $C_i = (l_{i_1} \wedge l_{i_2} \wedge l_{i_3})$ be the i-th conjunctive clause of ϕ, and let x_{i_j} be the variable of literal l_{i_j} $(1 \le j \le 3)$. Without loss of generality, we assume that $i_1 < i_2 < i_3$. Then pat_i is defined as follows.

$$
pat_i = \overbrace{*/\cdots/*}^{i_1-1\ *'s}/L_{i_1}/\overbrace{*/\cdots/*}^{i_2-i_1-1\ *'s}/L_{i_2}/\overbrace{*/\cdots/*}^{i_3-i_2-1\ *'s}/L_{i_3}/\overbrace{*/\cdots/*}^{m-i_3\ *'s}/b/c,
$$

where

$$
L_{i_j} = \begin{cases} T_{i_j} & \text{if } l_{i_j} = x_{i_j}, \\ F_{i_j} & \text{if } l_{i_j} = \neg x_{i_j}. \end{cases} \qquad (1 \le j \le 3)
$$

We show that ϕ is a non-tautology iff the rule $(q_{m+2}, c) \to c$ of R'_2 is applied to $c^{b,\lambda}$.

($\Rightarrow$) Assume that ϕ is a non-tautology. Then there is a truth assignment of $x_1, x_2, \cdots, x_m$ such that $\phi = false$. Let t be the tree representing this truth assignment. Since $C_i = false$ for every $1 \le i \le n$, none of $pat_1, pat_2, \cdots, pat_n$ 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 ϕ is a tautology. Then for any truth assignment of $x_1, x_2, \cdots, x_m$, $\phi = true$. Therefore, for any truth assignment of $x_1, x_2, \cdots, x_m$, $C_i = true$ 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 $|pat_i| \ge 2$ for any i, the rule in R'_2 is not applied to $c^{b,\lambda}$).

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]

[1] James Clark, editor. *XSL Transformations (XSLT) Version 1.0*. https://www.w3.org/TR/xslt.

[2] G. Guerrini, M. Mesiti, and D. Rossi. Impact of XML schema evolution on valid documents. In *Proc. WIDM*, pages 39–44, 2005.

[3] Kazuma Hasegawa, Kosetsu Ikeda, and Nobutaka Suzuki. An algorithm for transforming XPath expressions according to schema evolution. In *Proc. DChanges 2013*, 2013.

[4] K. Hashimoto, Y. Ishihara, and T. Fujiwara. A proposal of update operations for schema evolution in XML databases and their properties on preservation of schema's expressive power. *IEICE Transactions on Information and Systems (Japanese Edition)*, J90-D(4):990–1004, 2007.

[5] Kazuma Horie and Nobutaka Suzuki. Extracting differences between regular tree grammars. In *Proc. ACM SAC*, pages 859–864, 2013.

[6] E. Leonardi, T. T. Hoai, S. S Bhowmick, and S. Madria. DTD-Diff: A change detection algorithm for DTDs. *Data & Knowledge Engineering*, 61:384–402, 2007.

[7] Wim Martens and Frank Neven. Typechecking top-down uniform unranked tree transducers. In *Proc. ICDT*, pages 64–78, 2002.

[8] Wim Martens, Frank Neven, and Marc Gyssens. Typechecking top-down xml transformations: Fixed input or output schemas. *Inf. Comput.*, 206(7):806–827, July 2008.

[9] Manizheh Montazerian, Peter T. Wood, and Seyed R. Mousavi. Xpath query satisfiability is in ptime for real-world dtds. In *Proc. the 5th International Conference on Database and XML Technologies (XSym'07)*, pages 17–30, 2007.

[10] Raquel Oliveira, Pierre Genevès, and Nabil Layaïda. Toward automated schema-directed code revision. In *Proc. ACM DocEng'12*, pages 103–106, 2012.

[11] N. Suzuki. An algorithm for inferring k optimum transformations of XML document from update script to DTD. *IEICE Transactions on Information and Systems*, E93-D(8):2198—2212, 2010.

[12] N. Suzuki, Y. Fukushima, and K. Ikeda. Satisfiability of simple xpath fragments under duplicate-free dtds. *IEICE Transactions on Information and Systems*, E96-D(5):1029—1042, 2013.

[13] Yang Wu and Nobutaka Suzuki. Detecting XSLT rules affected by schema evolution. In *Proc. ACM DocEng'15*, pages 143–146, 2015.

[14] Yang Wu and Nobutaka Suzuki. An algorithm for correcting xslt rules according to dtd updates. In *Proc. DChanges'16*, page 8p., 2016.

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 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. 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.