Available online at www.sciencedirect.com

ScienceDirect

Journal of Computer and System Sciences 73 (2007) 636–647

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

# Dependency-preserving normalization of relational and XML data [☆]

Solmaz Kolahi [*]

*Department of Computer Science, University of Toronto, 10 King's College road, Toronto, ON, Canada, M5S 3G4*

Received 20 January 2006; received in revised form 24 May 2006

Available online 5 December 2006

**Abstract**

Having a database design that avoids redundant information and update anomalies is the main goal of normalization techniques. Ideally, data as well as constraints should be preserved. However, this is not always achievable: while BCNF eliminates all redundancies, it may not preserve constraints, and 3NF, which achieves dependency preservation, may not always eliminate all redundancies. Our first goal is to investigate how much redundancy 3NF tolerates in order to achieve dependency preservation. We apply an information-theoretic measure and show that only prime attributes admit redundant information in 3NF, but their information content may be arbitrarily low. Then we study the possibility of achieving both redundancy elimination and dependency preservation by a hierarchical representation of relational data in XML. We provide a characterization of cases when an XML normal form called XNF guarantees both. Finally, we deal with dependency preservation in XML and show that like in the relational case, normalizing XML documents to achieve non-redundant data can result in losing constraints.
© 2006 Elsevier Inc. All rights reserved.

*Keywords:* Database design; Normalization; Dependency preservation; XML

## 1. Introduction

Database design for relational data is defined as coming up with a "good" way of grouping the attributes of interest into tables, yielding a database schema [1]. Here "good" refers to schemas that prevent the database from storing anomalies. The notion of normalization has a key role in design theory and is a well-studied subject (refer to [2] for a survey). Given a database schema together with a set of dependencies defined over the attributes, normalization is the act of refining the schema into a "better" schema, considering three criteria: preserving the data, preserving the dependencies, and eliminating redundancy. In this paper, we focus on the last two criteria: how do we represent data to have minimum redundancy while preserving all the dependencies?

Normalization algorithms that produce schemas in BCNF guarantee to eliminate the possibility of redundancy. However, for some relational specifications it is not possible to achieve dependency-preserving BCNF relations. Normalizing into 3NF relations is guaranteed to be dependency-preserving; finding the price that we have to pay for this preservation, in terms of redundancy, is the first contribution of this paper. We apply a recently-introduced information-theoretic measure [3] to see where in a database and how much the normal form 3NF allows redundancy. We show

[☆] Part of this work appeared in the Proceedings of the Tenth International Symposium on Database Programming Languages, DBPL '05.
[*] Fax: +1 416 946 7132.
*E-mail address:* solmaz@cs.toronto.edu.

0022-0000/$ – see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2006.10.014

that in a database instance of a 3NF schema, only positions corresponding to prime attributes (members of candidate keys) admit redundant information, but their information content may be arbitrarily low.

Then we study the possibility of achieving both redundancy elimination and dependency preservation by a hierarchical representation of relational data in XML. A paper [4] that addresses a similar problem shows that any arbitrary mapping of a relation into a hierarchical XML document is redundancy-free if and only if the relation is in BCNF. However, the authors do not provide a characterization of cases when a redundancy-free XML document is obtainable from non-BCNF relational data. As our second contribution, we provide a PTIME algorithm that given a relational schema and a set of functional dependencies defined over it, decides if there is a corresponding dependency-preserving XML representation, which does not allow redundant information, and outputs such an XML specification if there is any. There are also papers [5,6] that address the problem of constraint-preserving transformations from XML to relational databases, which is not a subject of interest in this paper.

To do the above transformation, we need to study the design principles of XML documents as well. Defining dependency constraints over the schemas of XML documents, such as DTDs, has been a subject of interest over the past years. The semantics of key constraints [7,8], foreign keys and inclusion constraints [9–12], and functional dependencies [13–18] and their inference, consistency, and complexity issues have been studied. The normalization of XML documents has also been addressed in three papers [13,18,19]. All of them define similar normal forms for XML that do not allow redundancy with respect to a set of dependency constraints. However, the concept of dependency preservation is not addressed in any of the proposed XML normalization techniques. In this paper, we briefly talk about the concept of dependency preservation in the normalization of XML documents to show the necessity of defining a dependency-preserving normal form for XML.

The rest of this paper is organized as follows: Section 2 reviews some basic concepts of relational and XML design theory. In Section 3 we apply an information-theoretic measure of information content to 3NF. In Section 4 we show how to represent relational data in dependency-preserving XML documents that have no redundancy. We deal with XML dependency preservation in Section 5 to motivate the ideas for future research and bring some concluding remarks.

## 2. Preliminaries

### 2.1. Relational databases and normal forms

A relational specification $(\mathcal{R}, \Sigma)$ consists of an $m$-ary relational schema $\mathcal{R}$ and a set of integrity constraints $\Sigma$ defined over $\mathcal{R}$, where $m$ is the number of attributes associated with $\mathcal{R}$ denoted as $sort(\mathcal{R})$. An instance $I$ of $(\mathcal{R}, \Sigma)$, written as $I \in inst(\mathcal{R}, \Sigma)$, is a finite set of $m$-tuples such that $I$ satisfies the constraints in $\Sigma$.

In this paper, we focus on functional dependencies (FDs) and assume that all of the constraints in $\Sigma$ are of the form $X \to Y$, where $X, Y \subseteq sort(R)$. An instance $I$ satisfies $X \to Y$ iff for every two tuples $t_1, t_2 \in I$, $t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$. We write $\Sigma^+$ for the set of all FDs implied by $\Sigma$. For a set of attributes $X$, we write $X^+$ for the set of all attributes $A$ such that $X \to A \in \Sigma^+$.

A relational specification $(\mathcal{R}, \Sigma)$ is in BCNF iff for every non-trivial FD $X \to Y \in \Sigma$, we have $X \to sort(\mathcal{R}) \in \Sigma^+$ ($X$ is a key). A candidate key is a key whose proper subsets are not keys. Specification $(\mathcal{R}, \Sigma)$ is in 3NF iff for every non-trivial FD $X \to A$, $X$ is a key, or $A$ is a member of a candidate key ($A$ is prime).

Given $(\mathcal{R}, \Sigma)$, there are known algorithms (see [1]) that decompose the schema into $(\mathcal{R}_1, \Sigma_1), \ldots, (\mathcal{R}_n, \Sigma_n)$, such that $\bigcup_{i \in [1,n]} sort(\mathcal{R}_i) = sort(\mathcal{R})$, and for each $i \in [1, n]$, $sort(\mathcal{R}_i) \neq \emptyset$, $\Sigma_i = \pi_{\mathcal{R}_i}(\Sigma^+)$, and $(\mathcal{R}_i, \Sigma_i)$ is in BCNF (or 3NF). BCNF decompositions guarantee to produce relations that do not store any redundant data, while 3NF decompositions may not produce non-redundant relations, but they guarantee to preserve all the FDs, i.e. $\Sigma$ is equivalent to $\bigcup_{i \in [1,n]} \Sigma_i$.

### 2.2. XML documents and normal forms

#### 2.2.1. DTDs and XML trees

For a formal definition of a DTD, assume that we have the following disjoint sets: *El* of element names, *Att* of attribute names, which start with the symbol @, *Str* of possible values of string-valued attributes.

A *DTD* (*Document Type Definition*) $D$ is defined to be $D = (E, A, P, R, r)$, where $E \subseteq El$ is a finite set of element types, $A \subseteq Att$ is a finite set of attributes, $P$ is a set of rules $\tau \to P_\tau$ for each $\tau \in E$, where $P_\tau$ is a regular expression over $E - \{r\}$, $R$ assigns a subset of $A$ to each element $\tau \in E$, and $r \in E$ is the root.

An *XML tree* is a finite rooted directed tree $T = (N, G)$, where $N$ is the set of nodes, and $G$ is the set of edges, together with a labeling function $\lambda : N \to El$ and an attribute function $\rho_{@a} : N \to Str$ for each $@a \in Att$. We say tree $T$ conforms to DTD $D = (E, A, P, R, r)$, written as $T \models D$, if the root of $T$ is labeled $r$, and for every $x \in N$ with $\lambda(x) = a$, the word $\lambda(x_1) \dots \lambda(x_n)$ is in the language defined by $P_a$ where $x_1, \dots, x_n$ are children of $x$ in order, $@l \in R(a)$ iff the function $\rho_{@l}$ is defined on $x$.

### 2.2.2. *Functional dependencies for XML*

Given a DTD $D = (E, A, P, R, r)$, an *element path* $q$ is a word in the language $E^*$, and an *attribute path* is a word of the form $q.@l$, where $q \in E^*$ and $@l \in A$. An element path $q$ is consistent with $D$ if there is a tree $T \models D$ that contains a node reachable by $q$; if the nodes reachable by $q$ have attribute $@l$, then the attribute path $q.@l$ is consistent with $D$. The set of all paths consistent with a DTD $D$ is denoted by $paths(D)$. The last element type that occurs on a path $q$ is called $last(q)$.

Given an XML tree $T = (N, G)$ such that $T \models D$, a *tree tuple* [13] is a subtree of $T$ rooted at $r$ containing at most one occurrence of every path. Intuitively, the set of all tree tuples in $T$ forms a relational representation of $T$. Formally, a tree tuple is a mapping $t : paths(D) \to N \cup Str \cup \{\perp\}$ such that if for an element path $q$ whose last letter is $a$, we have $t(q) \neq \perp$, then $t(q) \in N$ and $\lambda(t(q)) = a$; if $q'$ is a prefix of $q$, then $t(q') \neq \perp$ and $t(q')$ lies on the path from the root to $t(q)$ in $T$; if $@l$ is defined for $t(q)$ and its value is $v \in Str$, then $t(q.@l) = v$.

A *functional dependency* over a DTD $D$ [13] is an expression of the form $\{q_1, \dots, q_n\} \to q$, where $q, q_1, \dots, q_n \in paths(D)$. A tree $T$ satisfies an FD $\{q_1, \dots, q_n\} \to q$ if for any two tree tuples $t_1, t_2$ in $T$, whenever $t_1(q_i) = t_2(q_i) \neq \perp$ for all $i \in [1, n]$, then $t_1(q) = t_2(q)$.

### 2.2.3. *XNF: An XML normal form*

Given a DTD $D$ and a set $\Sigma$ of FDs over $D$, $(D, \Sigma)^+$ is the set of all FDs implied by $(D, \Sigma)$. An FD is called *trivial* if it belongs to $(D, \emptyset)^+$. We say that $(D, \Sigma)$ is in *XML normal form* (*XNF*) [13] if for every non-trivial FD $X \to q.@l$ in $(D, \Sigma)^+$, the FD $X \to q$ is also in $(D, \Sigma)^+$. This normal form generalizes BCNF for XML documents and disallows any redundancy caused by FDs in the document.

## 3. Relational third normal form revisited

The amount of information provided by each cell of a relational database with respect to a set of integrity constraints can be determined using an information-theoretic measure that has been introduced recently [3]. Using this measure, it is known that the information content of every cell in an instance of a BCNF relational schema is maximum.

In this section, we characterize the relational 3NF using this measure and show that in an instance of a relational schema in 3NF, only for the cells corresponding to prime attributes the information content can be less than maximum, meaning that they store redundant information. However, the redundancy of such cells can be arbitrarily high. This is the price that we have to pay in 3NF decompositions to guarantee preservation of FDs.

### 3.1. *Information-theoretic measure of information content*

Here, we briefly review the information-theoretic measure that will be used in the rest of this section and refer the reader to [3] for more details. This measure assigns to each position in a database a number in [0, 1]; the closer it is to 1, the less redundancy the position carries. Intuitively, $\mathrm{INF}_I(p|\Sigma)$ measures the average information provided by position $p$ with respect to constraints $\Sigma$, given all possible ways of removing values in the instance $I$.

Let $\mathcal{R}$ be a relational schema, $\Sigma$ a set of constraints, and $I \in inst(\mathcal{R}, \Sigma)$. The set of positions in $I$, $Pos(I)$, is defined as $\{(\mathcal{R}, t, A) \mid t \in I \text{ and } A \in sort(\mathcal{R})\}$. Let $n = |Pos(I)|$, and suppose each position in $Pos(I)$ is assigned a unique number $p \in [1, n]$. When the active domain of instance $I$ is contained in $[1, k]$, the information content of a position $p \in Pos(I)$ with respect to the set of constraints $\Sigma$, written as $\mathrm{INF}_I^k(p|\Sigma)$, is informally defined as follows. Let $X \subseteq Pos(I) - \{p\}$. Suppose the values in those positions are lost, and someone restores them from the set $[1, k]$; we measure how much information this gives us about the value of $p$, by computing an entropy of a certain distribution. Then $\mathrm{INF}_I^k(p|\Sigma)$ is the average of such entropy over all sets $X \subseteq \mathit{Pos}(I) - \{p\}$.

To define the measure more formally, let $\Omega(I, p)$ be the set of $2^{n-1}$ vectors $(a_1, \ldots, a_{p-1}, a_{p+1}, \ldots, a_n)$ such that for every $i \in [1, n] - \{p\}$ $a_i$ is either a variable $v_i$ or the value in the $i$th position of $I$. Given a vector $\bar{a} \in \Omega(I, p)$, the conditional entropy $P(a|\bar{a})$ characterizes how likely $a$ is to occur in position $p$, if some values are removed from $I$ according to the tuple $\bar{a}$. Let $I_{(a,\bar{a})}$ be a table obtained from $I$ by putting $a$ in position $p$ and $a_i$ in position $i$ for $i \neq p$. Then $SAT_\Sigma^k(I_{(a,\bar{a})})$ is defined as the set of all substitutions $\sigma : \bar{a} \to [1, k]$ such that $\sigma(I_{(a,\bar{a})}) \models \Sigma$ and $|\sigma(I_{(a,\bar{a})})| = |I|$. The probability $P(a|\bar{a})$ is defined as:

$$P(a|\bar{a}) = \frac{|SAT_\Sigma^k(I_{(a,\bar{a})})|}{\sum_{b\in[1,k]} |SAT_\Sigma^k(I_{(b,\bar{a})})|}.$$

The measure of the amount of information in position $p$ is then defined as:

$$\mathrm{INF}_I^k(p|\Sigma) = \sum_{\bar{a}\in\Omega(I,p)} \left( P(\bar{a}) \sum_{a\in[1,k]} P(a|\bar{a}) \log \frac{1}{P(a|\bar{a})} \right),$$

where $P(\bar{a}) = 1/2^{n-1}$ since we consider a uniform distribution on $\Omega(I, p)$. For the case of infinite domain, the measure $\mathrm{INF}_I(p|\Sigma)$ is defined as:

$$\lim_{k\to\infty} \frac{\mathrm{INF}_I^k(p|\Sigma)}{\log k}.$$

This measure of information content can be used to distinguish a good design from a bad one. A database specification $(\mathcal{R}, \Sigma)$ is defined as *well-designed* if for every $I \in \mathit{inst}(\mathcal{R}, \Sigma)$ and every $p \in \mathit{Pos}(I)$, $\mathrm{INF}_I(p|\Sigma) = 1$. That is, every position of every instance should have the maximum information, and no redundancies are allowed. It is known that if $\Sigma$ only contains FDs, then $(\mathcal{R}, \Sigma)$ is well-designed if and only if it is in BCNF. However, we cannot always achieve a well-designed database without loosing some FDs. That is why another normal form, 3NF, is often used that allows redundancies to some extent in order to preserve all the FDs.

### *3.2. Characterizing 3NF*

We now apply the criterion of being well-designed to relational third normal form. We would like to know where and to what extent 3NF allows a database to store redundant information. The results in this section show that although 3NF disallows redundancy in positions corresponding to non-prime attributes, it does not impose any upper bound for the redundancy carried by a position corresponding to a prime attribute if we consider arbitrarily large schemas. In other words, the information content of a position in a 3NF relation can be arbitrarily small, which means the position can carry arbitrarily high amount of redundancy.

The following theorem says that only positions corresponding to prime attributes can store redundant information when we have 3NF.

**Theorem 1.** *Let $\Sigma$ be a set of FDs over a relational schema $\mathcal{R}$. The specification $(\mathcal{R}, \Sigma)$ is in 3NF if and only if for every $I \in \mathit{inst}(\mathcal{R}, \Sigma)$ and $p = (\mathcal{R}, t, A)$ in $\mathit{Pos}(I)$, $\mathrm{INF}_I(p|\Sigma) < 1$ implies $A$ is a prime attribute.*

**Proof.** ($\Rightarrow$) Suppose $(\mathcal{R}, \Sigma)$ is in 3NF and $I \in \mathit{inst}(\mathcal{R}, \Sigma)$. If $\mathrm{INF}_I(p|\Sigma) < 1$ for some $p = (\mathcal{R}, t, A)$ in $\mathit{Pos}(I)$, it means that there is redundant information in position $p$ [3]. Since we assume $\Sigma$ only contains FDs, there must be an FD $X \to A \in \Sigma^+$ and another tuple $t'$ in $I$, such that $t[X] = t'[X]$ and therefore $t[A] = t'[A]$. This can only happen when $X$ is not a key. Thus, $A$ should be a prime attribute since $(\mathcal{R}, \Sigma)$ is in 3NF.

($\Leftarrow$) Suppose $(\mathcal{R}, \Sigma)$ is not in 3NF, and there is an FD $X \to A \in \Sigma^+$, such that $X$ is not a key for $\mathcal{R}$ and $A$ is not prime. We show that there is an instance $I \in \mathit{inst}(\mathcal{R}, \Sigma)$ and position $p = (\mathcal{R}, t, A) \in \mathit{Pos}(I)$ such that $\mathrm{INF}_I(p|\Sigma) < 1$. Let $I$ be an instance of $(\mathcal{R}, \Sigma)$ containing two tuples $t_1$, $t_2$ defined as follows. For every $B \in \mathit{sort}(R)$, $t_1[B] = 1$. If $B \in X^+$, $t_2[B] = 1$, otherwise $t_2[B] = 2$. It is easy to see that $I$ satisfies $\Sigma$, and for position $p = (\mathcal{R}, t_1, A)$ we have $\mathrm{INF}_I(p|\Sigma) < 1$. This contradicts the assumption that for every non-prime attribute $A$ and position $p = (\mathcal{R}, t, A)$, we have $\mathrm{INF}_I(p|\Sigma) = 1$. □

Next we show that the redundancy of positions corresponding to prime attributes can be arbitrarily high when we have 3NF. The following theorem, which is the main result of this section, says that for any $\varepsilon$, however small, we can find a position in a database instance whose information content is less than $\varepsilon$, and yet the database schema satisfies 3NF.

**Theorem 2.** *For every $\varepsilon \in (0, 1]$, there exists a relational schema $\mathcal{R}$, a set of FDs $\Sigma$ over $\mathcal{R}$, an instance $I \in inst(\mathcal{R}, \Sigma)$, and position $p \in Pos(I)$ such that $(\mathcal{R}, \Sigma)$ is in 3NF, and* $\mathrm{INF}_I(p|\Sigma) < \varepsilon$.

**Proof.** Let $\mathcal{R} = (A, B, B_1, \ldots, B_m)$, and $\Sigma = \{AB \to B_1 \ldots B_m,\ B_1 \to A, \ldots, B_m \to A\}$. Suppose the domain of each attribute of $\mathcal{R}$ is $\mathbb{N}$. It is easy to see that $(\mathcal{R}, \Sigma)$ is in 3NF. We define a $k$ tuple instance $I$ of $(\mathcal{R}, \Sigma)$ as follows: for any tuple $t_j \in I$, $j \in [1, k]$, $t_j[B] = j$, and $t_j[A] = t_j[B_1] = \cdots = t_j[B_m] = 1$.

Now consider position $p = (\mathcal{R}, t_1, A) \in Pos(I)$. If we lose the value in this position, there are many other tuples that can help restore it considering the FDs. It is shown in [20] that for such an instance we have:

$$\mathrm{INF}_I(p|\Sigma) = \sum_{i=0}^{m} \frac{\binom{m}{i}(1 + 2^{-i})^{k-1}}{2^{k+m-1}}.$$

Using the following calculations, we show that for any $\varepsilon > 0$, we can make the information content of position $p$ smaller than $\varepsilon$ by choosing the appropriate $m$ and $k$:

$$\begin{aligned}
\mathrm{INF}_I(p|\Sigma) &= \frac{1}{2^{k+m-1}} \sum_{i=0}^{m} \binom{m}{i} \left(1 + 2^{-i}\right)^{k-1} \\
&= \frac{1}{2^{k+m-1}} \left(2^{k-1} + \sum_{i=1}^{m} \binom{m}{i} \left(1 + 2^{-i}\right)^{k-1}\right) \\
&< \frac{1}{2^{k+m-1}} \left(2^{k-1} + 2^m \left(1 + 2^{-1}\right)^{k-1}\right) \\
&= \left(\frac{1}{2}\right)^m + \left(\frac{3}{4}\right)^{k-1} \\
&< \varepsilon
\end{aligned}$$

as long as $m > -\log_2 \varepsilon + 1$ and $k > \log_{3/4} \varepsilon/2 + 1$, which proves Theorem 2. □

## 4. Dependency-preserving redundancy-free conversion of relational data into XML documents

In designing relational databases, relations are sometimes decomposed to avoid redundancies and update anomalies. Losslessness, dependency preservation, and redundancy elimination are the three desired properties for each decomposition. However, it is not always possible to achieve all the three: while BCNF decomposition eliminates all redundancies, it may not preserve dependencies, and 3NF decomposition, which achieves dependency preservation, may produce relations that store data with high degrees of redundancy, as we have seen in Section 3.

In this section, we want to show that for some relational specifications, it is possible to produce an FD-preserving XML representation, which is in XNF and hence avoids redundancies and update anomalies. This way we can take advantage of good properties of BCNF and 3NF that are not achievable together in relational representation.

**Example 3.** Consider the relational schema $\mathcal{R} = (A, B, C)$, with FDs $\mathcal{F} = \{AB \to C$ and $C \to B\}$. This is a classical example of a relational schema that does not have any FD-preserving BCNF decomposition. We can however convert it into a DTD $D = (E, A, P, R, r)$ and a set of FDs $\Sigma$ such that $(D, \Sigma)$ does not allow redundant data:

- $E = \{r, A, B, C\}$.
- $A = \{@a, @b, @c\}$.
- $P(r) = B^*$, $P(B) = A^*$, $P(A) = C^*$, $P(C) = \epsilon$.
- $R(r) = \emptyset$, $R(A) = \{@a\}$, $R(B) = \{@b\}$, $R(C) = \{@c\}$.

| A | B | C |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 2 | 2 |
| 2 | 1 | 1 |
| 3 | 2 | 2 |

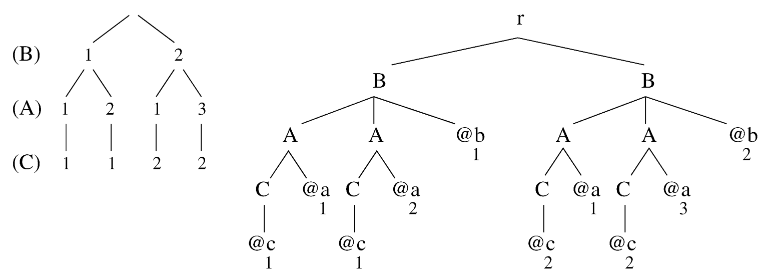

Fig. 1. Conversion of relational data into a redundancy-free XML document.

- $\Sigma = \{r.B.@b \to r.B,$
  $\{r.B, r.B.A.@a\} \to r.B.A,$
  $\{r.B.A, r.B.A.C.@c\} \to r.A.B.C,$
  $\{r.B.A.@a, r.B.@b\} \to r.B.A.C.@c,$
  $r.B.A.C.@c \to r.B.@b\}.$

Note that $\epsilon$ is the empty regular expression. This conversion is visualized in Fig. 1. Note that the first three FDs are the result of the nested structure of the document. The second FD for example means that given a $B$ element, a value of attribute $@a$ uniquely determines one of the children, which is an $A$ element. The last two FDs are the translations of relational FDs in $\mathcal{F}$. From the set of FDs $\Sigma$, we can easily infer the following two FDs: $\{r.B.A.@a, r.B.@b\} \to r.B.A.C$ and $r.B.A.C.@c \to r.B$. Thus, $(D, \Sigma)$ is in normal form XNF and hence does not allow redundancy.

In the above example, the correct hierarchical ordering of elements in the DTD makes it possible to have an XML representation in XNF from a non-BCNF relation. Since for each relational FD there is a path in the DTD containing all the participating attributes, the representation is also FD-preserving. We now formally define this hierarchical translation for an arbitrary relational specification and investigate the conditions that a relational specification needs to satisfy in order to have a hierarchical XML representation in XNF.

**Definition 4** *(Hierarchical translation of relational schema).* Let $\mathcal{R} = (A_1, \ldots, A_m)$ be a relational schema and $\mathcal{F}$ be a set of FDs defined over it. We define DTD $D = (E, A, P, R, r)$ and the set of XML FDs $\Sigma$ as a hierarchical translation of $(\mathcal{R}, \mathcal{F})$ as follows:

- $E = \{\tau_1, \ldots, \tau_m\} \cup \{r\}$; each element $\tau_i$ corresponds to a relational attribute $A_i \in \mathcal{R}$.
- $A = \{@l_1, \ldots, @l_m\}$.
- $R(r) = \emptyset$ and for $i \in [1, m]$, $R(\tau_i) = \{@l_i\}$; each element has an attribute to store a value.
- Elements in $E$ form an ordering $\tau_{\pi_1}, \ldots, \tau_{\pi_m}$ such that $P(r) = \tau_{\pi_1}^*$, $P(\tau_{\pi_m}) = \epsilon$, and for every $i \in [1, m)$, $P(\tau_{\pi_i}) = \tau_{\pi_{i+1}}^*$.
- The FD $r.\tau_{\pi_1}.@l_{\pi_1} \to \tau_{\pi_1}$ is in $\Sigma$. Also for each $i \in [2, m]$, there is an FD $\{p, p.\tau_{\pi_i}.@l_{\pi_i}\} \to p.\tau_{\pi_i}$ in $\Sigma$, where $p$ is the path from the root to the parent of $\tau_{\pi_i}$.
- For every FD $X_1 \to X_2 \in \mathcal{F}$, there is a corresponding FD $S_1 \to S_2 \in \Sigma$, such that for every attribute $A_i$ in $X_1$ ($X_2$), there is a path $p.\tau_i.@l_i$ in $S_1$ ($S_2$), where $\tau_i$ corresponds to $A_i$ and $p$ is the path from the root to the parent of $\tau_i$.

Note that this translation is dependency-preserving in the following sense: suppose we translate relational specification $(\mathcal{R}, \mathcal{F})$ into a hierarchical XML specification $(D, \Sigma)$. Then any instance $I$ of relation $\mathcal{R}$ satisfies $\mathcal{F}$ iff its hierarchical representation $T$ that conforms to $D$ satisfies $\Sigma$. In other words, none of the FDs in $\mathcal{F}$ will be lost.

Now let $\mathcal{R} = (A_1, \ldots, A_m)$ be a relational schema, $\mathcal{F}$ a set of FDs over $\mathcal{R}$, and $(D, \Sigma)$ a hierarchical translation of $(\mathcal{R}, \mathcal{F})$. Then:

**Theorem 5.** *The XML specification $(D, \Sigma)$ is in XNF iff for every FD $X \to p.@l \in (D, \Sigma)^+$ and every prefix $q$ of path $p$, it is the case that $X \to q.@m \in (D, \Sigma)^+$, where $R(last(q)) = \{@m\}$.*

**Proof.** ($\Rightarrow$) Suppose $(D, \Sigma)$ is in XNF and the FD $X \to p.@l$ is in $(D, \Sigma)^+$. Then the FD $X \to p$ is also in $(D, \Sigma)^+$. Let $T$ be an arbitrary XML tree conforming to $D$ and satisfying $\Sigma$. For every two tree tuples $t_1, t_2$ in $T$, if $t_1$ and $t_2$ agree on all paths in $X$ ($t_1(q') = t_2(q') \neq \bot$ for all $q' \in X$), then $t_1(p) = t_2(p) = v$. Trivially, $t_1(v') = t_2(v')$ for every node $v'$ that is an ancestor of $v$ in tree $T$, so for every path $q$ that is a prefix of $p$ and every attribute $@m$ defined for $last(q)$, $T$ satisfies the FDs $X \to q$ and $X \to q.@m$. Therefore, $X \to q.@m$ is in $(D, \Sigma)^+$.

($\Leftarrow$) The proof of the other direction follows from the FDs resulting from hierarchical representation of relational attributes. Suppose an FD $X \to p.@l$ is in $(D, \Sigma)^+$. Then for every prefix $q$ of $p$ and the attribute $@m$ defined for $last(q)$ the FD $X \to q.@m$ is also in $(D, \Sigma)^+$. Let $T$ be an arbitrary XML tree conforming to $D$ and satisfying $\Sigma$. If two tree tuples $t_1, t_2$ from $T$ agree on all the attributes of elements from the root to $last(p)$, they will agree on the nodes corresponding to element types from the root to $last(p)$ as well. This is because of the FDs of the form $\{p, p.\tau_i.@l_i\} \to p.\tau_i$ that are added to $\Sigma$ during the construction of $(D, \Sigma)$. Therefore, for every path $q$ that is a prefix of $p$, the FD $X \to q$ is in $(D, \Sigma)^+$. In particular, $X \to p \in (D, \Sigma)^+$, and hence $(D, \Sigma)$ is in XNF. $\square$

Since every attribute path $p.@l \in paths(D)$ represents exactly one relational attribute, the condition of Theorem 5 for $(D, \Sigma)$ to be in XNF translates to the following condition for $(\mathcal{R}, \mathcal{F})$.

**Corollary 6.** *$(\mathcal{R}, \mathcal{F})$ has a redundancy-free hierarchical translation into XML iff we can put the attributes of $\mathcal{R}$ in order $A_{\pi_1}, \ldots, A_{\pi_m}$ such that for every non-trivial FD $X \to A_{\pi_i} \in \mathcal{F}^+$ and every $j < i$, the FD $X \to A_{\pi_j}$ is also in $\mathcal{F}^+$.*

**Example 7.** Consider the following functional dependencies over the relational schema $\mathcal{R} = (A, B, C, D, F)$:

$ABCD \to F,$

$FD \to A,$

$FC \to B.$

Since none of the FDs $FC \to A$ or $FD \to B$ hold, we cannot put the attributes in the desired order, and the schema does not have an XNF hierarchical translation.

Let $\mathcal{F}_{\min} = \{X_1 \to A_1, \ldots, X_k \to A_k\}$ denote the minimal cover of the FDs over the relational schema $\mathcal{R}$. In order to find the appropriate order of attributes, we shall first compute the intersection of the closures of all $X_i$'s ($i \in [1, k]$). If the intersection is empty, there is no hierarchical translation for this relational schema in XNF. If not, we output the attributes in the intersection in an arbitrary order as the first elements of the ordering. We remove from $\mathcal{F}_{\min}$ the FDs whose right-hand sides are already in the output. Then we repeat computing the intersection of closures of the left-hand sides of the remaining FDs until there is no FD left or all the attributes are in the output. This procedure is described in the algorithm in Fig. 2. Note that if $\mathcal{F}_{\min} = \emptyset$, we can output the attributes of $\mathcal{R}$ in any arbitrary order.

Some relational specifications do not satisfy the condition of Corollary 6. However, there might be an FD-preserving decomposition of them such that each of the decomposed schemas can be hierarchically translated into an XML specification in XNF. Suppose $(D_1, \Sigma_1), \ldots, (D_n, \Sigma_n)$ are the XML translations of the decomposed relations as described above. We can combine all the DTDs into a single DTD by concatenating all the regular expressions assigned to their roots and assign the resulting expression to the new root. Then we have to take the union of element types, attributes, and FDs. Note that we assume the sets of element types of $D_1, \ldots, D_n$ are disjoint. This can be seen in the following example and is formally described in Section 4.1. It is easy to observe that the combined DTD will not violate XNF since every $(D_i, \Sigma_i)$, $i \in [1, n]$, is in XNF, and the sets $paths(D_1), \ldots, paths(D_n)$ are disjoint.

**Example 8.** Consider the following set $\mathcal{F}$ of functional dependencies over the relational schema $\mathcal{R} = (A, B, C, D, F, G)$:

$ABCD \to FG,$

$DF \to A,$

$G \to B,$

$DG \to A.$

Consider a dependency-preserving decomposition for this schema as follows: $\mathcal{R}_1 = (A, B, C, D, F)$ with FDs $\mathcal{F}_1 = \{ABCD \to F,\ DF \to A\}$, and $\mathcal{R}_2 = (A, B, C, D, G)$ with FDs $\mathcal{F}_2 = \{ABCD \to G,\ G \to B, DG \to A\}$. A possible XML specification in XNF would include DTD $D = (E, A, P, R, r)$ as follows. The set of FDs $\Sigma$, omitted here, consists of the FDs resulting from the hierarchical translation and the FDs corresponding to the relational FDs. It can be easily verified that $(D, \Sigma)$ is in XNF.

- $E = \{r, A, B, C, D, F, A', B', C', D', G'\}$.
- $A = \{@a, @b, @c, @d, @f, @g\}$.
- $P(r) = A^*B'^*$, $P(A) = D^*$, $P(D) = F^*$, $P(F) = B^*$, $P(B) = C^*$, $P(C) = \epsilon$, $P(B') = G'^*$, $P(G') = A'^*$, $P(A') = D'^*$, $P(D') = C'^*$, $P(C') = \epsilon$.
- $R(r) = \emptyset$, $R(A) = R(A') = \{@a\}$, $R(B) = R(B') = \{@b\}$, $R(C) = R(C') = \{@c\}$, $R(D) = R(D') = \{@d\}$, $R(F) = \{@f\}$, $R(G) = \{@g\}$.

Note that the original schema does not have a hierarchical translation in XNF since none of $DF \to B$ or $G \to A$ hold, so the decomposition is necessary. We will later see in Example 9 that how we come up with the above hierarchical ordering of elements in the DTD.

### 4.1. Algorithm

Given a relational specification $(\mathcal{R}, \mathcal{F})$, the algorithm in Fig. 2 decides, in polynomial time in the size of $(\mathcal{R}, \mathcal{F})$, whether there is a redundancy-free hierarchical XML representation for it and produces an XML specification $(D, \Sigma)$ if there is one.

First, the minimal cover of the FD set is computed. Then a 3NF decomposition is done based on the minimal cover (see [1]). This decomposition has to be FD-preserving and lossless, so we may add a relation containing attributes of a candidate key. The algorithm then finds the right ordering of attributes for each decomposed relation as described previously. Once an ordering is found, it should be attached to the DTD that is being constructed incrementally. This

```
Input: Relational schema $\mathcal{R}$ and set of FDs $\mathcal{F}$.
Output: Either $(D, \Sigma)$ in XNF or "No XNF Representation."

Initialize $(D, \Sigma)$ with only a root $r$;
Compute $\mathcal{F}_{\min}$ as a minimal cover of $\mathcal{F}$;
Let $(\mathcal{R}_1, \mathcal{F}_1), \ldots, (\mathcal{R}_n, \mathcal{F}_n)$ be a lossless dependency-preserving 3NF
decomposition of $(\mathcal{R}, \mathcal{F})$ based on $\mathcal{F}_{\min}$;
for $i := 1$ to $n$ do
  if there is no FD in $\mathcal{F}_i$ then
    $O_i :=$ an arbitrary ordering of attributes in $\mathcal{R}_i$;
  else
    Compute $X_i^{j+}$ for all FD $X_i^j \to A_i^j$ in $\mathcal{F}_i$;
    $X :=$ attributes in $\mathcal{R}_i$;
    $O_i :=$ empty ordering;
    while $X \neq \emptyset$ do
      if no FD in $\mathcal{F}_i$ has an attribute in $X$ on the right-hand side then
        $Y := X$;
      else
        $Y := (\bigcap_{A_i^j \in X} X_i^{j+}) \cap X$;
      if $Y = \emptyset$ then
        return "No XNF Representation";
      else
        Append attributes in $Y$ to the ordering $O_i$;
        $X := X - Y$;
  $(D, \Sigma) := attach((D, \Sigma),\ O_i,\ \mathcal{F}_i)$;
return $(D, \Sigma)$;
```

Fig. 2. FD-preserving translation of relational data into redundancy-free XML documents.

$$\mathcal{R} = (A, B, C, D, F, G)$$
$$\mathcal{F} = \{ABCD \to FG,\ \boldsymbol{DF \to A},\ \boldsymbol{G \to B},\ DG \to A\}$$

| | |
|---|---|
| $\mathcal{R}_1 = (A, B, C, D, F)$ | $\mathcal{R}_2 = (A, B, C, D, G)$ |
| $\mathcal{F}_1 = \{ABCD \to F,\ DF \to A\}$ | $\mathcal{F}_2 = \{ABCD \to G,\ G \to B,\ DG \to A\}$ |
| (1) $(ABCD)^+ \cap (DF)^+ = \{ADF\}$ **ordering: $\boldsymbol{A, D, F, \ldots}$** | (1) $(ABCD)^+ \cap G^+ \cap (DG)^+ = \{B, G\}$ **ordering: $\boldsymbol{B, G, \ldots}$** |
| (2) no FDs left. **ordering: $\boldsymbol{A, D, F, B, C}$** | (2) $(DG)^+ = \{A, B, D, G\}$ **ordering: $\boldsymbol{B, G, A, D, \ldots}$** |
| | (3) no FDs left. **ordering: $\boldsymbol{B, G, A, D, C}$** |

Fig. 3. Finding the orderings of attributes using the algorithm in Section 4.1.

is done by the operator *attach*, which given an XML specification $(D, \Sigma)$, an ordering of relational attributes $O_i$ and a set of FDs $\mathcal{F}_i$ over it, updates $(D, \Sigma)$ by performing the following steps:

- For each $j \in [1, |O_i|]$, create a fresh element type $\tau_j$ corresponding to $j$th attribute in $O_i$ and a fresh attribute name $@l_j$. Then assign $R(\tau_j) := \{@l_j\}$.
- Update $P(r) := P(r).\tau_1^*$, and for each $j \in [1, |O_i|)$, assign $P(\tau_j) := \tau_{j+1}^*$. Assign $P(\tau_{|O_i|}) := \epsilon$.
- Add to $\Sigma$ the FDs $r.\tau_1.@l_1 \to r.\tau_1$ and $\{p, p.\tau_j.@l_j\} \to p.\tau_j$ for each $j \in (1, |O_i|]$, where $p$ is the path from the root to the parent of $\tau_j$.
- Translate each FD in $\mathcal{F}_i$ into an XML FD by finding the corresponding DTD elements and add it to $\Sigma$.

**Example 9.** Figure 3 shows how the algorithm works on relational specification $(\mathcal{R}, \mathcal{F})$ of Example 8. The two FDs written in bold are the reason that we cannot find the ordering directly and need a decomposition, i.e. they violate the condition of Corollary 6.

### 4.2. A more general translation

So far we have considered a special way of converting relational data into a tree-like XML document, namely hierarchical translation. When there is no redundancy-free hierarchical XML representation for a relational specification $(\mathcal{R}, \mathcal{F})$, one might think of other ways of translating $(\mathcal{R}, \mathcal{F})$ into an XML specification $(D, \Sigma)$, such that $(D, \Sigma)$ is in XNF and hence does not allow redundancy. Here we claim that when $(\mathcal{R}, \mathcal{F})$ cannot be converted into a redundancy-free hierarchical XML specification, even a more general approach, named *semi-hierarchical translation*, does not help.

In this approach, instead of having one element type for each relational attribute, we allow element types to represent more than one relational attribute. The formal definition of semi-hierarchical translation would be similar to that of hierarchical translation. The only difference would be that the attribute set assigned to each element type would no longer be a singleton. The FDs added as the result of hierarchical structure would also reflect this change; they may have more than one attribute path on the left-hand side.

**Example 10.** A semi-hierarchical representation of relational schema of Example 3 consists of DTD $D = (E, A, P, R, r)$ and a set of FDs $\Sigma$ as follows:

- $E = \{r, AB, C\}$.
- $A = \{@a, @b, @c\}$.
- $P(r) = AB^*$, $P(AB) = C^*$, $P(C) = \epsilon$.
- $R(r) = \emptyset$, $R(AB) = \{@a, @b\}$, $R(C) = \{@c\}$.
- $\Sigma = \{\{r.AB.@a, r.AB.@b\} \to r.AB,$
  $\{r.AB, r.AB.C.@c\} \to r.AB.C,$
  $\{r.AB.@a, r.AB.@b\} \to r.AB.C.@c,$
  $r.AB.C.@c \to r.AB.@b\}.$

In this translation, the element type $AB$ represents two relational attributes $A$ and $B$ and therefore needs two attributes $@a$ and $@b$.

The following theorem says that checking the conditions of Corollary 6 is enough to know whether or not a relational specification has a non-redundant semi-hierarchical XML representation.

**Theorem 11.** *A relational specification* $(\mathcal{R}, \mathcal{F})$ *has a redundancy-free hierarchical translation iff it has a redundancy-free semi-hierarchical translation.*

**Proof** *(sketch).* One direction of the proof is trivial. It is enough to show that if $(\mathcal{R}, \mathcal{F})$ has a semi-hierarchical XML translation in XNF, then we can construct a hierarchical XML representation, which is also in XNF. Let $(D, \Sigma)$ be a semi-hierarchical translation of $(\mathcal{R}, \mathcal{F})$ in XNF. The non-root element types in $D$ form an ordering $\tau_1, \ldots, \tau_l$ from the root to the leaves. For instance, the ordering corresponding to DTD in Example 10 is $AB, C$. Each element type corresponds to one or more than one relational attribute. We construct the corresponding ordering of relational attributes as follows: for each element type $\tau_i$ representing $k$, $k \geqslant 1$, relational attributes $A_{i_1}, \ldots, A_{i_k}$, we output an arbitrary ordering of attributes $A_{\pi_{i_1}}, \ldots, A_{\pi_{i_k}}$. Let $A_{\pi_1}, \ldots, A_{\pi_m}$ be the final ordering of all attributes. Then according to Definition 4, we construct a hierarchical DTD $D'$ and set of FDs $\Sigma'$ using this ordering. Now it suffices to show that $(D', \Sigma')$ is in XNF.

Suppose there is an FD $X \to A_{\pi_i} \in \mathcal{F}^+$. This FD has a translation $S \to p.@l_i$ in the semi-hierarchical representation $(D, \Sigma)$. Since $(D, \Sigma)$ is in XNF, the FD $S \to p$ should also be in $(D, \Sigma)^+$. This means $S$ implies all the elements and their attributes from the root to (and including) *last*$(p)$. Therefore, $X$ implies all attributes $A_{\pi_j}$, $j < i$, since they all appear on the path from the root to *last*$(p)$ in XML representation. Then according to Theorem 5 and Corollary 6 the XML translation $(D', \Sigma')$ obtained from ordering $A_{\pi_1}, \ldots, A_{\pi_m}$ is in XNF. □

## 5. Conclusions and future research

We looked at the problem of designing dependency-preserving XML documents. We showed that for some relational specifications, for which there is no FD-preserving BCNF decomposition, it is possible to have a dependency-preserving XML design that is in normal form XNF [13] and hence eliminates all redundancies. This can be done by a hierarchical mapping of relational attributes to XML elements defined by a DTD. This transformation of data was justified by showing the fact that to guarantee dependency preservation, relational 3NF allows high degrees of redundancy in the positions of database that store values for prime attributes. This was shown using an information-theoretic measure [3].

Using an example, we next motivate our future work by observing that the normal form XNF is not dependency-preserving. Given an XML specification $(D, \Sigma)$, there is a decomposition algorithm [13] that produces a lossless decomposition $(D', \Sigma')$ in XNF. The following example shows how this algorithm works on an XML document to avoid redundancies.

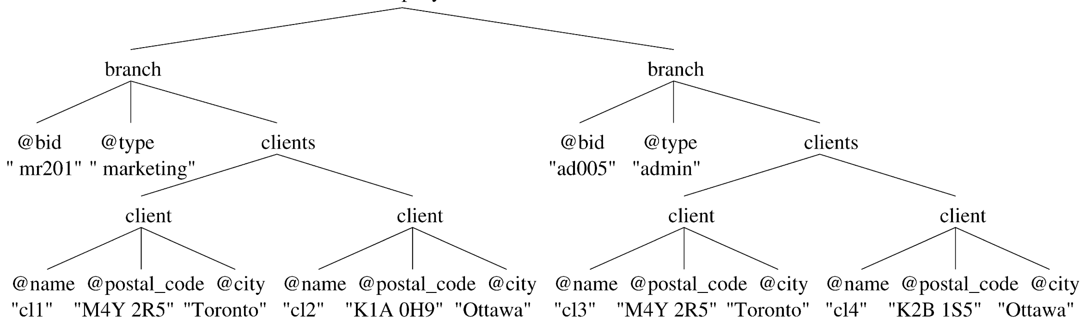

Fig. 4. An XML document containing redundant information.

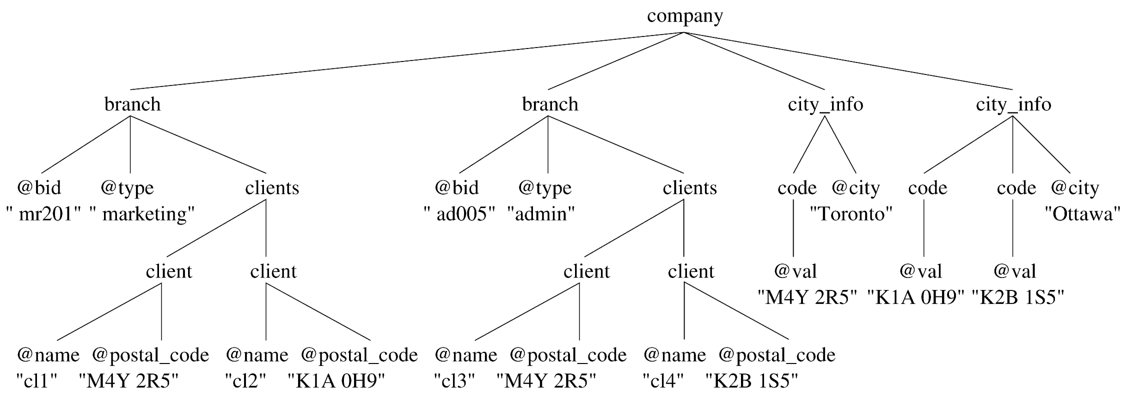

Fig. 5. A redundancy-free XML document.

**Example 12.** Consider the XML document in Fig. 4 that describes a company database. This document satisfies the following constraint: any two clients with the same postal code value must have the same city value. This can be expressed with the following XML FD:

$$company.branch.clients.client.@postal\_code \to company.branch.clients.client.@city.$$

Since the value of postal code does not identify a node corresponding to a *client* element, this document does not satisfy XNF and stores redundant information caused by the FD. To avoid this, the normalization technique suggests to split the information of cities and postal codes by creating a new element type *city_info*. The restructured version of the document that reflects this decomposition is shown in Fig. 5.

Now assume there is another constraint in the original document of Fig. 4: if two clients are in the same city and require a certain type of service, they are handled by the same branch; written as:

$$\{company.branch.clients.client.@city, company.branch.@type\} \to company.branch.$$

By splitting the cities information in the restructured document of Fig. 5, we actually break the nesting of the element *branch* and the attribute *@city*, so this functional dependency no longer holds over the new document.

The above example shows that the XNF decomposition algorithm [13] is not dependency-preserving. We have also seen that using the algorithm presented in Section 4, we cannot give an FD-preserving XML representation in XNF for some relational specifications, like the one in Example 7.

In general, the concept of dependency preservation seems to be more involved for the case of XML due to the fact that the implication problem of FDs is not even known to be decidable in presence of DTDs. It is claimed in [21] that like its relational counterpart, BCNF, the normal form XNF cannot be achieved for some XML specifications without losing some FDs, and therefore a new normal form, X3NF, is introduced for XML, which is a generalization of relational third normal form (3NF). It remains to prove that the normal form introduced in [21] always has a dependency-preserving decomposition.

A formal definition of dependency preservation for XML is the next step. Then using this definition, it would be nice to prove that a lossless dependency-preserving decomposition into XML third normal, introduced in [21], can be achieved for any XML document.

## Acknowledgments

The author thanks Leonid Libkin for the extremely helpful discussions and comments. The valuable comments from anonymous reviewers are also gratefully acknowledged.

## References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison–Wesley, 1995.
[2] C. Beeri, P.A. Bernstein, N. Goodman, A sophisticate's introduction to database normalization theory, in: VLDB '78, pp. 113–124.

[3] M. Arenas, L. Libkin, An information-theoretic approach to normal forms for relational and XML data, J. ACM 52 (2) (2005) 246–283.
[4] M.W. Vincent, J. Liu, C. Liu, Redundancy free mappings from relations to XML, in: WAIM '04, pp. 346–356.
[5] Y. Chen, S. Davidson, C. Hara, Y. Zheng, RRXS: Redundancy reducing XML storage in relations, in: VLDB '03, pp. 189–200.
[6] D. Lee, W.W. Chu, Constraints-preserving transformation from XML document type definition to relational schema, in: ER '00, pp. 323–338.
[7] P. Buneman, S. Davidson, W. Fan, C. Hara, W.-C. Tan, Reasoning about keys for XML, Inf. Syst. 28 (8) (2003) 1037–1063.
[8] P. Buneman, S.B. Davidson, W. Fan, C.S. Hara, W.C. Tan, Keys for XML, in: WWW '01, pp. 201–210.
[9] W. Fan, G.M. Kuper, J. Siméon, A unified constraint model for XML, in: WWW '01, pp. 179–190.
[10] W. Fan, L. Libkin, On XML integrity constraints in the presence of DTDs, in: PODS '01, pp. 114–125.
[11] W. Fan, J. Siméon, Integrity constraints for XML, in: PODS '00, pp. 23–34.
[12] M. Arenas, W. Fan, L. Libkin, On verifying consistency of XML specifications, in: PODS '02, pp. 259–270.
[13] M. Arenas, L. Libkin, A normal form for XML documents, in: PODS '02, pp. 85–96.
[14] M. Vincent, J. Liu, Functional dependencies for XML, in: APWEB '03, pp. 22–34.
[15] M.W. Vincent, J. Liu, C. Liu, Strong functional dependencies and their application to normal forms in XML, ACM Trans. Database Systems 29 (3) (2004) 445–462.
[16] M.-L. Lee, T.W. Ling, W.L. Low, Designing functional dependencies for XML, in: EDBT '02, pp. 124–141.
[17] S. Hartmann, S. Link, More functional dependencies for XML, in: ADBIS '03, pp. 355–369.
[18] J. Wang, R.W. Topor, Removing XML data redundancies using functional and equality-generating dependencies, in: Australian Database Conference, 2005, pp. 65–74.
[19] D.W. Embley, W.Y. Mok, Developing XML documents with guaranteed "good" properties, in: ER '01, pp. 426–441.
[20] S. Kolahi, L. Libkin, On redundancy vs dependency preservation in normalization: An information-theoretic study of 3NF, in: PODS '06, pp. 114–123.
[21] S. Kolahi, Dependency-preserving normalization of relational and XML data, in: DBPL '05, in: Lecture Notes in Comput. Sci., vol. 3774, Springer, 2005, pp. 247–261.