# Study on Paradigm and Its Normalization with Coexistence of XML Functional Dependence and Multivalues Dependence

Lijun Cao
Hebei Normal University of Science & Technology

Xiyin Liu, Jing Cao, Aiyong Liu, Zhongping Zhang
Hebei Normal University of Science & Technology, China
College of Information Science and Engineering Yanshan University, China

**Abstract—This paper proposes the paradigm and the corresponding normalization algorithms with coexistence of eXtensible Markup Language (XML) functional dependence and XML values dependence. The concepts of redundancy and keys are firstly described in this paper, while the paradigm with coexistence of XML functional dependence and XML values dependence is defined later. Then, the non-redundancy judgment theorem is also also presented. Furthermore, the XML document tree normalization algorithm is introduced, while the capability to be terminated, validity, and computation complexity are analyzed and proved. Finally, the effectiveness of the algorithm is demonstrated by the experiments.**

***Index Terms*—normalization; redundancy; paradigm; functional dependence; multi-value dependence; normalization rules**

## I. INTRODUCTION

While XML is widely used in more and more fields, a well-performed eXtensible Markup Language (XML) document with more Document Type Definitions (DTDs) is necessary to reduce the data redundancy. Model normalization theory is the theoretical basis to design a well-performed database model. Similar to relation databases, if the DTD design pattern is not suitable enough, abnormal data operation will take place. To fix this problem, normalization study is a good method to eliminate data redundancy and abnormal operation. Therefore, the paradigm theory is one of the hot topics in the research of database [1].

For the cases with coexistence of XML functional dependence and multivalues dependence, this paper proposes a paradigm and its normalization algorithm based on their logical implication and membership. After studying the relation database normalization theorem and characteristics of XML database, we also define 3XNF(XML Normal Form) and 4XNF, and further give normalization rules and a theorem on non-redundancy judgment. The normalization algorithm is finally designed while its effectiveness is then proved by the experiments.

## II PARADIGM IN XML FUNCTIONAL DEPENDENCE

In relation databases, the usage of paradigm can reduce data redundancy and decrease the number of abnormal operations. The concept of paradigm is also utilized in XML. People can discuss whether a DTD is well-designed using a paradigm, and then apply normalization operation on the DTDs which do not meet the paradigm.

### *A. Effective Change and Redundancy*

While XML is widely used nowadays, the design of a good model is becoming an important research topic [2]. Data redundancy not only wastes a lot of space for storage, but also increases the cost of data transfer and data operation. Therefore, a model with a good design pattern is needed to reduce data redundancy.

In relation databases, data redundancy means the repeated saving of the same data for many times. In a relation mode which satisfies dependence set Σ, if an attribute value can be obtained according to the dependence set as well as other attribute values, this attribute value is believed to be redundant. Particularly, there is a relationship instance based on the model, which satisfies the dependence set Σ. If an instance will not satisfy some dependencies in set Σ after a value in this instance has been changed,, the relation model is called to be redundant.

When borrowing the concept of redundancy theory in relation database, we present the definitions of effective change and redundancy as follows.

Definition 1 (Effective change) For a given DTD: D, T is a complete tree satisfying D. Let v be an arbitrary node in T. v is changed into v', where the identifier v' is different from other identifiers in T, and val(v)≠val(v'), a new tree T' is generated after this changing. If T' still satisfies D, then T is called as an effective change.

Definition 2 (Redundancy) For a given DTD: D, ∑ is a collection set of XFD and XMVD (XML Multi-valued


doi:10.4304/jsw.9.6.1595-1603

Dependency), where the paths in XFD and XMVD are all collected in the set Paths (D). In a complete tree T satisfying D and ∑, a node v is changed into v'. If this changing will generate a new tree T', which violates ∑, D is said to be redundant.

### B. *The Third Paradigm in XML (3XNF)*

In relation databases, BCNF is a paradigm with the highest normalization, which only considers the cases with functional dependence [3], while 3NF is a paradigm with the highest normalization to keep the functional dependence. Borrowing the related theory in relation data, we present the formal definition of the third paradigm in XML (3XNF) as below.

Definition 3 (The third paradigm in XML) For a DTD: D, ∑ is the smallest dependance set in XFD. If for each abnormal XFD with p1, … ,pk→qi, there exits XFD with p1, … ,pk→pparent(q), where, last(qi)∈E2∪A, then D is called to be the third paradigm in XML.

The definition of XML abnormal functional dependence which generates redundancy is given as follows.

Definition 4 (XML abnormal functional dependence) Assume there is a given DTD: D and a smallest dependance set containing XFD. For each abnormal XFD with p1, … ,pk→qi, if this XFD does not satisfy the definition of XNF, i.e. p1, … ,pk→qi is established in D, but p1, … ,pk→pparent(q) is not, where last(qi)∈E2∪A, then XFD:p1, … ,pk→qi is called to be abnormal functional dependence.

Theorem 1: Assume in a given DTD D, T is a complete tree satisfying D, ∑ is the XFD collection set, and T satisfies ∑. If D satisfies the 3XNF model, then T is called to be with non-redundancy.

Proof: This theorem is proved by reduction to absurdity. That is: If T is redundant, then D does not satisfy 3XNF mode.

Assume there is redundancy in T, following the definition of redundancy, we have: there exists a node v in T with v∈N(q), and abnormal XFD:P→q is not established when v turns to v′ with effective change.. According to definition 3.1, there are two different route instances q1:v1. ... .vn(vn=v) and q2:w1. ... .wn(wn=v) in Paths(q).

(1) When pparent(q1)≠pparent(q2), we have below conclusionswe have conclusions as given below . because of val(vn)=val(wn) and pparent(q1)≠ pparent(q2), P→parent(q) is not established according to the definition of XFD, and thus it does not satisfy the definition of 3XNF. While When considering that T is a complete tree satisfying D, it's known that D does not satisfy 3XNF.

(2) When pparent(q1)=pparent(q2), below discussion can be established. Because last(q)∈E2∪A, q1 and q2 are the same route according to definition 2.2. This conflicts with the assumption that q1 and q2 are different.

In conclusion, if D satisfies the 3XNF model, then T is called to be with non-redundancy.

### C. *Design and Algorithm for XML Function Normalization*

This section will firstly give normalization rules, then discuss how to change any DTD into normalization model to meet the paradigm of 3XNF, and further present the normalization algorithm.

XML functional dependence normalization rules are given as follows.

(1) Normalization rule 1: move sub-tree with attributes and simple elements.

If the complex element vparent (last (q)) is embedded in another complex element last(P), then vparent (last (q)) is with some attributes or with simple elements, while these attributes or simple elements are closer with last (P). In order to reduce the data redundancy, we can move attributes/child elements with simple elements being last (P), to construct a new DTD D' as shown in below figure to reduce the data redundancy.

Given a DTD: D, ∑ is a minimum function dependence set on XFD in D. Assume there exits an abnormal functional dependence p1, … ,pk→q in D, where last(q)∈E2∪A. If there exists a left redundancy route, i.e. p1, … ,pi→q is established while P′={ p1, … ,pi}, where p′ is the public prefix for all routes in P′, then last(q) is a child element of last(p′), i.e. vparent(last(q))=last(p′), where, pj satisfies pparent(pj)= p′, and pj⊂q,1≤j≤i.

After DTD D has been normalized, paradigm can be represented as D′=(E′,A′,P′,R′,r), where E′=E;A′=A;P′(last(p′))=P(last(p′))∪vparent(last(q));P′(vparent(last(q)))=P(vparent(last(q)))-{last(q)};

R′=R;other definitions of P′ are same to that of P. Let XFD set of D′ be ∑′, this kind of functional dependence is turned into p1, … ,pk→q in ∑′, i.e. ∑′=∑-p1, … ,pk→q∪p1, … ,pk→p′.vparent(last(q)).last(q),k≥1.

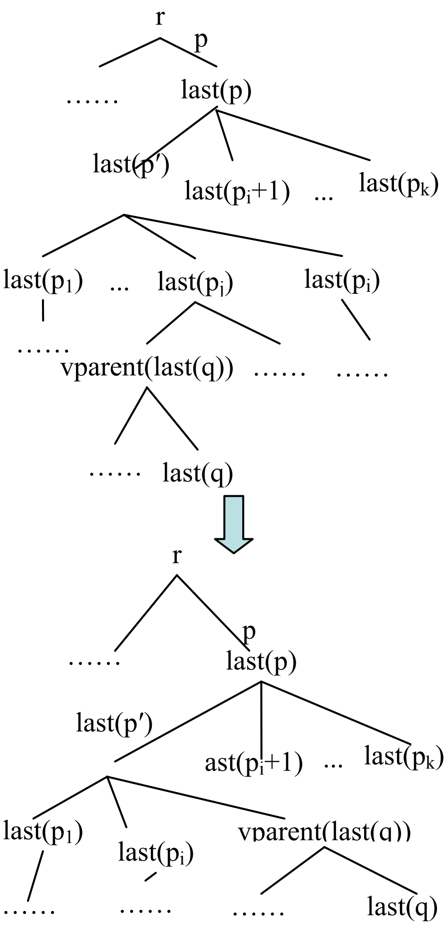



Figure 1 mobile attributes and sub-tree containing simple elements

(2) Normalization rule 2 Generation of element node

If the complex element vparent (last (q)) is embedded in a complex element last(P), attributes or simple type element in vparent (last (q)) have no direct relationship with last (P). This is the time to create new element to reduce redundancy, while the attributes of the new node are that of vparent (last(q)) in the tree.Child elements are the simple attributes of vparent (last(q)) in the original tree. DTD D′ has been constructed as below figure to reduce the data redundancy.

Given a DTD: D, ∑ is a minimum function dependence set on XFD in D. Assume there exits an abnormal functional dependence p1, … ,pk→q in D, where last(q)∈E2∪A. Let p1∩ … ∩pk=p,ancestor(q)∩p=s, if there exists last(pi)∈ E2∪A, we move last(q) to make it be a child element of last(s) i.e.vparent(last(q))= last(s).

In the left route set P={p1, … ,pk}, let all routes in last(pi)∈E2∪A be added in set Vpath(P), i.e.Vpath(P)={pg, … ,ph}, let all routes in last(pi)∈E1 be added in set Epath(P), i.e.Epath(P)={pe,… ,pf}.

After DTD D has been normalized, paradigm can be represented as: D′ =(E′ ,A′ ,P′ ,R′ ,r), where, E′=E∪{lab(vnew)}; A′=A; P′(last(s))=P(last(s))∪{lab(vnew)}; P′(last(vnew))={last(pg),…,last(pf),last(q)}; P′(vparent(last(q)))=P(vparent(last(q)))-{last(q)}; other definitions of P′ are same to that of P. R′(lab(vnew))={vi|vi=last(pi)∈A∧last(pi)∈P′(last(vnew))};R′(vparent(last(q)))= R(vparent(last(q)))-{last(q)|last(q)∈A}; other definition of P′ and R′ are same with that in D. Let the functional dependence set in D′ be ∑′, this kind of functional dependence in ∑ is turned into the formation of s. vnew. last(pg), … ,s. vnew. last(ph)→s. vnew.last(q),i.e.∑′ =∑-p1, … ,pk→q∪ s. vnew. last(pg), … ,s. vnew. last(ph)→s. vnew.last(q),k≥1.

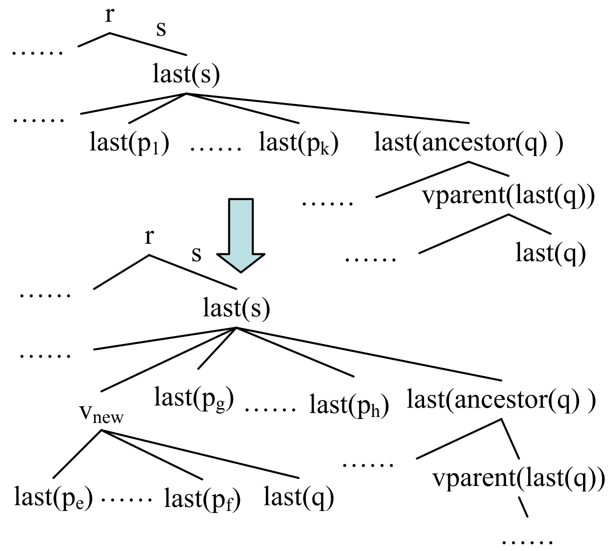



Figure 2 Generation of element node

First of all, D′ is initialized as D, and ∑′ is initialized as ∑. We first need to find out whether D′ satisfies 3XNF. If yes, it directly returns D; if not, it indicates that there exists abnormal functional dependence, and operation given below should be done. If the right tail node of the abnormal functional dependence is a simple element or attribute node, wherein the left path of this dependence is with public prefix of p and we find that ∑ containing logically p1, … ,pk→p according to the membership algorithm, then normalization can be done with moving elements and following normalization rules. Otherwise, normalization will be done by generating element node normalization rules.

Algorithm 1:Algorithm of converting DTD D into DTD D′, which satisfies 3XNF (DTD-to-3XNF).

Input:DTD D and functional dependence set ∑;

Output:DTD D′ satisfying 3XNF

DTD-to-3XNF(D, ∑)

Begin

(1) Inilization:D′:=D,∑′:=∑;

(2) If D′ satisfies 3XNF,turn to (6);

(3) While each abnormal XFD∈F do

(4) Case1:if p1, … ,pk→q∈F and p⊂q,where last(q)∈E2∪A,p1∩ … ∩pk=p and DEP_MERMBERSHIP(∑,p1, … ,pk → p) then use normalization 1;

(5) Case2:if p1, … ,pk→q and last(pi)∈E2∪A,where last(q)∈E2∪A,s=ancestor(q)∩p,p1∩ … ∩pk=p and DEP_MERMBERSHIP(∑,p1, … ,pk→p) then use normalization 2;

(6) Return(D′).

End

The analysis of algorithm 1 is as below.

(1) The correctness of the algorithm

The algorithm presents the corresponding solutions according to the different situations. It makes XFD satisfying its definition using not only the method of moving the attributes and simple element, but also the method of creating and copying element nodes. If the definition of XNF is satisfied, apparently the correctness of the algorithm is established.

(2) The capability to be terminated

This capability is determined by the number of the abnormal functional dependence because this number is limited, and can be terminated. Therefore, the whole algorithm can be terminated.

(3) The time complexity of the algorithm

Let B be the number of abnormal XFD in ∑. The algorithm's time complexity depends on the time complexity of DEP_MERMBERSHIP algorithm and the number of execution of DEP_MERMBERSHIP, while the DEP_MERMBERSHIP algorithm's time complexity is O(n×m3). DEP_MERMBERSHIP execution times are determined by the number of abnormal XFD b. Therefore, the algorithm's time complexity is $O(b\times n\times m^3)$

## III PARADIGM WITH COEXISTENCE OF XML FUNCTIONAL DEPENDENCE AND XML VALUES DEPENDENCE

### *A The Fourth Paradigm*

In relation databases, the fourth paradigm is with the highest paradigm with coexistence of XML functional dependence and XML values dependence [4]. Borrowing the related theories in relation databases, the formal definition of the fourth paradigm (4XNF) is given as below.

Key is an important part in semantic constraints [5]. Key will be set as an important data integrity constraint [6] in the definition of XML mode. The formal definition of key is given below.

Definition 5(XML key) Given DTD D, T is a complete tree satisfying D, p∈Paths(D). If two arbitrary different nodes v1,v2,(v1 and v2 belong to N(p)) satisfy val(v1)≠val(v2), p is called as key, and recorded as key(p).

Definition 6 (XML the fourth paradigm) Given DTD D, T is a complete tree satisfying D, ∑ is a XMVD set. If each XMVD in ∑: p1,…,pk →→ q1,…,qm|r1,…,rs, satisfies any one of below conditions, then DTD D is the fourth paradigm of XML.

(1) key(qi),key(rj);

(2) key(ph) and qi∩rj =ph;

(3) key(ph) and qi∩rj are the strict predix of ph ;

(4) qi∩rj =root;

(5) key(ph) and rj⊂qi or rj⊂ph;

where,ph=p1∩...∩pk;qi=q1∩...∩qf;rj=r1∩...∩re.

Lemma 1:Given DTD D, ∑ is a XMVD set, and all routes in it are in Paths(D). Assume that ∑ causes redundancy, then there exists a multi-value dependence p1,…,pk→→q1,…,qm|r1,…,rs, satisfying:

(1) v∈N(qi) or v∈N(rj),1≤i≤m,1≤j≤s;

(2) There exists two different instances v1.v2. … .vt-1.vt (vt=v) and w1.w2. … .wt-1.wt∈Path(qi),1≤t, which makes val(vt)=val(wt);

(3) there exists two instances z1,z2, z1∈Nodes(xij,rj),z2∈Nodes(yij,rj), making val(z1)≠val(z2);

(4) there exists two instances z3,z4, z3∈Nodes(xijh,ph),z4∈Nodes(yijh,ph) ,making val(z3)=val(z4),1≤h≤k.

Proof: (1) we first prove that if ∑ causes redundancy, then there exists a multi-value dependence XMVD p1,…,pk →→ q1,…,qm|r1,…,rs, making it satisfy condition (1) in lemma 1.

If the situation is not established, there are two possibilities. The first one is as below. If v∉N(qi), v∉N(rj) and v∉N(ph), because v does not belong to any node in a route of ∑, when node v turns to v, the generated new tree T does not violate ∑, and conflicts with ∑ to cause redundancy contradiction. The other possibility assumes v∈N(ph), in which conditions, if lab(v)∈E1, according definition 1, the changed v' does not belong to any node in T'. Same as v in T, T' satisfies ∑ according to definition 3. Therefore, ∑ cannot cause redundancy or conflict with ∑. If lab(v)∉E1, after effective changes, node v makes val(v') not appear in T. In the new tree T after the changes, XMVD:p1,…,pk→→ q1,…,qm|r1,…,rs still holds, and causes redundancy contradiction with ∑. Thus, condition (1) in lemma 1 is established.

(2) Next we want to prove that if ∑ causes redundancy and v∈N(qi), i.e. satisfying condition (1) in lemma 1, there exists a XMVD p1,…,pk →→ q1,…,qm|r1,…,rs, making itself satisfy condition (2) in lemma 1.

Assume this case is not established, thenthere are two conditions. The first one is: if there is only one route instance in Paths(qi), Paths(qi) in the new generated T' also has only one route instance after node v experiences an effective change, then XMVD p1,…,pk→→q1,…,qm|r1,…,rs is known to be established in T', and conflicts with ∑; if there are more than one routes in Paths(qi) with different tail node values, let v1.v2. … .vt-1.vt∈Paths(qi),vt=v. If in Paths(qi), there exist route instances w1.w2. … .wt-1.wt, where vt≠wt, and makes conditions (2) and (3) in definition 7satisfied, then according to definition 7, it's known that v'1. v'2. … . v't-1. v't are same to v1.v2. … .vt-1.vt. For the same reason, w'1. w'2. … . w't-1.w't are same to w1.w2. … .wt-1.wt according to condition (5) in definition 7, then after effective changes in node v and generation of new tree T',XMVD:p1,…,pk→q1,…,qm|r1,…,rs is still established, and conflicts with ∑. The other conditions is: if in Paths(qi) there is no route instances w1.w2. … .wt-1.wt, which satisfy condition (2) and (3) in definition 7, then it's known that after effective changes in node v and generation of new tree T', XMVD:p1,…,pk→q1,…,qm|r1,…,rs is still established, and conflicts with ∑. Therefore, condition (2) in lemma 1 is established.

(3) Then we prove that if ∑ causes redundancy, there exists a multi-value dependence XMVD p1,…,pk→q1,…,qm|r1,…,rs, which satisfy condition (3) in lemma 1.

Assume this case is not established. Then there do not exist two nodes z1, z2, where z1∈Nodes(xij,rj), z2∈Nodes(yij,rj), which make val(z1)≠val(z2). It's known that route rj only has one route instance. From definition 7, XMVD:p1,…,pk→q1,…,qm|r1,…,rs still holds after effective changes in node v and generation of new tree T', XMVD:p1,…,pk→q1,…,qm|r1,…,rs still holds, and it conflicts with Σ. Thus condition (3) in lemma 1 is established.

(4) Finally, we prove that if ∑ causes redundancy, there exists a multi-value XMVD p1,…,pk→q1,…,qm|r1,…,rs, which satisfies condition (4) in lemma 1.

Assume this case is not established. Then there do not exist two nodes z3, z4, where z3∈Nodes(xijh,ph), z4∈Nodes(yijh,ph), which make val(z3)= val(z4), 1≤h≤k. It's known that the tail nodes of the instances in route ph are different. From definition 7, after effective changes in node v and generation of new tree T' XMVD:p1,…,pk→q1,…,qm|r1,…,rs still holds, and conflicts with ∑. Therefore, condition (4) of lemma 1 is established.

Due to the symmetry property, the cases when v∈N(rj) keeps the same conclusions as above. here xij,yij,xijh,yijh,x'ij,y'ij,x'ijh,y'ijh holds the same meaning with that in definition3.

The non-redundancy judgment theorem can be obtained by following definition 6 and lemma 1.

Theorem 2: Given DTD D,T is a tree satisfying D, ∑ is a XMVD set, T satisfies ∑. If D is with 4XNF mode, then ∑ has no redundancy in T.

Proof: Statements given below prove that the cases of v∈N(rj) hold the same results with that in v∈N(qj) due to the symmetry property.

(1) Assume that condition (1) in definition 6 holds. That is when qi,rj are both keys, ∑ causes redundancy.

According to lemma 1, after node v's effective changes into v', then there exists v1∈N(qi),v2∈N(qi), and val(v1)=val(v2). Thus it's known that it conflicts with qi being the key, as well as the above assumption.

(2) Assume that condition (2) in definition 6 holds. That is when ph is a key and qi∩rj =ph, ∑ causes redundancy.

According to lemma 1, if ∑ causes redundancy, that is there exists a complete tree T satisfying D and ∑, if there is a node v in T, after effective changes into v', a new tree T' is generated, which leads that XMVD p1,…,pk→q1,…,qm|r1,…,rs does not hold, as well as v∈N(qi). Because XMVD p1,…,pk→q1,…,qm|r1,…,rs is not established in T', then Paths(qi) in T' keep two route instances v1.v2. … .vt-1.vt (vt=v') and w1.w2. … .wt-1.wt, satisfying condition (1),(2),(3), but not (4) or (5) in definition 7. Because ph is a key, i.e. the tail node values in instance ph are different, from condition (3) in definition 7, i.e. val(z3)=val(z4), it's known that xijh=yijh; from condition (4) in definition 7, i.e. val(z'3)=val(z3), as well as val(z'4)=val(z4) in (5), it's known that xijh=x'ijh,yijh=y'ijh. Because qi∩rj=ph,then we have N(qi∩rj)=N(qi∩rj∩ph)=N(ph), and xijh=yijh=x'ijh=y'ijh= xij=yij=x'ij=y'ij. Therefore, if T' does not satisfy XMVD p1,…,pk→q1,…,qm|r1,…,rs, then T does not satisfy XMVD p1,…,pk→q1,…,qm|r1,…,rs, which conflicts with the assumption.

(3) Assume that condition (3) in definition 6 holds. That is when ph is a key and ph is a strict prefix, ∑ causes redundancy.

According to lemma 1, if ∑ causes redundancy, that is there exists a complete tree T satisfying D and ∑, if there is a node v in T, after effective changes into v', a new tree T' is generated, which leads that XMVD p1,…,pk→q1,…,qm|r1,…,rs does not hold, as well as v∈N(qi). Because XMVD p1,…,pk→q1,…,qm|r1,…,rs is not established in T', then Paths(qi) in T' keeps two route instances v1.v2. … .vt-1.vt (vt=v') and w1.w2. … .wt-1.wt, satisfying condition (1),(2),(3), but not satisfying (4) or (5) in definition 7. Because ph is a key, i.e. the node values in instance ph are different. Further because qi∩rj is the strict prefix of ph, then we have qi∩rj=qi∩rj∩ph. From condition (2),(3) in definition 7, it's known that xij=yij=xijh=yijh. From condition (4),(5) in definition 7, it's known that xij=x'ij,yij=y'ij,xijh=x'ijh,yijh=y'ijh. In total, xij=yij=x'ij=y'ij=xijh=yijh=x'ijh=y'ijh. Therefore, if T' does not satisfy XMVD p1,…,pk→→q1,…,qm|r1,…,rs, then T does not satisfy XMVD p1,…,pk→→q1,…,qm|r1,…,rs, which conflicts with the assumption.

Assume that condition (4) in definition 6 holds. That is when qi∩rj =root, ∑ causes redundancy.

According to lemma 1, if ∑ causes redundancy, which means that there exists a complete tree T satisfying D and ∑, if there is a node v in T, after effective changes into v', a new tree T' is generated, which leads that XMVD p1,…,pk→→q1,…,qm|r1,…,rs does not hold, as well as v∈N(qi). Because XMVD p1,…,pk→→q1,…,qm|r1,…,rs is not established in T', then Paths(qi) in T' keeps two route instances v1.v2. … .vt-1.vt (vt=v') and w1.w2. … .wt-1.wt, satisfying condition (1),(2),(3), but not (4) or (5) in definition 7. Because qi∩rj =root, then xij=yij=x'ij=y'ij=xijh=yijh=x'ijh=y'ijh, and they are all root nodes. Following consition (3), ∑ can be proved to cause no redundancy, which conflicts with the assumption.

(5) Assume that condition (5) in definition 6 holds, i.e. key(p) and rj⊂qi or rj⊂ph,∑ causes redundancy. Because ph is a key, i.e. the tail node values are different. Then from condition (3) in definition 7, it's known that $val(z_3)=val(z_4)$ and $x_{ijh}=y_{ij}$. From $val(z'_3)=val(z_3)$ in (4) and $val(z'_3)=val(z_3)$ in (5), it's known that $x_{ijh}=y_{ijh}=x'_{ijh}=y'_{ijh}$. When $r_j \subset q_i, r_j \cap q_i=r_j$,and there exists a node v after effective change to v',and generating a new tree T', which does not affect ($r_j$). From (4) and (5) in definition 7, it's known that XMVD $p_1,\ldots,p_k \to\to q_1,\ldots,q_m|r_1,\ldots,r_s$. Therefore, ∑ does not cause any redundancy, which conflicts with the assumption.

where $x_{ij},y_{ij},x_{ijh},y_{ijh},x'_{ij},y'_{ij},x'_{ijh},y'_{ijh}$ have the same meaning to that in definition 3.

### *B Design and Algorithm of XML Multi-value Dependence Normalization*

Algorithm of XML multi-value dependence normalization is as follows.

Assume that complete XMLdocument tree T is a XMVD satisfying 3XNF and minimum dependence set ∑. P→→Q|R is assumed to be XMVD in ∑. Let {P1,...,Pn} represent instance set of all P in tree T satisfying P→→Q|R. All responding instance set in Q is represented as {Q1,...,Qqn}, and all responding instance set in Q is represented as {R1,...,Rn}.

First of all, for P→→Q|R, let p be the left path public prefix, q be the public prefix of all routes in Q, r be the public prefix of all routes in Q. For route instance set Ph∈Paths(P), where,ph,qh,rh are the public prefixes of all route instance sets Ph,Qh,Rh. If the tail node of ph is a complex element node, then the longer route between qh and rh is chose, which makes the parent node of the route's tail node be last(p); if the tail node of ph is a simple element node or attribute node, then the longer route between qh and rh is chose, which makes the parent node of the route's tail node be a brother node of last(ph).

For route instance Pt in P of different instance Ph, when Pt=Ph, we first find out whether last(qh) and last(qt) have the same value. If not, and the value of last(qt) does not equal to the tail node of any route, which have pparent(qh) as their perant node, then modify route qt,and make it be the brother node of last(qh); when last(qh) has same value as last(qt), then delete route qt. For the same reason, find out whether the value of last(rh) equals to that of last(rt), if not, the value of last(rt) does not equal to the tail node value of any route with parent node being pparent(rh) in T′ , then modify route rt to be brother node of last(rh); if last(rh) has the same value as last(rt), then delete route rt. At last, delete route instance set Pt. Above operations make p be key route.

After above operations, condition (2) and (3) in definiation of 4XNF are satisfied.

Algorithm 2: the algorithm to make DTD D be changed into D′ , which satisfies 4XFN.

Input: the mimimum dependence set ∑ of XFD and XMVD, the complete tree T satisfying ∑.

Output: XML document tree T' satisfying XMVD paradigm

DTD-to-4XNF(∑,T)

Begin

(1) T':=T,∑':= ∑;

(2) If T' satisfies 4XNF,then turn to (4);

(3) For each XMVD:P→→Q|R do

(4) Paths(P):= {P1,...,Pn};

(5) P={p1,...,pk };

(6) p=p1∩...∩pk;

(7) Paths(Q):={Q1,...,Qf};

(8) Q={ q1,...,qf };

(9) q=q1∩...∩qf;

(10) Paths(R):={R1,...,Re};

(11) R={r1,...,re };

(12) r=r1∩...∩re;

(13) For each Ph∈Paths(P) do

(14) if any condition in definition of 4XNF is not satisfied, then

(15) if last(ph)∈E1

(16) if length(qh)≤length(rh) then

(17) {vparent(vparent(last(rh)))=last(ph);

(18) Modify rh in T' to modified rh,and modify rh in ∑' to modified rh;}

(19) else

(20) {vparent(vparent(last(qh)))= last(ph); Modify qh in T' to modified qh,and modify qh in ∑' to modified qh; }

(20) else

(21) if length(qh)≤length(rh) then

(22) {vparent(vparent(last(rh)))= vparent(last(ph)); Modify rh in T' to modified rh,and modify rh in ∑' to modified rh; }

(23) else

(24) {vparent(vparent(last(qh)))=vparent(last(ph));

(25) Modify qh in T' to modified qh,and modify qh in ∑' to modified qh; }

(26) For route instance Pt of P different from Ph do

(27) If (Pt=Ph) then

(28) If val(last(qh))≠val(last(qt)) and val(last(qt)) is not equal with the tail node value of any route node with parent node being pparent(qh) then

(29) {x:= last(qt);

(30) vparent(x):=vparent(last(qh));

(31) Modify qt in ∑' to modified qt;}

(32) else if val(last(qh))=val(last(qt)) and vparent(last(qh)≠vparent(last(qt) then

(33) Paths(Q):=Paths(Q)- qt;

(34) if val(last(rh))≠val(last(rt)) and val(last(rt)) is not equal with the tail node value of any route node with parent node being pparent(rh) then

(35) {y:= last(rt);

(36) vparent(y):=vparent(last(rh));Modify rt in ∑' to modified rt;}

(37) else if val(last(rh))= val(last(rt)) and vparent(last(rh)≠vparent(last(rt) then

(38) Paths(R):=Paths(R)- rt;

(39) Paths(P):=Paths(P)- Pt;

(40) Return(T');

End

Analysis of the algorithm is as follows.

(1) The correctness of the algorithm

Initialization is firstly done to make T' be T, ∑' be ∑. For each multi-value dependence in ∑, P→→ Q|R, let p be the public prefix of all route in P, q be the public prefix of all route in Q, r the public prefix of all route in R. Then do the following operations: firstly find out whether this multi-value dependence satisfies the conditions in definition of 4XFN, if not, find whether last(p)∈E1 is established, if it's established, then judge length(qh)≤length(rh) is satisfied or not. If it is satisfied, move node in T' to realize vparent(vparent(last(rh)))=last(p), if it is not satisfied, mode node in T' to realize vparent(vparent(last(qh)))=last(p). The aim of above operations is to make route q∩r be the child node of p. If last(p)∈E1 is not established, then it's known that this multi-value dependence has only one route in its left, and the tail node is a simple element node or an attribute node, then find out whether length(qh)≤length(rh) is established or not. If yes, move node in T' to realize vparent(vparent(last(rh)))= vparent(last(p)); if not, move node in T' to realize vparent(vparent(last(rh)))= vparent(last(p)). The aim of above operations is to make route q∩r be a brother node of p.

The next step is to compare Pt with Ph. If they are equivalent, then find out whether val(last(qh))≠ val(last(qt)) is established or not. If yes, and val(last(qt)) does not equal to the tail node value of any route node with parent node being pparent(qh) in T', then copy tail node of qt to be a brother node of tail node of qh. If val(last(qh))= val(last(qt)) holds, then delete route qt in T'. Moreover, find whether val(last(rh))≠ val(last(rt)) is established or not. If yes, and val(last(qt)) does not equal to the tail node value of any route node with parent node being pparent(qh) in T', then copy tail node of rt to be a brother node of tail node of rh. If val(last(rh))= val(last(rt)) holds, then delete route rt in T'. Repeat above operations, and then combine q and r in the same P of T' to make routes satisfyq and r be all strong route. Then condition (1) in definition 6 is satisfied. For any two different routes Pt and Ph in P, if they are equivalent, delete route set Pt in T'. All routes that satisfy P in T' by this algorithm are strong. Then condition in definition 6 is satisfied.

When the outermost of the cycle is utilized, each XMVD in ∑' satisfies the conditions of the definition 6. Thus T' obtained by this algorithm meets 4XNF

(2) The capability to be terminated

The algorithm is constructed by three level cycles. The outermost cycle is determined by the number of XMVD, while the two internal cycles are determined by whether complete XML tree is limited. Therefore, the whole algorithm can be terminated because the number of XMVD and XML complete tree are both limited.

(3) The time complexity of the algorithm

The algorithm is constructed by three level cycles. The outermost cycle is determined by the number of XMVD, while the number of the loop of the second cycle is determined by the maximum rout instance set. Assume that the maximum route instance set contains d instances, (d is a multiple of 4). At best, in the first operation of the second cycle, the inetermost cycle operates for d-1 times. In the second operation of the second cycle, the inetermost cycle operates for d-5 times. From this trend, in the last operation of the second cycle, the inetermost cycle operates for 3 times. The second cycle operates for d/4 times at best, it's known that the total times is m(d2+2d)/8. At least, for the similar computing process, the total operation times is m(d2-d)/2. Therefore, the time complexity is O(m×d2).

## IV. Experimental Analysis

### A Experiment Settings

Experimental environment settings are as follows.

All algorithm realizations in this experiment are as follows.

(1) Hardware ,CPU: Intel 2.20GHz, Memory: 2.00GB, Hard Disk: 250G.

(2) Operation system: Microsoft Windows XP.

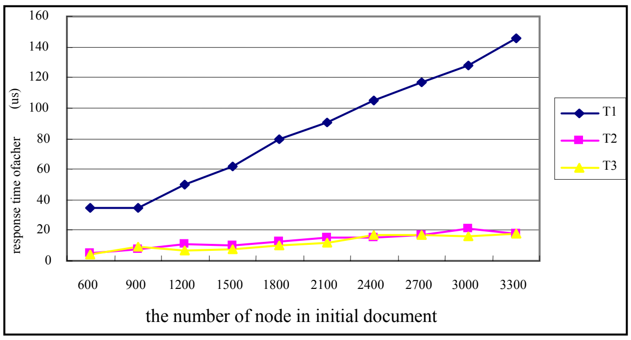
(3) Programming environment: Java, Altova XMLSpy 2010.

Data set settings are as given below . This experiment used combined data set Course and real data set reed to verify the correction of the paradigm of the algorithm.

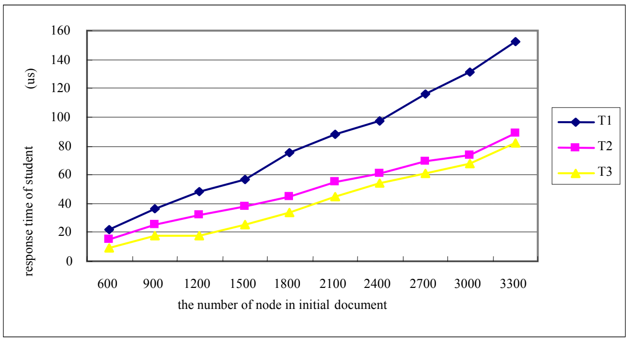
### B Experiment Result and Performance Analysis

According to the three kinds of DTDs: D1, D2, D3, given by this paper, and the corresponding document trees T1, T2, T3, effectiveness of the normalization algorithm is verified in storage and query efficiency.

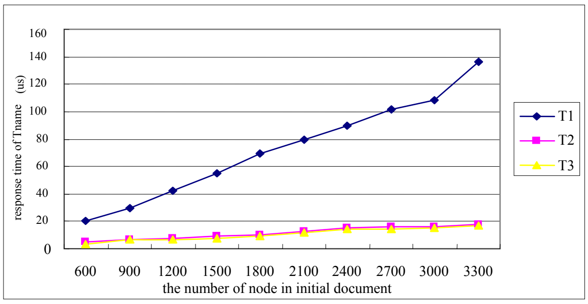
Along with the gradual increase of nodes in original document trees T1, T2 are generated after normalization by algorithm 1, and then T3 is generated by algorithm 2. The response time needed for searching a particular node is shown as Figure 3. The experimental results indicate that XML document query response time which satisfies paradigm with higher normalization degree is much lower than that with lower normalization degree.



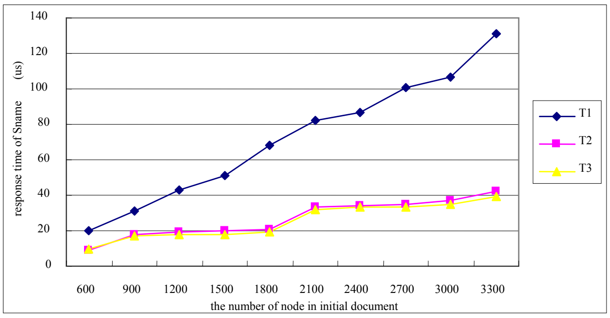

a) The relationship between response time of teacher and the number of nodes in initial document




b) The relationship between response time of student and the number of nodes in initial document




c) The relationship between response time of Tname and the number of nodes in initial document




d) The relationship between response time of Sname and the number of nodes in initial document

Figure 3 The relationship between response time and the number of nodes in initial document

Along with the gradual increase of nodes in original document trees T1, T2 are generated after normalization by algorithm 1, and then T3 is generated by algorithm 2. The storage needed for the three document trees is as shown in Figure 4. The experimental results indicate that XML document storage which satisfies paradigm with higher normalization degree is much lower than that with lower normalization degree.

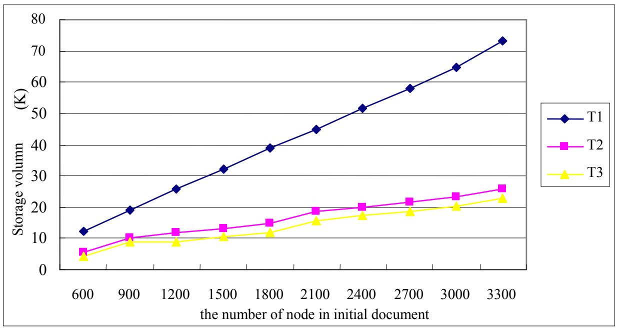



Figure 4 The relationship between storage volume and the number of nodes in initial document

The real data base uses the XML data base of course information of Reed College. T2 is generated based on original document tree T1 after normalization by algorithm 1, and then T3 is generated by algorithm 2. The query response time of the three document trees is shown as Figure 5. The experimental results indicate that XML document query response time which satisfies paradigm with higher normalization degree is much lower than that with lower normalization degree.

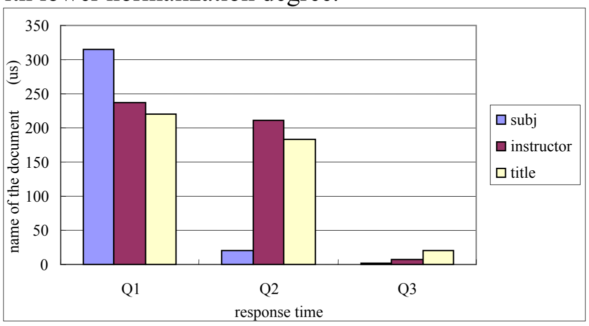



Figure 5 The relationship between response time and the number of node in initial document

T2 is generated based on T1 after normalization by algorithm 1, and then T3 is generated from T2 by algorithm 2. The storage needed of the three documents is as shown in Figure 6. The experimental results show that XML document storage which satisfies paradigm with higher normalization degree is much lower than that with lower normalization degree.

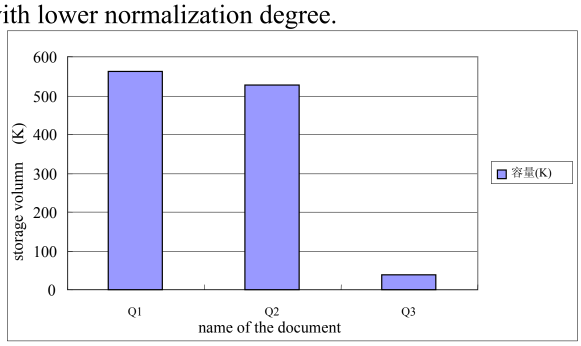



Figure 6 The relationship between storage volume and the number of node in initial document

## V CONCLUSIONS

This paper describes the redundancy and key in XML, and presents the formal definition of 3XNF and 4XNF. Normalization rules and non-redundancy judgement theorem are then proposed. Furthermore, normalization algorithm based on XML document is also given which is followed by corresponding analysis and proof with respect to not only the capability to be terminated, but also the correctness, as well as the computation complexity. Finally, it has been shown that the effectiveness of the algorithm can be demonstrated by the experiments.

## ACKNOWLEDGEMENT

This work has been financially supported by the Project of Major Reform to the Network Enginnering Programme in Hebei Normal University of Science and Technology, which was assigned by Hebei Education Department in 2012.

This work has been also sponsored by Qinhuangdao science and technology research (201101A038\201101A185\2012021A133)

## REFERENCES

[1] James J. Lu, Sonali Renjen. Normalizing XML Schemas through Relatinal Database[C]//Proceedings of the 43nd Annual Southeast Regional Conference, Kennesaw, Georgia, Alabama, USA, 2005: 220-221.

[2] Cong Yu, H.V.Jagadish. XML schema refinement through redundancy detection and normalization[J]. the VLDB Journal, 2008, 17(2): 203-223.

[3] Flavio Ferrarotti, Sven Hartmann, Henning Köhler, et al. The Boyce-Codd-Heath Normal Form for SQL[C]//Proceedings of 18th International Workshop on Logic, Language, Information and Computation, Philadelphia, PA, USA, 2011:110-122.

[4] Flavio Ferrarotti, Sven Hartmann, Henning Köhler, et al. Foundations for a Fourth Normal Form over SQL-Like Databases[C]//Conceptual Modelling and Its Theoretical Foundations.Essays Dedicated to Bernhard Thalheim on the Occasion of His 60th Birthday.London,USA,2012:85-100.

[5] Md. Sumon Shahriar, Jixue Liu. Towards the Preservation of Keys in XML Data Transformation for Integration[C]//Proceedings of the 14th International Conference on Management of Data, IIT Bombay, Mumbai, India, 2008: 116-126.

[6] Flavio Ferrarotti, Sven Hartmann, Schastian Link, et al. Performance Analysis of Algorithms to Reason about XML Keys[C]//Proceedings of 23rd International Conference of Database and Expert Systems Applications, Vienna, Austria,2012:101-115.

[7] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, Lizhu Zhou. EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data[C]// Proceedings of the ACM SIGMOD international conference on Management of data ,SIGMOD 2008, Vancouver, BC, Cannada, 2008: 903-914.

[8] Richard Cyganiak, David Wood. RDF 1.1 Concepts and Abstract Syntax[EB/OL]. (2011-08-30)[2012-10-15].http://www.w3.org/TR/2011/WD-rdf11-concepts-20110830.

[9] Flavio Ferrarotti, Sven Hartmann, Schastian Link. A Precious Class of Cardinality Constraints for Flexible XML Data Processing[C]//Proceedings of 30th International Conference on Conceptual Modeling, Brussels, Belgium. 2011:175-188.

[10] M. W. Vincent, J. Liu, M. Mohania. The implication problem for 'closest node' functional dependencies in complete XML documents[J]. Journal of Computer and System Sciences, 2012, 78(4): 1045-1098.

[11] Jixue Liu, Jiuyong Li, Chengfei Liu, Yongfeng Chen. Discover Dependencies from Data—A Review[J]. IEEE Transactions on Knowledge and Data Engineering.2012,24(2):251-264.

[12] Sven Hartmann, Sebastian Link, Thu Trinh. Solving the Implication Problem for XML Functional Dependencies with Properties[C]//Proceedings of the 28th international Workshop on Logic, Language, Information and Computation, Brasilia, Brazil, 2010:161-175

**Lijun Cao,** born in 1971, has a master degree. She is an associate professor in the college of mathematics and information science of Hebei Normal University of Science and Technology. her main research directions include data mining, grid technology.

**Xiyin Liu,** born in 1970, has a master degree. He is an associate professor in the college of mechanical and electrical engineering of Hebei Normal University of Science and Technology. His main research directions include data mining, grid technology.