Redundancy Free Mappings from relations to XML

Millist W. Vincent, Jixue Liu, and Chengfei Liu

School of Computer and Information Science
University of South Australia
{millist.vincent, jixue.liu, chengfei.liu }@unisa.edu.au

Abstract. Given the fact that relational and object-relational databases are the most widely used technology for storing data and that XML is the standard format used in electronic data interchange, the process of converting relational data to XML documents is one that occurs frequently. The problem that we address in this paper is an important one related to this process. If we convert a relation to an XML document, under what circumstances is the XML document redundancy free? Drawing on some previous work by the authors that formally defined functional dependencies and redundancy in XML documents, we show that for a very general class of mappings from a relation to an XML document, the XML document is always redundancy free if and only if the relation is in Boyce-Codd normal form (BCNF).

1 Introduction

The eXtensible Markup Language (XML) [5] has recently emerged as a standard for data representation and interchange on the Internet [14, 1]. As a result of this and the fact that relational and object-relational databases are the standard technology in commercial applications, the issue of converting relational data to XML data is one that frequently occurs. In this conversion process of relational data to XML data, there many different ways that relational data can be mapped to XML data, especially considering the flexible nesting structures that XML allows. This gives rise to the following important problem. Are some mappings 'better' than others? Firstly, one has to make precise what is meant by 'better'. In this paper we extend the classical approach used in relational database design and regard a good design as one which eliminates redundancy. The relationship between normal forms and redundancy elimination has been investigated, both for the relational case [11, 8, 10] and the nested relational case [2], and in particular it has been shown that *Boyce-Codd normal form* (BCNF) [7] is a necessary and sufficient condition for the elimination of redundancy in relations when the only constraints are *functional dependencies* (FDs). However, this approach to determining good database designs depends on having FDs defined in relations. In some recent work [12, 13], we showed how to extend the definition of FDs in relations to FDs in XML (*called XFDs*). Since this current paper depends heavily on this work, we first outline the contributions of this previous work.

The definition of an XFD was proposed in [12,13] and justified formally by showing that for a very general class of mappings from a relation to an XML document, a relation satisfies a unary FD (only one attribute on the l.h.s. of the FD) if and only if the corresponding XML document satisfies the corresponding XFD. Thus there is a natural correspondence between FDs in relations and XFDs in XML documents. The other contributions of [12] were firstly to define a set of axioms for reasoning about the implication of XFDs and to show that the axioms are sound for arbitrary XFDs. The final contribution was to define a normal form, based on a modification of the one proposed in [3], and prove that it is a necessary and sufficient condition for the elimination of redundancy in an XML document.

In this paper we address the following problem. Suppose we are given a single relation and wish to map it to an XML document. There are many such mappings and in particular a deeply nested structure, rather than a flat structure, may be chosen because it better represents the semantics of the data. We then want to determine what mappings result in the XML document being redundancy free. Knowing this is important for systems designers because they would obviously wish to avoid mappings which result in the introduction of redundancy to the XML document. The class of mappings that we consider is a very general class of mappings from a relation into an XML document first proposed in [12,13]. The class takes a relation, first converts it into a nested relation by allowing an *arbitrary* sequence of nest operations and then converts the nested relation into an XML document. This is a very general class of mappings and we believe that it covers all the types of mappings that are likely to occur in practice. The main result of the paper then shows that, for the case where all FDs in the relation are unary, any mapping from the general class of mappings from a relation to an XML document will always be redundancy free if and only if the relation is in BCNF. This result is of reassurance to system designers because it allows them a great degree of flexibility in determining how to map a relation into an XML document, and thus they can make their mapping decision on other criteria apart from eliminating redundancy. We also note, importantly, that if the relation is not in BCNF, then some mappings in the general class considered produce redundancy free XML documents, whereas others produce XML documents with redundancy.

2 Preliminary Definitions

In this section we present some preliminary definitions that we need before defining XFDs. We model an XML document as a tree as follows.

Definition 1. *Assume a countably infinite set* **E** *of element labels (tags), a countable infinite set* **A** *of attribute names and a symbol* S *indicating text. An* XML tree *is defined to be* $T = (V, lab, ele, att, val, v_r)$ *where* V *is a* finite *set of* *nodes in T; lab is a function from V to $\mathbf{E} \cup \mathbf{A} \cup \{S\}$; ele is a partial function from V to a sequence of V nodes such that for any $v \in V$, if $ele(v)$ is defined then $lab(v) \in \mathbf{E}$; att is a partial function from $V \times \mathbf{A}$ to V such that for any $v \in V$ and $l \in \mathbf{A}$, if $att(v,l) = v_1$ then $lab(v) \in \mathbf{E}$ and $lab(v_1) = l$; val is a function such that for any node in $v \in V, val(v) = v$ if $lab(v) \in \mathbf{E}$ and $val(v)$ is a string if either $lab(v) = S$ or $lab(v) \in \mathbf{A}$; v_r is a distinguished node in V called the root of T and we define $lab(v_r) = root$. Since node identifiers are unique, a consequence of the definition of val is that if $v_1 \in \mathbf{E}$ and $v_2 \in \mathbf{E}$ and $v_1 \neq v_2$ then $val(v_1) \neq val(v_2)$. We also extend the definition of val to sets of nodes and if $V_1 \subseteq V$, then $val(V_1)$ is the set defined by $val(V_1) = \{val(v) | v \in V_1\}$.*

For any $v \in V$, if $ele(v)$ is defined then the nodes in $ele(v)$ are called subelements *of v. For any $l \in \mathbf{A}$, if $att(v,l) = v_1$ then v_1 is called an* attribute *of v. Note that an XML tree T must be a tree. Since T is a tree the set of ancestors of a node v, is denoted by $Ancestor(v)$. The children of a node v are also defined as in Definition 1 and we denote the parent of a node v by $Parent(v)$.*

We note that our definition of *val* differs slightly from that in [6] since we have extended the definition of the *val* function so that it is also defined on element nodes. The reason for this is that we want to include in our definition paths that do not end at leaf nodes, and when we do this we want to compare element nodes by node identity, i.e. node equality, but when we compare attribute or text nodes we want to compare them by their contents, i.e. value equality. This point will become clearer in the examples and definitions that follow.

We now give some preliminary definitions related to paths.

Definition 2. *A* path *is an expression of the form $l_1.\cdots.l_n$, $n \geq 1$, where $l_i \in \mathbf{E} \cup \mathbf{A} \cup \{S\}$ for all $i, 1 \leq i \leq n$ and $l_1 = root$. If p is the path $l_1.\cdots.l_n$ then $Last(p) = l_n$.*

For instance, if $\mathbf{E} = \{$`root, Division, Employee`$\}$ and $\mathbf{A} = \{$`D#, Emp#`$\}$ then `root, root.Division, root.Division.D#,`
`root.Division.Employee.Emp#.S` are all paths.

Definition 3. *Let p denote the path $l_1.\cdots.l_n$. The function $Parnt(p)$ is the path $l_1.\cdots.l_{n-1}$. Let p denote the path $l_1.\cdots.l_n$ and let q denote the path $q_1.\cdots.q_m$. The path p is said to be a* prefix *of the path q, denoted by $p \subseteq q$, if $n \leq m$ and $l_1 = q_1, \ldots, l_n = q_n$. Two paths p and q are equal, denoted by $p = q$, if p is a prefix of q and q is a prefix of p. The path p is said to be a* strict prefix *of q, denoted by $p \subset q$, if p is a prefix of q and $p \neq q$. We also define the intersection of two paths p_1 and p_2, denoted but $p_1 \cap p_2$, to be the maximal common prefix of both paths. It is clear that the intersection of two paths is also a path.*

For example, if $\mathbf{E} = \{$`root, Division, Employee`$\}$ and $\mathbf{A} = \{$`D#, Emp#`$\}$ then `root.Division` is a strict prefix of `root.Division.Employee` and
`root.Division.D#` $\cap$ `root.Division.Employee.Emp#.S` $=$ `root.Division`.

Definition 4. *A* path instance *in an XML tree T is a sequence $\bar{v}_1.\cdots.\bar{v}_n$ such that $\bar{v}_1 = v_r$ and for all $\bar{v}_i, 1 < i \leq n, v_i \in V$ and $\bar{v}_i$ is a child of $\bar{v}_{i-1}$. A* *path instance* $\bar{v}_1.\cdots.\bar{v}_n$ *is said to be* defined *over the path* $l_1.\cdots.l_n$ *if for all* $\bar{v}_i, 1 \leq i \leq n$, $lab(\bar{v}_i) = l_i$. *Two path instances* $\bar{v}_1.\cdots.\bar{v}_n$ *and* $\bar{v}'_1.\cdots.\bar{v}'_n$ *are said to be* distinct *if* $v_i \neq v'_i$ *for some* i, $1 \leq i \leq n$. *The path instance* $\bar{v}_1.\cdots.\bar{v}_n$ *is said to be a* prefix *of* $\bar{v}'_1.\cdots.\bar{v}'_m$ *if* $n \leq m$ *and* $\bar{v}_i = \bar{v}'_i$ *for all* $i, 1 \leq i \leq n$. *The path instance* $\bar{v}_1.\cdots.\bar{v}_n$ *is said to be a* strict prefix *of* $\bar{v}'_1.\cdots.\bar{v}'_m$ *if* $n < m$ *and* $\bar{v}_i = \bar{v}'_i$ *for all* $i, 1 \leq i \leq n$. *The set of path instances over a path* p *in a tree* T *is denoted by* $Paths(p)$

For example, in Figure 1, $v_r.v_1.v_3$ is a path instance defined over the path `root.Division.Section` and $v_r.v_1.v_3$ is a strict prefix of $v_r.v_1.v_3.v_4$

We now assume the existence of a set of legal paths P for an XML application. Essentially, P defines the semantics of an XML application in the same way that a set of relational schema define the semantics of a relational application. P may be derived from the DTD, if one exists, or P be derived from some other source which understands the semantics of the application if no DTD exists. The advantage of assuming the existence of a set of paths, rather than a DTD, is that it allows for a greater degree of generality since having an XML tree conforming to a set of paths is much less restrictive than having it conform to a DTD. Firstly we place the following restriction on the set of paths.

Definition 5. *A set* P *of paths is* consistent *if for any path* $p \in P$, *if* $p_1 \subset p$ *then* $p_1 \in P$.

This is natural restriction on the set of paths and any set of paths that is generated from a DTD will be consistent.

We now define the notion of an XML tree conforming to a set of paths P.

Definition 6. *Let* P *be a consistent set of paths and let* T *be an XML tree. Then* T *is said to* conform *to* P *if every path instance in* T *is a path instance over a path in* P.

The next issue that arises in developing the machinery to define XFDs is the issue is that of missing information. This is addressed in [12] but in this paper, because of space limitations, we take the simplifying assumption that there is no missing information in XML trees. More formally, we have the following definition.

Definition 7. *Let* P *be a consistent set of paths, let* T *be an XML that conforms to* P. *Then* T *is defined to be* complete *if whenever there exist paths* p_1 *and* p_2 *in* P *such that* $p_1 \subset p_2$ *and there exists a path instance* $\bar{v}_1.\cdots.\bar{v}_n$ *defined over* p_1, *in* T, *then there exists a path instance* $\bar{v}'_1.\cdots.\bar{v}'_m$ *defined over* p_2 *in* T *such that* $\bar{v}_1.\cdots.\bar{v}_n$ *is a prefix of the instance* $\bar{v}'_1.\cdots.\bar{v}'_m$.

For example, if we take P to be {`root, root.Dept, root.Dept.Section, root.Dept.Section.Emp, root.Dept.Section.Project`} then the tree in Figure 1 conforms to P and is complete.

The next function returns all the final nodes of the path instances of a path p in T.

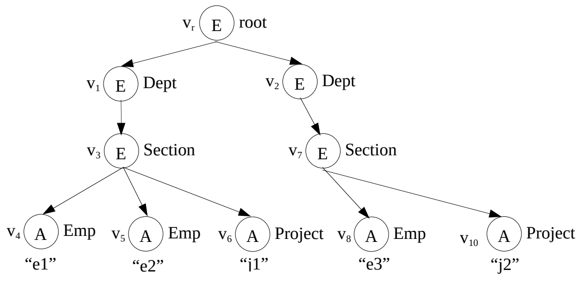

Fig. 1. A complete XML tree.

Definition 8. *Let P be a consistent set of paths, let T be an XML tree that conforms to P . The function $N(p)$, where $p \in P$, is the set of nodes defined by $N(p) = \{\bar{v} | \bar{v}_1 . \cdots . \bar{v}_n \in Paths(p) \wedge \bar{v} = \bar{v}_n\}$.*

For example, in Figure 1, $N($`root.Dept`$) = \{v_1, v_2\}$.
We now need to define a function that returns a node and its ancestors.

Definition 9. *Let P be a consistent set of paths, let T be an XML tree that conforms to P. The function $AAncestor(v)$, where $v \in V \cup \mathbf{N}$, is the set of nodes in T defined by $AAncestor(v) = v \cup Ancestor(v)$.*

For example in Figure 1, $AAncestor(v_3) = \{v_r, v_1, v_3\}$. The next function returns all nodes that are the final nodes of path instances of p and are descendants of v.

Definition 10. *Let P be a consistent set of paths, let T be an XML tree that conforms to P. The function $Nodes(v, p)$, where $v \in V \cup \mathbf{N}$ and $p \in P$, is the set of nodes in T defined by $Nodes(v, p) = \{x | x \in N(p) \wedge v \in AAncestor(x)\}$*

For example, in Figure 1, $Nodes(v_1,$`root.Dept.Section.Emp`$) = \{v_4, v_5\}$. We also define a partial ordering on the set of nodes as follows.

Definition 11. *The partial ordering $>$ on the set of nodes V in an XML tree T is defined by $v_1 > v_2$ iff $v_2 \in Ancestor(v_1)$.*

3 Strong Functional Dependencies in XML

We recall the definition of an XFD from [12]. For simplicity, we consider the case where there is only one path on the l.h.s.

Definition 12. *Let P be a set of consistent paths and let T be an XML tree that conforms to P and is complete. An XML* functional dependency (XFD) *is a statement of the form: $p \rightarrow q$ where $p \in P$ and $q \in P$. T* strongly satisfies *the XFD if $p = q$ or for any two distinct path intances $\bar{v}_1 . \cdots . \bar{v}_n$ and $\bar{v}'_1 . \cdots . \bar{v}'_n$ in*

$Paths(q)$ in T, $val(\bar{v}_n) \neq val(\bar{v}'_n)) \Rightarrow val(Nodes(x_1,p)) \cap val(Nodes(y_1,p)) = \emptyset$), where $x_1 = \max\{v | v \in \{\bar{v}_1, \cdots, \bar{v}_n\} \wedge v \in N(p \cap q)\}$ and $y_1 = \max\{v | v \in \{\bar{v}'_1, \cdots, \bar{v}'_n\} \wedge v \in N(p \cap q)\}$.

We note that since the path $p_i \cap q$ is a prefix of q, there exists only one node in $\bar{v}_1. \cdots .\bar{v}_n$ that is also in $N(p_i \cap q)$ and so x_i is always defined and unique. Similarly for y_i.

We now illustrate the definition by some an example.

Example 1. Consider the XML tree shown in Figure 2 and the XFD `root.Department.Lecturer.Lname` $\rightarrow$ `root.Department.Lecturer.Subject.SubjName.S`. Then $v_r.v_1.v_5.v_{13}.v_{17}.v_{22}$ and $v_r.v_2.v_9.v_{15}.v_{21}.v_{24}$ are two distinct path instances in $Paths($`root.Department.Lecturer.Subject.SubjName.S`$)$ and $val(v_{22})$ = `"n1"` and $val(v_{24})$ = `"n2"`. So $N($`root.Department.Lecturer.Lname`$\cap$ `root.Department.Lecturer.Subject.SubjName.S`$) = \{v_5, v_6, v_9\}$ and so $x_1 = v_5$ and $y_1 = v_9$. Thus $val(Nodes(x_1,$`root.Department.Lecturer.Lname`$))$ $= \{$`"l1"`$\}$ and $val(Nodes(y_1,$`root.Department.Lecturer.Lname`$)) = \{$`"l1"`$\}$ and so the XFD is violated. We note that if we change *val* of node v_{10} in Figure 2 to `"l3"` then the XFD is satisfied.

Consider next the XFD `root.Department.Head` $\rightarrow$ `root.Department`. Then $v_r.v_1$ and $v_r.v_2$ are two distinct paths instances in $Paths($`root.Department`$)$ and $val(v_1) = v_1$ and $val(v_2) = v_2$. Also $N($`root.Department.Head` $\cap$ `root.Department`$) = \{v_1, v_2\}$ and so $x_1 = v_1$ and $y_1 = v_2$. Thus $val(Nodes(x_1,$`root.Department.Head`$)) = \{$`"h1"`$\}$ and $Val(Nodes(y_1,$`root.Department.Head`$)) = \{$`"h2"`$\}$ and so the XFD is satisfied. We note that if we change *val* of node v_8 in Figure 2 to `"h1"` then the XFD is violated.

4 Mapping from relations to XML

As our technique for mapping relations to XML Trees is done via nested relations, we firstly present the definitions for nested relations.

Let U be a fixed countable set of atomic attribute names. Associated with each attribute name $A \in U$ is a countably infinite set of values denoted by $DOM(A)$ and the set **DOM** is defined by **DOM** $= \cup DOM(A_i)$ for all $A_i \in U$. We assume that $DOM(A_i) \cap DOM(A_j) = \emptyset$ if $i \neq j$. A *scheme tree* is a tree containing at least one node and whose nodes are labelled with nonempty sets of attributes that form a partition of a finite subset of U. If n denotes a node in a scheme tree S then:

- $ATT(n)$ is the set of attributes associated with n;
- $A(n)$ is the union of $ATT(n_1)$ for all $n_1 \in Ancestor(n)$.

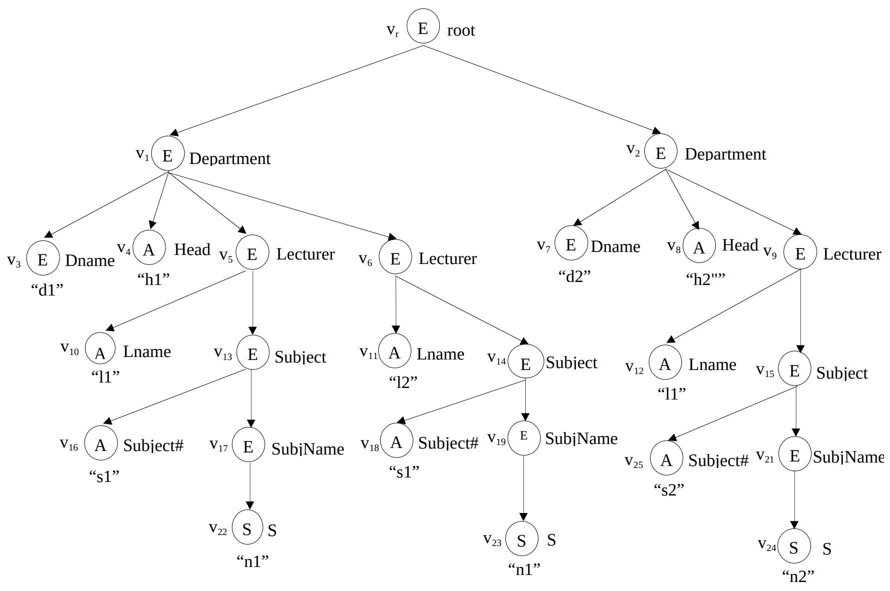

Fig. 2. A XML tree illustrating the definition of an XFD

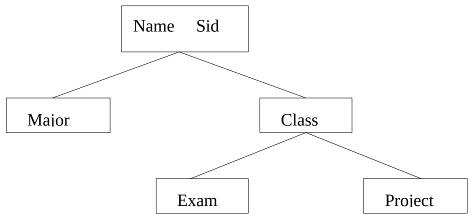

Fig. 3. A scheme tree

Figure 3 illustrates an example scheme tree defined over the set of attributes {`Name, Sid, Major, Class, Exam, Project`}.

Definition 13. *A* nested relation scheme *(NRS) for a scheme tree S, denoted by $N(S)$, is the set defined recursively by:*

(i) If S consists of a single node n then $N(S) = ATT(n)$;

(ii) If $A = ATT(ROOT(S))$ and $S_1, \cdots, S_k, k \geq 1$, are the principal subtrees of S then $N(S) = A \cup \{N(S_1)\} \cdots \{N(S_k)\}$.

For example, for the scheme tree S shown in Figure 3, $N(S) =$ {`Name, Sid`, {`Major`}, {`Class`, {`Exam`}, {`Project`}}}. We now recursively define the domain of a scheme tree S, denoted by $DOM(N(S))$.

Definition 14. *(i) If S consists of a single node n with $ATT(n) = \{A_1, \cdots, A_n\}$ then $DOM(N(S)) = DOM(A_1) \times \cdots \times DOM(A_n)$;*

(ii) If $A = ATT(ROOT(S))$ and $S_1, \cdots, S_k$ are the principal subtrees of S, then $DOM(N(S)) = DOM(A) \times P(DOM(N(S_1))) \times \cdots \times P(DOM(N(S_k)))$ where $P(Y)$ denotes the set of all nonempty, finite subsets of a set Y.

The set of *atomic attributes* in $N(S)$, denoted by $Z(N(S))$, is defined by $Z(N(S)) = N(S) \cap U$. The set of higher order attributes in $N(S)$, denoted by $H(N(S))$, is defined by $H(N(S)) = N(S) - Z(N(S))$. For instance, for the example shown in Figure 3, $Z(N(S)) = \{$`Name, Sid`$\}$ and $H(N(S)) = \{\{$`Major`$\}, \{$`Class`$, \{$`Exam`$\}, \{$`Project`$\}\}\}$.

Finally we define a nested relation over a nested relation scheme $N(S)$, denoted by $r^*(N(S))$, or often simply by r^* when $N(S)$ is understood, to be a finite nonempty set of elements from $DOM(N(S))$. If t is a tuple in r^* and Y is a nonempty subset of $N(S)$, then $t[Y]$ denotes the restriction of t to Y and the restriction of r^* to Y is then the nested relation defined by $r^*[Y] = \{t[Y] | t \in r\}$. An example of a nested relation over the scheme tree of Figure 3 is shown in Figure 4.

A tuple t_1 is said to be a *subtuple* of a tuple t in r^* if there exists $Y \in H(N(S))$ such that $t_1 \in t[Y]$ or there exists a tuple t_2, defined over some NRS N_1, such that t_2 is a subtuple of t and there exists $Y_1 \in H(N_1)$ such that $t_1 \in t_2[Y_1]$. For example in the relation shown in Figure 4 the tuples

$<$ `CS100`$, \{$`mid-year, final`$\}, \{$`Project A, Project B, Project C`$\} >$ and $<$ `Project A` $>$ are both subtuples of

$<$ `Anna`, `Sid1`$, \{$`Maths, Computing`$\}, \{$`CS100`$, \{$`mid-year, final`$\},$
$\{$`Project A, Project B,Project C`$\}\} >$.

Name	**Sid**	**{Major}**	**{Class**	**{Exam}**	**{Project}}**
Anna	Sid1	Maths	CS100	Mid-year	Project A
		Computing		Final	Project B
					Project C
Bill	Sid2	Physics	P100	Final	Prac 1
					Prac 2
		Chemistry	CH200	Test A	Experiment 1
				Test B	Experiment 2 1

Fig. 4. A nested relation.

We assume that the reader is familiar with the definition of the nest operator, $\nu_Y(r^*)$, and the unnest operator, $\mu_{\{Y\}}(r^*)$, for nested relations as defined in [9, 4].

The translation of a relation into an XML tree consists of two phases. In the first we map the relation to a nested relation whose nesting structure is arbitrary and then we map the nested relation to an XML tree.

In the first step we let the nested relation r^* be defined by $r_i = \nu_{Y_{i-1}}(r_{i-1}), r_0 = r, r^* = r_n, 1 \leq i \leq n$ where r represents the initial (flat) relation and r^* represents the final nested relation. The Y_i are allowed to be arbitrary, apart from the obvious restriction that Y_i is an element of the NRS for r_i.

In the second step of the mapping procedure we take the nested relation and convert it to an XML tree as follows. We start with an initially empty tree. For each tuple t in r^* we first create an element node of type `Id` and then for each $A \in Z(N(r^*))$ we insert a single attribute node with a value $t[A]$. We then repeat recursively the procedure for each subtuple of t. The final step in the procedure is to compress the tree by removing all the nodes containing nulls from the tree. We now illustrate these steps by an example.

Example 2. Consider the flat relation shown in Figure 5.

Name	**Sid**	**Major**	**Class**	**Exam**	**Project**
Anna	Sid1	Maths	CS100	Mid-year	Project A
Anna	Sid1	Maths	CS100	Mid-year	Project B
Anna	Sid1	Maths	CS100	Final	Project A
Anna	Sid1	Maths	CS100	Final	Project B

Fig. 5. A flat relation.

If we then transform the relation r in Figure 5 by the sequence of nestings $r_1 = \nu_{PROJECT}(r)$, $r_2 = \nu_{EXAM}(r_1)$, $r_3 = \nu_{CLASS,\{EXAM\},\{PROJECT\}}(r_2)$, $r^* = \nu_{MAJOR}(r_3)$ then the relation r^* is shown in Figure 6. We then transform the nested relation in Figure 6 to the XML tree shown in Figure 7

Name	**Sid**	**{Major}**	**{Class**	**{Exam}**	**{Project}}**
Anna	Sid1	Maths	CS100	Mid-year	Project A
				Final	Project B

Fig. 6. A nested relation derived from a flat relation.

We now recall the result from [12] which establishes the correspondence between satisfaction of FDs in relations and satisfaction of XFDs in XML. We denote by T_{r^*} the XML tree derived from r^*.

Theorem 1. *Let r be a flat relation and let $A \to B$ be a FD defined over r. Then r strongly satisfies $A \to B$ iff T_{r^*} strongly satisfies $p_A \to q_B$ where p_A denotes the path in T_{r^*} to reach A and q_B denotes the path to reach B.*

5 Redundancy free mappings from relations to XML

We now give our definition of redundancy taken from [12]. Firstly, let us denote by P_Σ the set of paths that appear on the l.h.s. or r.h.s. of any XFD in Σ, the set of XFDs for the application.

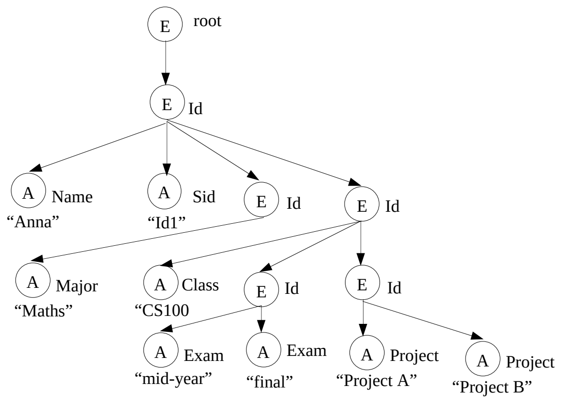

Fig. 7. A XML tree derived from a nested relation

Definition 15. *Let T be an XML tree and let v be a node in T. Then the change from v to v', resulting in a new tree T', is said to be a* valid change *if $v \neq v'$ and $val(v) \neq val(v')$.*

We note that the second condition in the definition, $val(v) \neq val(v')$, is automatically satisfied if the first condition is satisfied when $lab(v) \in \mathbf{E}$.

Definition 16. *Let P be a consistent set of paths and let Σ be a set of XFDs such that $P_\Sigma \subseteq P$ and let T be an XML tree that conforms to P and satisfies Σ. Then T is defined to* contain redundancy *if there exists a node v in T such that* every *valid change from v to v', resulting in a new XML tree T', causes Σ to be violated.*

We now illustrate this definition by an example.

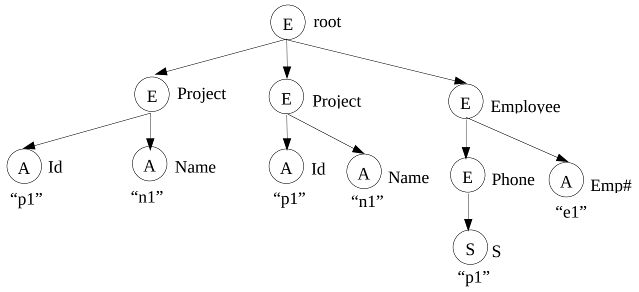

Fig. 8. XML tree illustrating redundancy.

Example 3. Let P be the set of paths {`root, root.Project, root.project.Id, root.Project.Name, root.Employee, root.Employee.Phone, root.Employee.Emp#, root.Employee.Phone.S`}. Consider the set of Σ of XFDs {`root.Project.Id` $\rightarrow$ `root.Project.Name`} and the XML tree T shown in Figure 8. Then T contains redundancy because T is consistent with P and satisfies Σ yet every valid change to either of the `Name` nodes results in `root.Project.Id` $\rightarrow$ `root.Project.Name` being violated.

One important benefit of an XML tree being redundancy free, as we shall now show, is that it eliminates certain update problems in a similar fashion to the way that eliminating redundancy in relations eliminates update problems [11].

Definition 17. *Let P be a consistent set of paths and let Σ be a set of XFDs such that $P_\Sigma \subseteq P$ and let T be an XML tree that conforms to P and satisfies Σ. Then T is defined to have a* modification anomaly *if there exists a node v in T such that there exists some valid change to v that results in Σ being violated.*

For instance, the tree in Figure 8 has a modification anomaly since the change of the *val* of either of the `Name` nodes to `''n2''` results in Σ being violated. We then have the following important result.

Theorem 2. *Let P be a consistent set of paths and let Σ be a set of XFDs such that $P_\Sigma \subseteq P$ and let T be an XML tree that conforms to P and satisfies Σ. Then T has no redundancy iff T has no modification anomaly.*

Proof.
If: The contrapositive, that if T contains redundancy then it has a modification anomaly follows directly from the definitions.
Only If: We shall show the contrapositive that if T has a modification anomaly then it contains redundancy. It follows directly from the definition of an XFD is that if one valid change to v results in the violation of Σ then all valid changes to v result in the violation of Σ. Thus if T has a modification anomaly then it will also contain redundancy. □

Next, we have the main result of the paper which shows that all mappings from a relation to an XML tree are redundancy free provided that the relation scheme is in BCNF.

Theorem 3. *Let Ω denote the set of all mappings from relations to XML trees as defined in Section 4. Let $R(A_1, \ldots, A_n)$ denote a relation scheme, let Σ_R denote a set of unary FDs defined over R and let $rel(R)$ denote the set of all relations defined over R which satisfy Σ_R. Let T_Ω be the set defined $T_\Omega = \{T | \exists r \in rel(R) \exists \omega \in \Omega(T = \omega(r))\}$. Then every tree in T_Ω is redundancy free iff R is in BCNF.*

Proof. See Appendix. □

We note that in the case of the relational scheme not being in BCNF, then some mappings result in redundancy whereas others are redundancy free. This is shown in the following example.

Example 4. Consider the relation scheme $R(A, B, C)$, the set Σ of FDs $\{A \rightarrow B\}$ and the relation r defined over R shown in Figure 9. Suppose we then map r to an XML document in two ways. In the first, we use the mapping ω_1 which does no nesting. The resulting tree is shown in Figure 10 (a). This tree contains redundancy since any valid change to either of the B nodes results in the violation of the XFD `root.Id.A` $\rightarrow$ `root.Id.B`. In the second mapping, ω_2, we first nest on C and then on B then convert to a tree. The resulting tree is shown in Figure 10 (b). This tree contains no redundancy since every valid change to the B node results in the XFD `root.Id.A` $\rightarrow$ `root.Id.Id..B` still being satisfied.

A	**B**	**C**
a_1	b_1	c_1
a_1	b_1	c_2

Fig. 9. A flat relation.

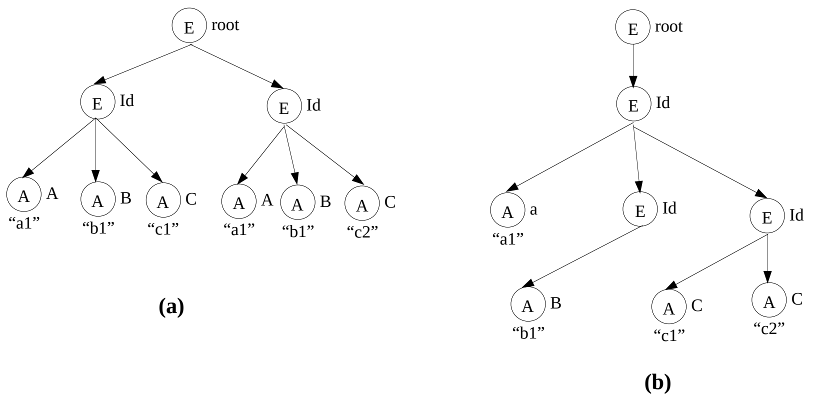

Fig. 10. XML trees from different mappings.

6 Conclusions

The problem that we have addressed in this paper is one related to this process of exporting relational data in XML format. The problem is that if one converts a relation to an XML document, under what circumstances is the XML document redundancy free? Being redundancy free is an important property of an XML document since, as we show, it guarantees the absence of certain types of update anomalies in the same fashion that redundancy elimination and BCNF ensures the elimination of update anomalies in relational databases [11].

Drawing on some previous work by the authors [13, 12] that formally defined functional dependencies and redundancy in XML documents, we show that for a very general class of mappings from a relation to an XML document, the XML document is always redundancy free if and only if the relation is in Boyce-Codd normal form (BCNF). This result gives systems designers a great degree of flexibility in deciding how to map relations to XML without introducing redundancy. We also show that if the relation is not in BCNF then some mappings produce XML documents with redundancy whereas other mappings produce redundancy free XML documents.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web.* Morgan Kauffman, 2000.
2. W.Y. Mok anmd Y.K. Ng and D. Embley. A normal form for precisely characterizing redundancy in nested relations. *ACM Transactions on Database Systems*, 21(1):77 – 106, 1996.
3. M. Arenas and L. Libkin. A normal form for xml documents. In *Proc. ACM PODS Conference*, pages 85–96, 2002.
4. P. Atzeni and V. DeAntonellis. *Foundations of databases.* Benjamin Cummings, 1993.
5. T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible markup language (xml) 1.0. Technical report, http://www.w3.org/Tr/1998/REC-xml-19980819, 1998.
6. P. Buneman, S. Davidson, W. Fan, and C. Hara. Reasoning about keys for xml. In *International Workshop on Database Programming Languages*, 2001.
7. E.F. Codd. Recent investigations in relational database systems. In *IFIP Conference*, pages 1017 –1021, 1974.
8. M. Levene and M. W. Vincent. Justification for inclusion dependency normal form. *IEEE Transactions on Knowledge and Data Engineering*, 12:281 –291, 2000.
9. S.J. Thomas and P.C. Fischer. Nested relational structures. In P. Kanellakis, editor, *The theory of databases*, pages 269 –307. JAI Press, 1986.
10. M. W. Vincent. A new redundancy free normal form for relational database design. In B. Thalheim and L. Libkin, editors, *Database Semantics*, pages 247 –264. Springer Verlag, 1998.
11. M. W. Vincent. Semantic foundations of 4nf in relational database design. *Acta Informatica*, 36:1 –41, 1999.
12. M.W. Vincent and J. Liu. Strong functional dependencies and a redundancy free normal form for xml. Submitted for publication, 2002.
13. M.W. Vincent and J. Liu. Functional dependencies for xml. In *Fifth Asian Pacific Web Conference*, 2003.
14. J. Widom. Data management for xml - research directions. *IEEE data Engineering Bulletin*, 22(3):44–52, 1999.

7 Appendix

Before proving Theorem 3, we need some preliminary definitions and lemmas.

Definition 18. *Let A and B be attributes in U and let S be a scheme tree defined over U. Then A and B are defined to be* siblings *if A and B are members of the label for a node, A is an* ancestor *of B is if A is a member of the label of a node which is the ancestor of a node for which B is a member of the label, and A and B are* unrelated *if A and B are not siblings and A is not an ancestor of B and B is not an ancestor of A.*

For example, in the scheme tree shown in Figure 3, `Name` and `Id` are siblings, `Name` is an ancestor of `Exam` and `Major` and `Project` are unrelated.

Also, we need the following result from [9]. Let us denote the NRS of nested relation r^* by $N(r^*)$.

Lemma 1. *For any nested relation r^* and any $Y \subseteq N(r^*)$, $\mu_{\{Y\}}(\nu_Y(r^*)) = r^*$.*

We note the well known result [9] that the converse of the above lemma does not hold, i.e. there are nested relations such that $\nu_Y(\mu_{\{Y\}}(r^*)) \neq r^*$.

Lemma 2. *Let r and r^* be as defined in the procedure given in Section 5.1.2 and let $A \in U$ and $B \in U$. Also, let t^*_A be a subtuple in r^* in which A is an atomic attribute and let t^*_B be a subtuple in r^* in which B is an atomic attribute. If $t^*_A[A] = a$ and $t^*_B[B] = b$ then there exists a tuple t in r such that $t[A,B] =< a,b >$ if any of the following conditions are true:*
(i) A and B are siblings in $N(r^*)$ and $t^*_A = t^*_B$;
(ii) A is an ancestor of B in $N(r^*)$ and t^*_B is a subtuple of t^*_A;
(iii) B is an ancestor A in $N(r^*)$ and t^*_A is a subtuple of t^*_B;
(iv) A and B are unrelated in $N(r^*)$.

Proof. Suppose that (i) is satisfied. We shall show by induction that there exists a tuple $t \in \mu^*(r^*)$ such that $t[A,B] =< a,b >$ from which it follows that $t \in r$ by Lemma 1. Since the ordering of unnesting is immaterial we unnnest r^* by $\mu_{\{Y_0\}} \cdots \mu_{\{Y_{n-1}\}}(r^*)$. Let Y_i be the NRS in the unnesting in which A and B are atomic attributes. Initially, we have a subtuple t^*_A in r^* for which $t^*_A[A,B] =< a,b >$. Assume inductively then that $\mu_{\{Y_j\}} \cdots \mu_{\{Y_{n-1}\}}(r^*), i+1 < j$ contains the subtuple t^*_A. It follows from the definition of unnest that t^*_A is will still be a subtuple $\mu_{\{Y_{j-1}\}} \cdots \mu_{\{Y_{n-1}\}}(r^*)$ and so by induction t^*_A is a subtuple in $\mu_{\{Y_{i-1}\}} \cdots \mu_{\{Y_{n-1}\}}(r^*)$. From the definition of unnest, it follows that $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{n-1}\}}(r^*)$ will contain a tuple t such that $t[A,B] =< a,b >$ and the property will then still hold, by a similar inductive argument and the definition of unnest, for $\mu_{\{Y_j\}} \cdots \mu_{\{Y_{n-1}\}}(r^*), j < i$ and so the property is proven.

Consider (ii). Let Y_i denote the NRS in the construction of r^* in which A appears as an atomic attribute and let Y_j denote the NRS in the construction of r^* in which B appears as an atomic attribute. Since A is an ancestor of B the unnesting on Y_i will be performed before Y_j in the total unnest. We firstly note that since t^*_A is a subtuple in r^* then it follows by a simple inductive argument similar to the one just given and definition of unnest that t^*_A will be a subtuple in $\mu_{\{Y_{i+1}\}} \cdots \mu_{\{Y_{n-1}\}}(r^*)$ and t^*_A has the subtuple t^*_B. It then follows by definition of unnest that there will be a tuple t in $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{n-1}\}}(r^*)$ such that $t[A] =< a >$ and t has the subtuple t^*_B. Then again by induction and the definition of unnest there will be a tuple t_1 in $\mu_{\{Y_j\}} \cdots \mu_{\{Y_{n-1}\}}(r^*)$ such that $t_1[A, B] =< a, b >$ and again by induction and the definition the same property will hold for $\mu_{\{Y_k\}} \cdots \mu_{\{Y_{n-1}\}}(r^*), k < j$ and so the result is proven.

The result (iii) follows , by symmetry using the same argument as in (ii).

Consider (iv). Let Y_i denote the NRS in the construction of r^* in which A appears as an atomic attribute and let Y_j denote the NRS in the construction of r^* in which B appears as an atomic attribute. Suppose that the unnesting on Y_i is performed first. Then usiong the same arguments as for the previous cases it follows that there exists a tuple t in $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{n-1}\}}(r^*)$ such that $t[A] =< a >$. The using the same arguments as before it follows that there will exist a tuple t_1 in $\mu_{\{Y_j\}} \cdots \mu_{\{Y_i\}} \cdots \mu_{\{Y_{n-1}\}}(r^*)$ such that $t[A, B] =< a, b >$ and the same property will then also hold for $\mu_{\{Y_k\}} \cdots \mu_{\{Y_{n-1}\}}(r^*), k < j$ and so the result is proven. □

Lemma 3. *If t is a tuple in r such that $t[A, B] =< a, b >$ and t is the only tuple in r such that $t[A, B] =< a, b >$ and A and B are siblings in $N(r^*)$, then there exists only one subtuple t^*_1 in $r*$, defined over a NRS N_1, such that A and B are atomic attributes in N_1 and $t^*_1[A, B] =< a, b >$.*

Proof. We shall prove the result by induction on the nesting operations. Let Y_i be the NRS in which A and B appear as atomic attributes. Initially the result is true for r and suppose inductively that it is true for r_j, where $j < i - 1$. Then by property (i) of the nest operator the result will be true after we nest r_j on Y_j and so the property is true for r_{j+1}. Consider then $r_i = \nu_{Y_{i-1}}(r_{i-1})$. By property (ii) of the nest operator, if there exists a tuple t with $t[A, B] =< a, b >$ before nesting on Y_i then after the nesting there will be a subtuple t_1 defined over Y_i such that $t_1[A, B] =< a, b >$. It then follows by a similar inductive argument and property (ii) of the nest operator that each relation r_j, $j > i$ will contain the subtuple t_1 and so the result is proven.

To prove the second part, suppose to the contrary that there are two subtuples t^*_2 and t^*_3 such that t^*_2 and t^*_3 are defined over N_1 and A and B are siblings. Then because the nest operator does not result in duplicate tuples, then there must exist another atomic attribute C such that either C is a sibling of A and B and $t^*_2[C] = c_1 \neq t^*_3[C] = c_2$, or there exists atomic attribute C such that A and B are ancestors of C and there exists a subtuple of t^*_2, call it t^*_4, and a subtuple of t^*_3, call it t^*_5 such that $t^*_4[C] = c_1 \neq t^*_5[C] = c_2$. Then using a similar argument to the one used in lemma 2, this implies that there is a tuple $t \in r$ such that $t[A, B, C] =< a, b, c_1 >$ and a tuple $t' \in r$ such that $t'[A, B, C] =< a, b, c_2 >$ and so t and t' must be distinct which is a contradiction and so the second part of the lemma is established. □

Lemma 4. *If t is a tuple in r such that $t[A, B] =< a, b >$ and t is the only tuple in r such that $t[A, B] =< a, b >$ and A is an ancestor of B in $N(r^*)$ then there exists a subtuple t_1^* (defined over NRS N'') and one (and only one) subtuple t_2^* (defined over NRS N'''), such that A is an atomic attribute in N'' and $t_1^*[A] =< a >$ and t_2^* is a subtuple of t_1^* and B is an atomic attribute in N''' and $t_2^*[B] =< b >$.*

Proof. We shall prove the result by induction on the nesting operations. Let Y_i be the NRS in which A appears as the atomic attribute and let Y_j be the NRS in which B appears as the atomic attribute. Since A is an ancestor of B we have that $i > j$. Initially r contains a tuple t with $t[A, B] =< a, b >$. The same argument as used in the previous lemma then shows that r_k, where $k < j$, will contain a tuple t_k such that $t_j[A, B] =< a, b >$. Consider then the effect of nesting on Y_j. By definition of the nest operator, after the nesting it will contain a tuple t_j and t'_j, a subtuple of t_j, such that $t_j[A] =< a >$ and $t'_j[B] =< b >$. It then follows by a similar inductive argument and properties of the nest operator that each relation r_k, $i > k > j$ will have the same property. Consider then the effect of nesting on Y_i. By property (ii) of the nest operator, after the nest on Y_i . there will exist a subtuple t_i in r_{i+1} and t'_i, a subtuple of t_i, such that $t_i[A] =< a >$ and $t'_i[B] =< b >$. Using a similar inductive argument one can show that the same property remains true for all r_j, $j > i$ and so the result is proven.

To prove the second part, suppose to the contrary that there are two subtuples t_2^* and t_3^* of t_1^* such that t_2^* and t_3^* are defined over N''' and$t_2^*[B] = t_3^*[B]$. Then because the nest operator does not result in duplicate tuples, then there must exist another atomic attribute C such that either C is a sibling of B and $t_2^*[C] = c_1 \neq t_3^*[C] = c_2$, or there exists atomic attribute C such that B is an ancestor of C and there exists a subtuple of t_2^*, call it t_4^*, and a subtuple of t_3^*, call it t_5^* such that $t_4^*[C] = c_1 \neq t_5^*[C] = c_2$. Then using a similar argument to the one used in lemma 2, this implies that there is a tuple $t \in r$ such that $t[A, B, C] =< a, b, c_1 >$ and a tuple $t' \in r$ such that $t'[A, B, C] =< a, b, c_2 >$ and so t and t' must be distinct which is a contradiction and so the second part of the lemma is established.

□

Lemma 5. *If t is a tuple in r such that $t[A, B] =< a, b >$ and t is the only tuple in r such that $t[A, B] =< a, b >$ and A and B are unrelated in $N(r^*)$ then there exists a tuple t_1^* in r^* and there exists subtuples t_2^* (defined over NRS N'') and t_3^* (defined over NRS N''') of t_1^*, such that A is an atomic attribute in N'' and $t_2^*[A] =< a >$ and B is an atomic attribute in N''' and $t_3^*[B] =< b >$ and neither t_2^* nor t_3^* are subtuples of each other. Also, if there exist subtuples t_4^* (defined over NRS N'') and t_5^* (defined over NRS N''') of t_1^*, such that $t_4^*[A] =< a >$ and $t_5^*[B] =< b >$ then $t_4^* = t_2^*$ and $t_5^* = t_3^*$.*

Proof. We shall prove the result by induction on the nesting operations. Let Y_i be the NRS in which A appears as the atomic attribute and let Y_j be the

NRS in which B appears as the atomic attribute. Since A and B are unrelated it does not matter which nest is performed first. We shall choose arbitrarily Y_j to be nested first.. Initially r contains a tuple t with $t[A, B] =< a, b >$. The same argument as used in the previous lemma then shows that r_k, where $k < j$, will contain a tuple t_k such that $t_j[A, B] =< a, b >$. Consider then the effect of nesting on Y_j. By definition of the nest operator, after the nesting it will contain a tuple t_i and t'_i, a subtuple of t_i, such that $t_i[A] =< a >$ and $t'_i[B] =< b >$. It then follows by a similar inductive argument and properties of the nest operator that each relation r_k, $i > k > j$ will have the same property. Consider then the effect of nesting on Y_{i-1}. By property (ii) of the nest operator, after the nest on Y_{i-1} there will exist a subtuples t_i and t'_i in r_i such that $t_i[A] =< a >$ and $t'_i[B] =< b >$ but t_i and t'_i are not subtuples of each other. Using a similar inductive argument one can show that the same property remains true for r_j, $j > i$ and so the result is proven.

The second part of the lemma can be established using similar arguments to those used in the previous lemmas. □

Proof of Theorem 3

If: We shall show the contrapositive that it T contains redundancy then R is not in BCNF. Let $\Sigma_R = \{A_{i_1} \rightarrow A_{j_1}, \ldots, A_{i_n} \rightarrow A_{j_n}\}$. Let r be any relation in $rel(R)$, ω any mapping in Ω and let $T = \omega(R)$. We firstly note that by Theorem 1, T satisfies the set of XFDs $\Sigma_\omega = \{p_{A_{i_1}} \rightarrow p_{A_{j_1}}, \ldots, p_{A_{i_n}} \rightarrow p_{A_{j_n}}\}$. So if T contains redundancy, then this means that there exists an XFD $p_{A_{i_k}} \rightarrow p_{A_{i_l}}$ in Σ_ω and two nodes v_1 and v_2 in $Paths(p_{A_j})$ such that $val(v_1) = val(v_2)$ and either: (a) two distinct nodes v_3 in $Nodes(x_1, p_{A_k})$ and v_4 in $Nodes(y_1, p_{A_l})$ such that $val(v_3) = val(v_4)$ or (b) $x_1 = y_1$.

We consider (a) first. We now consider several exhaustive subcases: (a.1) v_3 and v_4 are siblings; (a.2) $Parent(v_3)$ is an ancestor of v_4, (c) $Parent(v_4)$ is an ancestor of v_3, (d) $Parent(v_3)$ and $Parent(v_4)$ are unrelated.

Consider (a.1). By the construction procedure and since $x_1 \neq y_1$, this implies that there exist two subtuples t_1^* and t_2^* in r^* such that A and B are atomic attributes in both subtuples and $t_1[A_{i_k}, A_{i_l}] = t_2[A_{i_k}, A_{i_l}] =< a, b >$. It then follows from Lemma 2 (i) that there exists tuple t_3 and t_4 in r such that $t_3[A_{i_k}, A_{i_l}] =< a, b >$ and $t_4[A_{i_k}, A_{i_l}] =< a, b >$. Moreover, by Lemma 3, t_3 and t_4 are distinct. This implies that A_{i_k} cannot be a superkey since r satisfies Σ_R and t_3 and t_4 are identical on A_{i_k} and so BCNF is violated.

Consider (a.2). By the construction procedure for T, there exist subtuples t_1^* and t_2^* in r^*, where t_2 is a subtuple of t_1, such that A_{i_k} is an atomic attribute in t_1 and A_{i_l} is an atomic attribute in t_2^* and $t_1^*[A_{i_k}] =< a >$ and $t_2^*[A_{i_l}] =< b >$, where $a = val(v_3)$ and $b = val(v_1)$. Then since $x_1 \neq y_1$, by the construction procedure it follows that there exist subtuples t_3^* and t_4^* in r^*, where t_4^* is a subtuple of $_3^*$ and t_3^* is distinct from t_1^*, such that A_{i_k} is an atomic attribute in t_3^* and A_{i_l} is an atomic attribute in t_4^* and $t_1^*[A_{i_k}] =< a >$ and $t_2^*[A_{i_l}] =< b >$. So using Lemma 2 (iii), this implies that there exists two tuples t_5 and t_6 in r such that $t_5[A_{i_k}, A_{i_l}] =< a, b >$ and $t_6[A_{i_k}, A_{i_l}] =< a, b >$. It also follows from

Lemma 4 that t_5 and t_6 are distinct. It then follows, as for case (a.1), that R is not in BCNF.

By symmetry, case (a.3) is handled in the same fashion as case (a.2).

Consider case (a.4). By the construction procedure for T, if T contains redundancy then there exist subtuples t_1^* and t_2^* in r^* such that A_{i_k} is an atomic attribute in t_1^* and A_{i_l} is an atomic attribute in t_2^* and $t_1^*[A_{i_k}] =< a >$ and $t_2^*[A_{i_l}] =< b >$, where $a = val(v_3)$ and $b = val(v_1)$. Then since $x_1 \neq y_1$, by the construction procedure for T it follows that there exist subtuples t_3^* and t_4^* in r^*, where t_3^* is distinct from t_1^*, such that A_{i_k} is an atomic attribute in t_3^* and A_{i_l} is an atomic attribute in t_4^* and $t_1^*[A_{i_k}] =< a >$ and $t_2^*[A_{i_l}] =< b >$. So using Lemma 2 (iv), this implies that there exists two tuples t_5 and t_6 in r such that $t_5[A_{i_k}, A_{i_l}] =< a, b >$ and $t_6[A_{i_k}, A_{i_l}] =< a, b >$. It also follows from Lemma 5 that t_5 and t_6 are distinct. It then follows, as for case (a.1), that R is not in BCNF.

Consider then case (b). Let v_3 be any node in $Nodes(x_1, p_{A_k})$. Since $x_1 = y_1$ and because of the construction procedure for T, the only case that can arise is when $Parent(v_3)$ is an ancestor of v_1 and v_2. Then, by the construction procedure for T, there exist subtuples t_1^*, t_2^* and t_3^* in r^*, where t_2^* and t_3^* are subtuples of t_1^*, such that A_{i_k} is an atomic attribute in t_1^* and A_{i_l} is an atomic attribute in t_2^* and t_3^* and $t_1^*[A_{i_k}] =< a >$ and $t_2^*[A_{i_l}] = t_3^*[A_{i_l} =< b >$, where $a = val(v_3)$ and $b = val(v_1)$. So using Lemma 2 (iii), this implies that there exists two tuples t_5 and t_6 in r such that $t_5[A_{i_k}, A_{i_l}] =< a, b >$ and $t_6[A_{i_k}, A_{i_l}] =< a, b >$. It also follows from Lemma 4 that t_5 and t_6 are distinct. It then follows, as for case (a.1), that R is not in BCNF.

Only If: Suppose that R is not in BCNF and so there exists a FD $A_i \to A_j$ such that A_i is not a superkey. Then from a well known theorem in relation theory [11], there exists a relation consisting of two tuples which satisfies Σ_R and such that $t_1[A_i, A_j] = t_2[A_i, A_j]$. Construct then a mapping from r to T which contains no nesting at all, i.e. the mapping is as shown in Figure 11. Then T contains redundancy since any change to the value of either A_j nodes results in the XFD $root.Id.A_i \to root.Id.A_j$ being violated. □

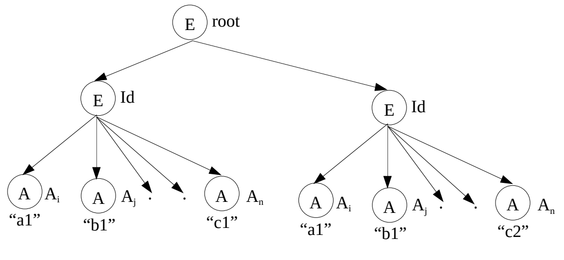

Fig. 11. XML trees from different mappings.