An Algorithm for Detecting and Correcting XSLT Rules Affected by Schema Updates

WU YANG

Graduate School of Library, Information and Media Studies
University of Tsukuba

March 2017

Contents

Chapter 1

Introduction

DTDs are continuously updated according to changes in the real world. Updates to a DTD affect the behavior of XSLT stylesheets as well as XML documents under the DTD. To maintain the consistencies of XSLT stylesheets with an updated DTD, we have to detect the XSLT rules affected by DTD updates and correct the affected XSLT rules so that the XSLT stylesheets transform documents under the updated DTD appropriately. However, correcting such affected XSLT rules manually are a highly difficult and time-consuming task due to the following reasons.

- Recent DTDs are becoming larger and more complex. In [6] 27 real-world DTDs are investigated and the average number of rules turns out to be more than 50.
- XSLT is complex especially for unskilled users, and writing an XSLT stylesheet is an expert task [2].
- Users do not always fully understand the dependencies between XSLT stylesheets and old/updated DTDs.

To address this problem, we propose an algorithm for correcting XSLT rules affected by DTD updates.

To illustrate our algorithm, let us consider the fragments of a DTD and an XSLT stylesheet shown in Fig. 1.1(A,B). With `DTD_old` and `XSLT_old`, an `sns` element is processed by rule 1 if the `sns` element is not a child of `affiliation`. On the other hand, an `sns` element that is a child of `affiliation` is processed by rule 3. Here, suppose that `contact` is nested between `affiliation` and `email`/`sns`. That is, `contact` is inserted as a child of `affiliation`, and `email` and `sns` are moved as children of `contact` as shown in `DTD_new` (Fig. 1.1(C)). Due to this DTD update, The rule applied to the `sns` element of an `affiliation` element is changed; in `DTD_old`, rule 3 is applied to the `sns` element as above, while in `DTD_new` rule 1 is applied to the `sns` element. In this situation, there are two possible choices for correcting XSLT rules.

1. Accept the above change as it is and do nothing.
2. Modify `XSLT_old` so that rule 3 is applied to the `sns` element of an `affiliation` element in

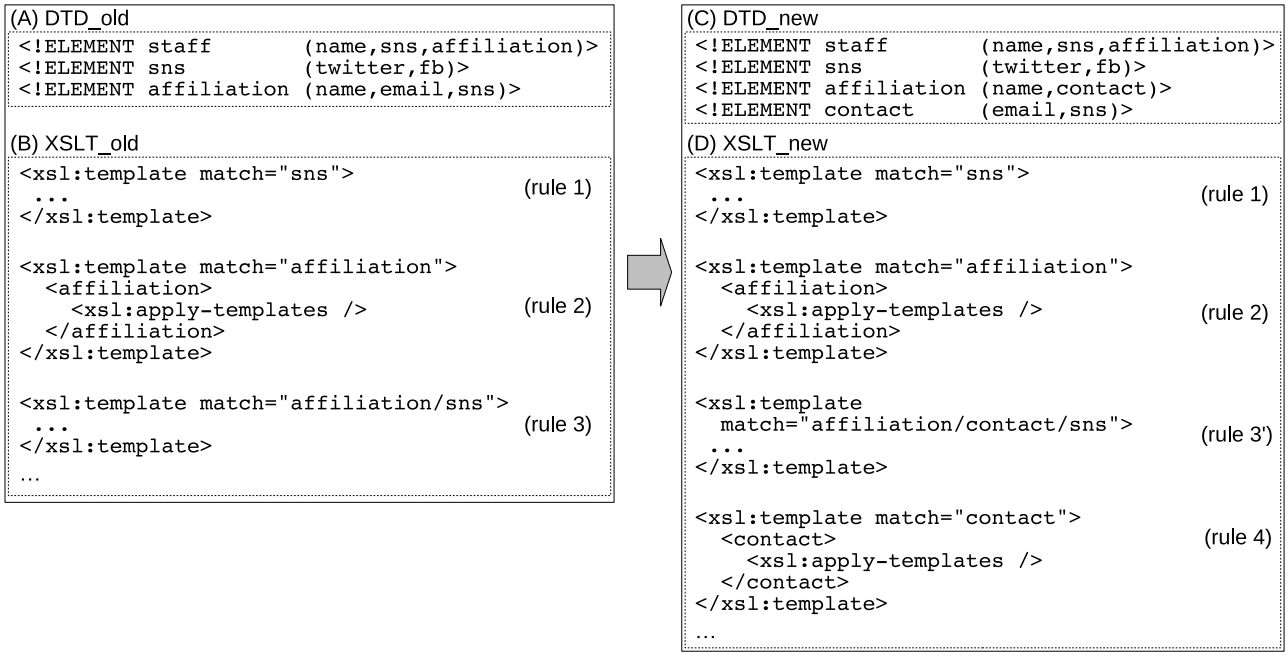

Figure 1.1 Old/new DTDs and XSLTs

`DTD_new`.

For the latter choice, our algorithm suggests rule 3' and rule 4 in Fig. 1.1(D). Here, rule 3' is obtained by modifying the pattern of rule 3 and rule 4 is newly added to `XSLT_old` in order to "relay" the transformation of rule 2 to rule 3'. Such suggested rules are possibly desirable correct rules, or even if it is not the case, at least useful hints to correct rules affected by DTD updates.

In this thesis, we first give an algorithm for detecting XSLT rules affected by DTD updates, which is an extension of the algorithm previously proposed in [13]. Based on the result, we propose an algorithm for correcting rules affected by DTD updates. We also give the result of a preliminary experiment.

Related Work

The algorithm in [4] transforms XPath expressions according to a schema update. Although XPath expressions are used as XSLT patterns, their algorithm cannot be applied to our problem. This is because XSLT rules affected by a schema update cannot be detected by checking each XSLT pattern independently, since XSLT rules may depend on each other as shown in dependency graph. To the best of our knowledge, there is no study on correcting XSLT rules affected by a schema update.

On the other hand, there are several studies dealing with XML schema updates. For example, [8, 5] propose algorithms for extracting "diff" between two schemas. [3, 12] propose update operations that assures any updated schema contains its original schema so that documents under an original schema remains valid under its updated schema. [11] introduces a taxonomy of possible problems for XQuery

induced by a schema update, and gives an algorithm to detect such problems. [7] studies query-update independence analysis, and shows that the performance of [1] can be drastically enhanced in the use of μ-calculus.

In [13] the authors propose an algorithm for detecting rules affected by DTD updates. The paper only considers detecting rules affected by DTD updates and does not consider correcting such rules. Moreover, [13] assumes one-to-one correspondence between elements in old/new DTDs. However, this may be too restrictive in real-world situations, since as shown in Chapter 4 this assumption does not always hold due to nest/unnest operations.

Chapter 2

Preliminaries

In this chapter, we first give some definitions related to DTD. Then we define tree transducers, a formal model of XSLT.

2.1 DTD and Update Operations to DTDs

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

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

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

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

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 [10], a decimal order like 1.3.2. For example, Fig. 2.1 shows the tree structure of $r = (a|b)(ca)^*$, where each node is associated with its position. For a regular expression r, the label at position u in r is denoted $l(r, u)$ and the subexpression at position u of r is denoted $sub(r, u)$. For example, in Fig. 2.1, $l(r, 1) =$ '|', $l(r, 1.1) = a$, $sub(r, 2.1) = ca$.

We define some operators to positions.

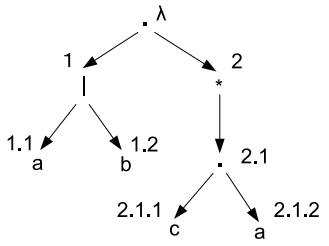

Figure 2.1 Tree structure of r

- For positions u and v, by $u + v$ we mean the position $u'v'$, where u' is the position obtained from u by deleting the rightmost number n of u and v' is obtained by adding n to the leftmost number of v. For example, if $u = 1.3.2$ and $v = 2.1.1$, then $u + v = 1.3.4.1.1$.
- For a position u and an integer i, by $inc(u, i)$ we mean the position obtained by incrementing the i-th number of u. For example, if $u = 2.1.1$, then $inc(u, 1) = 2.1.2$ and $inc(u, 2) = 2.2.1$.
- For a position u, by $len(u)$ we mean the *length* of u. For example, if $u = 2.1.1$, then $len(u) = 3$.

Let $D = (d, sl)$ be a DTD. We have the following four *update operations* to D.

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

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

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

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

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

example, $(q, a/b/c) \to c(p)$ corresponds to the following XSLT template.

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

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

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

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

The transformation of t by Tr, denoted by $Tr(t)$, is defined as $Tr(t) = Tr^{q_0}(t, v_0)$, where v_0 is the root node of t. The class of the tree transducers defined above is denoted UTT^{pat}. In particular, if for every rule $(q, pat) \to h \in R$ pat is a single label, then the restricted class is denoted UTT, which coincides with that of the standard unranked tree transducer[9].

2.3 Class $\mathrm{UTT}^{pat,sel}$

We first show the definitions of XPath location paths used in select. A *relative location path* is defined as $ls_1/\cdots/ls_n$, where $ls_i = ax_i :: l_i, ax_i \in \{\downarrow, \uparrow, \downarrow^*\}$, l_i is a label, and $\uparrow$ is a parent axis. An *absolute location path* consists of '/' optionally followed by a relative location path. The set of relative location paths and absolute location paths is denoted by SEL. By $H_\Sigma(Q \times SEL)$ we mean the set of hedges such that leaf nodes can be labeled with elements from $(q, sel) \in Q \times SEL$.

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

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

- The case where there is a rule $(q, pat) \to h \in R'$ such that $M_{pat}(t, v, pat) \neq \emptyset$:
  - If sel is a relative location path, then $Tr^q(t, v)$ is obtained from h as follows:
    - for each leaf node u in h, if $l(u) = (p, sel) \in Q \times SEL$, then replace u with hedge $Tr^p(t, v_1) \cdots Tr^p(t, v_n)$, where $S(t, v, sel) = \{v_1, \cdots, v_n\}$.

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

The transformation of t by Tr, denoted by $Tr(t)$, is defined as $Tr^{q0}(t, v_0)$, where v_0 is the root node of t. The class of the tree transducers defined above is denoted $\mathrm{UTT}^{pat,sel}$.

Chapter 3

Rules Affected by DTD Updates

In this chapter, we firstly define 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 labels is $a^{root,\lambda}$. By $D_{\#}$ we mean the *superscripted DTD* of D that is obtained by replacing each label in a content model with its corresponding superscripted label.

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

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

We say that a superscripted label $a^{b,u}$ is *reachable* from the start label sl if $a^{b,u} = sl^{root,\lambda}$ or for some superscripted label $c^{d,v}$ $c^{d,v}$ is reachable from sl and $a^{b,u}$ occurs in $d_{\#}(c)$.

For a tree t valid against D, $t_{\#}$ is a *superscripted tree* of t if $t_{\#}$ is obtained by replacing each label in t with its corresponding superscripted label so that $t_{\#}$ is valid against $D_{\#}$ (see Fig. 3.1).

Let $D = (d, sl)$ 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', and we say that $e^{f',w'}$ *corresponds* to $e^{f,w}$ (If no update script between old and new DTDs is given, the algorithms in [8, 5] can generate update scripts between DTDs). Note that more than one superscripted label in $s(D)_{\#}$ may correspond to $e^{f,w}$, as shown below.

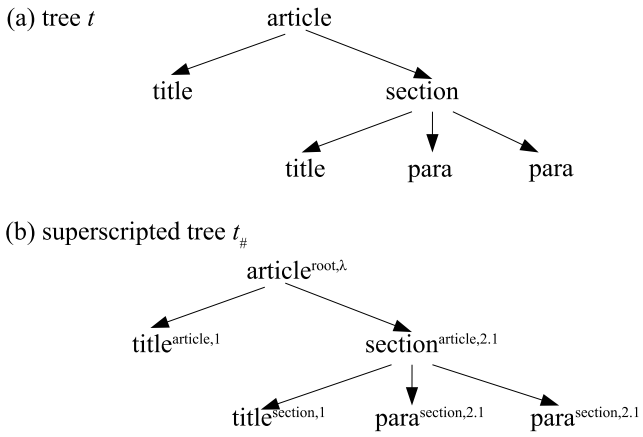

Figure 3.1 Tree t and its superscripted tree $t_{\#}$

In the following, we formally define the set of superscripted labels corresponding to $e^{f,w}$ w.r.t. edit script. The set of superscripted labels corresponding to $e^{f,w}$ w.r.t. an edit script s, denoted $C_D(e^{f,w}, s)$, is defined as follows. First, if $s = \epsilon$, then $C_D(e^{f,w}, s) = \{e^{f,w}\}$ for every superscripted label $e^{f,w}$ in $D_{\#}$. Consider next the case where $s = s'op$, where s' is an edit script and op is an edit operation. Let $s'(D) = (d', sl)$. We have the following four cases according to op.

- The case where $op = ins\_elm(a, b, u)$: Since $b^{a,u}$ is inserted in $d'_{\#}(a)$, the positions of the right siblings of $b^{a,u}$ and their descendants are incremented. Let $RS(b^{a,u})$ be the set of superscripted labels that are right siblings of $b^{a,u}$ or their descendants in $d'_{\#}(a)$. Then $C_D(e_{f,w}, s)$ is obtained from $C_D(e^{f,w}, s')$ by replacing each $c^{a,v} \in RS(b^{a,u}) \cap C_D(e^{f,w}, s')$ with $c^{a,inc(v,len(u))}$.
- The case where $op = del\_elm(a, u)$: Let $b^{a,u}$ be the deleted element in $d'_{\#}(a)$ by op. Then $b^{a,u}$ and its descendants become unreachable from the start label. Thus, we define $C_D(e^{f,w}, s) = C_D(e^{f,w}, s') \setminus Desc(b^{a,u})$, where $Desc(b^{a,u})$ is the set of superscripted labels in $s'(D)_{\#}$ that become unreachable from $sl^{root,\lambda}$ by deleting $b^{a,u}$.
- The case where $op = nest(a, b, u)$: $b^{a,u}$ is inserted between a and $sub(d'_{\#}(a), u)$, and thus each superscripted label $c^{a,v}$ in $sub(d'_{\#}(a), u)$ is changed to $c^{b,v'}$ with $uv' = v$. Thus, $C_D(e^{f,w}, s)$ is obtained from $C_D(e^{f,w}, s')$ by replacing each $c^{a,v} \in lab(sub(d'_{\#}(a), u)) \cap C_D(e^{f,w}, s')$ with $c^{b,v'}$, where v' is a position with $uv' = v$.
- The case where $op = unnest(a, u)$: Let $b^{a,u}$ is the label to be unnested. By op (1) $b^{a,u}$ is deleted from $C_D(e^{f,w}, s')$ and (2) each superscripted label $c^{b,v}$ in $d'_{\#}(b)$ is added to $d'_{\#}(a)$ as $c^{a,u+v}$. Let $C' = C_D(e^{f,w}, s') \setminus \{b^{a,u}\}$. We have two cases to be considered.
  - The case where $b^{a,u}$ is the only superscripted label of b in $s'(D)_{\#}$: $C_D(e^{f,w}, s)$ is obtained from C' by replacing each $c^{b,v} \in lab(d'_{\#}(b)) \cap C'$ with $c^{a,u+v}$.
  - The case where $s'(D)_{\#}$ contains more than one superscripted label of b: In this case, $b^{a,u}$ disappears from $s'(D)_{\#}$ by op but the other superscripted labels of b still exist (thus

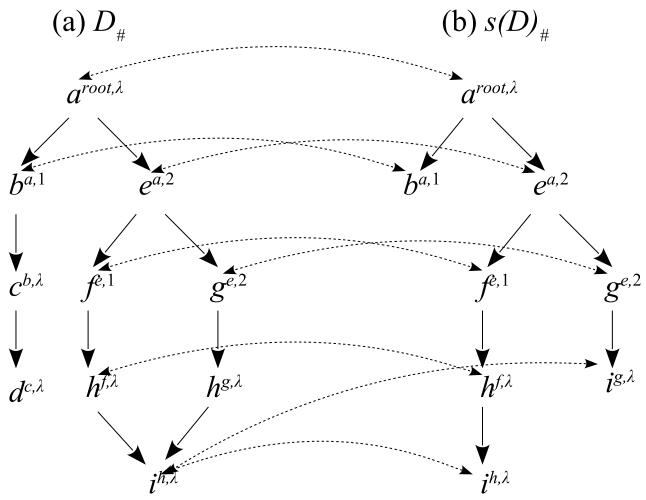

Figure 3.2 The correspondence between elements in $D_{\#}$ and $s(D)_{\#}$.

$c^{b,v}$ also exists). Thus, $C_D(e^{f,w}, s)$ is obtained by adding $c^{a,u+v}$ to C' for each $c^{b,v} \in lab(d'_{\#}(b)) \cap C'$.

If $a_{b',u'} \in C_D(a_{b,u}, s)$, then we say that $a_{b,u}$ *corresponds* to $a_{b',u'}$ and $a_{b',u'}$ *corresponds* to $a_{b,u}$.

Example 3 Let $D = (d, a)$ be a DTD, where $d(a) = be$, $d(b) = c$, $d(c) = d$, $d(e) = fg$, $d(f) = d(g) = h$, $d(h) = i$, $d(d) = d(i) = \epsilon$. Figure 3.2(a) illustrates $D_{\#}$. Each edge in the figure represents a parent-child relationship between superscripted labels. Let $s = del\_elm(b, \lambda)unnest(g, \lambda)$. Then $s(D) = (d', a)$, where $d'(a) = be$, $d'(e) = fg$, $d'(f) = h$, $d'(g) = d'(h) = i$, $d'(b) = d'(i) = \epsilon$. Figure 3.2(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 $Tr = (Q, \Sigma, q_0, R)$ be a tree transducer. For a superscripted label $a^{b,u}$ and a rule $rl \in R$, rl is *applicable* to $a^{b,u}$ in a tree t if for some node v in t, (1) rl is applied to v (i.e., the antecedent of rule R1 holds for rl and v) during the transformation 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 $a^{b,u}$ in $D_{\#}$ but

becomes applicable to $a^{b',u'}$ in $s(D)_{\#}$.

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

Chapter 4

NP-Hardness of Applicability of Transformation Rules

In this chapter, we show that whether a transformation rule is applicable at a subscripted label is NP-hard.

Theorem: Let D be a DTD, $Tr = (Q, \Sigma, q_0, R)$ be a tree transducer in $\mathrm{UTT}^{pat,sel}$. Then determining whether there is a rule $rl \in R$ applicable at $a_{b,u}$ in $D_{\#}$ is NP hard.

Proof: We reduce the 3SAT problem to the above problem. The 3SAT problem is defined 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 variables appearing ϕ.
- Problem: Determine whether there is an assignment of boolean values to $x_1, x_2, \cdots, x_m$ that satisfy ϕ.

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

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

where,

- c_i is a label representing clause C_i,
- $T_i (1 \leq i \leq m)$ lists "labels representing clauses which contain positive literal x_i" ,
- F_i $(1 \leq i \leq m)$ lists "labels representing clauses which contain negative literal $\neg x_i$" .

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

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

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

Without loss of generality, we assume that c_n appears at the head position of T_1. Then, let $(c_n)_{s,21}$ be the subscripted label of c_n, which appears at the head position of T_1. In the following, we will show that rule $(q_n, c_n) \to c'_n$ is applicable to $(c_n)_{s,21}$ if and only if ϕ is satisfiable.

($\Rightarrow$) Assume that rule $(q_n, c_n) \to c'_n$ is applicable to $(c_n)_{s,21}$. From the left-hand side of the rule, it means that c_n is assigned with state q_n, therefore rule $(q_{n-1}, c_{n-1}) \to c'_{n-1}(q_n, /s/c_n)$ needs to be applicable to c_{n-1}. Similarly, we can know that for each $i = n-1, \cdots, 2$, c_i is assigned with state q_i, therefore rule $(q_{i-1}, c_{i-1}) \to c'_{i-1}(q_i, /s/c_i)$ needs to be applicable to c_{i-1}. Consequently, there is a tree valid to D such that s has child elements $c_1, c_2, \cdots, c_n$. Let t be this tree, let S be the list of child elements of s in t. For every $1 \leq i \leq m$, if S contains T_i, $x_i = true$, if S contains F_i, $x_i = false$. It is obvious that this assignment of boolean values makes ϕ satisfiable.

($\Leftarrow$) Assume that ϕ is satisfiable. Then, there is an assignment of boolean values which makes $\phi = true$. Here, a sequence belongs to $L(d(s))$ should satisfy the following requirements.

- For every $1 \leq i \leq m$, if $f(x_i) = true$, it contains T_i, if $f(x_i) = false$, it contains F_i.

In this case, there exists a tree t valid to D such that t has all of $c_1, c_2, \cdots, c_n$ as child elements of s. For t, from the definition of Tr, it is easily shown that rule $(q_n, c_n) \to c'_n$ is applicable to $(c_n)_{s,21}$. □

In the following, we assume that a tree transducer is in UTT^{pat}.

Chapter 5

Detecting Rules Affected by DTD Updates

In this chapter, we present an algorithm for obtaining $R^+(a^{b,u}, a^{b',u'})$ and $R^-(a^{b,u}, a^{b',u'})$. Assuming this algorithm, in the next chapter we present algorithms for correcting rules in $R^+(a^{b,u}, a^{b',u'})$ and $R^-(a^{b,u}, a^{b',u'})$.

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 pair $(a^{b,u}, q)$ is a node in a dependency graph and means that state q is assigned to $a^{b,u}$. Consider rule R1 in the definition of tree transducer, and suppose that the antecedent of rule R1 holds for a node v with $l(v) = a^{b,u}$. This means that rule $(q, pat) \to h$ 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 $D_\# = (d_\#, a^{root,\lambda})$ 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. 5.1.

To define dependency graph formally, we need preliminary definitions. By $St(h)$ we mean the set of states in a hedge h. For example, if $h = a(pq)$, then $St(h) = \{p, q\}$. We say that a path $(a_1^{a_0,u_1}, q_1) \leftarrow (a_2^{a_1,u_2}, q_2) \leftarrow \cdots \leftarrow (a_n^{a_{n-1},u_n}, q_n)$ *matches* a pattern $pat = l_1/l_2/\cdots/l_n$ if $a_i = l_i$ for every $1 \le i \le n$. $(a_n^{a_{n-1},u_n}, q_n)$ is called the *source* of the path and $(a_1^{a_0,u_1}, q_1)$ is called the *target* of the path.

Now we define the dependency graph. Let $D = (d, sl)$ be a DTD and $Tr = (Q, \Sigma, q_0, R)$ be a tree transducer. Then the *dependency graph* of D and Tr is a graph $G_D = (V_D, E_D)$ satisfying the

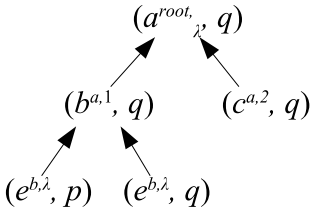

Figure 5.1 An example of dependency graph

following conditions.

- $(sl^{root,\lambda}, q_0) \in V_D$. $(sl^{root,\lambda}, q_0)$ is called the *root* of G_D.
- If there is a rule $(q, pat) \to h \in R$ satisfying the following (1) and (2), then $(c^{a,v}, p) \in V_D$ and $(a^{b,u}, q) \leftarrow (c^{a,v}, p) \in E_D$.
  1. E_D contains a path that matches pat.
  2. Let $(a^{u,v}, q')$ be the source of the path of (1). Then the rightmost label of pat is a, $q = q'$, $p \in St(h)$, and $d_\#(a)$ contains $c^{a,v}$.
- G_D contains no nodes and edges that do not satisfying the above conditions.

The following algorithm 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 constructed 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 $R^-(a^{b,u}, a^{b',u'})$.

Algorithm FindAffectedRules

Input: DTD $D = (d, sl)$, update script s to D, tree transducer $Tr = (Q, \Sigma, q_0, R)$.
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. Construct the dependency graph G_D of D and Tr.
2. Construct the dependency graph G'_D of $s(D)$ and Tr.
3. **for** each pair $(a^{b,u}, a^{b',u'})$ such that $a^{b',u'} \in C_D(a^{b,u}, s)$ **do**
4. $\quad M \leftarrow \{rl \in R \mid rl$ is applicable to $(a^{b,u}, q)$ in G_D, $q \in Q\}$
5. $\quad M' \leftarrow \{rl \in R \mid rl$ is applicable to $(a^{b',u'}, q)$ in G'_D, $q \in Q\}$
6. $\quad R^-(a^{b,u}, a^{b',u'}) \leftarrow M \setminus M'$
7. **return** $\{R^-(a^{b,u}, a^{b',u'}) \mid a^{b',u'} \in C_D(a^{b,u})\}$

Let $M_c = \max_{a \in \Sigma} |d_\#(a)|$ and $M_s = \max_{(q,pat) \to h \in R} |St(h)|$, where $|d_\#(a)|$ denotes the number of occurrences of superscripted labels in $d_\#(a)$. Then FindAffectedRules runs in $O(|R|^5 (M_c M_s)^3)$ (details are omitted).

Chapter 6

Correcting Rules Affected by DTD updates

In this chapter, we give algorithms for correcting rules affected by DTD updates. In the following, for simplicity we assume that for any rule $(q, pat) \to h$, h contains at most one state. However, this restriction can easily be relaxed.

We have the following three cases to be considered.

1. Correction for rules in $R^-(a^{b,u}, a^{b',u'})$
2. Correction for rules in $R^+(a^{b,u}, a^{b',u'})$
3. Generating rules to newly added elements

In the following, we mainly focus on Case 1. The other cases are explained briefly.

Case 1

This case is dealt with the following two steps.

1-a. Generating new rules to fix "path inconsistencies."
1-b. Correcting patterns in rules.

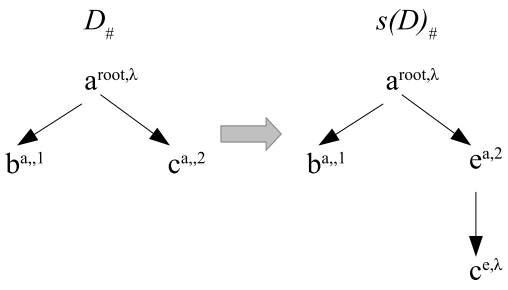

Figure 6.1 Example of nest

First, let us consider step 1-a. For a rule rl applied to $a^{b,u}$, if a label is nested/unnested between some ancestors of $a^{b,u}$, then rl may become inapplicable to $a^{b',u'}$. For example, consider $D_{\#}$ (Fig. 6.1(left)) and tree transducer $Tr = (Q, \Sigma, q, R)$, where $R = \{(q, a) \to a(q), (q, c) \to c\}$. Suppose that $e^{a,2}$ is inserted between $a^{root,\lambda}$ and $c^{a,2}$ by $nest\_elm$, as shown in Fig. 6.1(right). Then $rl = (q, c) \to c$ cannot be applied to $c^{e,\lambda}$ in $s(D)_{\#}$ since there is no rule applied to $e^{a,2}$. To recover rl, a rule applied to $e^{a,2}$, e.g., $(q, e) \to e(q)$, have to be added to Tr. We give an algorithm for generating such rules later.

Let D be a DTD, s be an edit script to D, $Tr = (Q, \Sigma, q_0, R)$ be a tree transducer, and $rl = (q, pat) \to h$ be a rule in R. Moreover, let G_D be the dependency graph of D and Tr and G'_D be the dependency graph of $s(D)$ and Tr. To generate rules to fix inconsistencies as above, we compare paths in G_D and those of G'_D and detect labels to which new rules should be applied. Let

$$p = (a_1^{a_0,u_1}, q_1) \leftarrow (a_2^{a_1,u_2}, q_2) \leftarrow \cdots \leftarrow (a_n^{a_{n-1},u_n}, q_n)$$

be a path in G_D. By $ls(p)$, we mean the sequence of labels on p, that is,

$$ls(p) = a_1^{a_0,u_1}\, a_2^{a_1,u_2}\, \cdots\, a_n^{a_{n-1},u_n}.$$

For a sequence $ls' = b_1^{b_0,v_1}\, b_2^{b_1,v_2}\, \cdots\, b_m^{b_{m-1},v_m}$ of labels in $s(D)_{\#}$, if $b_{i+1}^{b_i,v_{i+1}}$ occurs in $d_{\#}(b_i)$ for every $1 \leq i \leq m-1$ and $b_i^{b_{i-1},v_i} \neq b_j^{b_{j-1},v_j}$ for any $i \neq j$, then ls' is called a *parent-child chain* in $s(D)_{\#}$. Let

$$c(a_i^{a_{i-1},u_i}, ls') = \begin{cases} 1 & \text{if } b_j^{b_{j-1},v_j} \in C_D(a_i^{a_{i-1},u_i}, s) \text{ for some } j, \\ 0 & \text{otherwise.} \end{cases}$$

Then the *correspondence size* between $ls(p)$ and ls', denoted $cs(ls(p), ls')$, is defined as follows.

$$cs(ls(p), ls') = \sum_{i=1}^{n} c(a_i^{a_{i-1},u_i}, ls').$$

Intuitively, $cs(ls(p), ls')$ represents the "similarly" between $ls(p)$ and ls'.

We say that ls' *matches* $ls(p)$ if $b_1^{b_0,v_1}$ corresponds to $a_i^{a_{i-1},u_i}$ for some $i \geq 1$ and $b_m^{b_{m-1},v_m}$ corresponds to $a_n^{a_{n-1},u_n}$. Then a parent-child chain ls' in $s(D)_{\#}$ is *maximum* w.r.t. $ls(p)$ if ls' matches $ls(p)$ and $cs(ls(p), ls') \geq cs(ls(p), ls'')$ for any parent-child chain ls'' in $s(D)_{\#}$ that matches $ls(p)$.

We now give an algorithm that generates, for a rule $rl = (q, pat) \to h \in R^{-}(a^{b,u}, a^{b',u'})$, rules to fix path inconsistency w.r.t. rl. First, we find a path p in G_D such that the target node of p is the root of G_D and that a prefix of p matches pat (line 2). Then we find a parent-child chain ls' in $s(D)_{\#}$ that is maximum w.r.t. $ls(p')$ (line 3), and assign a state q'_i for each node $b_i^{b_{i-1},v_i}$ in ls' (lines 4 to 9). Then we find pairs $((b_{i-1}^{b_{i-2},v_{i-1}}, q'_{i-1}), (b_i^{b_{i-1},v_i}, q'_i))$ such that $(b_{i-1}^{b_{i-2},v_{i-1}}, q'_{i-1}) \leftarrow (b_i^{b_{i-1},v_i}, q'_i))$ is missing in G'_D, and generate new rules so that the missing edges are recovered, as follows. If there is a rule applicable to $(b_{i-1}^{b_{i-2},v_{i-1}}, q'_{i-1})$, then the rule is used as a template of a new rule to recover the edge (lines 12 to 14). Otherwise, a new simple rule is generated (lines 15 to 16).

Algorithm FixPathInconsistency

Input: DTD D, edit script s for D, tree transducer $Tr = (Q, \Sigma, q_0, R)$, the dependency graph G_D of D and Tr, the dependency graph G'_D of $s(D)$ and Tr, rule $(q, pat) \to h \in R^-(a^{b,u}, a^{b',u'})$.
Output: Set of rules to fix path inconsistency w.r.t. $(q, pat) \to h$.

1. $Result \leftarrow \emptyset$.
2. Find a simple path $p = (a_1^{root,\lambda}, q_1) \leftarrow (a_2^{a_1,u_2}, q_2) \leftarrow \cdots \leftarrow (a_n^{a_{n-1},u_n}, q_n)$ in G_D satisfying the following.
   - $(a_1^{root,\lambda}, q_1)$ is the root of G_D,
   - $q_n = q$, and
   - a prefix of p matches pat.
3. Find a parent-child chain $ls' = b_1^{root,\lambda}\, b_2^{b_1,v_2} \cdots b_m^{b_{m-1},v_m}$ in $s(D)_{\#}$ such that ls' is maximum w.r.t. $ls(p')$.
4. **for** $i = 1, 2, \cdots, m-1$ **do**
5. **if** $b_i^{b_{i-1},v_i}$ corresponds to $a_j^{a_{j-1},u_j}$ for some j **then**
6. $q'_i \leftarrow q_j$
7. **else**
8. $q'_i \leftarrow q'_{i-1}$ // use the same state as its parent
9. $q'_m \leftarrow q$ // the last state is q
10. **for** $i = 2, 3, \cdots, m$ **do**
11. **if** edge $(b_{i-1}^{b_{i-2},v_{i-1}}, q'_{i-1}) \leftarrow (b_i^{b_{i-1},v_i}, q'_i)$ is missing in G'_D **then**
12. **if** there is a rule $(q'_{i-1}, pat') \to h' \in R$ applicable to $(b_{i-1}^{b_{i-2},v_{i-1}}, q'_{i-1})$ in G'_D such that $St(h') \neq \emptyset$ **then**
13. Let q' be the state in h'. Construct a hedge h'' from h' by replacing q' with q'_i.
14. Let $rl' = (q'_{i-1}, pat') \to h''$.
15. **else**
16. Let $rl' = (q'_{i-1}, b_{i-1}) \to b_{i-1}(q'_i)$.
17. Add $(b_{i-1}^{b_{i-2},v_{i-1}}, rl')$ to $Result$.
18. **return** $Result$.

The rules in $Result$ are presented to a user and the user selects rules he/she wants to use.

Consider next step 1-b. Even with the rules obtained in step 1-a, some rules in $R^-(a^{b,u}, a^{b',u'})$ still cannot be recovered. This is caused by patterns of some rules becoming inconsistent with updated DTDs. For example, consider $D_{\#}$ (Fig. 6.2(left)) and a tree transducer $Tr = (Q, \Sigma, q, R)$, where $R = \{(q, a) \to a(q), (q, b) \to b(q), (q, c) \to c(q), (q, b/c/f) \to f, (q, a/c/f) \to f'\}$. Suppose that element $c^{b,2}$ is unnested, as shown in Fig. 6.2(right). Then $rl = (q, b/c/f) \to f$ becomes inapplicable to $f^{b,\lambda}$. The algorithm presented later corrects the pattern $b/c/f$ of rl and we obtain $(q, b/f) \to f$, which is applicable to $f^{b,\lambda}$.

We give a definition. We say that $e^{f,v}$ is the *nearest corresponding ancestor* of $a^{b,u}$ w.r.t. $a^{b',u'}$ (see Fig. 6.3) if

- $e^{f,v}$ is an ancestor of $a^{b,u}$ in $D_{\#}$, and
- There exists a label $e^{f',v'} \in C_D(e^{f,v}, s)$ such that $e^{f',v'}$ is an ancestor of $a^{b',u'}$ in $s(D)_{\#}$ and

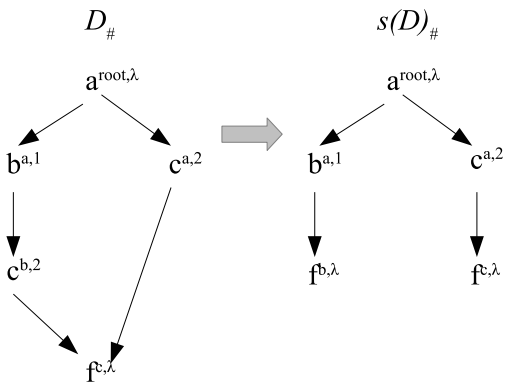

Figure 6.2 Example of unnest

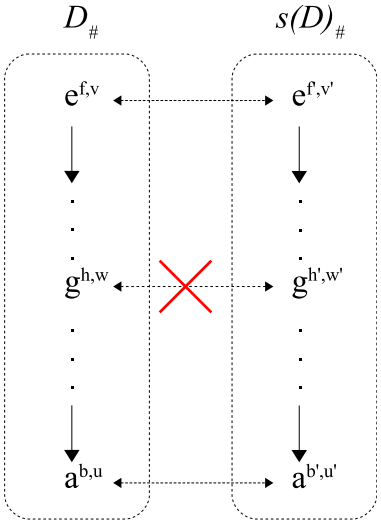

Figure 6.3 Nearest corresponding ancestor $e^{f,v}$ of $a^{b,u}$ w.r.t. $a^{b',u'}$

that no label between $a^{b',u'}$ and $e^{f',v'}$ corresponds to any label between $a^{b,u}$ and $e^{f,v}$.

We show an algorithm for correcting the pattern of a rule $rl = (q, pat) \rightarrow h \in R^{-}(a^{b,u}, a^{b',u'})$ such that rl is still inapplicable even with the rules obtained in step 1-a. We first find the nearest corresponding ancestor $e^{f,v}$ of $a^{b,u}$ w.r.t. $a^{b',u'}$ (line 2). Then we find the set NA of labels $g^{h,w}$ such that $g^{h,w}$ is the nearest corresponding ancestor of $a^{b,u}$ w.r.t. $a^{b'',u''}$, where $a^{b'',u''}$ corresponds to $a^{b,u}$ but $a^{b'',u''} \neq a^{b',u'}$ (line 3). Then modify rl as follows. We first find paths p in G_D such that p matches pat (excluding paths containing labels in NA to avoid paths irrelevant to $e^{f,v}$) and that p may have a corresponding path in G'_D (line 5). For each path p obtained above, we find a maximum parent-child chain ls' in $s(D)_{\#}$ w.r.t. $ls(p)$ (line 7), and modify pat of rl according to the correspondence between $ls(p)$ and ls' (lines 8 to 9).

Algorithm CorrectPattern

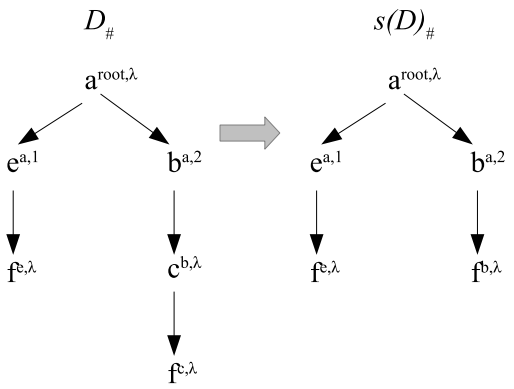

Figure 6.4 Example of unnest

Input: DTD $D = (d, sl)$, edit script s for D, tree transducer $Tr = (Q, \Sigma, q_0, R)$, the dependency graph G_D of D and Tr, rule $rl \in R^-(a^{b,u}, a^{b',u'})$.
Output: Correction for rl.

1. $Result \leftarrow \emptyset$
2. Find the nearest corresponding ancestor $e^{f,v}$ of $a^{b,u}$ w.r.t. $a^{b',u'}$.
3. $NA \leftarrow \{g^{h,w} \mid g^{h,w}$ is the nearest corresponding ancestor of $a^{b,u}$ w.r.t. $a^{b'',u''}$, $a^{b'',u''} \in C_D(a^{b,u}, s) \setminus \{a^{b',u'}\}\}$.
4. Let $rl = (q, pat) \rightarrow h$.
5. Let P be the set of path p in G_D such that p matches pat, the source of p is $(a^{b,u}, q)$, and that p does not contain any label in NA.
6. **for each** path $p \in P$ **do**
7. Let ls' be the maximum parent-child chain w.r.t. $ls(p)$ in $s(D)_{\#}$.
8. Modify pat according to the correspondence between $ls(p)$ and ls', as follows. Let pat' be the result.
   - For each $c^{e,w}$ in $ls(p)$, if ls' has no label corresponding to $c^{e,w}$, then delete the label corresponding to $c^{e,w}$ from pat.
   - For each $c^{e',w'}$ in ls', if $ls(p)$ has no label corresponding to $c^{e',w'}$, then insert e into pat at the corresponding position of $c^{e',w'}$ in ls'.
9. Let $rl' = (q, pat') \rightarrow h$.
10. Add (rl, rl') to $Result$.
11. **return** $Result$

Result is presented to a user and the user selects an appropriate rule from *Result*.

Case 2

Suppose that $rl \in R^+(a^{b,u}, a^{b',u'})$. We have the following choices.

- Accept this as it is and do nothing.
- Modify rl so that rl is not applicable to $a^{b',u'}$ while rl is kept applicable to any other labels to which rl is applicable before the update.

For the latter choice, our algorithm modifies rl. We explain this by a simple example. Consider $D_{\#}$ (Fig. 6.4(left)) and a tree transducer $Tr = (Q, \Sigma, q, R)$, where $R = \{(q, a) \to a(q), (q, e) \to e(q), (q, b) \to b(q), (q, c) \to c(q), (q, f) \to f, (q, c/f) \to f'\}$. Suppose that $c^{b,\lambda}$ is unnested (Fig. 6.4(right)). Let $rl = (q, f) \to f$. Then rl becomes applicable to $f^{b,\lambda}$ and thus we have $rl \in R^+(f^{c,\lambda}, f^{b,\lambda})$. Our algorithm modifies rl to $rl' = (q, e/f) \to f$ so that rl' is not applied to $f^{b,\lambda}$ while rl' remains applicable to $f^{e,\lambda}$.

Case 3

Suppose that a new label l not in D is inserted by ins_elm. Since l is not in D, there is no rule applicable to l. In such a case, our algorithm generate a new rule applicable to l, similarly to line 16 of FixPathInconsistency. For example, suppose that l is inserted as a child of a, and we have a rule $(q, pat) \to h$ applicable to a such that h contains a state, say q'. Then the algorithm generates a new rule $(q', l) \to l(q')$ that is applicable to l.

Chapter 7

Experiment

In this chapter, we present experimental results on our algorithm. We implemented our method in Java 1.8.0. We use two pairs of schemas, MSRMEDOC DTDs (version 2.1.1 and 2.2.2)[*1] and the NLM Journal Publishing Tag Set Tag Library DTDs (version 2.3 and 3.0)[*2]. MSRMEDOC DTD is a format for information interchange in the development process of production and supply. The NLM Journal Publishing Tag Set Tag Library DTD describes the content and metadata of journal articles, including research and non-research articles, letters, editorials, and book and product reviews.

Firstly, we give the evaluation of our algorithm on MSRME-DOC DTDs. In the following, let D_{211} be the version 2.1.1 MSRMEDOC DTD and D_{222} be the version 2.2.2 MSRMEDOC DTD. The numbers of elements in D_{211} and D_{222} are 183 and 204, respectively. In D_{211} each element has 3.6 child elements on average. In D_{222} each element has 2.9 child elements on average. The maximum number of child elements that an element holds in D_{211} and D_{222} are 19 and 20, respectively. Table 7.1 shows the number of update operations between D_{211} and D_{222}.

Since we didn't find any XSLT stylesheet for these DTDs, We made 10 XSLT stylesheets for XML to HTML transformation and used them in the experiment. The average number of rules of the stylesheets is 9. In the experiment, we have two examinees who are both graduate students and are

Table 7.1 Update operations between D_{211} and D_{222}

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

Table 7.2 Result of Experiment 1

Examinee	Case 1-a (nest)	Case 1-b (pattern)	Case 3 (*ins_elm*)	Total
1	12/12	22/22	0/3	34/37 (92%)
2	12/12	22/22	3/3	37/37 (100%)

[*1] http://www.msr-wg.de/medoc/downlo.html
[*2] http://dtd.nlm.nih.gov/publishing/

Table 7.3 Update operations between D_{23} and D_{30}

ins_elm	*del_elm*	*nest*	*unnest*	Total
734	100	39	26	899

familiar with DTD and XSLT. The experiment was conducted as follows.

1. We explained the definitions of the DTDs, R^+ rule, and R^- rule, and related examples to the examinees in advance.
2. We executed our algorithms for the 10 XSLT stylesheets, and obtained 37 corrected rules in total.
3. We presented the stylesheets and the corrected rules generated by our algorithm to the examinees, and asked them whether each of the corrected rules is "correct" or not.

Table 7.2 shows the result (in this experiment, no correction for Case 2 was made). For each XSLT stylesheet, our system constructed dependency graphs for old DTD and new DTD. Each dependency graph has 15.2 nodes and 8.5 edges on average. The average running time of our algorithm per XSLT stylesheet is 1.5 seconds under a mobile PC with Intel Core i3 2.60GHz. Each cell of the table gives the ratio of "the number of rules judged "correct" by the examinee" to "the number of rules corrected by the algorithm". It took about 32min on average for each examinee to judge the validity of each XSLT stylesheet.

In case 1-a, the algorithm made 12 corrections for nested elements, and both examinees judged that all the 12 corrections are "correct". The example of case 1-a in this experiment is shown in Figure 7.1. Similarly, in case 1-b, all the 22 corrections made by the algorithm were judged "correct" by both of the examinees. The example of case 1-b in this experiment is shown in Figure 7.2. On the other hand, in case 3 the three corrections made by the algorithm were not judged "correct" by Examinee 1. This is because Examinee 1 felt that adding new rules to newly inserted elements is unnecessary since new elements do not affect other existing elements in terms of XSLT transformation. The example of case 3 in this experiment is shown in Figure 7.3.

Secondly, we give a similar evaluation of our algorithm using the NLM Journal Publishing Tag Set Tag Library. Let D_{23} be the version 2.3 The NLM Journal Publishing Tag Set Tag Library DTD and D_{30} be the version 3.0 DTD. The number of elements of D_{23} is 213 and that of D_{30} is 235. In D_{23} each element has 11.1 child elements on average. In D_{222} each element has 13.1 child elements on average. The maximum number of child elements that an element holds in D_{211} and D_{222} are 73 and 82, respectively. Table 7.3 shows the number of update operations between D_{23} and D_{30}.

For the NLM Journal Publishing Tag Set Tag Library DTDs, we made 8 XSLT stylesheets based on XSLT 1.0 "NISO Journal Article Tag Suite (JATS) version 1.0"[*3] which is provided by National

[*3] https://github.com/ncbi/JATSPreviewStylesheets

Update Operations to DTDs:

```
L-4 element is nested as parent element of SUB.
```

Affected rule R_1:

```
<xsl:template match="SUB" mode="p">
    <sub>
        <xsl:apply-templates mode="p" />
    </sub>
</xsl:template>
```

Correcting option C_1 provided by our algorithm:
Add the following new rule.

```
<xsl:template match="L-4" mode="p">
    <span class="L-4">
        <xsl:apply-templates mode="p" />
    </span>
</xsl:template>
```

Figure 7.1 An example of case 1-a in Experiment 1

Table 7.4 Result of Experiment 2

Examinee	Case 1-a (nest)	Case 1-b (pattern)	Case 2 (*ins_elm*)	Total
1	7/7	19/19	2/2	28/28 (100%)
2	7/7	19/19	2/2	28/28 (100%)

Center for Biotechnology Information of U.S. National Library of Medicine (NLM). In the following, we call this XSLT JATS for short. In the package of JATS, there are two XSLT 1.0 transformations that can work standalone. They are jats-html.xsl XSLT stylesheets which is used for HTML transformation and jats-xslfo.xsl XSLT stylesheets used for XSL-FO transformation. In our experiment, we made XSLT stylesheets only based on jats-html.xsl for XML to HTML transformation and used them in the experiment. The average number of rules of the stylesheets is 22. In the experiment, we have two examinees who are both graduated students and are familiar with DTD and XSLT. The experiment was conducted similarly as the former experiment. We obtained 28 corrected rules in total for the 8 XSLT stylesheets.

Table 7.4 shows the result. (in this experiment, correction for Case 3 was ommited). For each XSLT stylesheet, our system constructed dependency graphs for old DTD and new DTD. Each dependency graph has 103.6 nodes and 834.2 edges on average. The average running time of our algorithm per XSLT stylesheet is 37.4 seconds under the same environment as the former experiment.

Update Operations to DTDs:

```
L-1 element is nested as parent element of TT.
```

Affected rule R_2:

```
<xsl:template match="MSR-QUERY-TEXT/MSR-QUERY-RESULT-TEXT/TT" mode="p">
    <tt>
        <xsl:apply-templates mode="q" />
    </tt>
</xsl:template>
```

Correcting option C_2 provided by our algorithm:
Fix R_2 as follows.

```
<xsl:template match="MSR-QUERY-TEXT/MSR-QUERY-RESULT-TEXT/L-1/TT" mode="p">
    <tt>
        <xsl:apply-templates mode="q" />
    </tt>
</xsl:template>
```

And add the following new rule.

```
<xsl:template match="MSR-QUERY-TEXT/MSR-QUERY-RESULT-TEXT/L-1" mode="p">
    <span class="L-1">
        <xsl:apply-templates mode="p" />
    </span>
</xsl:template>
```

Figure 7.2 An example of case 1-b in Experiment 1

In this experiment, we constructed dependency graphs with more nodes and edges than the former experiment, and the running time is longer, too. The following two points can be given as reasons.

1. DTD data sets in this experiment is larger than the former one. From the size of DTDs, we can see, the average number of child elements that an element has in this experiment is nearly three times more than that in the former experiment. Therefore, we constructed larger dependency graphs, and took more running time.
2. In this experiment, the dependency relationship of each elements in DTDs is higher than the former experiment. This also leads to the increase of the size of the dependency graphs.

Each cell of the table gives the ratio of “the number of rules judged “correct” by the examinee” to “the number of rules corrected by the algorithm”. It took about 37min on average for each examinee to judge the validity of each XSLT stylesheet.

Update Operations to DTDs:

```
L-1 element is nested as child element of DESC,
and XFILE element is newly inserted into L-1.
```

Affected rule R_3:

```
No affected rules.
```

Correcting option C_3 provided by our algorithm:
And add the following new rule.

```
<xsl:template match="CHG-ACTIONS/CHG-ACTION/DESC/L-1/XFILE" mode="p">
    <span class="XFILE">
        <xsl:apply-templates mode="p" />
    </span>
</xsl:template>
```

Figure 7.3 An example of case 3 in Experiment 1

In case 1-a, the algorithm made 7 corrections for nested elements, and both examinees judged that all the 7 corrections are "correct". The example of case 1-a in this experiment is shown in Figure 7.4. Similarly, in case 1-b, all the 19 corrections made by the algorithm were judged "correct" by both of the examinees. The example of case 1-b in this experiment is shown in Figure 7.5. On the other hand, in case 2 Examinee 1 couldn't find the affected rules successfully by himself. But the two corrections made by the algorithm were all judged "correct" by examinees. The example of case 2 in this experiment is shown in Figure 7.6.

We used Classes UTT and UTT^{pat} of tree transducers as the targets of our algorithm. In order to indicate the coverage of Classes UTT and UTT^{pat} of full XSLT set, we analyzed the details of two XSLT data sets.

Firstly, we used XSLT 1.0 "NISO Journal Article Tag Suite (JATS) version 1.0", which is the same one we used in Experiment 2. There are 323 templates(rules) in total, we can use 75 templates of them as our research target directly, and get 74 more templates just by some slight changing of the templates. Therefore, in JATS, 46% of templates can be used as our research target.

Secondly, we used XSL Transformations (XSLT) Version 1.0[*4], which is released by World Wide Web Consortium (W3C). There are 54 templates in total, we can use 27 templates of them as our research target directly, and get 17 more templates just by some slight changing of the templates. Therefore, in this XSLT data set, 81% of templates can be used as our research target. We can say that Classes UTT and UTT^{pat} of tree transducers are widely applied in XSLT. Thus, we believe that our algorithm is useful to detect and correct XSLT rules affected by schema updates.

[*4] https://www.w3.org/TR/xslt

Update Operations to DTDs:

```
citation element is unnested,
element-citation and mixed-citation elements are nested as parent elements of bold element.
```

Affected rule R_4:

```
  <xsl:template match="bold" mode="metadata" >
    <b>
      <xsl:apply-templates mode="metadata" />
    </b>
  </xsl:template>
```

Correcting option C_4 provided by our algorithm:
Add the following new rule.

```
<xsl:template match="element-citation" mode="metadata">
    <span class="element-citation">
        <xsl:apply-templates mode="metadata" />
    </span>
</xsl:template>

<xsl:template match="mixed-citation" mode="metadata">
    <span class="mixed-citation">
        <xsl:apply-templates mode="metadata" />
    </span>
</xsl:template>
```

Figure 7.4 An example of case 1-a in Experiment 2

Update Operations to DTDs:

```
custom-meta-wrap element is unnested,
and custom-meta-group elements is nested as parent element of custom-meta element.
```

Affected rule R_5:

```
<xsl:template match="article-meta/custom-meta-wrap/custom-meta" mode="metadata">
    <xsl:with-param name="label">
        <span class="custom-meta">
            <xsl:apply-templates mode=" metadata"/>
        </span>
    </xsl:with-param>
</xsl:template>
```

Correcting option C_5 provided by our algorithm:
Fix R_5 as follows.

```
<xsl:template match="article-meta/custom-meta-group/custom-meta" mode="metadata">
    <xsl:with-param name="label">
        <span class="custom-meta">
            <xsl:apply-templates mode=" metadata"/>
        </span>
    </xsl:with-param>
</xsl:template>
```

And add the following new rule.

```
<xsl:template match="article-meta/custom-meta-group" mode="metadata">
    <span class="custom-meta-group">
        <xsl:apply-templates mode="metadata" />
    </span>
</xsl:template>
```

Figure 7.5 An example of case 1-b in Experiment 2

Update Operations to DTDs:

```
label element is inserted as child element of back element.
```

Affected rule R_6:

```
<xsl:template match="label" mode="label-text">
    <span class="label">
        <xsl:apply-templates mode="label-text"/>
    </span>
</xsl:template>
```

Correcting option C_6 provided by our algorithm: Provide the following message to users. In the old DTD D_{23}, Rule R_6 is applicable to element $label^{notes,2}$, while in the new DTD D_{30}, Rule R_6 is applicable to both element $label^{notes,2}$ and lement $label^{back,1}$.

Figure 7.6 An example of case 2 in Experiment 2

Chapter 8

Conclusion

In this thesis, we proposed algorithms for detecting and correcting XSLT rules affected by DTD updates. We made an evaluation experiment and verified that most of rules generated by the algorithms were appropriate. However, the experiment was done under only two pairs of DTDs. As a future work, we also would like to conduct more experiments by using more different kinds of DTDs and XSLT stylesheets. In our experiment, we checked the validity of the corrected rules generated by our algorithm by examinees. As a future work, we consider to give a definition of "valid corrected rules", so that we can check the validity of the corrected rules generated by our algorithm automatically. We also consider extending our algorithm so that the algorithm can handle more powerful schema languages such as XML Schema and RELAX NG.

Acknowledgements

The author would first like to express my sincere gratitude to warm encouragement and support of my adviser, Associate Professor Nobutaka Suzuki in pursuing this study. Without his guidance and persistent help this thesis would not have been possible. The author would especially like to thank Professor Atsuyuki Morishima and Lecturer Mitsuharu Nagamori for their valuable suggestions and support. The author is also grateful to members of Nobutaka Suzuki laboratory for their daily supports, suggestions and encouragement.

Bibliography

[1] Benedikt, M., and Cheney, J. Destabilizers and independence of XML updates. *Proc. VLDB Endow. 3*, 1-2 (2010), 906–917.

[2] Blouin, A., and Beaudoux, O. Mapping paradigm for document transformation. In *Proc. ACM DocEng'07* (2007), pp. 219–221.

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

[4] Hasegawa, K., Ikeda, K., and Suzuki, N. An algorithm for transforming XPath expressions according to schema evolution. In *Proc. DChanges 2013* (2013).

[5] Horie, K., and Suzuki, N. Extracting differences between regular tree grammars. In *Proc. ACM SAC* (2013), pp. 859–864.

[6] Ishihara, Y., Suzuki, N., Hashimoto, K., Shimizu, S., and Fujiwara, T. XPath satisfiability with parent axes or qualifiers is tractable under many of real-world DTDs. In *Proc. DBPL 2013* (2013).

[7] Junedi, M., Genevès, P., and Layaïda, N. XML query-update independence analysis revisited. In *Proc. ACM DocEng'12* (2012), pp. 95–98.

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

[9] Martens, W., and Neven, F. Typechecking top-down uniform unranked tree transducers. In *Proc. ICDT* (2002), pp. 64–78.

[10] Online Computer Library Center. Introduction to the dewey decimal classification. *https://www.oclc.org/content/dam/oclc/content/dam /oclc/webdewey/help/introduction.pdf*.

[11] Oliveira, R., Genevès, P., and Layaïda, N. Toward automated schema-directed code revision. In *Proc. ACM DocEng'12* (2012), pp. 103–106.

[12] Suzuki, N. An edit operation-based approach to the inclusion problem for DTDs. In *Proc. ACM SAC* (2007), pp. 482–488.

[13] Wu, Y., and Suzuki, N. Detecting XSLT rules affected by schema evolution. In *Proc. ACM DocEng'15* (2015), pp. 143–146.