Functional Dependency based on XML Node Sets

Husheng Liao
Beijing University of Technology, Beijing, China
Email: liaohs@bjut.edu.cn

Jia Wu and Jia Liu
Beijing University of Technology, Beijing, China
Email: {daisy711, jeromeliu}@emails.bjut.edu.cn

***Abstract*—As an essential basis of relational database theory, integrity constraints such as functional dependency provide a basis for well-designed databases. Integrity constraints are also useful for the normalization of the XML schema design in the expensive applications of XML data. As a semi-structure feature, XML data are usually located by a path expression and multiple data items may be represented by the same path. Thus, functional dependencies for XML should be constraints between sets of XML data items if the path expression is used. These constraints also result in data redundancy. Same as functional dependency, this kind of data redundancy for XML can lead to update anomalies too. This paper proposes a kind of XML integrity constraint to describe the dependent relationship between different sets of XML data items, and defines a general functional dependency based on XML node sets. Moreover, this paper proposes a group of inference rules for the implication problem of the XML functional dependency, and proves that they are sound and complete.**

***Index Terms*—Integrity constraints, XML, Functional dependencies, Database semantics**

I. Introduction

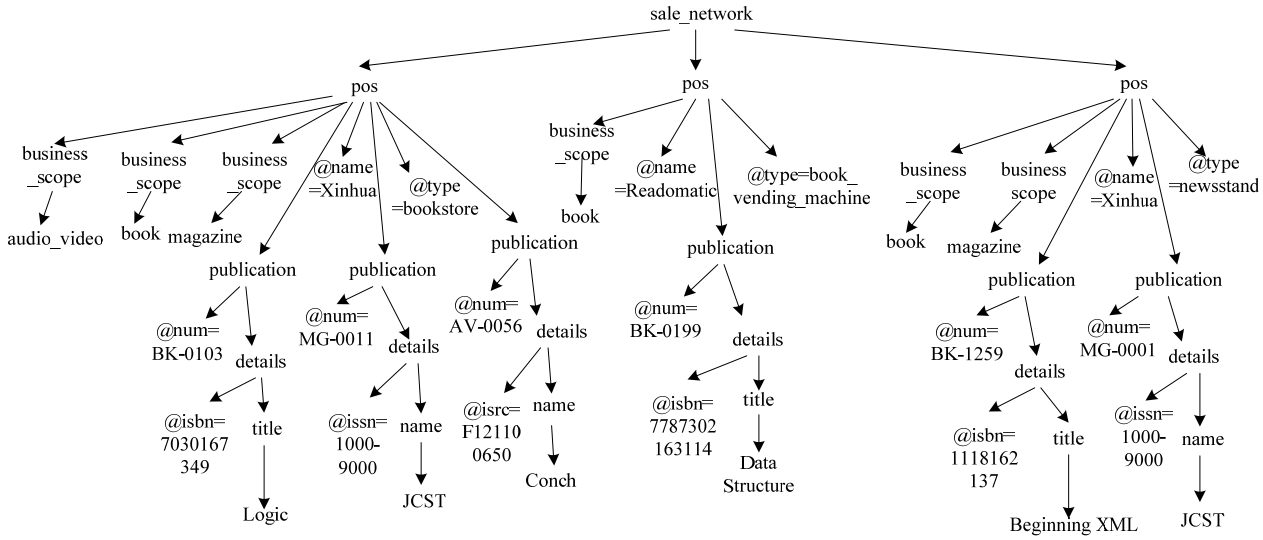
In database systems, redundant data would result in the anomalies of data updating. It is well-known that this phenomenon comes from the poorly designed data schema, and data should be organized according to functional dependencies between data items. Functional dependency is the basic theory of relational databases design. For similar reasons, the well-designed XML schema also needs the integrity constraint theory such as XML functional dependency. Due to the expensive applications of XML data and its unique characteristics, many research focus on the XML integrity constraint problem, including key of XML data[1], path constraint[2][3], functional dependency[4][5], multi-valued dependency[6], inclusion dependency[7], and so on. Functional dependency has been of great concern for many years and the definition of it has been considered to be an open problem in XML research. The researches nowadays on XML functional dependency commonly follow the concepts of functional dependency in relational databases, and study the functional dependencies between different XML data items. A problem is that if path expressions are used to locate the XML data items, the functional dependencies for XML should be constraints between sets of XML data items, since multiple items can be located by the single path. But most works on functional dependency for XML are based on the assumption that only one XML data item can be located for every path in their definitions.

Compared with relational databases, XML data is of tree structure and every XML element may have multiple child elements. In relational datebase, a relation is made up of tuples, and a tuple is made up of a list of attributes, each attribute is supposed to be diffirent with each other and only appear once, but in XML document data elements with the same label may be sibling node. Therefore, they may be located by the same path expression. It should be noted that there are some dependent relationships between these data item sets represented by the different path expressions. This kind of constraints may also bring data redundancy which results in update anomalies. Therefore, in this paper we call this constraint as functional dependency based on XML node sets, FDXS for short, and we propose a group of integrity constraint definitions to describe the XML functional dependencies.

An XML document example is given to explain FDXS. The XML document is used to store the information of different type of point of sale (pos) for publications. A sale network has various types of pos, a pos may be a bookstore, a book vending machine or a newsstand and every pos has its unique name. Pos with different type have different business scope, for example, the business scope for bookstore includes book, magazine and audio video, newsstand is restricted the sale of book and magazine, and only book can be saled by book vending machine, as shown in Fig.1. Suppose this sale network has several pos, so the root of this XML tree is the *sale_network* element, it has several child nodes labeled with *pos*. Each *pos* element includes an attribute *name*, an attribute *type*, several *business_scope* elements which

Manuscript received June 29, 2013; accepted August 2, 2013.

doi:10.4304/jsw.9.4.1036-1044



stands for the business scope of the pos and should appear at least once, and a sequence of *publication* elements for the information of publications. Each *publication* has a unique *num* attribute and a *details* element to store the specifications of the publication. The structures of *details* elements are different because three kinds of publications can be sold: books, magazines and audio video products. Hence, specification information of the different publications is different from each other. For books, it stores ISBN number and book name. For magazines, it includes ISSN number and name. For the audio video products, it stores ISCR number and name. The structure of the XML tree is described by the regular expression types [8] in Table 1.

TABLE 1.
REGULAR EXPRESSION TYPES FOR THE XML TREE IN FIG.1.

T_SALENETWORK = sale_network[T_POS]*
T_POS =pos[business_scope[text]+, @name, @type, publication[@num, T_DETAILS]]*
T_DETAILS = details[T_BOOK | T_MAGAZINE | T_AUDIO_VIDEO]
T_BOOK =@isbn, title[text]
T_MAGAZINE = @issn, name[text]
T_AUDIO_VIDEO = @isrc, name[text]

XML functional dependency is a kind of constraints between the values of the reachable nodes via different paths. For example, suppose the point of sale with different type has different business scope. For an element *pos*, if the set of values of its child element labeled by *business_scope* is { *book, magazine, audio_video*} , then the value of its *type* attribute must be *bookstore*. If the set of values of the *business_scope* elements is either { *book* } or { *book, magazine* }, then the value of its *type* attribute must be either *book_vending_machine* or *newsstand* respectively. This constraint is that the set of values of the *business_scope* element determines the value of its *type* attribute for every *pos* element in the XML tree. In other words, for every *pos* element, if the sets of values of its *business_scope* children are different, its *type* attributes must have different value. In this paper, the constraint is expressed as:

sale_network/pos: business_ scope → type

It means that for any two nodes in the reachable nodes via path *sale_network/pos*, if the sets of values that their child element labeled by *business_scope* are same, the values of their *type* attributes are same, too. Similarly, another FDXS for the XML tree is that *name* attribute determines the set of values of *business_scope* element for these points of sale, which is represented by the following form:

sale_network/pos: name → business_scope

It means that in different *pos* elements, if the values of the *name* attribute are same, the sets of values of *business_scope* element must be same. These FDXS may also lead to data redundancy: the value of *type* attribute will be stored repeatedly for *pos* elements with the same set of values of *business_scope* elements.

On the other hand, based on the two FDXS above, another FDXS can be obtained:

sale_network/pos: name → type

It means that there are some kinds of inference rules on FDXS and implication relations. Therefore, implication checking needs to be considered when reasoning about the XML functional dependencies.

In this paper, we study functional dependency based on XML node sets and its inference rules. Main contributions are as follows:

1. According to the integrity constraints between the sets of XML data items, we propose the concept of functional dependency based on XML node sets.
2. We give a formal definitions of functional dependencies based on XML node sets by the path-based notation. In the reachable nodes via a specific path (context path), they express a kind of integrity constraints between the set of reachable nodes via different relative paths.
3. For the FDXSs with the same context path, we propose a sound and complete set of FDXS inference rules which can be used for the judgment of the logical implication for various FDXSs. We also define an equivalent relation between FDXSs with different context paths and rewriting rules. The FDXS inference rules and rewriting rules forms an inference system called FDXS system.

In the following sections, preliminary definitions are introduced in Section II. Formal definitions of FDXS are presented in Section III. In Section IV, FDXS logical implication is studied and a sound and complete set of FDXS inference rules and rewriting rules is proposed. Some related works are discussed in Section V. Conclusion and some future works are presented in Section VI.

II. Preliminary Definitions

In this section, some basic definitions are given for the rest of the paper. Suppose that *Ele* is a finite set of XML labels, *Att* is a finite set of attributes and *text* is the symbol representing text in the following definitions.

Definition 2.1. *(XML tree) An XML tree is defined to be Tr=(V, lab, val, root), where V is a finite set of XML nodes; lab is a mapping from V to Ele ∪ Att ∪ {text}; a node v is called an element node if lab(v) ∈ Ele, an attribute node if lab(v) ∈ Att, and a text node if lab(v)=text; the function* ***val*** *is defined as:*

$val(v)$ = value of the attribute node if $lab(v) \in Att$
content of the text node if $lab(v)=text$
a sequence of its child nodes $<v_1, ..., v_n>$ if $lab(v) \in Ele$

Definition 2.2. *(path and path instance) A path is either an expression in the form of $s_1/s_2/.../s_n (n>0)$ where $s_i \in Ele \cup Att$ $(1 \le i \le n)$ or* ***self****. A path instance* ***w*** *defined over a path in an XML tree (V, lab, val, root) is a sequence of nodes in the form of $<v_1, v_2, ..., v_n>$ where v_i is a child of v_{i-1} for all $v_i \in V (1<i \le n)$. Any path instance w can be said to be defined over the path if $lab(v_i)=s_i$ $(0<i \le n)$. Moreover, no path instance is defined over* ***self****.*

For example, in the XML tree shown in Fig.1, six path instances are defined over the path *sale_network/pos/publication*, and every path includes a *sale_network* node, a *pos* node and a *publication* node sequentially.

Definition 2.3. *(Prefix of path) Every path in the form of $s/s_2/.../s_i$ $(0<i<n)$ is said to be the prefix of the path $s_1/s_2/.../s_n$. Path* ***self*** *is the prefix of any path.*

For example, paths *self* and *sale_network* are both the prefixes of the path *sale_network/pos.*

Definition 2.4. *(Reachable node and function Nodes) Given a node v in an XML tree and a path instance w in the form of $<v_1,v_2, ...,v_n>$ defined over a path, if v1 is a child node of v, we say w is the path instance relative to the node v. All the path instances, which are defined over a given path p and relative to a given node v, consist a path instance set denoted by paths(v,p). The last node of every path instance is called the reachable node relative to node v via path p. For path* ***self****, paths(v, self) = {<v>}. The function Nodes is a set of reachable nodes relative to v via the path p defined by:*

$Nodes(v, p) = \{ vn \mid <v_1,v_2, ...,v_n> \in paths(v, p) \}$

For example, from the root of the XML tree in Fig.1, *paths(pos/publication/num)* returns six path instances and each of them is composed of an *sale_network* element, an *pos* element, an *publication* element and a *num* attribute node in turn. *Nodes(sale_network, pos/publication/num)* will return six reachable nodes labeled with *num*.

Since FDXS are based on the value comparison between XML nodes, the definitions of value equality equality for XML nodes are given as follows:

Definition 2.5. *(Node value equality) Let u and w be nodes in an XML tree, they are value equal, denoted by $u=_{val}w$, if and only if the following conditions are satisfied:*

1. *lab(u) = lab(w)*
2. *if lab(u),lab(w) ∈ Att, then val(u)=val(w)*
3. *if lab(u)=lab(w)=text, then val(u)=val(w)*
4. *if lab(u),lab(w) ∈ Ele, and let $val(u)= <v_1, ..., v_n>$, $val(w)=<v'_1, ...,v'_m>$, then m=n and $v_i=_{val} v'_i (1 \le i \le n)$.*

If nodes *u* and *w* are not value equal, we denote it by $u \neq_{val} w$.

Definition 2.6. *(Value equality for reachable nodes) Let x and y be nodes in an XML tree, p a path, we say that x and y are value equal for the reachable nodes via the path p, denoted by $x.p =_{val} y.p$, if and only if the following conditions are satisfied:*

1. *if Nodes(x,p) is empty, then Nodes(y,p) is empty too, and vice versa.*
2. *for every $u \in Nodes(x,p)$, there is $w \in Nodes(y,p)$ that satisfies $u=_{val} w$.*
3. *for every $w \in Nodes(y,p)$, there is $u \in Nodes(x,p)$ that satisfies $u=_{val} w$.*

If *x* and *y* are not value equal on the reachable nodes via the path *p*, we denote it by $x.p \neq_{val} y.p$.

For example, the first two *pos* elements in Fig.1 are not value equal for reachable nodes via the path *business_scope*, because the *business_scope* element with the value *audio_vedio* is the reachable node via the path related to the first *pos* element and no *business_scope* element with the same value exists under the second *pos* element. In fact, any two *pos* elements are not value equal for reachable nodes via the path *business_scope.*

It should be noted that Definition 2.6 is based on the comparisons between XML node sets. This definition is different from the definitions on value equality of XML nodes adopted in the definitions of XML function dependencies [4][5][11].

Definition 2.7. *(Effective path set) Let Trs be an XML tree set which always ranges over all possible XML tree, s and p be paths, the effective path set of s in Trs is defined as $U(s)=\{ p \mid (V,lab,val,root) \in Trs, x \in Nodes(root,s), paths(x,p) \neq \Phi \}$ and for an XML tree Tr, its effective path set of s is defined as $E(s,Tr) = \{ p \mid x \in Nodes(root,s), paths(x,p) \neq \Phi, (V,lab,val,root)=Tr \}$.*

According to this definition, there must be some reachable nodes via any path in *U(s)* from some reachable nodes via the path *s* in the XML trees. And no reachable node exists via any path if it is not a member of *U(s).* In general, the effective path set of any path can be found according to the given DTD, XML Schema or regular expression types for XML dataset.

In this paper, FDXS is defined over the relationships between XML node sets and the effective path set of a context path in XML trees.

III. Functional Dependency Based On XML Data Set

For clear representation, a preliminary definition is given as follows:

Definition 3.1. *Given two distinct nodes x, y in an XML tree and a path set A, x and y are said to be value equal for the reachable nodes via the path set A, denoted by $A::x =_{val} y$, if $x.p =_{val} y.p$ for every path p in A. If they are not value equal, we denote it by $A::x \neq_{val} y$.*

Based on Definition 3.1, formal definition of functional dependency based on XML data set is given as follows:

Definition 3.2. *A functional dependency based on XML node set (FDXS) is a proposition in the form $s:A \rightarrow B$ where s is a path represeting the context; A, B are subsets of U(s). The XML tree Tr satisfies FDXS if and only if for any two distinct nodes $x, y \in Nodes(root, s)$, if $B::x \neq_{val} y$ then there exists $A::x \neq_{val} y$.*

For example, for any point of sale, suppose that its name determines its type, and its name also determines its bussiness scope in the XML tree shown in Fig.1, the following FDXSs hold:

sale_network/pos: name → type

sale_network/pos: name → business_scope

sale_network/pos: business_scope → type

These dependencies are also characterized by their transitivity. The first FDXS may be derived from other FDXSs. This situation causes new problems to emerge in the design theory of XML schema, such as logical implication for FDXSs and normalization with FDXSs.

Moreover, according to the FDXS definition, there is a FDXS hold in any XML trees. For any context path *s* and any path *q* and its prefix *p*, the following FDXS always holds:

s: p → q

In the XML tree shown in Fig.1, *sale_network: pos → pos/name* holds. Although such XML dependencies do not lead to data redundancy, they have effect on the logical implication of FDXSs.

IV. Logical Implication And Inference Rules For FDXS

Like functional dependencies for relational databases, the FDXSs for XML trees are often not independent, so that it is necessary to address the implication problem of FDXSs. To decide whether an XML tree satisfies a given set of FDXSs, such implication should be checked to reduce the cost of the computation. In the design of an XML schema, every FDXS which holds in the corresponding XML trees should be also taken into account. Therefore, it is necessary to derive other FDXSs satisfied by the XML tree from the given FDXSs.

Definition 4.1. *(FDXS logical implication) Given an FDXS set Σ and an FDXS σ, if σ holds for the XML trees that satisfy all FDXSs in Σ, then we say that Σ implies σ, denoted by $\Sigma \models \sigma$.*

By extending the Armstrong axioms of relational databases for the analysis of the logical implication on FDXSs, we propose a group of inference rules for the derivation between the FDXSs. Since the notation based on the context path is used in the definition of FDXSs, these inference rules involve not only the derivation with the same context path but also ones with different context paths.

A. FDXS Inference Rules with the Same Context Paths

In order to judge the implication, the inference rules between the different FDXSs should be taken into account. The inference rules for any context path *s* are given as follows:

Rule X1. *(Prefix-path value-dependency rule) If path p is the prefix of path q, then s: $p \rightarrow q$.*

Proof. Suppose that path *q* is concatenated by its prefix *p* and path q' ($q=p/q'$). FDXS $s:p \rightarrow q$ means that $y.p \neq_{val} z.p$ can be derived from $y.q \neq_{val} z.q$ for any two distinct nodes *y* and *z* obtained from *Nodes(root, s)*. By the reduction to absurdity, we assume that $y.p =_{val} z.p$ if $y.q \neq_{val} z.q$. Let $M = Nodes(y, p)$ and $N = Nodes(z, p)$, it means that either both *M* and *N* must be empty sets, or for every node $y' \in M$ there is a node z' in *N* that satisfy $y' =_{val} z'$ and vice versa. We prove Rule X1 for the three cases as follows:

1. If both *M* and *N* are empty, then *Nodes(y,q)* and *Nodes(z,q)* must return empty sets since path *p* is the prefix of path *q*. It contradicts the assumption $y.q \neq_{val} z.q$.
2. Consider the case that for every node $y' \in M$, there is a node z' in *N* that satisfies $y' =_{val} z'$. According to Definition 2.6, the corresponding descendants of y' and z' should be value equal. Hence, there are the two cases for every node y' in *M*:
   a) If $Nodes(y',q')=\Phi$, then $Nodes(z',q')=\Phi$. Since path *q* is concatenated by *p* and q' ($q=p/q'$), both of *Nodes(y,q)* and *Nodes(z,q)* should be Φ and it contradicts the assumption $y.q \neq_{val} z.q$.
   b) If $Nodes(y',q') \neq \Phi$, then for every node y'' obtained from $Nodes(y',q')$, which is the descendant of y', there must be a node z'' obtained from $Nodes(z',q')$ that satisfies $y'' =_{val} z''$ according to Definition 2.6, since $y' =_{val} z'$ and z'' is the descendant of z'. Since $q=p/q'$, there are $y'' \in Nodes(y,q)$ and $z'' \in Nodes(z,q)$. It means for every node y'' there is a node z'' that satisfies $y'' =_{val} z''$.
3. Consider the case that for every node $z' \in N$, there is a node y' in *M* that satisfies $y' =_{val} z'$. According to Definition 2.6, the corresponding descendants of y' and z' should be value equal. Hence, there are the two cases for every node z' in *N*:
   a) If $Nodes(z', q')= \Phi$, then $Nodes(y',q')=\Phi$. Since path *q* is concatenated by *p* and q' ($q=p/q'$), both of *Nodes(y,q)* and *Nodes(z,q)* should be Φ and it contradicts the assumption $y.q \neq_{val} z.q$.
   b) If $Nodes(z',q')$ is not empty, then for every node z'' obtained from $Nodes(z',q')$, which is the descendant of z', there must be a node y'' obtained from $Nodes(y',q')$ that satisfies $y'' =_{val} z''$ according to Definition 2.6, since $y' =_{val} z'$ and y'' is the descendant of y'. Since $q=p/q'$, there are $y'' \in Nodes(y,q)$ and $z'' \in Nodes(z,q)$. It means for

every node z'' there is a node y'' that satisfies $y''=_{val}z''$.

To sum up 2.b) and 3.b), this means that for node y and z, $y.q=_{val} z.q$ holds. It also contradicts the assumption $y.q\neq_{val} z.q$. In conclusion, the original proposition is proved.

Rule X2. *if $B\subseteq A$, then s: $A\rightarrow B$.*

Proof. For any two distinct nodes x and y obtained from *Nodes(root, s)*, if $B::x\neq_{val}y$ then $A::x\neq_{val}y$ since the path set B is a part of A. As a result, *s:* $A\rightarrow B$.

Rule X3. *(augmentation rules)if s: $A\rightarrow B$ then s: $AE\rightarrow BE$.*

Proof. The premise of Rule X3 means that for any two distinct nodes x and y in *Nodes(root,s),* if $B::x\neq_{val} y$ then $A::x\neq_{val} y$. Thus, if $B::x\neq_{val} y$, then $BE::x\neq_{val} y$ for the extended path set BE and it also derive $AE::x\neq_{val}y$. As a result, *s:* $AE\rightarrow BE$.

Rule X4. *(Transitivity rule) if s: $A\rightarrow B$ and s: $B\rightarrow C$, then s: $A\rightarrow C$.*

Proof. *s:* $B\rightarrow C$ means that for any two distinct nodes x and y in *Nodes(root,s)*, if $C::x\neq_{val} y$, then B.$::x\neq_{val} y$; On the other hand, *s:*$A\rightarrow B$ means that if $B::x\neq_{val} y$, then $A::x\neq_{val}y$. As a result, $A::x\neq_{val} y$ since $C::x\neq_{val}y$. In other words, *s:* $A\rightarrow C$.

Rule X1, X2 are trivial FDXSs since they have been proven for any XML tree. Thus, these trivial FDXS are implied by any FDXS set. Other inference rules are non-trivial FDXSs. The proof above states that if their premises are implied by a given FDXS set, it also implies their conclusions.

B. Extended FDXS Inference Rules

On the basis of FDXS inference rules showed above, we can get the following extended inference rules:

Rule X5. *(Union rule) If s: $A\rightarrow B$ and s: $A\rightarrow C$, then s: $A\rightarrow BC$.*

Proof. Under augmentation rules X3, if *s:*$A\rightarrow B$, then $AA\rightarrow BA$; In other words, *s:*$A\rightarrow BA$. Similarly, if *s:*$A\rightarrow C$, then *s:*$BA\rightarrow BC$. According to Rule X4, *s:*$A\rightarrow BC$.

Rule X6. *(Pseudo-reflexivity rule) If s: $A\rightarrow B$ and s: $DB\rightarrow C$, then s: $AD\rightarrow C$, where D is arbitrary path sets.*

Proof. According to Rules X3, if *s:* $A\rightarrow B$, then *s:* $AD\rightarrow BD$. Using transitivity rule X4, it can derive *s:* $AD\rightarrow C$ and it means Rule X6 is sound.

Rule X7. *(Path decomposition rule) When s: $A\rightarrow B$ holds, if $C\subseteq B$ then s: $A\rightarrow C$.*

Proof. According to Rules X2, *s:*$B\rightarrow C$ hodls, and using transitivity rule X4, *s:* $A\rightarrow C$ can be derived from them. It means Rule X7 is sound.

C. The Equivalence of FDXSs with Different Context Paths and Rewriting Rules

Due to the notation of the FDXSs above, some of FDXSs with different context paths may be equivalent. For example, in the XML tree shown in Fig. 1, the following two FDXSs are equivalent if they are exist:

sale_network/pos/publication: details $\rightarrow$ *details/isbn*

sale_network/pos/publication/details: self $\rightarrow$ *isbn*

It is easy to prove that they are equivalent because there is only one child node with label *details* for each *publication* element in the XML tree.

Definition 4.2. *(Equivalent prefix path) Let q be a path with length n(1<n), if there is only one path instance <u_1, u_2, ..., u_n> in the result for paths(root, q) for each path instance <v_1,v_2, ..., v_m>(m<n) in the result for paths(root, p), and $v_i=u_i$holds for 0<i<m+1, then we call path p the equivalent prefix path of path q.*

In general, the equivalent prefix paths of a path can be found according to a given DTD, XML Schema or regular expression types for XML. For example, we can find that path *sale_network/pos/publication* is an equivalent prefix path of path *sale_network/pos/publication/details*, and no equivalent prefix path exists for path *sale_network/pos/publication* itself. Path *self* may be an equivalent prefix path for any path, but it does not have any equivalent prefix path.

The equivalence of the FDXSs with different context paths can be expressed as the two rewriting rules as follows:

Rule X8. *Let σ be a FDXS in the form of s_1:$p_1\rightarrow q_1$ where p_1, q_1 are paths, if path s_2 is an equivalent prefix path of path s_1 and $|s_2|=|s_1|$-1, then σ can be rewritten as s_2: $p_1'\rightarrow q_1'$ where $p_1'=down(x, p_1)$, $q_1'=down(x, q_1)$ and x is the last label of s_1. The function down is used to transfer a path related to s_1 to one related to its equivalent prefix path s_2, and its definition is as follows:*

$$down(x, p) = \begin{cases} x & \text{if } p = self \\ x/p & \text{otherwise} \end{cases}$$

Rule X9. *Let σ be a FDXS in the form of s_1: $p_1\rightarrow q_1$ where p_1, q_1 are paths except **self**, if path s_1 is an equivalent prefix path of s_2 and, $|s_1|=|s_2|$-1, then σ can be rewritten as s_2:$p_1'\rightarrow q_1'$ if $p_1=x/p_1'$, $q_1= x/q_1'$ where x is the last label of path s_2.*

According to the definition of the equivalent prefix path above, the correctness of Rule X8 and Rule X9 are straightforward.

D. Soundnesss of FDXS system

The FDXS inference rules and FDXS rewriting rules above have formed an inference system, which is called FDXS system. In this section, we prove that the FDXS system is sound.

Definition 4.3. *Let Σ be a set of FDXSs and σ be another FDXS, we call σ derivable from Σ , denoted by Σ├ σ, if there is a finite sequence of FDXSs <σ_1, σ_2, ..., σ_n>,σ=σ_n and σ_i (1≤i≤n) satisfies any one of the following conditions:*

1. *σ_i is a trivial FDXS.*
2. *σ_i is a member of Σ.*
3. *σ_i is produced from union of Σ, trivial FDXSs and { σ_1, σ_2, ..., σ_{i-1} } by the application of any rule in the FDXS system.*

The soundness of the FDXS system is represented by the following theorem:

Theorem 4.1. *Let Σ be a set of FDXSs and σ be another FDXS, Σ logically implies σ if σ derivable from Σ by the FDXS system, that is, if* $\Sigma \vdash \sigma$ *then* $\Sigma \models \sigma$.

Proof. Let Σ be a set of FDXSs and σ be any FDXS, if $\Sigma \vdash \sigma$, σ is the last element of the FDXS sequence in Definition 4.3. For any XML tree which satisfies every FDXS in Σ, suppose that the length of the FDXS sequence is *n*. We use inductive reasoning to prove that the tree also satisfy σ. In case of $n=1$, the only FDXS σ in the sequence holds in the XML tree since it is either an element in Σ or a trivial FDXS in Rule X1 and X2. Suppose that the first n-1 elements in the FDXS sequence are satisfied in the XML tree, there are three cases for the last element σ:

1. σ is trivial FDXS. It holds for any XML trees.
2. σ is a member of Σ. It has been satisfied by the XML tree.
3. σ is produced from them by applying one of the FDXS inference rules and the FDXS rewriting rules except X1 and X2. Since these rules have been proved to be sound and their premises come from first n-1 elements, trivial FDXSs or Σ, the XML tree satisfies σ too.

As a result, σ holds for the XML trees which satisfy every FDXS in Σ. Thus, $\Sigma \models \sigma$ is proved to be sound.

E. Constructing a Counterexample XML Tree

In this section, we introduce the definitions of XML path closures and an algorithm for constructing an XML tree, which will be used as a counterexample for proving the completeness of the FDXS system.

Definition 4.4. *Given a FDXS set Σ, let σ be a FDXS in the form of* $s: X \rightarrow A$, *the XML value path closure* $X^+(s)$ *in Σ for FDXS σ is defined as following:*

$X^+(s) = \{ p \mid \Sigma \vdash s: X \rightarrow p \}$

Lemma 4.1. *Let σ be a FDXS in the form of* $s:X \rightarrow A$, *Σ a given FDXS set,* $\Sigma \vdash \sigma$ *if and only if* $A \subseteq X^+(s)$.

Proof. Let $A=\{a_1, ..., a_n\}$ where a_i is a path for $i=1, ..., n$. First, we prove it is necessary. According to the definition of $X^+(s)$, $a_i \in X^+(s)$ $(i=1, ..., n)$. It leads to $A \subseteq X^+(s)$. Next, we prove its sufficiency. According to Definition 4.4, $A \subseteq X^+(s)$ means that $\Sigma \vdash s: X \rightarrow a_i (i=1, ..., n)$. It leads to $\Sigma \vdash s: X \rightarrow A$ by applying union rule X5 to them. The sufficiency of Lemma 4.1 is proven.

For example, the following FDXS holds for the XML tree shown in Fig. 1:

sale_network/pos/publication: num → details

For a FDXS set with the single FDXS, according to Lemma 4.1, by applying the FDXS inference rules and FDXS rewriting rules, we can get its XML value path closure as follows:

X^+(*sale_network/pos/publication*) =
{ *num, details* } ∪ { *p* | *details is prefix path of p,*
p ∈ *U(sale_network/pos/publication)* }

It should be noted that Rule X5 and X7 have been used in th proof of Lemma 4.1 and they are proven to be sound by Rule X2, X3 and X4. It has covered every trivial FDXSs in the FDXS systems.

Algorithm 4.1 Given a FDXS set Σ, let σ be a FDXS in the form of $s:X \rightarrow A$, a counterexample XML tree *Tr* can be constructed by their $X^+(s)$. The algorithm is as follows:

1. For the context path *s* in the form of $s_1/s_2/.../s_n$, we construct an XML tree with a root node labeled with *r* and its two child nodes labeled with s_1. Then, we create one child node labeled with s_i under the s_{i-1} nodes for $i=2, ..., n$ respectively.
2. For every path in the form of $t_1/t_2/.../t_m$ in $X^+(s)$, we create a child node labeled with t_1 under every existing s_n node, and create one child node labeled with t_i under the t_{i-1} nodes for $i=2, ..., m$ respectively, if no child node with the same label exists. The same integer is assigned to the content of every t_m node. No node is created for *self* in $X^+(s)$.
3. Let Tr' be the XML tree constructed by the two steps above, for every path in the form of $t_1/t_2/.../t_m$ in $E(s,Tr')$-$X^+(s)$, we create a child node labeled with t_1 for every node s_n and create one child node labeled with t_i under the t_{i-1} nodes for $i=2, ..., m$ respectively, if no child node with the same label exists. A text node with distinct integer is created as the child node of each t_m node. For *self*, a text node with distinct integer is created as its child node of every node s_n.

In the three steps above, whenever we are about to create a child for any element node, the existing child with the same label should be used instead of creating a new child node. As a result, all child nodes of each element node except the root node should be labeled with the distinct labels. For the XML tree *Tr* constructed by the algorithm above, there is the following lemma.

Lemma 4.2 Given a FDXS set Σ, let σ be a FDXS in the form of $s:X \rightarrow A$, the following propositions are true in the XML tree *Tr*:

1. Any two distinct nodes reachable via the same path in $X^+(s)$ are value equal.
2. Any two distinct nodes reachable via the same path in $E(s,Tr')$-$X^+(s)$ are not value equal.

From step 2 and step 3 in the algorithm above, the correctness of Lemma 4.2 is straightforward.

For example, assumed that $a:b/c \rightarrow e/x$ is only FDXS in a given FDXS set Σ, we can get the following XML value path closure:

$X^+(a) = \{ b/c, e/x \}$

Following Algorithim 4.1, for path *b/c* in $X^+(a)$, two *b* node and two *c* node are created with same content, for path *e/x* in $X^+(a)$, two *e* node and two *x* node are created with same content. After step 1 and step 2, the XML tree is shown in Fig.2(a). In step 3, since paths *b* and *e* are in $E(a,Tr')$-$X^+(a)$, two text node with different integers are created as the child nodes of *b* nodes, and two text nodes with different integers are created as the child nodes of *e* nodes, as shown in Fig.2(b). Two text nodes with different integers are created since path *self* in $E(a,Tr')$-$X^+(a)$. According to Definition 2.7, the effective path set for the XML tree is as follows:

E(a, Tr) = { self, b, b/c, e, e/x }

The final XML tree *Tr* is shown in Fig.2(c).

Table 2 show the reachable nodes for every path in *E(a,Tr)*. It should be noted that the nodes reachable via any path in $X^+(a)$ are value equal, such as the nodes reachable via *e/x* have the same value. And the nodes reachable via any path in $E(a, Tr)\text{-}X^+(a)$ are not value equal, such as two *e* nodes have different values.

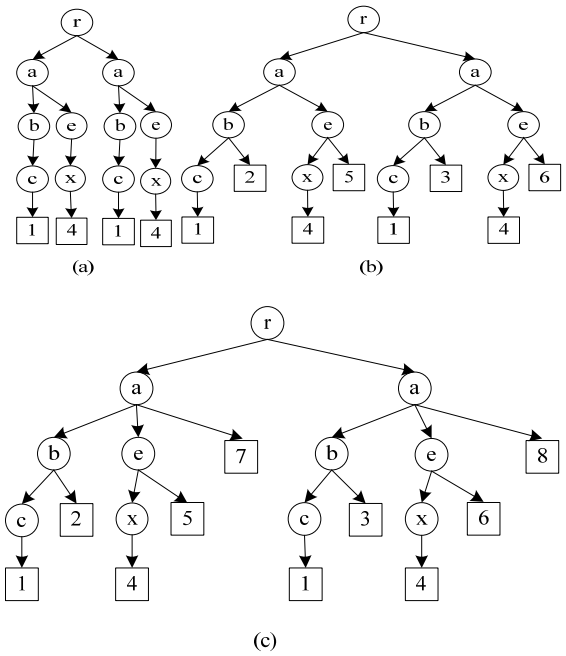


Figure 2. A counterexample XML tree.

F. Completeness of the FDXS System

The completeness of the FDXS system is represented as the following theoroem.

Theorem 4.2. *For any FDXS σ and any FDXS set Σ, if Σ logically implies σ, then σ can be derived from Σ by the FDXS system, that is, if* $\Sigma \models \sigma$ *then* $\Sigma \vdash \sigma$.

According to Definition 4.3, since all trivial FDXSs are derivable from *Σ*, only non-trivial FDXS should be take into account in the following proof.

Proof. Let *Σ* be a FDXS set, according to the reduction to absurdity, suppose that *σ* is an arbitrary FDXS which cannot be derived from *Σ* by the FDXS system and $\Sigma \models \sigma$, we will show that there is an XML tree such that FDXS *σ* does not hold and every FDXS in *Σ* holds in the XML tree. Following Algorithm 4.1, from *Σ* and FDXS *σ,* an XML tree *Tr* is built as the counterexample which is used to prove 1) the XML tree does not satisfy FDXS *σ*, and 2) the XML tree satisfies every FDXS in *Σ* . It means that *Σ* does not logically imply *σ*.

1) To prove that *Tr* does not satisfy FDXS *σ* in the assumption above, consider *σ* is in the form of $s{:}X \rightarrow K$. Since *σ* cannot be derived from *Σ* by the FDXS system, then *K* is not a subset of its $X^+(s)$ according to Lemma 4.1. Thus for XML tree *Tr*, some paths in *K* must belong to $E(s,Tr)\text{-}X^+(s)$. According to Lemma 4.2, the nodes in *Tr* reachable via the paths in $E(s,Tr)\text{-}X^+(s)$ have different values, including the nodes reachable via those paths in *K*, but the reachable nodes via every path in *X* are value equal, since $X \subseteq X^+(s)$. Hence, *Tr* violates $s{:}X \rightarrow K$.

TABLE 2.
THE REACHABLE NODES VIA EVERY PATH IN E(a,Tr).

path set	*path*	*reachable node sets*
$X^+(a)$	*b/c*	*{ <c>1</c> }*
	e/x	*{ <x>4</x> }*
$E(a,Tr)\text{-}X^+(a)$	*b*	*{ <c>1</c>2 }* *{ <c>1</c>3 }*
	e	*{ <e><x>4</x>5</e> }* *{ <e><x>4</x>6</e> }*
	self	*{ <a>.....7}* *{ <a>.....8}*

2) To prove that XML tree *Tr* satisfies every FDXS in *Σ*, considering every FDXS in *Σ* in the form of $s'{:}A \rightarrow C$, if s' is the same as *s*, there are three cases:
   a) According to Lemma 4.1, if $A \subseteq X^+(s)$, then $\Sigma \vdash s'{:}X \rightarrow A$, where $X^+(s)$ comes from FDXS *σ* in the assumption above. Following Rule X4, $s'{:}X \rightarrow C$. Thus, we have $C \subseteq X^+(s)$. According to Lemma 4.2, all reachable nodes via each path in $X^+(s)$ are value equal in *Tr*. Since both *A* and *C* are subsets of $X^+(s)$, $s'{:}A \rightarrow C$ is tenable.
   b) If *A* is not a subset of $X^+(s)$, then *A* must include some paths in $E(s,Tr)\text{-}X^+(s)$. According to Lemma 4.2, the reachable nodes via any path in $E(s,Tr)\text{-}X^+(s)$ are not value equal in *Tr*. Therefore $s'{:}A \rightarrow C$ holds for any path sets *C*.

Now, we have proven that the XML tree satisfies every FDXS in *Σ* whose context path is the same as *σ*.

If s' is different from *s*, its reachable nodes must be in XML tree *Tr*. Let *root* be the root node of *Tr*, there are the three cases for the context path s' as follows:

a) For every path s' whose reachable nodes except *root* are created at step 1 of Algorithm 4.1, s' must be an equivalent prefix paths of *s*, since there is only one child node for each of them. Hence, $s'{:}A \rightarrow C$ can be rewritten into a FDXS with the context path *s* by Rule 14 so that it is satisfied by *Tr*. If s'=*self*, $s'{:}A \rightarrow C$ holds for any *A* and *C*, since only *root* is a reachable node via *self*.
b) For every path s' whose reachable nodes are created at step 2 and step 3 of Algorithm 4.1, *s* must be an equivalent prefix path of s' according Definiton 4.2, since all these nodes are descendent nodes of the reachable node by *s,* and every one of them, which is the last node of the path instance of s', has a different label from its sibling nodes. Hence, $s'{:}A \rightarrow C$ can be rewritten into a FDXS with the context path *s* by Rule 13 so that it is also satisfied by XML tree *Tr*.

To sum up the descriptions above, XML tree *Tr* violates *σ* and satisfies every FDXS in *Σ*. It contradicts with the original assumption $\Sigma \models \sigma$. It means that $\Sigma \models \sigma$

does not hold if σ is not derivable from Σ for the XML tree. Therefore, for any FDXS σ, it is derivable from Σ by the FDXS system as long as $\Sigma \models \sigma$. Thus the completeness of the FDXS system is proven.

It should be noted that Lemma 4.1, Rule X7 and X8 are used in the proof. Since all non-trivial FDXSs has been used in the proof of Lemma 4.1. The completeness proof has covered every non-trivial FDXSs and FDXS rewriting rules in the FDXS system.

V. Related Work

Functional dependency has been the key concept for the normalization theory used in the database design since Codd[9] proposes the relational model. Armstrong[10] and Beeri[11] propose a set of inference rules used to derive functional dependencies and prove those axioms are sound and complete. Their works solve the implication problem of functional dependencies perfectly in the relational model. However, since the attributes of data entities must be atomic and different with each other according to the first normal form of relational model, the axioms can not be applied to the new data models such as XML document. How to resolve the problems such as integrity constraints and functional dependencies in such complex data also becomes a research hotspot.

In order to meet the needs of the exchange and sharing of information on the World Wide Web, W3C proposes semi-structured XML data model in 1998. Because XML data are expressed as a tree structure, so the dependencies on the XML data are also bound to this hierarchical structure. Those dependencies are obviously more complicated than the flat relational model dependencies, and thus there is no uniform definition for XML functional dependency so far, and not every definition is able to cover all kinds of XML function dependencies. The difference between existing definitions has a significant influence to the further studies, such as logical implication and XML normalization.

Arenas[4][19] comprehensively studies the problems on XML functional dependency and normalization. Analogy to the relational model, he puts forward the concept of "tree tuple", and using this concept we can map the XML documents to relational model. On this basis, he defines the functional dependency, describes a kind of normal form called XNF and gives an algorithm that can convert DTD to comply with the XNF. He also finds that, in some cases, there exists a sound algorithm for logic implication problem, but in other cases, that is an NP-complete problem due to the differences of DTDs. His work reveals the complexity of XML logic implication problem. In other words, the definition of XML functional dependency will have a major impact on solving the logical implication problem. Due to the diversity and complexity of XML functional dependency, to find a definition of better nature becomes one of the hotspots of academia. Kot and White[20] prove that there exists a sound and complete axiom system for the function dependents based on "tree tuple" in some of DTDs circumstances. Vincent et al. [5][21][22] give a definition based on the closest attribute value, and prove that the inference system based on such definition can be axiomatized. They also illustrates the primary key can serve as the special case of XML functional dependency in certain circumstances, although they do not consider the influence of the schema language. Shahriar [23] has given a unified representation of XML global functional dependencies and local functional dependencies. Expressions in the form of $(S, P \rightarrow Q)$ are used to express various kinds of XML functional dependencies while S represents the scope of local functional dependencies.

Unlike the means shown above, Hartmann [24] gives another definition based on "subtree", and he extends the expression of function dependencies through graphic schemas. Then he proposes a set of inference rules for functional dependencies with frequency [25], investigates XML functional dependency based tree homomorphism and finds an algorithm to solve logical implication problem based on Horn clause logic [26]. But these definitions do not cover the dependences between set of values of XML nodes as our work. In addition, its inference rules are also different with our work significantly, because of the difference for XML data representation.

Those works give various definitions of XML function dependency with different expressive power. Most of them study the logical implication problem and normal form and does not involved integrity constraints related to XML items which can appear more than once. Yu[27] extends the representation of XML functional dependency on the basis of the work of paper[4][19], making it possible to represent the generalized tree tuples and element sets. He also gives a normal form and its normalization algorithm based on his definitions. In our paper, the function dependence definition based on the path is closed to the definition in their paper since the context path in FDXS is similar to the "pivot node". Our work can cover the reasoning under the different contexts, and we focus on the integrity constraints between XML node sets reachable via different path especially.

VI. Conclusion

Various XML integrity constraints should be taken into account for the well-designed databases, in order to keep data consistency and eliminate update anomalies. We find that for XML data, the data redundancy may come from the functional dependencies between XML node sets. In order to describe this kind of XML integrity constraint, this paper study the XML functional dependency based on node sets, gives its formal definitions. Moreover, this paper studies the logical implication problem for the functional dependency based on XML node sets, proposes a set of FDXS inference rules and FDXS rewriting rules, and proves their soundness and completeness.

Acknowledgment

This work was both supported in part by the Beijing Nature Scince Fundation under Grant 4122011 and the

National Science Foundation for Young Scientists of China under Grant 61202074.

REFERENCES

[1] P. Buneman, S. Davidson, W. Fan, C. Hara and W. Tan, "Reasoning about Keys for XML," *Inform. Syst*, vol. 28, no. 8, pp. 1037-1063, December 2003.

[2] P. Buneman, W. Fan and S. Weinstein, "Path Constraints in Semistructured Database," *J. Comput. Syst. Sci*, vol 61, no 2, pp. 146-193, October 2000.

[3] W. Fan and J. Simeon, "Integrity Constraints for XML," *J. Comput. Syst. Sci*, vol. 66, no. 1, pp. 254-291, Feb. 2003.

[4] M. Arenas and L. Libkin, "A Normal Form for XML Documents," *ACM T Database Syst*, vol. 29, no. 1, pp. 195-232, March 2004

[5] M. Vincent, J. Liu and C. Liu, "Strong Functional Dependencies and their Application to Normal Forms in XML," *ACM T Database Syst*, vol. 29, no. 3, pp. 445-462, September 2004

[6] M. Vincent and J. Liu, "Multivalued Dependencies and a 4NF for XML," *Proc. Advanced information systems engineering*, pp. 14-29, 2003.

[7] K. Karlinger, M. Vincent and M. Schrefl, "Inclusion Dependencies in XML: Extending Relational Semantics," *Proc. Database and Expert System Application*, pp. 23-37, 2009.

[8] H. Hosoya, J. Vouillon and B.C. Pierce, "Regular Expression Type for XML," *ACM T Progr Lang Sys*, vol. 27, no. 1, pp. 46-90, January 2005.

[9] F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Commun ACM*, vol. 13, no. 6, pp. 377-387, June 1970.

[10] W. Armstrong, "Dependency Structures of Data Base Relationships," *Proc. IFIP'74*, pp. 580-583, 1974.

[11] C. Beeri, R. Fagin and J. Howard, "A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations," *Proc. SIGMOD'77*. pp. 47-61, 1977.

[12] M. Levene and G. Loizou, "The Nested Universal Relation Data Model," *J. Comput. Syst. Sci*, vol. 49, no. 3, :pp. 683-717, December 1994.

[13] K. Schewe and B. Thalheim, "Fundamental Concepts of Object Oriented Databases," *Acta Cybernetica*, vol. 11, no. 1, pp. 49-84, 1993.

[14] G. Gardarin, J. Cheiney, G. Kiernan, D. Pastre and H. Stora, "Managing Complex Objects in an Extensible Relational DBMS," *Proc. Very Large Data Base*, pp. 55-65, 1989.

[15] Z. Peng and Y. Kambayashi, "Deputy Mechanisms for Object-Oriented Databases," *Proc. Inter. Conf. Database Engineering*, pp. 333-340, 1995.

[16] Z. Tari, J. Stokes and S. Spaccapietra, "Object Normal Forms and Dependency Constraints for Object-oriented Schemata," *ACM T Database Syst*, vol. 22, no. 4, pp. 513-569, December 1977.

[17] P. Buneman, "Semistructured Data," *Proc. Symp. Princples of Database Systems*, pp. 117-121, 1997.

[18] P. Buneman, W. Fan and S. Weinstein, "Interaction between Path and Type Constraints," *ACM T Comput Logic*, vol. 4, no. 4, pp. 530-577, October 2003.

[19] M. Arenas and L. Libkin, "A Normal Form for XML Documents," *Proc. Symp. Principles of Database Systems*, pp. 85-96, 2002.

[20] L. Kot and W. White, "Characterization of the Interaction of XML Functional Dependencies with DTDs," *Proc. Inter. Conf. Database Engineering*, pp. 119-133, 2007.

[21] M. Vincent and J. Liu, "Functional Dependencies for XML," *Web Technology and Applications, Lect. Notes Comput. Sc*, vol. 2642, pp. 23-24, 2003.

[22] J. Liu, M. Vincent and C. Liu, "Local XML Functional Dependencies," *Proc. Fifth ACM International Workshop on Web Information and Data Management*, pp. 23-28, 2003.

[23] M. Shahriar and J. Liu, "On Defining Functional Dependency for XML," *Proc. IEEE Intern. Conf. Semantic Computing*, pp. 595-600, 2009.

[24] S. Hartmann and S. Link, "More Functional Dependencies for XML," Advances in Databases and Information Systems, Lect. Notes Comput. Sc, vol. 2798, pp. 355-369, 2003.

[25] S. Hartmann and T. Trinh, "Axiomatising functional dependencies for XML with frequencies," Fundation of Information and knowledge systems, Lect. Notes Comput. Sc, vol. 3861, pp. 159-178, 2006.

[26] S. Hartmann, S. Link and T. Trinh, "Solving the Implication Problem for XML Functional Dependencies with Properties," Logic Language Information and Computation, Lect. Notes Comput. SC, vol. 6188, pp. 161-175, 2010.

[27] C. Yu and H.V. Jagadish, "XML Schema Refinement through Redundancy Detection and Normalization," *VLDB Journal*, vol. 17, no. 2, pp. 203-223, March 2008.

[28] Husheng Liao, Weifeng Shan and Hongyu Gao, "Automatic Parallelization of XQuery Programs," *Journal of Software*, vol. 8, no. 4, April 2013.

[29] Xiaojie Yuan, Xiangyu Hu, Dongxing Wu, Haiwei Zhang and Xin Lian, "XML Data Storage and Query Optimization in Relational Database by XPath Processing Model," *Journal of Software*, vol. 8, no. 4, April 2013.

[30] Jianwei Wang and Zhongxiao Hao, "Research on Basic Operations for Query Probabilistic XML Document Based on Path Set," *Journal of Software*, vol. 8, no. 4, April 2013.

Husheng Liao is a professor of computer science at Beijing University of Technology in P.R.China. He received his M.S. in 1981 from Tsinghua University. From 1981 until 1993, he was lecturer and associate professor at Beijing Computer Institute.

He leads the software research group to perform cross-area research in programming languages and database systems. His current interests include XML technology, query languages, program transformation and program analysis.

Prof. Liao is a member of CCF.

Jia Wu is a M.S student in the Colledge of Computer Science at the Beijing University of Technology in P.R.China.

Her current interests include database theory and XML technology.

Jia Liu is a Ph.D. student in the Colledge of Computer Science at the Beijing University of Technology in P.R.China.

His current interests include database theory, XML technology and query languages.