2009 Ninth International Conference on Hybrid Intelligent Systems

XML Functional Dependency and Schema Normalization

Xiangguo Zhao Junchang Xin Ende Zhang
Key Laboratory of Medical Image Computing (Northeastern University), Ministry of Education
College of Information Science & Engineering, Northeastern University, P. R. China
zhaoxiangguo@mail.neu.edu.cn

Abstract

Normalization theory is necessary for designing a good XML document. A good document means that it has minimal redundancy. The research on XML functional dependency and normalization is still an open problem. This paper first presents a path language of XML model, then proposes a new kind of XML functional dependency (XFD) that has stronger expression ability to XML functional dependency that can result in redundancies. In this paper, the problems of XFD logical implication and XFD closure are studied and a group of the corresponding inference rules is proposed. Based on the XFD, a kind of XML normal form (XNF) and an algorithm converting into XNF are also proposed in this paper.

1. Introduction

XML has been a de facto standard of data representation and exchange on the Web. More and more information has been managed and stored in the form of XML documents in many research prototypes and application systems of XML databases. Function dependency and normalization play an important role in the relational database, which are used for integrity constraints and database design. Similarly, the similar application will contribute greatly to XML data management. XML data constraints all come from application semantics, and we hope to customize these semantic constraints in the XML data schema so that the database can avoid those operations updating the database which may cause the unconformity state. These abnormalities are brought from data redundancy in the database while the data redundancy has direct relationship with the rule of data dependency.

The theory of function dependency and normalization springs from relational database. After Provost proposed the thought of applying the theories of relational database on XML database schema standardization design [8], many researchers have worked on such topics. Areanas etc. proposed the conception of XML function dependency (XFD) based on the conception of tree tuple, defined XML schema normal form (XNF) and gave an algorithm about switching DTD to XNF [1]. Tok etc. defined semi-structured normal form and gave an algorithm about converting semi-structured model into that satisfied some normal form [16]. Liu etc. defined XFD, XMVD [9, 12–14] and relative normal form based XML intermediate attributes FD and MVD. And some literatures researched XML Key [2, 3], XML functional dependency [4–6, 10, 11, 15] and XML normalization [1, 7, 16]. Whatever, the above research has not solved the problems of XML function dependency and XML standardization perfectly. For example, in Figure 1, the problem of redundancy cannot be resolved in a formalized way by these methods. We hope to get a representation method of XML function dependency which has more powerful expression, and then normalize the XML documents on this basis.

The research about XML functional dependency and XML document normalization on the basis of XFD is still an open problem. Some research achievements have been proposed to solve these problems, but these works still have insufficiencies. Therefore, we consulted a large number of XML documents to explain problems and used DTD as the model file which is widely used in the practice. Take the following as examples:

In Figure 1, it is a DTD of the XML documents about some school administrative activities. The school has many common courses and departments. Every department includes department name, a set of courses, some classrooms and offices. Every classroom has classroom number and building number of its located building. Every office has office number, office name and building number of the building it located. Every course has course number and a student list which chose these courses. The student list contains a set of students, every student's number and name.

There are constraints information in the example DTD as follows.

a) In the total document, the student number determines student name;

978-0-7695-3745-0/09 $25.00 © 2009 IEEE
DOI 10.1109/HIS.2009.276

```
<!ELEMENT school (course*,department*)>
<!ELEMENT course (slist*)>
    <!ATTLIST course cname CDATA #REQUIRED>
<!ELEMENT department (course*,office*,classroom*)>
    <!ATTLIST department dname CDATA #REQUIRED>
<!ELEMENT office (oname,addr)>
    <!ATTLIST office roomno CDATA #REQUIRED>
<!ELEMENT classroom (addr )>
    <!ATTLIST classroom roomno CDATA #REQUIRED>
<!ELEMENT slist (student*)>
<!ELEMENT student (sname£ ‹grade)>
<!ATTLIST student sno CDATA #REQUIRED>
```

Figure 1. The DTD about the school

b) All in the school , course name determines course and the courses are not repeated;

c) In some department, the office name can determine the room number which is invalid in the total document;

d) The department name determines the addresses of the offices and classrooms (supposed that every department locates in the certain building).

We can find that there is redundancy in the XML document satisfied this DTD. A student may take several courses, so the students' names are stored repeatedly in the courses selected. For some department, whatever the office or the classroom, their building numbers are stored repeatedly. The redundancy in the database will cause the operation abnormality, so the redundancy should be avoided in the schema design and which needs XML documents standardization theories. In Figure 1, the redundancy of XML documents comes from the dependency relationship among data, that is, XML function dependency. Proceeding with the representation of XML function dependency, we discuss how to standardize XML documents in a formative way, and then gradually eliminate data redundancy caused by XML function dependency.

2. Symbol Definition

2.1. DTD and XML Document Tree

Some definitions of DTD and XML are given in the following.

Definition 1. ***DTD*** *is defined as a 6-tuple:* $D = (E_c, E_s, A, F, R, r)$*, where*

1) E_c *is the finite set of complex elements;*

2) E_s *is the finite set of simple elements;*

3) A *is finite set of attribute names;*

4) F *is the map from* E_c *to element type:* $\forall e \in E_c$*,* $F(e)$ *is a regular expression*

$$F(e) = \varepsilon | e' | F(e)\,|F(e)\ |F(e), F(e)|F(e)^*$$

where ε *denotes null string,* $e' \in E_c \cup E_s$*, and "*$|$*", ",", "*$*$*" denote union, connection and Kleene closure;*

5) R *denotes the map from* E_c *to the attribute set; for* $e \in E_c$*, any* $F(e)$ *not* ε *or* $R(e)$ *not* $\emptyset$*;*

6) $r \in E_c$ *is another different element name,*$\forall e \in E_c$*,* r *is the only flag in* E_c *which is not in the alphabet of* $F(e)$*.*

Definition 2. ***XML Tree*** *is usually defined as the following type* $T = (V, lab, ele, att, val, tex, root)$*, where*

1) V *is the node set of XML tree;*

2) Lab is the map from V *to* $E_c \cup E_s \cup A$*, it can get the node type in* V*; if* $lab(v) \in E_c$*, then a node* $v \in V$ *is called a complex element node; if , then it is called a simple element node; if* $lab(v) \in E_s$*, then it is called attribute node.*

3) ele and att are the functions defined on the set of complex elements: for every $v \in V$*, if* $lab(v) \in E_c$ *then* $ele(v)$ *denotes a list of element nodes and* $att(v)$ *denotes a set of different attribute names;*

4) val *denotes the function of node number from every attribute or element;*

5) tex *denotes the function of string value from every attribute or element, if* $lab(v) \in E_c$*,* $tex(v)$ *is null, and if* $lab(v) \in E_s \cup A$*,* $tex(v)$ *is the document contained;*

6) $root$ *denotes the only root node;*

The XML document of the school information can be defined in Figure 2 according to the definition, and it is an XML tree satisfies to the DTD in Figure 1.

Definition 3. ***Node equal and value equal*** *Nodes* v_1 *and* v_2 *are called value equal if and only if the values of the two nodes are equal, denoted as* $v_1 =_v v_2$*. Similarly,* v_1 *and* v_2 *are called node equal if and only if* v_1 *and* v_2 *are the same node in the XML tree, denoted as* $v_1 = v_2$*. Two node equal nodes* v_1 *and* v_2 *satisfy the following properties.*

1) $lab(v_1) = lab(v_2)$*,* v_1 *and* v_2 *are both simple elements or attributes and* $tex(v_1) = tex(v_2)$*, or*

2) v_1 *and* v_2 *are both complex elements and (1) the sequence of their children elements is equal in pairs; (2) for every attribute* α *of* v_1*, there is a attribute* β *in* v_2 *satisfies* $\alpha = \beta$*, and vice versa.*

2.2. Path Language

Definition 4. ***Simple path*** *For a given DTD,* $D = (E_c, E_s, A, F, R, r)$*, a simple path is a path with the form of* $l_1.\cdots.l_m$*, where* $l_i \in E_c$*,* $l_m \in E_c \cup E_s \cup A$ *(*$i < m$*), is called a path step.*

Definition 5. ***Complex path*** *A complex path expression is an expression with the form of* $l_1.\cdots.l_n$*, where* $l_i \in E_c \cup \{/, //\}, i \in [1, n-1]$*,* $l_n \in E_c \cup E_s \cup A$*, and there exists at least a* l_i *satisfies* $l_i \in \{/, //, \}$*. The symbol* $/$ *denotes a wildcard that means it can match any symbol in the next step of the path. The symbol* $//$ *denotes the closure of wildcards.*

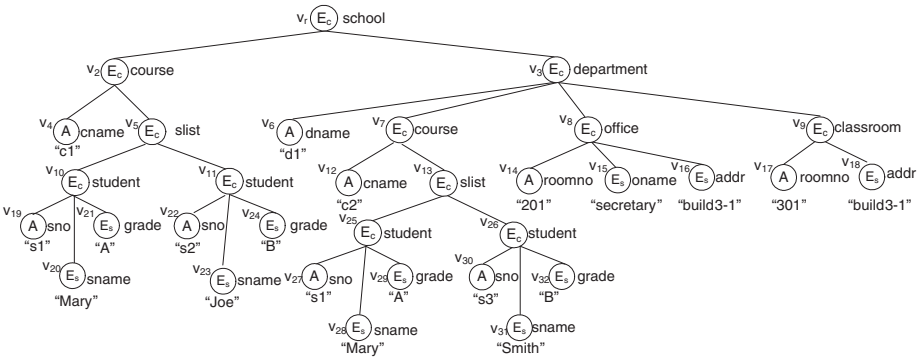

Figure 2. An XML document conforming the DTD in Figure 1

Definition 6. ***Matching path*** *If $l_1.\cdots.l_n \in Cpaths(D)$, where l_i is /, $l'_i \in E_c \cup E_s \cup A$, $l_1.\cdots.l_{i-1}.l'_i.l_{i+1}.\cdots.l_n$ is a simple path in D, then $l_1.\cdots.l_{i-1}.l'_i.l_{i+1}.\cdots.l_n$ matches with $l_1.\cdots.l_n$; if $l_1.\cdots.l_n \in Cpaths(D)$ where l_i is //, P is a simple path in D and $l_1.\cdots.l_{i-1}.P.l_{i+1}.\cdots.l_n$ is a simple path in D, then $l_1.\cdots.l_{i-1}.P.l_{i+1}.\cdots.l_n$ matches with $l_1.\cdots.l_n$. The simple path matched with complex path q is called q's matching path. All the simple paths matched with complex path q are q's matching path set denoting $\{q\}$.*

Definition 7. ***Path Instance*** *T is an XML tree satisfies D, and v_0 is a node of T, the path P in D is $l_1.\cdots.l_n$. If the node sequence $v_0.v_1.\cdots.v_n$ $i \in [1,n]$ in T satisfies one of the following conditions: 1) if $l_i \in E_c \cup E_s \cup A \cup \{/\}$, v_i is the child of v_{i-1}, and it is denoted with flag l_i, and any node can match with the symbol / of wildcard, and 2) if l_i is //, ω denotes any path instance in the sub tree determined by v_0; if there exists a path instance $v_1.\cdots.v_{i-1}.\omega.v_{i+1}.\cdots.v_n$, where $v_1.\cdots.v_{i-1}$ matching with $l_1.\cdots.l_{i-1}$ and $v_{i+1}.\cdots.v_n$ matching with $l_{i+1}.\cdots.l_n$, then $v_1.\cdots.v_n$ is called a path instance of P relative to v_0.*

Definition 8. ***Target set*** *We use $v_0[p]$ denoting the set $\{v|v = last(the\ path\ instance\ of\ P\ relative\ to\ v_0)\}$, and get the node set of path P relative to node v_0 that is called the target set of P relative to v_0.*

3. XML Function Dependency

3.1. Absolute Function Dependency

Definition 9. ***Absolute Function Dependency*** *For a given DTD $D = (E_c, E_s, A, F, R, r)$, an absolute function dependency φ on D with the form of P, $(Q1, Q2, Qn \rightarrow Qn+1(S1, Sm))$, where*

1) P ,$Q_i(i \in [1, n+1])$, and $S_j(j \in [1, m])$ are paths on D, and $P.Q_i$, $Q_{n+1}.S_j$, $P.Q_{n+1}$ are also the paths on D;

2) P is a path containing the root node, $P \in paths(D)$ is called the target path of the function dependency, and $P \neq \varepsilon$, $last(P) \in E_c$ denoting the target node set of the establishment of functional dependency.

3) $Q_1, Q_2, \cdots Q_n$ is called φ's left path, and $Q_i(i \in [1, n])$ is the simple path on D, $last(Q_i) \in E_c \cup E_s \cup A$ and at least one of $Q_1, Q_2, \cdots Q_n$ is not ε, denoting the determinants of function dependency;

4) Q_{n+1} is called φ's right target path, and $Q_{i+1} \in paths(D)$, $last(Qi) \in E_c \cup E_s \cup A$, denoting the range of the right of function dependency;

5) $S_1, \cdots S_m$ are called right paths and they are simple paths denoting the determined factors in XML tree. If $S_1, \cdots S_m$ are all null path ε, then $last(Q_{n+1})$ can be the determined factor. If Q_{n+1} and $S_1, \cdots S_m$ are both null path ε, $last(P)$ can be the determined factor.

Applying the definition of absolute XFD, the four kinds of data constraints showing in a), b), c), e) of example 1 can be represented the functional dependency in the following forms.

a) $//.student, (@sno(sname))$;

b) $//.course, (@cnamev())$;

d) $school.department, (@dname/(addr))$.

Pay attention to the particular meaning of the last form of functional dependency. If functional dependency e) is replaced with another form;

a) $school.department, (@dname/addr())$.

Their semantic are different, / in e) is right objective path, denoting the fifth constraint in example 1. If two departments have the same name, they have the same building number, as figure 3 shows that XML document satisfies constraint e). But the document showing in Figure 3 doesn't satisfy this constraint. The right objective path in f) is $/.addr$ and the right path null means its semantic is that if two departments have the same names then they have the same building number set of their rooms, which are all {build3-1,build4-2}. The document in figure 4 satisfies the

constraint f). If the value of d_2 of v_7 is changed to d_1, then constraint f) is not satisfied.

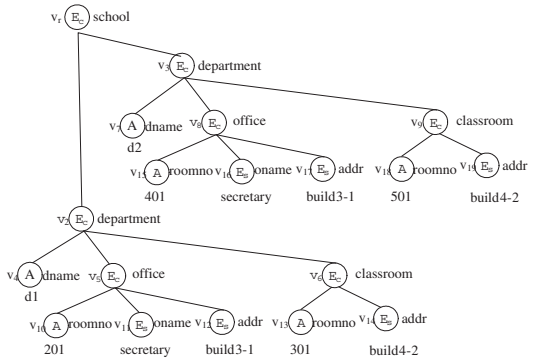

Figure 3. Part of XML document of school academic information

3.2. Relative Functional Dependency

Definition 10. ***Relative Functional Dependency*** *Given DTD $D = (E_c, E_s, A, F, R, r)$, the relative functional dependency φ on D is in the form of $O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1}(S_1, \cdots S_m)))$, where*

1) O is the path on D, and $O.P$ is also the path of D.

2) O is called context path, indicating the range of XFD tenable, and O is root path, $last(O) \in E_c$.

3) $P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1}(S_1, \cdots S_m))$is absolute functional dependency defined on the context of O.

Applying the definition of relative XFD, the constraint c) in example 1 can be represented functional dependency as $school.department, (office, (oname@roomno))$.

Absolute functional dependency is the particular case compared with relative functional dependency, and it works in the entire document, but the relative functional dependency works on the segment document. When the context path is null, relative functional dependency is just absolute one. So we can use relative functional dependency as the usual form.

XML Key [9] is a particular case of proposed XFD. Functional dependency b) expresses the semantic of the key, and $@cname$ as the key of course guarantees the integrity of the entity, which is the conception of absolute key in [9]. At the same time, it can be described the relative key. As on the document segments designated by nodes with department type in example1, @roomno can determine a node of some office type, that is the room number of some building can determine the office without repetition. This semantic is represented as the functional dependency $school.department, (office, (@roomno \rightarrow_v \varepsilon))$.

Without loss of generality, we can disconnect right part of the functional dependency $\varphi O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1}(S_1, \cdots S_m)))$ on D and write as m functional dependencies $O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1}(S_i)))$, $i \in [1, m]$. So in the following text we always use single path.

4. Axiom System of XFD

Definition 11. ***Logical implication*** *Given a DTD $D = (E_c, E_s, A, F, R, r)$ and the functional dependency set $\sum$ on D, if any document satisfied D satisfy $\sum$, it must satisfy the functional dependency $\varphi O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1}(S)))$, it is called that $\sum$ implicates logically φ, denoting $\sum \Rightarrow \varphi$.*

As in the example 1, the document also satisfies XFD: $//course, (@cno(slist.student.@sno))$.

If given DTD, the functional dependency of D is null set $\emptyset$, the functional dependency logically implicated by $(D, \emptyset)$ is called ordinary functional dependency.

For any XML document satisfied D, if they always satisfy $\sum$, then they satisfy other XFD automatically. The closure of $\sum(\sum^+)$is a enlarged set, including $\sum$ and subsidiaries satisfied $\sum$.

Definition 12. ***The closure of*** $\sum(\sum^+)$ *The set of all the functional dependencies logically implicated by $\sum$. It is represented using symbols: $\sum^+ = \{\varphi | \sum \Rightarrow \varphi\}$.*

In order to solve the problem of logical implication and closure, a group of inference rules are needed to deduce other XFD from known XFD.

For a given DTD, the functional dependency $\varphi O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1}(S)))$ tenable on $D = (E_c, E_s, A, F, R, r)$ and D, there are following rules:

1) Reflexive rule. $O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow (Q_i)))$is tenable, $i \in (1, n)$

2) Transmissible rule. If $O, (P, (Q_1, Q-2, \cdots Q-n \rightarrow Q_{n+1}))O, (P, (Q_{n+1} \rightarrow R)), O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow R))$ is tenable.

3) Enlargement rule. If $O.P.T$ is the path on D, then $O, (P, (Q_1, Q_2, \cdots Q_n, T \rightarrow Q_{n+1}(S), (T))))$ is tenable;

4) Objective path upward sliding. If $P = R.T$ and T is the only path of R, then $O, (R, (T.Q_1, T.Q_2, \cdots T.Q_n \rightarrow T.Q_{n+1}(S))$is tenable;

5) Objective path downward sliding. If T is the common prefix of $Q_1, Q_2, \cdots Q_n, Q_{n+1}$, that is $Qi = T.Qi'$, ,and T is the only path of P, then $O, (P.T, (Q'_1, Q'_2, \cdots Q'_n Q'_{n+1}(S)))$ is tenable. Similarly, the right objective path can also slide upward and downward according to the rules 4) and 5);

6) Path replacement. The complex path T in XFD can be replaced with the matching path of $\{T\}$, and the functional dependency is tenable. For φ given in above text, if P is a complex, $R \in \{P\}$, then $O, (R, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1}(S)))$ is tenable;

7) Expansion of left paths. If there exists a path $O.P.Qi.T, i \in (1,n)$ on D and T is the only path of Q_i, $O,(P,(Q_1,\cdots Q_i.T,\cdots Q_n \rightarrow Q_{n+1}(S)))$is tenable;

8) Expansion of right paths. If there exists a path $O.P.Q_{n+1}.S.T$, T is the only path of S, then $O,(P,(Q_1,\cdots Q_i,\cdots Q_n \rightarrow Q_{n+1}(S.T)))$ is tenable.

Theorem 1. *Inference rules 1)-8) are correct and completed.*

Beginning with XFD set $\sum$ initialized from a application semantic, using the inference rules, XFD of $\sum^+$ can be deduced. Considering a fixed left paths set $X \subset paths(D)$ from some type P of node, according to reflexive rule, $O,(P,(X \rightarrow X)) \in \sum^+$, begin(beginning) from here and produce XFD using the inference rules repeatedly so that the left paths can be increased into the right paths of XFD. Using$(X;\sum)$the maximum of right paths can be obtained, that is: $(X;\sum) = \{q|O,(P,(X \rightarrow q))$ can be deduced by the rules. $(X;\sum)$ has many properties as follows

$O,(P,(X \rightarrow (X;\sum)))\sum^+$;

$(X;\sum$ can be deduced from $\sum$ application rules, and $X;\sum$ is maximum right of any XFD whose left is X;

$X \in (X;\sum)$;

$Y \in (X;\sum)$only and if only when $\sum$ application rules can deduce $O,(P,(X \rightarrow Y))$.

5. An XML Normal Form

Definition 13. *XML normal form(XNF). Given a DTD D and the functional dependency set $\sum$, $(D,\sum)$ satisfies XML normal form with premise that the following conditions are tenable:*

1) $\sum^+$ doesn't contain those functional dependency of right objective path without null;

2) Any unordinary functional dependency of $\sum^+ O,(P,(Q_1,Q_2,\cdots Q_n \rightarrow (S)))$, $Q_1,Q_2,\cdots Q_n$ is the key.

The XML normal forms satisfied these conditions can eliminate the redundancy brought by XML functional dependency. In the example 1 in the preview, we explain why the document design is not good through XNF.

Considering the constraints given in example 1 and XML documents given from figure 3, we know this document satisfied XFD a), XFD b), XFD c)and XFD d) these functional dependencies. XFD d) requires that the building number of the rooms must be the same if the offices and classrooms of some department are in the fixed building.

In the document of figure 3, build3-1 are stored twice in the node of v_{16} and v_{18}, introducing redundancies, because this design doesn't satisfy functional dependencies XFD d) does not satisfy the first condition of XNF. In order to improve this flaw design, we move the element of addr to be the child of department as it shows in figure 5, such design satisfies both XFD dand the first condition of XNF.

XFD a) requires every student must have the same name, thus the names are the same when the same student numbers appear. In the document of figure 3, the name Mary of S1 is stored repeatedly in the node v_{20} and v_{28} results of redundancies. But this design doesn't satisfy XNF for it satisfy functional dependency XFD a), but @sno is not the key of student that is not satisfy $//.student,(@sno \rightarrow_v (\varepsilon))$, which is not constant with the second condition of XNF. As we know XFD a)is an absolute functional dependency, and the similar problems will be introduced for relative functional dependency. To rectify this imperfect design, the student information and course information will be stored separately. The single element sinfo is established for student information, we modify DTD in example 1 into which shows in Figure 4. Thus the design satisfy both XFD aand the second condition of XNF.

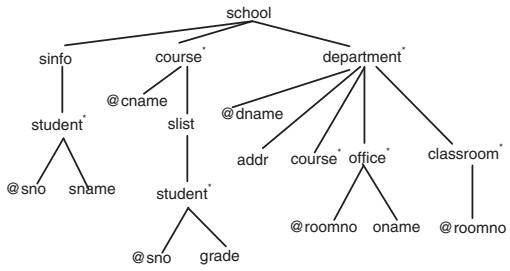

Figure 4. Modified DTD

6. Generating XNF Algorithm

In view of the problems of XML document not satisfying XNF, we need to develop a new method to convert any given $(D,\sum)into(D,\sum)$satisfying XNF. Supposing DTD is not recursive, the similar methods can also resolve the recursive problems. If the functional dependency violates the requirements of XNF, they are called abnormal functional dependency. And those which violate the condition 1 of XNF are called the first XFD abnormality.

To confirm the abnormal XFD and convert XFD into the one satisfying XNF, we need the definition system of XML keys and the methods of determining the keys according to functional dependency. But we will do no deep research; just give several definitions and assumptions. Supposing the key of given DTD has been defined, for example it can be given from the attribute ID of DTD standardization. We use such method in [5] to give the constraints of absolute key and relative key on known DTD, thus candidate keys can be reduced from functional dependency.

Given $O,(P,(Q_1,Q_2,\cdots Q_n \rightarrow_v (\varepsilon))),Q_1,Q_2,\cdots Q_n$ is the key of $last(P)$, if $O,(P,(Q_{n+1} \rightarrow Q_1,Q_2,\cdots Q_n)$, then Q_{n+1} is also the candidate key of $last(P)$.

In order to eliminate abnormal XFD, we use the method similar with the one in [2] and [6]. But both methods cannot solve the text problems and not consider relative functional dependency when creating a new element.

Algorithm 1 gives the normalization algorithm including two kinds of reconstruction of DTD as showed in figure 5. Before the normalization of $(D, \sum)$, XFD of the given $\sum$ should be preprocessed. According to the inference rules 4) and 5), slide the objective path and right objective path upward until it cannot continue. $\sum$ after this process will be used in the algorithm.

Algorithm 1. *DTD normalization* algorithm.

Input: Any given $(D, \sum)$ after preprocess

Output: normalized $(D, \sum)$

Step 1. If DTD is normalized, output $(D, \sum)$, algorithm ends.

Step 2. If there exists the first abnormality XFD: $O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1}(S)))$, move the sub tree S of $lasts(Q_{n+1})$ in D to the sub tree of $last(P)$, remove $O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1}(S)))$ from $\sum$.

Step3. If there exists the second abnormality XFD: $O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1})$, process according to the following two cases:

(1) If $last(O)$ is indicated by an absolute key, then a new element type INFO -$last(P)$is generated under the root in D. Define the closure of $P'P'^*$, P' is defined as $(Q_1, Q_2, \cdots Q_n; \sum)$, P is defined as $\{P'children\}$-$(Q_1, Q_2, \cdots Q_n; \sum)$; Delete $O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1})$ from $\sum$, and add r. INFO -$last(P)$. $P'(Q_1, Q_2, \cdots Q_n \rightarrow_v (\varepsilon))$.

(2) If $last(O)$ is indicated by a relative key, $O = R.S$ and $R, (S, (K \rightarrow_v (\varepsilon)))$, then a new element type INFO -$last(P)$ is generated under $last(R)$. Define the closure of $P'P'^*$, P' is defined as $(Q_1, Q_2, \cdots Q_n; \sum)$, P is defined as $\{P'children\}$-$(Q_1, Q_2, \cdots Q_n; \sum)$. Delete $O, (P, (Q_1, Q_2, \cdots Q_n \rightarrow Q_{n+1})$ from $\sum$, and add R,$(r.INFO - last(P).P'(Q_1, Q-2, \cdots Q_n \rightarrow v(\varepsilon)))$.

Step4. Turn to Step 1. In the course of the cycle algorithm, abnormal XFD was removed one by one and DTD convert gradually. Because abnormal XFD is limited, the algorithm will have an end and the output $(D, \sum)$ has been normalized.

7. Conclusion

XML normalization has been researched in this paper, a novel expression of XFD is proposed. In order to better express the XML data constraints that can result in redundancies, XFD and the relationship between key and relative key are indicated, a group of expression methods is proposed. On this basis, there are researches about file normalization and a kind of definition of XML normal form is proposed. After analyzing the problems corresponding to logical implication, a lossless conversion algorithm of DTD is proposed. There are more jobs in the research algorithms of generating XFD, such as key reasoning, calculation of the closure of XFD, calculation of $(X; \sum)$ and so on. The main work in the future is the research of the inference and the algorithm based on XFD.

Acknowledgment: This research is supported by the National Natural Science Foundation of China (Grant No. 60873011, 60773221, 60773219 and 60803026), National Basic Research Program of China (Grant No. 2006CB303103), and the 863 High Technique Program (Grant No. 2007AA01Z192).

References

[1] M. Arenas and L. Libkin. A normal form for xml documents. *ACM Transactions on Database Systems*, 29:195–232, 2004.

[2] P. Buneman, S. Davidson, W. Fan, and et al. Reasoningabout keys for xml. *Lecture Notes in Computer Science*, 39(5):133–148, 2001.

[3] P. Buneman, S. B. Davidson, W. Fan, and C. S. Hara. Keys for xml. *Computer Networks*, 39(5):473–487, 2002.

[4] L. Kot and W. M. White. Characterization of the interaction of xml functional dependencies with dtds. In *Proc of ICDT*, pages 119–133, 2007.

[5] M. L. Lee, T. W. Ling, and W. L. Low. Designing functional dependencies for xml. In *Proc of EDBT*, pages 124–141, 2002.

[6] J. Liu, M. Vincent, and C. Liu. Local xml functional dependencies. In *Proc of WIDM*, pages 23–28, 2003.

[7] T. Lv and P. Yan. Xml normal forms based on constraint-tree-based functional dependencies. In *Proc of APWeb/WAIM Workshops*, pages 348–357, 2007.

[8] W. Provost. *Normalizing XML.* www.Xml.com/pub/a/2002/11/13/ normalizing.html, 2002.

[9] M. S. Shahriar and J. Liu. Preserving functional dependency in xml data transformation. In *Proc of ADBIS*, pages 262–278, 2008.

[10] T. Trinh. Using transversals for discovering xml functional dependencies. In *Proc of FoIKS*, pages 199–218, 2008.

[11] M. Vincent, J. Liu, C. Liu, and M.Mohania. *On the definition of functional dependencies in XML.* http://www.cis.unisa.edu.au/čismwv/papers/index.html, 2005.

[12] M. W. Vincent and J. Liu. Functional dependencies for xml. In *Proc of APWeb*, pages 22–34, 2003.

[13] M. W. Vincent, J. Liu, and C. Liu. *Multivalued dependencies and a 4NF for XML.* http://www.cis.unisa.edu.au/people/mwv/papers/index1html, 2003.

[14] M. W. Vincent, J. Liu, and C. Liu. Strong functional dependencies and theirapplicatiopn to normal forms in xml. *ACM Transactions on Database Systems*, 29(3):445–462, 2004.

[15] J. Wang. A comparative study of functional dependencies for xml. In *Proc of APWeb*, pages 308–319, 2005.

[16] W. X. ying, L. W. Tok, and L. Y. S. et al. NF-SS: A normal form for semistructured schema. In *Proc of the Intl Workshop on Data Semantics in Web Information Systems*, 2001.