International Journal of Security and its Applications
Vol. 3, No. 1, January, 2009

On Transiting Key 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, Adelaide, SA-5095, Australia
E-mail: shamy022@students.unisa.edu.au, Jixue.Liu@unisa.edu.au

Abstract

Transformation of a source schema with its conforming data to a target schema is an important activity in data integration with any data model. In last decade, with the advent of XML as an widely used and adopted data representation and storage format over the web, the task of data transformation for the purpose of data integration solely in XML is getting much attention to the database researchers and application developers. In XML data transformation, when an XML source schema with its conforming data is transformed to the target XML schema, one of the important XML constraints, XML keys that are defined on the source schema for expressive semantics can also be transformed. Thus, whether keys should be transformed and preserved, and if not preserved, whether keys can be captured in another form of XML constraints are important research questions. To answer these questions, first, we define XML keys and XML Functional Dependencies(XFD) on Document Type Definition(DTD). Second, we show key preservation in transformation. If keys are not preserved, we then show how to capture them as XFDs. We term this as **key transition**. *Our research on XML key preservation and transition is towards handling the issues of integrity constraints in XML data integration.*

1. Introduction

Transformation of data is regarded as an important activity in data integration, data exchange, data translation, data migration, data publishing with any data model[1, 2]. Specially, in data integration, transformation of a source schema with its conforming data to a target schema with its conforming data is an important task. In recent years, with the growing use of XML as data representation and storage format over the web[10], the task of data transformation for the purpose of XML data integration is getting much attention to the database community.

In XML data transformation, a source schema with its conforming data is transformed to the target schema with its conforming data. An XML source schema is often defined with XML constraints[27, 28, 29, 30] to convey integrity and semantics of data. The important XML constraints are XML key and XML functional dependency. When the schema with its conforming data is transformed, the constraints can also be transformed. We illustrate this problem using simple motivating example.

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

A motivating example: We give here the motivating example. Consider the DTD D_a in Figure 1 that depicts the enrollment of students in the courses for each department. Each department has name $dname$, each course has id cid, and each student has id sid. In each department, there are many courses and students are enrolled each course. It is quite natural that courses can be unique not only in the departments but also in the university. Now consider the XML key on D_a as $\mathbb{k}_{a_1}(enroll/dept, \{cid\})$ 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 fields as `selector/field` are valid paths 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 Figure 2 because under the selector nodes v_1 and v_2, the values $(v_4 : cid : Phys01)$, $(v_7 : cid : Phys02)$, and $(v_{10} : cid : Chem02)$ are distinct. We see that the XML document T_a in Figure 2 is satisfied by the key $\mathbb{k}_{a_1}$.

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

Figure 1. XML DTD D_a

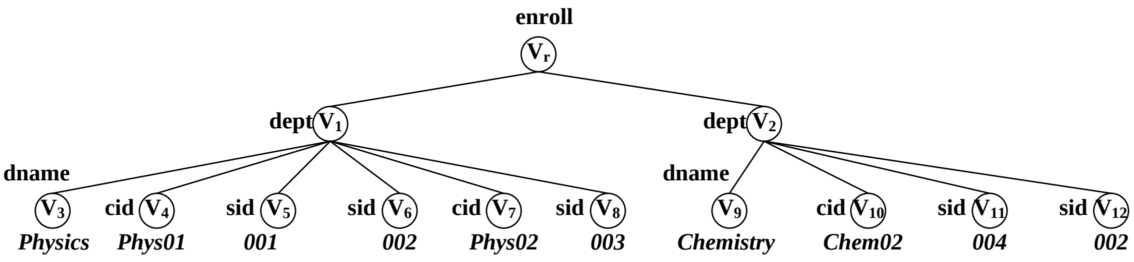

Figure 2. XML Tree T_a

If we observe the DTD D_a and the XML key $\mathbb{k}_{a_1}$, we get an obvious conclusion that students can enroll course-wise and course id with its value is distinct in the whole document.

Now we want to transform the DTD D_a and the document T_a to the DTD D_b and the document T_b. The purpose of this transformation is to make the nested structure $(cid, sid^+)^+$ to flat structure $(cid, sid)^+$. We term this transformation as $unnest$. This transformation operation is widely found in data transformation with any data model when it is necessary to unnest a nested relation or schema. Specially, it is important when we want to store or to represent XML data to relational data [16, 17, 18, 19].

We see that the key $\mathbb{k}_{a_1}$ is still valid on D_b. The reason is that no path is transformed or changed in the key. Now our question is whether the key is still satisfied by the transformed document T_b.

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

Figure 3. XML DTD D_b

We see that the key $\mathbb{k}_{a_1}$ is not satisfied by the transformed document T_b as the there are duplicate course id generated in transformation.

Observation 1.1 *An XML Key may not be preserved after the transformation.*

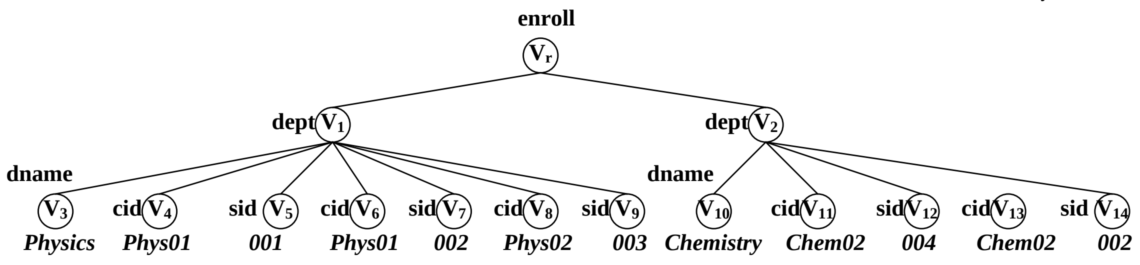

Figure 4. XML Tree T_b

Now our next question is whether XML key $\mathbb{k}_{a_1}$ can be captured to XML functional dependency(XFD). The motivation behind this question is because XML key is the restricted case of XFD[33, 34]. We say that $\mathbb{k}_{a_1}$ can be transformed to $\Phi_{a_1}(enroll/dept, \{cid\} \rightarrow \epsilon)$ meaning that cid determines $dept$. For now, by the satisfaction of Φ_{a_1}, we mean that in each tree rooted at $dept$, there is at least one cid element with its value(the formal definition of XFD and its satisfaction will be given later). We find that the Φ_{a_1} is satisfied by the transformed document T_b. We say that the key $\mathbb{k}_{a_1}$ is **transited** to XFD Φ_{a_1}.

Observation 1.2 *An XML Key can be transformed to an XML functional dependency in data transformation.*

While addressing the problems from the observations, we consider the following contributions.

- First, we define the XML keys and XFDs over DTD and their satisfactions [33, 34]. We consider the DTD because of its simpler design over the XML Schema[32]. While defining, we introduce a novel concept, **tuple** for generating semantically correct tuples both for XML keys and XFDs.

- Second, we show whether the XML keys should be transformed using important transformation operator. In addition, we check whether the transformed keys are valid on the transformed DTD. We then check the satisfaction of the transformed keys by the transformed documents(we say $key\ preservation$).

- Third, if the XML key is not preserved, we then check whether XML key can be transformed to XFD. We term this problem as $key\ transition.$

- Lastly, we discuss how the transition of XML keys to XFDs in data transformation is important in pure XML data integration.

2. Related Work

Constraints played an important role in data transformation and integrations involving different data models. We categorize the transformations as followings.
Relational-to-Relational[R-R]: In pure relational data integration [3], how queries should be affected in the presence of keys and foreign keys on the global schema and no constraints on the source schema. Chen Li[4] showed a data integration system where both source schemas and the global schema can have constraints and then he showed how the source derived global constraints and the original constraints on the global constraints can be further be used to answer the query.

In [5], how the consistent local constraints on the local schemas are transformed and simplified to the global schema is shown. In relational schema integration, the correspondence between local integrity constraints and the global extensional assertions is investigated in [6]. In [7], the query preserving transformation in data integration system is shown with or without constraints on the global schema. In [8], how the integrity constraints over the global schema can affect the query answering is discussed. In [9], the inconsistency of a data integration system is illustrated when source constraints over the source schemas are not same as global constraints over the global schema.

Relational-to-XML[R-X]: In relational to XML data transformation, the issue of constraints preservation is studied in [25] where the constraints like primary keys, foreign keys, unique constraints, and not null are considered when a relation is transformed to XML Schema. In publishing data in XML from XML and relational data [26], constraints are also exploited for better query formulation. In data translation[19], both XML and relational schemas are considered with some constraints like nested referential constraints.

XML-to-Relational[X-R]: In [21], constraints(e.g., cardinality constraints, domain constraints, inclusion dependencies etc.) are considered for preservation when XML DTD is transformed to relational schema. In [20], XML keys are transformed to functional dependencies(FDs) for relations and this process is termed as constraint propagation. In [23], how relational keys can be captured from XML keys is shown. Some work[22, 24] where constraints like cardinality constraints, inclusion dependencies, constraints on DTD etc. are considered where preservation is the issue for XML to relational data transformation perspective.

XML-to-XML[X-X]: In [14], XML keys and foreign keys are taken into consideration on the XML global schema where source schemas are also in XML. In [15], data from relational sources are integrated to the XML target schema where keys and foreign keys in relations are captured as XML keys and XML inclusion constraints on the target schema using constraint compilation. However, in XML to XML data transformations and integration [11, 12, 13, 16, 17, 18], how the important XML constraints(e.g. XML Keys, XML Functional Dependencies) on the source schema should be transformed, preserved, or transited to the target schema is little investigated to the best of our knowledge.

Organization: Our paper is organized as follows. We give the necessary definitions and notations in Section 3. In Section 4, we define the transformation of both XML key and XFDs using important transformation operations. We then define the transition of XML constraints in Section 5. In Section 7, we conclude with some remarks and future works.

3. Preliminaries

In this section, we give the basic notation and definitions needed throughout the paper.

Our model is XML DTD[31] 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 3.1 *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 3.1 *The DTD in Figure 1 can be represented as $D_a = (EN, G, \beta, \rho)$ where $EN = \{enroll, dept, dname, sid, cid\}$, $G = \{Str, [dept]^+, [dname{\times}[sid{\times}cid^+]^+]\}$, $\beta(enroll) = [dept]^+$, $\beta(dept) = [dname{\times}[cid{\times}sid^+]^+]$, $\beta(dname) = Str$, $\beta(sid) = Str$ and $\beta(cid) = Str$.*

Definition 3.2 *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 3.2 *The XML tree T_a in Figure 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 Figure 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 3.3 (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 $\forall i = 1, \cdots, f(H_i \Cap g)$ 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 3.4 (Tree Conformation) *Given a DTD $D = (EN, G, \beta, \rho)$ and XML Tree T, T conforms to D denoted by $T \Cap D$ if T=(ρ $H^{\beta(\rho)}$).* □

Definition 3.5 (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 $i = 1, \cdots, m(T_i =_v T_i')$ □

$T_x \equiv T_y$ 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 3.6 (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. □

Example 3.3 *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 3.7 (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. We define intersected path $\wp_1 \cap \wp_2 = e_1/\cdots/e_i$ where $j \in [1, \cdots, i](e_j = e_j')$ and $e_{i+1} \neq e_{i+1}'$. Paths satisfying this definition are said **valid** on D.* □

Example 3.4 *In Figure 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 $len(enroll/dept/sid) = 3$.*

Now we define the XML key and its satisfaction.

Definition 3.8 (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.* $\Box$

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 3.8 is not satisfied.

Example 3.5 *Let $\Bbbk_{a_1}(enroll/dept, \{cid\})$ be a key notation on D_a in Figure 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 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 3.6 *We define T^{dept} to mean the tree T_{v_1} or T_{v_2} in T_a of Figure 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 3.7.

$$\begin{array}{ll} <!ELEMENT & db(univ^+)> \\ <!ELEMENT & univ(dept, staff^+)^+> \\ <!ELEMENT & staff(fname, lname)> \end{array}$$

Figure 5. XML DTD D

Example 3.7 *Consider an XML key on D in Figure 5 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 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.*

Definition 3.9 (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.*

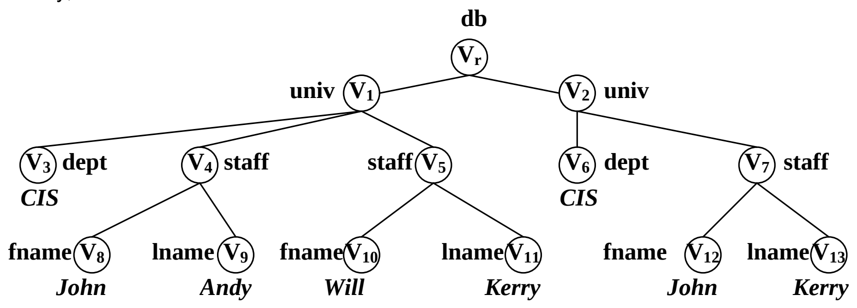

Figure 6. XML Tree

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 $\beta(e_k)$. $\Box$

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 3.10 (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.* $\Box$

Example 3.8 *$\Bbbk_{a_1}(enroll/dept, \{cid\})$ be a key notation on D_a in Figure 1. We want to check whether the key $\Bbbk_{a_1}$ is satisfied by the document T_a. We see the scope is dept. So for the node v_1, we get the P-tuples $F_1 = (T_{v_4})$ and $F_2 = (T_{v_7})$. For the node v_2, the P-tuple is $F_3 = (T_{v_{10}})$. As the P-tuples are value distinct for nodes v_1, v_2, the XML document T_a satisfies the key $\Bbbk_{a_1}$.*

Let $\Bbbk_{a_2}(enroll, \{dept/cid, dept/sid\})$ be a key on D_a in Figure 1. We want to check whether $\Bbbk_{a_2}$ is satisfied by the XML document T_a in Figure 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 $T_a \prec \Bbbk_{a_2}$.

Now consider another key $\Bbbk_{a_3}(enroll, \{dept/sid\})$. So, for v_r, there are P-tuples $F_1 = (T_{v_5})$, $F_2 = (T_{v_6})$, $F_3 = (T_{v_8})$ for node v_1 and $F_4 = (T_{v_{11}})$, $F_5 = (T_{v_{12}})$ for node v_2. But as $F_2 =_v F_5$, so $T_a \not\prec \Bbbk_{a_3}$.

Theorem 3.1 *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.*

We define the XML functional dependency here.

Definition 3.11 (Functional Dependency) *An XML functional dependency over the XML DTD can be defined as $\Phi(S, P \rightarrow Q)$ where S is a complete path, P is a set of simple paths as $\{\wp_1, \cdots, \wp_i, \cdots \wp_l\}$, and Q is a simple path or empty path. S is called* **scope**, *P is called* **LHS** *or* **determinant**, *and Q is called* **RHS** *or* **dependent**. *S/P and S/Q are complete paths.* $\Box$

If $Q = \epsilon$, then XFD is $\Phi(S, P \rightarrow \epsilon)$. It implies that $P \rightarrow last(S)$ meaning that P determines S. An XFD following the above definition is valid, denoted as $\Phi \sqsubset D$.

Example 3.9 *Consider the XFD $\Phi_{a_2}(enroll/dept, \{cid\} \rightarrow dname)$ on the DTD D_a in Figure 1. Here, $S = enroll/dept$ is a complete path. $P = \{cid\}$, and $Q = dname$ are simple paths. We see the paths $enroll/dept/cid$ and $enroll/dept/dname$ are also complete paths.*

Consider another XFD $\Phi_{a_1}(enroll/dept, \{cid\} \rightarrow \epsilon)$ meaning that cid determines dept.

We denote $F[P] = (T^{\wp_1} \cdots T^{\wp_l})$ as P-tuple and $F[Q] = (T^{\wp_{l+1}})$ as Q-tuple. A P-tuple $F[P]$ is complete if $\forall\, T^{\wp_i} \in (T^{\wp_1} \cdots T^{\wp_l})(T^{\wp_i}$ is complete)). Similarly, a Q-tuple is complete if $T^{\wp_{l+1}}$ is complete.

We use $\langle F[P] \rangle$ to denote all possible P-tuples and $|\langle F[P] \rangle|$ means the number of such P-tuples. Two P-tuples $F_1[P] = (T_1^{\wp_1} \cdots T_1^{\wp_l})$ and $F_2[P] = (T_2^{\wp_1} \cdots T_2^{\wp_k})$ are value equivalent, denoted by $F_1[P] =_v F_2[P]$ if $l = k$ and for each $i = 1, \cdots, k$ $(T_1^{\wp_i} =_v T_2^{\wp_i})$. Similarly, we denote $F_1[Q] =_v F_2[Q]$ to mean $F_1[Q]$ and $F_2[Q]$ are value equivalent where $F_1[Q] = (T_1^{\wp_{l+1}})$ and $F_2[Q] = (T_2^{\wp_{l+1}})$.

Example 3.10 *Consider the XFD $\Phi_{a_2}(enroll/dept, \{cid\} \rightarrow dname)$ on the DTD D on Figure 1. We take the paths $\wp_1 = cid$ and $\wp_2 = dname$ to generate the tuples. So the minimal structure for these paths is $[dname_\times[cid_\times sid]^+]$ because the paths have intersected path $\wp_1 \cap \wp_2 = \epsilon$ and then $dname \neq cid$. The minimal hedges for the structure $[dname_\times[cid_\times sid]^+]$ are $H_1 = T_{v_3}T_{v_4}T_{v_5}T_{v_6}T_{v_7}T_{v_8}$ in the T_{v_1} and $H'_1 = T_{v_9}T_{v_{10}}$
$T_{v_{11}}T_{v_{12}}$ in the T_{v_2}.*

Now we are ready to generate the tuples from H_1 and H'_1.

To generate the P-tuple meaning that tuples for the paths P, we now consider the paths $\wp_1 =$ cid. So in T_{v_1}, P-tuples are $F_1[P] = (T_{v_4}) = ((v_4 : cid : Phys01)), F_2[P] = (T_{v_7}) = ((v_7 : cid : Phys02))$ for H_1. For Q-tuple, meaning that tuples for the paths Q, we take the path $\wp_2 = dname$. So in T_{v_1}, the Q-tuples are $F_1[Q] = (T_{v_3}) = ((v_3 : dname : Physics))$ for H_1.

In T_{v_2}, the P-tuples are $F'_1[P] = (T_{v_{10}}) = ((v_{10} : cid : Chem02))$ for H'_1. The Q-tuples are $F'_1[Q] = (T_{v_9}) = ((v_9 : dname : Chemistry))$ for H'_1.

Definition 3.12 (Functional Dependency Satisfaction) *Given a DTD D, an XML document T satisfies the XML functional dependency $\Phi(S, P \rightarrow Q)$, denoted as $T \prec \Phi$ if the followings are held.*

(a) If $Q = \epsilon$, then $\forall F[P] \in T^S$, $F[P]$ is complete.

(b) Else

(i) $\exists(F[P], F[Q]) \in T^S$ and $F[P], F[Q]$ are complete.

(ii) For every pair of tuples F_1 and F_2 in T^S, if $F_1[P] =_v F_2[P]$, then $F_1[Q] =_v F_2[Q]$. □

Example 3.11 *Consider the XFD $\Phi_{a_2}(enroll/dept, \{cid\} \rightarrow dname)$ on the DTD D_a. We want to check whether the document T_a satisfies the XFD Φ_{a_2}. As the scope is $enroll/dept$, we see that there are two trees rooted at dept. These are T_{v_1} and T_{v_2}. Now we refer to the Example 3.10 for tuples generated by the paths. As $F_1[P] \neq_v F_2[P]$ in T_{v_1} and also $F_1'[P]$ and $F_1'[Q]$ are complete in T_{v_2}, so T_a satisfies Φ_{a_2}.*

Now consider the XFD $\Phi_{a_1}(enroll/dept, \{cid\} \rightarrow \epsilon)$. Here, the scope is dept. Then $T_a \prec \Phi_{a_1}$ because all the P-tuples $F[P] = (T^{cid})$ are complete in **each** *T^{dept}.*

4. Transformation of XML Data

We study here the important transformation operations τ for transforming a source XML DTD and its conforming data to target DTD and its conforming data.

UnNest Operation. We give the procedure for transforming a source DTD D_a with its conforming document T_a to a target DTD D_b and its conforming document T_b using $UnNest$ operation. We use top bar($^-$) to mean the transformed result. Formally, $\tau(D,T) \rightarrow (\bar{D}, \bar{T})$ where $\tau = UnNest, T \Subset D$ and $\bar{T} \Subset \bar{D}$.

Definition 4.1 (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 $\bar{H} = H_1^{g_1} H_{11}^{g_2} \cdots H_1^{g_1} H_{1n_1}^{g_2} \cdots H_m^{g_1} H_{m1}^{g_2} \cdots H_m^{g_1} H_{mn_m}^{g_2}$.* □

Example 4.1 *Consider the DTD D_a in Figure 1. In D_a, we consider the structure $g = [cid_\times sid^+]^+$ where $g_1 = cid$, $g_2 = sid$, $c_2 = +$ and $c = +$. So, the hedges conforming to g are $H_1^g = T_{v_4} T_{v_5} T_{v_6}$, $H_2^g = T_{v_7} T_{v_8}$ in T_{v_1} and $H_3^g = T_{v_{10}} T_{v_{11}} T_{v_{12}}$ in T_{v_2}.*

Then $UnNest(sid) \rightarrow \bar{g} = [cid_\times sid]^+$. The transformed DTD is now D_b in Figure 3. Thus the transformed hedges are $\bar{H}_1^{\bar{g}} = T_{v_4} T_{v_5} T_{v_6} T_{v_7} T_{v_8} T_{v_9}$ in T_{v_2} and $\bar{H}_2^{\bar{g}} = T_{v_{11}} T_{v_{12}} T_{v_{13}} T_{v_{14}}$ in T_{v_2} for the transformed document T_b in Figure 4.

Transformation on XML Keys using Unnest. We check the effect of transformation operation $UnNest$ on XML keys. 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 $g \diamond \wp \rhd e$, 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 the 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. Let $\wp = e_1/\cdots e_{k-1}/e_k/e_{k+1}/\cdots/e_m$ be a path in $\Bbbk$. The operator $unnest$ doesn't not change a key because they manipulate the multiplicities of a type structure but doesn't not change paths. In other words, $\tau(\Bbbk) = \Bbbk$ meaning $\forall \wp \in \Bbbk, \tau(\wp) = \wp$.

XML Key Preservation. Now we investigate whether the key is preserved after the transformation.

Definition 4.2 (Key Preservation) *Given the transformation on $D, T, \mathbb{k}$ as $\tau(D, T, \mathbb{k}) \rightarrow (\bar{D}, \bar{T}, \bar{\mathbb{k}}) \wedge \bar{\mathbb{k}} \sqsubset \bar{D}$, if $T \prec \mathbb{k}$ and $\bar{T} \prec \bar{\mathbb{k}}$, we say that $\mathbb{k}$ is preserved by the transformation τ.* $\square$

We give here an example that shows the key preservation.

Example 4.2 *Consider the XML key $\mathbb{k}_{a_2}(enroll/dept, \{cid, sid\})$ on D_a in Figure 1. We see $T_a \prec \mathbb{k}_{a_2}$. After $UnNest$, the key is not transformed and so is valid on D_b. We see that the key $\mathbb{k}_{a_2}$ is satisfied by the transformed document T_b, that is $T_b \prec \mathbb{k}_{a_2}$. Thus $\mathbb{k}_{a_2}$ is preserved.*

Now we recall the motivating example in the introduction where we considered the key $\mathbb{k}_{a_1}(enroll/dept, \{cid\})$. We showed how the key is not preserved by the transformation $UnNest$. We give the following theorem for key preservation.

Theorem 4.1 *The $UnNest$ operator defined as $unnest(g_2) \rightarrow [g_1 \times g_2^{c_2 \ominus +}]^{c \oplus +}$ is key preserving if a) the element structure g_1 doesn't cross the selector Q, or b) the element structure g_2 crosses some fields P_i of an XML key.*

5. Transition of XML keys in XML Data Transformation

In this section, we study how XML key can be transformed to XFD in order to capture the transformed constraint. We first show how an XML key is transformed to an XFD as constraint transition.

We give the formal definition of key transition.

Definition 5.1 (Key Transition) *We define key transition as $\tau(D, T, \mathbb{k}) \rightarrow (\bar{D}, \bar{T}, \Phi)$ where $T \Subset D, \mathbb{k} \sqsubset D, T \prec \mathbb{k}, \bar{T} \Subset \bar{D}, \Phi \sqsubset \bar{D}$ and $\bar{T} \prec \Phi$.*$\square$

We give here the key transition algorithm.

Algorithm 5.1 (Key Transition) `Input`: *XML source DTD D that is conformed by the XML source document T, XML key $\mathbb{k}$ valid on source DTD D, and T satisfies $\mathbb{k}$.* `Output`: *XML target DTD $\bar{D}$ that is conformed by the XML target document $\bar{T}$, XFD Φ valid on $\bar{D}$, and $\bar{T}$ satisfies Φ.*

```
1: check=CheckKeyTransformation($\mathbb{k}$, $UnNest$);
2: if check=TRUE then
3:     TransformKeyToXFD($\mathbb{k}$);
4: end if
5: if $\bar{T}$ satisfies the XFD $\Phi$ then
6:     return $\Phi$ and "$KeyTransited$";
7: end if
```

Function 5.1 *CheckKeyTransformation($\mathbb{k}$, $UnNest$)*

```
1: if $g_1$ crosses the selector $Q$ or $g_2$ doesn't cross any fields $P_i \in [P_1, \cdots, P_n]$ then
2:     return TRUE;
3: else
4:     return FALSE;
5: end if
```

Function 5.2 *TransformKeyToXFD($\mathbb{k}$)*

```
1: $\Phi[S] := \mathbb{k}[Q]$;
```

```
2: for all $i$ such that $1 \leq i \leq n$ do
3:     $\Phi[P_i] := \mathbb{k}[P_i]$;
4: end for
5: $\Phi[Q] := \epsilon$;
6: return $\Phi(S, \{P\} \rightarrow Q)$;
```

Example 5.1 *We give here an example how an XML key can be transited to an XFD. We consider the key $\mathbb{k}_{a_1}(enroll/dept, \{cid\})$ given in the introduction. We see that the key is valid on D_a and is satisfied by the document T_a. After $UnNest(sid)$, we get the DTD D_b and the transformed document T_b conforms to D_b. The key is no longer preserved by T_b as the structure $g_1 = cid$ in DTD D_a is crossed by the field $P_1 = cid$ of the key $\mathbb{k}_{a_1}$ at the common element cid(according to the theorem 4.1). We transform the key to XFD $\Phi_{a_1}(enroll/dept, \{cid\} \rightarrow \epsilon)$ which is valid on D_b and T_b satisfies Φ_{a_1}.*

Theorem 5.1 *If a key is satisfied by the source document but violates condition (a) or (b) of Theorem4.1, the key is transformed to an XFD on the target DTD.*

6. Transition of Key with Other Operators

We have shown how an XML key can be transited as XFD when the UnNest operator is used in transformation with necessary condition. Some other important transformation operations are $Nest$, $Expand$ and $Collapse$[18]. $Nest$ operator doesn't transform key definition and the key is preserved after transformation. $Expand$ and $Collapse$ operators transform the key definition but key is preserved after transformation. Thus, for $Nest$, $Expand$ and $Collapse$ operations key is not transited as XFD to the target schema.

Basically, key transition is a degraded way of representing the key constraint. In case a key is not changed by the transformation or it is changed but preserved by the transformation, we do not degrade the key. In other words, the semantics of the original key is represented by the new key and not by an XFD.

7. Conclusions

We investigated how an XML key can be propagated as an XFD in data transformation. Towards this problem, we defined XML keys and XFDs using a novel concept $tuple$ for this purpose. An important transformation operation $UnNest$ is used to capture the XML key on source schema as XFD to the target schema. We found the cases when the XML key can't be preserved and therefore can be transited as XFD. We also discussed some other important transformation in the sense of key transition and use of key transition in XML data integration purposes.

Our research on key transition can be used for data integration when multiple source schemas with their keys are transformed to the global schema and the keys may no longer be preserved by the global data and the keys can be captured as XFDs which is less restricted than XML keys.

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] C. Li, Describing and utilizing Constraints to Answer Queries in Data Integration Systems,IIWeb, 2003.

[5] H. Christiansen and D. Martinenghi, Simplification of Integrity Constraints for Data Integration, FoIKs, LNCS 2942, 2004, pp. 31-48.

[6] C. Turker and G. Saake, Consistent Handling of Integrity Constraints and Extensional Assertions for Schema Integration, ADBIS, LNCS 1691, 1999, pp. 31-45.

[7] A. Cali, D. Calvanese, G. D. Giacomo and M. Lenzerini, On the Expressive Power of Data Integration Systems, ER 2002, LNCS 2503, 2002, pp. 338-350.

[8] A. Cali, D. Calvanese, G. D. Giacomo and M. Lenzerini, On the Role of Integrity Constraints in Data Integration, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2002.

[9] A. Fuxman and R. e. J. Miller, Towards Inconsistency Management in Data Integration Systems, IIWeb-03, 2003.

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

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

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

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

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

[15] 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.

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

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

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

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

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

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

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

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

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

[25] C. Liu, M. W. Vincent, and J. Liu, Constraint Preserving Transformation from Relational schema to XML Schema, World Wide Web: Internet and Web Information Systems, 9, 93-110, 2006.

[26] A. Deutsch, and V. Tannen, MARS: A System for Publishing XML from Mixed and Redundant Storage, VLDB, 2003.

[27] P. Buneman, W. Fan, J. simeon and S. Weinstein, Constraints for Semistructured Data and XML, SIGMOD Record, 2001, pp. 47-54.

[28] W. Fan, XML Constraints: Specification, Analysis, and Applications, DEXA, 2005, pp.805-809.

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

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

[31] 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`.

[32] 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/`.

[33] Md. S. Shahriar and J. Liu, On Defining Keys for XML, CIT 2008, pp. 86-91.

[34] Md. S. Shahriar and J. Liu, Preserving Functional Dependency in XML Data Transformation, ADBIS 2008, LNCS 5207, pp. 262-276.

Biography

Md.Sumon Shahriar: Sumon Shahriar is currently PhD researcher in Data and Web Engineering Lab, School of Com puter and Information Science, University of South Australia. He achieved his Bachelor of Science (Honours) and Master of Science (Research) degrees both with first class in Computer Science and Engineering from University of Dhaka, Bangladesh. His research interests include XML database, Data Integration, Data Quality and Data Mining.

Dr. Jixue Liu: Jixue Liu got his bachelor's degree in engineering from Xian University of Architecture and Technology in 1982, his Masters degree (by research) in engineering from Beijing University of Science and Technology in 1987, and his PhD in computer science from the University of South Australia in 2001. His research interests include view maintenance in data warehouses, XML integrity constraints and design, XML and relational data, constraints, and query translation, XML data integration and transformation, XML integrity constraints transformation and transition, and data privacy.