Mapping Normalized Relations to Normalized XML Documents

Millist W. Vincent  Jixue Liu  Chengfei Liu
School of Computer and Information Science
University of South Australia
email{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? In some allied work we formally defined functional dependencies in XML (XFDs) and, based on this definition, formally defined redundancy in an XML document. We then introduced a normal form for an XML document (XNF) and showed that it is a necessary and sufficient condition for the elimination of redundancy. In this paper we address the problem of determining what class of mappings map a relation in BCNF to an XML document in XNF. The class of mappings we consider is very general and allows arbitrary nesting of the original flat relation. Our main result establishes a necessary and sufficient condition on the DTD induced by the mapping for it to be in XNF.

1 Introduction

The eXtensible Markup Language (XML) [3] has recently emerged as a standard for data representation and interchange on the Internet [21, 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 are 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 mapping as good if it produces an XML document which is free of redundancy. This then raises the question of what is precisely meant by redundancy in an XML document. The relationship between normal forms and redundancy elimination has been investigated, both for the relational case [11, 7, 10] and the nested relational case [8], and in particular it has been shown that Boyce-Codd normal form (BCNF) [5] is a necessary and sufficient condition for the elimination of redundancy in relations when the only constraints are functional dependencies (FDs). In some recent work [13, 14, 20], we showed how to extend the definition of FDs in relations to FDs in XML (called XFDs) and how to extend the definition of redundancy from relations to XML. In particular we defined a normal form for a set of XFDs (XNF) and showed that it is a necessary and sufficient condition for every document satisfying the set of XFDs to be free of redundancy.

In a previous paper [19] we addressed 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 since, as shown in [19], they can lead to update problems which parallel those that occur in unnormalized flat relations. The class of mappings that we considered is a very general class of mappings from a relation into an XML document first proposed in [13, 14]. 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 showed 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.

However, there are certain limitations to this previous work that we address in this paper. In particular, in [19] we were only concerned with eliminating redundancy from those XML documents which are the result of mappings from normalised relations, rather than all possible XML documents defined over the schema. As a result, it is possible that an XML document mapped from a relation to be redundancy free even though the set of XFDs applying to the document is not in XNF. While this is fine if the document is to remain unchanged, it is not satisfactory if the XML document is to be subsequently updated because the fact that the document is not in XNF means that redundancy can arise even if the XFDs in the document are checked after an update. To make this point clearer, consider the following example.

Example 1 Consider the relation scheme $R = \{Name, Address\}$ and suppose that the set of FDs that apply to $R$ is the set $\Sigma = \{Name \rightarrow Address\}$, i.e. $Name$ is a key for $R$. Consider then the relation $r$ defined over $R$ shown in Figure 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>Mars</td>
</tr>
<tr>
<td>Anna</td>
<td>Venus</td>
</tr>
</tbody>
</table>

Figure 1. A flat relation.

Suppose we then map the relation $r$ into an XML document in two different ways. The first way nests on $Address$ first and is shown in Figure 2. The second nests on $Name$ first and is shown in Figure 3 (we note that it is also possible to map without nesting).

```
<root>
  <Id Name="Bill">
    <Id Address="Mars"/>
  </Id>
  <Id Name="Anna">
    <Id Address="Venus"/>
  </Id>
</root>
```

Figure 2. First XML document derived from Figure 1

From the results in [19], it follows that both the XML documents are redundancy free, as is also intuitively clear. However, the XML document in Figure 2 is not normalised. As will be shown later, the XFDs satisfied by the XML document in Figure 2 is $\Sigma_1 = \{root.Id,Name \rightarrow root.Id,Id,Address; root.Id,Name,root.Id,Id,Address \rightarrow root.Id\}$. $\Sigma_1$ does not satisfy the definition of XNF given [13, 14] because $\Sigma_1$ does not imply the XFD $root.Id,Name \rightarrow root.Id,Id$. As a result, the XML document in Figure 2 will not remain redundancy free if updates are made to the document. For instance, if we add a new $Id$ element containing an $Address$ attribute to the document then the resulting document, shown in Figure 4, satisfies $\Sigma_1$ but either of the $Address$ attributes is now redundant. In contrast the XML document shown in Figure 3 is normalized. The set of XFDs satisfied by the document is $\Sigma_2 = \{root.Id,Id,Name \rightarrow root.Id,Address; root.Id,Id,Name,root.Id,Address \rightarrow root.Id\}$ and this satisfies XNF because $root.Id,Id,Name \rightarrow root.Id$ is implied by $\Sigma_2$. As a result, ensuring that $\Sigma_2$ is satisfied after an update is sufficient to ensure the absence of redundancy.

As a result of this discussion, in this paper we address the following problem. Given a relation scheme which is in BCNF, and a set of unary keys, what mappings from a relation defined over a scheme to an XML document result in the set of XFDs induced by the mapping being in XNF? If a mapping has this property, then because of the result on the relationship between XNF and redundancy elimination...
mentioned earlier, one is guaranteed that any XML document will be free of redundancy, even if later updated, as long as the XFDs are checked after an update. The main result of the paper (Theorem 3) derives a necessary and sufficient condition on the DTD induced by the mapping for it to be in XNF. Not surprising, because we are placing more stringent requirements on the mapping from relations to XML documents, Theorem 3 shows that only a very limited class of mappings produce normalized documents whereas the result in [19] showed that any mapping will result in a redundancy free document as long as the relation scheme is in BCNF.

The rest of this paper is organized as follows. Section 2 contains some preliminary definitions. Section 3 contains the definition of XFDs. Section 4 addresses the topic of how to map flat relations to XML documents and establishes one of the main results of the paper concerning the XFDs induced by the mappings. Section 5 addresses the problem of how to determine when a mapping from a relation to an XML document results in the document being in XNF. In Section 5 we derive the main result of the paper which provides a characterization in terms of the induced DTD. Finally, Section 6 contains some concluding remarks.

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 $\mathcal{S}$ indicating text. An XML tree is defined to be $T = (V, lab, ele, att, val, v_0)$ where $V$ is a finite set of nodes in $T$; lab is a function from $V$ to $E \cup \mathcal{A} \cup \{\mathcal{S}\}$; ele is a partial function from $V$ to a sequence of nodes such that for any $v \in V$, if $ele(v)$ is defined then $lab(v) \in E$; att is a partial function from $V \times A$ to $V$ such that for any $v \in V$ and $l \in A$, if $att(v, l) = v_1$ then $lab(v_1) \in E$ and $lab(v_1) \neq l$; val is a function such that for any node in $v \in V$, $val(v) = v$ if $lab(v) \in E$ and $val(v)$ is a string if either $lab(v) = \mathcal{S}$ or $lab(v) \in \mathcal{A}$; $v_0$ is a distinguished node in $V$ called the root of $T$ and we define $lab(v_0) = root$. Since node identifiers are unique, a consequence of the definition of $val$ is that if $v_1 \in E$ and $v_2 \in E$ 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 $V \subseteq V$, then $val(V)$ is the set defined by $val(V) = \{val(v) | v \in V\}$.

For any $v \in V$, if $ele(v)$ is defined then the nodes in $ele(v)$ are called subelements of $v$. For any $l \in A$, 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 $\text{Ancestor}(v)$. The children of a node $v$ are also defined as in Definition 1 and we denote the parent of a node $v$ by $\text{Parent}(v)$.

We note that our definition of $val$ differs slightly from that in [4] 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 $l_1, \ldots, l_n$, $n \geq 1$, where $l_i \in E \cup A \cup \{\mathcal{S}\}$ for all $i$, $1 \leq i \leq n$ and $l_1 = root$. If $p$ is the path $l_1, \ldots, l_n$ then $Last(p) = l_n$.

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

Definition 3 Let $p$ denote the path $l_1, \ldots, l_n$. The function $\text{Parent}(p)$ is the path $l_1, \ldots, l_{n-1}$. Let $p$ denote the path $l_1, \ldots, l_n$ and let $q$ denote the path $q_1, \ldots, q_m$. 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 $E = \{\text{root}, \text{Division}, \text{Employee}\}$ and $A = \{D\#, \text{Emp}\#\}$ then $\text{root}.\text{Division}.\text{Employee}.\text{D}\#, \text{Division}.\text{Employee}.\text{Emp}\#.S$ is a root.

Definition 4 A path instance in an XML tree $T$ is a sequence $v_1, \ldots, v_n$ such that $v_1 = v_0$ and for all $v_i, 1 \leq i < n, v_i \in V$ and $v_i$ is a child of $v_{i-1}$. A path instance $v_1, \ldots, v_n$ is said to be defined over the path $l_1, \ldots, l_n$ if for all $v_i, 1 \leq i \leq n, lab(v_i) = l_i$. Two path instances $v_1, \ldots, v_n$ and $v_1', \ldots, v_n'$ are said to be distinct if $v_i \neq v_i'$ for some $i, 1 \leq i \leq n$. The path instance $v_1, \ldots, v_n$ is said to be a prefix of $v_1', \ldots, v_n'$ if $n \leq m$ and $v_i = v_i'$ for all $i, 1 \leq i \leq n$. The path instance $v_1, \ldots, v_n$ is said to be a strict prefix of $v_1', \ldots, v_n'$ if $n < m$ and $v_i = v_i'$ for all $i, 1 \leq i \leq n$. The set of path instances over a path $p$ in a tree $T$ is denoted by $\text{Paths}(p)$.
For example, in Figure 5, \(v_r, v_1, v_3\) is a path instance defined over the path \(\text{root}, \text{Division}, \text{Section}\) 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 \(p_1 \subseteq p\) 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 that is of missing information. This is addressed in [13] 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 \(p_1 \subseteq p_2\) and there exists a path instance \(v_1, \ldots, v_n\) defined over \(p_1\), in \(T\), then there exists a path instance \(v_1', \ldots, v_m'\) defined over \(p_2\) in \(T\) such that \(v_1, \ldots, v_n\) is a prefix of the instance \(v_1', \ldots, v_m'\).

For example, if we take \(P\) to be \(\{\text{root, root, Dept, root, Dept, Section, root, Dept, Section, Emp, root, Dept, Section, Project}\}\) then the tree in Figure 5 conforms to \(P\) and is complete.

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

**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 \(N(p) = \{v | v_1, \ldots, v_n \in \text{Paths}(p) \land v = v_n\}\).

For example, in Figure 5, \(N(\text{root, Dept}) = \{v_1, v_2\}\). 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 \(\text{AAcestor}(v)\), where \(v \in V\), is the set of nodes in \(T\) defined by \(\text{AAcestor}(v) = v \cup \text{Ancestor}(v)\).

For example, in Figure 5, \(\text{AAcestor}(v_3) = \{v_r, v_1, v_3\}\). 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 \(\text{Nodes}(v, p)\), where \(v \in V\) and \(p \in P\), is the set of nodes in \(T\) defined by \(\text{Nodes}(v, p) = \{x | x \in N(p) \land v \in \text{Ancestor}(x)\}\).

For example, in Figure 5, \(\text{Nodes}(v_1, \text{root, Dept, Section, Emp}) = \{v_4, v_5\}\). 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 \(v_2 \in \text{Ancestor}(v_1)\).

### 3 Strong Functional Dependencies in XML

We recall the definition of the XFD from [13, 14].

**Definition 12** Let \(P\) be a set of consistent paths and let \(T\) be an XML tree that conforms to \(P\). An XML functional dependency (XFD) is a statement of the form: \(p_1, \ldots, p_k \rightarrow q,\ k \geq 1\), where \(p_1, \ldots, p_k\) and \(q\) are paths in \(P\). \(T\) strongly satisfies the XFD if \(p_k = q\) for some \(i, 1 \leq i \leq k\) or for any two distinct path instances \(v_1, \ldots, v_n\) and \(v_1', \ldots, v_n'\) in \(\text{Paths}(q)\) in \(M(T)\), \(\text{val}(v_n) \neq \text{val}(v_n')\) ⇒ \(\exists i, 1 \leq i \leq k\), such that \(x_i \neq y_i\) if \(\text{Last}(p_i)\) is an element of \(E\), else \(\exists \{x_i, y_i\} \in \text{Nodes}(x_1, p_1)\) and \(\exists \{x_i, y_i\} \in \text{Nodes}(y_1, p_1)\) and \(\text{val}(\text{Nodes}(x_1, p_1)) \cap \text{val}(\text{Nodes}(y_1, p_1)) = \emptyset\), where \(x_i = \max \{v | v \in \{v_1, \ldots, v_n\} \land v \in N(p_i \cap q)\}\) and \(y_i = \max \{v | v \in \{v_1', \ldots, v_n'\} \land v \in N(p_i \cap q)\}\).
We note that since the path \( p_i \cap q \) is a prefix of \( q \), there exists only one node in \( v_1, \ldots, v_n \) 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 an example.

**Example 2** Consider the XML tree shown in Figure 6 and the XFD

\[
root, Department, Lecturer, Lname \rightarrow root, Department, Lecturer, Subject, SubjName, S.
\]

Then \( v_r, v_1, v_2, v_3, v_4, v_5 \) and \( v_r, v_2, v_3, v_4, v_5 \) are two distinct path instances in \( Paths(root, Department) \) and \( val(v_2) = "n1" \) and \( val(v_4) = "n2" \). So

\[
N(root, Department, Lecturer, Lname \cap root, Department, Lecturer, Subject, SubjName, S) = \{v_5, v_1, v_2\}
\]

and so \( x_1 = v_5 \) and \( y_1 = v_3 \). Thus

\[
val(Nodes(x_1), root, Department, Lecturer, Lname)) = \{"n1"\}
\]

and

\[
val(Nodes(y_1), root, Department, Lecturer, Lname)) = \{"n1"\}
\]

and so the XFD is violated. We note that if we change \( val \) of node \( v_1 \) in Figure 6 to "h1" then the XFD is satisfied.

Consider next the XFD \( root, Department, Hhead \rightarrow root, Department \). Then \( v_r, v_1 \) and \( v_r, v_2 \) are two distinct path instances in \( Paths(root, Department) \) and \( val(v_1) = v_1 \) and \( val(v_2) = v_2 \).

\[
N(root, Department, Hhead \cap root, Department) = \{v_1, v_2\}
\]

and so \( x_1 = v_1 \) and \( y_1 = v_2 \). Thus

\[
val(Nodes(x_1), root, Department, Hhead)) = \{"h1"\}
\]

and

\[
val(Nodes(y_1), root, Department, Hhead)) = \{"h2"\}
\]

and so the XFD is satisfied. We note that if we change \( val \) of node \( v_1 \) in Figure 6 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 taken from [12].

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 \( n_1 \in Ancestor(n) \).

**Figure 7. A scheme tree**

Figure 7 illustrates an example scheme tree defined over the set of attributes \{\( Name, Sid, Major, Class, Exam, Proj \)\}.

**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 \( S_1, \ldots, S_k, k \geq 1 \), are the principal subtrees of \( S \) then \( N(S) = A \cup \{N(S_1), \ldots, N(S_k)\} \).

For example, for the scheme tree \( S \) shown in Figure 7, \( N(S) = \{Name, Sid, \{Major\}, \{Class, \{Exam\}, \{Proj\}\}\} \). 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 \( ATT(n) = \{A_1, \ldots, A_n\} \) then \( DOM(N(S)) = DOM(A_1) \times \cdots \times DOM(A_n) \);

(ii) If \( A = ATT(ROOT(S)) \) and \( S_1, \ldots, S_k \) are the principal subtrees of \( S \), then \( DOM(N(S)) = DOM(A) \times \cdots \times DOM(N(S_1)) \times \cdots \times DOM(N(S_k)) \).
\[ P(DOM(N(S_1))) \times \cdots \times P(DOM(N(S_k))) \] 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 7, \( Z(N(S)) = \{ \text{Name}, \text{Sid} \} \) and \( H(N(S)) = \{ \{ \text{Major} \}, \{ \text{Class}, \{ \text{Exam} \} \}, \{ \text{Proj} \} \} \).

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 \( r^*[Y] = \{ t[Y] \mid t \in r \} \). An example of a nested relation over the scheme tree of Figure 7 is shown in Figure 8.

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 8 the tuples \( < \text{CS100}, \{ \text{Mid}, \text{Final} \}, \{ \text{ProjA}, \text{ProjB}, \text{ProjC} \} > \) and \( < \text{Anna}, \text{Sid1}, \{ \text{Maths}, \text{Physics} \}, \{ \text{CS100}, \{ \text{Mid}, \text{Final} \}, \{ \text{ProjA}, \text{ProjB}, \text{ProjC} \} > \).

\[ \begin{array}{|c|c|c|c|c|c|} \hline \text{Name} & \text{Sid} & \{ \text{Major} \} & \{ \text{Class} \} & \{ \text{Exam} \} & \{ \text{Proj} \} \tabularnewline \hline \text{Anna} & \text{Sid1} & \text{Maths} & \text{CS100} & \text{Mid} & \text{ProjA} \tabularnewline & & \text{Physics} & & \text{ProjB} & \\
\text{Bill} & \text{Sid2} & \text{Physics} & \text{P100} & \text{Final} & \text{Prac1} \tabularnewline & & & & \text{Prac2} & \\
\text{Chem} & \text{CH200} & \text{Test A} & & \text{Prac A} \tabularnewline & & \text{Test B} & & \text{Prac B} \tabularnewline \hline \end{array} \]

Figure 8. A nested relation.

We assume that the reader is familiar with the definition of the nest operator, \( \nu_r(r^*) \), and the unnest operator, \( \mu_Y(r^*) \), for nested relations as defined in [9, 2].

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 \( r_i = \nu_{r_{i-1}}(r_{i-1}), r_0 = r, r^* = r_n, 1 \leq i \leq n \) 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 sub-tuple 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 3 Consider the flat relation shown in Figure 9.

<table>
<thead>
<tr>
<th>Name</th>
<th>Sid</th>
<th>Major</th>
<th>Class</th>
<th>Exam</th>
<th>Proj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Sid1</td>
<td>Maths</td>
<td>CS100</td>
<td>Mid</td>
<td>Proj A</td>
</tr>
<tr>
<td>Anna</td>
<td>Sid1</td>
<td>Maths</td>
<td>CS100</td>
<td>Mid</td>
<td>Proj B</td>
</tr>
<tr>
<td>Anna</td>
<td>Sid1</td>
<td>Maths</td>
<td>CS100</td>
<td>Final</td>
<td>Proj A</td>
</tr>
<tr>
<td>Anna</td>
<td>Sid1</td>
<td>Maths</td>
<td>CS100</td>
<td>Final</td>
<td>Proj B</td>
</tr>
</tbody>
</table>

Figure 9. A flat relation.

If we then transform the relation \( r \) in Figure 9 by the sequence of nestings \( r_1 = \nu_{Proj}(r), r_2 = \nu_{Exam}(r_1), r_3 = \nu_{CLASS, Exam, Proj}(r_2), r^* = \nu_{Major}(r_3) \) then the relation \( r^* \) is shown in Figure 10. We then transform the nested relation in Figure 10 to the XML tree shown in Figure 11.

![Figure 10. A nested relation derived from a flat relation.](image)

![Figure 11. An XML tree derived from a nested relation.](image)
We now have the major result 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 $r^*$. We refer to the XFDs satisfied in $T_r^*$ as the induced XFDs.

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

**Proof.** See Appendix.

In addition the mapping induces a DTD for $T_r^*$, what we refer to as the induced DTD. To describe this we firstly formally define a restricted class of DTDs which is sufficient for the purpose of this paper.

**Definition 15** A DTD is a specification of the form $D ::= e[(e(D_1, \ldots, D_n))*]$ where $e \in E$ and $D_1, \ldots, D_n$ are DTDs and $*$ represents repetition one or more times.

For example, in the previous example the induced DTD is:

$$D ::= (Id(Name,Sid,(Id(Major))*),(Id(Class,(Id(Exam))*(Id(Proj))*))*).$$

**5 Mapping from normalized relations to Normalized XML Documents**

In this section we consider the question of how to characterize those mappings from relations to XML documents which result in the XML document being free of redundancy. Firstly, we discuss the issue of how to define redundancy in XML and then outline the relationship between redundancy elimination and the normal form for XML documents (XNF) given in [13, 14].

Defining redundancy, even in the relational model, is not quite so straightforward as might first appear. The most obvious approach in the relational model is, given a relation $r$ and an FD $A \rightarrow B$, to find tuples $t_1$ and $t_2$ to define a value $t_1[B]$ to be redundant if $t_1[B] = t_2[B]$ and $t_1[A] = t_2[A]$. While this definition is fine for FDs in relations, it doesn’t generalize in an obvious way to other classes of relational integrity constraints, such as multi-valued dependencies (MVDs) or join dependencies (JDS) or inclusion dependencies (INDs), nor to other data models. The key to finding the appropriate generalization is based on the observation that if a value $t_1[B]$ is redundant in the sense just defined then every change of $t_1[B]$ to a new value results in the violation of $A \rightarrow B$. One can then define a data value to be redundant if every change of it to a new value results in the violation of the set of constraints (whatever they may be). This is essentially the definition proposed in [11] where it was shown that BCNF is a necessary and sufficient condition for there to be no redundancy in the case of FD constraints and fourth normal form (4NF) is a necessary and sufficient condition for there to be no redundancy in the case of FD and MVD constraints. Interestingly though, it was shown in [10] that when the set of constraints contains JDS, the necessary and sufficient condition for there to be no redundancy is a new normal form that is different from the standard definition of 5NF [6].

We now give our definition of redundancy which is an extension of the definition given in [11]. Firstly, let us denote by $P\Sigma$ the set of paths that appear on the l.h.s. or r.h.s. of any XFD in $\Sigma$.

**Definition 16** 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($v$) $\in$ E.

**Definition 17** Let $P$ be a consistent set of paths and let $\Sigma$ be a set of XFDs such that $P\Sigma \subseteq P$. Then $\Sigma$ is said to cause redundancy if there exists an XML tree $T$ such that $T$ conforms to $P$ and satisfies $\Sigma$ and a node $v$ in $T$ such that every valid change from $v$ to $v'$, resulting in a new XML tree $T'$, causes $\Sigma$ to be violated.

We now illustrate this definition by an example.

**Figure 12. XML tree illustrating redundancy.**

**Example 4** 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 $\Sigma$ of XFDs {root, Project, Id $\rightarrow$ root, Project, Name} and the XML tree $T$ shown in Figure 12. Then $\Sigma$ causes redundancy because $T$ is consistent with $P$ and satisfies $\Sigma$ yet every valid change to either of the Name nodes results in root, Project, Id $\rightarrow$ root, Project, Name being violated.
We now define the notion of a trivial XFD and then define the normal form for XML taken from [13, 20].

**Definition 18** An XFD \( p_1, \ldots, p_k \rightarrow q \) is trivial if any of the conditions hold:

(i) \( q = p_i \) for some \( i, 1 \leq i \leq k \);
(ii) \( \text{Last}(p_i) \in \mathcal{E} \) and \( q \) is a prefix of \( p_i \) for some \( i, 1 \leq i \leq k \);
(iii) \( \text{Last}(q) \in \mathcal{A} \) and \( p_i = \text{Partd}(q) \) for some \( i, 1 \leq i \leq k \);
(iv) \( q = \text{root} \).

**Definition 19** Let \( P \) be a consistent set of paths and let \( \Sigma \) be a set of XFDs such that \( P_\Sigma \subseteq P \). \( \Sigma \) of XFDs is in XML normal form (XNF) if for every non-trivial XFD \( p_1, \ldots, p_k \rightarrow q \in \Sigma^+ \), \( \text{Last}(q) \notin \Sigma \) and if \( \text{Last}(q) \in \mathcal{A} \) then \( p_1, \ldots, p_k \rightarrow \text{Parnd}(q) \in \Sigma^+ \), where \( \Sigma^+ \) denotes the set of XFDs logically implied by \( \Sigma \).

We now illustrate the definition by an example.

**Example 5** Consider the XML tree \( T \) shown in Figure 12 and the sets of XFDs \( \Sigma_1 = \{ \text{root,Employee} \rightarrow \text{root,Employee,Phone,S}\} \), \( \Sigma_2 = \{ \text{root,Project,Id} \rightarrow \text{root,Project,Name}\} \) and \( \Sigma_3 = \{ \text{root,Project} \rightarrow \text{root,root,Employee,Phone,S} \rightarrow \text{root,Employee,Phone,root,Employee,Phone,S} \rightarrow \text{root,Employee,Emp#,root,Employee,Emp#} \rightarrow \text{root,Employee,root,Employee,Phone,S} \rightarrow \text{root,Employee,Emp#,root,Employee,Emp#}\} \). Then \( \Sigma_1 \) is not in XNF because the last label in the path \( \text{root,Employee,Phone,S} \) is a text node and \( \Sigma_2 \) is not in XNF since \( \text{root,Project,Id} \rightarrow \text{root,Project,Name} \) does not imply \( \text{root,Project,Id} \rightarrow \text{root,Project} \). \( \Sigma_3 \) is in XNF because the first XFD is trivial, the last label in the path on the r.h.s. of the second XFD is an element, as is the last label in the path on the r.h.s. of the third XFD and for the last XFD, it follows from applying inference axiom A3 (see the set of inference rules in [13]) to \( \text{root,Employee,Emp#} \rightarrow \text{root,Employee,root,Employee,Phone,S} \rightarrow \text{root,Employee,Emp#,root,Employee,Emp#} \) that \( \text{root,Employee,Phone,S} \rightarrow \text{root,Employee,Emp#} \in \Sigma^+ \) and so \( \Sigma_3 \) is in XNF.

This leads to the following important result which was established in [13].

**Theorem 2** Let \( P \) be a consistent set of paths and let \( \Sigma \) be a set of XFDs such that \( P_\Sigma \subseteq P \). Then \( \Sigma \) does not cause redundancy iff \( \Sigma \) is in XNF.

Before proceeding to the main result of the paper, we firstly address the issue of what XFDs to apply to the XML document from a mapping as given in Section 4. As already shown in Theorem 1, the only XFDs induced by a mapping from a relation to an XML document are those of the form \( p_A \rightarrow p_B \) corresponding to an FD \( A \rightarrow B \). However, if these are the only XFDs that we assume apply to the XML document then no XML document will be in XNF. To see why, consider the XML document shown in Figure 3. Since the only FD that holds in the flat relation is \( \text{Name} \rightarrow \text{Address} \), the only XFD that holds in Figure 3 is \( \text{root,Id,Id,Name} \rightarrow \text{root,Id,Address} \). However if this is the only XFD then \( \text{root,Id,Id,Name} \rightarrow \text{root,Id,Address} \) does not imply \( \text{root,Id,Id,Name} \rightarrow \text{root,Id} \). To see this consider the XML document shown in Figure 13. What have done is we have simply duplicated the \( \text{Id} \) element in Figure 3. However, the XML document satisfies the XFD \( \text{root,Id,Id,Name} \rightarrow \text{root,Id,Address} \) but not \( \text{root,Id,Id,Name} \rightarrow \text{root,Id} \) since different \( \text{Id} \) tags are assigned different identifiers when mapped to an XML tree. This situation can be considered somewhat pathological. Since there is no concept of duplicate elimination in XML, XML allows for the duplication of elements. In the relational case, the fact that BCNF ensures the elimination of redundancy depends entirely on the fact that the relational model uses set semantics. If we changed the semantics of the relational model to use bag semantics instead of set semantics, then BCNF would no longer ensure the elimination of redundancy. So what we propose is to add an extra XFD to the set of induced XFDs to ensure that this pathological situation does not occur. The XFD that we add is the combination of every path terminating in an attribute node determines the \( \text{root,Id} \) node in the tree. This ensures that there are no duplicate subtrees rooted at the highest \( \text{Id} \) node. For instance, in Figure 3 we assume that the extra XFD \( \text{root,Id,Id,Name,root,Id,Address} \rightarrow \text{root,Id} \) applies to the XML document.

```
<root>
  <Id Address="Mars"/>
  <Id Name="Bill"/>
</Id>
  <Id Address="Venus"/>
  <Id Name="Anna"/>
</Id>
  <Id Address="Mars"/>
  <Id Name="Bill"/>
</Id>
  <Id Address="Venus"/>
  <Id Name="Anna"/>
</Id>
</root>
```

**Figure 13. An XML document with duplicate trees**

In general, we have the following definition.
The statement that we allow from relations to XML documents is XNF. Let \( R(A_1, \ldots, A_n) \) denote a relation scheme which is in BCNF, let \( \Sigma_R \) denote a set of unary FDs defined over \( R \). Let \( \Omega \) be the set of all mappings from relations defined over \( R \) to XML trees as defined in Section 4 and let \( \omega \in \Omega \). Let \( \Sigma_\omega \) be the set of XFDs as defined by Theorem 1. Then the set of XFDs induced by \( \omega \), denoted by \( \Sigma_{i_\omega} \), is defined by \( \Sigma_{i_\omega} = \Sigma_\omega \cup \{ p_{A_1}, \ldots, p_{A_n} \rightarrow \text{root}, \text{Id} \} \).

We now turn to the main topic of this paper, determining when \( \Sigma_{i_\omega} \) is in XNF. One approach would be to use the definition of XNF directly since the implication problem for unary XFDs has been shown to be decidable and a linear time algorithm has been developed for it [17]. In other words, we simply proceed by taking each XFD \( p \rightarrow q \in \Sigma_{i_\omega}^+ \) and using the implication algorithm in [17] to determine if \( p \rightarrow \text{Part}(q) \in \Sigma_{i_\omega}^+ \). However this approach neither provides any insight into the properties of those mappings which result in XML trees in XNF, nor is it easy to use. It requires that the system designer be familiar both with the theory of XFDs and the implication algorithm for them. Instead, what we do now is a much more efficient and simpler approach. Instead of using the induced XFDs directly, we shall characterize when \( \Sigma_{i_\omega} \) is in XNF by the structure of the induced DTD. This result is much easier to apply as it requires only the knowledge of the keys in the relation scheme and the structure of the DTD; it does not require any understanding of XFDs. Thus we now have the main result of this paper.

**Theorem 3** Let \( R(A_1, \ldots, A_n) \) denote a relation scheme which is in BCNF and let \( \Sigma_R \) denote a set of unary FDs defined over \( R \). Let \( D_\omega \) be the DTD induced by \( \omega \). Then \( \Sigma_{i_\omega} \) is in XNF if and only if the following hold:

(i) if \( R \) has more than one key then \( D_\omega \) has the form \( D_\omega := (\text{Id}(A_1, \ldots, A_n))^* \);  
(ii) if \( R \) has only one key, call it \( A_1 \), then \( D_\omega \) has the form \( D_\omega := (\text{Id}(A_1, \ldots, A_n))^* \) or \( D_\omega := (\text{Id}(A_2, \ldots, A_n, (\text{Id}(A_1))^*))^* \).

**Proof.** See Appendix.

For example, consider the example XML document in Figure 2 which has the DTD \( D_1 := (\text{Id}(\text{Name}(\text{Id}(\text{Address}))))^* \). Then \( D_1 \) violates the condition since \( \text{Address} \) is not a key. Consider then the example in Figure 3 which has the DTD \( D_2 := (\text{Id}(\text{Address}(\text{Id}(\text{Name})))^* \). Then \( D_2 \) satisfies the condition because there is only one key \( \text{Name} \).

**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 will the XML document be normalized, i.e. be in XNF as defined in [13].

The XML document being in XNF is important since, as shown in [13], it is a necessary and sufficient condition for the elimination of redundancy in an XML document. Being redundancy free is an important property of an XML document since, as shown in [19], it guarantees the absence of certain types of update anomalies in the same fashion that redundancy elimination and BCNF ensure the elimination of update anomalies in relational databases [11].

The mappings that we allow from relations to XML documents were first introduced in [13, 14] and are very general. Essentially, they allow arbitrary nesting of a flat relation into an XML document and we believe that the class considered includes all the mappings that are likely to occur in practice. Drawing on some previous work by the authors [14, 13, 20] that formally defined functional dependencies and redundancy in XML documents, we firstly showed that FDs in flat relation map to XFDs in XML documents in a natural fashion, i.e. if an FD holds in a flat relation and only if a related XFD holds in the mapped XML document. The main result of the paper (Theorem 3) then gave a necessary and sufficient condition on the DTD induced by the mapping for the XML document to be in XNF.

There are several other related issues that we intend to investigate in the future. The first is to relax the assumption that the exported relation is in BCNF. In general, the relation to be exported will be a materialized view of the source database and so, even if the relations in the source database are normalized, the materialized will not necessarily be normalized. We need then to derive a similar result to the one derived in this paper to determine when an unnormalized flat relation is mapped to an XML document. The second issue is to consider the same problem for the case where the constraints in the source relation are MVDs. In some allied work [16, 15, 18], we have shown how to extend the definition of multivalued dependencies to XML documents and have defined a 4NF for XML (4XNF) and shown it to be a sufficient condition for the elimination of redundancy. Thus an important problem to be addressed is determining which mappings from a relation to XML are in 4XNF. The last issue that we intend to address is the issue of mappings between XML documents, rather than between relations and XML documents as considered in this paper. In this case the class of mappings will be different and may be those that can be expressed in a language such as Xquery. The problem then to be addressed is which mappings produce normalized XML documents.
References


7 Appendix

(This section is to help the reviewer assess the paper and will not be in the final version of the paper)

Before proving Theorem 1 we need some preliminary definitions and lemmas.

Definition 21 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 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 7, Name and Sid are siblings, Name is an ancestor of Exam and Major and Project are unrelated.

Next, 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 $Y \subseteq N(r^*)$, 
$\mu_{\{Y\}}(t_{r^*}(r^*)) = r^*$.  

Lemma 2 Let $r$ and $r^*$ be as defined in the procedure given in Section 5.1.2, i.e. $r$ is any flat relation and $r^* = \nu_{\{Y_i\} \cdots \nu_{Y_j}(r)}$ and let $A \subseteq U$ and $B \subseteq U$. 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 $S(r^*)$ and $t^*_A = t^*_B$; 
(ii) $A$ is an ancestor of $B$ in $S(r^*)$ and $t^*_B$ is a subtuple of $t^*_A$; 
(iii) $B$ is an ancestor $A$ in $S(r^*)$ and $t^*_A$ is a subtuple of $t^*_B$; 
(iv) $A$ and $B$ are unrelated in $S(r^*)$.  

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 unnest $r^*$ by $\mu_{\{Y_j\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$. 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 $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$, $i + 1 < j$, contains the subtuple $t^*_A$. It follows from the definition of unnest that $t^*_A$ is still a subtuple in $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$ and so by induction $t^*_A$ is a subtuple in $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$. From the definition of unnest, it follows that $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$ 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 $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$, $j < i$ 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 $Y_j$ will be performed before the unnesting on $Y_j$ 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 in (i) and the definition of unnest that $t^*_A$ will be a subtuple in $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$ and $t^*_A$ has the subtuple $t^*_B$. It then follows by the definition of unnest that there will be a tuple $t^*$ in $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$ 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 $t^*_i$ in $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$ such that $t^*_i[A, B] = < a, b >$ and again by induction and the definition the same property will hold for $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$, $k < j$ 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 using the same arguments as for the previous cases it follows that there exists a tuple $t^*$ in $\mu_{\{Y_1\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$ such that $t^*[A] = < a >$. Using the same arguments as before it follows that there will exist a tuple $t^*_i$ in $\mu_{\{Y_1\}} \cdots \mu_{\{Y_j\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$ such that $t^*_i[A, B] = < a, b >$ and the same property will then also hold for $\mu_{\{Y_i\}} \cdots \mu_{\{Y_{j-1}\}}(r^*)$, $k < j$ and so the result is proven.  

The next results are essentially converses of the above result.  

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

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 $r_{j-1}$, where $j < i - 1$. Then by property (i) of the nest operator the result will be true after we nest $r_{j-1}$ on $Y_j$ and so the property is true for $r_{j+1}$. Consider then $r_i = \nu_{\{Y_{j-1}\}}(r_{j-1})$. By property (ii) of the nest operator, if there exists a tuple $t^*$ with $t^*[A, B] = < a, b >$ before nesting on $Y_j$ then after the nesting there will be a subtuple $t^*_i$ defined over $Y_j$ such that $t^*_i[A, B] = < a, b >$. It then follows by a similar inductive argument and property (ii) of the nest operator that each relation $r_{j-1} j > i$ will contain the subtuple $t^*_i$ and so the result is proven.  

Lemma 4 If $t$ is a tuple in $r$ such that $t[A, B] = < a, b >$ and $A$ is an ancestor of $B$ in $S(r^*)$, then there exist sub-tuples $t^*_j$ (defined over a NRS $N''$) and $t^*_j$ (defined over a NRS $N'''$), such that $A$ is an atomic attribute in $N''$ and $t^*_j[A] = < a >$ and $t^*_j$ is a subtuple of $t^*_j$ and $B$ is an atomic attribute in $N'''$ and $t^*_j[B] = < a, b >$.  

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 $t_{k+1}$, where $k < j$, will contain a tuple $t^*_k$ such that $t^*_k[A, B] = < a, b >$. 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 \( t''_j \), a subtuple of \( t'_i \), such that \( t''_i[A] = < a > \) and \( t''_j[B] = < b > \). It then follows by a similar inductive argument and properties of the nest operator that each relation \( r_z \), \( i > k > j \) will have the same property. Consider then the effect of nesting on \( Y_j \). By property (ii) of the nest operator, after the nesting on \( Y_j \) there will exist a subtuple \( t''_i[R_j] \) in \( r_j \) such that \( t''_i[A] = < a > \) and \( t''_j[B] = < b > \). 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.

\[ \square \]

**Lemma 5** If \( t \) is a tuple in \( r \) such that \( t[A, B] = < a, b > \) and \( A \) and \( B \) are unrelated in \( S(r^*) \) then there exists a tuple \( t''_i \) in \( r^* \) and there exist subtuples \( t''_3 \) (defined over \( N^{'''} \)) and \( t''_2 \) (defined over \( N^{''''} \)) of \( t''_i \), such that \( A \) is an atomic attribute in \( N^{'''} \) and \( t''_3[A] = < a > \) and \( B \) is an atomic attribute in \( N^{''''} \) and \( t''_3[B] = < b > \) and neither \( t''_2 \) nor \( t''_3 \) are subtuple of each other.

**Proof.** We shall prove the result by induction on the nesting operations. Let \( Y_j \) 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 \( t_k[A, B] = < a, b > \). Consider then the effect of nesting on \( Y_j \). By definition of the nest operator, after the nesting \( r_{j+1} \) will contain a tuple \( t''_i \) and \( t''_j \), a subtuple of \( t''_i \), such that \( t''_i[A] = < a > \) and \( t''_j[B] = < b > \). It then follows by a similar inductive argument and the properties of the nest operator that each relation \( r_z \), \( i > k > j \) will have the same property. Consider then the effect of nesting on \( Y_{j+1} \). By property (ii) of the nest operator, after the nesting on \( Y_j \) there will exist a subtuple \( t''_i \) and \( t''_j \) in \( r_j \) such that \( t''_i[A] = < a > \) and \( t''_j[B] = < b > \) but \( t''_i \) and \( t''_j \) are not subtuple 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.

\[ \square \]

We also need the following result (Lemma 6.1 in [2]) which gives a syntactic characterization of strong satisfaction in flat relations.

**Lemma 6** An FD \( A \to B \) is strongly satisfied in a relation \( r \) if for every \( t_1, t_2 \in r \), either both \( t_1[A] \) and \( t_2[A] \) are non null and not equal or \( t_1[B] \) and \( t_2[B] \) are non null and equal.

We now prove the main result of Section 5.1.2

**Proof of Theorem 1**

If: We shall show the contrapositive that if \( r \) violates \( A \to B \) then \( T_{r^*} \) violates \( p_A \to p_B \). We shall consider several cases.

- **Case A:** \( \exists t_1, t_2 \in r \) such that \( t_1[A] \neq \bot \) and \( t_2[A] \neq \bot \) and \( t_2[A] = t_2[B] = < a > \) and \( t_1[B] = b_1 \neq b_2 = t_2[B] \).

Suppose firstly that \( A \) and \( B \) are siblings in \( S(r^*) \). By Lemma 3 there exist subtuples \( t''_2 \) and \( t''_3 \) in \( r^* \), defined over a NRS \( N_1 \), such that \( A \) and \( B \) are atomic attributes in \( N_1 \) and \( t''_2[A] = < a, b_1 > \) and \( t''_3[A, B] = < a, b_2 > \). It then follows by the construction procedure that the node in \( M(T_{r^*}) \) corresponding to \( t''_2[A] \), call it \( v_1 \), and the node corresponding to \( t''_3[A, B] \), call it \( v_2 \), are both children of the same node (call it \( v_{p2} \)). It then follows from the definition of \( x_1 \) in Definition 12 that \( x_1 = v_p \), and so \( v_1 \in Node(r, p_A) \) and \( v_2 \in Node(r, p_B) \). Thus since \( val(v_1) = val(v_2) = a \) and \( val(v'_1) = b_1 \neq b_2 = val(v'_2) \) it follows from the definition of an XFD that \( T_{r^*} \) violates \( p_A \to p_B \).

Suppose next that \( A \) is an ancestor of \( B \) in \( S(r^*) \). It follows from Lemma 4 that there exist subtuples \( t''_2 \) (defined over \( N^{''''} \) and \( t''_3 \) (defined over \( N^{''''} \)), such that \( A \) is an atomic attribute in \( N^{'''} \) and \( t''_2[A] = < a > \) and \( t''_3[A] = < a > \) and \( t''_2[B] \) is a subtuple of \( t''_2 \) and \( B \) is an atomic attribute in \( N^{''''} \). Using a similar inductive argument as used in the previous lemma, then shows that \( v_1 \) is an ancestor of \( v_2 \). It thus follows from the definition of \( y_1 \) in Definition 12 that \( y_1 = v_p \) and so \( v_1 \in Node(r, p_A) \). It then follows by the construction procedure that the node in \( M(T_{r^*}) \) corresponding to \( t''_2[A] \), call it \( v_2 \), and the node corresponding to \( t''_2[B] \), call it \( v'_2 \), are such that the parent of \( v_2 \), (call it \( v_{p2} \), is an ancestor of \( v'_2 \). It thus follows from the definition of \( x_1 \) in Definition 12 that \( x_1 = v_p \) and so \( v_1 \in Node(r, p_A) \). Thus since \( val(v_1) = val(v'_2) = a \) and \( val(v'_1) = b_1 \neq b_2 = val(v'_2) \) it follows from the definition of an XFD that \( T_{r^*} \) violates \( p_A \to p_B \). The case where \( B \) is an ancestor of \( A \) is handled similarly.

Lastly consider the case where \( A \) and \( B \) are not related in \( S(r^*) \). By Lemma 5 there exist subtuples \( t''_2 \) (defined over \( N^{''''} \) and \( t''_3 \) (defined over \( N^{''''} \)), such that
A is an atomic attribute in \( N'' \) and \( t_2^*[A] = < a > \) and \( t_2^* \) is a subtree of \( t_2^* \) and \( B \) is an atomic attribute in \( N''' \) \( t_2^*[B] = < b > \) and neither \( t_2^* \) nor \( t_2^* \) are subtuples of each other. Again, by Lemma 5 there exist subtuples \( t_2^* \) (defined over NRS \( N''' \)) and \( t_2^* \) (defined over NRS \( N''' \)), such that \( A \) is an atomic attribute in \( N'' \) and \( t_2^*[A] = < a > \) and \( t_2^* \) is a subtree of \( t_2^* \) and \( B \) is an atomic attribute in \( N''' \) \( t_2^*[B] = < b > \) and neither \( t_2^* \) nor \( t_2^* \) are subtuples of each other. It then follows by the construction procedure that the node in \( M(T_{r^*}) \) corresponding to \( t_1[A] \), call it \( v_1 \), and the node corresponding to \( t_1[B] \), call it \( v_1' \), have a maximal common ancestor (call it \( v_{p_1} \)). Moreover, because of the construction of \( M(T_{r^*}) \) and the definition of of \( x_1 \) it follows that \( x_1 = v_{p_1} \) and so \( v_1 \in Nodes(x_1, p_A) \). Similarly, it follows by the construction procedure that the node in \( M(T_{r^*}) \) corresponding to \( t_2[A] \), call it \( v_2 \), and the node corresponding to \( t_2[B] \), call it \( v_2' \), have a maximal common ancestor (call it \( v_{p_2} \)). Moreover, because of the construction of \( M(T_{r^*}) \) and the definition of of \( y_1 \) it follows that \( y_1 = v_{p_2} \) and so \( v_2 \in Nodes(x_1, p_A) \). Thus since \( val(v_i) = val(v_{p_1}) \) and \( val(v_i') = val(v_{p_2}) \) it follows from the definition of an XFD that \( T_{r^*} \) violates \( p_A \rightarrow p_B \).

**Case B:** \( \exists t_1, t_2 \in r \) such that \( t_1[A] \neq \perp \) and \( t_2[A] \neq \perp \) and \( t_1[A] = t_2[A] = < a > \) and \( t_2[B] = \perp \).

Suppose firstly that \( A \) and \( B \) are siblings in \( S(r^*) \). Using the same argument as in Case A, the node in \( M(T_{r^*}) \) corresponding to \( t_1[A] \), call it \( v_1 \), and the node corresponding to \( t_1[B] \), call it \( v_1' \), are both children of the same node (call it \( v_{p_1} \)) and so \( x_1 = v_{p_1} \) and \( v_1 \in Nodes(x_1, p_A) \). Using the same argument in Case A, the node in \( M(T_{r^*}) \) corresponding to \( t_2[A] \), call it \( v_2 \), and the node corresponding to \( t_2[B] \), the null node \( v_2' \), have the same parent \( v_{p_2} \). It thus follows that \( y_1 = v_{p_2} \) and so \( v_2 \in Nodes(y_1, p_A) \). So \( T_{r^*} \) violates \( p_A \rightarrow p_B \) since \( val(v_1') \neq val(v_2) \) (since \( v_2' \) is a null node) and \( val(v_1) = val(v_2) \).

Suppose next that \( A \) is an ancestor of \( B \) in \( S(r^*) \). Using the same argument as in Case B, it follows that the node in \( M(T_{r^*}) \) corresponding to \( t_1[A] \), call it \( v_1 \), and the node corresponding to \( t_1[B] \), call it \( v_1' \), are such that the parent of \( v_1 \), (call it \( v_{p_1} \)), is an ancestor of \( v_1' \) and so \( x_1 = v_{p_1} \) and thus \( v_1 \in Nodes(x_1, p_A) \). Following Case A, the node in \( M(T_{r^*}) \) corresponding to \( t_2[A] \), call it \( v_2 \), and the null node \( v_2' \) (corresponding to \( t_2[B] \)), are such that the parent of \( v_2 \), (call it \( v_{p_2} \)), is an ancestor of \( v_2' \). Thus \( y_1 = v_{p_2} \) and so \( v_2 \in Nodes(y_1, p_A) \) and thus \( T_{r^*} \) violates \( p_A \rightarrow p_B \) for the same reasons as before. The case where \( B \) is an ancestor of \( A \) is handled similarly.

Lastly consider the case where \( A \) and \( B \) are not related in \( S(r^*) \). As for previous case.

The other cases are handled similarly to the ones just given.

**Only If:** We shall show the contrapositive that if \( T_{r^*} \) violates \( p_A \rightarrow p_B \) then \( r \) violates \( A \rightarrow B \). Let \( v_1, \ldots, v_n \)
and \(v'_1, \ldots, v'_n\) be two distinct path instances in \(\text{Paths}(p_B)\) in \(M(T_r)\). We firstly note that because of the construction for \(T_r\) and axiom A7 we can assume that \(p_A \neq p_B\) and \(\text{Last}(p_A) \in A \) and \(\text{Last}(p_B) \in A\). There are several cases to consider.

Case A: \(\text{Parnt}(p_A)\) is a prefix of \(p_B\).

Since \(p_A \rightarrow p_B\) is violated we must have that \(\text{val}(v'_n) \neq \text{val}(v_n)\) and \(\exists \subseteq \text{Nodes}(x_1, p)\) or \(\exists \subseteq \text{Nodes}(y_1, p)\) or \(\text{val}(\text{Nodes}(x_1, p)) \cap \text{val}(\text{Nodes}(y_1, p)) \neq \emptyset\) where \(x_1\) and \(y_1\) are defined as in Definition 12. We consider each of these cases.

Case A.1: \(\text{val}(\text{Nodes}(x_1, p)) \cap \text{val}(\text{Nodes}(y_1, p)) \neq \emptyset\)

Let \(v_1\) be a node in \(\text{Nodes}(x_1, p)\) and let \(v_2\) be any node in \(\text{Nodes}(y_1, p)\) such that \(\text{val}(v_1) = \text{val}(v_2)\) and \(v_1\) and \(v_2\) may be the same node). Then because \(\text{Parnt}(p_A)\) is a prefix of \(p_B\) and the definition of \(x_1\) and \(y_1\) in Definition 12 it follows that \(x_1 = \text{Parent}(v_1)\) and \(y_1 = \text{Parent}(v_2)\). Hence by the construction procedure for \(T_r\) and the fact that \(A\) is an ancestor of \(B\) in \(S(r^*)\) (because \(\text{Parnt}(p_A)\) is a prefix of \(p_B\)), there exists a subtree \(t_1^*\) in \(r^*\) such that \(t_1^*[A] = \text{val}(v_1)\) and there exists a subtree \(t_2^*\) of \(t_1^*\) such that \(t_2^*[B] = \text{val}(v_n)\). So by Lemma 2 there exists a tuple \(t \in r\) such that \(t[A, B] = \text{val}(v_1), \text{val}(v_n)\). Using the same argument for \(v_1\) and \(v_n\) shows that there exists another tuple \(t' \in r\) such that \(t'[A, B] = \text{val}(v_2), \text{val}(v'_n)\) and by so by Lemma 6 \(A \rightarrow B\) is violated in \(r\) since \(\text{val}(v'_n) \neq \text{val}(v_n)\) and \(\text{val}(v_1) = \text{val}(v_2)\).

Case A.2: \(\exists \subseteq \text{Nodes}(y_1, p)\)

Let \(v_1\) the node in \(\text{Nodes}(x_1, p)\) that is null and let \(v_2\) be any node in \(\text{Nodes}(y_1, p)\). Following the argument used in the previous case, if \(\text{Parnt}(p_B)\) is violated then there exists a subtree \(t_1^*\) in \(r^*\) such that \(t_1^*[A] = \bot\) and there exists a subtree \(t_2\) of \(t_1^*\) such that \(t_2^*[B] = \text{val}(v_n)\). So by Lemma 2 there exists a tuple \(t \in r\) such that \(t[A, B] = \text{val}(v_1), \text{val}(v_n)\). Using the same argument for \(v_2\) and \(v_n\) shows that there exists another tuple \(t' \in r\) such that \(t'[A, B] = \text{val}(v_2), \text{val}(v'_n)\) and by so by Lemma 6 \(A \rightarrow B\) is violated in \(r\) since \(\text{val}(v'_n) \neq \text{val}(v_n)\).

Case A.3: \(\exists \subseteq \text{Nodes}(y_1, p)\)

As for the previous case.

Case B: \(\text{Parnt}(p_B)\) is a prefix of \(p_A\).

As for Case A.

Case C: \(\text{Parnt}(p_B)\) is not a prefix of \(p_B\).

Case C.1: \(\text{val}(\text{Nodes}(x_1, p)) \cap \text{val}(\text{Nodes}(y_1, p)) \neq \emptyset\)

Let \(v_1\) be the node in \(\text{Nodes}(x_1, p)\) and let \(v_2\) the node in \(\text{Nodes}(y_1, p)\) such that \(\text{val}(v_1) = \text{val}(v_2)\) and \(v_1\) and \(v_2\) may be the same node). Hence by the construction of \(T_r\) and Lemma 2 there exists a tuple \(t \in r\) such that \(t[A, B] = \text{val}(v_1), \text{val}(v'_n)\) and by so by Lemma 6 \(A \rightarrow B\) is violated in \(r\) since \(\text{val}(v'_n) \neq \text{val}(v_n)\).

Case C.2: \(\exists \subseteq \text{Nodes}(x_1, p)\)

Let \(v_1\) be a node in \(\text{Nodes}(x_1, p)\) that is null and let \(v_2\) be any node in \(\text{Nodes}(y_1, p)\). Hence there exists a subtree \(t_1^*\) in \(r^*\) such that \(t_1^*[A] = \bot\) and there exists a subtree \(t_2^*\) of \(t_1^*\) such that \(t_2^*[B] = \text{val}(v_n)\). So by Lemma 2 there exists a tuple \(t \in r\) such that \(t[A, B] = \text{val}(v_1), \text{val}(v_n)\). Using the same argument for \(v_2\) and \(v_n\) shows that there exists another tuple \(t' \in r\) such that \(t'[A, B] = \text{val}(v_2), \text{val}(v'_n)\) and by so by Lemma 6 \(A \rightarrow B\) is violated in \(r\) since \(\text{val}(v'_n) \neq \text{val}(v_n)\).

Case C.3: \(\exists \subseteq \text{Nodes}(y_1, p)\)

As for previous case.

\(\Box\)

**Proof of Theorem 3.**

*If* Suppose that (i) is true. Suppose that the keys are \(A_{i_1}, \ldots, A_{i_n}\). By definition \(\Sigma_{i_\omega}\) is the set \{root, \text{Id}, A_{i_1}, \ldots, root, \text{Id}, A_{i_n}\}. Hence by the transitivity rule A3 (see axioms in [13]) for every key \(A_{i_k}\), the XFD root, \text{Id}, A_{i_k} \rightarrow root, \text{Id} holds. So, because \(D_w\) has the form \(D_w := (\text{Id}(A_{i_1}, \ldots, A_{i_n}))*\), then \(\text{Parnt}(\text{root}, A_{i_k}) = \cdots = \text{Parnt}(\text{root}, A_{i_n}) = \text{root, Id}\) and so \(\Sigma_{i_\omega}\) is in XNF.

Suppose then that (ii) holds. If \(D_w\) has the form \(D_w := (\text{Id}(A_{i_1}, \ldots, A_{i_n}))*\) then the same argument just given applies and so assume that \(D_w := (\text{Id}(A_{i_1}, \ldots, A_{i_n}))*\). Then \(\Sigma_{i_\omega}\) is the set \{root, \text{Id}, A_{i_1} \rightarrow root, \text{Id}, A_{i_1}, \ldots, root, \text{Id}, A_{i_n} \rightarrow root, \text{Id}, A_{i_n}\}. It follows by the transitivity rule A3 that the XFD root, \text{Id}, A_{i_1} \rightarrow root, \text{Id} holds. So, because \(D_w\) has the form \(D_w := (\text{Id}(A_{i_1}, \ldots, A_{i_n}))*\), then \(\text{Parnt}(\text{root}, A_{i_1}) = \cdots = \text{Parnt}(\text{root}, A_{i_n}) = \text{root, Id}\) and so \(\Sigma_{i_\omega}\) is in XNF.

Only *If* Suppose firstly that there is more than one key. Then we shall show the contrapositive that if \(D_w\) does not have the form \(D_w := (\text{Id}(A_{i_1}, \ldots, A_{i_n}))*\) then \(\Sigma_{i_\omega}\) is not in XNF. If \(D_w\) does not have the form \(D_w := (\text{Id}(A_{i_1}, \ldots, A_{i_n}))*\), then there exists at least one attribute that does not appear in a node under \(\text{Id}\). Let the attribute be \(A_{i_1}\). Then since there are at least two keys, there must exists another key, call it \(A_{i_2}\). Construct a tree \(T^*\) such that for every path there is exactly one path instance. Because of this property it is immediate that \(T^*\) satisfies \(\Sigma_{i_\omega}\). Modify the tree by adding a new ancestor node of \(A_{i_1}\) and children such the \(\text{val}\) for ant two paths ending in an attribute are the same. Let the resulting tree be \(T^*\). We now consider several cases.

(a) root, \text{Id}, A_{i_2} is a prefix of root, \text{Id}, A_{i_1}.
The situation for $T^*$ is illustrated in Figure 14. We firstly claim that $T^*$ satisfies $\Sigma_{i_2}$. This follows immediately because of the fact that by construction of $T^*$, every path instance of a path ending in an attribute has the same $val$ for its final node. Now consider the XFD $\text{root},Id,A_2 \rightarrow \text{root},Id,Id$. We firstly note that because node identifiers are unique, we have that in Figure 14 $val(v_2) \neq val(v_3)$. Consider then $x_1$ and $y_1$ as defined Definition 12. Then because $\text{root},Id,A_2$ is a prefix of $\text{root},Id,Id,A_1$, it follows that $x_1 = y_1$ and so by definition of an XFD $\text{root},Id,A_2 \rightarrow \text{root},Id,Id$ is violated in $T^*$. However, since $T^*$ satisfies $\Sigma_{i_2}$, then $\text{root},Id,A_2 \rightarrow \text{root},Id,Id$ is not in XNF because $\text{root},Id,Id = \text{Parent}(\text{root},Id,Id,A_1)$.

(b) $\text{Parent}(p_{A_2})$ is prefix of $p_{A_2}$, where $p_{A_1}$ is the path that ends in $A_1$ and $p_{A_2}$ is the path ending in $A_2$.

The situation for $T^*$ is illustrated in Figure 15. For the same reasons as in the previous case, $T^*$ satisfies $\Sigma_{i_2}$. Now consider the XFD $p_{A_2} \rightarrow \text{Parent}(p_{A_1})$. We firstly note that the path instances of $\text{Parent}(p_{A_1})$ end in nodes which have different $val$ because $val(v_2) \neq val(v_3)$. Next because of the definition of $x_1$ and $y_1$ in Definition 12 and the fact that $\text{Parent}(p_{A_1})$ is prefix of $p_{A_2}$, it follows that $x_1 = v_2$ and $y_1 = v_3$. However, it then follows that $v_4 \in \text{Nodes}(x_1,p_{A_2})$ and $v_5 \in \text{Nodes}(y_1,p_{A_2})$ and so, since $val(v_4) = val(v_5)$ from the construction of $T^*$, it follows that $val(\text{Nodes}(x_1,p_{A_2})) \cap val(\text{Nodes}(y_1,p_{A_2})) \neq \emptyset$. Thus $p_{A_2} \rightarrow \text{Parent}(p_{A_1})$ is violated in $T^*$ and so $\Sigma_{i_2}$ is not in XNF.

(c) $\text{Parent}(p_{A_1})$ is not prefix of $p_{A_2}$ and $\text{Parent}(p_{A_2})$ is not prefix of $p_{A_2}$, where $p_{A_1}$ is the path that ends in $A_1$ and $p_{A_2}$ is the path ending in $A_2$.

Same argument as in (b).

Consider next the case where $R$ has only one key, $A_1$. We shall show the contrapositive that if (ii) is violated then $\Sigma_{i_2}$ is not in XNF. If (ii) is violated then there must exist another attribute, call it $A_2$, such that $A_2$ is not a child of $Id$. As before construct a tree $T$ such that for every path there is exactly one path instance in $T$. Because of this property it is immediate that $T$ satisfies $\Sigma_{i_2}$. Modify the tree by adding a new ancestor node of $A_2$ and whatever children are necessary such the $val$ for any two paths ending in an attribute are the same. Let the resulting tree be $T^*$. We now consider several cases.

(d) $p_{A_1} \cap p_{A_2} = \text{root},Id$

The situation is illustrated in Figure 16. Now consider the XFD $p_{A_1} \rightarrow \text{Parent}(p_{A_2})$. We firstly note that the path instances of $\text{Parent}(p_{A_2})$ end in nodes which have different $val$ because $val(v_2) \neq val(v_3)$. Also, because $p_{A_1} \cap p_{A_2} = \text{root},Id$ and the definition of $x_1$ and $y_1$ in Definition 12 it follows that $x_1 = y_1$. It then follows automatically from the definition of an XFD that $p_{A_1} \rightarrow \text{Parent}(p_{A_2})$ is violated in $T^*$ and so $\Sigma_{i_2}$ is not in XNF.

(e) $\text{Parent}(p_{A_2})$ is a prefix of $p_{A_2}$.

The situation is illustrated in Figure 17. Now consider the XFD $p_{A_1} \rightarrow \text{Parent}(p_{A_2})$. We firstly note that the path instances of $\text{Parent}(p_{A_2})$ end in nodes which have different $val$ because $val(v_2) \neq val(v_3)$. Also, because $\text{Parent}(p_{A_2})$ is a prefix of $p_{A_2}$ and the definition of $x_1$ and $y_1$ in Definition 12 it follows that $x_1 = v_2$ and $y_1 = v_3$. However, it then follows that $v_4 \in \text{Nodes}(x_1,p_{A_2})$ and $v_5 \in \text{Nodes}(y_1,p_{A_2})$ and so, since $val(v_4) = val(v_5)$ from the construction of $T^*$, it follows that $val(\text{Nodes}(x_1,p_{A_2})) \cap val(\text{Nodes}(y_1,p_{A_2})) \neq \emptyset$. Thus $p_{A_1} \rightarrow \text{Parent}(p_{A_2})$ is violated in $T^*$ and so $\Sigma_{i_2}$ is not in XNF.
(f) $\text{Parent}(p_{A_2})$ is a prefix of $p_{A_2}$.

The situation is illustrated in Figure 18. Now consider the XFD $p_{A_3} \rightarrow \text{Parent}(p_{A_3})$. We firstly note that the path instances of $\text{Parent}(p_{A_3})$ end in nodes which have different $\text{val}$ because $\text{val}(v_4) \neq \text{val}(v_5)$. Also, because $\text{Parent}(p_{A_3})$ is a prefix of $p_{A_3}$ and the definition of $x_1$ and $y_1$ in Definition 12 it follows that $x_1 = y_1$ and thus $p_{A_3} \rightarrow \text{Parent}(p_{A_3})$ is violated in $T^*$ and so $\Sigma_{i_\omega}$ is not in XNF. This completes the proof.