Detecting XML Functional Dependencies through Formal Concept Analysis

Katalin Tunde Janosi-Rancz[1], Viorica Varga[2], and Timea Nagy[2]

[1] Hungarian University of Transylvania, Tirgu Mures, Romania
tsuto@ms.sapientia.ro
[2] Babeş-Bolyai University, Cluj-Napoca, Romania,
ivarga@cs.ubbcluj.ro

Abstract. As XML becomes a popular data representation and exchange format over the web, XML schema design has become an important research area. Formal Concept Analysis (FCA) has been widely applied in many fields recently. In this paper, we propose the application of FCA to find functional dependencies (FDs) in XML databases. Our work is based on the definitions of the Generalized Tree Tuple, XML functional dependency and XML key notion presented by [26]. We propose a framework which parses the XML document and constructs the Formal Context corresponding to the flat representation of the XML data. The obtained Conceptual Lattice is a useful graphical representation of the analyzed XML document's elements and their hierarchy. The software also finds the keys and functional dependencies in XML data, which are attribute implications in the constructed Formal Context.

Keywords and phrases: XML design, Formal Concept Analysis, XML Functional Dependency.

1 Introduction

In the last few years several papers discussed the relationship between Formal Concept Analysis (FCA) and relational databases [9, 11]. Our approach intends to extend these results by reformulating the XML functional dependency inference with an FCA viewpoint.

FCA is a mathematical theory of concept hierarchies which is based on Lattice Theory. It is used as a technique for data analysis, knowledge representation; it is a useful tool to represent knowledge contained in a database.

Designing XML data means to choose an appropriate XML schema, which usually come in the form of DTD (Document Type Definition) or XML Scheme. A big number of classical database subjects have been reexamined in the XML context [8, 2, 21, 20, 4] because XML became more and more popular. Discovering XML data redundancies from the data itself becomes necessary and it is an integral part of the schema refinement (or re-design) process.

Recently, there were several attempts to define XFDs (see [7, 19, 10, 22]) but in general, these approaches have different semantics regarding the *tree tuple* and

closest node XFDs and do not preserve the semantics of FDs when relational data is mapped to XML via arbitrary nesting.

Functional dependencies (FDs) are a key factor in XML design. Our paper proposes a framework to mine FDs from an XML database; it is based on the notions of Generalized Tree Tuple, XML functional dependency and XML key notion presented by [26]. Our contribution is the construction of the formal context for a tuple class or the whole XML document. Non-leaf and leaf level elements (or attributes) and corresponding values are inserted in the formal context, then the concept lattice of the XML data is constructed too. The obtained Conceptual Lattice is a useful graphical representation of the analyzed XML document's elements and their hierarchy. The software also finds the keys in the XML document. The set of implications resulted from this concept lattice will be equivalent to the set of functional dependencies that hold in that database.

The outline of the paper is the following: Section 2 discusses related work, Section 3 introduces the minimal definitions needed to understand how formal concept analysis is used. Section 4 presents Functional dependency for XML data. Section 5 explains how we combine these two techniques to mine functional dependencies and then discuss the obtained results. Finally, Section 6 summarizes the conclusions and future work.

2 Related work

The first authors who presented the problem of finding functional dependencies using FCA were Ganter and Wille [9]. Different authors [16, 17] use FCA concepts and methods like agree sets, maximal sets and closed sets, which are closely related to the concept of closed sets and generators, described previously, to avoiding the transformation of the original database. The authors use these concepts to find efficient algorithms to extract functional dependencies for a given set of data. [5] studied the lattice characterization and its properties for Armstrong and symmetric dependencies. Hereth has already described the relationship between FCA and functional dependencies in [11], he has introduced the formal context of functional dependencies. In this context, implications hold for functional dependencies. The paper [12] presents an FCA based approach to detect functional dependencies in a relational database table.

The first authors who formally defined XML FD and normal form (XNF) were Arenas and Libkin introducing the so-called *tree tuple* approach [2]. In [14] and [21], the authors used a *path-based* approach and built their XML FD notion in a way similar to the XML Key notion proposed in [6].

Yu and Jagadish [26] show, that these XML FD notions are insufficient and propose a Generalized Tree Tuple (GTT) based XML functional dependency and key notion, which include particular redundancies involving set elements. Based on these concepts, the GTT-XNF normal form is presented too.

In later work [3], Arenas and Libkin provided a formal justification for the use of XNF in XML database design, using the classical information theory approach. A measure of the information content of data (independent of updates

and queries) is introduced, as entropy of a suitably chosen probability distribution. A formal definition of a well designed XML schema was given and the fact that XNF is both a necessary and sufficient condition for an XML schema to be well designed was proved.

Vincent et al. in [20] investigated the problem of justifying XML normal forms [21], in the terms of *closest node* XFDs using redundancy elimination. In [20], a normal form for XML documents is proposed and it has been proved to be a necessary and sufficient condition for the elimination of redundancy.

3 Formal Concept Analysis in a Nutshell

FCA studies how objects can be hierarchically grouped together according to their common attributes. In FCA the data is represented by a cross table, called formal context.

A *formal context* is a triple (G, M, I). The sets G and M are finite sets of objects and attributes, respectively. The relation $I \subseteq G \times M$ is a binary relation between objects and attributes. Each couple $(g, m) \in I$ denotes the fact that the object $g \in G$ is related to the item $m \in M$.

The following cross table describes for some hotels the attributes they have. In this case the objects are: *Oasis, Royal, Amelia, California, Grand, Samira*; and the attributes are: *Internet, Sauna, Jacuzzi, ATM, Babysitting.*

	Internet	Sauna	Jacuzzi	ATM	Babysitting
Oasis	X	X	X	X	
Royal	X	X	X		
Amelia	X				X
California		X	X		
Grand		X	X		
Samira				X	X

Table 1. Formal context of the Hotel facilities example

For a set $A \subseteq G$ of objects we define

$$A' := \{m \in M \mid gIm \text{ for all } g \in A\}$$

the set of all attributes common to the objects in A. Dually, for a set $B \subseteq M$ of attributes we define

$$B' := \{g \in G \mid gIm \text{ for all } m \in B\}$$

the set of all objects which have all attributes in B.

For example ({California, Grand })= { Sauna, Jacuzzi }.

A *formal concept* of the context $\mathbb{K} := (G, M, I)$ is a pair (A, B) where $A \subseteq G$, $B \subseteq M$, $A' = B$, and $B' = A$. We call A the *extent* and B the *intent* of the

concept (A, B). The set of all concepts of the context (G, M, I) is denoted by $\mathfrak{B}(G, M, I)$.

For example ({Amelia, Royal},{Internet}) is a concept of our context. In our case we have 10 concepts, shown in 2.

c_1	({Oasis, Royal, Amelia, California, Grand, Samira },{ ⊘ })
c_2	({Amelia, Royal},{Internet})
c_3	({Samira, Oasis},{ATM})
c_4	({Samira, Amelia},{Babysitting})
c_5	({Oasis, Royal, California, Grand},{Jacuzzi, Sauna})
c_6	({Royal},{Internet, Jacuzzi, Sauna})
c_7	({Oasis},{ATM, Internet, Jacuzzi, Sauna})
c_8	({Amelia},{ Babysitting, Internet})
c_9	({Samira},{Babysitting, ATM})
c_{10}	({ ⊘ },{ Internet, Sauna, Jacuzzi, ATM, Babysitting })

Table 2. Formal concepts of Hotel facilities context

Between the concepts of a given context there is a natural hierarchical order, the *subconcept-superconcept* relation.

A concept (X_0, Y_0) is a *subconcept* of concept (X_1, Y_1), denoted by $(X_0, Y_0) \leq (X_1, Y_1)$, if $X_0 \subseteq X_1$ (or, equivalently, $Y_1 \subseteq Y_0$). Inversely we define that the concept (X_1, Y_1) is a *superconcept* of concept (X_0, Y_0).

The set of all formal concepts of a context, ordered by this relation, is called the *concept lattice* of the formal context. In FCA the data is transformed into a concept lattice, which can be represented by a hierarchical line diagram. The concept lattice is the basis for further data analysis. In the lattice every circle represents a concept and the information of the context can be read from it in the following way: an object g has an attribute m if and only if there is an upward leading path from the circle named by g to the circle name by m.

An implication $Y1 \rightarrow Y2$ between the attributes of context (G, M, I) is a pair of subsets of M. An implication $Y1 \rightarrow Y2$ holds in a context (G, M, I) if for all intent Y of the objects of the context, $Y1 \not\subseteq Y$ or $Y2 \subseteq Y$. For more details see [9, 23].

4 Functional dependency for XML data

Arenas and Libkin introduced first the so-called *tree tuple* notion in [2], they have defined functional dependency in XML data and XNF normal form for XML. Redundancies in XML data have several distinct features due to the heterogeneous nature of XML data, which makes them richer in semantics as compared with redundancies in relational data. Yu and Jagadish [26] show, that the XML FD notion introduced by [2] doesn't include all possible features of an XML document and propose a Generalized Tree Tuple (GTT) based XML functional

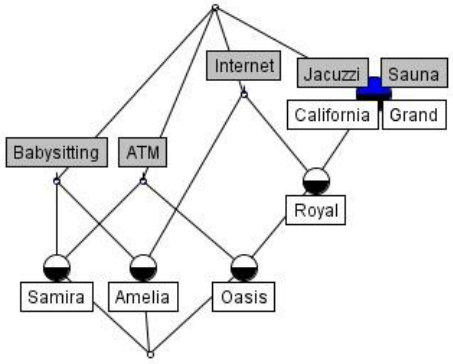


Fig. 1. Concept lattice of hotel facilities context

dependency and XML key notion, which include particular redundancies involving set elements. They propose the GTT-XNF normal form too based on these notions. As [26] we treat leaf level elements and attributes in the same manner. The definitions of this section are based on [26].

Definition 1. *(Schema)* A *schema* is defined as a set $S = (E, T, r)$, where:

- E is a finite set of element labels;
- T is a finite set of element types, and each $e \in E$ is associated with a $\tau \in T$, written as $(e : \tau)$, τ has the next form:
  $\tau ::= \texttt{str} \mid \texttt{int} \mid \texttt{float} \mid \texttt{SetOf}\ \tau \mid \texttt{Rcd}[e_1 : \tau_1, \ldots, e_n : \tau_n]$;
- $r \in E$ is the label of the root element, whose associated element type can not be $\texttt{SetOf}\ \tau$.

This definition corresponds to basic constructs in XML Scheme [24]. Types `str,int` and `float` are the system defined *simple types* and `Rcd` indicate *complex scheme elements* (elements with children elements). Keyword `SetOf` is used to indicate *set schema elements* (elements that can have multiple matching data elements sharing the same parent in the data). We will treat attributes and elements in the same way, with a reserved "@" symbol before attributes.

The examples of this paper are based on XML tree of Figure 2.

Example 1. The scheme $S_{CustOrder}$ of XML document from Figure 2 is:

```
CustOrder:Rcd
   Customers:SetOf Rcd
      CustomerID: str
      CompanyName: str
      Address: str
      City: str
      Country: str
```

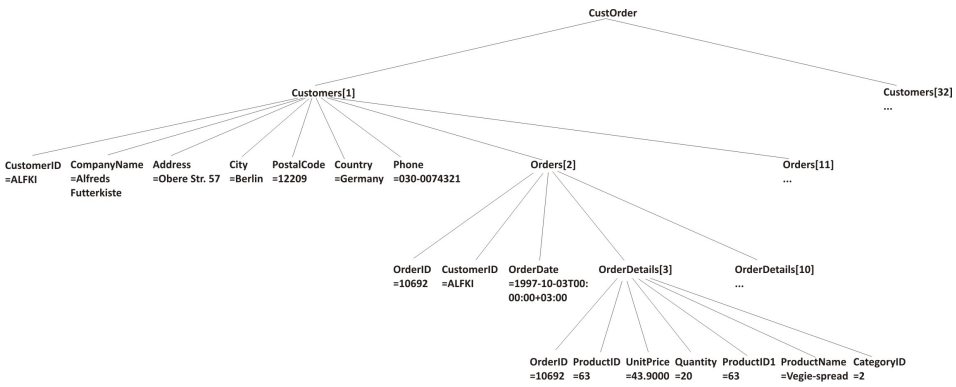


Fig. 2. Example tree

```
Phone: str
Orders: SetOf Rcd
   OrderID: int
   CustomerID: str
   OrderDate: str
   OrderDetails: SetOf Rcd
       OrderID: int
       ProductID: int
       UnitPrice: float
       Quantity: float
       ProductName: str
       CategoryID: int
```

A schema element e_k can be identified through a path expression, $path(e_k) = /e_1/e_2/.../e_k$, where $e_1 = r$, and e_i is associated with type $\tau_i ::= \texttt{Rcd}\ [..., e_{i+1} : \tau_{i+1}, ...]$ for all $i \in [1, k-1]$. A path is *repeatable*, if e_k is a set element. We adopt XPath steps "." (self) and ".." (parent) to form a relative path given an anchor path.

Definition 2. *(Data tree)* An XML database is defined to be a rooted labeled tree $T = \langle N, \mathcal{P}, \mathcal{V}, n_r \rangle$, where:

- N is a set of labeled data nodes, each $n \in N$ has a label e and a node key that uniquely identifies it in T;
- $n_r \in N$ is the root node;
- $\mathcal{P}$ is a set of parent-child edges, there is exactly one $p = (n', n)$ in $\mathcal{P}$ for each $n \in N$ (except n_r), where $n' \in N, n \neq n', n'$ is called the parent node, n is called the child node;
- $\mathcal{V}$ is a set of value assignments, there is exactly one $v = (n, s)$ in $\mathcal{V}$ for each leaf node $n \in N$, where s is a value of simple type.

We assign a node key, referred to as @key, to each data node in the data tree in a pre-order traversal. A data element n_k is a descendant of another data element n_1 if there exists a series of data elements n_i, such that $(n_i, n_{i+1}) \in \mathcal{P}$ for all $i \in [1, k-1]$. Data element n_k can be addressed using a path expression, $path(n_k) = /e_1/\ldots/e_k$, where e_i is the label of n_i for each $i \in [1, k]$, $n_1 = n_r$, and $(n_i, n_{i+1}) \in \mathcal{P}$ for all $i \in [1, k-1]$.

A data element n_k is called *repeatable* if e_k corresponds to a set element in the schema. Element n_k is called a *direct descendant* of element n_a, if n_k is a descendant of n_a, $path(n_k) = \ldots/e_a/e_1/\ldots/e_{k-1}/e_k$, and e_i is not a set element for any $i \in [1, k-1]$.

In considering data redundancy, it is important to determine the equality between the "values" associated with two data elements, instead of comparing their "identities" which is represented by @key. So, we have:

Definition 3. *(Element-value equality)* Two data elements n_1 of $T_1 = \langle N_1, \mathcal{P}_1, \mathcal{V}_1, n_{r1}\rangle$ and n_2 of $T_2 = \langle N_2, \mathcal{P}_2, \mathcal{V}_2, n_{r2}\rangle$ are *element-value equal* (written as $n_1 =_{ev} n_2$) if and only if:

- n_1 and n_2 both exist and have the same label;
- There exists a set M, such that for every pair $(n_1', n_2') \in M$, $n_1' =_{ev} n_2'$, where n_1', n_2' are children elements of n_1, n_2, respectively. Every child element of n_1 or n_2 appears in exactly one pair in M.
- $(n_1, s) \in \mathcal{V}_1$ if and only if $(n_2, s) \in \mathcal{V}_2$,where s is a simple value.

Definition 4. *(Path-value equality)* Two data element paths p_1 on $T_1 = \langle N_1, \mathcal{P}_1, \mathcal{V}_1, n_{r1}\rangle$ and p_2 on $T_2 = \langle N_2, \mathcal{P}_2, \mathcal{V}_2, n_{r2}\rangle$ are *path-value equal* (written as $T_1.p_1 =_{pv} T_2.p_2$) if and only if there is a set M' of matching pairs where

- For each pair $m' = (n_1, n_2)$ in M', $n_1 \in N_1$, $n_2 \in N_2$, $path(n_1) = p_1$, $path(n_2) = p_2$, and $n_1 =_{ev} n_2$;
- All data elements with path p_1 in T_1 and path p_2 in T_2 participate in M', and each such data element participates in only one such pair.

The definition of functional dependency in XML data needs the definition of so called Generalized Tree Tuple.

Definition 5. *(Generalized tree tuple)* A generalized tree tuple of data tree $T = \langle N, \mathcal{P}, \mathcal{V}, n_r\rangle$, with regard to a particular data element n_p (called pivot node), is a tree $t^T_{n_p} = \langle N^t, \mathcal{P}^t, \mathcal{V}^t, n_r\rangle$, where:

- $N^t \subseteq N$ is the set of nodes, $n_p \in N^t$;
- $\mathcal{P}^t \subseteq \mathcal{P}$ is the set of parent-child edges;
- $\mathcal{V}^t \subseteq \mathcal{V}$ is the set of value assignments;
- n_r is the same root node in both $t^T_{n_p}$ and T ;
- $n \in N^t$ if and only if: 1) n is a descendant or ancestor of n_p in T, or 2) n is a non-repeatable direct descendant of an ancestor of n_p in T ;
- $(n_1, n_2) \in \mathcal{P}^t$ if and only if $n_1 \in N^t$, $n_2 \in N^t, (n_1, n_2) \in \mathcal{P}$;
- $(n, s) \in \mathcal{V}^t$ if and only if $n \in N^t$, $(n, s) \in \mathcal{V}$.

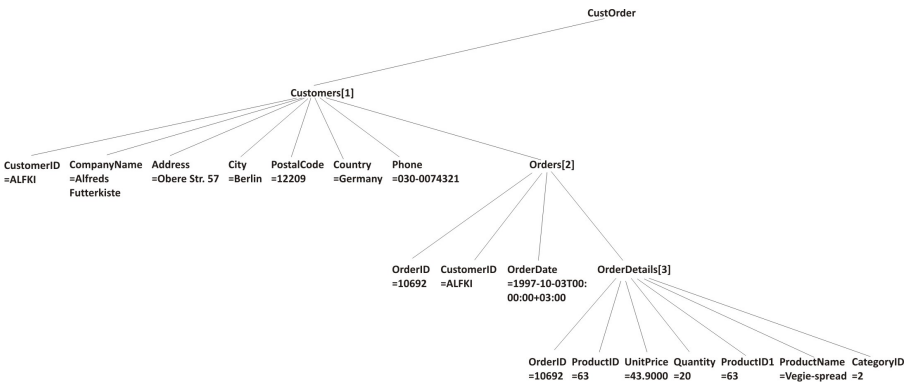


Fig. 3. Example tree tuple

A generalized tree tuple is a data tree projected from the original data tree. It has an extra parameter called a pivot node. In contrast with tree tuple defined in [2], which separate sibling nodes with the same path at all hierarchy levels, the generalized tree tuple separate sibling nodes with the same path above the pivot node. See an example generalized tree tuple of tree from Figure 2 in Figure 3. Based on the pivot node, generalized tree tuples can be categorized into tuple classes:

Definition 6. *(Tuple class)* A tuple class C_p^T of the data tree T is the set of all generalized tree tuples t_n^T, where $path(n) = p$. Path p is called the pivot path.

Definition 7. *(XML FD)* An XML FD is a triple $\langle C_p, LHS, RHS \rangle$, written as $LHS \rightarrow RHS$ w.r.t. C_p, where C_p denotes a tuple class, LHS is a set of paths (P_{li} , $i = [1, n]$) relative to p, and RHS is a single path (P_r) relative to p.

An XML FD holds on a data tree T (or T satisfies an XML FD) if and only if for any two generalized tree tuples $t_1, t_2 \in C_p$

- $\exists i \in [1, n]$, $t_1.P_{li} = \perp$ or $t_2.P_{li} = \perp$, or
- If $\forall i \in [1, n]$, $t_1.P_{li} =_{pv} t_2.P_{li}$, then $t_1.P_r \neq \perp, t_2.P_r \neq \perp, t_1.P_r =_{pv} t_2.P_r$.

A null value, $\perp$, results from a path that matches no node in the tuple, and $=_{pv}$ is the path-value equality defined in Definition 4.

Example 2. (XML FD) In our running example whenever two products agree on `ProductID` values, they have the same `ProductName`. This can be formulated as follows:
`{./ProductID}` $\rightarrow$ `./ProductName` w.r.t $C_{OrderDetails}$
Another example is:
`{./ProductID}` $\rightarrow$ `./CategoryID` w.r.t $C_{OrderDetails}$

In our approach we find the XML keys of a given XML document, so we need the next definition:

Definition 8. *(XML key)* An XML Key of a data tree T is a pair $\langle C_p, LHS \rangle$, where T satisfies the XML FD $\langle C_p, LHS, ./@key \rangle$.

Example 3. We have the XML FD: $\langle C_{Orders}, ./OrderID, ./@key \rangle$, which implies that $\langle C_{Orders}, ./OrderID \rangle$ is an XML key.

Tuple classes with repeatable pivot paths are called *essential tuple classes*.

Definition 9. *(Interesting XML FD)* An XML FD $\langle C_p, LHS, RHS \rangle$ is *interesting* if it satisfies the following conditions:

- RHS $\notin$ LHS;
- C_p is an essential tuple class;
- RHS matches to descendent(s) of the pivot node.

Definition 10. *(XML data redundancy)* A data tree T contains a redundancy if and only if T satisfies an interesting XML FD $\langle C_p, LHS, RHS \rangle$, but does not satisfy the XML Key $\langle C_p, LHS \rangle$.

5 Overview of the Approach

In this section we describe the methodology of a general approach to use FCA to build tools that identify functional dependencies in XML documents. To achieve this, as a first step, we need to define the objects and attributes of interest and create models of XML in terms of FCA context. Our approach is carried out by a sequence of processing steps. The output of each step provides the input to the next step. Every step is illustrated with an example. Our method is supported by a framework named FCAMineXFD. We will now describe each processing step in detail.

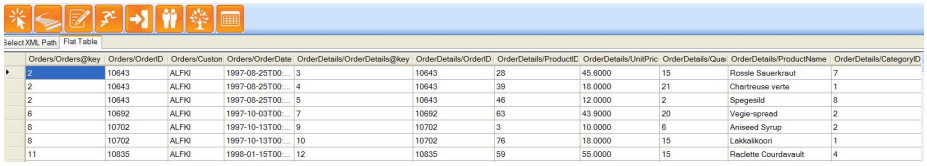
5.1 Constructing the Formal Context, the Input to FCA

In this step the most important issue is how to map the XML document to metamodel entities. Our software can analyze the whole XML document or a *tuple class* C_p given by the path p. Tuple-based XML FD notion proposed in the above section suggests a natural technique for XFD discovery. XML data can be converted into a fully unnested relation, a single relational table, and apply existing FD discovery algorithms directly. Given an XML document, which contains at the beginning the schema of the data, we create generalized tree tuples from it.

Each tree tuple in a tuple class has the same structure, so it has the same number of elements. We use the flat representation which converts the generalized tree tuples into a flat table. Each row in the table corresponds to a tree tuple in the XML tree. In the flat table there are non-leaf and leaf level elements (or attributes) introduced from the tree.

For non-leaf level nodes the associated keys (see Section 4) are used as values.

Example 4. Let us construct the flat table for tuple class C_{Orders}. There are two non-leaf nodes: Orders and OrderDetails. These appear as Orders@key and OrderDetails@key. In Figure 4 we can see the flat representation of tuple class C_{Orders}.


Select XML Path | Flat Table

Orders/Orders@key	Orders/OrderID	Orders/Custon	Orders/OrderDate	OrderDetails/OrderDetails@key	OrderDetails/OrderID	OrderDetails/ProductID	OrderDetails/UnitPric	OrderDetails/Quai	OrderDetails/ProductName	OrderDetails/CategoryID
2	10643	ALFKI	1997-08-25T00:...	3	10643	28	45.6000	15	Rossle Sauerkraut	7
2	10643	ALFKI	1997-08-25T00:...	4	10643	39	18.0000	21	Chartreuse verte	1
2	10643	ALFKI	1997-08-25T00:...	5	10643	46	12.0000	2	Spegesild	8
6	10692	ALFKI	1997-10-03T00:...	7	10692	63	43.9000	20	Vegie-spread	2
8	10702	ALFKI	1997-10-13T00:...	9	10702	3	10.0000	6	Aniseed Syrup	2
8	10702	ALFKI	1997-10-13T00:...	10	10702	76	18.0000	15	Lakkalikoori	1
11	10835	ALFKI	1998-01-15T00:...	12	10835	59	55.0000	15	Raclette Courdavault	4

Fig. 4. Flat table for tuple class C_{Orders}

Applying our experience in detecting functional dependencies in relational databases (see more details in [12]), we use the definitions introduced by Hereth in [11]. Hereth gives the translation from the relational database into a power context family and based on it he defines the formal context of functional dependencies as follows:

Definition 11. Let $\overrightarrow{\mathbb{K}}$ be a power context family, and let $m \in M_k$ be an attribute of the k-th context. Then, the formal context of functional dependencies of m with regard to $\overrightarrow{\mathbb{K}}$ is defined as $FD\left(m, \overrightarrow{\mathbb{K}}\right) := \left(m^{I_k} \times m^{I_k}, \{\ 1, 2, ..., k\}, J\right)$ with $((g,h), i) \in J :\Leftrightarrow \pi_i(g) = \pi_i(h)$ with $g, h \in m^{I_k}$ and $i \in \{\ 1, 2, ..., k\}$.

The π is the relational algebra projection operation. In the next paragraph we will see how we construct this formal context of functional dependencies.

In this step the formal context of functional dependencies for XML data is built, mapping from metamodel entities to FCA objects and attributes.

- Choice of FCA Attributes: *PathEnd/ElementName*
  Due to space considerations we will not specify the whole path to the element (or attribute) names, only the end of the path. FCA attribute names are built from the end of the path to the element: *PathEnd* and element name as follows:
  - for non-leaf level nodes the name of the attribute is constructed as: <ElementName>+"@key" and its value will be the associated key value as specified in Section 4. More elements, which have the same path, will have the same attribute name, but the values will be different.
  - the leaves (actually not the values of the leaves, but the element names of the leaves) of the tree tuple.
- Choice of Objects: the objects are considered to be the tree tuple pairs, actually the tuple pairs of the flat table. The key values associated to non-leaf elements and leaf element's values are used in these tuple pairs.

– Choice of Properties: the mapping between objects and attributes is defined by a binary relation, this incidence relation of the context shows which attributes of this tuple pairs have the same value.

	Orders/Orders@key	Orders/OrderID	Orders/CustomerID	Orders/OrderDate	OrderDetails/OrderDetails@key	OrderDetails/OrderID
2,10643,ALFKI,1997-08-25T00:...	X	X	X	X		X
2,10643,ALFKI,1997-08-25T00:...	X	X	X	X		X
2,10643,ALFKI,1997-08-25T00:...			X			
2,10643,ALFKI,1997-08-25T00:...			X			
2,10643,ALFKI,1997-08-25T00:...			X			
2,10643,ALFKI,1997-08-25T00:...			X			
2,10643,ALFKI,1997-08-25T00:...			X			
2,10643,ALFKI,1997-08-25T00:...			X			
2,10643,ALFKI,1997-08-25T00:...			X			
2,10643,ALFKI,1997-08-25T00:...			X			
2,10643,ALFKI,1997-08-25T00:...			X			

Fig. 5. Beginning of the Formal Context of functional dependencies for tuple class C_{Orders}

The analyzed XML document may have a large number of tree tuples. Creating the tree tuple pairs, our context table may have a very large number of rows, therefore, we need to clear the concepts of irrelevant entities. We filter the tuple pairs and we leave out those pairs in which there are no common attributes, by an operation called "clarifying the context", which does not alter the conceptual hierarchy.

Example 5. The beginning of the formal context of our running example for tuple class C_{Order} can be seen in Figure 5. There are only a few columns of it in the image, due to space considerations. We can see the attributes as column names, like *Orders/Orders@key* (for non-leaf element), *Orders/CustomerID* (for leaf element). Rows contain the tuple pairs, only the beginning of them can be seen. If tuple pairs has the same value for an attribute, then X appears in the context table. This file will be the input for the next step.

5.2 Creating the Concept Lattice

Once the objects and attributes of the context are defined, we run the Concept Explorer (ConExp) [25] engine to generate the concepts and create the concept lattice. The main output produced by FCA is the concept lattice.

Example 6. The concept lattice for the formal context of functional dependencies for XML data constructed in previous step for tuple class C_{Orders} can be seen in Figure 6.

5.3 Processing the Output of FCA

A concept lattice consists of the set of concepts of a formal context and the subconcept-superconcept relation between the concepts, see [9]. Every circle in

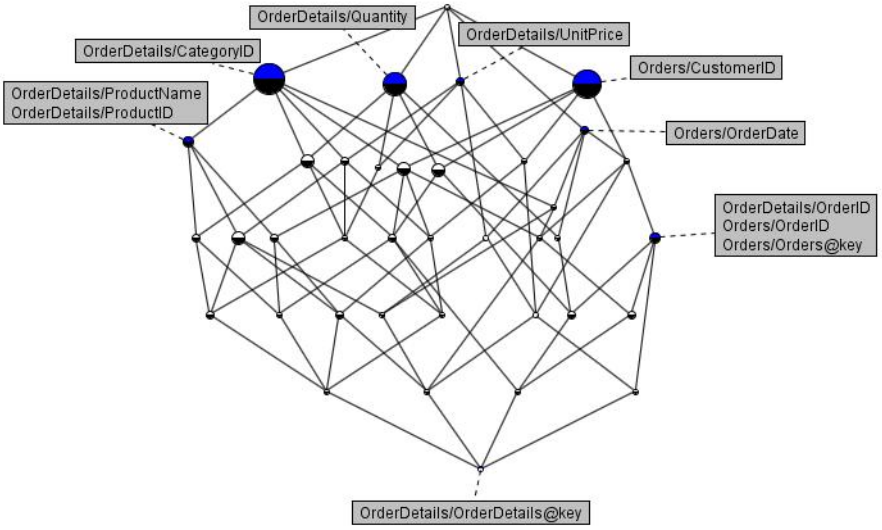


Fig. 6. Concept Lattice of functional dependencies' Formal Context for tuple class C_{Orders}

Figure 6 represents a formal concept. Each concept is a tuple of a set of objects and a set of common attributes, but only the attributes are listed. An edge connects two concepts if one implies the other directly. Each link connecting two concepts represents the transitive subconcept-superconcept relation between them. The top concept has all formal objects in its extension. The bottom concept has all formal attributes in its intension.

Example 7. In Figure 6 node labeled with $Orders/CustomerID$ is on upward path from node labeled by $OrderDetails/OrderID$, $Orders/OrderID$, $Orders/Orders@key$. In FCA language, concept with label $OrderDetails/OrderID$, $Orders/OrderID$, $Orders/Orders@key$ implies concept with label $Orders/CustomerID$.

5.4 Mining XFDs according to the concept hierarchy

In this step, we examine the candidate concepts resulting from the previous steps and use them to explore XFDs. Once the lattice is constructed, we can interpret each concept and generate the list of all functional dependencies.

The relationship between FDs in databases and implications in FCA was pointed out in [9]: a FD $X \rightarrow Y$ holds in a relation r over R iff the implication $X \rightarrow Y$ holds in the context (G, R, I) where $G = \{(t_1, t_2) | t_1, t_2 \in r, t_1 \neq t_2\}$ and $\forall A \in R$, $(t_1, t_2) I A \Leftrightarrow t_1[A] = t_2[A]$.

This means that objects of the context are couples of tuples and each object intent is the agree set of this couple. Thus, the implications in this lattice corresponds to functional dependencies in XML.

Example 8. Analyzing the Conceptual Lattice obtained for tuple class C_{Orders} (Figure 6) we can detect functional dependencies like:

$\langle C_{Orders}, ./OrderID, ./CustomerID \rangle$

$\langle C_{Orders}, ./key, ./CustomerID \rangle$

$\langle C_{Orders}, ./OrderDetail/OrderID, ./CustomerID \rangle$

In the lattice we list only the attributes, these are relevant for our analysis. Let there be a concept, labeled by A, B and a second concept labeled by C. A, B and C are FCA attributes. Let concept labeled by A, B be the subconcept of concept labeled by C. Therefore tuple pairs of concept labeled by A, B have the same values for attributes A, B, but for attribute C too. Tuple pairs of concept labeled by C do not have the same values for attribute A, nor for B, but have the same value for attribute C. Tuple pairs of every subconcept of concept labeled by A, B have the same values for attributes A, B. The labeling of the lattice is simplified by putting each attribute only once, at the highest level. We analyze attributes A and B. If we have only $A \rightarrow B$, then A would be a subconcept of B. If only $B \rightarrow A$ holds then B should be a subconcept of A. We have $A \rightarrow B$ and $B \rightarrow A$, that's why they come side by side in the lattice. So attributes from a concept imply each other.

Example 9. In concept node with label $OrderDetails/OrderID$, $Orders/OrderID$, $Orders/Orders@key$ the associated objects are tree tuple pairs, where the values for $OrderDetails/OrderID$ are the same. So we have the next XML FDs:

$\langle C_{Orders}, ./OrderID, ./OrderDetails/OrderID \rangle$

$\langle C_{Orders}, ./OrderID, ./Orders@key \rangle$

$\langle C_{Orders}, ./Orders@key, ./OrderID \rangle$

$\langle C_{Orders}, ./Orders@key, ./OrderDetails/OrderID \rangle$

$\langle C_{Orders}, ./OrderDetails/OrderID, ./Orders@key \rangle$

$\langle C_{Orders}, ./OrderDetails/OrderID, ./OrderID \rangle$

The functional dependencies found by software FCAMineXFD are in Figure 7.

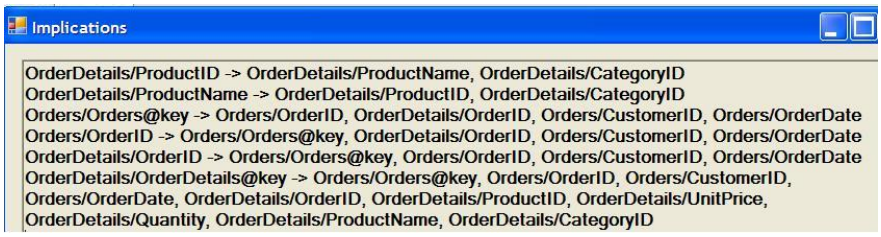


Fig. 7. Functional dependencies in tuple class C_{Orders}

Example 10. The concept lattice for the whole XML document of Example 1 is in Figure 8. We can see the hierarchy of the analyzed data. The node labeled by $Customers/Country$ is on a higher level, than node labeled by $Customers/City$. The Customer's node with every attribute is a subconcept of node labeled $Customers/City$. In our XML data, every customer has different name, address, phone number, so these attributes appear in one concept node and imply each other. The Orders node in XML is child of Customers. In the lattice, the node labeled with the key of Orders node, is subconcept of Customers node, so the hierarchy is visible. These were 1:n relationships, from Country to City, from City to Customers, from Customers to Orders.

The information about products is on the other side of the lattice. Products are in n:m relationship with Customers, linked by OrderDetail node in this case.

Therefore, we say that FCA can serve as a guideline for dependency mining.

5.5 Finding XML keys

The implications found by FCAMineXFD contain some FDs with RHS as ./key values. These can be used to detect the keys in XML.

Example 11. In tuple class C_{Orders} we have XML FD:

$\langle C_{Orders}, ./OrderID, ./@key \rangle$, which implies that $\langle C_{Orders}, ./OrderID \rangle$ is an XML key. Another XML FD is $\langle C_{Orders}, ./OrderDetails/OrderID, ./@key \rangle$, so $\langle C_{Orders}, ./OrderDetails/OrderID \rangle$ is an XML key too.

Example 12. In tuple class $C_{Customers}$ software found XML FD: $\langle C_{Customers}, ./CustomerID, ./@key \rangle$, which implies that $\langle C_{Customers}, ./CustomerID \rangle$ is an XML key. There are more FDs, with RHS as ./key in tuple class $C_{Customers}$, the detected XML keys are: $\langle C_{Customers}, ./Orders/CustomerID \rangle$, $\langle C_{Customers}, ./CompanyName \rangle$, $\langle C_{Customers}, ./Address \rangle$, $\langle C_{Customers}, ./Phone \rangle$.

5.6 Detecting XML data redundancy

Having the set of functional dependencies for XML data in a tuple class, we can detect interesting functional dependencies. In essential tuple class C_{Orders}, the

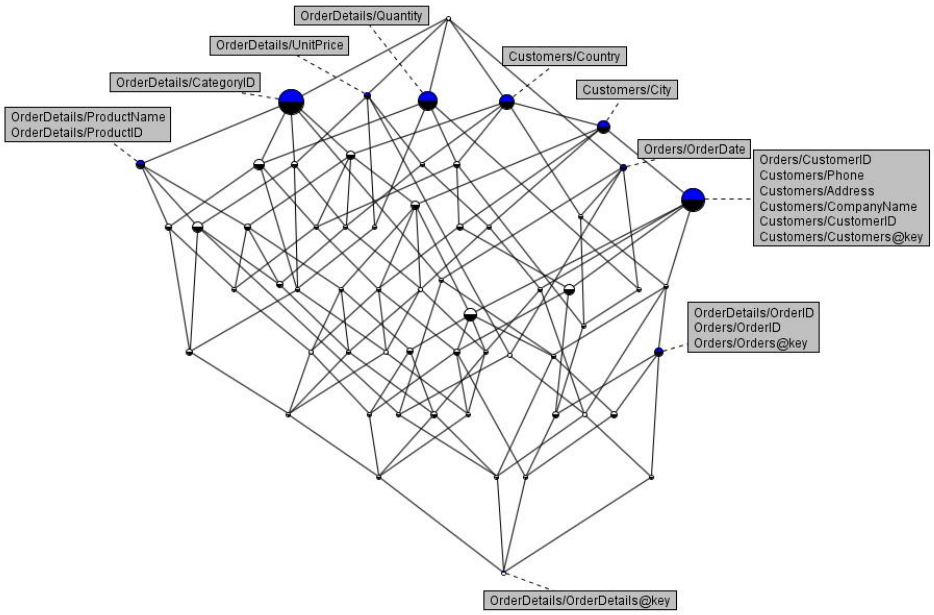


Fig. 8. Conceptual Lattice of CustOrders XML data

XML FD $\langle C_{Orders}, ./OrderDetails/ProductID, ./OrderDetails/ProductName\rangle$ found by the software in Figure 7 is an interesting FD, but $\langle C_{Orders}, ./OrderDetails/ProductID\rangle$ is not an XML key. So it is a data redundancy.

The same reason applies for XML FD $\langle C_{Orders}, ./OrderDetails/ProductName, ./OrderDetails/ProductID\rangle$.

The other XML FD's from Figure 7 have as LHS a key for tuple class C_{Orders}.

6 Conclusion and Future Work

This paper introduces an approach for mining functional dependencies in XML documents based on FCA. We proposed a framework to analyze XML documents using Concept Analysis. Based on the flat representation of XML, we constructed the concept lattice. We analyzed the resulted concepts, which allowed us to discover a number of interesting dependencies. The flat representation is not an advantageous one, since a lot of attributes are repeated for several times, but our goal is not to find a way of storing XML documents efficiently, but rather to find functional dependencies in their data. Our framework offers an interactive visualization for dependency exploration. Taking in consideration our preliminary results, we believe that FCA is a promising technique in XML database design too, but a lot of further research is necessary. We had also previously used FCA to explore functional dependencies in relational databases, see more details in [12]. In that approach, we analyzed the relationships between dependencies in relational databases and implications in FCA. In this paper, we complemented the information with XML design exploration.

We have started to use our approach on several case studies and in the future we plan to analyze in depth the obtained results at different levels of abstraction. Given the set of dependencies discovered by our tool, as a future work we intend to propose a normalization algorithm for converting any XML schema into a correct one.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases. Addison Wesley (1996)
2. Arenas, M., Libkin, L.: A normal form for XML documents. TODS 29(1), 195-232 (2004)
3. Arenas, M., Libkin, L.: An information-theoretic approach to normal forms for relational and XML data. JACM 52(2), 246-283 (2005)
4. Arenas, M., Libkin, L., Fan, W.: On the Complexity of Verifying Consistency of XML Specifications. SIAM J. Comput. 38(3), 841-880 (2008)
5. Baixeries, J.: A formal concept analysis framework to mine functional dependencies. Proceedings of Mathematical Methods for Learning (2004)
6. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.-C.: Keys for XML. In: Proc. WWW, 201-210. Hong Kong, China (2001)
7. Chen, Y., Davidson, S., Hara, C., Zheng, Y.: RRXS:redundancy reducing XML storage in relations. In: VLDB, 189-200 (2003)

8. Embley, D.W., Mok, W.Y.: Developing XML documents with guaranteed "good" properties. In: ER 2001, 20th International Conference on Conceptual Modeling, 426-441 (2001)
9. Ganter, B., Wille, R.: Formal Concept Analysis. Mathematical Foundations. Springer (1999)
10. Hartmann, S.: Axiomatising functional dependencies for XML with frequencies. In: FOIKS, 159-178 (2006)
11. Hereth, J.: Relational Scaling and Databases. Proceedings of the 10th International Conference on Conceptual Structures: Integration and Interfaces LNCS 2393, Springer Verlag, 62-76 (2002)
12. Janosi Rancz, K.T., Varga, V., Puskas, J.: A Software Tool for Data Analysis Based on Formal Concept Analysis. Studia Univ. Babeş-Bolyai, Informatica, 53, 2, 67-78 (2008)
13. Kolahi, S.: Dependency-preserving normalization of relational and XML data. In: DBPL, 247-261 (2005)
14. Lee, M., Ling, T., Low, W.L.: Designing functional dependencies for XML. In: EDBT Conference, 124-141 (2002)
15. Lin, T.W., Lee, M.M., Dobbie, G.: Semistructured Database Design. Springer (2004)
16. Lopes, S., Petit, J-M., Lakhal, L.: Functional and approximate dependency mining: database and FCA points of view. Special issue of Journal of Experimental and Theoretical Artificial Intelligence (JETAI) on Concept Lattices for KDD, 14(2-3): 93-114, Taylor and Francis (2002)
17. Lopes, S., Petit, J-M., Lakhal, L.: Efficient Discovery of Functional Dependencies and Armstrong Relations. Proceedings of the 7th International Conference on Extending Database Technology (EDBT), Konstanz, Germany (2000)
18. Novelli, N., Cicchetti, R.: Functional and embedded dependency inference: a data mining point of view. IS, 26(7): 477-506 (2001)
19. Schewe, K.D.: Redundancy, dependencies and normal forms for XML databases, In: ADC, 7-16 (2005)
20. Vincent, M. W., Liu, J., Mohania, M.: On the Equivalence between FDs in XML and FDs in Relations. Acta Informatica 44(3-4), 207-247 (2007)
21. Vincent, M. W., Liu, J., Liu, C.: Strong functional dependencies and their application to normal forms in XML. ACM TODS, 29(3): 445-462 (2004)
22. Wang, J.,Topor, R.: Removing XML data redundancies using functional and equality-generating dependencies, In: ADC, 65-74 (2005)
23. Wille, R.: Restructuring lattice theory: an approach based on hierarchies of concepts. In I. Rival (ed.) Ordered sets, Reidel, Dordrecht, Boston, 445-470 (1982)
24. W3C. XML Schema, http://www.w3.org/TR/xmlschema-0/ (2004)
25. Serhiy Yevtushenko, A.: System of data analysis "Concept Explorer". (In Russian). Proceedings of the 7th National Conference on Artificial Intelligence KII-2000, Russia, 127-134 (2000)
26. Yu, C., Jagadish, H. V.: XML schema refinement through redundancy detection and normalization. VLDB J. 17(2): 203-223 (2008)