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, 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^new, and UTT^old, 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^new and UTT^old are extensions of UTT, where put 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 as XSLT. We next show that the problem becomes NP-hard for UTT^new and UTT^old, 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_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, 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 l(v) = sl for the root v of t and for any node n in t, l(v1)l(v2)···l(vn) ∈ L(d(l(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+)>

\[1\]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

© The Database Society of Japan

Information and Media Technologies 12: 103–110 (2017)
To define update operations to DTDs, we need to define the positions of elements/operands 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 \((a(b(c)))\) and each node in the tree is associated with its position. For a regular expression, the label at position \(u\) in \(r\) is denoted \(d(r, u)\) and the subexpression at position \(u\) of \(r\) is denoted \(s(r, u)\). For example, consider the regular expression \(r = (a(b(c)))\) in Fig. 2(a). Then \(l(r, 1) = 1\), \(s(r, 1) = a\), \(l(r, 2) = 1\), \(s(r, 2) = a\).

Let \(D = (d, \Sigma, \Lambda, \Lambda')\) be a DTD. Following [11], we define six update operations to \(D\). The first four operations insert/delete/unnest operators (Fig. 2 shows an example).

- \(\text{ins} a(b, u)\): inserts a label \(b\) at position \(u\) in \(d(a)\).
- \(\text{del} a(u)\): deletes the label at position \(u\) in \(d(a)\).
- \(\text{nest} a(b, u)\): nests the subexpression at \(u\) in \(d(a)\) by \(b\).
  
  This operation replaces the subexpression at \(u\) in \(d(a)\) by \(b\) and sets \(d(b) = s(b, a)\).
- \(\text{unnest} a(u)\): this is the inverse operation of \(\text{nest}\), and replaces the label \(l = l(d(a), u)\) at \(u\) in \(d(a)\) by regular expression \(l\).
- \(\text{ins} Opr a(op, u, \Sigma)\): inserts an operator \(Opr\) as the parent of the sibling subexpressions in \(a(u, u + 1, \ldots, u)\) in \(d(a)\), where \(op\) is \([\cdot, \cdot, \cdot, \cdot]\). We assume that \(op\) has exactly one operand (child) whenever \(op\) is \([\cdot, \cdot, \cdot, \cdot]\). \(\text{ins} Opr a(op, u, \Sigma)\) is applicable to \(d(a)\) only if \(i = j\) (\(op\) has only one operand) or \(i < j\) and \(op\) is \([\cdot, \cdot, \cdot, \cdot]\).
- \(\text{del} Opr a(u)\): 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 \(l(d(a), u)\) is \([\cdot, \cdot, \cdot, \cdot]\), 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 \(s = op_1op_2\cdots op_n\), we define \(s(D) = op_n(\cdots (op_1(D)))\).

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 \(D'\).
is, it is of the form \(l_s/\cdots/l_s\), where \(l_s\) 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 hugs exactly such that leaf nodes can be labeled with elements from \(Q \times SEL\).

A tree transducer in \(\text{UTT}^{\text{root,sol}}\) 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 \(\text{UTT}^{\text{root}}\).

For example, \((q, a/b/c) \rightarrow c(p/\_d)\) 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 \(T_r = (Q, \Sigma, q_0, R)\) on node \(v\) of tree \(t\) in state \(q\), denoted by \(T_r^q(t,v)\), is defined as follows.

- If there is a rule \((q, pat) \rightarrow h\) in \(R\) such that \(v\) matches \(pat\), then \(T_r^q(t,v)\) is obtained from \(h\) as follows. For each leaf node \(u\) in \(h\)
  - if \(sel\) is a relative location path, then replace \(u\) with \(h\)
  - if \(sel\) is an absolute location path, then replace \(u\) with \(h\)

  - Otherwise, \(T_r^q(t,v) = \epsilon\).

  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 \(\text{UTT}^{\text{root}}\) we mean the subclass of \(\text{UTT}^{\text{root,sol}}\) such that \(pat\) is a single label for any rule \((q, pat) \rightarrow h\).

### 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 from multiple content model definitions, and we have to distinguish such elements when detecting the rules affected by DTD updates. By \(\phi^a\) we mean the element \(a\) at position \(u\) in \(\phi(b)\). We say that \(\phi^a\) is a superscripted label. If \(a\) is the start label, then the corresponding superscripted label is \(\phi^{s\text{ef}}\). For a DTD \(D = (d, sl, \Sigma)\), by \(D_s = (d_s, sl^{\text{root}}, \Sigma, \Sigma)\) 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 \(D = (d, \text{article, } \Sigma)\) be the DTD in Example 1. Then \(D_s = (d_s, \text{article}^{\text{root}}, \Sigma, \Sigma_s)\), where

\[
\begin{align*}
  d_s(\text{article}) &= \text{article}^{\text{ef}}(\text{section}^{\text{article},1})^*, \\
  d_s(\text{section}) &= \text{section}^{\text{article},1}(\text{para}^{\text{section},2})^*, \\
  d_s(\text{title}) &= \epsilon, \\
  d_s(\text{para}) &= \epsilon,
\end{align*}
\]

and

\[
\Sigma_s = \{\text{article}^{\text{ef}}, \text{title}^{\text{article},1}, \text{section}^{\text{article},1}, \text{para}^{\text{section},2}\}.
\]

For a tree \(t\) valid to \(D_s\), \(t_s\) 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 \(\phi\) be an update script to \(D\). For a superscripted label \(\phi^{s\text{ef}}\) in \(D_s\), if \(\phi^{s\text{ef}}\) is not deleted by \(\phi\), then \(\phi^{s\text{ef}}\) also appears in \(s(D_s)\) as \(\phi^{s\text{ef}}\) for some label \(f\) and some position \(w\).

We say that \(\phi^{s\text{ef}}\) corresponds to \(\phi^{s\text{ef}}\) and that \(\phi^{s\text{ef}}\) corresponds to \(\phi^{s\text{ef}}\). As shown below, more than one superscripted label in \(s(D_s)\) may correspond to \(\phi^{s\text{ef}}\). For a superscripted label \(\phi^{s\text{ef}}\) in \(D_s\), by \(\phi^{s\text{ef}}\) we mean the set of superscripted labels in \(s(D_s)\) corresponding to \(\phi^{s\text{ef}}\) (the formal definition of \(\phi^{s\text{ef}}\) is omitted because of space limitation).

---

1. If no update script between old and new DTDs is given, the algorithms in [6, 5] can generate update scripts between DTDs.
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 \( a^{\alpha} \) and a rule \( r \in R \), if \( r \) is applicable to \( a^{\beta} \) under \( D_k \) for some tree \( t \) valid to \( D \) and some node \( v \) in \( t \), then it is applicable to \( a^{\alpha} \) under \( D_k \) if some node \( v' \) in \( t' \) is applicable to \( a^{\alpha} \).

Let \( a^{\alpha} \) be a superscripted label in \( D_k \) and \( a^{\alpha'} \in C_{D_k}(a^{\alpha'}, s) \). We define two sets of rules affected by \( a^{\alpha} \) and \( a^{\alpha'} \), denoted \( R'(a^{\alpha}, a^{\alpha'}) \) and \( R'(a^{\alpha'}, a^{\alpha'}) \), as follows.

- \( R'(a^{\alpha}, a^{\alpha'}) \) is the set of rules \( r' \in R \) such that \( r' \) is not applicable to \( a^{\alpha} \) under \( D_k \) but becomes applicable to \( a^{\alpha'} \) under \( D_k \).
- \( R'(a^{\alpha'}, a^{\alpha'}) \) is the set of rules \( r' \in R \) such that \( r' \) is applicable to \( a^{\alpha'} \) under \( D_k \) but not applicable to \( a^{\alpha} \) under \( D_k \).

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

- Input: DTD \( D = (d, \Sigma, \Sigma) \), update script \( s \), tree transducer \( Tr = (Q, \Sigma, q_0, R) \), rule \( r' \in R \), pair \((a^{\alpha}, a^{\alpha'})\) of corresponding superscripted labels, and flag \( f \in \{+, -\} \).
- Problem: Determine whether \( r' \in R'(a^{\alpha}, a^{\alpha'}) \) if \( f = '+' \), and determine whether \( r' \in R'(a^{\alpha'}, a^{\alpha'}) \) 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
• \( G_D \) does not contain any other nodes and edges stated above.

We say that rule \( rl = (q, a) \rightarrow h \) is applicable to a node \((a_n^{\text{out}}, q_n)\) in \( G_D \) if \( G_D \) contains a path \( p' \) from \((a_0^{\text{out}}, q_0)\) 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 \( D = (d, s, \Sigma) \) 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 \((a^{\text{out}}, q)\) in \( G_D \) if and only if \( rl \) is applicable to \((a^{\text{out}}, q)\) in \( G_D \).

**Proof:** Assume that \( rl \) is applicable to \((a^{\text{out}}, q)\) in \( G_D \). Then \( G_D \) contains a path

\[
(a_0^{\text{out}}, q_0) \leftarrow (a_0^{\text{out}}, q_1) \leftarrow \cdots \leftarrow (a_n^{\text{out}}, q_n),
\]

where \( q_0 = s \) and \( (a_n^{\text{out}}, q_n) = (a^{\text{out}}, q) \). This implies that we can construct a superscripted tree \( t_i \) valid to \( D_k \) such that

- \( t_i \) contains a path \( v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_n \) such that \( v_0 \) is the root with \( l(v_0) = a_0^{\text{out}} \) and that \( l(v_i) = a_i^{\text{out}} \) for \( 1 \leq i \leq n \), and that
- there is a sequence \( a_0^{\text{out}}, a_1^{\text{out}}, \ldots, a_n^{\text{out}} \) such that \( q_i \in S(h_i) \) for \( 1 \leq i \leq n \), where \( rl = (q, a) \rightarrow h \).

Since \( rl = (q, a) \rightarrow h \), and \( a_n = a \), \( rl \) is applied to \( v_n \) with \( l(v_n) = a^{\text{out}} \) during a translation of \( Tr \).

Thus, \( rl \) is applicable to \((a^{\text{out}}, q)\) in \( G_D \).

**Lemma 2** Let \( G_D \) be the dependency graph of \( D \) and \( Tr \), and let \( p' = (a_0^{\text{out}}, q_0) \leftarrow (a_1^{\text{out}}, q_1) \leftarrow \cdots \leftarrow (a_{i-1}^{\text{out}}, q_{i-1}) \rightarrow (a_i^{\text{out}}, q_i) \) be a path in \( G_D \). Then there is a simple (i.e., containing no cycle) path \( p'' \) from \((a_0^{\text{out}}, q_0)\) to \((a_n^{\text{out}}, q_n)\).

**Proof:** Consider the opposite direction and let us show that every edge \( e \) in \( H_D \) is of length \( k \leq n \). Let \( \lambda \) be the superscripted DTD of \( D \) whose \( \lambda \) is applied to \((a^{\text{out}}, q)\) in \( G_D \) during a translation of \( Tr \).

Since \( \lambda \) is a simple dependency path of length \( k \), we say that rule \( rl \) is applicable to \((a^{\text{out}}, q)\) in \( G_D \). The following lemma holds immediately.

**Lemma 3** Let \( G_D \) be the result of \( \text{CONSTDG} \) 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 \((a^{\text{out}}, q)\), it is clear that \( G_D \) is a subgraph of \( H_D \).

Consider the opposite direction and let us show that every node and edge in \( H_D \) is in \( G_D \). For a path \( p' \) from \((a^{\text{out}}, q)\) to the root in \( H_D \) and a rule \( rl \), if an edge \( e = (a^{\text{out}}, q) \leftarrow (a^{\text{in}}, p) \) depends on \( rl \) and \( p' \) is a simple path, then we say that \( e \) is a simple dependency path of \( rl \). 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 \( \text{CONSTDG} \).

**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 \( \text{CONSTDG} \). Consider an edge \( e = (a^{\text{out}}, q) \leftarrow (a^{\text{in}}, p) \) and a rule \( rl \) whose simple dependency path is of length \( k \), and let \( rl' \) be the simple dependency path.

Let \( p''(\text{length } k - 1) \) be a path in \( G_D \). Then there is a simple (i.e., containing no cycle) path \( p''(\text{length } k - 1) \) from \((a_0^{\text{out}}, q_0)\) to \((a_i^{\text{out}}, q_i)\).

\[
p'' = (a_0^{\text{out}}, q_0) \leftarrow (a_1^{\text{out}}, q_1) \leftarrow \cdots \leftarrow (a_i^{\text{out}}, q_i),
\]

where \((a_i^{\text{out}}, q_i) = (a^{\text{out}}, q) \). Since the simple dependency path \( p'' \) is of length \( k - 1 \), \( p'' \) is obtained in \( G_D \). Thus \( e' \) is also obtained in \( G_D \). By \( \text{CONSTDG} \). Therefore, \( p' \) is in \( G_D \), and thus \( e' \) is obtained in line 10 of \( \text{CONSTDG} \).

Consider the time complexity of \( \text{CONSTDG} \). The numbers of nodes of a dependency graph is \( O(|D||q_0|) \), 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, s, ∆), update script s to D, tree transducer
Tr = (Q, ∆, qo, R) in UTT.
Output: R′(a′, a′′) for every pair (a′, a′′) of superscripted labels such that a′′ ∈ C D(a′, s).

1. GD ← CONSTDG(D, Tr).
2. G′D ← CONSTDG(s(D), Tr).
3. Compute C D(a′, s) for each a′ in Ds.
4. For each pair (a′, a′′) such that a′′ ∈ C D(a′, s) do
   5. M ← {rl ∈ R | rl is applicable to (a′, q) in GD}
   6. M′ ← {rl ∈ R | rl is applicable to (a′′, q) in G′D}
   7. R′(a′, a′′) ← M \ M′
   8. return R′(a′, a′′)|a′′ ∈ C D(a′, s))

Figure 7: Algorithm FINDAFFECTEDRULES

loop is in O(D(|Q|· |D||Q| · |R| · |D|)) = O((R(|D||Q|)^3).

4.3 Detecting Rules Affected by DTD Updates

Algorithm FINDAFFECTEDRULES (Fig. 7) computes R′(a′, a′′) for every pair (a′, a′′) of corresponding superscripted labels a′ and a′′ (R′(a′, a′′)) can be obtained similarly. This algorithm constructs dependency graphs GD and G′D for old/new DTDs, and then calculates the diff between GD and G′D to obtain R′(a′, a′′).

The following theorem immediately follows from Lemmas 1 and 3.

Theorem 1 FINDAFFECTEDRULES correctly computes R′(a′, a′′) for any pair (a′, a′′) such that a′′ ∈ C D(a′, s).

Consider the running time of FINDAFFECTEDRULES. Let DMax = max i∈{1,2,...,n} |s(D)|, where s = op₁op₂...opₙ and s₁ = op₀op₁...opₙ. Lines 1 and 2 require O(|R|(|Dmax||Q|)^3) by (1). In line 3, CD(a′, s) can be obtained by computing C D(a′, s) for i = 1, 2, ..., n, which can be done in O(|S||Dmax|) time. The for loop in line 4 iterates O(|Dmax|^3) times and lines 5 to 6 can be done in O(|R|) time, lines 4 to 7 requires O(|Dmax|^3). Thus FINDAFFECTEDRULES runs in

O(|R|(|Dmax||Q|)^3 + |S||Dmax|^3 + |R||Dmax|^3) = O(|R|(|Dmax||Q|)^3 + |S||Dmax|^3).

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′ and UTT′′ under restricted DTDs.

5.1 Intractability for UTT′

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

• Input: DTD D, superscripted label a′ in Ds, tree transducer Tr = (Q, ∆, qo, R), and a rule rl ∈ R.
• Problem: Determine whether rl is applicable to a′ under Ds.

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, s, ∆) 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 UTT′ under such simple DTDs.

Lemma 4 The rule applicability problem for UTT′ 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 φ = C₁ ∨ C₂ ∨ ··· ∨ Cₙ, where Ci is a clause with three literals. Let x₁, x₂, ..., xₙ be the variables appearing in φ.
• Problem: Determine whether there is a truth assignment of x₁, x₂, ..., xₙ such that φ = true.

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

d(s) = (T₁,F₁)(T₂,F₂)···(Tₙ,Fₙ),
d(cᵢ) = ε (1 ≤ i ≤ n),

and

Σ = {s} ∪ {C₁, C₂, ..., Cₙ} ∪ {c₁, c₂, ..., cₙ+1},

where,

• c₁ is a label representing clause C₁,
• cᵢ is a label used in tree transducer Tr defined below,
• Tₙ (1 ≤ i ≤ m) lists “the labels representing clauses which contain positive literal xi”,
• Fₙ (1 ≤ i ≤ m) lists “the labels representing clauses which contain negative literal ¬xi”.

For example, if φ = C₁ ∨ C₂ ∨ C₃ ∨ C₄, C₁ = x₁ ∨ ¬x₂ ∨ x₃, C₂ = x₂ ∨ x₃ ∨ x₄, C₃ = ¬x₁ ∨ ¬x₂ ∨ x₅, C₄ = x₁ ∨ x₂ ∨ ¬x₄, then T₁ = c₁c₂, F₁ = c₁, T₃ = c₃c₁, F₃ = c₃c₁, and so on.

Secondly, we define a tree transducer Tr = (Q, ∆, qo, R) as follows.

Q = {q₀, q₁, ..., qₙ+1},

R = {(q₀, s) → s(q₀, /s/c₁), (q₁, c₁) → c₁(q₁, /s/c₂), (q₂, c₂) → c₂(q₂, /s/c₃), ..., (qₙ₋₁, cₙ₋₁) → cₙ₋₁(qₙ−₁, /s/cₙ), (qₙ, cₙ) → cₙ(qₙ, /s/), (qₙ, s) → cₙ₊₁(qₙ, /s/cₙ₊₁).}

In the following, we show that the last rule (qₙ₊₁, s) → cₙ₊₁ is applicable to sroot under Ds if and only if φ is satisfiable.

(⇒) Assume that (qₙ₊₁, s) → cₙ₊₁ is applicable to sroot under Ds. From the left-hand side of the rule, s is assigned with state qₙ₊₁, therefore rule (qₙ₊₁, cₙ₊₁) → cₙ₊₁(qₙ₊₁, /s) needs to be applied to cₙ₊₁. Similarly, for each i = n, n − 1, ..., 1, cᵢ needs to be assigned with state qᵢ, which implies that rule (qᵢ, cᵢ) → cᵢ(qᵢ, /s/cᵢ) needs to be applied to cᵢ for 2 ≤ i ≤ n. This implies that there is a tree t valid to D such that s has all of c₁, c₂, ..., cₙ as its children. Let cls ∈ L(t(s)) be the sequence of child labels of s in t. Consider the following truth assignment of x₁, x₂, ..., xₙ.

• For every 1 ≤ i ≤ n, if cls contains Ti, xi = true, if cls contains Fi, xi = false.


Then it is obvious that the truth assignment makes \( \phi = \text{true} \).

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

- For every disjunction (\( T_i[F_i] \)) in \( \phi \), if \( x_i = \text{true} \), then \( T_i \) is selected, and if \( x_i = \text{false} \), then \( F_i \) is selected.

Since \( \phi = \text{true} \) (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 \( t \). Thus it is easy to show that \( (q_{n+1}, s) \rightarrow c_{n+1}' \) is applicable to \( s_{\text{root}} \) under \( D_{\phi} \).

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

**Theorem 2** The affected rule detection problem for UTT\textsuperscript{pat} is NP-hard even if \( D \) is flat and \(|x| = 1 \).

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

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\textsuperscript{pat} is NP-hard under duplicate-free DTDs.

**Proof:** Again we reduce the SAT 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.

\[
\begin{align*}
    d(s) &= X_1X_2 \cdots X_m, \\
    d(X_i) &= T_i[F_i], \quad (1 \leq i \leq m) \\
    d(c_i) &= \epsilon, \quad (1 \leq i \leq n)
\end{align*}
\]

and

\[
\Sigma = \{s\} \cup \{X_1, X_2, \ldots, X_m\} \cup \{c_1, c_2, \ldots, c_n\} \cup \{c_1', c_2', \ldots, c_{n+1}'\},
\]

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.

\[
Q = \{q_0, q_1, q_2, \ldots, q_{m+2}\},
\]

\[
R = \{(q_0, s) \rightarrow s(q_1), (q_1, c_1) \rightarrow c_1(q_2), (q_2, c_2) \rightarrow c_2(q_3), \ldots, (q_{m+1}, c_{m+1}) \rightarrow c_{m+1}(q_{m+2})\}.
\]

Similarly to Lemma 4, we can show that the last rule \( (q_{n+1}, s) \rightarrow c_{n+1}' \) in \( R \) is applicable to \( s_{\text{root}} \) under \( D_\phi \) 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 is NP-hard under duplicate-free DTDs.

**5.2 Intractability for UTT\textsuperscript{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\textsuperscript{pat}.

**Theorem 4** The affected rule detection problem for UTT\textsuperscript{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 \( \phi = C_1 \lor C_2 \lor \cdots \lor C_n \), where \( C_i \) is a conjunctive clause with three literals. Let \( x_1, x_2, \ldots, x_n \) be the variables appearing \( \phi \).

- Problem: Determine whether \( \phi \) is a non-tautology, that is, there is a truth assignment of \( x_1, x_2, \ldots, x_n \) such that \( \phi = \text{false} \).

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.

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

and

\[
\Sigma = \{s, b, c\} \cup \{T_1, T_2, \ldots, T_m\} \cup \{F_1, F_2, \ldots, F_m\} \cup \{c_1, c_2, \ldots, c_m\}.
\]

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 \( \phi \).

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

\[
Q = \{q_0, q_1, \ldots, q_{m+2}\},
\]

\[
R = R_1 \cup R_2 \cup R_3,
\]

where

\[
R_1 = \{(q_0, s) \rightarrow s(q_1), (q_1, T_1) \rightarrow c_1(q_2), (q_2, T_2) \rightarrow c_2(q_3), \ldots, (q_{m}, T_m) \rightarrow c_m(q_{m+1}), (q_{m+1}, b) \rightarrow b(q_{m+2})\}.
\]
Here, the rules in $R_2$ are applied to the nodes from the root to the $b$ node (the parent of the leaf $c$), and either a rule in $R_3$ 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 $C_i = (l_{i_1} \land l_{i_2} \land \ldots \land l_{i_n})$ 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 $i_1 < i_2 < i_3$. Then $pat_i$ is defined as follows.

$$
\begin{align*}
\forall i_1 < i_2 < i_3, & \\
\text{if } & l_{i_1} = x_{i_1}, \\
\text{then } & l_{i_2} = x_{i_2}, \\
\text{and } & l_{i_3} = x_{i_3}, \\
\text{where } & L_i = \{ T_{ij} \}.
\end{align*}
$$

We show that $\phi$ is a non-tautology iff the rule $(q_{p_{12}}, c) \rightarrow c$ of $R'_2$ is applied to $c^{\phi\downarrow}$.

$(\Rightarrow)$ Assume that $\phi$ is a non-tautology. Then there is a truth assignment of $x_1, x_2, \ldots, x_n$ such that $\phi = \text{false}$. Let $r$ be the tree representing this truth assignment. Since $C_i = \text{false}$ for every $1 \leq i \leq n$, none of $pat_i, pat_2, \ldots, pat_n$ matches the node $c$ in $r$, thus no rule in $R_3$ 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 $x_1, x_2, \ldots, x_n$, $\phi = \text{true}$. Therefore, for any truth assignment of $x_1, x_2, \ldots, x_n$, $C_i = \text{true}$ for some $i$. Thus, for any tree $r$ valid to $D$, some rule in $R_3$ is applied to the $c$ node in $r$ (since $|pat_i| \geq 2$ for any $i$, the rule in $R'_2$ is not applied to $c^{\phi\downarrow}$).

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]


