Towards the Preservation of Keys in XML Data Transformation for Integration *

Md. Sumon Shahriar and Jixue Liu

Data and Web Engineering Lab
School of Computer and Information Science
University of South Australia, SA-5095
South Australia, Adelaide, Australia
shamy022@students.unisa.edu.au, Jixue.Liu@unisa.edu.au

Abstract

Transformation of a source schema with its conforming data to a target schema with its conforming data is an important activity in XML as two schemas in XML can represent same real world information. Specifically in XML data integration, transformation of a source to a target is regarded as an important task. An XML source schema can often be defined with XML key which is an important integrity constraint. Thus when a source schema with keys is transformed, keys need to be transformed as they are defined on the schema. Moreover there is a need to investigate whether the transformed keys are valid and preserved. In this paper, we study how XML keys are transformed, and whether the transformed keys are valid and preserved to the target schema. Towards this problem, we firstly define XML keys and their satisfactions. We then show how the XML keys are transformed using transformation operations. Finally, we study the key preservation property of important XML transformation operators. We show that the important XML transformation operations are key preserving with necessary and sufficient conditions.

1 Introduction

Transformation of data plays an important role in data integration with any data model[1, 2, 3, 4, 7, 8]. In recent years, with the advent of XML as an widely used data representation and storage format over the world wide web, transformation of data from an XML source to an XML target is also considered as an important activity[5, 6, 10, 11, 12]. An XML source schema can be defined with XML keys. Thus, when an XML source schema is transformed to a target schema, XML keys can also be transformed. We note here that though the transformation of XML data is researched in past[9, 13, 14, 16], but the transformation of data with keys solely in XML(XML to XML) is little investigated to the best of our knowledge[15, 17, 18, 19]. We illustrate the research problems using motivating examples.

<!*ELEMENT* $enroll(dept^+)$ >
<!*ELEMENT* $dept(dname, (cid, sid^+)^+)$ >

Figure 1: XML DTD D_a

Example 1: Consider the DTD D_a in Fig.1 that describes the enrollment of students in the courses of departments where *dname* stands for department name, *cid* stands for course id, and *sid* stands for student id(for simplicity, we omit the type ($\#PCDATA$) from the DTD). We see that for each department, student ids are grouped under each course id. Now consider the key $\mathbb{k}_{a_1}(enroll/dept, \{cid\})$ on D_a where $enroll/dept$ is called the `selector` and *cid* is called the `field`. We say the key $\mathbb{k}_{a_1}$ is valid because both the selector and the connection of the selector and the field as `selector/field`, $enroll/dept/cid$, is a valid path on D_a, and the type of last element of the field is $\#PCDATA$. The $\mathbb{k}_{a_1}$ requires that *cid* values under all selector nodes are distinct. This requirement is satisfied by T_a in Fig.2 because under the selector nodes v_1 and v_2(as last element of the selector path $enroll/dept$ is *dept*), the values $(cid : Phys01)$, $(cid : Phys02)$, and $(cid : Chem02)$ are distinct. Now

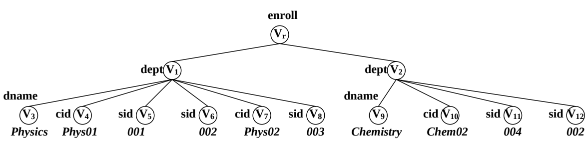

Figure 2: XML Tree T_a

*This research is supported with Australian Research Council(ARC) Discovery Project Fund(Project No. DP0559202)

International Conference on Management of Data
COMAD 2008, Mumbai, India, December 17–19, 2008

we want to transform the DTD D_a in Fig.1 with its document T_a in Fig.2 to the DTD D_b in Fig.3 and its document T_b in Fig.4 where the *cid* value is distributed to each of the associated *sid* values and hence we expect to get the flat structure in D_b. We term this transformation as *unnest*(a technical definition will be given later). After transformation, we see that the key $\mathbb{k}_{a_1}$ needs no change as it is valid on D_b. But $\mathbb{k}_{a_1}$ is not satisfied by T_b as there are two values $(cid : Phys01)$ and $(cid : Phys01)$ which are not distinct under the selector node v_1 and also there are two values $(cid : Chem02)$ and $(cid : Chem02)$ which are not distinct under the selector node v_2. We say the key $\mathbb{k}_{a_1}$ is not preserved by the unnest operator in this case.

$<!ELEMENT\ enroll(dept^+)>$
$<!ELEMENT\ dept(dname,(cid,sid)^+)>$

Figure 3: XML DTD D_b

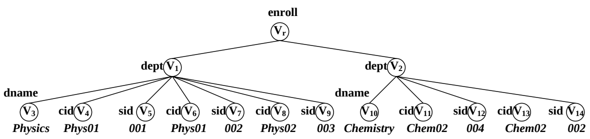

Figure 4: XML Tree T_b

Observation 1 *XML key(s) may not be preserved after the transformation of a DTD and its conforming document.*

Example 2: Consider the key $\mathbb{k}_{a_2}(enroll/dept, \{cid, sid\})$ on D_a in Fig.1. We note that there are two fields in $\mathbb{k}_{a_2}$. In this case, $\mathbb{k}_{a_2}$ is satisfied if the tuples of (cid, sid) are value distinct under all selector nodes. By tuple, we mean the value of a close pair of the fields(a technical definition will be given later). So the tree T_a satisfies the key $\mathbb{k}_{a_2}$ as the tuples under nodes v_1 $((cid : Phys01)(sid : 001))$, $((cid : Phys01)(sid : 002))$, and $((cid : Phys02)(sid : 003))$ are all value distinct and also the tuples under node v_2, $((cid : Chem02)(sid : 004))$, and $((cid : Chem02)(sid : 002))$ are all value distinct. We note here that $((cid : Phys02)(sid : 002))$ is not a correct tuple under node v_1 as the two nodes are not close. Similarly, $((cid : Phys01)(sid : 003))$ is not a correct tuple under node v_1.

$<!ELEMENT\ enroll(dept^+)>$
$<!ELEMENT\ dept(dname, course^+)>$
$<!ELEMENT\ course(cid, sid^+)>$

Figure 5: XML DTD D_c

We now consider another transformation that transforms the DTD D_a in Fig.1 to D_c in Fig.5. In the transformation, we use a new element *course* to push away the structure (cid, sid^+) from $enroll/dept$. We term this transformation as *expand*.

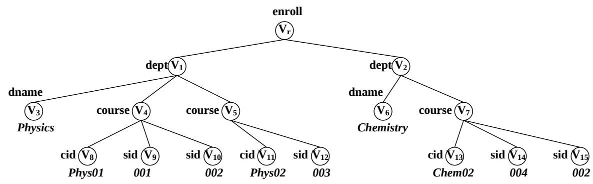

Figure 6: XML Tree T_c

Now the key $\mathbb{k}_{a_2}(enroll/dept, \{cid, sid\})$ defined on D_a is no longer valid as the connection of the selector and the fields: $enroll/dept/cid$ and $enroll/dept/sid$ are not valid paths on D_c. So there is a need to transform the key to make the key valid. There are two options to transform $\mathbb{k}_{a_2}$: One is to add the *course* element to the beginning of fields as $\mathbb{k}'_{a_2}(enroll/dept, \{course/cid, course/sid\})$ and the other is to add the *course* element at the last of selector as $\mathbb{k}''_{a_2}(enroll/student/course, \{cid, sid\})$.

Observation 2 *How XML keys should be transformed needs to be defined when the DTD is transformed.*

While addressing the problems, our paper aims at the following contributions.

- *Firstly*, we define XML keys over XML DTDs and the satisfaction of XML keys[25]. This definition is between the strong key definition and the weak key definition [20] and addresses some shortages of both definitions. At the same time, the key satisfaction uses a novel concept called P-tuple which is more precise in capturing the semantics than existing definitions [20, 22].

- *Secondly*, We show how XML keys are transformed so that the transformed keys have valid syntax and contain valid paths of the transformed DTD. This is presented on the basis of important transformation operations. As full transformation operations use many operators proposed in [33], but for space reasons, this paper considers only the commonly used operators namely *nest*, *unnest*, *expand* and *collapse* which are also appeared in [10, 11]. These operators are core ones when XML data is transformed and the study of key transformation against these operators has great importance.

- *Lastly*, we show whether a key is preserved by a transformation. Again this is studied based on the operators. In general, key preservation against some of the operators does not hold. However, we identified sufficient and necessary conditions indicating cases where keys are preserved.

Our paper is organized as follows. In section 2, we give basic definitions and notations used throughout the paper. We define the transformation definitions on XML keys using operators in section 3. In section 4, we show the key preservation properties of transformation operations using the transformed and valid keys. We note our research on preservation of XML functional dependency and XML referential integrity in XML data transformation in section 5. A list of future research issues are discussed in section 6. We conclude in section 7.

2 Basic Definitions

In this section, we give some preliminary definitions that are used throughout the paper.

Our model is XML Document Type Definition(DTD)[23] with some restrictions. We allow the same element names to appear in disjunctions, but not among conjunctions in DTD. We do not allow recursion. We do not consider attributes because there is an one-to-one correspondence between an attribute and an element with multiplicity '1'.

We define operations on multiplicities. The meaning of a multiplicity can be represented by an integer interval. Thus the intervals of ?, 1, +, and $*$ are $[0,1]$, $[1,1]$, $[1,m]$, $[0,m]$ respectively. The operators for multiplicities c_1 and c_2 are $\oplus$, $\ominus$ and $\supseteq$. $c_1 \oplus c_2$ is the multiplicity whose interval encloses those of c_1 and c_2: $+\oplus? = *$ and $1\oplus? =?$. $c_1 \ominus c_2$ is the multiplicity whose interval equals to the interval of c_1 and c_2 adding that of $'1'$. Thus $?\ominus? = 1$ and $* \ominus + =?$. $c_1 \supseteq c_2$ means that c_1's interval contains c_2's interval.

Definition 2.1 *An XML DTD is defined as $D = (EN, G, \beta, \rho)$ where*

(a) EN contains element names.

(b) G is the set of element definitions and $g \in G$ is defined as

(i) $g = Str$ where Str means #PCDATA;

(ii) $g = e$ where $e \in EN$;

(iii) $g = \epsilon$ means EMPTY type;

(iv) $g = g_1{\times}g_2$ or $g_1|g_2$ is called conjunctive or disjunctive sequence respectively where $g_1 = g$ is recursively defined, $g_1 \neq Str \wedge g_1 \neq \epsilon$;

(v) $g = g_2^c \wedge g_2 = e \wedge e \in EN$, or $g_2 = [g{\times}\cdots{\times}g]$ or $g_2 = [g|\cdots|g]$, called a component where $c \in \{?, 1, +, *\}$ is the multiplicity of g_2, [] is the component constructor;

(c) $\beta(e) = [g]^c$ is the function defining the type of e where $e \in EN$ and $g \in G$.

(d) ρ is the root of the DTD and that can be only be used as $\beta(\rho)$. □

Example 2.1 *The DTD in Fig.1 can be represented as $D = (EN, G, \beta, \rho)$ where $EN = \{enroll, dept, dname, sid, cid\}$, $G = \{Str, [dept]^+, [dname{\times}[cid{\times}sid^+]^+]\}$, $\beta(enroll) = [dept]^+$, $\beta(dept) = [dname{\times}[cid{\times}sid^+]^+]$, $\beta(dname) = Str$, $\beta(sid) = Str$ and $\beta(cid) = Str$.*

Definition 2.2 *An XML tree T parsed from an XML document in our notation is a tree of nodes and each is represented as $T = (v : e\ (T_1T_2 \cdots T_f))$ if the node is internal or $T = (v : e : txt)$ if the node is a leaf node with the text txt. v is the node identifier which can be omitted when the context is clear, e is the label on the node. $T_1 \cdots T_f$ are subtrees.* □

Example 2.2 *The XML tree T_a in Fig.2 can be represented as $T_{v_r} = (v_r : enroll(T_{v_1}T_{v_2}))$, $T_{v_1} = (v_1 : dept(T_{v_3}T_{v_4}T_{v_5}T_{v_6}T_{v_7}T_{v_8}))$, $T_{v_2} = (v_2 : dept(T_{v_9}T_{v_{10}}T_{v_{11}}T_{v_{12}}))$, $T_{v_3} = (v_3 : dname : Physics)$, $T_{v_4} = (v_4 : cid : Phys01)$, $T_{v_5} = (v_5 : sid : 001)$, $T_{v_6} = (v_6 : sid : 002)$, $T_{v_7} = (v_7 : cid : Phys02)$, $T_{v_8} = (v_8 : sid : 003)$, $T_{v_9} = (v_9 : dname : Chemistry)$, $T_{v_{10}} = (v_{10} : cid : Chem02)$, $T_{v_{11}} = (v_{11} : sid : 004)$, and $T_{v_{12}} = (v_{12} : sid : 002)$.*

Now we give an example to show the important concept *hedge*. Consider $g_1 = [cid{\times}sid^+]^+$ for the DTD D_a in Fig.1. The trees $T_{v_4}T_{v_5}T_{v_6}T_{v_7}T_{v_8}$ form a sequence conforming to g_1 for node v_1 and the trees $T_{v_{10}}T_{v_{11}}T_{v_{12}}$ form a sequence for node v_2. However, when we consider $g_2 = cid{\times}sid^+$, there are two sequences conforming to g_2 for node v_1: $T_{v_4}T_{v_5}T_{v_6}$ and $T_{v_7}T_{v_8}$. For node v_2, there is only one sequence conforming to g_2: $T_{v_{10}}T_{v_{11}}T_{v_{12}}$. To reference various structures and their conforming sequences, we introduce the concept *hedge*, denoted by H^g, which is a sequence of trees conforming to the structure g. Thus $H_1^{g_2} = T_{v_4}T_{v_5}T_{v_6}$, $H_2^{g_2} = T_{v_7}T_{v_8}$ for node v_1 and $H_3^{g_2} = T_{v_{10}}T_{v_{11}}T_{v_{12}}$ for node v_2.

Definition 2.3 (Hedge) *A hedge H is a sequence of consecutive primary sub trees $T_1T_2 \cdots T_n$ of the same node that conforms to the definition of a specific structure g, denoted by $H \Subset g$ or H^g:*

(1) if $g = e \wedge \beta(e) = Str, H = T = (v : e : txt)$;

(2) if $g = e \wedge \beta(e) = g_1$, $H = T = (v : e : H')$ and $H' \Subset g_1$;

(3) if $g = \epsilon$, $H = T = \phi$;

(4) if $g = g_1{\times}g_2$, $H = H_1H_2$ and $H_1 \Subset g_1$ and $H_2 \Subset g_2$;

(5) if $g = g_1|g_2$, $H = H_0$ and $H_0 \Subset g_1$ or $H_0 \Subset g_2$;

(6) if $g = g_1^c \wedge g_1 = e$, $H = (eH_1) \cdots (eH_f)$ and $\forall i = 1, \cdots, f\ (H_i \Subset \beta(e))$ and f satisfies c;

(7) if $g = g_1^c \wedge g_1 = [g]$, $H = H_1 \cdots H_f$ and $\forall i = 1, \cdots, f(H_i \Subset g)$ and f satisfies c. □

Because gs are different substructures of an element definition, then H^gs are different groups of child nodes. Because of the multiplicity, when there are multiple H^gs, we use H_j^g to denote one of them and H^{g*} to denote all of them.

Definition 2.4 (Tree Conformation) *Given a DTD $D = (EN, G, \beta, \rho)$ and XML Tree T, T conforms to D denoted by $T \Subset D$ if T=(ρ $H^{\beta(\rho)}$).* $\square$

Definition 2.5 (Hedge Equivalence) *Two trees T_a and T_b are value equivalent, denoted by $T_a =_v T_b$, if*

(1) $T_a = (v_1 : e : txt1)$ and $T_b = (v_2 : e : txt1)$, or
(2) $T_a = (v_1 : e : T_1 \cdots T_m)$ and $T_b = (v_2 : e : T_1^{'} \cdots T_n^{'})$ and $m = n$ and for $i = 1, \cdots, m(T_i =_v T_i^{'})$.

Two hedges H_x and H_y are value equivalent, denoted as $H_x =_v H_y$, if

(1) both H_x and H_y are empty, or
(2) $H_x = T_1 \cdots T_m$ and $H_y = T_1^{'} \cdots T_n^{'}$ and $m = n$ and for $i = 1, \cdots, m(T_i =_v T_i^{'})$ $\square$

$T_x \equiv T_y$ if T_x and T_y refer to the same tree. We note that, if $T_x \equiv T_y$, then $T_x =_v T_y$.

Definition 2.6 (Minimal hedge) *Given a DTD definition $\beta(e)$ and two elements e_1 and e_2 in $\beta(e)$, the minimal structure g of e_1 and e_2 in $\beta(e)$ is the pair of brackets that encloses e_1 and e_2 and any other structure in g does not enclose both.*
Given a hedge H of $\beta(e)$, a minimal hedge of e_1 and e_2 is one of H^gs in H. $\square$

Example 2.3 *Let $\beta(dept) = [dname_\times[cid_\times sid^+]^+]$ in D_a. Thus, The minimal structure of dname and sid is $g_1 = [dname_\times[cid_\times sid^+]^+]$. Thus the minimal hedge conforming to g_1 is $H_1^{g_1} = T_{v_3}T_{v_4}T_{v_5}T_{v_6}T_{v_7}T_{v_8}$ for node v_1 and $H_2^{g_1} = T_{v_9}T_{v_{10}}T_{v_{11}}T_{v_{12}}$ for node v_2 in T_a.*

But the minimal structure of cid and sid is $g_2 = [cid_\times sid^+]$. So the the minimal hedges conforming to g_2 are $H_1^{g_2} = T_{v_4}T_{v_5}T_{v_6}$, $H_2^{g_2} = T_{v_7}T_{v_8}$ for node v_1 and $H_3^{g_2} = T_{v_{10}}T_{v_{11}}T_{v_{12}}$ for node v_2 in T_a.

Definition 2.7 (Paths) *Given a D = (EN, G, β, ρ), a simple path $\wp$ on D is a sequence $e_1/\cdots/e_m$, where $\forall e_i \in EN$ and $\forall\, e_w \in [e_2, \cdots, e_m]$ (e_w is a symbol in the alphabet of $\beta(e_{w-1})$). A simple path $\wp$ is a complete path if $e_1 = \rho$. A path $\wp$ is empty if $m = 0$, denoted by $\wp = \epsilon$. We use function $last(\wp)$ to return e_m, $beg(\wp) = e_1$, $par(e_w) = e_{w-1}$, the parent of e_w. We use $len(\wp)$ to return m. Paths satisfying this definition are said* ***valid*** *on D.* $\square$

Example 2.4 *In Fig.1 on the DTD D_a, dept/sid is a simple path and enroll/dept/sid is a complete path. The function beg(enroll/dept/sid) returns enroll. The function last(enroll/dept/sid) returns sid, par(sid) returns dept, and len(enroll/dept/sid) = 3.*

Definition 2.8 (XML Key) *Given a DTD D = (EN, G, β, ρ), an XML key on D is defined as $\Bbbk(Q, \{P_1, \cdots, P_l\})$, where $l \geq 0$, Q is a complete path called the* **selector***, and $\{P_1, \cdots, P_i, \cdots, P_l\}$ (often denoted by P) is a set of* **fields** *where each P_i is defined as:*

(a) $P_i = \wp_{i1} \cup \cdots \cup \wp_{in_i}$,where "$\cup$" means disjunction and $\wp_{ij}$ ($j \in [1, \cdots, n_i]$) is a simple path on D, and $\beta(last(\wp_{ij})) = Str$, and $\wp_{ij}$ has the following syntax:
$\wp_{ij} = seq$
$seq = e \mid e/seq$ where $e \in EN$;
(b) $Q/\wp_{ij}$ is a complete path. $\square$

A path $\wp$ is in P if $\exists\, P_i \in P(\wp \in P_i)$. $\wp \in \Bbbk$ if $\wp = Q$ or $\wp \in P$. We use $\wp_i$ to mean a path in P_i if there is no ambiguity. A key following this definition is called a *valid* key on D, denoted by $\Bbbk \sqsubset D$. A key is not valid if some conditions in the definition 2.8 is not satisfied.

Example 2.5 *Let $\Bbbk_{a_1}(enroll/dept, \{cid\})$ be a key notation on D_a in Fig.1. The selector is Q = enroll/dept which is a complete path and field is $P_1 = \wp_{11} = cid$ is a simple path, $\beta(cid) = Str$. We see that $Q/\wp_{11}$ = enroll/dept/cid is a complete path. This notation represents a valid key.*

We give here an example where the disjunctive paths are considered in XML key definition according to the DTD.

```
<!ELEMENT db(stud+)>
<!ELEMENT stud(sname,(email|tellno)+)>
```

Figure 7: XML DTD D

Example 2.6 *Using our DTD notation, we represent D as $\beta(db) = [stud^+]$, $\beta(stud) = [sname_\times[email|tellno]^+]$ in Fig.7. We define a key as $\Bbbk(db/stud, \{sname, email|tellno\}$ on D where selector is Q = db/stud and the fields are $P_1 = \wp_{11}$ = sname, $P_2 = \wp_{21}|\wp_{22}$ = email|tellno. $Q/\wp_{11}$ = db/stud/sname, $Q/\wp_{21}$ = db/stud/email, and $Q/\wp_{22}$ = db/stud/tellno are complete paths. The type of sname, email, and tellno are Str.*

We define some additional notation. T^e means a tree rooted at a node labeled by the element name e. Given path $e_1/\cdots/e_m$, we use $(v_1 : e_1).\cdots.(v_{m-1} : e_{m-1}).T^{e_m}$ to mean the tree T^{e_m} with its ancestor nodes in sequence, called the *prefixed tree* or the *prefixed format* of T^{e_m}. Given path $\wp = e_1/\cdots/e_m$, $T^{\wp} = (v_1 : e_1).\cdots.(v_{m-1} : e_{m-1}).T^{e_m}$. $\langle T^{\wp}\rangle$ is the set of all $T^{\wp}$ and $\langle T^{\wp}\rangle = \{T_1^{\wp}, \cdots, T_f^{\wp}\}$. $|\langle T^{\wp}\rangle|$ returns the number of $T^{\wp}$ in $\langle T^{\wp}\rangle$. Because $P_i = \wp_{i1}|\cdots|\wp_{in_i}$, we use $\langle T^{P_i}\rangle$ to mean all $T^{\wp_{ij}}$s and $T^{P_i} = T^{\wp_i}$ to mean one of $T^{\wp_{ij}}$s. We use $T^{\wp_i} \in T^Q$ to mean that $T^{\wp_i}$ is a sub tree of T^Q. Similarly, $\langle T^{P_i}\rangle \in T^Q$ means that all trees T^{P_i} are sub trees of T^Q.

Example 2.7 *We define T^{dept} to mean the tree T_{v_1} or T_{v_2} in T_a of Fig.2. Consider a path $Q = enroll/dept$. Then $last(Q) = dept$. We use T^Q to mean the tree T_{v_1} or T_{v_2}, $\langle T^Q \rangle = \{T_{v_1}, T_{v_2}\}$, and $|\langle T^Q \rangle| = 2$. Now let a path be $\wp = cid$. So $\langle T^{\wp} \rangle = \{T_{v_4}, T_{v_7}\} \in T_{v_1}$ and $\langle T^{\wp} \rangle = \{T_{v_{10}}\} \in T_{v_2}$ in T_a.*

We now introduce the novel concept P-tuples using an example 2.8.

```
<!ELEMENT db(univ+) >
<!ELEMENT univ(dept, staff+)+ >
<!ELEMENT staff(fname, lname) >
```

Figure 8: XML DTD D

Example 2.8 *Consider an XML key on D in Fig.8 as $\Bbbk(db/univ, \{dept, staff/fname, staff/lname\})$ where $Q = db/univ$, $P_1 = dept$, $P_2 = staff/fname$, $P_3 = staff/lname$. All the pair combinations of the fields are (P_1, P_2), (P_1, P_3), and (P_2, P_3). Then in Fig.9, the tuple $(T_{v_3}T_{v_8}T_{v_9})$ is a P-tuple because, with regard to (P_1, P_2) and (P_1, P_3), T_{v_3} and T_{v_4} (the parent of both T_{v_8} and T_{v_9}) are from the same minimal hedge of $[dept_\times staff^+]$ under T_{v_1}; with regard to (P_2, P_3), T_{v_8} and T_{v_9} are from the same minimal hedge of $[fname_\times lname]$ under T_{v_4}. In the same way of reasoning, $(T_{v_3}T_{v_{10}}T_{v_{11}})$ is another P-tuple under T_{v_1}, and $(T_{v_6}T_{v_{12}}T_{v_{13}})$ is a P-tuple under T_{v_2}. On the contrary, the tuple $(T_{v_3}T_{v_8}T_{v_{11}})$ is not a P-tuple because T_{v_8} and $T_{v_{11}}$ are not in the same minimal hedge of $[fname_\times lname]$ with regard to (P_2, P_3). Another non-P-tuple under T_{v_1} is $(T_{v_3}T_{v_{10}}T_{v_9})$. The tuples prevent incorrect trees from being combined in the key satisfaction test, for example, when the first name of one staff member combines with the last name of another staff member, the tuples does not make sense in the application.*

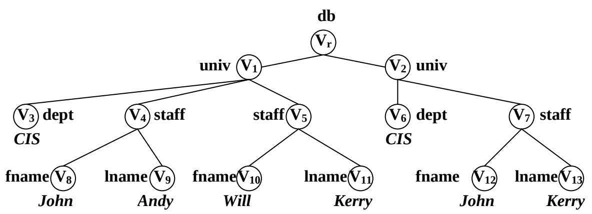

Figure 9: XML Tree

Definition 2.9 (P-tuple) *Given a key $\Bbbk(Q, \{P_1, ..., P_l\})$ and a tree T, let T^Q be a tree in T. A P-tuple under T^Q is a tuple of pair-wise close subtrees $(T^{P_1} \cdots T^{P_l})$ as we define next.*
Let $\wp_i = e_1/\cdots/e_k/e_{k+1}/\cdots/e_m$ where $\wp_i \in P_i$, and $\wp_j = e'_1/\cdots/e'_k/e'_{k+1}/\cdots/e'_n$ where $\wp_j \in P_j$, for any P_i and P_j. Let $(v_1 : e_1).\cdots.(v_k : e_k).(v_{k+1} : e_{k+1}).\cdots.T^{P_i}$ and $(v'_1 : e'_1).\cdots.(v'_k : e'_k).(v'_{k+1} : e'_{k+1}).\cdots.T^{P_j}$ be the prefixed formats of T^{P_i} and T^{P_j} where $(v_m : e_m) = root(T^{P_i})$ and $(v'_n : e'_n) = root(T^{P_j})$. Then T^{P_i} and T^{P_j} are pair-wise close if
(a) If $e_1 \neq e'_1$, then $(v_1 : e_1)$ and $(v'_1 : e'_1)$ are the nodes of the same minimal hedge of e_1 and e'_1 in $\beta(last(Q))$.
(b) If $e_1 = e'_1$, $\cdots$, $e_k = e'_k$, $e_{k+1} \neq e'_{k+1}$, then $v_k = v'_k$, $(v_{k+1} : e_{k+1})$ and $(v'_{k+1} : e'_{k+1})$ are two nodes in the same minimal hedge of e_{k+1} and e'_{k+1} in $\beta(e_k)$. □

A P-tuple $(T^{P_1} \cdots T^{P_l})$ is complete if $\forall T^{P_i} \in (T^{P_1} \cdots T^{P_l})(T^{P_i} \neq \phi)$. We use $\langle T^P \rangle$ to denote all possible P-tuples under a T^Q tree and $|\langle T^P \rangle|$ means the number of such P-tuples. Two P-tuples $F_1 = (T_1^{P_1} \cdots T_1^{P_l})$ and $F_2 = (T_2^{P_1} \cdots T_2^{P_k})$ are value equivalent, denoted by $F_1 =_v F_2$ if $l = k$ and for each $i = 1, \cdots, k$ $(T_1^{P_i} =_v T_2^{P_i})$.

Definition 2.10 (Key Satisfaction) *An XML tree T satisfies a key $\Bbbk(Q, \{P_1, ..., P_l\})$, denoted by $T \prec \Bbbk$, if the followings are hold:*

(i) If $\{P_1, ..., P_l\} = \phi$ in $\Bbbk$, then T satisfies $\Bbbk$ iff there exists one and only one T^Q in T;
(ii) else,
  - *(a) $\forall T^Q \in \langle T^Q \rangle$ (exists at least one P-tuple in T^Q);*
  - *(b) $\forall T^Q \in \langle T^Q \rangle$ (every P-tuple in T^Q is complete);*
  - *(c) $\forall T^Q \in \langle T^Q \rangle$ (every P-tuple in T^Q is value distinct);*
  - *(d) $\forall T_1^Q, T_2^Q \in \langle T^Q \rangle$(exists two P-tuples $(T_1^{P_1} \cdots T_1^{P_l}) \in T_1^Q \wedge (T_2^{P_1} \cdots T_2^{P_l}) \in T_2^Q \wedge (T_1^{P_1} \cdots T_1^{P_l}) =_v (T_2^{P_1} \cdots T_2^{P_l})$ $\Rightarrow T_1^Q \equiv T_2^Q$). This requires that P-tuples under different selector nodes must be distinct.* □

Example 2.9 *Let $\Bbbk_{a_3}(enroll, \{dept/cid, dept/sid\})$ be a key on D_a in Fig.1. We want to check whether $\Bbbk_{a_3}$ is satisfied by the XML document T_a in Fig. 2. In T_a, we have only one node v_r for the selector $Q = enroll$. For the node v_r, we have the P-tuples $F_1 = (T_{v_4}T_{v_5})$, $F_2 = (T_{v_4}T_{v_6})$, $F_3 = (T_{v_7}T_{v_8})$, $F_4 = (T_{v_{10}}T_{v_{11}})$ and $F_5 = (T_{v_{10}}T_{v_{12}})$. As as F_1, F_2, F_3, F_4, F_5 are all value different, so $T_a \prec \Bbbk_{a_3}$.*

Now consider another key $\Bbbk_{a_4}(enroll, \{dept/sid\})$. So, for v_r, there are P-tuples $F_1 = (T_{v_5})$, $F_2 = (T_{v_6})$, $F_3 = (T_{v_{11}})$ and $F_4 = (T_{v_{12}})$. But as $F_2 =_v F_4$, so $T_a \not\prec \Bbbk_{a_4}$.

Theorem 2.1 *Let $\Bbbk(Q, P)$ be a key and $P \neq \phi$. $T \prec \Bbbk(Q, P)$, iff there exists a P-tuple for every T^Q and all P-tuples are complete and value distinct in T.*

2.1 Discussion

Our definition for XML key has some advantages over other definitions proposed in the literature [20, 21, 22, 27].

(i) Our definition uses leave nodes for key fields and these leave nodes must appear in the tree to satisfy the key definition. This prevents empty subtrees from being under $last(P_i)$.

(ii) Our definition directly corresponds to the relational keys in the sense that both types of keys uses value comparison.

(iii) Our key definition addresses the ambiguity in tuple production for key fields in [20]. As shown in Example 2.8, our definition prevents semantically incorrect combinations in tuple production.

(iv) There are two types of key definitions proposed in the literature[20, 24], strong key definition and weak key definition. Strong key definition allows only one P-tuple of fields in the key to appear under each target node. While the weak key definition allows multiple P-tuples for the key fields under each target node and some of these P-tuples can be the same(duplicate) but tuples between different target nodes must be different. However, our definition is between the two definitions in the sense that we allow multiple tuples, but all tuples of the key fields must be distinct in the tree.

3 Transformation on Key Validity and Compliance

Given a DTD D and a document T such that $T \Subset D$, a transformation τ transforms D to $\bar{D}$ and T to $\bar{T}$, denoted by $\tau(D, T) \rightarrow (\bar{D}, \bar{T})$. We use top bar($^-$) to mean the transformation of DTD D, or XML document T, or XML key $\Bbbk$. The problem whether $\bar{T}$ conforms to $\bar{D}$ was investigated in [12]. In this paper, we investigate how the transformation affects the properties of a key defined on D. More formally, given D, T, $\Bbbk$ and a transformation τ such that $\Bbbk \sqsubset D \wedge T \Subset D \wedge T \prec \Bbbk$, and $\tau(D, T, \Bbbk) \rightarrow (\bar{D}, \bar{T}, \bar{\Bbbk})$), we would like to know what $\bar{\Bbbk}$ is, whether $\bar{\Bbbk}$ is valid on $\bar{D}$, and whether $\bar{T}$ satisfies $\bar{\Bbbk}$. In this section, we answer two questions: how a key can be transformed in correspondence to the transformation of the DTD and the document, and whether a transformed definition is valid. Recall that key validity means that every path $\wp$ in $\bar{\Bbbk}$ is valid on $\bar{D}$; Q in $\bar{\Bbbk}$ is not empty; if $\wp \in P$, $\wp$ must be ended with a leaf element having *Str* type and $Q/\wp$ is a complete path.

3.1 Transformation Operations

Before answering the two questions, we introduce four transformation operations. These transformation operators are *expand*, *collapse*, *nest*, and *unnest*. We note that these transformation operators are considered in most literatures [10, 11, 12] for transforming XML data. We refer to [32] for a complete set of operators for XML data transformation to the interested readers. To understand the effect of these operators on XML key transformation and preservation, we need to know how they work.

Definition 3.1 (Expand) *The expand operator uses a new element name to push a component one level away from the root. Let e_{new} be the new element and g_d be the component to be pushed. If $g = g_d \neq \epsilon$ where g_d is a component in $\beta(e)$ and $e_{new} \notin \beta(e)$, then $expand(g_d, e_{new}) \rightarrow g = e_{new} \wedge \beta(e_{new}) = g_d$. With documents, $expand(H) \rightarrow \bar{H}$ where $H = H^g$ and $\bar{H} = (e_{new}H^g)$.* $\square$

Example 3.1 *Let $\beta(\rho) = [A_\times[B_\times C]^+]^+$. Let the tree be $T = (\rho(A:1)(B:2)(C:3)(B:4)(C:5)(A:6)(B:7)(C:8))$. Note that we can expand either $[B_\times C]$(without '+') or $[B_\times C]^+$(with '+'). Then, after $expand([B_\times C], E)$, $\beta_1(\rho) = [A_\times E^+]^+, \beta_1(E) = [B_\times C]$ and $\bar{T} = (\rho(A:1)(E(B:2)(C:3))(E(B:4)(C:5))(A:6)(E(B:7)(C:8)))$. If we do $expand([B_\times C]^+, E)$, $\beta_1(\rho) = [A_\times E]^+, \beta_1(E) = [B_\times C]^+$ and $\bar{T} = (\rho(A:1)(E(B:2)(C:3)(B:4)(C:5))(A:6)(E(B:7)(C:8)))$.*

Definition 3.2 (Collapse) *The collapse operator uses the definition of an element to replace that element name. Let e_{coll} be the element to be collapsed. If $g = e^c_{coll} \wedge \beta(e_{coll}) = [g_{e_{coll}}]^{c_1}$ and $g_{e_{coll}} \cap par(e_{coll}) = \phi$, then the transformation $collapse(e_{coll}) \rightarrow g = [g_{e_{coll}}]^{c \oplus c_1}$. With documents, $collapse(H) \rightarrow \bar{H}$ where $H = H^g = (e_{coll}H_1^{[g_{e_{coll}}]^*}) \cdots (e_{coll}H_m^{[g_{e_{coll}}]^*})$, m satisfies c and $\bar{H} = H_1^{[g_{e_{coll}}]^*} \cdots H_m^{[g_{e_{coll}}]^*}$.* $\square$

Example 3.2 *Let $\beta(\rho) = [A_\times B]^+$,$\beta(B) = [C]$. Let the tree be $T = (\rho(A:1)(B(C:3))(A:1)(B(C:5)))$. Then, after $collapse(B)$,$\beta_1(\rho) = [A_\times C]^+$ and $\bar{T} = (\rho(A:1)(C:3)(A:1)(C:5))$. Note that we can't $collapse(C)$ because $\beta(C) = Str$.*

Definition 3.3 (UnNest) *The unnest operation on g_2 in $[g_1 \times g_2^{c_2}]^c$ is defined as, if $g = [g_1 \times g_2^{c_2}]^c \wedge c_2 = +|*$, then $unnest(g_2) \rightarrow [g_1 \times g_2^{c_2 \ominus +}]^{c \oplus +}$. Also, $unnest(H) \rightarrow \bar{H}$ where $H = H^g = H_1^{g_1} H_{11}^{g_2} \cdots H_{1n_1}^{g_2} \cdots H_m^{g_1} H_{m1}^{g_2} \cdots H_{mn_m}^{g_2}$ and the transformed hedge is $\bar{H} = H_1^{g_1} H_{11}^{g_2} \cdots H_1^{g_1} H_{1n_1}^{g_2} \cdots H_m^{g_1} H_{m1}^{g_2} \cdots H_m^{g_1} H_{mn_m}^{g_2}$.* $\square$

Example 3.3 *Let $\beta(\rho) = [A_\times B^+]$ and the tree $T = (\rho(A:1)(B:2)(B:3)(A:4)(B:5))$. Then, after $unnest(B)$, $\beta_1(\rho) = [A_\times B]^+$ and $\bar{T} = (\rho(A:1)(B:2)(A:1)(B:3)(A:4)(B:5))$.*

Definition 3.4 (Nest) *The nest operation on g_2 in $[g_1 \times g_2^{c_2}]^c$ is defined as, if $g = [g_1 \times g_2^{c_2}]^c \wedge c \supseteq +$, then $nest(g_2) \rightarrow [g_1 \times g_2^{c_2 \oplus +}]^c$. Also, $nest(H) \rightarrow \bar{H}$ where $H = H^g = H_1^{g_1} H_1^{g_2 *} \cdots H_n^{g_1} H_n^{g_2 *}$ and $\bar{H} =$*

$H_1^{g_1} H_{1_1}^{g_2*} \cdots H_{1_{f_1}}^{g_2\ *} \cdots H_h^{g_1} H_{h_1}^{g_2*} \cdots H_{h_{f_h}}^{g_2\ *}$ *where* $\forall i = 1, \cdots h(H_{i_1}^{g_1} H_{i_1}^{g_2*} \cdots H_{i_{f_i}}^{g_1} H_{i_{f_i}}^{g_2\ *}$ *in H s.t.* $H_i^{g_1} = H_{i_1}^{g_1} = \cdots = H_{i_{f_i}}^{g_1})$ *and* $\nexists i, j \in [1, \cdots, h](i \neq j \Rightarrow H_i^{g_1} \neq H_j^{g_1})$. □

Example 3.4 *Let* $\beta(\rho) = [A_\times B]^*$. *Let the tree be* $T = (\rho(A:1)(B:2)(A:1)(B:3)(A:2)(B:5))$. *Then, after* $nest(B)$,$\beta_1(\rho) = [A_\times B^*]^*$ *and* $\bar{T} = (\rho(A:1)(B:2)(B:3)(A:2)(B:5))$.

3.2 Transformation of Key Definition

We now define the transformation of keys. We assume that τ be a primitive transformation operator because if every τ transforms a key to a key valid, then a sequence of operators also transforms the key valid. In defining the transformation, we need to refer to the DTD type structure g and the paths $\wp$ of a key. We now define the notation.

To describe the relationship between a type structure g and a path $\wp$ on a DTD, we define $g \diamond \wp \triangleright e$, reading g **crossing** $\wp$ **at** e, to mean that element e is in the type structure g and is also a label on path $\wp$, that is $g = [\cdots e \cdots] \wedge \wp = e_1/\cdots/e/\cdots e_m$. e is called the *cross point* of g and $\wp$. Given D, τ, and $\Bbbk(Q, \{P_1, \cdots, P_l\})$, if a path $\wp$ in $\Bbbk$ is not crossed by the transformed structure $g \in D$, then $\bar{\wp} = \wp$. We note that each operator changes either Q or a path in $\{P_i\}$, but not both as P_i is a path that connects to Q to form a complete path. We now present the transformation of keys with regard to the transformation operations.

Let $\wp = e_1/\cdots e_{k-1}/e_k/e_{k+1}/\cdots/e_m$ be a path in $\Bbbk$. The operators *nest* and *unnest* do not change a key because they manipulate the multiplicities of a type structure but do not change paths. In other words, $\tau(\Bbbk) = \Bbbk$ meaning $\forall \wp \in \Bbbk, \tau(\wp) = \wp$. We note that these two operators may still affect the satisfaction of a key, which will be shown in the next section.

3.2.1 Transformation on key using *expand*

If $g \in \beta(e_{k-1}) \wedge e_k \in g$ and $\tau = expand(g, e_{new})$ then

(a) If $\wp \in (\{Q\} \cup P)$ and $(g \diamond \wp \triangleright e)$ and $(last(\wp) \neq e_{k-1})$, then $\tau(\wp) \rightarrow \bar{\wp} = e_1/\cdots/e_{k-1}/e_{new}/e_k/e_{k+1}/\cdots/e_m$.

(b) If $last(Q) = e_{k-1} \wedge \forall \wp \in P(beg(\wp) \in g)$, then
  (1) Option 1: $\bar{Q} = Q/e_{new}$ and $\forall \wp \in P(\bar{\wp} = \wp)$.
  (2) Option 2: $\bar{Q} = Q$ and $\forall \wp \in P(\bar{\wp} = e_{new}/\wp)$.

(c) If $last(Q) = e_{k-1} \wedge \exists \wp \in P(beg(\wp) \notin g)$, then $\bar{Q} = Q$ and $\forall \wp \in P \wedge beg(\wp) = e_k(\bar{\wp} = e_{new}/\wp)$.

Example 3.5 *We recall the example 2 in the introduction. In example,* $\Bbbk_{a_2}(enroll/dept, \{cid, sid\})$ *on* D_a *in Fig.1 is transformed to* $\Bbbk'_{a_2}(enroll/dept, \{course/cid, course/sid\})$ *as the DTD* D_a *is transformed to the DTD* D_c *in Fig.5. Note that we use the option 2 of (b) of the transformation rules using expand operation.*

3.2.2 Transformation on key using *collapse*

If $(\beta(e_k) = g \wedge e_{k+1} \in g)$ and $\tau = collapse(e_k)$, then $\tau(\wp) \rightarrow \bar{\wp} = e_1/\cdots/e_{k-1}/e_{k+1}/\cdots/e_m$.

$$<!ELEMENT\ enroll(dname, (cid, sid^+)^+)^+>$$

Figure 10: XML DTD D_d

Example 3.6 *Again, we recall the example 2 in the introduction. In example, if the transformation* $collapse(dept)$ *is applied on* D_a *in Fig.1, the key* $\Bbbk_{a_2}(enroll/dept, \{cid, sid\})$ *on* D_a *is transformed to* $\Bbbk'''_{a_2}(enroll, \{cid, sid\})$. *We show only the transformed DTD* D_d *in Fig.10 using* $collapse(dept)$. *Note that the path* $Q = enroll/dept$ *is transformed to* $\bar{Q} = enroll$.

Theorem 3.1 *Let* τ *be a transformation defined above such that* $\tau(\Bbbk) = \Bbbk$. *Then* $\bar{\Bbbk}$ *is valid on* $\bar{D}$, *denoted as* $\bar{\Bbbk} \sqsubset \bar{D}$.

Proof sketch: τ doesn't transform Q to empty and Q is a valid complete path on $\bar{D}$. *collapse* operator does not collapse a leaf node. So $\forall \wp \in P$, $\beta(last(\wp)) = Str$ is true in $\bar{\Bbbk}$. *collapse* and *expand* ensure that $\forall \wp \in P$, $Q/\wp$ is a valid complete path on $\bar{D}$.

4 Satisfaction of Transformed Keys and Key preservation

In this section, we investigate how a transformation affects the satisfaction of transformed keys. More specifically, given $\tau(D, T, \Bbbk) \rightarrow (\bar{D}, \bar{T}, \bar{\Bbbk}) \wedge \bar{\Bbbk} \sqsubset \bar{D}$, we investigate whether $\bar{T}$ satisfies $\bar{\Bbbk}$.

Definition 4.1 (Key Preservation) *Given the transformations on* $D, T, \Bbbk$ *as* $\tau(D, T, \Bbbk) \rightarrow (\bar{D}, \bar{T}, \bar{\Bbbk}) \wedge \bar{\Bbbk} \sqsubset \bar{D}$, *if* $T \prec \Bbbk$ *and* $\bar{T} \prec \bar{\Bbbk}$, *we say that* $\Bbbk$ *is preserved by the transformation* τ. □

4.1 Key Preserving Properties of Operators

We recall the definition of key satisfaction in which for the key $\Bbbk(Q, \{P_1, \cdots, P_l\})$, the satisfaction requires that, if every P-tuple $(T_1^{P_1} \cdots T_1^{P_l})$ under T_1^Q is value different, every P-tuple $(T_2^{P_1} \cdots T_2^{P_l})$ under T_2^Q is value different, and for every T_1^Q, T_2^Q, $(T_1^{P_1} \cdots T_1^{P_l}) \neq_v (T_2^{P_1} \cdots T_2^{P_l})$, then T_1^Q and T_2^Q are different. The definition indicates that when key satisfaction is studied, our focus is to see (a) whether any $T^Q \in \langle T^Q \rangle$ is changed by the transformation, (b) how the P-tuple $(T_1^{P_1} \cdots T_1^{P_l})$ is changed by the transformation, (c) whether new P-tuples like $(T_k^{P_1} \cdots T_k^{P_w})$ are produced as the *fields* in $\Bbbk$ are changed by the transformation. We now present two lemmas relating to DTD and document transformation and the proofs of the lemmas are straight forward from the definitions.

Lemma 4.1 *Given a P-tuple* $(T_k^{P_1}\cdots T_k^{P_l})$*, a tree* T^Q *in a document s.t.* $(T_k^{P_1}\cdots T_k^{P_l}) \in T^Q$*, and a transformation* τ*, if* τ *does not change* T^Q*,* τ *does not change* $(T_k^{P_1}\cdots T_k^{P_l})$.

Lemma 4.2 *If* $\forall p \in \Bbbk(\tau(\wp) = \wp)$ *and* $\tau(T) = T$*, then* $\Bbbk$ *is preserved.*

We now show the key preservation property of each operators.

Theorem 4.1 *The unnest operator is key preserving if a) the element structure* g_1 *doesn't cross the selector* Q*, or b) the element structure* g_1 *doesn't cross some fields* P_i.

Proof: With $H = H_1^{g_1} H_1^{g_2} H_2^{g_2}$, $unnest(H) \to \bar{H} = H_1^{g_1} H_1^{g_2} H_1^{g_1} H_2^{g_2}$. Note that $H_1^{g_1}$ is duplicated and $H_1^{g_2}, H_2^{g_2}$ are unchanged in $\bar{H}$. So if g_2 crosses a path Q or a path $\wp \in P$, then either the number of T^Q or the number of P-tuples is not changed in $\bar{T}$. But g_1 crosses a path Q or a path $\wp \in P$, then either the number of T^Q or the number of P-tuples can be changed in $\bar{T}$ as $H_1^{g_1}$ is duplicated using *unnest*. Now if g_1 and g_2 cross $\wp_1 \in P_1$ and $\wp_2 \in P_2$ respectively, then the number of P-tuples are unchanged as P-tuples are produced with the combination of paths $\wp_1$ and $\wp_2$. Note that both g_1 and g_2 can't cross Q. Now we discuss this as:

Case 1:$[g_2 \diamond Q \rhd e]$. Let T^e be a tree. Because $e \in Q$, so either $T^Q \in T^e$ or $T^Q = T^e$ and $(T^e \in H_1^{g_2} \wedge T^e \in H_2^{g_2})$. In either case, by Lemma 4.1, because T^e is not changed, so T^Q as well as P-tuple $(T^{\wp_1}\cdots T^{\wp_l})$ is not changed. Moreover, $|\langle T^Q\rangle|$ is not changed after *unnest* because $H_1^{g_2}$ and $H_2^{g_2}$ are not changed. Because $T \prec \Bbbk$, so $\bar{T} \prec \Bbbk$.

Case 2: $[g$ crosses some $\wp \in P]$. There are two subcases to consider.

Subcase 1 $(g_2 \diamond \wp_i \rhd e)$.
Consider a P-tuple $F_1 = (T_1^{\wp_1}\cdots T_1^{\wp_i}\cdots T_1^{\wp_l})$ under a tree T_1^Q and another P-tuple $F_2 = (T_2^{\wp_1}\cdots T_2^{\wp_i}\cdots T_2^{\wp_l})$ under a tree T_2^Q where $\wp_1 \in P_1, \cdots, \wp_i \in P_i, \cdots \wp_l \in P_l$, $T_1^{\wp_i} \in H_1^{g_2}$ and $T_2^{\wp_i} \in H_2^{g_2}$. As $T \prec \Bbbk$, so $F_1 \neq_v F_2$. After *unnest*, $H_1^{g_2}$ and $H_2^{g_2}$ are not changed. So F_1 and F_2 are not changed. Because $T \prec \Bbbk$, so $\bar{T} \prec \Bbbk$.

Subcase 2 $(g_1 \diamond \wp \rhd e_1$ and $g_2 \diamond \wp \rhd e_2)$.
Consider a P-tuple $F_1 = (T_1^{\wp_1}\cdots T_1^{\wp_j}\cdots T_1^{\wp_k}\cdots T_1^{\wp_l})$ under T_1^Q where $\exists \wp \in \wp_j \wedge e_1 \in \wp$ and $(T_1^{\wp_j} \in H_1^{g_1} \wedge T_1^{\wp_k} \in H_1^{g_2})$, and another P-tuple $F_2 = (T_2^{\wp_1}\cdots T_2^{\wp_j}\cdots T_2^{\wp_k}\cdots T_2^{\wp_l})$ under T_2^Q where $\exists \wp \in \wp_k \wedge e_2 \in \wp$ and $(T_2^{\wp_j} \in H_1^{g_1} \wedge T_2^{\wp_k} \in H_2^{g_2})$. If $T \prec \Bbbk$, then $F_1 \neq_v F_2$ in T. After *unnest*, F_1 and F_2 are not changed because the P-tuple F_1 is produced from $H_1^{g_1} H_1^{g_2}$ and P-tuple F_2 is produced form $H_1^{g_1} H_2^{g_2}$ in $\bar{H}$. So, $\bar{T} \prec \Bbbk$, if $T \prec \Bbbk$.

Note 1: $g_1 \diamond Q \rhd e$. In this case, though T^Q and $(T^{\wp_1}\cdots T^{\wp_l})$ are not changed according to Lemma 4.1, but $|\langle T^Q\rangle|$ can be changed because $H_1^{g_1}$ is duplicated in $\bar{H}$ after the *unnest* operation. Thus, the duplication of T^Q as T_1^Q, T_2^Q having P-tuples as $F_1 = (T_1^{\wp_1}\cdots T_1^{\wp_l})$ and $F_2 = (T_2^{\wp_1}\cdots T_2^{\wp_l})$ respectively where $F_1 =_v F_2$, can cause violation of key satisfaction in $\bar{T}$. So, $\bar{T} \not\prec \Bbbk$, if $T \prec \Bbbk$.

Note 2: $g_1 \diamond \wp_i \rhd e$. In this case, the element structure g_1 of g crosses $\wp_i$. Consider a P-tuple $F_1 = (T_1^{\wp_1}\cdots T_1^{\wp_i}\cdots T_1^{\wp_l})$ under a tree T_1^Q where $\wp_1 \in P_1, \cdots, \wp_i \in P_i, \cdots \wp_l \in P_l$ and another P-tuple $F_2 = (T_2^{\wp_1}\cdots T_2^{\wp_i}\cdots T_2^{\wp_l})$ under a tree T_2^Q. As $T \prec \Bbbk$, so $F_1 \neq_v F_2$. After *unnest*, there are duplicate P-tuples as F_1 and F_1' under T_1^Q where $F_1 =_v F_1'$, or F_2 and F_2' under T_2^Q where $F_2 =_v F_2'$ in $\bar{T}$ because $H_1^{g_1}$ is duplicated in $\bar{H}$. Thus, after *unnest*, $\bar{T} \not\prec \Bbbk$, if $T \prec \Bbbk$.

Example 4.1 *We recall the example given in the introduction as motivation. We showed that the tree* T_a *satisfies the key* $\Bbbk_{a_1}(enroll/dept, \{cid\})$ *on the DTD* D_a*. In* D_a*,* $g = [cid_{\times} sid^+]^+$ *where* $g_1 = cid$ *and* $g_2 = sid^+$*. So,* $g_1 = cid$ *crosses* $\wp_1 = cid$*. After* $unnest(sid)$*,* $g = [cid_{\times} sid]^+$ *and the hedge* $H^{g_1} = H^{cid}$ *is duplicated in* T_b*. Thus there are duplicated P-tuples* $(v_4 : cid : Phys01)$*,* $(v_6 : cid : Phys01)$ *in* T_{v_1} *and* $(v_{11} : cid : Chem02)$*,* $(v_{13} : cid : Chem02)$ *in* T_{v_2} *and* $\Bbbk_{a_1}$ *is not satisfied by* T_b*. The reason for this is that* g_1 *crossed the field path.*

Theorem 4.2 *The nest operator is key preserving.*

Proof sketch: With hedges $H = H_1^{g_1} H_1^{g_2} H_2^{g_1} H_2^{g_2}$, If $H_1^{g_1} =_v H_2^{g_1}$, then $nest(H) \to \bar{H} = H_1^{g_1} H_1^{g_2} H_2^{g_2}$; Otherwise, $H = \bar{H}$. So if g_1 crosses Q or a path $\wp \in P$, then $H_1^{g_1} \neq_v H_2^{g_1}$ and thus $H = \bar{H}$. So the key is preserved because of lemma 4.2. If g_2 crosses Q or a path $\wp \in P$, then key is preserved as $H_1^{g_1}$ and $H_2^{g_2}$ are not changed in $\bar{T}$ and thus the number of T^Q or the number of P-tuples are not changed in $\bar{T}$. If g_1 and g_2 cross $\wp_1 \in P_1$ and $\wp_2 \in P_2$ respectively, then $H = \bar{H}$ and key is preserved by lemma 4.2.

Theorem 4.3 *The expand operator is key preserving if when* $\bar{Q} = Q/e_{new}$*, then every* $T^{Q/e_{new}}$ *has a P-tuple.*

Proof: We consider the following cases.

Case 1$[\wp \in (\{Q\} \cup P) \wedge g \diamond \wp \rhd e_k]$. Let T^{e_k} be a tree. The *expand* adds e_{new} between e_{k-1} and e_k in $\wp$. By lemma 4.1, because T^{e_k} is not changed, so $T^{\wp}$ is not changed.

Case 2$[last(Q) = e_{k-1} \wedge \forall \wp \in P(beg(\wp) \in g)]$.
Option 1: The path Q is transformed to Q/e_{new} and all paths in P are not changed. A T^Q corresponds to Q before the transformation and a $T^{Q/e_{new}}$ corresponds to Q/e_{new} after the transformation. Under each T^Q, there may be multiple $T^{Q/e_{new}}$s, all $T^{Q/e_{new}}$s will divide the P-tuples of T^Q, and each $T^{Q/e_{new}}$ may contain less number of P-tuples in comparison to T^Q, but there is at least one P-tuple in each $T^{Q/e_{new}}$ because

of the condition of the theorem. By theorem 2.1, before the transformation, all P-tuples are distinct in T as $T \prec \mathbb{k}$ and in this case the transformation does not change any path in P and therefore any P-tuples. So all P-tuples are still distinct after the transformation. So $\bar{T} \prec \bar{\mathbb{k}}$.
Option 2: The path $\wp \in P$ are transformed to $\bar{\wp} = e_{new}/\wp$. So the $last(\wp) = last(\bar{\wp})$ and thus $T^{\wp} =_v T^{\bar{\wp}}$. So the P-tuples are not changed. Thus $\bar{T} \prec \bar{\mathbb{k}}$.
Case 3$[last(Q) = e_{k-1} \wedge \exists \wp \in P(beg(\wp) \notin g)]$. Assume $g \diamond \wp \rhd e_k$ and two trees: $T_1^{\wp}$ as $\cdots.e_{k-1}.e_k.\cdots.T_1^{\wp}$ and $T_2^{\wp}$ as $\cdots.e_{k-1}.e_k.\cdots.T_2^{\wp}$. The transformation changes the trees to $\cdots.e_{k-1}.e_{new}.e_k.\cdots.T_1^{\wp}$ and $\cdots.e_{k-1}.e_{new}.e_k.\cdots.T_2^{\wp}$. So if $T_1^{\wp} \neq_v T_2^{\wp}$ before the transformation, $T_1^{\wp} \neq_v T_2^{\wp}$ after the transformation. If $(T_1^{\wp_1}\cdots T_1^{\wp_l}) \neq_v (T_2^{\wp_1}\cdots T_2^{\wp_l})$ before the transformation, $(T_1^{\wp_1}\cdots T_1^{\wp_l}) \neq_v (T_2^{\wp_1}\cdots T_2^{\wp_l})$ after the transformation. So, $\bar{T} \prec \bar{\mathbb{k}}$ if $T \prec \mathbb{k}$.

Theorem 4.4 *The collapse operator is key preserving.*

Proof: We discuss the following cases.
Case 1$[g \diamond Q \rhd e_k]$. Let T^{e_k} be a tree. The *collapse* operator changes the path Q according to the transformation of key definition. Now, either $T^Q = T^{par(e_k)}$ or $T^Q \in T^{par(e_k)}$ in $\bar{T}$. In either case, $(T^{P_1}\cdots T^{P_l})$ is not changed and $(T^{P_1}\cdots T^{P_l}) \in T^Q$. However, there can be the change(possibly the decrease of T^Q) of $|\langle T^Q\rangle|$ which doesn't cause the violation of key satisfaction. Thus, as $T \prec \mathbb{k}$, so $\bar{T} \prec \bar{\mathbb{k}}$.
Case 2$[g$ crosses some $\wp_i$ at $e_k]$. Consider a P-tuple $F_1 = (T_1^{\wp_1}\cdots T_1^{\wp_i}\cdots T_1^{\wp_l})$ under a tree T_1^Q and another tree sequence $F_2 = (T_2^{\wp_1}\cdots T_2^{\wp_i}\cdots T_2^{\wp_l})$ under a tree T_2^Q. If $g \diamond \wp_i \rhd e_k$, after the transformation, the two P-tuples F_1 and F_2 become $\bar{F}_1 = (T_1^{\wp_1}\cdots T_1^{\bar{\wp}_i}\cdots T_1^{\wp_l})$ and $\bar{F}_2 = (T_2^{\wp_1}\cdots T_2^{\bar{\wp}_i}\cdots T_2^{\wp_l})$ where transformation of $\wp_i$ to $\bar{\wp}_i$ follows the transformation definition of key using collapse operator. Also, the number of P-tuples under T_1^Q, T_2^Q are not changed. If $F_1 \neq_v F_2$, then $\bar{F}_1 \neq_v \bar{F}_2$. This means that if $T \prec \mathbb{k}$, $\bar{T} \prec \bar{\mathbb{k}}$.

4.2 Reversibility of Transformation Operators with Key Preservation

We showed the key preservation properties of important transformation operators for XML. Another important property of these operators is that these operators have reversibility in terms of key preservation. By reversibility, we mean that if an operator is key preserving(with or without any condition), then its reverse operation is also key preserving(with or without any condition). For example, we showed that *unnest* operator is key preserving(surely with some conditions). The reverse operation of *unnest* is *nest* which is also key preserving. Note that the *nest* operator is key preserving without any conditions.

4.3 Transition of Keys in XML Data Transformation

We showed that the *nest* and *collapse* operators are key preserving without any conditions. Other two operators, *unnest* and *expand* are key preserving when necessary conditions are met. As we transform XML key using operators, there is a natural question whether XML keys are transformed to XML functional dependency(XFD) when keys are not preserved by transformation. The motivation behind this question is because we already showed that XML key is a special case of XFD[25] and *unnest* operator introduces the redundant tuple values in the transformed document due to multiplicity change in the DTD. We discussed this research issue in our paper [28] where we defined the XFD and how XML key can be transformed XFD when *unnest* operator is used and we termed this problem as *Key Transition*.

5 Preservation of Other XML Constraints in XML Data Transformation

In XML to XML data transformation, we not only consider XML key preservation, but also we consider XFD, another important integrity constraints for XML. We showed how XFDs are defined on the DTD and their satisfactions by XML documents, how XFDs are transformed by the transformation operations and to what degree the transformed XFDs are preserved after transformations in [29]. We note here that the XFD preservations are very different because of important reasons: (a)In key preservation, multiplicity in DTD is important and *nest* and *unnest* operations are more important and (b) In XFD preservation, XFDs can either Local XFD(LXFD) or Global XFD(GXFD) due to scope of the XML document and hence the operators *expand* and *collapse* are very important because these operators transform the scope. On the other hand, *nest* and *unnest* operators are always XFD preserving. We refer our paper[29] for interested readers.

Another important integrity constraint is XML foreign key(XFK). It can also be a natural question whether XFKs are preserved or not by the transformations. In this case, we note here that we already defined referential integrity for XML, namely XML inclusion dependency (XID) and XFK in [26]. We found that XIDs are always preserved after transformation but the preservation of XFKs are dependent on the preservation of XML keys because XFK is a restricted case of XID and XFK uses the definition of XML key.

6 Future Work

We showed the transformation of XML keys and the preservation of keys in XML data transformation using a set of primitive transformation operators. We now discuss the use of transformation and preservation of

XML keys in XML data transformation in the context of XML data integration.

6.1 Preserving Keys in XML Data Integration

In XML data integration, multiple source schemas are transformed and integrated to the target schema. When schemas are transformed and integrated to the target schema, XML data conforming to the schemas and the XML keys over the schemas, satisfying the XML data, can also be transformed and integrated. Thus, there is a problem of integrating the transformed keys to the target schema. We term the notion of integrating the transformed XML keys as *key merging* in XML data integration.

We plan to use the transformation operators *ojoin* and *xunion* [12, 32] for integrating the XML sources to the target schema. Thus, the effects on XML constraints using these two operators in XML data transformation and integration needs to be investigated. We argue that these operators are mainly for joining two XML DTDs with conforming data. So, the full treatment of these operators in the context of key transformation and preservation in XML data integration is different from the other operators perviously described.

When we use *ojoin* of two XML sources to the target schema for the key preservation checking, we need to consider different *join* of the relational database perspective. The task can be related with the referential integrity constraints(inclusion constraints, or foreign key). We argue that the key preservation of *ojoin* operator in XML can be done with the same line of relational database.

We also plan to use *xunion* operator for integrating two XML sources to the target schema. The operator *xunion* actually does the *aggegating* or *merging* of two XML sources. We don't show the key preservation of both *ojoin* and *xunion* operators in this paper because the task of integrating multiple sources to the target schema needs different approach which is beyond the scope of this paper.

6.2 XML Constraints in Peer-to-Peer Data Exchange and Data Integration

In recent years, XML data exchange and XML data integration in peer-to-peer(P2P) settings are getting much attention[30, 31]. In both cases, there is a need to transform the source XML schema with its conforming document to the target schema. Thus how XML constraints specially XML keys and XFDs are transformed and preserved in P2P setting need to be investigated. In addition, how XML constraints can be utilized for query processing and updating is also important.

6.3 Heterogeneous Data Model in Data Transformation and Integration

Another research worth investigating is to consider the different data models in transformation of schemas and their conforming data with constraints for integration purposes. For example, one source schema can be in relational data model with constraints(e.g., primary key, foreign key, functional dependency) and another source schema can be in XML schema(e.g., DTD) with XML constraints(e.g., XML keys, XML functional dependency). The target schema can be either in XML or in relational data model with or without constraints. Then how we should transform source schemas with constraints in any data model to target schema in any data model with constraints preservation becomes an obvious research issue. The main challenge will be how to define the XML constraints so that we can capture the characteristics of relational constraints. We argue here that our definition for XML key can capture the semantics of relational key and hence the task of key preservation issues in data transformation and integration with heterogeneous data model can be achieved with great ease.

7 Conclusions

We showed the preservation of key in XML data transformation. To accomplish this task, we defined the XML keys on DTD and their satisfactions. Then we showed the transformation of XML keys using important transformation operations and the transformed keys are valid on the transformed DTD. Last we showed the key preservation of the transformation operations with necessary and sufficient conditions. We plan to study the performance of checking XML key satisfactions for preservation along with transformation operations. Our study of key preservation in data transformation is towards the handling of XML constraints in XML data integration.

Acknowledgement: We thank to anonymous reviewers for their valuable comments and suggestions to improve our paper.

References

[1] M. Lenzerini, *Data Integration: A Theoretical Perspective, ACM PODS*, 2002, pp. 233-246.

[2] A. Y. Halevy, A. Rajaraman and J. J. Ordille, *Data Integration: The Teenage Years, VLDB*, 2006, pp.9-16.

[3] A. Cali, D. Calvanese, G. D. Giacomo and M. Lenzerini, *Data Integration under Integrity Constraints, CAISE,* LNCS 2348, 2002, pp. 262-279.

[4] D. Suciu, *On Database Theory and XML, SIGMOD Record*, 2001, vol.30, No.3, pp.39-45.

[5] H. Jiang, H. Ho, L. Popa, and W. Han, *Mapping-Driven XML Transforamtion, WWW 2007*, 2007, pp. 1063-1072.

[6] L. Zamboulis and A. Poulovassilis, *Using Automed for XML Data Transformation and Integration, DIWeb*, 2004, pp.58-69.

[7] L. Zamboulis, *XML Data Integration by Graph Restructuring, BNCOD*, 2004, pp.57-71.

[8] A. Poggi and S. Abiteboul, *XML Data Integration with Identification, DBPL,* 2005, pp. 106-121.

[9] M. Benedikt, C. Y. Chan, W. Fan, J. Freire and R. Rastogy, *Capturing both Types and Constraints in Data Integration, ACM SIGMOD*, 2003, pp.277-288.

[10] H. Su, H. Kuno and E. A. Rudensteiner, *Automating the Transformation of XML Documents, WIDM,* 2001, pp. 68-75.

[11] M. Erwig, *Toward the Automatic Derivation of XML Transformations, ER*, 2003, pp. 342-354.

[12] J. Liu, H. Park, M. Vincent and C. Liu, *A Formalism of XML Restructuring Operations, ASWC*, LNCS 4185, 2006, pp. 126-132.

[13] D. Barbosa, J. Freire and A. O. Mendelzon, *Information Preservation in XML-to-Relational Mappings, XSym,* LNCS 3186, 2004, pp. 66-81.

[14] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez and R. Fagin, *Translating the web data, VLDB*, 2002, pp. 598-609.

[15] S. Davidson, W. Fan, C. Hara and J. Qin, *Propagating XML Constraints to Relations, ICDE*, 2003, pp. 543-554.

[16] D. Lee and W. W. Chu, *Constraint Preserving Transformation from XML Document Type Definition to Relational Schema, ER,* 2000, LNCS 1920, pp. 323-338.

[17] Y. Liu, H. Zhong and Y. Wang, *XML Constraints Preservation in Relational Schema, CEC-East'04*, 2004.

[18] Q. Wang, H. Wu, J. Xiao and A. Zhou, *Deriving Relation Keys from XML Keys, ADC,* 2003.

[19] Yunsheng Liu, Hao Zhong, and Yi Wang, *Capturing XML Constraints with Relational Schema, CIT,* 2004.

[20] P. Buneman, S. Davidson, W. Fan, C. Hara and W. C. Tang, *Keys for XML, WWW10*, 2001, pp.201-210.

[21] P. Buneman, S. Davidson, W. Fan, C. Hara and W. C. Tan, *Reasoning about Keys for XML, DBPL,* 2002, LNCS 2397, pp. 133-148.

[22] W. Fan and J. Simeon, *Integrity constraints for XML, PODS*, 2000, pp.23-34.

[23] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen, Extensible Markup Language (XML) 1.0., World Wide Web Consortium (W3C), Feb 1998. `http://www.w3.org/TR/REC-xml`.

[24] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn, XML Schema Part 1:Structures, W3C Working Draft, April 2000. `http://www.w3.org/TR/xmlschema-1/`.

[25] Md. Sumon Shahriar and Jixue Liu, *On Defining Keys for XML, citworkshops,* pp. 86-91, 2008 IEEE 8th International Conference on Computer and Information Technology Workshops, 2008.

[26] Md. Sumon Shahriar and Jixue Liu, *On Defining Referential Integrity for XML, IEEE CSA 2008,* pp. 286-291.

[27] W. Fan and L. Libkin, *On XML Integrity Constraints in the Presence of DTDs, Journal of the ACM*, 2002, vol.49, pp. 368-406.

[28] Md. Sumon Shahriar and Jixue Liu, *Transition of Keys in XML Data Transformation, IEEE CSA 2008,* pp. 175-180.

[29] Md. Sumon Shahriar and Jixue Liu, *Preserving Functional Dependency in XML Data Transformation, ADBIS 2008,*LNCS 5207, pp. 262-278.

[30] G. Koloniari and E. Pitoura, *Peer-to-peer Management of XML Data: Issues and Research Challenges. SIGMOD Record, 34(2):6-17,* 2005.

[31] S. D. Gribble, A. Y. Halevy, Z. G. Ives, M. Rodrig, and D. Suciu, *What Can Database Do for Peer-to-Peer? In Proc. of WebDB,*2001.

[32] J. Liu, Ho-Hyun Park, M. Vincent and C. Liu , *Transformation of XML Data, extended version,* `http://www.cis.unisa.edu.au/~cisjl/publications/publ.html`.

[33] Md. S. Shahriar and J. Liu, *Preserving key in XML Data Transformation, extended version,* `http://www.cis.unisa.edu.au/~cisjl/publications/publ.html`.