

Title Normalization Theory for XML
Contact Author Murali Mani
Email mmani@cs.wpi.edu
Address Computer Science Department,
Worcester Polytechnic University,
100 Institute Road,
Worcester, MA 01605
Phone (508) 831 6421
Fax (508) 831 5776

Normalization Theory for XML

Murali Mani

Department of Computer Science

Worcester Polytechnic University,

100 Institute Road, Worcester, MA 01605

e-mail: {mmani}@cs.wpi.edu

Abstract XML is being widely used as the logical data model because of the several favorable and powerful features it provides such as union types. In order to come up with good XML designs, it is necessary to study normalization theory, which helps the database designer to understand and decrease the redundancies that may be present. Normalization theory has been studied for different logical data models, especially the relational model and nested relational model. There has been some work on normalization for XML, however most of the existing work for XML normalization assume a model for XML similar to nested relational model. We show that there are some fundamental differences between the nested relational model and the XML model, so we cannot use results from nested relational model directly to XML. However we can use very similar approaches. In this paper, we describe how we can obtain an abstract relational representation for an XML representation. This illustrates the fundamental differences between nested

relational model and XML, and allows us to define functional and multivalued dependencies and normal forms similar to relational normal forms. Our systematic approach facilitates easier understanding of the normalization theory for XML, and allows the database designer to choose the normalization level as suited for the application. Further, we provide a set of examples as a proof of concept for our ideas.

Keywords: XML, Normalization, Data Modeling.

1 Introduction

XML (eXtensible Markup Language) [4] has established itself as a powerful logical data model, because of the several favorable features it provides such as union types. The usages of XML as a logical model is in two scenarios (1) Applications provide an XML view of an underlying data source, which could be relational, text or object-oriented data source. Users/applications see this XML

view and perform queries/updates on this view (2) Applications store the data as XML in specialized “XML databases”. Again users/applications see an XML view and perform queries/updates on this XML view. For both the above usage scenarios, normalization theory is useful. Normalization theory characterizes the redundancies present in the data representation of our application.

Our data representation has redundancy if an object is specified more than once. The problem with redundancies is in inserts, deletes and updates: suppose an object is specified multiple times in the logical model, then to update the object, the user must explicitly update the multiple copies. However redundancy in a logical model is not always bad. As an example, consider a *person* relation with attributes (*name, street, city, state, zip*). There is redundancy in the above relation because given a zip, we know the city and state. However the redundancy is often desirable, because of ease of use for the user, and also possibly for efficiency. What a database designer needs is an understanding of the redundancies present in the data representation, and he/she can choose whether it is good to have the redundancy or not. In this paper, we present normalization theory for XML, which will assist a database designer in coming up with good XML representations.

The traditional approach to normalization theory has been to define multiple normal forms, and depending on the application requirements, the database designer

can choose the level of normalization that is appropriate. For example, normalization has been studied very widely with respect to relational model [11,9], and we have multiple normal forms, each provide a different level of normalization. For example, one normal form for relational model is BCNF. However sometimes BCNF may not be desirable because of our application requirements; it is also possible that we may lose some information on functional dependencies by converting a relation to BCNF [11].

A model similar to XML is nested relational model, and normalization theory for this model has also been studied quite extensively in [17,16,18,14]. These approaches provide lot of insight into normalization for models similar to XML. However we cannot use the results from nested relational model directly because when we consider un-nesting of XML and nested relational representations, they provide different flat relational representations, as we will see in Section 3.

Normalization theory for XML also has been studied in the recent past. In [1], the authors approach XML normalization similar to nested relational model normalization. Another approach to XML normalization using ideas from nested relational model is in [8]. In [2], the authors use an information theoretic approach to study normalization for XML, the motivation being the fact that XML standards are still very flexible with respect to constraint specification, structural specification [15], and update language. In this paper, we study normaliza-

tion theory for XML similar to relational normalization.

Our important contributions are

- Describe a framework for specifying structures, and constraints such as keys, foreign keys, and functional and multivalued dependencies for the XML model, using our prior work in conceptual models for XML [13].
- Define an abstract relational representation that illustrates the differences between nested relational model and XML model, and helps us relate our normal forms for XML to corresponding relational normal forms.
- Define normalization steps, similar to those defined in [1].
- Define different normal forms for XML 2NF, 3NF, BCNF and 4NF, equivalent to the relational counterparts, show how we can verify, given a set of functional and multivalued dependencies, whether our XML representation satisfies the different normal forms with respect to these dependencies, and how we can redesign our XML representation to satisfy the normal forms.

Outline of the paper

In Section 2, we describe how we will specify structures and constraints in XML model using XGrammar. We use XGrammar in stead of a particular schema language for XML because the XML standards are still very flexible with respect to structural and constraint specification. In Section 3, we introduce our abstract relational

representation, distinguishing between nested relational model and XML model. In Section 4, we describe our normalization steps. In Section 5, we describe our different normal forms for XML, how we can verify, given an XGrammar and the set of functional and multivalued dependencies, whether the XGrammar is in a particular normal form with respect to these dependencies, and how to obtain an XGrammar that satisfies each normal form. In Section 6, we consider a few examples from previous work and compare them to XML normalization. We finally conclude with future research directions.

2 XGrammar

There are three different XML Schema languages that are widely used at present: DTD [4], XML-Schema [19], and RELAX-NG [7]. These three schema languages specify structures for XML using different subclasses of regular tree grammars [15], DTD uses local tree grammars, XML-Schema uses single-type tree grammars, and RELAX-NG uses full fledged regular tree grammars. Similarly there are different constraint specification proposals for XML in [19, 5, 10]. There is very little study with respect to the kind of XML Schema language needed for practical database applications. In [13], we study the above by considering conceptual models such as Entity Relationship (ER) model [6], as a fair representative of database application requirements.

Based on our study and to provide a flexible way of specifying structures and constraints, we use XGram-

mar. XGrammar in effect combines structural specification provided by RELAX-NG, data type specification provided by XML-Schema and constraint specification similar to [10]. Further more, the schemas specified in XGrammar corresponding to database applications can be specified using XML-Schema. In this section, we define XGrammar formally, and sketch some of the reasons for choices made with respect to constraint specification.

We use \mathbb{G} to denote an XGrammar and $L(\mathbb{G})$ to denote the language that \mathbb{G} generates. We assume the existence of a set \hat{N} of non-terminal symbols, a set \hat{E} of element names, a set \hat{A} of attribute names, and a set $\hat{\tau}$ of atomic data types as defined in [3], such as string, integer, ID, IDREF(S) etc. We use the following notations: ϵ denotes the empty string, $+$ denotes union, “ $,$ ” denotes concatenation, “ $a^?$ ” denotes zero or one occurrence, “ a^* ” denotes Kleene star, and “ a^+ ” denotes “ a, a^* ”.

Definition 1 (XGrammar) An XGrammar is denoted by a 6-tuple $\mathbb{G} = (N, E, A, S, P, \Sigma)$, where

- N is a finite set of non-terminal symbols, where $N \subseteq \hat{N}$.
- E is a finite set of element names, where $E \subseteq \hat{E}$.
- A is a finite set of attribute names, where $A \subseteq \hat{A}$.
- S is the set of start symbols, where $S \subseteq N$.
- P is the set of production rules of the form $X \rightarrow x (RE)$, where $X \in N$, $x \in E$, and RE is:

$$RE ::= \epsilon \mid \tau \mid @a \mid Y \mid (RE + RE) \mid (RE, RE) \mid (RE)^? \mid (RE)^* \mid (RE)^+, \text{ where } \tau \in \hat{\tau}, a \in A, Y \in N.$$

We enforce that if a non-terminal symbol X has two production rules: $X \rightarrow x_1(RE_1)$, and $X \rightarrow x_2(RE_2)$, then $x_1 = x_2$. We say that the type of an element x in a document D is X , if $D \in L(\mathbb{G})$, and D can be generated from \mathbb{G} such that the tree rooted at x is generated by the production rule $X \rightarrow x (RE)$.

- Σ is the set of constraints, constraints include the following
 - Attribute type. For any attribute $a \in A$, we define the type $\tau \in \hat{\tau}$ for it. Further if the attribute type is IDREF(S), we define the target type(s), that is the element types the attribute values must point to. This is done as: (a) if the attribute type is IDREF, the target type RE_1 is defined as: $RE_1 ::= X \mid (RE_1) \mid (RE_1 + RE_1)$ (b) if the attribute type is IDREFS, we define the target type RE_2 as: $RE_2 ::= \epsilon \mid X \mid (RE_2) \mid (RE_2 + RE_2) \mid (RE_2, RE_2) \mid (RE_2)^? \mid (RE_2)^* \mid (RE_2)^+$. Here $X \in N$.
 - Key and Unique constraints as: (X, Y, F) , where $X, Y \in N$, and F is a set of path expressions. This specifies that the set of path expressions given by F is a primary key (or unique) for all elements of type Y , that occur as descendants of one element of type X .
 - Foreign-key constraints as: (X_1, Y_1, F_1) references (X_2, Y_2, F_2) , where $X_1, Y_1, X_2, Y_2 \in N$, and F_1, F_2 are sets of path expressions. This specifies that the set of values for path expressions F_1 for an

element of type Y_1 that is a descendant of an element of type X_1 also occurs for the set of path expressions F_2 for an element of type Y_2 that is a descendant of type X_2 .

- *Functional Dependencies* as: $(X, Y, S \rightarrow a)$, where $X, Y \in N$, and S is a set of path expressions and a is a path expression. This specifies that for the set of elements of type Y , that are descendants of an element of type X , the values of S functionally determine the values for a .
- *Multivalued Dependencies* as: $(X, Y, S \twoheadrightarrow a)$ where $X, Y \in N$, and S is a set of path expressions and a is a path expression. This specifies that for the set of elements of type Y , that are descendants of an element of type X , the values of S determine the set of values for a . □

Example 1 An example XGrammar is given in Table 1. We also give an XML document that is valid with respect to this XGrammar. Note the following in this example: Given the document and the XGrammar, we can identify the type (non-terminal symbol) for each element. For example, the type for the element with name *library* is *Library*. □

2.1 Reasoning behind our Constraint Specification

The full reasons behind our constraint specification is outside the scope of this paper, and can be found in [13], we shall briefly outline some of the reasons here.

Our constraint specification is based on experiences from conceptual (ER) models for XML. We observe that non-terminal symbols or types in XGrammar correspond to entity types in ER model. Keys and foreign keys are specified for entity types in ER model, and hence for types in XGrammar. Further when we represent $m : n$ or n -ary relationships in XML, we may need “relative keys”. For example, in Example 1, we have $m : n$ relationship between *Book* and *Author*, which is represented by nesting *Author* within *Book*. In this case, the key for *Author* includes the *book* and *@aname*. This is specified as a relative key as $(Book, Author, \langle @aname \rangle)$, which says that relative to a book, the set of authors are identified by $\langle @aname \rangle$. Further, IDREF(S) represent relationships between types and so they identify target types.

We have represented functional dependencies similar to our key specification. A key is a special case of functional dependency, for example, the key (X, Y, S) is equivalent to the functional dependency $(X, Y, S \rightarrow yid)$, where *yid* is the nodeid for elements of type Y , which will be explained in the next section.

We would further like to remark that our constraint specification is similar in spirit to [10]. It is also worth while to note that our results apply directly to constraint specification scenario where path expressions are used instead of types as in [5, 19].

$N = \{Root, Library, Book, Author\}$ $E = \{root, library, book, author\}$ $S = \{Root\}$ $P = \{Root \rightarrow root(Library^*),$ $Library \rightarrow library(@lname, @address, Book^*)$ $Book \rightarrow book(@title, @loc, Author^*)$ $Author \rightarrow author(@aname, @age)\}$ $\Sigma = \text{Key constraint: } \{(Root, Library, \langle @lname \rangle),$ $(Root, Book, \langle @title \rangle),$ $(Book, Author, \langle @aname \rangle)\}$ <p>Functional Dependencies:</p> $\{(Root, Book, parent :: library/@lname \rightarrow @loc),$ $(Root, Author, @aname \rightarrow @age)$	<pre> <root> <library @lname='EMS' @address='SEAS'> <book @title='T1' @loc='UCLA'> <author @aname='RRM' @age='62' / > <author @aname='CZ' @age='55' / > </book> <book @title='T2' @loc='UCLA'> <author @aname='RRM' @age='62' / > </book> </library> </root> </pre>
--	---

Table 1 Example XGrammar and a valid XML document

3 Abstract Relational Representation

In this section, we will see how an XML representation can be unnested to a flat relational representation. This unnesting will also show fundamental differences between the traditional nested relational model studied and XML. Figure 1 shows how the XML instance in Example 1 is represented in the nested relational model, and the unnesting of this nested relational model as in [17].

We base our unnesting of XML model on XQuery semantics [20]. The main difference is that XML considers each path expression independent of others. For example, if we unnest *library*, and we have two columns, one

corresponding to *book/@title* and other corresponding to *book/author/@aname*, we will have two sets of values corresponding to each of these columns: *book/@title* will return the set $\{T1, T2\}$, and *book/author/@aname* will return the set $\{RRM, CZ\}$. So there will be a tuple in our relational representation corresponding to $\langle T2, CZ \rangle$, which does not exist in the unnesting of the nested relational model as in Figure 1. The astute reader can observe that when there is only single attribute nesting, both the nested relational representation and the corresponding XML representation will produce the same flat relational representation.

Library					
lname	address	Book			
		title	loc	Author	
				aname	age
EMS	SEAS	T1	UCLA	RRM	62
				CZ	55
		T2	UCLA	RRM	62

(a) Nested Relational Representation of

Example 1

Library					
lname	address	title	loc	aname	age
EMS	SEAS	T1	UCLA	RRM	62
EMS	SEAS	T1	UCLA	CZ	55
EMS	SEAS	T2	UCLA	RRM	62

(b) Unnesting of the nested

relational representation

Fig. 1 Nested Relational Representation and the corresponding flat relational representation of the XML instance in Example 1

Let us now define how to obtain our flat relational representation from an XML representation. Given an XGrammar, we create a separate table for each non-terminal symbol (type), say X . For each table, we have columns corresponding to “interesting” attributes and elements of X , which include the id for X , which is xID ¹, all attributes of X , ids corresponding to all child types of X , and any attribute or ids of types that participate in constraints for X , such as key, foreign key, and functional and multivalued dependencies. Further, we give a unique id for each element in the instance. We show an example of our unnesting in Figure 2.

Let us now study some properties of functional and multivalued dependencies for XML based on our abstract relational model. We already mentioned before that a key is just a special case of functional dependency.

For example, the key constraint (X, Y, S) is equivalent to the functional dependency $(X, Y, S \rightarrow yid)$. We will now see some “trivial” functional and multivalued dependencies, these are true for all XGrammars. One can verify the correctness of the following using our abstract relational representation.

- For any two types X and Y we have the trivial functional dependency $(X, Y, p \rightarrow yID)$, where p is a path expression that selects one/multiple descendant elements. This says that a descendant type “functionally determines” the ancestor. This is also mentioned in [1]. For example, in Figure 2, $(Root, Book, author \rightarrow bookID)$ is a trivial functional dependency.
- For any two types X and Y , we have the trivial functional dependency $(X, Y, yID \rightarrow @a)$ where $@a$ is an attribute of Y . We also have $(X, Y, yID \rightarrow a)$ where a is an element/attribute that will occur

¹ We use the convention that the *id* for type X is xID .

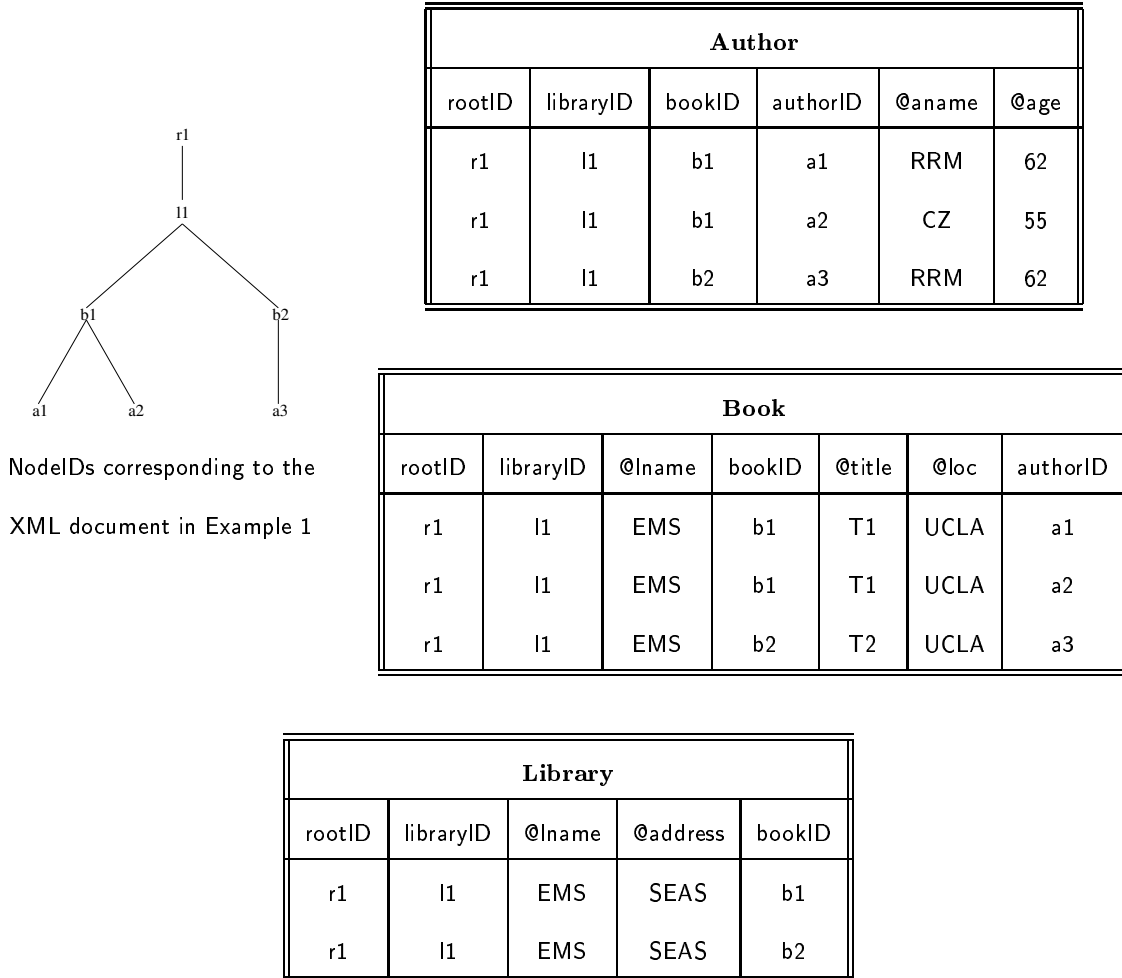


Fig. 2 Unnesting the XML representation in Example 1. The interesting attributes for *Author* are *authorID*, attributes of *Author* which include *@aname*, *@age*, the key for *Author* includes *Root*, so we include *rootID*, *bookID* and *libraryID* as *book* and *library* are in the path from *Root* to *Author*. Similarly the interesting attributes for *Book* include *bookID*, attributes of *Book*, which include *@title* and *@loc*, its child types, which include *authorID*, *rootID* and *libraryID* participate in the key constraint, and we have a functional dependency which includes *@lname*. The interesting attributes for *Library* include *libraryID*, its attributes *@lname*, *@address*, its child types *bookID*, and from the key constraint, *rootID*.

once for *Y*. In Figure 2, the functional dependencies $(Root, Book, bookID \rightarrow @title)$ and $(Root, Book, bookID \rightarrow parent :: library/@lname)$ are trivial. This also says that (X, Y, yID) is a trivial key constraint.

– For any two types *X* and *Y*, we have the trivial multivalued dependency $(X, Y, yID \twoheadrightarrow a)$, where *a* is an

attribute or an element with single/multiple values.

In Figure 2, $(Root, Book, bookID \twoheadrightarrow author)$ and $(Root, Book, bookID \twoheadrightarrow author/@aname)$ are trivial multivalued dependencies.

We can further simplify our representation of functional and multivalued dependencies by the most common scenarios, which is equivalent to the representation

used in [1]. Here the authors specify functional or multivalued dependencies of the form $(S \rightarrow @a)/(S \twoheadrightarrow @a)$. Such a dependency, where $@a$ is an attribute of type X is equivalent to $(Root, X, S \rightarrow @a)/(Root, X, S \twoheadrightarrow @a)$ in our representation, where $Root$ is the type corresponding to the root of the document. Similarly if we have a functional dependency of the form $(S \rightarrow xID)$, this is equivalent to $(Root, X, S \rightarrow xID)$. This is in turn equivalent to $(Root, X, S)$ is a key constraint.

4 Normalization Steps

In this section, we consider how we can remove “anomalous” functional and multivalued dependencies². The intuition is fairly simple and is similar to [1]: if we have an anomalous functional or multivalued dependency of the form $(X, Y, S \rightarrow a)/(X, Y, S \twoheadrightarrow a)$, then we remove a from Y , and move it to a type Z , with key constraint (X, Z, S) . If there is no such type, we create a new type Z . We give the details below. Before we proceed, we would like to make some observations which makes our normalization steps much easier.

Observation 1 *In a functional dependency of the form $(X, Y, S \rightarrow a)$ the set of path expressions in S need to end only in attributes.* \square

We can prove the above observation using the following lemmas.

² A dependency is anomalous if the design is not in a desired normal form because of that dependency. We will see what dependencies are anomalous in the next section.

Lemma 1. *A functional dependency of the form $(X, Y, S \rightarrow a)$, where S contains a path returning one/multiple elements that is a descendant of Y is trivial.*

PROOF. We will use the abstract relational representation to prove the lemma. Let $S = \langle S', p \rangle$, where p is a path expression that returns one/multiple descendant elements. Now consider the relation corresponding to Y in our abstract relational representation with columns (xID, yID, S', p, a) . Now for each value of yID , we may have a set of values for S' , and a set of values for p , but we should have only one value for a (otherwise the functional dependency is violated). Therefore we see that the following functional dependency also holds $(X, Y, yID \rightarrow a)$, and is stricter than $(X, Y, \langle S', p \rangle \rightarrow a)$, because $(X, Y, p \rightarrow yID)$. Now $(X, Y, yID \rightarrow a)$ is a trivial functional dependency. \blacksquare

Lemma 2. *The functional dependency $(X, Y, \langle S', p \rangle \rightarrow a)$, where p is a path expression that returns one/multiple elements that are not descendants or ancestors of Y can be replaced by a functional dependency $(X, Y, \langle S', p' \rangle \rightarrow a)$, where p' is a path expression that returns elements that are ancestors of Y .*

PROOF. We will again use the abstract relational representation to prove the lemma. Let p be of the form $parent :: p_1/parent :: p_2/\dots/parent :: p_n/c_1/c_2/\dots/c_m$. Let the type corresponding to p_n be Z . We shall consider the relation for Y with columns $(xID, yID, zID, S', p, a)$. We know that for each value of yID , we have one value

for xID , and one value for zID , and we could have a set of values for S' and p . We should further have only one value for a . We know that the following functional dependency holds $(X, Y, p \rightarrow zID)$, and the following multivalued dependency holds $(X, Y, zID \twoheadrightarrow p)$. From these, we know that the functional dependency $(X, Y, \langle S', p' \rangle \rightarrow a)$ also holds, and is stricter than the original functional dependency, where $p' = parent :: p_1/parent :: p_2/\dots/parent :: p_n$. We therefore replace $(X, Y, \langle S', p \rangle \rightarrow a)$ with $(X, Y, \langle S', p' \rangle \rightarrow a)$. ■

Lemma 3. *A functional dependency of the form $(X, Y, \langle S, p \rangle \rightarrow a)$, where p is a path expression that returns one/multiple elements that are ancestors of type Z can be replaced by a functional dependency $(A, Y, S \rightarrow a)$, where $A \in \{X, Z\}$ is the ancestor closest to Y .*

PROOF. First observe that $(X, Y, \langle S, p \rangle \rightarrow a)$ is equivalent to $(Root, Y, \langle S, p, p' \rangle \rightarrow a)$, where p' specifies how we can reach X from Y . Now both p and p' produce ancestor elements of Y . If p is a closer ancestor to Y than p' , then we have $(Root, Y, p \rightarrow p')$, and vice versa. We can therefore replace the original functional dependency with $(A, Y, S \rightarrow a)$ where $A \in \{X, Z\}$ is the ancestor closest to Y . ■

The above lemmas prove the observation for functional dependencies. We can use exactly the same arguments as above for the following observation for multivalued dependencies.

Observation 2 *In a multivalued dependency of the form $(X, Y, S \twoheadrightarrow a)$ the set of path expressions in S need to end only in attributes.* □

Now let us examine how we can remove anomalous functional and multivalued dependencies. First let us consider an anomalous functional dependency of the form $(X, Y, S \rightarrow a)$ (a could be a path expression ending in an attribute or an element), we remove it as follows.

- Case 1: a produces one element or attribute
 1. If there is a type Z such that $(X, Z, S \rightarrow zID)$ is a functional dependency, rather (X, Z, S) is a key
 - (a) Remove a from the production rule of the type that produces it presently; add a to the production rule for Z .
 - (b) Add a foreign key constraint (X, Y, S) references (X, Z, S) .
 2. If there is no such type Z , then
 - (a) Create a type Z , and add Z^* to the production rule for X .
 - (b) Add attributes corresponding to path expressions in S to Z , if a path expression in S produces only one attribute, then we add that attribute to Z , if it can produce multiple attributes, then we add a new non-terminal to Z , as Z'^* , and the production rule for Z' produces the attribute.

- (c) Remove a from the production rule of the type that produces it presently; add a to the production rule for Z .
- (d) Add the key constraint (X, Z, S') , where S' is the path expression corresponding to S in Z .
- (e) Add the foreign key constraint (X, Y, S) references (X, Z, S') .

– Case 2: a produces multiple elements or attributes.

In this case the procedure is the same as above, the only difference is we add Z''^* to the production rule for Z , and we add a to the production rule for Z'' .

Anomalous multivalued dependencies are removed similar to Case 2 above, we create a new type Z'' , and add Z''^* to the production rule of Z , and add a to the production rule of Z'' .

Example 2 Consider the functional dependency $(Root, Book, parent :: library/@lname \rightarrow @loc)$. Let us try to remove this, we first check if there is a type Z with key $(Root, Z, @lname)$, and we find that we have such a type, *Library*. We therefore use the above steps and get the XGrammar and document in Table 2. We do not add the foreign key constraint $(Root, Book, parent :: library/@lname)$ references $(Root, Library, @lname)$, because it is redundant in this case.

Example 3 Consider the functional dependency $(Root, Author, @aname \rightarrow @age)$. When we remove this, we

find that we have to create a new type, the resulting XGrammar and document are shown in Table 3.

5 Normal Forms for XGrammar

In this section we shall define normal forms for XGrammar analogous to normal forms for relational model.

Definition 2 *2NF* An XGrammar is said to be in 2NF if and only if we do not have an anomalous functional dependency $(X, Y, S \rightarrow a)$ with key constraint (X, Y, S') , where $S' \supset S$, and $a \notin S'$. \square

Example 4 The XGrammar in Table 1 is in 2NF because we have two functional dependencies, and none of them are anomalous as per the 2NF definition. The functional dependency $(Root, Book, parent :: library/@lname \rightarrow @loc)$, specifies the LHS as $(rootID, @lname)$, whereas the key for *Book* is $(rootID, @title)$. The functional dependency $(Root, Author, @aname \rightarrow @age)$ specifies the LHS as $(rootID, @aname)$, whereas the key for *Author* is $(bookID, @aname)$.

Definition 3 *3NF* An XGrammar is said to be in 3NF if and only if we do not have an anomalous functional dependency $(X, Y, S \rightarrow a)$ with key constraint (X, Y, S') , where $S' \neq S$ and $a \notin S'$. \square

Example 5 The XGrammar in Table 1 is not in 3NF, both the functional dependencies are anomalous with respect to 3NF. We therefore use the steps discussed to remove these anomalous dependencies and we get the XGrammar as in Table 3. \square

$N = \{Root, Library, Book, Author\}$ $E = \{root, library, book, author\}$ $S = \{Root\}$ $P = \{Root \rightarrow root(Library^*),$ $Library \rightarrow library(@lname, @address, @loc,$ $Book^*)$ $Book \rightarrow book(@title, Author^*)$ $Author \rightarrow author(@aname, @age)\}$ $\Sigma = \text{Key constraints: } \{(Root, Library, \langle @lname \rangle),$ $(Root, Book, \langle @title \rangle),$ $(Book, Author, \langle @aname \rangle)\}$ <p style="text-align: center;">Functional Dependencies:</p> $\{(Root, Author, @aname \rightarrow @age)$	<pre> <root> <library @lname='EMS' @address='SEAS' @loc='UCLA'> <book @title='T1'> <author @aname='RRM' @age='62' / > <author @aname='CZ' @age='55' / > </book> <book @title='T2'> <author @aname='RRM' @age='62' / > </book> </library> </root> </pre>
--	---

Table 2 XGrammar and the XML document after removing the functional dependency $(Root, Book, parent :: library/@lname \rightarrow @loc)$ from Table 1

Definition 4 BCNF An XGrammar is said to be in BCNF if and only if we do not have an anomalous functional dependency $(X, Y, S \rightarrow a)$ with key constraint (X, Y, S') , where $S' \neq S$ □

Example 6 The XGrammar in Table 3 has no anomalous functional dependencies, and is therefore in BCNF. □

Definition 5 4NF An XGrammar is said to be in 4NF if and only if we do not have an anomalous functional dependency $(X, Y, S \twoheadrightarrow a)$ with key constraint (X, Y, S') , where $S' \neq S$ □

Example 7 Table 4 shows a classic example of multivalued dependencies and 4NF normal form. When we

consider the first multivalued dependency, $(Root, Cal, course/@num \twoheadrightarrow text/@text)$, we find that we have to create a new type $Cal1$ and add $Cal1^*$ to the production rule for $Root$. We set the production rule for $Cal1$ to be $(@num, Text1^*)$, and add the foreign key constraint: $(Root, Cal, \langle course/@num \rangle)$ references $(Root, Cal1, \langle @num \rangle)$. When we consider the multivalued dependency $(Root, Cal, course/@num \twoheadrightarrow prof/@name)$, we already have $Cal1$ with key $(Root, Cal1, @num)$, so we add $Prof1^*$ to the production rule for $Cal1$. Note that we can remove both $Text$ and $Prof$ from N , because they have empty production rules. □

N	$=$	$\{Root, Cal, Course, Text, Prof\}$
E	$=$	$\{root, cal, course, text, prof\}$
S	$=$	$\{Root\}$
P	$=$	$\{Root \rightarrow root(Cal^*)$
		$Cal \rightarrow cal(Course, Text, Prof)$
		$Course \rightarrow course(@num)$
		$Text \rightarrow text(@title)$
		$Prof \rightarrow prof(@name)$
Σ	$=$	Key constraint: $\{(Cal, Course, \langle @num \rangle),$
		$(Cal, Text, \langle @title \rangle),$
		$(Cal, Prof, \langle @name \rangle)\}$
		Multivalued Dependencies:
		$(Root, Cal, course/@num \twoheadrightarrow text/@text)$
		$(Root, Cal, course/@num \twoheadrightarrow prof/@name)$

N	$=$	$\{Root, Cal, Cal1, Text1,$
		$Prof1, Course, Text, Prof\}$
E	$=$	$\{root, cal, cal1, text1,$
		$prof1, course, text, prof\}$
S	$=$	$\{Root\}$
P	$=$	$\{Root \rightarrow root(Cal^*, Cal1^*)$
		$Cal \rightarrow cal(Course, Text, Prof),$
		$Course \rightarrow course(@num)$
		$Text \rightarrow text(), Prof \rightarrow prof()$
		$Cal1 \rightarrow cal1(@num, Text1^*, Prof1^*),$
		$Text1 \rightarrow text1(@title)$
		$Prof1 \rightarrow prof1(@name)\}$
Σ	$=$	Key constraints:
		$\{(Cal, Course, \langle @num \rangle),$
		$(Root, Cal1, \langle @num \rangle)\}$
		Foreign key Constraints:
		$(Root, Cal, \langle course/@num \rangle)$
		references $(Root, Cal1, \langle @num \rangle)$

Table 4 Example of 4NF Conversion. Note that the types *Text* and *Prof* can be removed from the resulting XGrammar

viewed as flat tables in our abstract relational model. After these, we defined different normal forms corresponding to 2NF, 3NF, BCNF and 4NF, and showed how we can convert our XGrammar to the desired normal form. We also studied some previously studied examples with respect to normal forms for other models, and saw that our normal forms for XML produce desired results. We believe that our work will help an XML database designer

to understand the redundancies present in the design and to come up with better designs.

There are several important problems to be still studied in normalization theory for XML. One important problem is implication (or inference) of functional and multivalued dependencies. This problem is important to verify whether our XML representation belongs to a particular normal form. Our approach in this paper assumed that we have the entire set of functional and multivalued

N	$=$	$\{Root, Course, TakenBy, Student\}$
E	$=$	$\{root, course, takenBy, student\}$
S	$=$	$\{Root\}$
P	$=$	$\{Root \rightarrow root(Course^*)$ $Course \rightarrow course(@cno, @title, TakenBy^*)$ $TakenBy \rightarrow takenBy(Student^*)$ $Student \rightarrow student(@sno, @name, @grade)\}$
Σ	$=$	Key constraints: $\{(Root, Course, \langle @cno \rangle),$ $(Course, Student, \langle @sno \rangle)\}$ Functional Dependencies: $(Root, Student, @sno \rightarrow @name)$

N	$=$	$\{Root, Course, TakenBy, Student, Student1\}$
E	$=$	$\{root, course, takenBy, student, student1\}$
S	$=$	$\{Root\}$
P	$=$	$\{Root \rightarrow root(Course^*, Student1^*)$ $Course \rightarrow course(@cno, @title, TakenBy^*)$ $TakenBy \rightarrow takenBy(Student^*)$ $Student \rightarrow student(@sno, @grade)$ $Student1 \rightarrow student1(@sno, @name)\}$
Σ	$=$	Key constraints: $\{(Root, Course, \langle @cno \rangle),$ $(Root, Student1, \langle @sno \rangle),$ $(Course, Student, \langle @sno \rangle)\}$ Foreign key Constraints: $\{(Course, Student, \langle @sno \rangle)$ references $(Root, Student1, \langle @sno \rangle)\}$

Table 5 Example from [1]. This was not in 3NF, and our normalization produces an XGrammar in 3NF.

dependencies already specified to us, and now we want to normalize our XML representation. Another important research direction is to examine how the study of null inclusion dependencies as in [12] relate to XML.

References

1. M. Arenas and L. Libkin. “A normal form for XML documents”. In *ACM PODS*, Madison, Wisconsin, June. 2002.
2. M. Arenas and L. Libkin. “An information-theoretic approach to normal forms for relational and XML data”. In *ACM PODS*, San Diego, CA, June. 2003.
3. P. V. Biron and A. Malhotra (Eds). “XML Schema Part 2: Datatypes”. W3C Recommendation, May 2001. <http://www.w3.org/TR/xmlschema-2/>.
4. T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds). “Extensible Markup Language (XML) 1.0 (2nd Edition)”. W3C Recommendation, Oct. 2000. <http://www.w3.org/TR/2000/REC-xml-20001006>.
5. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. “Keys for XML”. *Computer Networks*, 39(5):473–487, Aug. 2002.
6. P. P. Chen. “The Entity-Relationship Model”. *ACM Trans. on Database Systems (TODS)*, 1:9–36, 1976.
7. J. Clark and M. Murata (Eds). “RELAX NG Specification”. OASIS Committee Specification,

$N = \{Univ, Hobby, Dept, Student\}$ $E = \{univ, hobby, dept, student\}$ $S = \{Univ\}$ $P = \{Univ \rightarrow univ(Hobby^*)$ $Hobby \rightarrow hobby(@hname, Dept^*)$ $Dept \rightarrow dept(@dname, Student^*)$ $Student \rightarrow student(@sname)\}$ $\Sigma = \text{Key constraints:}$ $\{(Hobby, Student, \langle@sname\rangle)\}$ $\text{Functional Dependencies:}$ $(Univ, Student, @sname \rightarrow$ $parent :: dept/@dname)$	$N = \{Univ, Hobby, Dept, Student, Student1\}$ $E = \{univ, hobby, dept, student, student1\}$ $S = \{Univ\}$ $P = \{Univ \rightarrow univ(Hobby^*, Student1^*)$ $Hobby \rightarrow hobby(@hname, Dept^*)$ $Dept \rightarrow dept(Student^*)$ $Student \rightarrow student(@sname)$ $Student1 \rightarrow student1(@sname, @dname)\}$ $\Sigma = \text{Key constraints:}$ $\{(Hobby, Student, \langle@sname\rangle),$ $(Univ, Student1, \langle@sname\rangle)\}$ $\text{Foreign key Constraints:}$ $(Hobby, Student, \langle@sname\rangle) \text{ references}$ $(Univ, Student1, \langle@sname\rangle)$
--	---

Table 6 Example from [14]. This was not in 3NF, and our normalization produces an XGrammar in 3NF.

- Dec. 2001. <http://www.oasis-open.org/committees/relaxing/spec-20011203.html>.
8. D. W. Embley and W. Y. Mok. "Developing XML Documents with Guaranteed "Good" Properties". In *Int'l Conf. on Conceptual Modeling (ER)*, Yokohama, Japan, Nov. 2001.
9. R. Fagin. "A normal form for relational databases that is based on domains and keys". *ACM Trans. on Database Systems (TODS)*, 6(3):387–415, 1981.
10. W. Fan, G. M. Kuper, and J. Simeon. "A Unified Constraint Model for XML". *Computer Networks*, 39(5):489–505, Aug. 2002.
11. H. Garcia-Molina, J. D. Ullman, and J. Widom. "Database Systems - The Complete Book". Prentice Hall, 2002.
12. M. Levene and G. Loizou. "Null Inclusion Dependencies in Relational Databases.". *Information and Computation*, 136(2):67–108, 1997.
13. M. Mani. "Data Modeling with XML Schemas". Technical Report, UCLA CS, TR# 030027, May. 2003. <http://www.cs.ucla.edu/~mani/publications.html>.
14. W. Y. Mok, Y Ng, and D. W. Embley. "A Normal Form for Precisely Characterizing Redundancy in Nested Relations". *ACM Trans. on Database Systems (TODS)*, 21(1):77–106, Mar. 1996.
15. M. Murata, D. Lee, and M. Mani. "Taxonomy of XML Schema Languages using Formal Language Theory". In *Extreme Markup Languages*, Montreal, Canada, Aug.

- 2001.
16. Z. M. Özsoyoglu and L. Y. Yuan. “A New Normal Form for Nested Relations”. *ACM Trans. on Database Systems (TODS)*, 12(1):111–136, Mar. 1987.
 17. M. A. Roth and H. F. Korth. “The Design of \neg 1NF Relational Databases into Nested Normal Form”. In *ACM SIGMOD*, San Francisco, CA, May. 1987.
 18. M. A. Roth, H. F. Korth, and A. Silberschatz. “Extended Algebra and Calculus for Nested Relational Databases”. *ACM Trans. on Database Systems (TODS)*, 13(4):389–417, Dec. 1988.
 19. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn (Eds). “XML Schema Part 1: Structures”. W3C Recommendation, May 2001. <http://www.w3.org/TR/xmlschema-1/>.
 20. W3C. Xquery working group. <http://www.w3c.org/XML/Query.html>.

A How different are nested relational and XML model

In this appendix, we show an example where a key constraint (functional dependency) holds for nested relational model, whereas it does not hold for the “corresponding” XML model. This is based on the differences in the un-nesting for nested relational and XML models. Consider the XGrammar and a valid XML document shown in Table 7.

The nested relational representation and the corresponding flat relational representation from un-nesting are shown in Tables 8 and 9 respectively. The un-nesting

for *Person* for the XML model is shown in Table 10. Note that the functional dependency ($@art, @rating \rightarrow @name$) holds for the nested relational representation, whereas it does not hold for the XML representation.

N	$= \{Library, Person, Book, Paper, Review\}$
E	$= \{library, person, book, paper, review\}$
S	$= \{Library\}$
P	$= \{Library \rightarrow library (Person^*),$ $Person \rightarrow person (@name, ((@city, @state) + @zip),$ $(Book^* + Paper^*), Review^*),$ $Book \rightarrow book (@btitle, @ISBN, @year^?, @BID),$ $Paper \rightarrow paper(@ptitle, @year^?, @journal^?, @PID),$ $Review \rightarrow review(@art, @rating)\}$
Σ	$=$ Attribute type constraint: $\{@BID::ID, @PID::ID,$ $@art::IDREF\}$

```

<library>
  <person @name='RRM'
    @city='LA' @state='CA'>
    <book @btitle='T1' @ISBN='I1'
      @BID='B1'>
    <book @btitle='T2' @ISBN='I2'
      @BID='B2'>
    <review @art='P1' @rating='9'>
    <review @art='B2' @rating='9'>
  </person>
  <person @name='CZ' @zip='90095'>
    <paper @ptitle='T3' @PID='P1'>
    <review @art='B1' @rating='9'>
    <review @art='B2' @rating='10'>
  </person>
</library>
  
```

Table 7 An XGrammar and a valid XML document

Person										
name	city	state	zip	Book			Paper		Review	
				btitle	ISBN	BID	ptitle	PID	art	rating
RRM	LA	CA		T1	I1	B1			P1	9
				T2	I2	B2			B2	9
CZ			90095				T3	P1	B1	9
									B2	10

Table 8 Nested Relational Representation of XGrammar and document in Table 7

Person										
name	city	state	zip	btitle	ISBN	BID	ptitle	PID	art	rating
RRM	LA	CA	null	T1	l1	B1	null	null	P1	9
RRM	LA	CA	null	T1	l1	B1	null	null	B2	9
RRM	LA	CA	null	T2	l2	B2	null	null	P1	9
RRM	LA	CA	null	T2	l2	B2	null	null	B2	9
CZ	null	null	90095	null	null	null	T3	P1	B1	9
CZ	null	null	90095	null	null	null	T3	P1	B2	10

Table 9 Un-nesting of the nested relational representation of Table 8

Person								
personID	@name	@city	@state	@zip	bookID	paperID	@art	@rating
p1	RRM	LA	CA	null	b1	null	P1	9
p1	RRM	LA	CA	null	b1	null	B2	9
p1	RRM	LA	CA	null	b2	null	P1	9
p1	RRM	LA	CA	null	b2	null	B2	9
p2	CZ	null	null	90095	null	p1	B1	9
p2	CZ	null	null	90095	null	p1	B1	10
p2	CZ	null	null	90095	null	p1	B2	9
p2	CZ	null	null	90095	null	p1	B2	10

Table 10 Un-nesting of Person for the XML representation of Table 7