Liu, J., Vincent, M., & Liu, C. (2003). Functional dependencies, from relational to XML.

Originally published in *Perspectives of system informatics: revised papers from the 5th International Andrei Ershov Memorial Conference (PSI 2003), Akademgorodok, Novosibirsk, Russia, 09–12 July 2003.* Lecture notes in computer science (Vol. 2890, pp. 531–538). Berlin: Springer.

Available from: [http://dx.doi.org/10.1007/978-3-540-39866-0_51](http://dx.doi.org/10.1007/978-3-540-39866-0_51)
Functional dependencies, from relational to XML

Jixue Liu       Millist Vincent        Chengfei Liu
School of Computer and Information Science
The University of South Australia
Email: {jixue.liu, millist.vincent, chengfei.liu}@unisa.edu.au

Abstract

The flexibility of XML allows the same data to be represented in many different ways. Some representations may be better than others in that they require less storage or have less redundancy. In this paper we present a new definition of functional dependencies in XML (XFDs) and investigate their effect on the design of XML documents. We then define a subclass of XFDs that corresponds to FDs in relations, called primary XFDs, and show that two subtypes of primary XFDs, namely partial and transitive XFDs, cause the same problems in XML document design as the corresponding types of FDs in relations. We then show that the removal of such types of XFDs can lead to a better document design.

Keywords: XML, functional dependency, normalization

1 Introduction

XML has become accepted as the universal standard for data interchange and publication on the internet. Because of its flexible syntax, XML allows the same data to be represented in many different ways. Similar to relational databases, some XML documents may be better designed than other in that they may require less storage or have less redundancy. In relational databases, the basis for deciding between different database designs is the theory of functional dependencies and normalization. However, in spite of the variety of integrity constraints that have been defined for XML [9, 6, 8], functional dependencies in XML (XFDs) have attracted little attention and their definition has been posed as an open problem in XML research [13].

A few papers have referred to the issue of XFDs [7, 10, 1]. Only the paper [1] has addressed the issue of how to define FDs in XML. The functional dependencies defined in [1] are on both internal nodes and on leaf nodes of XML trees and a normal form is proposed for XML documents. However, because the internal nodes of an XML tree do not have counterpart in the relational model, the involvement of internal nodes in the XFD definition and normal form in [1] causes difficulties in comparing relational database design techniques to XML document design techniques. This paper adopts a different approach to define XFDs so that the definition of XFDs does not involve internal nodes of XML trees.

The focus of this paper is to define XFDs and then to compare various classes of XFDs with FDs in relations and their effect on XML document design. We do this by firstly define primary XFDs for XML trees. We note that primary XFDs are defined on only the leaf nodes of an XML tree to match FDs of relations. They are not defined on internal nodes of the XML tree because internal nodes have no counterpart in relations. We then define two sub classes of primary XFDs, partial XFDs and transitive XFDs, and show that they can lead to similar problems in the design of XML documents as the same types of FDs in relational databases. We also show that the design problems can be eliminated if the partial and transitive dependencies are removed. We
note that we are not aiming in this paper to define a normal form in this paper, we are merely illustrating that some types of XFDs cause the same problems in XML documents as they do in relational database. We believe that the development of appropriate normal forms for XML is a more complex and subtle issue and will address the issue in a forthcoming paper.

We start our presentation by using an example.

2 An example

In this section, we give an example of a relation and RFDs on it. We then consider an XML representation of the relation. In the sections following, we define functional dependencies for XML and compare the relational functional dependencies on the relation with the XML functional dependencies on the XML representation.

<table>
<thead>
<tr>
<th>Table 1: A relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Did</td>
</tr>
<tr>
<td>d1</td>
</tr>
<tr>
<td>d1</td>
</tr>
<tr>
<td>d2</td>
</tr>
</tbody>
</table>

Table 1 depicts a relation that stores the data about departments, employees, projects and their relationships. A department is identified by Did. An employee is identified by Eid and has a name Ename. A project has an ID Pid and a name Pname. Each department and employee pair uniquely identifies a project. In other words, the following functional dependencies are proposed on the relation:

(FD1) Did, Eid → Pid
(FD2) Eid → Ename
(FD3) Pid → Pname

We note that FD2 is a partial dependency [11] and FD3 is a transitive dependency [11]. We also note that these functional dependencies are satisfied by the relation instance and because of the partial dependency and the transitive dependency, this relation is not normalized. It is obvious that redundancy exists in the table since, for example, the same employee name “john” is stored twice in the relation.

We now give an XML representation of the relation in Table 1. Because XML syntax allows information to be represented in different structures, there are many correct XML structures for representing Table 1. We have decided to employ the direct translation of each relation tuple to an XML tree branch. The translated XML tree is given in Figure 1.

As in its relational counterpart, the XML tree representation has redundancy. In the following sections, we discuss, from a functional dependency point of view, how this redundancy can be avoided during XML document design.

3 XML functional dependencies

In this section, we define primary XML functional dependencies. Before we give the definition, we introduce some notation.
**Definition 3.1 (XML DTD)** An XML DTD is a directed graph denoted by $D = (E, A, C, R, r)$ where $E$ is a countable infinite set of element names, $A$ is a countable infinite set of attribute names each preceded with the symbol @, $C$ and $R$ are two partial mappings, and $r$ is a special element name called root. For an element $e \in E$, the mapping $C$ defines child element names of $e$ and their allowed occurrence; if $e$ has the type of #PCDATA or EMPTY [3], $C(e)$ is empty. The mapping $R$ defines attributes of an element $e$. All elements in $E$ and $A$ are called labels. A label $l$ is a leaf label if $l$ is in $R(e)$ or $l = e' \in E$ and $C(e')$ is empty. Otherwise, $l$ is an interior label.

**Definition 3.2 (XML tree)** An XML tree is a tree defined to be $T = (V, children, lab, val, v_r)$ where $V$ is a set of nodes in $T$ and $v_r$ is the root node of the tree. $children$ is a partial function from $V$ to a sequence of nodes in $V$ which represent the children of the node. $lab$ is a total function from $V$ to $E \cup A$, i.e., it associates either an element name or an attribute name to a node. The partial function $val$ is defined on $V$ as follows. If a node $v$ is a leaf node of $T$, $val(v)$ is a string value which is either the content of a text element (#PCDATA [3]) or the content of an attribute; otherwise, $val(v)$ is not defined. We define a function called root to return $v_r$. For convenience of referencing sub trees, we call a sub tree $T'$ by $v_r$ if $root(T') = v_r$ where no confusion is raised.

We note that the above definition differs slightly in [1] in two ways. Firstly, we do no use String to label nodes in a tree. We use string values as values of leaf nodes. Secondly the $val$ function is defined on leaf nodes only.

**Definition 3.3 (conform)** Let $T = (V, children, lab, val, v_r)$ be an XML tree and $D = (E, A, C, R, r)$ be an XML DTD. For a node $v \in V$, let $V_1 = \{v_1, \cdots, v_m\}$ and $V_2 = \{v_{m+1}, \cdots, v_n\}$ be two partitions of $children(v)$, $\forall v_i \in V_1$, $lab(v_i) \in E$ and $\forall v_j \in V_2$, $lab(v_j) \in A$. $T$ conforms to $D$ if $lab(v_1), \cdots, lab(v_m)$ are defined in $C(lab(v))$ and $lab(v_{m+1}), \cdots, lab(v_n)$ are defined in $R(lab(v))$.

In the rest of this paper, we assume that the XML tree under discussion conforms to the DTD in context except otherwise being specified.

**Example 3.1** Figure 1 shows an XML tree. $v_7$ through to $v_{24}$ are nodes. $v_1$ is the root node of the tree. $v_2, v_4, v_5, v_7, v_8, v_{10}, \cdots$ are leaf nodes and $v_1, v_3, v_6, v_9, \cdots$ are interior nodes. There are no attribute nodes in
the tree; all leaf nodes are text nodes. $lab(v) = r$, $lab(v_1) = \text{dept}$, and $lab(v_2) = \text{Did}$. Labels on leaf nodes are bold faced. $val(v_1)$ is not defined because $v_1$ is an internal node. $val(v_2)$ is the string value “d1” because $v_2$ is a leaf node. The tree conforms to the DTD alongside.

**Definition 3.4 (path and primary path)** A path $p$ in an XML DTD $D = (E, A, C, R, r)$ is an expression of the form $l_1, \ldots, l_n$ ($n \geq 1$) where $l_1 = r, l_i \in E$ ($2 \leq i \leq n$), $l_n \in E$ or in $A$ such that $l_i \in C(l_{i-1})$, and $l_n \in C(l_{n-1})$ if $l_n \in E$ or $l_n \in R(l_{n-1})$ if $l_n \in A$. A path is a primary path if $l_n \in A$ or (if $l_n \in E$ and $C(l_n)$ is empty). That is, a primary path ends with a label for the leaf node of an XML tree. $l_n$ is called the end label of the path and denoted by $\text{endlab}(p)$. Two paths $p_1 = l_1, \ldots, l_n$ and $p_1' = l_1', \ldots, l_m'$ equivalent, denoted by $p_1 = p_2$, if $m = n$ and $l_i = l_i', 1 \leq i \leq n$. We use the notation $\text{subpath}(p, l_i)$ to mean the prefix of $p$ from $l_1$ to $l_i$. A sub path is also a path.

**Definition 3.5 (path instance)** A path instance $s$ over a path $p = l_1, \ldots, l_n$ in an XML tree $T = (V, \text{children}, \text{lab}, \text{val}, v_p)$ is an expression of the form $v_1, \ldots, v_n$ in $V$ such that $\text{lab}(v_i) = l_i$ for all $i, 1 \leq i \leq n$ and for any adjacent pair $v_i, v_{i+1}, v_i \in \text{children}(v_{i-1})$. $v_n$ is called the end node of the instance and is denoted by $\text{endnode}(s)$. The value of an instance $s$, denoted by $\text{instval}(s)$, is either defined by $\text{instval}(s) = \text{val(\text{endnode}(s))}$ if $s$ is an instance of a primary path or not defined if $s$ is an instance of a non-primary path.

We use the following notation conventions for paths and path instances. $p$ and $q$ denote paths. The capital letters $P$ and $Q$ denote sets of paths. $s$ and $t$ denote path instances and $S$ and $T$ sets of path instances. Note the difference between $T$ and $\mathcal{T}$; the latter is for a tree.

**Example 3.2** Consider the XML DTD and the tree in Figure 1. $r, r.\text{dept}$ are examples of paths and $r.\text{dept}.\text{Did}$ is a primary path. $\text{endlab}(r) = r$ and $\text{endlab}(r.\text{dept}.\text{Did}) = \text{Did}$. The only instance of the path $r$ is $v_r$. $v_r,v_1$ and $v_r,v_9$ are two of the instances over the path $r.\text{dept}$. $v_r,v_3,v_2$ and $v_r,v_9,v_{10}$ are two of the instances over the path $r.\text{dept}.\text{Did}$. $\text{instval}(v_r, v_1, v_2) = \text{val}(v_2) = “d1”$.

**Definition 3.6 (intersection of paths)** The intersection of two paths $p$ and $q$, denoted by $p \cap q$, is the maximal common prefix of both paths. It is clear that the intersection of two paths is also a path. For example in the DTD in Figure 1, let $p=r.\text{dept}.\text{Did}$ and $q=r.\text{dept}.\text{empl}.\text{Eid}$. Then, the path intersection $p \cap q = r.\text{dept}$.

**Definition 3.7 (intersection of path instances and meeting node)** Let $s$ be an instance over path $p$ and $t$ be an instance over path $q$. The intersection of the two instances $s$ and $t$, denoted by $s \cap t$, is the maximal common prefix of $s$ and $t$. We call $\text{endnode}(s \cap t)$ the meeting node of $s$ and $t$. It is clear that $s \cap t$ is an instance over a path. However, $s \cap t$ may not be the instance over the path $p \cap q$.

**Example 3.3** In Figure 1, let $p=r.\text{dept}.\text{Did}$ and $q=r.\text{dept}.\text{empl}.\text{Eid}$. Let $s_1 = v_r,v_1,v_2$ and $s_2 = v_r,v_9,v_{10}$ be two instances over $p$. Let $t = v_r,v_1,v_2,v_3$ be an instance over $q$. Then the path instance intersections $s_1 \cap t = v_r,v_1$ with the meeting node being $v_1$ and $s_2 \cap t = v_2$ with the meeting node being $v_2$. We note that $s_1 \cap t$ is an instance over $p \cap q$ but $s_2 \cap t$ is not although $s_1$ and $s_2$ both are the instances over $p$.

**Definition 3.8 (fellow)** Let $s$ be an instance over path $p$ and $t$ be an instance over path $q$. $s$ and $t$ are fellows if $s \cap t$ is an instance over the path $p \cap q$. That is, $\text{endnode}(s \cap t)$ is labelled with $\text{endlab}(p \cap q)$. Obviously when $p = q$, $s$ and $t$ are fellows if $s = t$. In Example 3.3, $s_1 \cap t$ are fellows because $s_1 \cap t$ is an instance over $p \cap q$. However, $s_2$ and $t$ are not fellows.
The concept of fellows requires that \( s \) and \( t \) be within a subtree specified by \( p \cap q \). This prevents path instances from different subtrees from being compared.

**Definition 3.9 (instance set)** Let \( P \) be a path set. The instance set \( S \) over \( P \) is a set such that (1) for every path \( p \) in \( P \) there is one and only one instance \( s \) over \( p \) in \( S \); (2) any pair of instances \( s_i \) and \( s_j \) in \( S \) are fellows.

**Example 3.4** Let \( P = \{ \text{r.A, r.M, r.M.B} \} \) be a path set in the DTD in Figure 2. Let \( S_1 = \{ v_5, v_1, v_4, v_2, v_3, v_6, v_3 \} \) and \( S_2 = \{ v_5, v_1, v_6, v_9, v_3, v_9, v_9 \} \). Then \( S_1 \) is an instance set over \( P \) and \( S_2 \) is not as we now explain. There is one and only instance in \( S_1 \) for each path in \( P \) and every pair of instances in \( S_1 \) are fellows and so \( S_1 \) is an instance set over \( P \). In contrast, in \( S_2 \), the instance \( v_5, v_5 \) is over the path \( r.M \) and \( v_9, v_9 \) is over \( r.M.B \). The intersection of the two path instances is \( v_5 \) and the intersection of the two paths is \( r.M, r.M \) is not an instance over \( r.M \). This means that the two instances are not fellows. Consequently \( S_2 \) is not an instance set over \( P \). In fact the instances \( v_5, v_9 \) and \( v_6, v_2, v_3 \) in \( S_2 \) are scattered in two sub trees.

![Figure 2: A DTD and an XML tree to show the complexities of XFD](image)

We now define values for instance sets. Let \( P \) be a set of primary paths and \( S \) be an instance set over \( P \). The value of \( S \), denoted by \( instSetVal(S) \), is defined by \( instSetVal(S) = \{ instval(s) | s \in S \} \).

**Definition 3.10 (fellow sets)** Let \( P \) and \( Q \) be two path sets. Let \( S \) be an instance set over \( P \) and \( T \) be an instance set over \( Q \). \( S \) and \( T \) are fellow sets if \( \forall s_i \in S \) and \( \forall t_j \in T \), \( s_i \) and \( t_j \) are fellows. Then \( P = Q \), \( S \) and \( T \) are fellow sets if \( S = T \).

The definition of fellows requires that instances in two fellow sets are within the same sub tree. This is demonstrated in the next example.

**Example 3.5** Let \( P = \{ \text{r.A, r.M, r.M.B} \} \) and \( Q = \{ \text{r.M.N, r.M.N.C, r.B} \} \) be two path sets in the DTD in Figure 2. Let \( S_1 = \{ v_5, v_1, v_4, v_2, v_3, v_6, v_3 \} \) and \( S_2 = \{ v_5, v_1, v_6, v_9, v_3, v_9, v_9 \} \). Let \( T_1 = \{ v_5, v_2, v_4, v_6, v_2, v_3, v_5, v_3, v_9, v_3, v_9, v_3, v_9 \} \) and \( T_2 = \{ v_5, v_2, v_1, v_6, v_6, v_6, v_6, v_6 \} \). Obviously, \( S_1 \) and \( S_2 \) are two instance sets over \( P \) and \( T_1 \) and \( T_2 \) are two instance sets over \( Q \). We can prove that \( T_1 \) and \( T_2 \) are fellow sets of \( S_1 \). However, \( T_1 \) and \( T_2 \) are not fellow sets of \( S_2 \). We explain this using \( T_1 \) and \( S_2 \). The instance \( v_5, v_2, v_3, v_3 \) in \( T_1 \) is over the path \( r.M.N \) and the instance \( v_5, v_5 \) in \( S_2 \) is over the path \( r.M \). The intersection \( v_5, v_2, v_3 \cap v_5, v_5 \) is not an instance over the path \( r.M \). As a result, \( v_5, v_2, v_3 \) and \( v_5, v_5 \) are not fellows. As a further consequence, \( T_1 \) and \( S_2 \) are not fellow sets. The key reason here is that the two instance sets are on two sub trees.
Definition 3.11 (Primary XML functional dependency) A primary XML functional dependency (PXFD) is a statement that \( P \) functionally determines \( Q \), denoted by \( P \rightarrow Q \), where \( P \) and \( Q \) are two sets of primary paths in an XML tree \( T \). \( P \) is called the left-hand side (LHS) or determinant and \( Q \) is called the right-hand side (RHS) or dependent.

An XML tree \( T \) conforming to the DTD satisfies the PXFD if for any two instance sets \( T_1 \) and \( T_2 \) over \( Q \), \( instSetVal(T_1) \neq instSetVal(T_2) \), then there exist non-empty \( S_1 \), the set of all formal sets of \( T_1 \) over \( P \), and non-empty \( S_2 \), the set of all formal sets of \( T_2 \) over \( P \), such that \( \exists S_1 \neq \text{null} \in (S_1 - S_2) \land \exists S_2 \neq \text{null} \in (S_2 - S_1) \), \( instSetVal(S_1) \neq instSetVal(S_2) \). A PXFD is single LHSed if there is only one path in \( Q \). Otherwise, it is multiple LHSed. If there is only one path in \( P \) or \( Q \), we use the path to replace the path set.

In the rest of this paper, we use XFD to mean PXFD unless otherwise specified. The following example shows the application of the XFD definition.

Example 3.6 Let \( P = \{r.A, r.M.B\} \) and \( Q = \{r.M.N.C, r.D\} \) be two path sets in the DTD in Figure 2. Then the XML tree in the figure violates \( P \rightarrow Q \) but satisfies \( Q \rightarrow P \).

Proof of the violation of \( P \rightarrow Q \): Let \( T_1 = \{v_1,v_2,v_3,v_7, v_9,v_{13}\} \) and \( T_2 = \{v_1,v_2,v_6,v_8, v_9,v_{13}\} \) be two instance sets over \( Q \). Then, \( instSetVal(T_1) = \{"c1","d1"\} \neq instSetVal(T_2) = \{"c2","d1"\} \). We can find that \( S = \{v_9,v_1, v_9,v_2,v_3\} \) is the formal set for both \( T_1 \) and \( T_2 \) (i.e., \( S_1 = S_2 = \{S\} \)). There does not exist a formal set in \( S_1 \) but not in \( S_2 \). By the XFD definition, \( P \rightarrow Q \) is not satisfied.

Proof of the satisfaction of \( Q \rightarrow P \): Let \( T_1 = \{v_1,v_1, v_9,v_{13}/v_3\} \) and \( T_2 = \{v_1,v_1, v_9,v_{10}\} \) be two instance sets over \( P \). Actually, \( T_1 \) and \( T_2 \) are the only two instance sets of \( P \). \( instSetVal(T_1) = \{"c1","d1"\} \neq instSetVal(T_2) = \{"c1","d2"\} \). We can find two informal sets \( S_1' = \{v_1,v_2,v_6,v_7, v_9,v_{13}\} \) and \( S_2' = \{v_1,v_2,v_6,v_8, v_9,v_{13}\} \) of \( T_1 \) over \( P \), i.e., \( S_1 = \{S_1', S_2\} \). We can also find a formal set \( S_2 = \{v_1,v_9,v_{11},v_2, v_9,v_{13}\} \) of \( T_2 \), i.e., \( S_2 = \{S_2\} \). In \( S_1 \), we choose \( S_1' \) which is not in \( S_2 \). Then \( instSetVal(S_1' = \{"c1","d1"\} \neq instSetVal(S_2) = \{"c3","d1"\} \). By the XFD definition, \( Q \rightarrow P \) is satisfied.

The following example shows the satisfaction of XFDs using our running example in Figure 1.

Example 3.7 Let \( P = \{r.deptEMPL.Eid\} \) and \( Q = \{r.deptEMPL.Ename\} \). Then the XFD \( P \rightarrow Q \) is satisfied by the tree in Figure 1. We arbitrarily choose two instance sets of \( Q \): \( T_1 = \{v_1,v_1,v_3,v_5\} \) and \( T_2 = \{v_1,v_9,v_1,v_{13}\} \). \( instSetVal(T_1) = \{"ohn"\} \neq instSetVal(T_2) = \{"fre\}" \). There exist two informal sets \( S_1 = \{v_1,v_1,v_3,v_5\} \) and \( S_2 = \{v_1,v_9,v_1,v_{12}\} \) over \( P \), \( S_1 \) being the only formal set of \( T_1 \) and \( S_2 \) the only formal set of \( T_2 \), \( S_1 \neq S_2 \), such that \( instSetVal(S_1) = \{"ohn"\} \neq instSetVal(S_2) = \{"e2\}" \). The tree satisfies the XFD.

Definition 3.12 (XFD implication) Let \( \Sigma \) be a set of XFDs and \( P' \rightarrow Q' \) be an XFD. \( \Sigma \) implies \( P' \rightarrow Q' \) if every XML tree \( T \) that satisfies \( \Sigma \) also satisfies \( P' \rightarrow Q' \). Then we use \( \Sigma^+ \) to denote all XFDs implied by \( \Sigma \).

Theorem 3.1 Let \( P \) and \( Q \) be path sets. \( P \rightarrow Q \) implies \( P \rightarrow q_i \) (\( \forall q_i \in Q \)); \( P \rightarrow q_i \) (\( \forall q_i \in Q \)) implies \( P \rightarrow Q \).

The theorem can be explained by re-examining the definition of an XFD. For any path instances \( t_1 \) and \( t_2 \) over the path \( q_i \), if \( instval(t_1) \neq instval(t_2) \), we can find formal sets \( T_1 \) and \( T_2 \) over \( Q \), \( t_1 \in T_1 \), \( t_2 \in T_2 \), and \( instSetVal(T_1) \neq instSetVal(T_2) \). Because \( P \rightarrow Q \), there exist formal sets \( S_1 \) of \( T_1 \) and \( S_2 \) of \( T_2 \) over \( P \) such that \( instSetVal(S_1) \neq instSetVal(S_2) \).

With this theorem, an XFD with multiple paths in the RHS can always be split into several XFDs each of which has a single path in the RHS. Similarly a group of XFDs with the same LHS and single path in the RHS can be combined into a single XFD.
We now use an example to show a special category of XFDs. In Figure 3, stud stands for student and subj stands for subject. Let root.stud \rightarrow root.subj be an XFD. Then the tree in part (b) of the figure violates the XFD and the tree in part (c) satisfies the XFD. We first explain Part (b) of the figure. Let \( t_1 = v_r,v_3 \) and \( t_2 = v_r,v_4 \) be two instances of the RHS path root.subj, \( instud(t_1) \neq instud(t_2) \). There exist two paths instances \( s_1 = v_r,v_1 \) and \( s_1' = v_r,v_1 \) which are fellows of both \( t_1 \) and \( t_2 \). That is, \( S_1 = S_2 = \{ s_1, s_2 \} \). There does not exist an instance in \( S_1 - S_2 \) and so the XFD is violated. Now we look at part (c). This part satisfies the definition because we can not find two different instances of the RHS path.

\[
\begin{align*}
\text{XFD: root.stud} & \rightarrow \text{root.subj} \\
\text{Fig. 3: Flimsy XFD}
\end{align*}
\]

From this example we see that an XFD like \( r.stud \rightarrow r.subj \) can only be satisfied by XML trees that have one subject value. This is formalized by the following definition.

**Definition 3.13 (flimsy XFD)** An XFD \( P \rightarrow Q \) is flimsy if \( \forall p \in P \) and \( \forall q \in Q \), \( p \cap q = r \).

Flimsy XFDs form a special category of general XFDs. It is a distinct category compared with the functional dependencies in the relational model. Because a flimsy XFD has to be satisfied by trees with one instance set of the RHD, we exclude this type of XFDs from the rest of our discussion.

We now define the concepts that correspond to partial and transitive dependencies of the relational model.

**Definition 3.14 (Partial XFD)** \( P \rightarrow Q \) is a partial XFD if there exists another XFD \( P' \rightarrow Q' \) in \( \Sigma \), \( Q \cap P = \phi \) and \( Q \cap Q' = \phi \), such that \( P \subseteq P' \).

**Definition 3.15 (Transitive XFD)** \( P \rightarrow Q \) is a transitive XFD if there exists another XFD \( P' \rightarrow Q' \) in \( \Sigma \), \( Q \cap P' = \phi \), such that \( Q' = P \). \( P' \rightarrow Q' \) is called the primary XFD of the transitive XFD.

We use Figure 1 to show examples. The following XFDs hold over the tree in Figure 1.

(XFD1) \[ \{r.dept.bid, r.dept.empl.Eid\} \rightarrow r.dept.empl.proj.Pid \]

(XFD2) \[ r.dept.empl.Eid \rightarrow r.dept.empl.Ename \]

(XFD3) \[ r.dept.empl.proj.Pid \rightarrow r.dept.empl.proj.Pname \]

By the definitions above, we see that XFD2 is partial dependency because we can find XFD1 such that the LHS of XFD2 is contained in the LHS of XFD1. We also see that XFD3 is a transitive dependency because the LHS of XFD3 is the RHS of XFD1. Consequently, these functional dependencies correspond to their flat relation counterparts FD1-FD3 given in Section 2 (see Page 2).

### 4 Inference rules of XFDs

In the relational model, there are three basic inference rules: reflexivity, augmentation, and transitivity. We now show that each of these rules can be extended to XFDs directly.
Theorem 4.1 The following inference rules hold over XFDs.

1. Reflexivity: \( P \rightarrow P \) if \( P \subseteq P \).
2. Augmentation: \( P \rightarrow Q \) implies \( (P \cup H) \rightarrow (Q \cup H) \).
3. Transitivity: \( P \rightarrow Q \) and \( Q \rightarrow Q' \) implies \( P \rightarrow Q' \).

Proof:

1. Reflexivity. The proof is similar to that of Theorem 3.1.

2. Augmentation. For two different instance sets \( T_1 \) and \( T_2 \) over \((P \cup H)\), let \( S_1 \) be the set of all fellow sets of \( T_1 \) over \((P \cup H)\) and \( S_2 \) be the set of all fellow sets of \( T_2 \) over \((P \cup H)\). We need to prove that \( \exists S_1 \in (S_1 - S_2) \) and \( \exists S_2 \in (S_2 - S_1) \) such that the values of \( S_1 \) and \( S_2 \) are different.

Firstly, we partition \( T_1 \), \( T_2 \), \( S_1 \), and \( S_2 \) so that the partitions are over \( P \), \( Q \), and \( H \) respectively. Let \( T_1^P \), \( T_2^P \), \( T_1^Q \), \( T_2^Q \) be partitions of \( T_1 \) and \( T_2 \) such that \( T_1^P \), \( T_2^P \) are over \( Q \), and \( T_1^Q \), \( T_2^Q \) are over \( H \). Let \( S_1^P \), \( S_2^P \), \( S_1^Q \), \( S_2^Q \) be partitions of \( S_1 \) and \( S_2 \) such that \( S_1^P \), \( S_2^P \) are over \( P \), and \( S_1^Q \), \( S_2^Q \) are over \( H \).

There are two cases for \( T_1 \) and \( T_2 \) being different:

- (a) \( T_1^P \) and \( T_2^P \) are different. Because \( P \rightarrow Q \), \( S_1^P \neq S_2^P \). This causes \( S_1 \neq S_2 \). The rule is true.
- (b) \( T_1^Q \) and \( T_2^Q \) are different. Then \( T_1^Q = S_1^P \) and \( T_2^Q = S_2^P \) because of the definition of fellow sets. Because \( T_1^Q \) and \( T_2^Q \) are different, we have \( S_1^Q \) and \( S_2^Q \) are different. So, \( S_1 \neq S_2 \). The rule is true.

3. Transitivity. For two different instance sets \( T_1 \) and \( T_2 \) over \( Q' \), there exist two different \( T_1 \) and \( T_2 \) over \( Q \) because \( Q \rightarrow Q' \). For \( T_1 \) and \( T_2 \), there exist two different \( S_1 \) and \( S_2 \) over \( P \). Because \( P \rightarrow Q \), the rule is proved.

We use the next example to show how the augmentation rule works.

Example 4.1 The tree in Figure 4 satisfies \( r.m.A \rightarrow r.m.B \). So it satisfies \{\{r.m.A, r.m.C\}\} \rightarrow \{\{r.m.B, r.m.C\}\} too.

This is because for two instance sets \( T_1 = \{v_r, v_1, v_2, v_r, v_1, v_4\} \) and \( T_2 = \{v_r, v_1, v_3, v_r, v_1, v_5\} \) over \{\{r.m.B, r.m.C\}\}, \( valInstSet(T_1) \neq valInstSet(T_2) \), we can find \( S_1 = \{v_r, v_1, v_2, v_r, v_1, v_4\} \) and \( S_2 = \{v_r, v_1, v_2, v_r, v_1, v_5\} \), where \( S_1 \) is the only fellow set of \( T_1 \) and \( S_2 \) is the only fellow set of \( T_2 \). \( S_1 \) and \( S_2 \) are over \{\{r.m.A, r.m.C\}\}. \( S_1 \neq S_2 \), \( valInstSet(S_1) \neq valInstSet(S_2) \). We note that in this example, \( v_r, v_1, v_4 \) is not a fellow of \( v_r, v_1, v_5 \) because both instances are over \( r.m.C \).

![Figure 4: An example of the augmentation rule](image-url)
5 Design issues

In this section, we discuss design issues of XML. We first review the normalization method of the relational model. We then extend the method to XML. We define the concepts of XFD trees to enable us to transform DTD and XML trees to remove redundancy. We further define the direction of an XFD. With the help of the direction, we can maximize the nesting of elements without causing redundancy.

We note that in this paper, we aim to improve XML DTD design, but do not propose a normal form. Normal forms in XML have been proposed in [1, 7]. The normal form defined in [1] requires XFDs defined on internal nodes. This is not always the case because internal nodes in XML trees do not have counterparts in the relational model. When data in a relational database is exported to XML, the XFDs translated would be primary. The normal form defined in [7] requires functional dependencies defined in a conceptual model which limits its use to those situation where XML document is derived from a high level conceptual models. We believe the problem of normal form is more complex when XFDs do not involve internal nodes or no keys [4] are involved. As a result, we focus on improving XML document design.

5.1 Flat relations

In the relational model, if a relation has partial dependencies or transitive dependencies, then the relation potentially has redundancy. The relation can be normalized by decomposing the relation into smaller relations such that the decomposed relations do not have partial or transitive dependencies. For example, partial dependencies and transitive dependencies exist in Table 1. The design of the table can be improved. A better design is given in Table 2.

<table>
<thead>
<tr>
<th>Eid</th>
<th>Ename</th>
<th>Pid</th>
<th>Pname</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>john</td>
<td>p1</td>
<td>mining</td>
</tr>
<tr>
<td>e2</td>
<td>fred</td>
<td>p2</td>
<td>integration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bid</th>
<th>Eid</th>
<th>Pid</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>e1</td>
<td>p1</td>
</tr>
<tr>
<td>d1</td>
<td>e2</td>
<td>p2</td>
</tr>
<tr>
<td>d2</td>
<td>e1</td>
<td>p2</td>
</tr>
</tbody>
</table>

This method of normalization can be extended to XML design.

5.2 XML documents

In this subsection, we extend the normalization method used in the relational model to XML. Before we do so, we define some terms.

Definition 5.1 (isomorphic) Two trees $T$ and $T'$ are isomorphic if there exists a one-to-one mapping $f$ between the vertices of $T$ and the vertices of $T'$ such that (1) a node $n_1$ is a child of a node $n_2$ in $T$ iff $f(n_1)$ is a child of $f(n_2)$ in $T'$; (2) for a node $n \in T$, $lab(n) = lab(f(n))$.

Definition 5.2 (information equality) Two trees $T$ and $T'$ are information equivalent if they are isomorphic and for a leaf node $n$ in $T$, $val(n) = val(f(n))$ where $f$ is the one-to-one mapping between the vertices of $T$ and the vertices of $T'$.
**Definition 5.3** (XFD graph) Let $\mathcal{F}: P \rightarrow Q$ be an XFD on a DTD $\mathcal{D}$. Let $pp$ denote a path that is the intersection of all paths in $P$ and in $Q$. An XFD graph, denoted by $\mathcal{F}$, is a sub DTD graph of $\mathcal{D}$ such that (1) the root of $\mathcal{F}$ is the end label of $pp$; (2) all other labels of $\mathcal{F}$ belong to a path in $P$ or in $Q$.

**Definition 5.4** (XFD instance tree) Let $\mathcal{F}: P \rightarrow Q$ be an XFD on a DTD. Let $\mathcal{T}$ be a tree conforming to the DTD. Let $pp$ be the intersection of all paths in $P$ and in $Q$. Let $s$ be an instance over $pp$. An XFD instance tree, denoted by $\mathcal{T}$, over $\mathcal{F}$ is a sub tree of $\mathcal{T}$ such that (1) the root node of $\mathcal{T}$ is the end node of $s$; (2) all nodes of $\mathcal{T}$ belong to a path instance over a path in $P$ or in $Q$. Figure 5. (b) shows examples of XFD instance trees.

![Figure 5: XFD trees](image)

**Proposition 5.1** In an XFD instance tree $\mathcal{T}$ over an XFD $\mathcal{F}: P \rightarrow Q$, if there are more than one instance set over $Q$, then all the instance sets are information equivalent.

This proposition is true because the XFD would be violated otherwise. The XFD instance tree in the middle of Figure 5. (b) is an example.

**Definition 5.5** (concise XFD instance tree) An XFD instance tree $\mathcal{T}$ over $\mathcal{F}: P \rightarrow Q$ is concise if there do not exist two information equivalent instance sets over $Q$ and there do not exist two information equivalent instance sets over $P$.

**Definition 5.6** (redundancy) [12] Let $\Sigma$ be a set of XFDs over a DTD. Let $\mathcal{T}$ be a tree conforming to the DTD. A value $x$ in $\mathcal{T}$ is redundant with respect to $\Sigma$ if a replacement of $x$ by $x'$ ($x' \neq x$) causes $\Sigma$ to be violated. If a value in $\mathcal{T}$ is redundant with respect to $\Sigma$, then $\Sigma$ is redundant.

We now extend the normalization method in the relational model to XML.

**Proposition 5.2** Let $\Sigma$ be a set of XFDs over a DTD. The $\Sigma$ is redundant if there are partial and transitive XFDs in $\Sigma$. 

10
The proposition is true because the XML tree in Figure 1 has redundancy. Note that DTD has redundancy means as long as there exists a tree conforming to the DTD and having redundancy.

A DTD containing partial and transitive XFDs can be transformed to one that does not containing partial and transitive XFDs and redundancy can be removed in this way. The transformation is done by the following algorithm. The algorithm also transforms XFDs that applicable to the old DTD to those that are applicable to the new DTD. The main idea of the algorithm is to move partial and transitive XFD graphs to the root of the DTD.

**Algorithm 5.1 (transform DTD and XFDs)**

Input: an XML DTD $\mathcal{D}$ and a set $\Sigma$ of XFDs.

Do:

1. If there is no partial and transitive XFDs, exit.
2. Let $\mathcal{F}$: $P \rightarrow Q$ be a partial XFD or a transitive XFD which is not a primary XFD of other transitive XFDs in $\Sigma$. Let $pp$ be the intersection of all paths in $P$ and $Q$. Let $l = endlab(pp)$. Let $\mu \mathcal{D}$ be the XFD tree of $\mathcal{F}$. Make a copy of $\mu \mathcal{D}$ and denote the copy as $\mu \mathcal{D}'$.
   a. Rename all internal labels of $\mu \mathcal{D}'$ by appending the apostrophe symbol (') to each internal label of $\mu \mathcal{D}'$. This process would change $l$ to $l'$.
   b. Delete the leaf labels of all dependent paths of the original $\mu \mathcal{D}$;
   c. Delete the labels that were interior labels and have become leaf labels after the deletion in (b).
3. $\mathcal{F}$ is transformed to $\mathcal{F}'$: $P' \rightarrow Q'$ by:
   
   $P' = \{ p' \mid \forall p \in P \ (p' = \text{rename}(\text{prefix}(p, l)) \}$ and
   
   $Q' = \{ q' \mid \forall q \in Q \ (q' = \text{rename}(\text{prefix}(q, l)) \}$
   
   where $\text{prefix}(q, l)$ means replace the prefix before $l$ of $q$ by $r$, and $\text{rename}(x)$ means to append the apostrophe symbol after each label (except for $r$ and $endlab(x)$) of $x$.
4. goto step (1).

Output: $\mathcal{T}'$ and $\Sigma'$.

Figure 6 shows a DTD which is transformed from the DTD in Figure 1 using the above algorithm. The transformed XFDs are given alongside.

When a DTD is transformed from $\mathcal{D}$ to $\mathcal{D}'$, a tree $\mathcal{T}$ conforming to $\mathcal{D}$ has to be transformed accordingly to $\mathcal{T}'$ so that $\mathcal{T}'$ conforms to $\mathcal{D}'$. This is done by the following algorithm. The algorithm also removes duplicating path instances to make XFD instance tree concise.

**Algorithm 5.2 (transform XML tree)**

Input: an XML tree $\mathcal{T}$ and a set $\Sigma$ of XFDs.

Do:

1. For every XFD $\mathcal{F}$ in $\Sigma$, if there exists a XFD instance tree $\mathcal{T}'$ over $\mathcal{F}$ that is not concise, delete the leaf nodes of a instance set $S$ (or $T$) over $P$ (or over $Q$) in $\mathcal{T}'$ if $S$ (or $T$) is information equivalent to another instance set over $P$ (or over $Q$) in $\mathcal{T}$. Then delete any nodes that were internal nodes and that have become leaf nodes. Repeat this step until no non-concise XFD trees exist.
(2) If there is no partial and transitive XFDs, exit.
(3) Let $\mathcal{F} : P \rightarrow Q$ be a partial XFD or a transitive XFD which is not a primary XFD of other transitive XFDs in $\Sigma$. Let $pp$ be the intersection of all paths in $P$ and $Q$. Let $l = endlab(pp)$.
(4) If there are more than one XFD instance tree over $\mathcal{F}$, let $\mathcal{T}$ be the one that is the most further from the root node of $\mathcal{F}$.
(a) If there does not exist an XFD instance tree $\mathcal{T}'$ of $\mathcal{F}$ directly attached to root($\mathcal{T}$) such that $\mathcal{T}$ and $\mathcal{T}'$ are information equivalent, then make a copy of $\mathcal{T}$ and denote the copy by $\mathcal{T}''$;
 Otherwise, goto (c);
(b) renumber all interior nodes of $\mathcal{T}''$ and all leaf nodes that belong to an instance set of $P$; rename all labels on internal nodes of $\mathcal{T}''$ by adding the apostrophe symbol after them; attach $\mathcal{T}''$ to root($\mathcal{T}$);
(c) delete the leaf nodes of all dependent instances of $\mathcal{T}$ and then delete any nodes that were internal nodes and that have become leaf nodes.
(5) Remove $\mathcal{F}$ from $\Sigma$.
(6) goto step (2).

Output: normalized $\mathcal{T}$.

Figure 7 shows an XML tree which is transformed from Figure 1 using the above tree transformation algorithm.

6 Related work

The problem of functional dependencies relates to constraints. There is a number of publications on constraints of XML [8, 7, 5, 1]. These publications discuss different types of constraints for XML such as key constraints [8, 7, 5], inclusion constraints [8, 7], multivalued constraints [7], and inverse constraints [9]. Some of these [9, 6] also investigated implication and satisfiability of constraints.

XML functional dependencies form an important part of constraints and have been addressed in several work [1, 7, 10]. The functional dependencies defined in [1] are on both internal nodes and on leaf nodes. However, the satisfaction definition in [1] allows a dependent to exist without a dependent when there are more than
one tree tuples in the tree. This is shown in Figure 8. We believe this does not match the case of relational functional dependency satisfaction when null values are not allowed or when strong satisfaction interpretation [2] of null values is used. We improved this definition by defining XML functional dependencies with a stronger satisfaction definition.

Functional dependencies are used in [7] in a straightforward way. This is because [7] focuses on transforming a conceptual hypergraph model to an XML document. The paper also proposed an normal form based on redundancy. This is different from our method of using partial dependencies and transitive dependencies as a judging criteria, which is on DTD level.

The definition of functional dependencies in [10] uses key attributes. It requires that on each level of the tree (except for the root), there has to be an attribute identifying the node. We believe this requirement is too strong and is not generally applicable. Furthermore, the model used in the paper does not use node identifiers. This results in that a path instance can not be identified. Even for the path instances that end with leaf nodes that has a key value on it, the unique identification is still not possible because replication of key values in the tree. In other words, it is not possible in this model to distinguish same elements in different positions of an XML document.
7 Conclusion

In this paper, we defined the primary XML functional dependencies which are on leaf nodes of XML trees. We discussed the properties of a special type of XFDs: flimsy functional dependencies. We studied the inference rules of the primary XFDs. Based on this, we extended the relational normalization method to XML to improve XML document design. We proposed two algorithms for DTD and XML tree normalization.

The future work in this area is to investigate completeness of the inference rules of primary XFDs and to investigate normal forms based on primary XFDs and keys.

References


