Towards a Definition of Referential Integrity Constraints for XML

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

In relational data model, two important referential integrity constraints are inclusion dependency (ID) and foreign key (FK). In last decade, with the growing use of XML as data representation and exchange format over the web, the issue of integrity constraints in XML has received great importance to the database community. In this paper, we propose XML Inclusion Dependency (XID) and XML foreign key (XFK). When proposing, we show how both XID and XFK can be defined over the Document Type Definition (DTD) and are satisfied by the XML documents. We introduce a novel concept tuple that produces semantically correct values in the XML documents when satisfactions are checked. We also show that XFK is defined with the combination of XID and XML Key.

1. Introduction

Referential integrity plays an important role in data modeling. Specially in relational data model [1, 2], inclusion dependency and foreign key are well studied and are widely used. In recent years, XML [22] has gained an wide acceptance as data representation and storage format over the web. A massive amount of data is being stored or published in XML. The growing data centric use of XML has necessitated to define the integrity constraints for XML [17, 20, 18, 19, 21]. XML inclusion dependency and XML foreign key are the important constraints if we consider to enrich the semantics of XML data. We give motivating examples to show the important issues for XML referential integrity constraints.

Example 1: We give here an example that is very common in relational database design. Consider the DTD $D$ in Figure 1 that depicts the student database of various departments of a university. The DTD $D$ describes student’s information with student id $sid$ and student name $sname$, course id $cid$ and course name $cname$ offered in the department, and enrollment of students in the courses. Note that the DTD $D$ represents many departments and each department is identified by its id $did$. We see that the student id $sid$ in the $enroll$ must be included in the $studs$ and the course $cid$ in the $enroll$ must be included in the $courses$. We denote them as $\Upsilon(depts, \{enroll/sid\} \subseteq \{studs/sid\})$ and $\Upsilon(depts, \{enroll/cid\} \subseteq \{courses/cid\})$. We term $\Upsilon$ as XML Inclusion Dependency (XID). We say the path $depts$ as selector, the paths...
{enroll/sid} and {enroll/cid} as dependents, and the paths {studs/sid} {courses/cid} as referenced. We say that \( \Upsilon \) is valid because the selector starts from the root element (for now, we say the complete path) and concatenation of the selector with dependents or referenced paths are also complete paths. By XIDs \( \Upsilon \), we mean that for departments, the values of sid in enrollment must be in the values of sid in studs and the values of cid in enrollment must be in the values of cid in courses. We see that the XIDs \( \Upsilon \) are satisfied by the XML document \( T \) in Figure 2.

\[
<!ELEMENT depts(did, studs, courses, enroll) > \\
<!ELEMENT studs(sid, sname) > \\
<!ELEMENT courses(cid, cname) > \\
<!ELEMENT enroll(cid, sid) > 
\]

Figure 1. XML DTD \( D \)

Figure 2. XML Tree \( T \)

Now we are interested in XML foreign key (XFK). In relational database design, foreign key is defined using inclusion dependency (ID) and primary key [1, 2]. We aim to define the XFK using XID and XML Key (actually absolute key). We denote XFK as \( F(\text{depts}, \{\text{enroll/sid}\} \subseteq \{\text{studs/sid}\}) \) where \( \Upsilon(\text{depts}, \{\text{enroll/sid}\} \subseteq \{\text{studs/sid}\}) \) is the XID and \( \kappa(\text{depts}, \{\text{studs/sid}\}) \) is the XML key. Similarly, \( F(\text{depts}, \{\text{enroll/cid}\} \subseteq \{\text{courses/cid}\}) \) where \( \Upsilon(\text{depts}, \{\text{enroll/cid}\} \subseteq \{\text{courses/cid}\}) \) is the XID and \( \kappa(\text{depts}, \{\text{courses/cid}\}) \) is the XML Key.

We already defined the meaning of XIDs. Now we define XML keys. For XML key \( \kappa(\text{depts}, \{\text{studs/sid}\}) \), we say depts is selector and studs/sid is field. Like the validity of XID, we say the XML key \( \kappa \) is valid because the selector starts with the root element depts and the concatenation of the selector and field depts/studs/sid is a complete path. In the same way, the XML key \( \kappa(\text{depts}, \{\text{courses/cid}\}) \) is also valid. We see that the key \( \kappa(\text{depts}, \{\text{studs/sid}\}) \) is satisfied by the document \( T \) in Figure 2 because the tuples produced by the fields under the selector nodes are all value different. In similar way, \( \kappa(\text{depts}, \{\text{courses/cid}\}) \) is also satisfied by the document \( T \). We see that the XFK \( F \) is satisfied the document \( T \) in Figure 2 because the XIDs \( \Upsilon \), and the XML keys \( \kappa \) are satisfied by the document \( T \).

The satisfaction of XFK given above seems straightforward as the XID and the XML key
satisfactions are apparently simple. However, we give an example which shows that the checking of satisfactions of XIDs as well as XML keys are not so trivial and can also be ambiguous.

**Example 2:** Consider the DTD $D$ in the Figure 3 that describes the employees and the faculties who are also employees using XFK. Consider the XFK $F (\text{univ/dept}, \{\{\text{faculty/first}, \text{faculty/last}\} \subseteq \{\text{emp/fname}, \text{emp/lname}\})$ on the DTD $D$. To check the XFK, we need to check whether the XML document $T$ in Figure 4 satisfies both XID $\Upsilon (\text{univ/dept}, \{\{\text{faculty/first}, \text{faculty/last}\} \subseteq \{\text{emp/fname}, \text{emp/lname}\})$ and the XML key $k(\text{univ/dept}, \{\text{emp/fname}, \text{emp/lname}\})$.

We now check how to produce values of pair as $\langle \text{fname}, \text{lname} \rangle$ for faculty. For example, for the node $v_2$ in Figure 4, the correct pair values for $\langle \text{fname}, \text{lname} \rangle$ are $\langle \text{John}, \text{Andy} \rangle$ and $\langle \text{Henry}, \text{Ford} \rangle$. However, $\langle \text{John}, \text{Ford} \rangle$ is not correct value pair as $\text{fname}$ and $\text{lname}$ are taken from two different employees which doesn’t make sense. We term the correct value as tuple. We see that the key $k(\text{univ/dept}, \{\text{emp/fname}, \text{emp/lname}\})$ is satisfied by the document $T$ in Figure 4 because for the nodes $v_1$ and $v_2$, the tuples $\langle \text{Lisa}, \text{Carol} \rangle$, $\langle \text{John}, \text{Andy} \rangle$ and $\langle \text{Henry}, \text{Ford} \rangle$ are value distinct.

We now check whether $\langle \text{fname}, \text{lname} \rangle$ for faculty is included as $\langle \text{first}, \text{last} \rangle$ for emp. We need to produce correct tuples for $\langle \text{first}, \text{last} \rangle$ for emp in the document. For the node $v_1$ in $T$, we find that $\langle \text{Lisa}, \text{Carol} \rangle$ for faculty is included in emp. However, for the node $v_2$, the tuple $\langle \text{John}, \text{Ford} \rangle$ is not included in emp. Thus the XID is not satisfied by the document and hence the XFK is also not satisfied.

**Figure 3. XML DTD $D$**

```xml
<!ELEMENT univ(dept)+ >
<!ELEMENT dept(emp+, faculty+) >
<!ELEMENT emp(fname, lname) >
<!ELEMENT faculty(first, last) >
```

**Figure 4. XML Tree $T$**

**Observation 1:** In checking the satisfaction of XFK(both XID and XML Key), we need to generate tuples that are not ambiguous.

While addressing the problem, we consider the following contributions.

- We define the XML foreign key(XFK) on the XML Document Type Definition (DTD)\[22\]. To define the XFK, we also define XML Inclusion Dependency(XID) and XML key on
We consider DTD because of its simpler design over the XML Schema[23].

- We also define the satisfaction of XIDs and XML Keys by XML documents. A novel concept tuple is used when satisfaction is checked to prevent the production of tuples which are semantically ambiguous.
- We last discuss how our proposed definitions for XID and XFK are useful for data modeling in XML.

**Organization:** We define some basic definitions and notation in Section 2. In Section 3, we define XML keys and their satisfactions. We introduce the novel concept tuple here. We then define XML inclusion dependency and XML foreign keys and their satisfactions in Section 4. We give discussions about XID and XFK in Section 5. Some applications of XML referential integrity constraints are discussed in Section 6. We conclude with some remarks and future works in Section 7.

### 2. Basic Definitions and Notation

We give here basic definitions and notation used throughout the paper.

**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

- $g = Str$ where $Str$ means $\#PCDATA$;
- $g = e$ where $e \in EN$;
- $g = \epsilon$ means $EMPT Y$ type;
- $g = g_1 \times g_2$ or $g_1 | g_2$ is called conjunctive or disjunctive sequence respectively where $g_1 = g$ is recursively defined, $g_1 \neq Str \land g_1 \neq \epsilon$;
- $g = g_1[\ldots g_n]$ or $g_2 = [g] \cdot [g]$, called a component where $c \in \{?, 1, +, *\}$ is the multiplicity of $g_2$; $[,]$ is the component constructor;
- $\beta(e) = |g|^c$ is the function defining the type of $e$ where $e \in EN$ and $g \in G$.
(c) $\rho$ is the root of the DTD and that can be only be used as $\beta(\rho)$. □

**Example 2.1** The DTD in Figure 1 can be represented as $D = (EN, G, \beta, \rho)$ where $EN = \{depts, did, stu ds, sid, sname, courses, cid, c name, enroll\}$, $G = \{Str, [\ldots did \times stu ds \times courses \times enroll]^{+}, [\ldots sid \times sname]^{+}, [\ldots cid \times c name]^{+}, [\ldots sid \times sid]^{+}\}$, $\beta(depts) = [\ldots did \times stu ds \times courses \times enroll]^{+}$, $\beta(studs) = [\ldots sid \times sname]^{+}$, $\beta(courses) = [\ldots cid \times c name]^{+}$, $\beta(enroll) = [\ldots sid]^{+}$ and $\beta(cid) = \beta(sname) = \beta(sid) = Str$.

**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 $T = (v : e (T_1, T_2, \ldots T_f))$ 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. $T_1, \ldots, T_f$ are subtrees. □

**Example 2.2** The XML tree $T$ in Figure 2 can be represented as $T_{v_r} = (v_r : dept(T_{v_1}, T_{v_2}, T_{v_3}, T_{v_4}))$, $T_{v_1} = (v_1 : did : CIS), T_{v_2} = (v_2 : stu ds(T_{v_9} T_{v_{10}}, T_{v_{12}})), T_{v_3} = (v_3 : courses(T_{v_{13}}, T_{v_{14}}, T_{v_{15}} T_{v_{16}})), T_{v_4} = (v_4 : enroll(T_{v_{17}}, T_{v_{18}}, T_{v_{19}}, T_{v_{20}})), T_{v_5} = (v_5 : did : MATH), T_{v_6} = (v_6 : stud(T_{v_{22}}, T_{v_{23}})), T_{v_7} = (v_7 : courses(T_{v_{22}}, T_{v_{24}})), T_{v_8} = (v_8 : enroll(T_{v_{25}} T_{v_{26}})), T_{v_9} = (v_9 : sid : 001), T_{v_{10}} = (v_{10} : sname : John), T_{v_{11}} = (v_{11} : sid : 003), T_{v_{12}} = (v_{12} : sname : Carol). We define other trees in the same way.
Now we give an example to show the important concept hedge. Consider $g_1 = [\text{name}, \text{name}]$ for the DTD $D$ in Figure 3. The trees $T_{v9} T_{v9}$ form a sequence conforming to $g_1$ for node $v_1$ and the trees $T_{v11} T_{v13} T_{v14} T_{v15}$ form a sequence for node $v_2$. 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 $H_1^{g_1} = T_{v9} T_{v9}$ for node $v_1$, $H_2^{g_1} = T_{v12} T_{v13}$ and $H_3^{g_1} = T_{v14} T_{v15}$ for node $v_2$.

**Definition 2.3 (Hedge)** A hedge $H$ is a sequence of consecutive primary sub trees $T_1 T_2 \cdots T_n$ of the same node that conforms to the definition of a specific structure $g$, denoted by $H \in g$ or $H^g$:

1. if $g = \epsilon \land \beta(e) = \text{Str}$, then $H = (v : e : \text{txt})$;
2. if $g = \epsilon \land \beta(e) = g_1$, then $H = (v : e : H')$ and $H' \in g_1$;
3. if $g = \epsilon$, then $H = \emptyset$;
4. if $g = g_1 \land g_2$, then $H = H_1 H_2$ and $H_1 \in g_1$ and $H_2 \in g_2$;
5. if $g = g_1 g_2$, then $H = H_0$ and $H_0 \in g_1$ or $H_0 \in g_2$;
6. if $g = g_1 \land g_2 = e$, then $H = (e H_1) \cdots (e H_f)$ and $\forall i = 1, \cdots, f \ (H_i \in \beta(e))$ and $f$ satisfies $c$;
7. if $g = g_1 \land g_1 = [g], H = H_1 \cdots H_f$ and $\forall i = 1, \cdots, f (H_i \in g)$ and $f$ satisfies $c$.

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 $H^g$ 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 $T \in D$ if $T = (\rho H^g(\rho))$.

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

1. $T_a = (v_1 : e : \text{txt1})$ and $T_b = (v_2 : e : \text{txt1})$, or
2. $T_a = (v_1 : e : T_1 \cdots T_n)$ 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. $H_x = T_1 \cdots T_n$ 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 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 $H^g$s in $H$.

**Example 2.3** Let $\beta(\text{studs}) = [\text{sid}, \text{name}]^+$ for $D$ in Figure 1. The minimal structure of $\text{sid}$ is $g_1 = [\text{sid}, \text{name}]$. The minimal hedge conforming to $g_1$ is $H_1^{g_1} = T_{v9} T_{v10} T_{v11} T_{v12}$ for node $v_3$ and $H_2^{g_1} = T_{v21} T_{v22}$ for node $v_6$ in $T$ in Figure 2.

But the minimal structure of $\text{did}$ and $\text{sid}$ is $g_2 = [\text{did}, \text{studs}, \text{courses}, \text{enroll}]^+$. This is because the $\text{did}$ is in $g_2$ and $\text{sid}$ is in $\beta(\text{stud})$ which is also in $g_2$. So the minimal hedges conforming to $g_2$ are $H_1^{g_2} = T_{v1} T_{v2} T_{v3} T_{v4}$ and $H_2^{g_2} = T_{v5} T_{v6} T_{v7} T_{v8}$ in $T$. 

73
Definition 2.7 (Paths) Given a D = (EN, G, β, ρ), a simple path ϕ 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 \( β(e_{w-1}) \)). A simple path \( ϕ \) is a complete path if \( e_1 = ρ \). A path \( ϕ \) is empty if \( m = 0 \), denoted by \( ϕ = ε \). We use function \( \text{last}(ϕ) \) to return \( e_m \), \( \text{beg}(ϕ) = e_1 \), \( \text{par}(e_w) = e_{w-1} \), the parent of \( e_w \). We use \( \text{len}(ϕ) \) to return \( m \). We define intersected path \( ϕ_1 \cap ϕ_2 = e_1/\cdots/e_i \) where \( j \in [1, \cdots, i] \) (\( e_j = e'_j \)) and \( e_{i+1} ≠ e'_{i+1} \). Paths satisfying this definition are said valid on \( D \).

Example 2.4 In Figure 1, \( \text{studs/sid} \) is a simple path and \( \text{depts/studs/sid} \) is a complete path on the DTD \( D \). The function \( \text{beg}(\text{depts/studs/sid}) \) returns \( \text{depts} \), \( \text{par}(\text{sid}) \) returns \( \text{studs} \), and \( \text{len}(\text{depts/studs/sid}) = 3 \).

3. XML Keys

We define XML keys in this section.

Definition 3.1 (XML Key) Given a DTD \( D = (EN, β, ρ) \), an XML key on \( D \) is defined as \( k(Q, \{P_1, \cdots, P_l\}) \), where \( l ≥ 0 \), \( Q \) is a complete path on \( D \) called the selector, and \( \{P_1, \cdots, P_l, \cdots, P_r\} \) (often denoted by \( P \)) is a set of fields where each \( P_i \) is defined as:

(a) \( P_i = ϕ_1 \cup \cdots \cup ϕ_m \), where "\( \cup \)" means disjunction and \( ϕ_{i,j} \) (\( j \in [1, \cdots, n_i] \)) is a simple path on \( D \), \( β(\text{last}(ϕ_{i,j})) = \text{Str} \), and has the following syntax:

\[
\begin{align*}
\text{seq} = e \mid e/\text{seq} \quad \text{where} \quad e \in EN; \\
\text{ϕ}_{i,j} = \text{seq}
\end{align*}
\]

(b) \( Q/ϕ_{i,j} \) is a complete path.

A path \( ϕ \) in \( P \) if \( \exists P_i \in P(ϕ ∈ P_i) \). \( ϕ \in k \) if \( ϕ = Q \) or \( ϕ ∈ P \). We use \( ϕ_i \) 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 \( k □ D \). A key is not valid if a path \( ϕ \) in \( k \) is not valid on \( D \) or \( Q \) is empty, or the type of \( β(\text{last}(ϕ)) ≠ \text{Str} \).

Example 3.1 Consider the XML key \( k(\text{depts}, \{\text{studs/sid}\}) \) on the DTD in Figure 1 where the scope is \( \text{depts} \), the field is \( \text{studs/sid} \), \( \text{depts/studs/sid} \) is a complete path, and the type of \( \text{sid} \) is \( \text{Str} \). Similarly, \( k(\text{depts}, \{\text{courses/cid}\}) \) is another XML key. Consider another XML key \( k(\text{depts}, \{\text{enroll/sid}, \text{enroll/cid}\}) \) where there are two fields.

We define some additional notation for the rest of the paper.

- \( T^ε \) means a tree rooted at a node labeled by the element name \( e \). For example, if the element is \( \text{did} \), then we use \( T^{\text{did}} \) to mean the tree rooted at \( \text{did} \) and the trees are \( T_{v_1}, T_{v_5} \) in the document \( T \) in Figure 2.

- 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 \( T^e_m \). For example, consider the path \( \text{depts/studs/sid} \) over the DTD \( D \) in Figure 1. We use \( T^{\text{sid}} \) to mean the prefixed tree.

- Given path \( ϕ = e_1/\cdots/e_m \), \( T^ϕ = (v_1 : e_1) \cdots (v_{m-1} : e_{m-1}).T^e_m \). \( T^ϕ \) is the set of all \( T^ϕ \) and \( (T^ϕ) = \{T^ϕ_1, \cdots, T^ϕ_p\} \). \( |(T^ϕ)| \) returns the number of \( T^ϕ \) in \( (T^ϕ) \). Consider the path \( ϕ = \text{depts/did} \). Then \( (T^ϕ) = (T^{\text{did}}) = \{T_{v_1}, T_{v_5}\} \) and \( |(T^ϕ)| = 2 \) in Figure 2.
Definition 3.2 (P-tuple) Given a key \( k(Q, \{P_1, \ldots, P_l\}) \) and a tree \( T \), let \( T^Q \) be a tree in \( T \). A P-tuple in \( T^Q \) is a tuple of pair-wise close subtrees \( (T^P_1, \ldots, T^P_l) \) as we define next.

Let \( \varphi_i = e_1/\cdots/e_k/e_{k+1}/\cdots/e_m \land \varphi_i \in P_i \), and \( \varphi_j = e'_1/\cdots/e'_k/e'_{k+1}/\cdots/e'_n \land \varphi_j \in P_j \) for any \( P_i \) and \( P_j \). Let \( (v_1 : e_1), (v_k : e_k), (v_{k+1} : e_{k+1}), \ldots, (v_l : e_l) \) be the prefixed formats of \( T^P_1 \) and \( T^P_2 \) where \( (v_m : e_m) = \text{root}(T^P_1) \) and \( (v_n : e_n) = \text{root}(T^P_2) \). Then \( T^P_1 \) and \( T^P_2 \) are pair-wise close if

(a) If \( e_1 \neq e'_1 \), then \( (v_1 : e_1) \) \( (v'_1 : e'_1) \) 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 \( v_k = v'_k, (v_{k+1} : e_{k+1}) \) \( (v'_{k+1} : e'_{k+1}) \) are two nodes in the same minimal hedge of \( e_k \) and \( e'_k \) in \( \beta(e_k) \).

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

Example 3.2 Consider the key \( k(\text{depts}, \{\text{studs/sid}\}) \). Here \( P_1 = \text{studs/sid} \). Thus the P-tuples \( (T^\text{sid}) \) are \( (\text{sid} : 001), (\text{sid} : 003) \) and \( (\text{sid} : 004) \) in the tree \( T_{v_t} \). Note that the scope of the root is the document. In similar way, P-tuples \( (T^{\text{sid}^c}) \) are \( (\text{cid} : \text{CIS02}) \), \( (\text{cid} : \text{CIS03}) \), \( (\text{cis} : \text{MATH02}) \) for key \( k(\text{depts}, \{\text{courses/cid}\}) \) in the document \( T \) in Figure 2.

Now consider the key \( k(\text{depts}, \{\text{enroll/sid, enroll/cid}\}) \) where \( P_1 = \text{enroll/sid} \) and \( P_2 = \text{enroll/cid} \). So the P-tuples \( (T^{\text{sid}^c, \text{cid}^c}) \) are \( (\text{sid} : 001 \text{cid} : \text{CIS03}), (\text{sid} : 004 \text{cid} : \text{CIS02}) \) and \( (\text{sid} : 004 \text{cid} : \text{MATH02}) \) in the tree \( T \).

Definition 3.3 (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 \( \{P_1, \ldots, P_l\} = \phi \) in \( k \), then \( T \) satisfies \( k \) 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) \( \forall T^Q, T^Q_2 \in \langle T^Q \rangle \) (exists two P-tuples \( (T^P_1, \ldots, T^P_l) \in T^Q_1 \land (T^P_1, \ldots, T^P_l) \in T^Q_2 \land T^Q_1 \neq T^Q_2 \Rightarrow (T^P_1, \ldots, T^P_l) \Rightarrow (T^P_1, \ldots, T^P_l) \Rightarrow T^Q_1 \neq T^Q_2 \). This requires that P-tuples in different \( T^Q \) must be value distinct.

Example 3.3 For checking satisfaction of keys, we consider the P-tuples generated in the Example 3.2. The XML key \( k(\text{depts}, \{\text{studs/sid}\}) \) is satisfied by the document \( T \) because in \( T_{v_t} \), the P-tuples \( (\text{sid} : 001), (\text{sid} : 003) \) and \( (\text{sid} : 004) \) are value distinct. In the same
way, $\kappa(\text{depts}, \{\text{courses}/\text{cid}\})$ is satisfied by the document $T$ as the P-tuples (cid : CIS02), (cid : MATH02) are value distinct.

The key $\kappa(\text{depts}, \{\text{enroll}/\text{sid}, \text{enroll}/\text{cid}\})$ is also satisfied by the document $T$ as the P-tuples (sid : 001cid : CIS03), (sid : 004cid : CIS02) and (sid : 004cid : MATH02) are all value distinct.

4. XML Inclusion Dependency and XML Foreign Key

We define XML inclusion dependency and XML foreign key here.

**Definition 4.1 (XML Inclusion Dependency)** An XML inclusion dependency over the DTD $D$ can be defined as $\Upsilon(Q, \{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\})$ where $Q$ is a complete path called **selector**, $P_i$ is a simple path called **dependent** and $R_i$ is a simple path called a **referenced** path and both $P_i$ and $R_i$ follow the rules (a) and (b) of definition 3.1.

An XID following the above definition is valid, denoted as $\Upsilon \subseteq D$.

**Example 4.1** Consider the XID $\Upsilon(\text{dept}, \{\text{enroll}/\text{sid} \subseteq \{\text{studs}/\text{sid}\})$ on the DTD $D$ in Figure 1. Here $P_1 = \text{enroll}/\text{sid}$ is simple path, $R_1 = \text{studs}/\text{sid}$ is also simple path, depts/enroll/sid and depts/studs/sid are complete paths.

Consider another XID $\Upsilon(\text{univ}/\text{dept}, \{\text{faculty}/\text{first}, \text{faculty}/\text{last} \subseteq \{\text{emp}/\text{fname}, \text{emp}/\text{name}\})$ on the DTD $D$ in Figure 3. We see that there are two paths in the dependent $P$ and two paths in the referenced $R$. The selector univ/dept is a complete path and the concatenation of the selector with all dependents and referenced paths are also complete paths over the DTD.

**Definition 4.2 (XML Inclusion Dependency Satisfaction)** An XML document $T$ satisfies an XML inclusion dependency $\Upsilon(Q, (P \subseteq R))$, denoted as $T \prec \Upsilon$ if there exists a P-tuple in $T$, then there must be a R-tuple where $(T^{P_1}) = v (T^{R_i})$ and $i \in [1, \cdots, n]$.

**Example 4.2** Consider the XID $\Upsilon(\text{dept}, \{\text{enroll}/\text{sid} \subseteq \{\text{studs}/\text{sid}\})$. We want to check whether $T$ satisfies the $\Upsilon$. We see that the P-tuples ($T^{\text{sid}}$) are $(v_{18} : \text{sid} : 001)$, $(v_{20} : \text{sid} : 004)$, and $(v_{26} : \text{sid} : 004)$. The P-tuples are also in the R-tuples: $(v_9 : \text{sid} : 001)$, $(v_{11} : \text{sid} : 003)$, and $(v_{21} : \text{sid} : 004)$. We use the node identifier $v$ to distinguish the different values.

Now consider the XID $\Upsilon(\text{dept}, \{\text{enroll}/\text{cid} \subseteq \{\text{courses}/\text{sid}\})$. We want to check whether $T$ satisfies the $\Upsilon$. We see that the P-tuples ($T^{\text{cid}}$) are $(v_{17} : \text{cid} : \text{CIS03})$, $(v_{19} : \text{cid} : \text{CIS02})$, and $(v_{25} : \text{cid} : \text{MATH02})$. The P-tuples are also in the R-tuples: $(v_{13} : \text{cid} : \text{CIS02})$, $(v_{15} : \text{cid} : \text{CIS03})$, and $(v_{23} : \text{cid} : \text{MATH02})$.

We then check the satisfaction of $\Upsilon(\text{univ}/\text{dept}, \{\text{faculty}/\text{first}, \text{faculty}/\text{last} \subseteq \{\text{emp}/\text{fname}, \text{emp}/\text{name}\})$ for the document $T$ in Figure 4. We already showed in the motivating Example 2 that this XID is not satisfied by the document because the faculty (John, Ford) is not included in the employees of the university.

**Definition 4.3 (XML Foreign Key)** Given an XID $\Upsilon(Q, \{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\})$ on the DTD, we define $\text{XFK}$ as $\text{XFK}(Q, \{P_1, \cdots, P_n\} \subseteq \{R_1, \cdots, R_n\})$ if there is an XML key as $\kappa(Q, \{R_1, \cdots, R_n\})$.

**Example 4.3** Consider the XID $\Upsilon(\text{dept}, \{\text{enroll}/\text{sid} \subseteq \{\text{studs}/\text{sid}\})$ on the DTD $D$ in Figure 1. As $\kappa(\text{depts}, \{\text{studs}/\text{sid}\})$ is a valid key on $D$, then we say $\text{XFK}(\{\text{depts}, \{\text{enroll}/\text{sid}\} \subseteq \{\text{studs}/\text{sid}\})$.
\{studs/sid\}) is an XFK. Similar way, \(F(\text{depts}, \{\text{enroll/cid} \subseteq \{\text{courses/cid}\})\) is an XFK on the DTD \(D\) in Figure 1.

Also consider the XID \(\Upsilon(\text{univ/dept}, \{\text{faculty/first, faculty/last} \subseteq \{\text{emp/fname, emp/lname}\})\) on the DTD \(D\) in Figure 3. As \(k(\text{univ/dept}, \{\text{emp/fname, emp/lname}\})\) is a valid key in the DTD \(D\) in Figure 3, so we say \(F(\text{univ/dept}, \{\text{faculty/first, faculty/last} \subseteq \{\text{emp/fname, emp/lname}\})\) is a valid XFK on \(D\).

**Definition 4.4 (XML Foreign Key Satisfaction)** An XML document \(T\) satisfies the XFK \(F(Q, \{P_1, \ldots, P_n\} \subseteq \{R_1, \ldots, R_n\})\) denoted as \(T \prec F\) if both XID \(\Upsilon(Q, \{P_1, \ldots, P_n\} \subseteq \{R_1, \ldots, R_n\})\) and XML key \(k(Q, \{R_1, \ldots, R_n\})\) are satisfied by the document \(T\).

**Example 4.4** Consider the \(F((\text{depts}, \{\text{enroll/sid} \subseteq \{\text{studs/sid}\}))\). We see that \(\Upsilon(\text{dept}, \{\text{enroll/sid} \subseteq \{\text{studs/sid}\})\) is satisfied in the Example 4.2 and the key \(k(\text{depts}, \{\text{studs/sid}\})\) is satisfied in the Example 3.3 by the document \(T\) in Figure 2. Thus the document \(T\) satisfied the XFK \(F\) denoted as \(T \prec F\).

In similar way of reasoning, the XFK \(F(\text{depts}, \{\text{enroll/cid} \subseteq \{\text{courses/cid}\})\) is satisfied by the document \(T\) in Figure 2.

However, \(F(\text{univ/dept}, \{\text{faculty/first, faculty/last} \subseteq \{\text{emp/fname, emp/lname}\})\) is not satisfied by the document \(T\) in Figure 4 denoted as \(T \not\prec F\) because the XID \(\Upsilon(\text{univ/dept}, \{\text{faculty/first, faculty/last} \subseteq \{\text{emp/fname, emp/lname}\})\) is not satisfied by the document \(T\) in Figure 4 though the key \(k(\text{univ/dept}, \{\text{emp/fname, emp/lname}\})\) is satisfied by the document \(T\) in Figure 4.

**Theorem 4.1** An XFK \(F(Q, \{P_1, \ldots, P_n\} \subseteq \{R_1, \ldots, R_n\})\) is satisfied by an XML document \(T\) if and only if \(T\) satisfies both XID \(\Upsilon(Q, \{P_1, \ldots, P_n\} \subseteq \{R_1, \ldots, R_n\})\) and XML key \(k(Q, \{R_1, \ldots, R_n\})\).

5. Discussions

We now discuss how our proposal for XML referential integrity constraints is useful.

(i) We use the concept tuple that produces semantically correct values when the satisfactions for both XID and XFK are checked. This property is not achievable from both ID and IDREF of XML DTD[22] and Key and KeyRef of XML Schema[23].

(ii) We define XML foreign key on the DTD as schema which is absent in [10]. Though the DTD has the ID and IDREF definition for denoting key and referential integrity, the drawbacks of ID and IDREF are well recognized as their scope is the entire document and behave as object identifier.

(iii) Our definition for XID and XFK can be used for XML update, deletion, normalization, XML query and view maintenance[14, 15, 16].

6. Applications of XML Referential Integrity

In this section, we show how XIDs and XFKs are useful in XML data applications.

6.1. Data Quality for XML

Integrity constraints is an important metric for ensuring, measuring, and assessing of quality of data[11, 12, 13] in the sense of redundancy and consistency. As XML is being widely used for
different data centric purposes, quality of XML data is necessary. We now discuss some important issues of XML data quality where XML referential integrity can be employed.

Ensure Data Quality: To enforce data quality, use of referential integrity plays an important role. The most useful way is to use normalization techniques not to allow redundant values in the schema. We give two examples to explain how integrity constraints can ensure data quality.

![Example 6.1](image)

Example 6.1 Consider the DTD $D$ in Figure 5 and its conforming document $T$ in Figure 6. Let $k(enrollment, \{sid, cid\})$ be a key on $D$. The document $T$ satisfied the key $k$. But we observe that still there are redundant data in the document $T$ such as cid 'CS100' and its associated cname 'Java' are stored two times if we assume that every course id can have only one name, which is a typical case. If we design the schema according to the DTD $D$ in Figure 7 and its conforming document $T$ in Figure 8 with key $k(db, \{enrollment/sid, enrollment/cid\})$ and foreign key $f(db, (\{enrollment/cid\} \subseteq \{course/cid\}))$, we can reduce redundant data by ensuring quality of XML data.

![Figure 7. XML DTD $D$](image)

An opposite example is that a database designer may design the schema with key $k(db, \{enrollment/sid, enrollment/cid\})$ but without foreign key $f(db, (\{enrollment/cid\} \subseteq \{course/cid\}))$. Then, the schema can easily allow errors. For example, we can have the document $T$ in Figure 9 having a student with a course id 'CS301' having no existence of the course.

Assess Data Quality: Given XML DTD $D$ and document $T$, if we conclude that XID $\Upsilon_1$ and Key $k_1$ should hold on $T$ by analyzing $D$, then checking $T$ against $\Upsilon_1$ and $k_1$ will results in whether $T$ is having expected quality.
Figure 8. XML document \( T \)

```xml
<db>
  <enrollment>
    <sid>001</sid><cid>CS100</cid>
    <sid>002</sid><cid>CS100</cid>
    <sid>001</sid><cid>CS101</cid>
    <sid>002</sid><cid>CS301</cid>
  </enrollment>
  <course>
    <cid>CS100</cid><cname>Java</cname>
    <cid>CS101</cid><cname>XML</cname>
  </course>
</db>
```

Figure 9. XML document \( T' \)

```xml
<db>
  <enrollment>
    <sid>001</sid><cid>CS100</cid>
    <sid>002</sid><cid>CS100</cid>
    <sid>001</sid><cid>CS101</cid>
    <sid>002</sid><cid>CS301</cid>
  </enrollment>
  <course>
    <cid>CS100</cid><cname>Java</cname>
    <cid>CS101</cid><cname>XML</cname>
  </course>
</db>
```
Another side of the problem is the discovery of constraints. Given a document $T$ without a DTD, how do we know the quality of data? In this case, discovering the integrity constraints defined becomes very important.

**Example 6.2** We take the document $T$ in Figure 8 in the Example 6.1. In this case, we assume that the document $T$ is without DTD. After analyzing data in the document $T$, we can come to the conclusion that there are existences of XML key and XIND(also XFK). However, if we analyze the document $T$ in Figure 9, we can’t guarantee of having referential integrity constraints.

### 6.2. Guided Mapping Creation in XML Data Integration

In data integration[3, 4], transformation of data plays an important role. When a source schema with its data is transformed to a target schema with its conforming data, there is also a need to map the source constraints to the target constraints. In mapping constraints, the task of finding equivalence between source constraints and target constraints is an important issue. In XML data transformation for integration purposes[5, 6, 7, 8, 9], this issue needs attention with the help of different integrity constraints.

Given two XML trees $T_1$ and $T_2$, XID $\Upsilon_1$ and Key $k_1$ defined on $D_1$ of $T_1$ and Key $k_2$ defined on $D_2$ of $T_2$, $T_1$ satisfies $\Upsilon_1$ and $k_1$, $T_2$ satisfies $k_2$, if $k_2 = k_1 + \Upsilon_1$, then $T_1$ can be transformed to have the structure of $T_2$ and satisfy $k_2$. Alternatively, $T_2$ can be transformed to have the structure of $T_1$ and satisfy $k_1$ and $\Upsilon_1$. This result is very useful in case of data integration.

**Example 6.3** In this case, we can take the Example 6.1. In data integration, we may need to transform the source DTD $D$ in Figure 5 with its conforming document $T$ in Figure 6 and key $k(enrollment,\{sid,cid\})$ to the target DTD $D$ in Figure 7 with its conforming document $T$ in Figure 8 and key $k(db,\{enrollment/sid, enrollment/cid\})$ and foreign key $\wp(db,\{\{enrollment /cid\} \subseteq \{course/cid\}))$. We see that we can map data from source schema to target preserving target constraints. Then it is necessary to find the equivalence of constraints defined on two schemas. It is worthy to mention that still data quality can be an important issue in data transformation and integration when constraints are considered.

### 7. Conclusions

We proposed the XML referential integrity constraints as XML inclusion dependency and XML foreign key. We used the novel concept tuple for producing correct values for satisfactions. We also discussed how proposal is useful over the standards and proposals. We plan to derive the inference rules for XID and to use referential constraints for XML in the broader task of XML data transformation and integration with constraints.

### References


Biography

Md. Sumon Shahriar: He 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.