Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 543–550
ISBN 978-83-60810-14-9
ISSN 1896-7094

Dealing with redundancies and dependencies in normalization of XML data

Tadeusz Pankowski
Institute of Control and Information Engineering
Poznań University of Technology
Pl. M.S.-Curie 5, 60-965 Poznań, Poland
Email: tadeusz.pankowski@put.poznan.pl

Tomasz Piłka
Faculty of Mathematics and Computer Science
Adam Mickiewicz University
ul. Umultowska 87, 61-614 Poznań, Poland
Email: tomasz.pilka@amu.edu.pl

***Abstract*—In this paper we discuss the problem of redundancies and data dependencies in XML data while an XML schema is to be normalized. Normalization is one of the main tasks in relational database design, where 3NF or BCNF, is to be reached. However, neither of them is ideal: 3NF preserves dependencies but may not always eliminate redundancies, BCNF on the contrary—always eliminates redundancies but may not preserve constraints. We discuss the possibility of achieving both redundancy-free and dependency preserving form of XML schema. We show how the XML normal form can be obtained for a class of XML schemas and a class of XML functional dependencies.**

I. Introduction

As XML becomes popular as the standard data model for storing and interchanging data on the Web and more companies adopt XML as the primary data model for storing information, XML schema design has become an increasingly important issue. Central objectives of good schema design is to avoid data redundancies and to preserve dependencies enforced by the application domain. Existence of redundancy can lead not only to a higher data storage cost but also to increased costs for data transfer and data manipulation. It can also lead to update anomalies.

One strategy to avoid data redundancies is to design redundancy-free schema. One can start from an intuitively correct XML schema and a given set of functional dependencies reflecting some rules existing in application domain. Then the schema is normalized, i.e. restructured, in such a way that the newly obtained schema has no redundancy, preserves all data (is a lossless decomposition) and preserves all dependencies. In general, obtaining all of these three objectives is not always possible, as was shown for relational schemas [1]. However, in the case of XML schema, especially thanks to its hierarchical structure, this goal can be more often achieved [2].

The normalization process for relational schemas was proposed in the early 70s by Codd [3], [4]. Then a number of different normal forms was proposed, where the most important of them such as 2NF, 3NF [3], BCNF [4], and 4NF [5] are discussed today in every database textbook [1], [6], [7]. These normal forms together with normalization algorithms, aim to deal with the design of relational database taking into account different types of data dependencies [8], [7]. The result of the normalization should be a well-designed database [9]. The process of normalization of an XML schema is similar: we have to choose an appropriate XML schema for a given DTD and a set of data dependencies.

Recently, research on normalization of XML data was reported in papers by Arenas and Libkin [10], [11], Kolahi and Libkin [2], [12], [13], Yu and Jagadish [14].

In this paper we discuss a method for normalizing a class of XML schemas into an XML normal form that is redundancy-free and preserves all XML functional dependencies. To this order we apply the theory proposed in [12] and [11]. The novelty of the paper is the following:

- We use a new language (preliminarily proposed in [15]) for expressing schemas in a form of tree-pattern formulas, and for specifying XML functional dependencies.
- We show how the proposed formalism can be used for normalizing XML schemas into normal form similar to that of BCNF with eliminating redundancies but preserving all functional dependencies.

The structure of the paper is the following. In Section 2 we introduce an running example and motivate the research. In Section 3 a relational form of the running example is considered and some problems with its normalization are discussed. Basic notations relevant to the discussed issue from the XML perspective, are introduced in Section 4. We define a notation for defining tree-pattern formulas and for specifying data dependencies: XML functional dependencies and keys. Next, in Section 5, we discussed different normalization alternatives—we show advantages and drawbacks of some schema choices. Finally, in Section 6, an XML normal form (XNF) is defined (according to [11]) and we show how this form can be reached for our running example. We discuss the problem from theoretical and practical points of view. Section 7 concludes the paper.

II. Motivation—Redundancies in XML Data

In XML data, like in relational ones, redundancies are caused by bad design of schemas. There are two kinds of design problems [11]: first of them is caused by non-key functional dependencies and is typical for relational schema design, while the other is more closely related to the hierarchical structure of XML documents.

As we mentioned above, in the case of XML schemas some redundancy problems may also occur because of bad design of hierarchical structure of XML document.

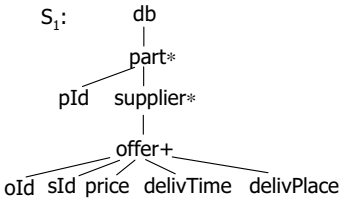

Fig. 1. Sample XML schema tree

```
db          →  part*
part        →  pId supplier*
pId         →  ϵ
supplier    →  offer+
offer       →  oId sId price delivTime delivPlace
oId         →  ϵ
sId         →  ϵ
price       →  ϵ
delivTime   →  ϵ
delivPlace  →  ϵ
```

Fig. 2. A DTD describing the XML schema in Fig. 1. The symbol ϵ denotes the empty string.

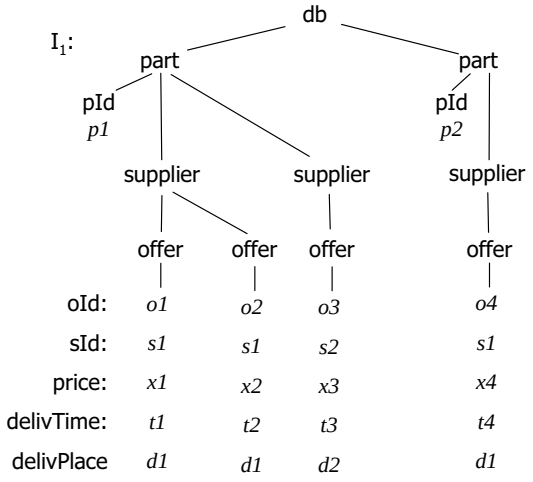

Fig. 3. Sample instance of schema S_1

Example 2.1: Let us consider the XML schema tree in Fig. 1 that describes a fragment of a database for storing data about parts and suppliers offering these parts. Its DTD specification is given in Fig. 2, and an instance of this schema is depicted in Fig. 3.

Each $part$ element has identifier pId. For each part may be zero or more suppliers offering this part. Offers are stored in $offer$ elements. Each offer has: offer identifier oId, supplier identifier sId, price $price$, delivery time $delivTime$, and delivery place $delivPlace$.

We assume that the following constraints must be satisfied by any instance of this schema:

- all $offer$ children of the same $supplier$ must have the same values of sId; this is similar to relational functional dependencies, but now we refer to both the values (text value of sId), and to structure (children of the same $supplier$).
- $delivPlace$ functionally depends on part (pId) and supplier (sId), i.e. when a supplier has two different offers for the same part (possibly with different $delivTime$ and/or $price$) the $delivPlace$ is the same - see offers $o1$ and $o2$ in Fig. 3.
- $delivPlace$ functionally determines supplier (sId). It means that having a delivery place ($delivPlace$) we know exactly which supplier is associated to this place; although one supplier can own many delivery places. For example, in Fig. 3 $d1$ is delivery place uniquely associated to the supplier $s1$.

It is easy to show that schema in Fig. 1 leads to redundancy: sid (an also all other data describing suppliers such as e.g.: name and address) and $delivPlace$ are stored multiple times for a supplier.

Further on we will show that a special caution should be paid to such kind of dependencies as these in which participates $delivPlace$. In this case we have to do with "cyclic" dependencies, i.e. $delivPlace$ depends on pId and sId ($pId, sId \rightarrow delivPlace$) and sId depends on $delivPlace$ ($delivPlace \rightarrow sId$).

First, we will discuss difficulties caused by such "cyclic" dependencies in relational databases, and next, we will show how this problem can be solved in the case of XML data.

III. Dealing With Redundancies and Dependencies in Relational Databases

A. Relational schemas and functional dependencies

In relational data model, a relational schema is understood as a pair

$$\mathcal{R} = (U, F),$$

where U is a finite set of *attributes*, and F is a set of *functional dependencies* over F. A functional dependence (FD) as an expression of the form

$$X \rightarrow Y,$$

where $X, Y \subseteq U$ are subsets of U. If $Y \subseteq X$, then $X \rightarrow Y$ is a *trivial* FD. By F^+ we denote all dependencies which can be inferred from F by means of Armstrong's axioms [16], [7].

A *relation* of type U is a finite set of tuples of type U. Let $U = \{A_1, ..., A_n\}$ and $dom(A)$ be the *domain* of attribute $A \in U$. Then a tuple $[A_1 : a_1, ..., A_n : a_n]$, where $a_i \in dom(A_i)$, is a tuple of type U.

A relation R conforms to schema $\mathcal{R} = (U, F)$ (is an instance of this schema) if R is of type U, and all dependencies

from F^+ are satisfied by R. An FD $X \to Y$ is satisfied by R, denoted $R \models X \to Y$, if for all tuples $r_1, r_2 \in R$ holds

$$\pi_X(r_1) = \pi_X(r_2) \Rightarrow \pi_Y(r_1) = \pi_Y(r_2),$$

where $\pi_X(r)$ is the *restriction* (*projection*) of tuple r on the set X of attributes.

A *key* in $\mathcal{R} = (U, F)$ is such a minimal set K of attributes that $K \to U$ is in F^+. Then each $A \in K$ is a *prime* attribute.

B. Normalization of relational schemas

The main task in relational schema normalization is producing such a set of schemas that posses the required form, usually 3NF or BCNF. The normalization process consists in decomposition of a given input schema. The other approach consists in synthesizing 3NF from functional dependencies [8].

Ideally, a decomposition of a schema should be lossless, i.e. should preserve data and dependencies. Let $\mathcal{R} = (U, F)$, $U_1, U_2 \subseteq U$, and $U_1 \cup U_2 = U$, then schemas $\mathcal{R}_1 = (U_1, F_1)$ and $\mathcal{R}_2 = (U_2, F_2)$ are a lossless decomposition of $\mathcal{R} = (U, F)$, iff:

- The decomposition preserves data, i.e. for each instance R of $\mathcal{R}$ the natural join of projections of R on U_1 and U_2 produces the relation equal to R, i.e.

  $$R = \pi_{U_1}(R) \bowtie \pi_{U_2}(R).$$

- The decomposition preserves dependencies, i.e.

  $$F^+ = (F_1 \cup F_2)^+,$$

  where $F_1 = \{X \to Y \mid X \to Y \in F \wedge X \cup Y \subseteq U_1\}$, and similarly for F_2.

The decomposition $((U_1, F_1), (U_2, F_2))$ of (U, F) preserves data, if $U_1 \cap U_2 \to U_1 \in F^+$ (or, symmetrically, $U_1 \cap U_2 \to U_2 \in F^+$) [9], [7]. Then we say that the *decomposition is determined* by the functional dependence $U_1 \cap U_2 \to U_1 \in F^+$.

A schema $\mathcal{R} = (U, F)$ is in 3NF if for every FD $X \to A \in F^+$ holds:

- X is a superkey, i.e. a key is a part of X, or
- A is prime.

The second condition says that only prime attributes may be functionally dependent on a set of attributes which is not a key. A schema is in BCNF if only the first condition of the two above is allowed. It means, that if whenever a set X determines functionally an attribute A, then X is a superkey, i.e. determines the whole set U.

The aim of a normalization process is to develop normal forms by analyzing functional dependencies and successive decomposition of the input relational schema into its projections. This way a well-designed schema can be obtained, where unnecessary redundancies and update anomalies had been eliminated. In practice, 3NF is accepted as the most desirable form of relational schemas It does not eliminate all redundancies but guaranties dependency preservation. On contrast, BCNF eliminates all redundancies but does not preserve all dependencies. In [13] it was shown that 3NF has the least amount of redundancy among all dependency preserving normal forms. The research adopts a recently proposed information-theoretic framework for reasoning about database designs [10].

C. Relational analysis of XML schema

Let us consider the relational representation of data structure presented in Fig. 1. Then we have the following relational schema:

$$\begin{aligned}
\mathcal{R} &= (U, F), \text{ where} \\
U &= \{oId, sId, pId, price, delivTime, delivPlace\}, \\
F &= \{oId \to sId, pId, price, delivTime, delivPlace, \\
&\quad sId, pId \to delivPlace, \\
&\quad delivPlace \to sId\}.
\end{aligned}$$

In $\mathcal{R}$ there is only one *key*. The key consists of one attribute oId since all attributes in U functionally depends on oId. Thus, R is in 2NF and oId is the only prime (key) attribute in $\mathcal{R}$. Additionally, we assume that a given supplier delivers a given part exactly to one place ($pId, sId \to delivPlace$). Moreover, delivery place $delivPlace$ is connected with only one supplier ($delivPlace \to sId$).

R is not in 3NF because for the functional dependency $sId, pId \to delivPlace$:

- sId, pId is not a superkey, and
- $delivPlace$ is not a prime attribute in U.

Similarly for $delivPlace \to sId$.

The lack of 3NF is the source of redundancies and update anomalies. Indeed, for example, the value of $delivPlace$ will be repeated as many times as many different tuples with the same value of the pair (sId, pId) exist in the instance of $\mathcal{R}$. To eliminate this drawback, we can decompose $\mathcal{R}$ into two relational schemas, $\mathcal{R}_1$ and $\mathcal{R}_2$, which are in 3NF. The decomposition must be based on the dependency $sId, pId \to delivPlace$ which guarantees that the decomposition preserves data. As a result, we obtain:

$$\begin{aligned}
\mathcal{R}_1 &= (U_1, F_1), \text{ where} \\
U_1 &= \{oId, sId, pId, price, delivTime\}, \\
F_1 &= \{oId \to sId, pId, price, delivTime\}.
\end{aligned}$$

$$\begin{aligned}
\mathcal{R}_2 &= (U_2, F_2), \text{ where} \\
U_2 &= \{sId, pId, delivPlace\}, \\
F_2 &= \{sId, pId \to delivPlace, \\
&\quad delivPlace \to sId\}.
\end{aligned}$$

The discussed decomposition is both data and dependencies preserving, since:

$$R(U) = \pi_{U_1}(R) \bowtie \pi_{U_2}(R),$$

for every instance R of schema $\mathcal{R}$, and

$$F = (F_1 \cup F_2)^+.$$

However, we see that $\mathcal{R}_2$ is not in BCNF, since $delivPlace$ is not a superkey in $\mathcal{R}_2$.

The lack of BCNF in $\mathcal{R}_2$ is the reason of redundancies. For example, in table R_2 we have as many duplicates of sId as

R_2

sId	pId	$delivPlace$
s1	p1	d1
s1	p2	d1
s1	p3	d2
s2	p1	d3

many tuples with the same value of $delivPlace$ exist in this table.

We can further decompose $\mathcal{R}_2$ into BCNF schemas $\mathcal{R}_{21}$ and $\mathcal{R}_{22}$, taking $delivPlace \rightarrow sId$ as the base for the decomposition. Then we obtain:

$$\begin{aligned} \mathcal{R}_{21} = &\quad (U_{21}, F_{21}), \text{ where} \\ U_{21} = &\quad \{delivPlace, sid\}, \\ F_{21} = &\quad \{delivPlace \rightarrow sId\}. \end{aligned}$$

$$\begin{aligned} \mathcal{R}_{22} = &\quad (U_{22}, F_{22}), \text{ where} \\ U_{22} = &\quad \{pId, delivPlace\}, \\ F_{22} = &\quad \emptyset. \end{aligned}$$

After applying this decomposition to R_2 we obtain tables R_{21} and R_{22}:

R_{21}

sId	$delivPlace$
s1	d1
s1	d2
s2	d3

R_{22}

pId	$delivPlace$
p1	d1
p2	d1
p3	d2
p1	d3

This decomposition is information preserving, i.e.

$$R_2 = R_{21} \bowtie R_{22},$$

but does not preserve functional dependencies, i.e.

$$F_2 \neq (F_{21} \cup F_{22})^+ = F_{21}.$$

We can observe some negative consequences of the loss of functional dependencies in the result decomposition.

Assume that we insert the tuple $(p1, d2)$ into R_{22}. The tuple will be inserted because it does not violate any constrain imposed on $\mathcal{R}_{22}$. However, taking into account table R_{21} we see that supplier $s1$ (determined by $d2$ in force of $delivPlace \rightarrow sId$) offers part $p1$ in the place $d1$. Thus, the considered insertion violates functional dependency $sId, pId \rightarrow delivPlace$ defined in $\mathcal{R}_2$.

The considered example shows that in the case of relational databases we are not able to completely eliminate redundancies and also preserve all functional dependencies. It turns out ([13]) that the best form for relation schema is 3NF, although some redundancies in tables having this form can still remain.

In next section we will show that the hierarchical structure of XML documents can be used to overcome some of the limitations of relational normal forms [11]. As it was shown in [12], there are decompositions of XML schemas that are both information and dependency preserving. In particular, we can obtain a form of XML schema that is equivalent to BCNF, i.e. eliminates all redundancies, and additionally preserves all XML functional dependencies.

IV. XML SCHEMAS AND INSTANCES

Schemas for XML data are usually specified by DTD or XSD [17]. In this section we will define XML schema by means of *tree-pattern formulas* (TPF) [18], [15]. Furthermore, we do not consider attributes in XML trees since they can always be represented by elements. Schemas will be used to specify structures of *XML trees*. Some other properties of XML trees are defined as *schema constraints.*

Definition 4.1: Let L be a set of *labels*, and $\mathbf{x}$ be a vector of variables. A schema TPF over L and $\mathbf{x}$ is an expression conforming to the syntax:

$$\begin{aligned} S &::= /l[E] \\ E &::= l = x \mid l[E] \mid E \wedge ... \wedge E, \end{aligned}$$

where $l \in L$, and $x \in \mathbf{x}$. In order to indicate the set and ordering of variables in S we will write $S(\mathbf{x})$.

□

Example 4.1: For the schema tree from Fig. 1, the schema TPF has the following form:

$$\begin{aligned} S_1 = \quad & /db[part[pId = x_1 \wedge supplier[offer[\\ & oid = x_2 \wedge sid = x_3 \wedge price = x_4 \wedge \\ & delivTime = x_5 \wedge delivTime = x_6]]]]. \end{aligned}$$

□

We see that a schema TPF has the following properties:

- reflects the tree structure of XML data,
- binds variables to paths in the schema tree,
- is a well-formed XPath predicate according to [19].

Definition 4.2: Let S be a schema TPF over $\mathbf{x}$ and let an atom $l = x$ occur in S. Then the path p starting in the root and ending in l is called the type of the variable x, denoted $type_S(x) = p$.

□

The type of x_1 in S_1 is: $type_{S_1}(x_1) = /db/part/pId$.

An XML database consists of a set of XML data. We define XML data as an unordered rooted node-labeled tree (XML tree) over a set L of labels, and a set $Str \cup \{\perp\}$ of strings and the distinguished null value $\perp$ (both strings and the null value, $\perp$, are used as values of text nodes).

Definition 4.3: An *XML tree* I is a tuple $(r, N^e, N^t, child, \lambda, \nu)$, where:

- r is a distinguished *root node*, N^e is a finite set of *element nodes*, and N^t is a finite set of *text nodes*;
- $child \subseteq (\{r\} \cup N^e) \times (N^e \cup N^t)$ – a relation introducing tree structure into the set $\{r\} \cup N^e \cup N^t$, where r is the root, each element node has at least one child (which is an element or text node), text nodes are leaves;
- $\lambda : N^e \rightarrow L$ – a function labeling element nodes with names (labels);
- $\nu : N^t \rightarrow Str \cup \{\perp\}$ – a function labeling text nodes with *text values* from Str or with the null value $\perp$.

□

It will be useful to perceive an XML tree I with schema S over tuple of variables $\mathbf{x}$, as a pair (S, Ω) (called a *description*), where S is the schema TPF, and Ω is a set of valuations of variables in $\mathbf{x}$. A valuation $\omega \in \Omega$ is a function assigning values from $Str \cup \{\perp\}$ to variables in $\mathbf{x}$, i.e. $\omega : \mathbf{x} \to Str \cup \{\perp\}$.

Example 4.2: The instance I_1 in Figure 3 can be represented by the following description:

$I_1 := (S_1(x_1, x_2, x_3, x_4, x_5, x_6), \{(p1, o1, s1, x1, t1, d1),$
$(p1, o2, s1, x2, t2, d1), (p1, o3, s2, x3, t3, d2),$
$(p2, o4, s1, x4, t4, d1)\}).$

□

An XML tree I satisfies a description (S, Ω), denoted $I \models (S, \Omega)$, if I satisfies (S, ω) for every $\omega \in \Omega$, where this satisfaction is defined as follows:

Definition 4.4: Let S be a schema TPF over $\mathbf{x}$, and ω be a valuation for variables in $\mathbf{x}$. An XML tree I satisfies S by valuation ω, denoted $I \models (S, \omega)$, if the root r of I satisfies S by valuation ω, denoted $(I, r) \models (S, \omega)$, where:

1) $(I, r) \models (/l[E], \omega)$, iff $\exists n \in N^e\ child(r, n) \wedge (I, n) \models (l[E], \omega)$;
2) $(I, n) \models (l[E_1 \wedge ... \wedge E_k], \omega)$, iff $\lambda(n) = l$ and $\exists n_1, ..., n_k \in N^e(child(n, n_i) \wedge (I, n_i) \models (E_i, \omega))$ for $1 \leq i \leq k$;
3) $(I, n) \models (l = x, \omega)$, iff $\lambda(n) = l$ and $\exists n' \in N^t(child(n, n') \wedge \nu(n') = \omega(x))$.

□

A description (S, Ω) represents a class of S instances with the same set of values (the same Ω), since elements in instance trees can be grouped and nested in different ways.

For example, the XML tree in Fig. 4 satisfies two descriptions (S_1, Ω_1), and (S_2, Ω_2) where:

$$\begin{aligned} S_1 &= /A[B = x_1 \wedge C = x_2], \\ \Omega_1 &= \{(b, c1), (b, c2)\}; \end{aligned}$$

$$\begin{aligned} S_2 &= /A[B = x_1 \wedge C = x_2 \wedge D = x_3], \\ \Omega_2 &= \{(b, c1, d1), (b, c2, d1)\}. \end{aligned}$$

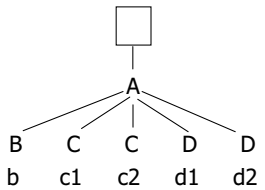

Fig. 4. A simple XML tree

V. XML FUNCTIONAL DEPENDENCIES AND KEYS

Over a schema TPF we can define some constraints, which specify functional dependencies between values and/or nodes in instances of the schema. These constraints are called XML functional dependencies (XFD).

Definition 5.1: An XML functional dependency (XFD) over a set L of *labels* and a set $\mathbf{x}$ of variables is an expression with the syntax:

$$\begin{aligned} fd &::= /P[C]/.../P[C], \\ P &::= l \mid P/l, \\ C &::= TRUE \mid P = x \mid C \wedge ... \wedge C, \end{aligned} \tag{1}$$

where $l \in L$, and x is a variable in $\mathbf{x}$. If variable names are significant, we will write $fd(\mathbf{x})$.

□

An XFD is an XPath expression that for a given valuation ω of its variables returns a sequence of objects (nodes or text values). An XML tree $I = (S, \Omega)$ satisfies an XFD $f(x_1, ..., x_k)$ if for each valuation $\omega \in \Omega$ of its variables, $f(x_1, ..., x_k)$ returns a singleton.

Definition 5.2: Let I be an instance of schema TPF $S(\mathbf{x})$ and f be an XFD defined over S. The instance I satisfies f, denoted $I \models f$, if for every valuation ω of variables in $\mathbf{x}$, the implication holds

$$I \models (S, \omega) \Rightarrow count([\![f(\omega)]\!]) \leq 1,$$

where $[\![f(\omega)]\!]$ is the result of f computed by the valuation ω.

□

In the following example we discuss some XFDs over S_1.

Example 5.1: Over S_1 the following XFDs can be defined:

$$\begin{aligned} f_1(x) &= /db/part[supplier/offer/oId = x], \\ f_2(x) &= /db/part[supplier/offer/oId = x]/pId, \\ f_3(x_1, x_2) &= /db/part[pId = x_1]/supplier/offer[\\ &\quad sId = x_2]/delivPlace, \\ f_4(x) &= /db/part/supplier/offer[\\ &\quad delivPlace = x]/sId, \\ f_5(x_1, x_2) &= /db/part[pId = x_1]/supplier[\\ &\quad offer/sId = x_2]. \end{aligned}$$

According to XPath semantics [19] the expression $f_1(x)(\omega)$ is evaluated against the instance I_1 as follows: (1) first, a set of nodes of type $/db/part$ is chosen; (2) next, for each chosen node the predicate $[supplier/offer/oId = x]$ is tested, this predicate is true in a node n, if there exists a path of type $supplier/offer/oId$ in I_1 leading from n to a text node with the value $\omega(x)$. We see that $count([\![f_1(x)(\omega)]\!])$ equals 1 for all four valuations satisfied by I_1, i.e. for $\omega_i = [x \mapsto o_i]$, $1 \leq i \leq 4$.

Similarly, execution of $f_2(x)(\omega)$ returns a text value of the path $/db/part/pId$, where from the set of nodes determined by $/db/part$ are taken only nodes satisfying the predicate $[supplier/offer/oId = \omega(x)]$. We see that also this XFD is satisfied by I_1.

However, none of the following XFDs is satisfied in I_1:

$$\begin{aligned} g_1(x) &= /db/part[supplier/offer/sId = x], \\ g_2(x) &= /db/part[pId = x]/supplier/offer/sId \\ g_3(x) &= /db/part[pId = x]/supplier/offer \\ g_4(x_1, x_2) &= /db/part[pId = x_1]/supplier/offer[\\ &\quad sId = x_2], \\ g_5(x) &= /db/part/pId/supplier/offer[\\ &\quad delivPlace = x]. \end{aligned}$$

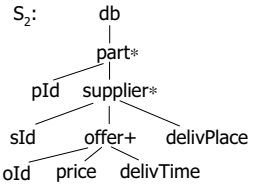

Fig. 5. Restructured form of schema in Fig. 1

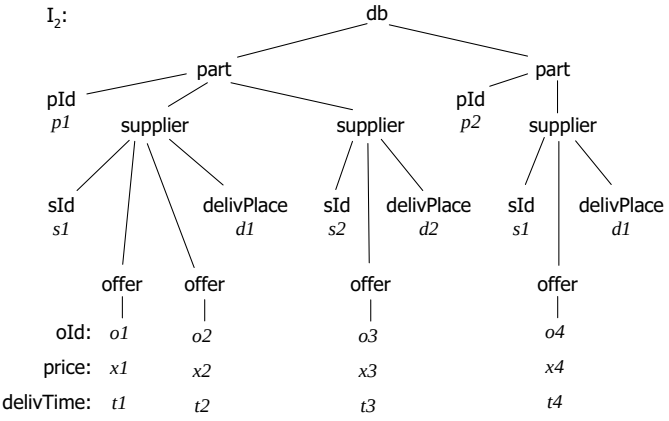

Fig. 6. Instance of schema in Fig. 5

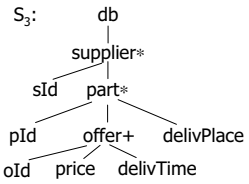

Fig. 7. Restructured form of schema in Fig. 1

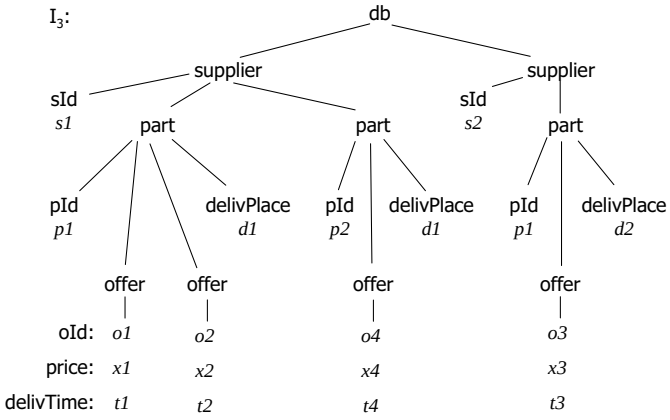

Fig. 8. Instance of schema in Fig. 7

Evaluating the above XFDs against I_1, we obtain:

$$\begin{aligned} count([\![g_1(x)([x \mapsto s1])]\!]) &= 2, \\ count([\![g_2(x)([x \mapsto p1])]\!]) &= 2, \\ count([\![g_3(x)([x \mapsto p1])]\!]) &= 3, \\ count([\![g_4(x_1,x_2)([x_1 \mapsto p1, x_2 \mapsto s1])]\!]) &= 2, \\ count([\![g_5(x)([x \mapsto d1])]\!]) &= 3. \end{aligned}$$

An XFD can determine functional relationship between a tuple of text values of a given tuple of paths and a path denoting either a text value (e.g. $f_2(x)$) or a subtree (a node being the root of the subtree) (e.g. $f_1(x)$).

There are also different notations to express functional notations. For example, according to the notation used in [11], the XFD $f_5(x_1, x_2)$ can be expressed as:

$$db.part.@pId, db.part.supplier.offer.@sId \rightarrow db.part.supplier,$$

and using the language proposed in [20], as

$$(db.part, \{pId\}),$$
$$(db.part, (supplier, \{offer/sId\})),$$

where the first expression is an *absolute key* saying that $db.part$ is absolutely determined by pId, and the second expression is a *relative key* saying that in the context of $db.part$ a tree $supplier$ is determined by the path $offer.sId$.

Further on, by (S, F) we will denote a schema TPF S together with a set of XFDs defined over S. The closure of (S, F), denoted by $(S, F)^+$, is the schema TPF S and the set of all XFDs which can be derived from (S, F).

VI. Normal form for XML

To eliminate redundancies in XML documents, some normal forms for XML schemas have been proposed [21], [11], [2], [12]. We will define XNF (XML Normal Form) using tree-pattern formulas and functional dependencies defined in the previous section and adapting the approach proposed in [21].

Definition 6.1: Let S be a schema TPF and F be a set of functional dependencies over S. (S, F) is in XML normal form (XNF) iff for every XFD $f/l \in (S, F)^+$, also $f \in (S, F)^+$, i.e.

$$(S, F) \text{ is in } XNF \text{ iff } f/l \in (S, F)^+ \Rightarrow f \in (S, F)^+.$$

□

Intuitively, let $f(x_1, ..., x_k)/l$ be XFD and I be an instance of S. I satisfies $f(x_1, ..., x_k)/l$, if for any valuation ω of the tuple of variables $(x_1, ..., x_k)$, there is at most one text value of the type $type(f/l)$. Thus, to avoid redundancies, for every valuation of $(x_1, ..., x_k)$ we should store the value of f/l only once, i.e. there must be only one subtree of type $type(f)$ denoted by the expression $f(x_1, ..., x_k)$. In other words, XFD f must be implied by (S, F) [11].

Let us consider schema S_2 in Fig. 5 and its instance I_2 in Fig. 6. Then

$$f(x_1)/sId = /db/part/supplier[delivPlace = x_1]/sId$$

is XFD over S_2. This dependency corresponds to relational functional dependency $delivPlace \rightarrow sId$ and says that delivery place determines the supplier (sId).

However, S_2 is not in XNF, since its instance I_2 does not satisfy

$$f(x_1) = /db/part/supplier[delivPlace = x_1],$$

because for the valuation $\omega = [x_1 \mapsto "d1"]$ there are two different element nodes $supplier$ with value $d1$ of $delivPlace$. It means that I_2 is not free of redundancy.

In the case of schema S_3 (Fig. 7) the corresponding XFD has the form:

$$f(x_1)/sId = /db/supplier[part/delivPlace = x_1]/sId.$$

This dependency and also the XFD

$$f(x_1) = /db/supplier[part/delivPlace = x_1]$$

are satisfied by I_3 in Fig. 8. We see that this time for the valuation $\omega = [x_1 \mapsto "d1"]$, there is only one element node of type $supplier$ from which we can reach $delivPlace$ with value $d1$. This means that schema S_3 not only captures the XML functional dependency under consideration, but also is free of redundancies which may be caused by capturing this dependencies that happens in the case of schema S_2.

The other dependency of interest is $sId, pId \rightarrow delivPlace$. Its specification with respect to S_2 and S_3 is as follows:

$$/db/part[pId = x_1]/supplier[sId = x_2]/delivPlace,$$

and

$$/db/supplier[sId = x_1]/part[pId = x_2]/delivPlace.$$

It is easy to show that if these XFDs are satisfied by valuations, respectively, ω and ω' in instances of S_2 and S_3, then also

$$/db/part[pId = x_1]/supplier[sId = x_2],$$

and

$$/db/supplier[sId = x_1]/part[pId = x_2]$$

are satisfied by these valuations and these instances.

However, neither S_2 nor S_3 is in XNF. We have already shown that there is redundancy in instances of S_2. Similarly, we see that also in instances of S_3 redundancies may occur. Indeed, since one part may be delivered by many suppliers then the description of one part may be multiplied under each supplier delivering this part, so such data as *part name*, *type*, *manufacturer* etc. will be stored many times.

In Fig. 9 there is the final schema, S_4, for schemas under considerations: S_1, S_2, and S_3; S_4 is in XNF. To make the example more illustrative, we added node $name$ to $part$ data. Also the instance in Fig. 10 was slightly extended as compared with instances I_2 and I_3.

XSD (XML Schema Definition) for S_4 in notation proposed in [17] is shown in Fig. 11.

Note that we cannot use DTD since there are two subtrees labeled $part$, where each of them has different type: the $part$ subtree under $supplier$ consists of pId and $delivPlace$, whereas the $part$ subtree under $parts$ consists of pId and $name$. Recall that in the case of DTD each nonterminal symbol (label) can have only one type (definition), i.e. can appear on the left-hand side of exactly one production rule [17].

A set of XFDs for S_4 is defined in Fig. 12. XFDs derived from them are listed in Fig. 13.

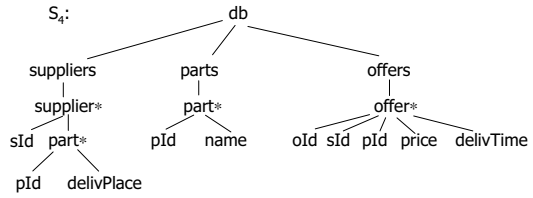

Fig. 9. XNF schema of schemas S_1, S_2, and S_3

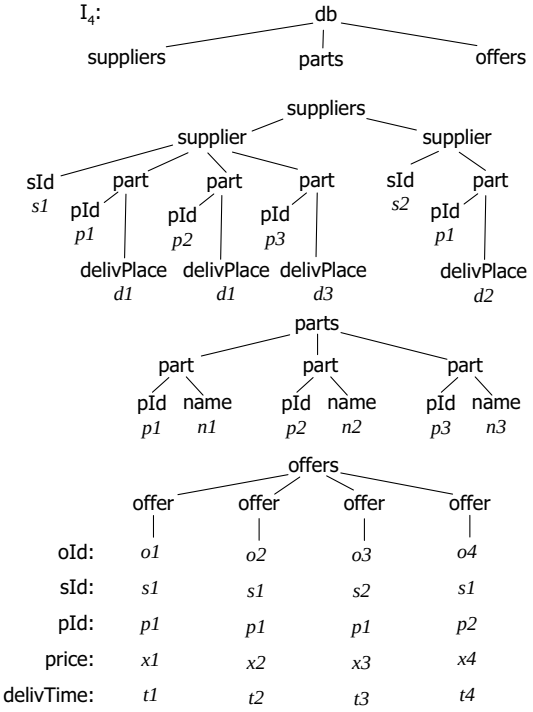

Fig. 10. Instance of schema in Fig. 9

$$
\begin{array}{lcl}
db & \rightarrow & \mathrm{db}[content] \\
content & \rightarrow & \mathrm{suppliers}[suppliers], \mathrm{parts}[parts], \\
 & & \quad \mathrm{offers}[offers] \\
suppliers & \rightarrow & \mathrm{supplier}[supplier]* \\
parts & \rightarrow & \mathrm{parts}[part_1]* \\
offers & \rightarrow & \mathrm{offers}[offer]* \\
supplier & \rightarrow & \mathrm{sId}[\epsilon], \mathrm{part}[part_2] \\
part_2 & \rightarrow & \mathrm{pId}[\epsilon], \mathrm{delivPlace}[\epsilon] \\
part_1 & \rightarrow & \mathrm{pId}[\epsilon], \mathrm{name}[\epsilon] \\
offer & \rightarrow & \mathrm{oId}[\epsilon], \mathrm{sId}[\epsilon], \\
 & & \quad \mathrm{pId}[\epsilon], \mathrm{price}[\epsilon], \mathrm{delivTime}[\epsilon]
\end{array}
$$

Fig. 11. An XSD describing the XML schema in Fig. 9. The symbol ϵ denotes the empty string.

$$
\begin{array}{lcl}
f_1(x) & = & /db/suppliers/supplier[sId = x] \\
f_2(x_1, x_2) & = & /db/suppliers/supplier[sId = x]/ \\
 & & \quad part[pId = x_2]/delivPlace \\
f_3(x) & = & /db/suppliers/supplier[\\
 & & \quad part/delivPlace = x]/sId \\
f_4(x) & = & /db/parts/part[pId = x]/name \\
f_5(x_1, x_2) & = & /db/offers/offer[sId = x_1 \wedge \\
 & & \quad pId = x_2]/oId \\
f_6(x) & = & /db/offers/offer[oId = x]/price \\
f_7(x) & = & /db/offers/offer[oId = x]/delivTime
\end{array}
$$

Fig. 12. A set of XFD for the schema S_4

$$
\begin{array}{lcl}
f'_2(x_1, x_2) & = & /db/suppliers/supplier[sId = x]/ \\
 & & \quad part[pId = x_2] \\
f'_3(x) & = & /db/suppliers/supplier[\\
 & & \quad part/delivPlace = x] \\
f'_4(x) & = & /db/parts/part[pId = x] \\
f'_5(x_1, x_2) & = & /db/offers/offer[sId = x_1 \wedge pId = x_2] \\
f'_6(x) & = & /db/offers/offer[oId = x]
\end{array}
$$

Fig. 13. A set of XFD derived from those in Fig. 12

We see that the schema S_4 satisfies the condition of XNF. Thus, this schema is both redundant-free and dependency preserving.

VII. Conclusion

In this paper, we discussed how the concept of database normalization can be used in the case of XML data. Normalization is commonly used to develop a relational schema free of unnecessary redundancies and preserving all data dependencies existing in application domain. In order to apply this approach to design XML schemas, we introduced a language for expressing XML functional dependencies. In fact, this language is a class of XPath expressions, so its syntax and semantics are defined precisely. We define the notion of satisfaction of XML functional dependence by an XML tree. To define XNF we use the approach proposed in [11].

All considerations are illustrated by the running example. We discuss various issues connected with normalization and compare them with issues faced in the case of relational databases. We show how to develop redundancy-free and dependency preserving XML schema. It is worth mentioning that the relational version of the schema cannot be structured in redundancy-free and dependency preserving form. In this case, preservation of all dependencies requires 3NF but then some redundancy is present. Further normalization to BCNF eliminates redundancies but does not preserve dependencies. In the case of XML, thanks to its hierarchical nature, we can achieve both properties. However, it is not clear if this is true in all cases (see e.g. [12]).

In [15], [22], [23], XML functional dependencies (XFD) have been used in XML data integration settings, in particular for controlling query propagation in P2P environment and for reconciliation of inconsistent data.

Acknowledgement: This work was supported in part by the Polish Ministry of Science and Higher Education under grant 0014/R/2/T00/06/02.

References

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Reading, Massachusetts: Addison-Wesley, 1995.

[2] S. Kolahi, "Dependency-Preserving Normalization of Relational and XML Data," in *DBPL*, ser. Lecture Notes in Computer Science, G. M. Bierman and C. Koch, Eds., vol. 3774. Springer, 2005, pp. 247–261.

[3] E. Codd, "Further normalization of the data base relational model," *R. Rustin (ed.): Database Systems, Prentice Hall, and IBM Research Report RJ 909*, pp. 33–64, 1972.

[4] E. F. Codd, "Recent investigations in relational data base systems," in *IFIP Congress*, 1974, pp. 1017–1021.

[5] R. Fagin, "Multivalued dependencies and a new normal form for relational databases," *ACM Transactions on Database Systystems*, vol. 2, no. 3, pp. 262–278, 1977.

[6] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*. Redwood City: The Benjamin/Cummings, 1994.

[7] T. Pankowski, *Podstawy baz danych (in Polish, Fundamentals of databases)*. Warszawa: Wydawnictwo Naukowe PWN, 1992.

[8] P. A. Bernstein, "Synthesizing third normal form relations from functional dependencies," *ACM Transactions on Database Systtems*, vol. 1, no. 4, pp. 277–298, 1976.

[9] J. Rissanen, "Independent components of relations," *ACM Transactions on Database Systems*, vol. 2, no. 4, pp. 317–325, 1977.

[10] M. Arenas and L. Libkin, "An information-theoretic approach to normal forms for relational and XML data." *J. ACM*, vol. 52, no. 2, pp. 246–283, 2005.

[11] M. Arenas, "Normalization theory for XML," *SIGMOD Record*, vol. 35, no. 4, pp. 57–64, 2006.

[12] S. Kolahi, "Dependency-preserving normalization of relational and XML data," *Journal of Computer and System Sciences*, vol. 73, no. 4, pp. 636–647, 2007.

[13] S. Kolahi and L. Libkin, "On redundancy vs dependency preservation in normalization: an information-theoretic study of 3NF," in *PODS '06*. New York, NY, USA: ACM, 2006, pp. 114–123.

[14] C. Yu and H. V. Jagadish, "XML schema refinement through redundancy detection and normalization," *VLDB Journal*, vol. 17, no. 2, pp. 203–223, 2008.

[15] T. Pankowski, "XML data integration in SixP2P: a theoretical framework," in *EDBT Workshop on Data Management in P2P Systems DaMaP 2008*, ser. ACM International Conference Proceeding Series, A. Doucet, S. Gançarski, and E. Pacitti, Eds. ACM, 2008, pp. 11–18.

[16] W. W. Armstrong, "Dependency structures of data base relationships," in *IFIP Congress*, 1974, pp. 580–583.

[17] W. Martens, F. Neven, and T. Schwentick, "Simple off the shelf abstractions for XML schema," *SIGMOD Record*, vol. 36, no. 3, pp. 15–22, 2007.

[18] M. Arenas and L. Libkin, "XML Data Exchange: Consistency and Query Answering," in *PODS Conference*, 2005, pp. 13–24.

[19] XML Path Language (XPath) 2.0, 2006, www.w3.org/TR/xpath20.

[20] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, and W. C. Tan, "Reasoning about keys for XML," *Information Systems*, vol. 28, no. 8, pp. 1037–1063, 2003.

[21] M. Arenas and L. Libkin, "A normal form for XML documents." *ACM Trans. Database Syst.*, vol. 29, pp. 195–232, 2004.

[22] T. Pankowski, "Query propagation in a P2P data integration system in the presence of schema constraints," in *Data Management in Grid and P2P Systems (DEXA Globe'2008)*, vol. Lecture Notes in Computer Science **5187**, 2008, pp. 46–57.

[23] ——, "Reconciling inconsistent data in probabilistic XML data integration," in *British National Conference on Databases (BNCOD) 2008*, vol. Lecture Notes in Computer Science **5071**, 2008, pp. 75–86.