Defining Functional Dependency for XML

Kamsuriah Ahmad, Ali Mamat, Hamidah Ibrahim and Shahrul Azman Mohd Noah
Faculty of Information Science and Technology,
Universiti Kebangsaan Malaysia, 43600, Bangi, Selangor, Malaysia.
Email: kam@ftsm.ukm.my, ali@fsktm.upm.edu.my, hamidah@fsktm.upm.edu.my,
samn@ftsm.ukm.my

ABSTRACT
In relational model, functional dependency is a very well established concept. It is the technique used to enforce data integrity by specifying the constraints that may exist in the relations and in discovering redundancies during database design. XML as a new technology in data management need a similar technique to achieve data integrity in its environment. As the number of applications using native XML documents is increasing rapidly, there exist great interests in how to extend this concept in XML environment. However generalizing relational constraints to XML constraints is nontrivial because of the hierarchical and flexible structure of XML compared with flat representation of relational table. In this paper we propose a formal definition of functional dependency for XML that can specify constraints in hierarchical nested structure of XML. The advantage of this definition compared with the existing definitions is that, it’s able to specify constraints that hold not only in the entire document (globally) but also can scope within the sub-document (locally).

Keywords: Functional Dependency, XML, Database Design.

1.0 Introduction
Extensible Markup Language (XML) has recently emerged as a new format for data exchange. The number of applications using native XML document is increasing rapidly. Even though XML provides a high degree of syntactic flexibility, but this language lacks of mechanism for specifying the semantics of its data. Consequently, the study of integrity constraints has been recognized as one of the most important yet challenging areas of XML research (Fan 2005; Fan and Simeon 2003). The importance of XML constraints is due to a wide range of applications ranging from schema design, query optimization, efficient storing and updating, data exchange and data integration (Fan 2005). Semantic constraints in relational databases such as in keys and functional dependencies play a fundamental role, where they are used in integrity enforcement and in database design (Elmasri and Navathe 2006). Similarly, these constraints would be expected to play a fundamental role in XML. However it is not immediately obvious how to extend the definitions of functional dependencies from relations to XML because of the hierarchical and flexible structure of XML. As in relational databases, functional dependency for XML (XFD) is used to describe the property that the values of some attributes of a tuple uniquely determine the values of other attributes of the tuple (Lee et al. 2002; Arenas and Libkin 2004). The difference lies in the attributes and tuples, which are basic units in relational databases, whereas in XML document, they must be defined using path expressions. For example, a language for defining XFD must allow us to specify when one path is a prefix of another and is able to specify local constraints that hold only on sub-document rather than on an entire document. Therefore, another notation that takes into consideration of XML’s hierarchical characteristics is needed badly.

The theory of functional dependency for relational databases cannot be directly applied in XML document as there are significant differences in their structures: relational are flat while XML schemas are nested. Attempts to define functional dependency have been made by several groups of researchers (Arenas and Libkin 2004; Vincent et al. 2004; Wang and Topor 2005; Yu and Jagadish 2008). However the definitions of XFD in these works are all difference with different expressive power and do not always represent the same type of constraints in an XML document. Not surprisingly there is no consensus yet as to which definition is the “best”. A common problem with these definitions is the limited expressive power. As an example none of the existing definitions are able to express the constraints in the presence of shared element...
and local element in one specification language. A definition of XFD that based on sub-graph has been defined in Hartman and Link (2003), which is orthogonal to our definition of XFD that based on path expressions. The concept of XML keys is also proposed in (Buneman et al. 2001; Hartmann and Link 2007) but the concept of XFD is more generalized than keys and more constraints can be specified. To define this concept, therefore the study of path expression and value equality between nodes needs to investigate further to adopt the hierarchical nested nature of XML and how to define XFD in this setting. To investigate this issue, the paper is organized as follows. A motivating example is given in section 2 to demonstrate the difference of functional dependency when they are applied in relational databases and XML documents. The general definition of XML functional dependency is given in section 3. Section 4 concludes the paper and points out the directions of future work.

2.0 Motivating Example

As an example, consider the XML tree representation in Figure 1, which describes the information on courses and students. The number of courses offered and the list of students who took particular courses are shown in the tree. For each student who took the course a grade will be given. Given this document and our understanding of its semantics, we may wish to state the following constraints:

C1: Each courses is identified by its name (cname),
C2: If two students have the same student number (sno), they must have the same name,
C3: In a given course, each student gets a single grade.

The first constraint is an example of an absolute key constraint, where sub-element cno is a key for element course in the whole document or the occurrence of cno is unique globally. The second constraint is an example of functional dependency (FD) and the third constraint is an example of a local constraint, where each student will get their grade for each course they took. It means that the student number (sno) will determine the student grade and is defined locally only for student of particular course. The student number (sno) will not determine the grade of a student for other course. For example the student number “A100” got an “A” for “Database” and she got a “B” for “Networking”. This type of constraint will lead to redundancies as we can see that student with the name “Siti” is redundantly stored as she took Database and Networking.
course. This is because a student node is nested under course node. And for each courses, her name will appear repeatedly according to the number of courses they took and redundantly stored in the tree. Most existing definitions of XFD (Arenas and Libkin, 2004; Vincent et al., 2004) are based on globally distinct values in the whole XML document. Therefore they cannot express that the student number (sno) will determine the grade because it violates the local constraint in the tree. In other words, we want to be able to express the semantics that a functional dependency holds locally distinct values in a sub tree but not in the whole tree. From the Figure 1, the constraint C3 hold in each of the sub trees rooted locally at node v6, v10, v18 and v22 separately but not in the whole tree. From the above observation functional dependency in XML (XFD) has two major characteristics when comparing with that in relational databases:

i. FD in relational is global presentation; while in XML we may need to define local dependency as XML is nested and tree-structured.

ii. FD in relation only consider attribute in its presentation; while in XML they may consider complex elements.

Being able to express local and global XFD is important to adopt the hierarchical structure of XML data. A more general definition of functional dependency for XML is needed to express these constraints.

3.0 Functional Dependency for XML (XFDs)

Our notion of functional dependency is based on a tree model of XML data, as illustrated in Figure 1. Although the model is quite simple, we need to do two things prior to define functional dependency for XML. The first is to give a precise definition of value equality for XML functional dependency. In the relational model, no duplicate tuples are allowed, and so value equality is sufficient in FDs. However, in an XML document, we can have two different sub-trees that are isomorphic and have exactly the same values. This is a clear instance of data duplication, but allowed by the data model. Therefore, if XFDs are to be used for normalizing database, they must be able to specify equality between nodes in the document. The second is to describe a path language that will be used to locate sets of nodes in an XML document. We therefore introduce a class of regular expression, and define XFDs in terms of this path language. In order to capture the hierarchical nature of XML, the definition of XFD must be able to express global XFDs that hold on an entire XML document, and local XFDs that hold only on sub-document. To capture the nested structure of XML, we defined XFD with more general path expressions that allow the Kleene star (in regular expressions), wildcard or descendants in XPath (Clark and Derose 1999).

3.1 Basic Notations

The Document Object Model (DOM) (Apparao et al. 1998) provides some insight into semantics for XML documents. According to DOM, the document is a hierarchical structure of nodes and can be viewed as a node-labeled tree. Nodes consist of three important types: element node, attribute nodes and text nodes. Element nodes (E) have names but do not carry text, text nodes have no name but carry text while attribute nodes (A) have both name and carry text. Element nodes may have children; attribute and text nodes, which are leaf nodes. We call elements that have sub-elements and/or attributes as complex elements denote as \( E_1 \) and elements that only have a single value or a text node as a simple element denote as \( E_2 \). Simple elements serve similar purposes to attributes, the only difference is that multiple simple elements with the same label may appear as children of a complex element, but each attribute of a complex element has a distinct label. We have explicitly distinguished complex and simple elements so that the special text node under a simple element is not needed since they can be represented by attribute. The intuition for this is to have easier presentation and computation of proposed functional dependencies later. Now, let \( E_1 \) and \( E_2 \) be disjoint sets of element names, \( A \) be a set of attribute names, then \( E = E_1 \cup E_2 \), where \( E \) and \( A \) be disjoint. Since we considered defining
XFDs in the presence of DTD (Data Type Definition) that consists of a set of complex and simple elements, and attributes then we need to define the formal approach of XML DTDs, a set of paths in a DTD, a valid XML tree and a set of path in an XML tree. DTD is a schema language that used to strict the structure of XML document and defined as below:

**Definition 1**: A DTD schema $D$ is denoted by 6 tuples $= (E_1, E_2, A, P, R, r)$ where
- $E_1$ is a finite set of complex element types
- $E_2$ is a finite set of simple element types
- $A$ is a finite set of attributes
- $P$ is a mapping function from $E_1$ to element type definitions: $\forall \tau \in E_1, P(\tau)$ is a regular expression
  $$\alpha = \varepsilon | \tau^* | \alpha|\alpha | \alpha, \alpha | \alpha^*$$
  where $\varepsilon$ is the empty sequence, $\tau^* \in E_1 \cup E_2$, and “$|$”, “$*$”, and “$^*$” denote union, concatenation, and the Kleene closure, respectively;
- $R$ is a mapping function from $E_1$ to a set of attributes
- $r$ is the start symbol and is called the element type of the root. Without loss of generality we assume that $r$ does not occur in $P(\tau)$ for any $E_1 \cup E_2$ and distinct from all other symbols.

**Example**: According to the above definition, DTD $D$ in Figure 1 is defined as $D = (E_1, E_2, A, P, R, r)$, where
- $E_1 = \{\text{courses, course, students, student}\}$,
- $E_2 = \{\text{cname, name, grade}\}$,
- $A = \{\text{cno, sno}\}$
- $P(\text{courses}) = \text{course}^*$,
- $P(\text{course}) = \text{cname, students}$,
- $P(\text{students}) = \text{students}^*$,
- $P(\text{student}) = \text{name, grade}$
- $P(\text{course}) = P(\text{cname}) = P(\text{name}) = P(\text{grade}) = \text{string}$
- $R(\text{course}) = \text{cno}$
- $R(\text{student}) = \text{sno}$,
- $R(\text{cno}) = R(\text{sno}) = \varepsilon$,
- $r = \text{courses}$

Element names and attribute names are called labels. A label $l$ is a leaf label if $l$ is in $R(e)$ or $l = E_2$, otherwise it is an internal label. Here, we are not concerned with the question of whether some information is better modeled as an attribute or text element. In addition DOM also specifies how to reach the children of an element node. A consequence of this model is that a path of edge labels uniquely identifies children. Such paths are called path id and written as $l_1/\ldots/l_n$.

**Definition 2**: A path $p$ in $D = (E_1, E_2, A, P, R, r)$ is defined to be $p = l_1, \ldots, l_n$ where
  i. $l_i = r$;
  ii. $l_i \in P(l_{i-1})$, $i \in \{2, \ldots, n-1\}$;
  iii. $l_n \in P(l_{n-1})$ if
      - $l_n \in E_1$ and $P(l_n) \neq \varepsilon$ , or
      - $l_n = \text{string}$ if $l_n \in E_2$ and $P(l_n) = \varepsilon$ , or
  iv. $l_n \in R(l_{n-1})$ if $l_n \in A$.

Let $\text{paths}(D) = \{p \mid p$ is a path in $D\}$ then courses/course/cno, courses/course/students/student/sno are some paths in $\text{paths}(D)$, and courses/course $\subseteq$ courses/course/student. We assume that all DTDs considered in this study are consistent (i.e.
there is at least one finite tree satisfying the DTD). The conformity of an XML tree to a DTD is defined as follows:

**Definition 3**: An XML tree is defined to be \( T = (V, \text{lab}, \text{ele}, \text{att}, \text{val}, \text{root}) \), where

1. \( V \) is a set of nodes in \( T \);
2. \( \text{lab} \) is a mapping from \( V \rightarrow E \cup A \) which assigns a label to each node in \( V \); a node \( v \) in \( V \) is called a complex element node if \( \text{lab}(v) \in E_1 \), a simple element node if \( \text{lab}(v) \in E_2 \), and an attribute node of \( \text{lab}(v) \in A \).
3. \( \text{ele} \) and \( \text{att} \) are functions from the set of complex elements in \( V \): for every \( v \in V \), if \( \text{lab}(v) \in E_1 \), then \( \text{ele}(v) \) is a set of complex element nodes, and \( \text{att}(v) \) is a set of attribute nodes with distinct labels.
4. \( \text{val} \) is a function that assigns a values to each attribute or simple element. If a node \( v \) is a leaf node of \( T \), \( \text{val}(v) \) is a string value which is either the content of a text element (#PCDATA) or the content of an attribute; otherwise \( \text{val}(v) \) returns the node identifier of \( v \).
5. \( \text{root} \) is the unique root node labeled with complex element name \( r \).
6. If \( v' \in \text{ele}(v) \cup \text{att}(v) \), then we call \( v' \) a child of \( v \). The parent-child relationships defined by \( \text{ele} \) and \( \text{att} \) will form a tree rooted at root.

Note that the definition of \( \text{val} \) differs slightly from that in (Buneman et al., 2001a) since it was extended to define on complex element nodes. The reason for this is that to include the complex element node in the path definitions and able to compare elements by node identity, i.e node equality as illustrates in the following example:

**Example:**
From Figure 1, node \( v_1 \) through \( v_{25} \) are nodes. Node \( v_1 \) is the root node of the tree. \( v_2, v_3, v_7, v_8, \ldots \) are leaf nodes and \( v_1, v_2, v_5, \ldots \) are internal nodes. \( \text{lab}(v_1) = \text{courses}, \text{lab}(v_2) = \text{course} \) and \( \text{lab}(v_3) = \text{cno} \). \( \text{val}(v_2) = v_2 \) because \( v_2 \) is an internal node. \( \text{val}(v_3) = \text{c100} \) because \( v_3 \) is the leaf node.

**Definition 4**: An XML tree \( T = (V, \text{lab}, \text{ele}, \text{att}, \text{val}, \text{root}) \) is said to conform to a DTD \( D = (E_1, E_2, A, P, R, r) \) if

1. \( \text{lab}(\text{root}) = r \),
2. \( \text{lab} \) maps every node in \( V \) to \( E_1 \cup E_2 \cup A \),
3. for every complex node \( v \in V \), if \( \text{ele}(v) = \{v_1, \ldots, v_k\} \), then a permutation of the sequence \( \text{lab}(v_1), \ldots, \text{lab}(v_k) \) must be in the language defined by \( P(\text{lab}(v)) \); if \( \text{att}(v) = \{v'_1, \ldots, v'_m\} \) then \( \text{lab}(v'_1), \ldots, \text{lab}(v'_m) \) must be in the set of \( R(\text{lab}(v)) \).

Clearly if XML tree \( T \) conforms to DTD \( D \), then every simple path of \( T \) is valid with respect to the root and written as \( T \models D \). In this study, we assume that the XML tree under discussion conforms to the DTD unless stated otherwise. Now we give the definition of value equality of two nodes. Intuitively, two nodes are value equal if and only if the two sub-tree rooted on the two nodes are identical.

**Definition 5**: Value and node equality

1. Let \( n_1 \) and \( n_2 \) be two nodes in \( T \). We say \( n_1 \) and \( n_2 \) are value equal, denoted \( n_1 \equiv n_2 \), if \( n_1 \) and \( n_2 \) are of the same label in the trees, and
2. \( n_1 \) and \( n_2 \) are both attribute nodes or simple element nodes, and the two nodes have the same value, or
3. \( n_1 \) and \( n_2 \) are both complex elements, and for every child node \( m_1 \) of \( n_1 \), there is a child node \( m_2 \) of \( n_2 \) such that \( m_1 \equiv m_2 \), and vice versa.
3.2 A Functional Dependency Constraint Language for XML

After defining the value equality and the path expression features, a definition of functional dependency constraint language for XML is proposed that based on these features.

**Definition 6: XFDs constraint language**

Let \( \varphi = P, Q: X \rightarrow Y \) be a functional dependencies for XML (XFDs) where,

i. \( P \in \text{paths}(D) \) is downward context path starting from the root, which identifies the scope of \( \varphi \) over \( D \). If \( P \neq r \) and \( P \neq \epsilon \) (where \( \epsilon \) means empty path), then \( \varphi \) is called a local XFD, which means that the scope of \( \varphi \) is the sub-tree rooted at \( P \) otherwise, \( \varphi \) is called a global XFD, which means the scope of \( \varphi \) is the whole \( D \) or at the root.

ii. \( Q \) is called downward target path, where \( Q \in \text{paths}(D), Q \subseteq P \).

iii. \( X \rightarrow Y \) is a Left-Hand-Side (LHS) and Right Hand Side (RHS) of \( \varphi \), which is a non-empty subsets of \( \text{paths}(D) \) rooted at \([Q]\).

If \( n \) is a node in XML tree and \( P \) is the path expression then \( n[[P]] \) denotes a set of nodes that can be reached by following the path expression \( P \) from node \( n \). The idea is that the context path \( P \) identifies a set of nodes \([P]\), each of which is refer to as a header or context path. For each context node \( n[[P]] \), the XFD constraint must hold on the target set \( n[[Q]] \). A node \( m \) in the target set is called a target node. Figure 2 illustrates the way that the XFD been defined in the XML tree and specifies the following:

i. The context path \( P \), starting from the root of an XML tree \( T \), identifies a set of nodes in \([P]\);

ii. For each \( n_1 \in [P] \), \( \varphi \) defines a global XFD on the subtree rooted at \( n_1 \); specifically,

iii. The target path \( Q \) identifies a set of nodes \( n_1[[Q]] \) in the subtree, referred as the target set,

iv. The node in \( X \) path uniquely identifies the node in the \( Y \) path in the target set. That is, for each \( n_2 \in n_1[[Q]] \) the values of the nodes reached by following \( X \) path from \( n_2 \) uniquely identify the values of nodes reached by following \( Y \) path from \( n_2 \).

![Figure 2: Illustration of XFD \( \varphi = P: Q : X \rightarrow Y \)](image)

Drawing an analogy with FD in relational databases, the target set \( Q \) corresponds to a relation name in the relational model, and the \( X \) and \( Y \) path corresponds to the attributes of FD in the relation. There is no relational counterpart to the context path \( P \) since relations are flat (not hierarchical). Given the basic notation of XFD, next we need to define what it means for an XML document to satisfy a functional dependency constraint.

**Definition 7: XFDs satisfaction**

An XML tree \( T \) conforming to DTD \( D \), \( T \models D \), is said to satisfy an XFDs \( \varphi \) \((P, Q: X \rightarrow Y)\) denote \( T \models \varphi \), if and only if for any nodes \( n \in [P] \) (let \( n = \text{root} \) if \( P = \epsilon \)) and for any two nodes \( x_1, x_2 \in n[[Q]] \) if they agree on \( X \) i.e. for non-empty \( x_1[[X]] \) and \( x_2[[X]] \) such that \( x_1[[X]] = v, x_2[[X]] \), then they must agree on \( Y \) i.e. for non-empty \( x_1[[Y]] \) and \( x_2[[Y]] \) such that \( x_1[[Y]] = v, x_2[[Y]] \). That is,

\[
\forall n \in [P], \forall x_1, x_2 \in n[[Q]], (x_1[[X]] = v, x_2[[X]]) \rightarrow (x_1[[Y]] = v, x_2[[Y]])
\]
Observe that when \( P = \varepsilon \), i.e., when \( \varphi \) is a local XFDs, the set \([[[P]]]\) consists of a unique node, namely, the root of the tree. In this case \( T \models \varphi \) if and only if
\[
\forall x_1, x_2 \in [[[Q]]], (x_1[[X]] = v, x_2[[X]]) \rightarrow (x_1[[Y]] = v, x_2[[Y]])
\]
The definition claimed that for any two instances of sub-trees identified by the XFD header, \( P \) if all LHS entities agree on the values, then they must also agree on the value of the RHS entity if it exists and the LHS entities must exist. If a LHS entity (or RHS entity) is a simple element type (\( E_2 \)), its value is equivalent to that of the text child (PCDATA) of the element, and to the value of the attribute otherwise. For example, for the XFD
\[
XFD: \quad //</student>: sno -> name
\]
the value of \( sno \) is equal to the value of the attribute \( sno \) while the value of \( name \) is equal to the PCDATA under simple element \( name \). The above XFD is satisfied since \([[[/student]]] = \{ v_6, v_{10}, v_{18}, v_{22} \} \) and \( \text{val}(v_6/sno) = \text{val}(v_{10}/sno) \) and \( \text{val}(v_{18}/name) = \text{val}(v_{22}/name) \). But if a LHS entity (or RHS entity) is end by a complex element type (\( E_1 \)), its value is equivalent to the node identifier. For example,
\[
XFD: \quad //</course>: cno -> students
\]
where \([[[course]]] = \{ v_2, v_4 \} \) and \( \text{val}(cno) = "c100" \) and \( \text{val}(students) = v_5 \). Being able to use identifier equality, allow us to express the structure of XML in the form of XFDs. Drawing an analogy with functional dependency in relational databases, the target path \( Q \) corresponds to a relation name in the relational model, and the set of X path and Y path corresponds to the set of attributes in that relation. There is no relational counterpart to the context path \( P \) since relations are flat (not hierarchical). As an example to express the three constraints that been discussed in section 2, the following XFDs are:
\[
\begin{align*}
F_1: & </course>: cno -> \varepsilon \\
F_2: & </student>: sno -> sname \\
F_3: & </course>: student: sno -> grade
\end{align*}
\]
where \( F_1 \) is a global XFDs, which implies that a course number (\( cno \)) can uniquely determines a course node within the whole XML document, where the symbol \( \varepsilon \) means all paths from the root; \( F_2 \) is a global XFDs which implies that a student number (\( sno \)) can uniquely determines a student’s name within the whole XML document. However \( F_3 \) is a local XFDs, which implies that a student number (\( sno \)) can uniquely determine the grade within the sub-tree rooted at the course node.

### 4.0 Conclusion and Future Works

Functional dependencies are very important semantic information in XML document, which are fundamental to other related XML research topics such as normalizing XML documents and query optimization. This paper extends the theory of functional dependency in relational database world to the XML world and proposes a more general definition of functional dependency for XML (XFD) that is able to differentiate between local and global dependencies in XML documents. These types of dependencies are important to adopt the hierarchical structure of XML documents. To increase the expressiveness of XFD, we need to view this constraint from a different angle by using more expressive path languages. As proposed by Clark and Derose (1999), path in XML can go upward and downward direction in the tree and the study of XFD with respect to this path language is interesting to explore. This suggestion make proposing a standard definition of XFD are still open. In this study we are only concerned on functional dependency (FD) and neglect another aspect of dependency that is multi-valued dependency, which is a generalization of FD. This area needs to explore further if we want to add more semantic features of XML. As in relations, the notion of functional dependency constraints generalizes the constraints in key. These two notions are very important in designing normalized relations. Therefore we would like to extend the work to study the relationship.
between functional dependencies and keys in XML document and further develop a methodology for XML normalization.

References


