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 $k_{a1}(enroll/dept, \{cid\})$ where $enroll/dept$ is called the selector and $cid$ is called the field. We say the key $k_{a1}$ 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 $k_{a1}$ 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 $(v_4: cid: Phys01), (v_7: cid: Phys02), \text{and } (v_{10}: cid: Chem02)$ are distinct. We see that the XML document $T_a$ in Figure 2 is satisfied by the key $k_{a1}$.

If we observe the DTD $D_a$ and the XML key $k_{a1}$, 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 $k_{a1}$ 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 $T_b$.

We see that the key $k_{a1}$ 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.
Now our next question is whether XML key $k_{a_1}$ 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 $k_{a_1}$ can be transformed to $\Phi_{a_1}(\text{enroll/dept}, \{\text{cid}\} \rightarrow \epsilon)$ meaning that cid determines dept. For now, by the satisfaction of $\Phi_{a_1}$, 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 $\Phi_{a_1}$ is satisfied by the transformed document $T_b$. We say that the key $k_{a_1}$ is transited to XFD $\Phi_{a_1}$.

**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 $c_2$: $1 \oplus ? = *$ 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
1. $EN$ contains element names,
2. $G$ is the set of element definitions and $g \in G$ is defined as
   (i) $g = Str$ where $Str$ means $PDCDATA$;
   (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, $g_1 \neq Str \land g_1 \neq e$;
   (v) $g = g_1 \land g_2 = e \land e \in EN$ or $g_2 = [g \times \ldots \times g]$ or $g_2 = [g ; \ldots ; g]$, called a component
   where $c \in \{?, 1, +, *\}$ is the multiplicity of $g_2$; $[\cdot]$ is the component constructor;
   (c) $\beta(e) = [g]^c$ is the function defining the type of $e$ where $e \in EN$ and $g \in G$.
3. $\rho$ 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 $EN = \{enroll, dept, dname, sid, cid\}$, $G = \{Str, [dept]^+, [dname]?[sid+cid+]\}$, $\beta(enroll) = [dept]^+$, $\beta(dept) = [dname+?][cid+]$, $\beta(dname) = Str$, $\beta(sid) = Str$ 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 $T = (v : e (T_1, T_2, \ldots, T_f))$ 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, $T_1, T_2, \ldots, T_f$ 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_2} = (v_2 : dept(T_{v_5}, T_{v_6}, T_{v_7}, T_{v_8})), 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 $T_{v_9}, T_{v_{10}}, T_{v_{12}}$ 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 $H^9$, which is a sequence of trees conforming to the structure $g$. Thus $H^9_{g_2} = T_{v_4}, T_{v_5}, T_{v_6}, H^9_{g_2} = T_{v_7}, T_{v_8}$ for node $v_1$ and $H^9_{g_2} = T_{v_{10}}, T_{v_{11}}, T_{v_{12}}$ for node $v_2$.

**Definition 3.3 (Hedge)** A hedge $H$ is a sequence of consecutive primary sub trees $T_1, T_2, \ldots, T_n$ of the same node that conforms to the definition of a specific structure $g$, denoted by $H \in g$ or $H^9$:
1. If $g = e$ then $H = T = (v : e : txt)$;
2. If $g = e \land \beta(e) = g_1$ then $H = T = (v : e : H')$ and $H' \in g_1$;
3. If $g = e$ then $H = T = \phi$;
4. If $g = g_1 \times g_2$, $H = H_1 \times H_2$ and $H_1 \in g_1$ and $H_2 \in g_2$;
5. If $g = g_1 | g_2$, $H = H_0$ and $H_0 \in g_1$ or $H_0 \in g_2$;
6. If $g = g_1 \land g_1 = e$, $H = (eH_1) \cdots (eH_f)$ and $\forall i = 1, \ldots, f (H_i \in \beta(e))$ and $f$ satisfies $c$;
(7) if \( g = g^1 \land g_1 = [g] \), \( H = H_1 \cdots H_f \) and \( \forall i = 1, \ldots, f(H_i \in g) \) and \( f \) satisfies c. \( \square \)

Because \( g_s \) are different substructures of an element definition, then \( H^g_s \) are different groups of child nodes. Because of the multiplicity, when there are multiple \( H^g_s \), we use \( H^g_j \) to denote one of them and \( H^g_s \) 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 \( T \equiv D \) if \( T = (\rho \ H^D(\rho)) \). \( \square \)

**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 \( T_b = (v_2 : e : T'_1 \cdots T'_m) \) and \( m = n \) and for \( i = 1, \ldots, m(T_i = v_i T'_i) \).

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 \( i = 1, \ldots, m(T_i = v_i T'_i) \). \( \square \)

\( T_x \equiv T_y \) 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 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 \( H^g_s \) in \( H \). \( \square \)

**Example 3.3** Let \( \beta(\text{dept}) = [\text{dname}, \text{cid}, \text{sid}]^+ \) in \( D_a \). Thus, the minimal structure of dname and sid is \( g_1 = [\text{dname}, \text{cid}, \text{sid}]^+ \). Thus the minimal hedge conforming to \( g_1 \) is \( H^g_1 = T_{v_1} T_{v_2} T_{v_3} T_{v_4} T_{v_5} T_{v_6} T_{v_7} T_{v_8} \) for node \( v_1 \) and \( H^g_2 = T_{v_9} T_{v_{10}} T_{v_{11}} T_{v_{12}} \) for node \( v_2 \) in \( T_a \).

But the minimal structure of cid and sid is \( g_2 = [\text{cid}, \text{sid}]^+ \). So the the minimal hedges conforming to \( g_2 \) are \( H^g_1 = T_{v_2} T_{v_3} T_{v_4}, H^g_2 = T_{v_7} T_{v_8} \) for node \( v_1 \) and \( H^g_3 = T_{v_{10}} T_{v_{11}} T_{v_{12}} \) for node \( v_2 \) in \( T_a \).

**Definition 3.7 (Paths)** Given a D = (EN, G, β, ρ), a simple path \( \varphi \) on D is a sequence \( e_1 / \cdots / e_m \), where \( \forall e_i \in EN \) and \( \forall e_w \in \{e_2, \ldots, e_m\} \) (\( e_w \) is a symbol in the alphabet of \( \beta(e_{w-1}) \)). A simple path \( \varphi \) is a complete path if \( e_1 = \rho \). A path \( \varphi \) is empty if \( m = 0 \), denoted by \( \varphi = \epsilon \). We use function last(\( \varphi \)) to return \( e_m \), beg(\( \varphi \)) = \( e_1 \), par(\( e_w \)) = \( e_{w-1} \), the parent of \( e_w \). We use len(\( \varphi \)) to return \( m \). We define intersected path \( \varphi_1 \cap \varphi_2 = e_1 / \cdots / e_i \) where \( j \in [1, \ldots, i] \) (\( e_j = e_j' \)) and \( e_{i+1} \neq e_{i+1}' \). Paths satisfying this definition are said valid on \( D \). \( \square \)

**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 \( D = (EN, G, \beta, \rho) \), an XML key on \( D \) is defined as \( \mathbb{K}(Q, \{P_1, \ldots, P_l\}) \), where \( l \geq 0 \), \( Q \) is a complete path called the selector, and \( \{P_1, \ldots, P_l, \ldots, P_l\} \) (often denoted by \( P \)) is a set of fields where each \( P_i \) is defined as:
We define \( \varphi_{ij} \) where \( \varphi_{ij} \) is a pair combination of the fields are where 
\( \varphi_{ij} = \text{seq} \)
\( \text{seq} = e | e/\text{seq} \) where \( e \in EN; \)
\( Q \langle \varphi_{ij} \rangle \)
\( Q \langle \varphi_{ij} \rangle \) is a complete path. \( \square \)

A path \( \varphi \) is in \( P \) if \( \exists P_i \in P(\varphi \in P_i) \). \( \varphi \in k \) if \( \varphi = Q \) or \( \varphi \in P \). We use \( \varphi_i \) 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 \( k \subseteq D \). A key is not valid if some conditions in the definition 3.8 is not satisfied.

Example 3.5 Let \( k_{a1}(\text{enroll/dept}, \{\text{cid}\}) \) be a key notation on \( D_a \) in Figure 1. The selector is \( Q = \text{enroll/dept} \) which is a complete path field and is \( P_1 = \varphi_{11} = \text{cid} \) is a simple path, \( \beta(\text{cid}) = \text{Str} \). We see that \( Q/\varphi_{11} = \text{enroll/dept/cid} \) 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 \( e_1 \cdots e_n \), we use \( (v_1 : e_1) \cdots (v_{n-1} : e_{m-1}) \). \( T_{e_m} \) 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 \( \varphi = e_1 \cdots e_n \), \( T^\varphi = (v_1 : e_1) \cdots (v_{n-1} : e_{m-1}) \). \( T^\varphi \) is the set of all \( T^\varphi \) and \( \langle T^\varphi \rangle = \{ T_1^\varphi, \cdots, T_p^\varphi \} \). \( |\langle T^\varphi \rangle| \) returns the number of \( T^\varphi \) in \( T^\varphi \). Because \( P_i = \varphi_{11} \cdots \varphi_{2m} \), we use \( \langle T^P_i \rangle \) to mean all \( T^\varphi \)'s and \( T^P_i = T^\varphi \) to mean one of \( T^\varphi \)'s. We use \( T^\varphi \) to mean that \( T^\varphi \) 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 \( T^\text{dept} \) to mean the tree \( T_{v_1} \) or \( T_{v_2} \) in \( T_a \) of Figure 2. Consider a path \( Q = \text{enroll/dept} \). Then \( T^Q \) = dept. We use \( T^Q \) to mean the tree \( T_{v_1} \) or \( T_{v_2} \). \( T^Q \) = \( \{ T_{v_1}, T_{v_2} \} \), and \( |\langle T^Q \rangle| = 2 \). Now let a path be \( \varphi = \text{cid} \). So \( \langle T^\varphi \rangle = \{ T_{v_1}, T_{v_2} \} \in T_{v_1} \) and \( \langle T^\varphi \rangle = \{ T_{v_{10}} \} \in T_{v_2} \) in \( T_a \).

We now introduce the novel concept P-tuples using an example 3.7.

Example 3.7 Consider an XML key on \( D \) in Figure 5 as \( \varphi(db/\text{univ}, \{\text{dept}, \text{staff} / f\text{name}, \text{staff} / l\text{name}\}) \) where \( Q = \text{db/unic} \), \( P_1 = \text{dept} \), \( P_2 = \text{staff} / f\text{name} \), \( P_3 = \text{staff} / l\text{name} \). All the pair combinations of the fields are \( (P_1, P_2) \), \( (P_1, P_3) \), and \( (P_2, P_3) \). Then the tuple \( (T_{v_5} T_{v_4} T_{v_9}) \) is a P-tuple because, with regard to \( (P_1, P_2) \) and \( (P_1, P_3) \), \( T_{v_9} \) and \( T_{v_4} \) (the parent of both \( T_{v_9} \) and \( T_{v_4} \)) are from the same minimal hedge of \( [\text{dept} \times \text{staff} f^*] \) under \( T_{v_8} \); with regard to \( (P_2, P_3) \), \( T_{v_8} \) and \( T_{v_9} \) are from the same minimal hedge of \( [\text{fname} \times \text{lname}] \) under \( T_{v_8} \). In the same way of reasoning, \( (T_{v_5} T_{v_{10}} T_{v_{11}}) \) is another P-tuple under \( T_{v_9} \) and \( (T_{v_8} T_{v_{12}} T_{v_{13}}) \) is a P-tuple under \( T_{v_{12}} \). On the contrary, the tuple \( (T_{v_3} T_{v_4} T_{v_{11}}) \) is not a P-tuple because \( T_{v_8} \) and \( T_{v_{11}} \) are not in the same minimal hedge of \( [\text{fname} \times \text{lname}] \) with regard to \( (P_2, P_3) \). Another non-P-tuple under \( T_{v_9} \) is \( (T_{v_3} T_{v_{12}} T_{v_{13}}) \). 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 \( k(Q, \{P_1, \ldots, P_i\}) \) 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 \( (T^{P_1} \cdots T^{P_i}) \) as we define next.
Let \( \varphi_i = e_1/\cdots/e_k/e_{k+1}/\cdots/e_m \) where \( \varphi_i \in P_i \), and \( \varphi_j = e'_1/\cdots/e'_k/e'_{k+1}/\cdots/e'_n \) where \( \varphi_j \in P_j \), for any \( P_i \) and \( P_j \). Let \((v_1 : e_1),\ldots,(v_k : e_k),(v_{k+1} : e_{k+1}),\ldots,T^{P_i}\) and \((v'_1 : e'_1),\ldots,(v'_k : e'_k),(v'_{k+1} : e'_{k+1}),\ldots,T^{P_j}\) be the prefixed formats of \( T^{P_i} \) and \( T^{P_j} \) where \((v_m : e_m) = \text{root}(T^{P_i}) \) and \((v'_m : e'_m) = \text{root}(T^{P_j}) \). 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 \( \beta(\text{last}(Q)) \).

(b) If \( e_1 = e'_1, \ldots, e_k = e'_k, e_{k+1} \neq e'_{k+1} \), 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 \((T^{P_1}\cdots T^{P_k})\) is complete if \( \forall T^{P_i} \in (T^{P_1}\cdots T^{P_k})(\exists \phi) \). We use \( \langle T^{P} \rangle \) to denote all possible \( P \)-tuples under a \( T \) tree and \( |\langle T^{P} \rangle| \) means the number of such \( P \)-tuples. Two \( P \)-tuples \( F_1 = (T^{P_1}\cdots T^{P_k}) \) and \( F_2 = (T^{P_1}\cdots T^{P_k}) \) are value equivalent, denoted by \( F_1 =_v F_2 \) if \( l = k \) and for each \( i = 1, \ldots, k \) \((T_1^{P_i} =_v T_2^{P_i})\).

**Definition 3.10 (Key Satisfaction)** An XML tree \( T \) satisfies a key \( k(Q, \{P_1, \ldots, P_l\}) \), denoted by \( T \prec k \), if the followings are hold:

(i) If \( \{P_1, \ldots, P_l\} = \varnothing \), then \( T \) satisfies \( k \) iff there exists one and only one \( T^{P_i} \) in \( T \);

(ii) else,

(a) \( \forall T^{P_i} \in \langle T^{P_i} \rangle \) (exists at least one \( P \)-tuple in \( T^{P_i} \));

(b) \( \forall T^{P_i} \in \langle T^{P_i} \rangle \) (every \( P \)-tuple in \( T^{P_i} \) is complete);

(c) \( \forall T^{P_i} \in \langle T^{P_i} \rangle \) (every \( P \)-tuple in \( T^{P_i} \) is value distinct);

(d) \( \forall T^{P_i} \in \langle T^{P_i} \rangle \) (exists two \( P \)-tuples \((T^{P_1}\cdots T^{P_k}) \in T_1^{P_i} \land (T^{P_1}\cdots T^{P_k}) \in T_2^{P_i} \land (T_1^{P_1}\cdots T_1^{P_k}) =_v (T_2^{P_1}\cdots T_1^{P_k}) \)

\( \Rightarrow T_1^{P_i} \equiv T_2^{P_i} \)). This requires that \( P \)-tuples under different selector nodes must be distinct. □

**Example 3.8** \( k_{a_1}(\text{enroll/dept}, \{\text{cid}\}) \) be a key notation on \( D_a \) in Figure 1. We want to check whether the key \( k_{a_1} \) is satisfied by the document \( T_a \). We see the scope is dept. So for the node \( v_1 \), we get the \( P \)-tuples \( F_1 = (T_{v_4}\cdot T_{v_5}) \) 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 \( k_{a_1} \).

Let \( k_{a_2}(\text{enroll/\{dept/cid, dept/sid\}}) \) be a key on \( D_a \) in Figure 1. We want to check whether \( k_{a_2} \) is satisfied by the XML document \( T_a \) in Figure 2. In \( T_a \), we have only one node \( v_3 \) for the selector \( Q = \text{enroll} \). For the node \( v_r \), we have the \( P \)-tuples \( F_1 = (T_{v_4}\cdot T_{v_5}), F_2 = (T_{v_4}\cdot T_{v_6}), F_3 = (T_{v_7}\cdot T_{v_8}), F_4 = (T_{v_{10}}\cdot T_{v_{11}}) \) and \( F_5 = (T_{v_{10}}\cdot T_{v_{12}}) \). As \( F_3, F_4, F_5 \) are all value different, so \( T_a \prec k_{a_2} \).
Now consider another key $\mathbb{a}_3(\text{enroll}, \{\text{dept/sid}\})$. So, for $v$, 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 $T_n \not\in \mathbb{a}_3$.

**Theorem 3.1** Let $k(Q, P)$ be a key and $P \neq \Phi$. $T\sim k(Q, P)$, 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 \{\$v_1, \ldots, v_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. □

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

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

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

We denote $F[P] = (T^{v_1} \cdots T^{v_a})$ as $P$-tuple and $F[Q] = (T^{v_{a+1}})$ as $Q$-tuple. A $P$-tuple $F[P]$ is complete if $\forall T^{v_i} \in (T^{v_1} \cdots T^{v_a})(T^{v_i}$ is complete)). Similarly, a $Q$-tuple is complete if $T^{v_{a+1}}$ 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 $F_1[P] = (T_1^{v_1} \cdots T_l^{v_a})$ and $F_2[P] = (T_2^{v_1} \cdots T_l^{v_a})$ are value equivalent, denoted by $F_1[P] = v F_2[P]$ if $l = k$ and for each $i = 1, \ldots, k (T_1^{v_i} = v T_2^{v_i})$. Similarly, we denote $F_1[Q] = v, F_2[Q]$ to mean $F_1[Q]$ and $F_2[Q]$ are value equivalent where $F_1[Q] = (T_1^{v_{a+1}})$ and $F_2[Q] = (T_2^{v_{a+1}})$.

**Example 3.10** Consider the XFD $\Phi_{a_2}(\text{enroll/dept}, \{\text{cid}\} \rightarrow \text{dname})$ on the DTD $D$ on Figure 1. We take the paths $\varphi_1 = \text{cid}$ and $\varphi_2 = \text{dname}$ to generate the tuples. So the minimal structure for these paths is $[\text{dname}\times[\text{cid}\times\text{sid}]^+]$ because the paths have intersected path $\varphi_1 \cap \varphi_2 = \epsilon$ and then dname $\not\equiv \text{cid}$. The minimal hedges for the structure $[\text{dname}\times[\text{cid}\times\text{sid}]^+]$ are $H_1 = T_{v_3} \cdots T_{v_4} T_{v_6} T_{v_7} T_{v_8}$ in the $T_{v_1}$ and $H'_1 = T_{v_9} T_{v_{10}} T_{v_1} \cdots T_{v_{12}}$ 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 $\varphi_1 = \text{cid}$. So in $T_{v_1}$, $P$-tuples are $F_1[P] = (T_{v_4}) = ((v_4 : \text{cid} : \text{Phys}01))$, $F_2[P] = (T_{v_7}) = ((v_7 : \text{cid} : \text{Phys}02))$ for $H_1$. For $Q$-tuple, meaning that tuples for the paths $Q$, we take the path $\varphi_2 = \text{dname}$. So in $T_{v_3}$, the $Q$-tuples are $F_1[Q] = (T_{v_3}) = ((v_3 : \text{dname} : \text{Physics}))$ for $H_1$.

In $T_{v_2}$, the $P$-tuples are $F_1[P] = (T_{v_{10}}) = ((v_{10} : \text{cid} : \text{Chem02}))$ for $H'_1$. The $Q$-tuples are $F_1[Q] = (T_{v_{10}}) = ((v_{10} : \text{dname} : \text{Chemistry}))$ 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.
Definition 4.1 (UnNest) \( \tau = T \) forming document \( g \) where \( \forall \) be a path in \( k \). Given \( g \) \( T \) for the transformed document \( F \) 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 \( F_1 [Q] \) and \( F_2 [Q] \) are complete in \( T_{v_2} \), so \( T_a \) satisfies \( \Phi_{a_2} \).

Now consider the XFD \( \Phi_{a_1} \) (enroll/dept, \( \{cid\} \rightarrow dname \) on the DTD \( D_a \). We want to check whether the document \( T_a \) satisfies the XFD \( \Phi_{a_2} \). 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 \( F_1 [Q] \) and \( F_2 [Q] \) are complete in \( T_{v_2} \), so \( T_a \) satisfies \( \Phi_{a_2} \).

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 \( \Phi_{a_2} \). 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 \( F_1 [Q] \) and \( F_2 [Q] \) are complete in \( T_{v_2} \), so \( T_a \) satisfies \( \Phi_{a_2} \).

Now consider the XFD \( \Phi_{a_1} \) (enroll/dept, \( \{cid\} \rightarrow e \). Here, the scope is dept. Then \( T_a \neq \Phi_{a_1} \) 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 \( \tau \) 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 (\( ^{\top} \)) to mean the transformed result. Formally, \( \tau (D,T) \rightarrow (D,T) \) where \( \tau = UnNest, T \in D \) and \( T \in D \).

Definition 4.1 (UnNest) The unnest operation on \( g_2 \) in \( [g_1 \times g_2^{\top}]^{\top} \) is defined as, if \( g = [g_1 \times g_2]^{\top} \) and \( c_2 = +[^{\top}] \), then \( unnest(g_2) \rightarrow [g_1 \times g_2^{\top}]^{\top} = [g_1 \times g_2]^{\top} \). Also, \( unnest(H) \rightarrow H \) where \( H = H^{\top} = H_1^{\top}H_2^{\top} \ldots H^{\top} \). The transformed document is \( \bar{T} = H_1^{\top}H_2^{\top} \ldots H^{\top} \).

Example 4.1 Consider the DTD \( D_a \) in Figure 1. In \( D_a \), we consider the structure \( g = [cid \cdot sid]^{\top} \) where \( g_1 = cid, g_2 = sid, c_2 = + \) and \( c = + \). So, the edges conforming to \( g \) are \( H^{\top} = H_1^{\top}H_2^{\top} \).

Then \( UnNest(sid) \rightarrow \bar{g} = [cid \cdot sid]^{\top} \). The transformed DTD is now \( D_b \) in Figure 3. Thus the transformed edges are \( H^{\top} = H_1^{\top}H_2^{\top} \).

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 \( \varphi \) of a key. We now define the notation.

To describe the relationship between a type structure \( g \) and a path \( \varphi \) on a DTD, we define \( g \circ \varphi \circ \varphi = e \), reading g crossing \( \varphi \) at \( e \), to mean that element \( e \) is in the type structure \( g \) and is also a label on path \( \varphi \), that is \( g = [\ldots \cdot e \cdot \ldots] \land \varphi = e_1 / \ldots / e / \ldots / e_m \). \( e \) is called the cross point of \( g \) and \( \varphi \). Given \( D, \tau \), and \( k(Q, \{P_1, \ldots, P_i\}) \), if a path \( \varphi \) in \( k \) is not crossed by the transformed structure \( g \in D \), then \( \bar{\varphi} = \varphi \). 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 \( \varphi = e_1 / \ldots / e_{k-1} / e_k / e_{k+1} / \ldots / e_m \) be a path in \( k \). The operator \( unnest \) doesn’t change a key because they manipulate the multiplicities of a type structure but doesn’t change paths. In other words, \( \tau (k) = k \) meaning \( \forall \varphi \in k, \tau (\varphi) = \varphi \).

XML Key Preservation. Now we investigate whether the key is preserved after the transformation.
Definition 4.2 (Key Preservation) Given the transformation on $D, T, k$ as $\tau(D, T, k) \rightarrow (\bar{D}, \bar{T}, \bar{k})$ ∧ $\bar{k} \sqsubseteq \bar{D}$, if $T \prec k$ and $\bar{T} \prec \bar{k}$, we say that $k$ is preserved by the transformation $\tau$. □

We give here an example that shows the key preservation.

Example 4.2 Consider the XML key $k_{a2}(enroll/dept, \{cid, sid\})$ on $D_a$ in Figure 1. We see $T_a \prec k_{a2}$. After UnNest, the key is not transformed and so is valid on $D_b$. We see that the key $k_{a2}$ is satisfied by the transformed document $T_b$, that is $T_b \prec k_{a2}$. Thus $k_{a2}$ is preserved.

Now we recall the motivating example in the introduction where we considered the key $k_{a1}(enroll/dept, \{cid\})$. 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 $\text{unnest}(g_2) \rightarrow [g_1 \times g_2^{Q \cap} \oplus e^{B \cap}]$ 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 $\tau(D, T, k) \rightarrow (\bar{D}, \bar{T}, \Phi)$ where $T \sqsubseteq D, k \sqsubseteq D, T \prec k, \bar{T} \sqsubseteq \bar{D}, \Phi \sqsubseteq \bar{D}$ and $\bar{T} \prec \Phi$. □

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 $k$ valid on source DTD $D$, and $T$ satisfies $k$. Output: XML target DTD $\bar{D}$ that is conformed by the XML target document $\bar{T}$, XFD $\Phi$ valid on $\bar{D}$, and $\bar{T}$ satisfies $\Phi$.

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

Function 5.1 CheckKeyTransformation($k$, $UnNest$)
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($k$)
1: $\Phi[S] := k[Q]$;
2: \textbf{for all } i \textbf{ such that } 1 \leq i \leq n \textbf{ do}
3: \quad \Phi[P_i] := k[P_i];
4: \textbf{end for}
5: \quad \Phi[Q] := \epsilon;
6: \textbf{return } \Phi(S, \{P\} \rightarrow Q);

\textbf{Example 5.1} We give here an example how an XML key can be transited to an XFD. We consider the key $k_{a1}(enroll/dept, \{cid\})$ given in the introduction. We see that the key is valid on $D_a$ and is satisfied by the document $T_a$. After $\text{UnNest}(\text{sid})$, 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 $g_1 = \text{cid}$ in DTD $D_a$ is crossed by the field $P_1 = \text{cid}$ of the key $k_{a1}$ at the common element $\text{cid}$ (according to the theorem 4.1). We transform the key to XFD $\Phi_{a1}(enroll/dept, \{cid\} \rightarrow \epsilon)$ which is valid on $D_b$ and $T_b$ satisfies $\Phi_{a1}$.

\textbf{Theorem 5.1} If a key is satisfied by the source document but violates condition (a) or (b) of Theorem 4.1, the key is transformed to an XFD on the target DTD.

\section{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.

\section{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 $\text{UnNest}$ 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.

\section{References}


**Biography**

**Md. Sumon Shahriar:** Sumon Shahriar is currently PhD researcher in Data and Web Engineering Lab, School of Computer 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.
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 a 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)