Normalization Theory for XML[*]

Marcelo Arenas
Department of Computer Science
Pontificia Universidad Católica de Chile
marenas@ing.puc.cl

1 Introduction

Since the beginnings of the relational model, it was clear for the database community that the process of designing a database is a nontrivial and time-consuming task. Even for simple application domains, there are many possible ways of storing the data of interest.

During the 70s and 80s, a lot of effort was put into developing methodologies to aid in the process of deciding how to store data in a relational database. The most prominent approaches developed at that time – which today are an standard part of the relational technology– were the entity-relationship and the normalization approach. In the normalization approach, an already designed relational database is given as input, together with some semantic information provided by a user in the form of relationships between different parts of the database, called *data dependencies*. This semantic information is then used to check whether the design has some desirable properties, and if this is not the case, it is also used to convert the poor design into an equivalent well-designed database.

The normalization approach was proposed in the early 70s by Codd [9, 10]. In this approach, a *normal form*, defined as a syntactic condition on data dependencies, specifies a property that a well-designed database must satisfy. Normalization as a way of producing good relational database designs is a well-understood topic. In the 70s and 80s, normal forms such as 3NF [9], BCNF [10], 4NF [13], and PJ/NF [14] were introduced to deal with the design of relational databases having different types of data dependencies. These normal forms, together with normalization algorithms for converting a poorly designed database into a well-designed database, can be found today in every database textbook.

With the development of the Web, new data models have started to play a more prominent role. In particular, XML (eXtensible Markup Language) has emerged as the standard data model for storing and interchanging data on the Web. As more companies adopt XML as the primary data model for storing information, the problem of designing XML databases is becoming more relevant.

The concepts of database design and normal forms are central in relational database technology. In this paper, we show how these concepts can be extended to XML databases. The goal of this paper is to present principles for good XML data design. We believe this research is especially relevant nowadays, since a huge amount of data is being put on the Web. Once massive Web databases are created, it is very hard to change their organization; thus, there is a risk of having large amounts of widely accessible, but poorly organized legacy data.

Designing a relational database means choosing an appropriate relational schema for the data of interest. A relational schema consists of a set of relations, or tables, and a set of data dependencies over these relations. Designing an XML database is similar: An appropriate *XML schema* has to be chosen, which usually consists of a DTD (Document Type Definition) and a set of data dependencies. However, the structure of XML documents, which are trees as opposed to relations, and the rather expressive constraints imposed by DTDs make the design problem for XML databases quite challenging.

This paper is organized as follows. In Section 2, we show that XML documents may contain redundant information, which could be related to the hierarchical structure of these documents. In Section 3, we present the basic terminology used in this paper. In Section 4, we introduce a functional dependency language for XML, which is used in Section 5 to define XNF, a normal form for XML documents. In Section

[*] **Database Principles Column.** Column editor: Leonid Libkin, School of Informatics, University of Edinburgh, Edinburgh, EH8 9LE, UK. E-mail: libkin@inf.ed.ac.uk.

6, we study the complexity of verifying whether an XML document is in XNF. In Section 7, we present an information-theoretic approach that can be used to justify normal forms and, in particular, XNF. We conclude the paper with some final remarks in Section 8.

2 Motivation: Redundant Information in XML

How does one identify bad designs? We have looked at a large number of DTDs and found two kinds of commonly present design problems. One of them is reminiscent of the canonical example of bad relational design caused by non-key functional dependencies, while the other one is more closely related to the hierarchical structure of XML documents, as we illustrate in the example below.

Example 2.1 Consider the following DTD that describes a part of a database for storing data about conferences.

```
<!ELEMENT db (conf*)>
<!ELEMENT conf (issue+)>
  <!ATTLIST conf
            title CDATA #REQUIRED>
<!ELEMENT issue (inproceedings+)>
<!ELEMENT inproceedings EMPTY>
  <!ATTLIST inproceedings
            title CDATA #REQUIRED
            pages CDATA #REQUIRED
            year CDATA #REQUIRED>
```

Each conference has a title, and one or more issues (which correspond to years when the conference was held). Papers are stored in `inproceedings` elements; the year of publication is one of its attributes.

Such a document satisfies the following constraint: any two `inproceedings` children of the same `issue` must have the same value of `year`. This too is similar to relational functional dependencies, but now we refer to the values (the `year` attribute) as well as the structure (children of the same `issue`). Moreover, we only talk about `inproceedings` nodes that are children of the same `issue` element. Thus, this functional dependency can be considered relative to each `issue`.

The functional dependency here leads to redundancy: year is stored multiple times for a conference. The natural solution to the problem in this case is not to create a new element for storing the year, but rather restructure the document and make `year` an attribute of `issue`. That is, we change attribute lists as:

```
<!ATTLIST issue
          year CDATA #REQUIRED>
<!ATTLIST inproceedings
          title CDATA #REQUIRED
          pages CDATA #REQUIRED>
```

□

Our goal is to show how to detect anomalies of those kinds.

3 Notation

We shall use a somewhat simplified model of XML trees in order to keep the notation simple. We assume a countably infinite set of labels L, a countably infinite set of attributes A (we shall use the notation $@a_1, @a_2$, etc for attributes to distinguish them from labels), and a countably infinite set V of values of attributes. Furthermore, we do not consider `PCDATA` elements in XML trees since they can always be represented by attributes.

A DTD (Document Type Definition) D is a 4-tuple (L_0, P, R, r) where L_0 is a finite subset of L, P is a set of rules $\ell \to P_\ell$ for each $\ell \in L_0$, where P_ℓ is a regular expression over $L_0 - \{r\}$, R assigns to each $\ell \in L_0$ a finite subset of A (possibly empty; $R(\ell)$ is the set of attributes of ℓ), and $r \in L_0$ (the root).

Example 3.1 The DTD shown in Example 2.1 is represented as (L_0, P, R, r), where $r = db$, $L_0 = \{db, conf, issue, inproceedings\}$, $P = \{db \to conf^*, conf \to issue^+, issue \to inproceedings^+, inproceedings \to \epsilon\}$, $R(conf) = \{@title\}$, $R(inproceedings) = \{@title, @pages, @year\}$ and $R(db) = R(issue) = \emptyset$. □

An XML tree is a finite rooted directed tree $T = (N, E)$ where N is the set of nodes and E is the set of edges, together with the labeling function $\lambda : N \to L$ and partial attribute value functions $\rho_{@a} : N \to V$ for each $@a \in A$. We furthermore assume that for every node x in N, its children $x_1, \ldots, x_n$ are ordered and $\rho_{@a}(x)$ is defined for a finite set of attributes $@a$. We say that T conforms to DTD $D = (L_0, P, R, r)$, written as $T \models D$, if the root of T is labeled r, for every $x \in N$ with $\lambda(x) = \ell$, the word $\lambda(x_1) \cdots \lambda(x_n)$ that consists of the labels of its children belongs to the language denoted by P_ℓ, and for every $x \in N$ with $\lambda(x) = \ell$, we have that $@a \in R(\ell)$ if and only if the function $\rho_{@a}$ is defined on x (and thus provides the value of attribute $@a$).

4 Functional Dependencies for XML

To present a functional dependency language for XML we need to introduce some terminology. An *element path* q is a word in L^*, and an *attribute path* is a word of the form $q.@a$, where $q \in L^*$ and $@a \in A$. An element path q is consistent with a DTD D if there is a tree $T \models D$ that contains a node reachable by q (in particular, all such paths must have r as the first letter); if in addition the nodes reachable by q have attribute $@a$, then the attribute path $q.@a$ is consistent with D. The set of all paths (element or attribute) consistent with D is denoted by $paths(D)$. This set is finite for a non-recursive D and infinite if D is recursive.

An *XML functional dependency (XFD)* over DTD D [4] is an expression of the form $\{q_1, \ldots, q_n\} \to q$, where $n \geq 1$ and $q, q_1, \ldots, q_n \in paths(D)$. To define the notion of satisfaction for XFDs, we use a relational representation of XML trees from [4]. Given $T = (N, E)$ that conforms to D, a *tree tuple* in T is a mapping $t : paths(D) \to N \cup V \cup \{\bot\}$ such that if q is an element path whose last letter is ℓ and $t(q) \neq \bot$, then

- $t(q) \in N$ and its label, $\lambda(t(q))$, is ℓ;
- if q' is a prefix of q, then $t(q') \neq \bot$ and the node $t(q')$ lies on the path from the root to $t(q)$ in T;
- if $@a$ is defined for $t(q)$ and its value is $v \in V$, then $t(q.@a) = v$.

Intuitively, a tree tuple assigns nodes or attribute values or nulls ($\bot$) to paths in a consistent manner. A tree tuple is maximal if it cannot be extended to another one by changing some nulls to values from $N \cup V$. The set of maximal tree tuples is denoted by $tuples_D(T)$. Now we say that XFD $\varphi = \{q_1, \ldots, q_n\} \to q$ is true in T, denoted by $T \models \varphi$, if for any $t_1, t_2 \in tuples_D(T)$, whenever $t_1(q_i) = t_2(q_i) \neq \bot$ for all $i \leq n$, then $t_1(q) = t_2(q)$ holds.

Example 4.1 Among the XFDs over the DTD from Example 2.1 one can find the following:

db.conf.@title → db.conf,

db.conf.issue →
db.conf.issue.inproceedings.@year.

The first functional dependency specifies that two distinct conferences must have distinct titles. The second one specifies that any two *inproceedings* children of the same *issue* must have the same value of *@year*. □

Given a DTD D and a set $\Sigma \cup \{\varphi\}$ of XFDs over D, we say that (D, Σ) *implies* φ, written $(D, \Sigma) \vdash \varphi$, if for every tree T with $T \models D$ and $T \models \Sigma$, it is the case that $T \models \varphi$. The set of all XFDs implied by (D, Σ) is denoted by $(D, \Sigma)^+$. Furthermore, an XFD φ is *trivial* if $(D, \emptyset) \vdash \varphi$. In relational databases, the only trivial FDs are $X \to Y$, with $Y \subseteq X$. Here, DTD forces some more interesting trivial functional dependencies. For instance, for each element path p in D and p' prefix of p, $(D, \emptyset) \vdash p \to p'$, and $(D, \emptyset) \vdash p \to p.@a$. As a matter of fact, trivial functional dependencies in XML documents can be much more complicated than in the relational case, as we show in the following example.

Example 4.2 Assume that $r \to (a|b|c)$ is a rule in a DTD D, being r the type of the root. Then, for every path p in D, XFD $\{r.a, r.b\} \to p$ is trivial since for every XML tree T conforming to D and every tree tuple t in T, $t(r.a) = \bot$ or $t(r.b) = \bot$. □

We conclude this section by pointing out that other proposals for XFDs exist in the literature [16, 20, 28]. In particular, the language introduced in [28] is similar to the one presented in this paper.

5 XNF: An XML Normal Form

With the definitions of the previous section, we are ready to present a normal form for XML documents.

Definition 5.1 [4] *Given a DTD D and a set Σ of XFDs over D, (D, Σ)* is in XML normal form (XNF) *iff for every nontrivial XFD $X \to p.@a \in (D, \Sigma)^+$, it is the case that $X \to p$ is in $(D, \Sigma)^+$.*

The intuition is as follows. Suppose that $X \to p.@a$ is in $(D, \Sigma)^+$. If T is an XML tree conforming to D and satisfying Σ, then in T for every set of values of the elements in X, we can find only one value of $p.@a$. Thus, to avoid storing redundant information, for every set of values of X we should store the value of $p.@a$ only once; in other words, $X \to p$ must be implied by (D, Σ).

In this definition, we impose the condition that φ is a nontrivial XFD. Indeed, the trivial XFD $p.@a \to$

$p.@a$ is always in $(D, \Sigma)^+$, but often $p.@a \rightarrow p \notin (D, \Sigma)^+$, which does not necessarily represent a bad design.

To show how XNF distinguishes good XML design from bad design, we revisit our running example.

Example 5.2 The conference example 2.1 seen earlier may contain redundant information: year is stored multiple times for the same issue of a conference. This XML specification is *not* in XNF since

$$\texttt{db.conf.issue} \rightarrow \\ \texttt{db.conf.issue.inproceedings.@year} \quad (1)$$

is a nontrivial XFD in the specification but

$$\texttt{db.conf.issue} \rightarrow \texttt{db.conf.issue.inproceedings}$$

is not in $(D, \Sigma)^+$, as several papers are usually published in a conference. The solution we proposed in the introduction was to make year an attribute of issue. XFD (1) is not valid in the revised specification, which can be easily verified to be in XNF. Note that we do not replace (1) by

$$\texttt{db.conf.issue} \rightarrow \texttt{db.conf.issue.@year},$$

since it is a trivial XFD and thus is implied by the new DTD alone. □

5.1 BCNF and XNF

In this section, we show that XNF generalizes BCNF. Recall that a relation specification (G, FD) is in BCNF, where relation G has attributes $A_1, \ldots, A_n$ and FD is a set of functional dependencies over G, if for every nontrivial FD $X \rightarrow Y$ implied by FD, we have that X is a superkey, that is, $X \rightarrow A_i$ is implied by Σ, for each $i \in [1, n]$.

Relational databases can be easily mapped into XML documents. Given a relation $G(A_1, \ldots, A_n)$ and a set of FDs Σ over G, we translate the schema (G, FD) into an XML representation, that is, a DTD D_G and a set of XFDs Σ_{FD}. The DTD $D_G = (L_0, P, R, db)$ is defined as follows: $L_0 = \{db, G\}$, $A = \{@A_1, \ldots, @A_n\}$, $P = \{db \rightarrow G^*, G \rightarrow \epsilon\}$, $R(db) = \emptyset$ and $R(G) = \{@A_1, \ldots, @A_n\}$. Without loss of generality, assume that all FDs are of the form $X \rightarrow A$, where A is an attribute. Then Σ_{FD} over D_G is defined as follows.

- For each FD $A_{i_1} \cdots A_{i_m} \rightarrow A_i \in FD$, $\{db.G.@A_{i_1}, \ldots, db.G.@A_{i_m}\} \rightarrow db.G.@A_i$ is in Σ_{FD}.
- $\{db.G.@A_1, \ldots, db.G.@A_n\} \rightarrow db.G$ is in Σ_{FD}.

The latter is included to avoid duplicates.

Example 5.3 A schema $G(A, B, C)$ can be coded by the following DTD:

```
<!ELEMENT db (G*)>
<!ELEMENT G EMPTY>
  <!ATTLIST G
            A CDATA #REQUIRED
            B CDATA #REQUIRED
            C CDATA #REQUIRED>
```

In this schema, an FD $A \rightarrow B$ is translated into $db.G.@A \rightarrow db.G.@B$. □

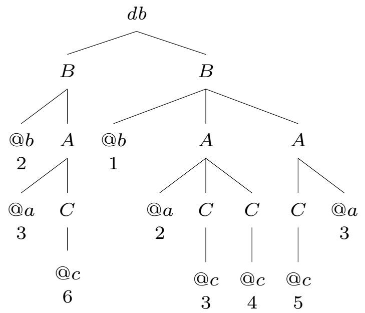
The following proposition shows that BCNF and XNF are equivalent when relational databases are appropriately coded as XML documents.

Proposition 5.4 [4] *Given a relation G and a set of functional dependencies FD over G, (G, FD) is in BCNF iff (D_G, Σ_{FD}) is in XNF.*

We conclude this section by showing that the hierarchical structure of XML documents can be used to overcome some of the limitations of relational normal forms. It is well known that every relational schema can be decomposed into an equivalent one in BCNF, but some constraints may be lost along the way. For instance, $(R(A, B, C), \{AB \rightarrow C, C \rightarrow B\})$ is a classical example of a schema that does not have any dependency preserving BCNF decomposition. By Proposition 5.4, one may be tempted to think that if we translate this schema into XML, the situation will be similar; every XNF decomposition of the translated schema should lose some constraints. The following example, taken from [18], shows that this is not the case, as we can use the hierarchical structure of XML to obtain a dependency preserving XNF decomposition.

Example 5.5 Let $D = (L_0, P, R, db)$ be a DTD, with $L_0 = \{db, A, B, C\}$, $P = \{db \rightarrow B^*, B \rightarrow A^*, A \rightarrow C^*, C \rightarrow \epsilon\}$, $R(db) = \emptyset$, $R(A) = \{@a\}$, $R(B) = \{@b\}$ and $R(C) = \{@c\}$. XML trees conforming to D are used to store instances of relation $R(A, B, C)$ as shown in the following figure:

A	B	C
2	1	3
2	1	4
3	1	5
3	2	6



Let Σ be the following set of XFDs over D:

$$\begin{array}{rcl} db.B.@b & \rightarrow & db.B, \\ \{db.B,\ db.B.A.@a\} & \rightarrow & db.B.A, \\ \{db.B.A,\ db.B.A.C.@c\} & \rightarrow & db.B.A.C, \\ \{db.B.A.@a,\ db.B.@b\} & \rightarrow & db.B.A.C.@c, \\ db.B.A.C.@c & \rightarrow & db.B.@b. \end{array}$$

The first three XFDs indicate how the instances of R are stored as XML trees conforming to D. Thus, for example, the first XFD indicates that all the tuples in R with the same value of attribute B are grouped together, in a tree that stores in its root the value of attribute B. The last two XFDs are the translations of relational FDs $AB \rightarrow C$ and $C \rightarrow B$, respectively.

XML specification (D, Σ) is equivalent to our original relational specification $(R(A, B, C),\ \{AB \rightarrow C,\ C \rightarrow B\})$. In fact, all the constraints in our original relational schema can be inferred from (D, Σ). Moreover, it is easy to prove that (D, Σ) is in XNF. □

The approach shown in the previous example was proposed in [18], under the name of hierarchical translation of relational schemas, as a way to obtain dependency preserving decompositions of relational schemas. The advantages and limitations of this approach are explored in [18].

6 The Complexity of Testing XNF

In the previous section, we introduce XNF, a normal form for XML specifications. As in the case of relational databases, testing whether an XML schema is in XNF involves testing some conditions on the functional dependencies implied by the specification. In this section, we present some results on the complexity of the implication problem for XFDs, and then we use these results to establish some complexity bounds for the problem of testing whether an XML specification is in XNF.

Throughout the section, we assume that the DTDs are non-recursive. This can be done without any loss of generality. Notice that in a recursive DTD D, the set of all paths is infinite. However, a given set of XFDs Σ only mentions a finite number of paths, which means that it suffices to restrict one's attention to a finite number of "unfoldings" of recursive rules.

6.1 The implication problem for XFDs

Although XML FDs and relational FDs are defined similarly, in this section we show that the implication problem for the former class is far more intricate.

Regular expressions used in DTDs are typically rather simple, which allows for efficient implication algorithms. We now formulate a criterion for simplicity that corresponds to a very common practice of writing regular expressions in DTDs [6, 8]. Given an alphabet E, a regular expression over E is called *trivial* if it is of the form $s_1, \ldots, s_n$, where for each s_i there is a letter $a_i \in E$ such that s_i is either a_i or $a_i?$ or a_i^+ or a_i^*, and for $i \neq j$, $a_i \neq a_j$. We call a regular expression s *simple* if there is a trivial regular expression s' such that any word w in the language denoted by s is a permutation of a word in the language denoted by s', and vice versa. Simple regular expressions were also considered in [1] under the name of *multiplicity atoms*.

For example, $(a|b|c)^*$ is simple: a^*, b^*, c^* is trivial, and every word in $(a|b|c)^*$ is a permutation of a word in a^*, b^*, c^* and vice versa. Simple regular expressions are prevalent in DTDs [6]. We say that a DTD D is *simple* if all productions in D use only simple regular expressions. It turns out that the implication problem for simple DTDs can be solved efficiently.

Theorem 6.1 [4] *The implication problem for XFDs over simple DTDs is solvable in quadratic time.*

In a simple DTD, disjunction can appear in expressions of the form $(a|\epsilon)$ or $(a|b)^*$, but a general disjunction $(a|b)$ is not allowed. For example, the following DTD cannot be represented as a simple DTD:

```
<!ELEMENT uni (student*)>
<!ELEMENT student ((name | FL), grade)>
<!ELEMENT name (EMPTY)>
  <!ATTLIST name
            value CDATA #REQUIRED>
<!ELEMENT FL (first, last)>
<!ELEMENT first (EMPTY)>
```

```
    <!ATTLIST first
              value CDATA #REQUIRED>
  <!ELEMENT last (EMPTY)>
    <!ATTLIST last
              value CDATA #REQUIRED>
```

In this example, every student must have a name. This name can be a string or it can be a composition of a first and a last name. It is desirable to express constraints on this kind of DTDs. For instance,

$$\{\texttt{uni.student.FL.first.@value}, \\ \texttt{uni.student.FL.last.@value}\} \rightarrow \texttt{uni.student},$$

is a functional dependency in this domain. It is also desirable to reason about these constraints. Unfortunately, allowing disjunction in DTDs, even in some restricted ways, makes the implication problem for XFDs harder.

A regular expression s over an alphabet E is a *simple disjunction* if $s = \epsilon$, $s = \ell$, where $\ell \in E$, or $s = s_1|s_2$, where s_1, s_2 are simple disjunctions over alphabets E_1, E_2 and $E_1 \cap E_2 = \emptyset$. We say that a DTD D is *disjunctive* if every production in D uses a regular expression of the form $s_1, \ldots, s_m$, where each s_i is either a simple regular expression or a simple disjunction over an alphabet E_i ($i \in [1, m]$), and $E_i \cap E_j = \emptyset$ ($i, j \in [1, m]$ and $i \neq j$). It turns out that the implication problem for disjunctive DTDs is coNP-complete.

Theorem 6.2 [4] *The implication problem for XFDs over disjunctive DTDs is coNP-complete.*

The previous result can be extended to a larger class of DTDs that was called *relational DTDs* in [4]. In some sense these results are positive, as we can expect that in practice we will be able to solve the implication problem for disjunctive and relational DTDs, since XML specifications tend to be relatively small in practice, as opposed to XML documents which can be very large, and today we can find SAT solvers like BerkMin [17] and Chaff [22] that routinely solve NP problems with thousands of variables.

We conclude this section by showing that the implication problem for XFDs is decidable. The exact complexity of this problem is unknown.

Theorem 6.3 [2] *The implication problem for XFDs over DTDs is solvable in co-NEXPTIME.*

6.2 Testing XNF

Testing whether an XML specification (D, Σ) is in XNF involves testing a condition on the functional dependencies implied by (D, Σ). Since this set of XFDs can be very large, it is desirable to find ways to reduce the number of XFDs to be considered. In the previous section we introduce the class of disjunctive DTDs. This class has the following useful property that lets us find efficient algorithms for testing XNF.

Proposition 6.4 [4] *Given a disjunctive DTD D and a set Σ of XFDs over D, (D, Σ) is in XNF iff for each nontrivial XFD $X \rightarrow p.@a \in \Sigma$, it is the case that $X \rightarrow p \in (D, \Sigma)^+$.*

From this and Theorems 6.1 and 6.2 we derive:

Corollary 6.5 [4] *Testing if (D, Σ) is in XNF can be done in cubic time for simple DTDs, and is coNP-complete for disjunctive DTDs.*

CoNP-completeness also holds for the case of relational DTDs [4]. As pointed out in the previous section, this theorem can also be seen as a positive result, as we can expect that in practice we will be able to test whether a specification is in XNF, since XML specifications tend to be relatively small in practice and today we can find SAT solvers that routinely solve NP problems with thousands of variables [17, 22].

We conclude this section by pointing out that from Theorem 6.3, we know that the problem of testing whether an XML specification is in XNF is decidable. The exact complexity of this problem is an open problem.

7 Justifying XNF

What constitutes a good database design? This question has been studied extensively, with well-known solutions presented in practically all database texts. But what is it that makes a database design good? This question is usually addressed at a much less formal level. For instance, we know that BCNF is an example of a good design, and we usually say that this is because BCNF eliminates update anomalies. Most of the time this is sufficient, given the simplicity of the relational model and our good intuition about it.

Several papers [15, 26, 21] attempted a more formal evaluation of normal forms, by relating it to the elimination of update anomalies. Another criterion is the existence of algorithms that produce good designs: for example, we know that every database scheme can be losslessly decomposed into one in BCNF, but some constraints may be lost along the way.

The previous work was specific for the relational model. As new data formats such as XML are becoming critically important, classical database theory problems have to be revisited in the new context [25, 23]. However, there is as yet no consensus on how to address the problem of well-designed data in the XML setting [12, 4, 28].

It is problematic to evaluate XML normal forms based on update anomalies; while some proposals for update languages exist [24], no XML update language has been standardized. Likewise, using the existence of good decomposition algorithms as a criterion is problematic: for example, to formulate losslessness, one needs to fix a small set of operations in some language, that would play the same role for XML as relational algebra for relations.

This suggests that one needs a different approach to the justification of normal forms and good designs. Such an approach must be applicable to new data models *before* the issues of query/update/constraint languages for them are completely understood and resolved. Therefore, such an approach must be based on some intrinsic characteristics of the data, as opposed to query/update languages for a particular data model. Such an approach was introduced in [5] based on information-theoretic concepts, more specifically, on measuring the information content of the data (or entropy [11]) of a suitably chosen probability distribution. The goal there was twofold. It was introduced an information-theoretic measure of "goodness" of a design, and tested in the relational world. It was shown in [5] that the measure can be used to characterize familiar relational normal forms such as BCNF, 4NF, and PJ/NF. Then the measure was applied in the XML context to show that it justifies the normal form XNF.

To present the information-theoretic measure proposed in [5], we need to introduce some terminology. Let $D = (L_0, P, R, r)$ be a DTD, Σ a set of constraints over D and $T = (N, E)$ an XML tree conforming to D and satisfying Σ. Then the set of positions in T, denoted by $Pos(T)$, is defined as $\{(x, @a) \mid x \in N,\ @a \in R(\lambda(x))\}$, and the active domain of T is defined as the set of all values of attributes in T. Without loss of generality, we assume that the active domain of every XML tree is contained in $\mathbb{N}^+$.

The goal in [5] is to define a function $\textsc{Inf}_T(p \mid \Sigma)$, the information content of a position $p \in Pos(T)$ with respect to the set of constraints Σ. To do this, it is defined, for each $k > 0$, a function $\textsc{Inf}^k_T(p \mid \Sigma)$ that would only apply to instances whose active domain is contained in $\{1, \ldots, k\}$. More precisely, let $X \subseteq Pos(T) - \{p\}$. Suppose the values in those positions X are lost, and then someone restores them from the set $1, \ldots, k$. It is measured how much information about the value in p this gives by calculating the entropy [11] of a suitably chosen probability distribution[1]. Then $\textsc{Inf}^k_T(p \mid \Sigma)$ is defined as the average such entropy over all sets $X \subseteq Pos(T) - \{p\}$, which corresponds to a conditional entropy. The ratio $\textsc{Inf}^k_T(p \mid \Sigma)/\log k$ is used in [5] to tell how close the given position p is to having the maximum possible information content, for XML trees with active domain in $[1, k]$ (recall that the maximum value of entropy is $\log k$ for a discrete distribution over k elements). Then measure $\textsc{Inf}_T(p \mid \Sigma)$ is taken to be the limit of these ratios as k goes to infinity. It is shown in [5] that such a limit exists for every XML tree and set Σ of XFDs.

$\textsc{Inf}_T(p \mid \Sigma)$ measures how much information is contained in position p, and $0 \leq \textsc{Inf}_T(p \mid \Sigma) \leq 1$. A well-designed schema should not have an instance with a position that has less than maximum information, as we do not expect to have redundant information on any instance of this schema. This motivates the following definition:

Definition 7.1 **[5]** *An XML specification (D, Σ) is* well-designed *if for every XML tree conforming to D and satisfying Σ, and every $p \in Pos(T)$, $\textsc{Inf}_T(p \mid \Sigma) = 1$.*

As in the case of relational databases (in particular, BCNF and 4NF), it is possible to show that well-designed XML and XNF coincide.

Theorem 7.2 **[5]** *Let D be a DTD and Σ a set of XFDs over D. Then (D, Σ) is well-designed if and only if (D, Σ) is in XNF.*

It is worth mentioning that an alternative notion of redundancy for XML trees is introduced in [28], where it is proved that an XML specification is in XNF if and only if no tree conforming to the specification contains redundant information.

8 Final Remarks

In this paper, we show how the concepts of database design and normal forms can be extended to XML

[1] Due to the lack of space we cannot formally introduce the probability distributions used in the definition of $\textsc{Inf}_T(p \mid \Sigma)$. See [5] for a detail description.

databases. More precisely, we introduce a functional dependency language for XML, we use this language to define a normal form for XML specifications, and we justify this normal form by using an information-theoretic approach.

To conclude the paper, we provide some links for further reading. Many dependency languages have been proposed for XML, see [7] for an early survey on this subject and [3] for a more recent survey including results on the complexity of reasoning about constraints in the presence of DTDs. Some of these languages have been used to define other normal forms for XML such as X3NF [18] and 4XNF [27]. The information-theoretic approach presented in Section 7 has also been used to provide justification for "non-perfect" relational normal forms such as 3NF [19].

Acknowledgments

The research presented in this article was conducted while the author was a Ph.D student at the University of Toronto under the supervision of Professor Leonid Libkin.

References

[1] S. Abiteboul, L. Segoufin, and V. Vianu. Representing and Querying XML with Incomplete Information. *TODS*, 31(1):208–254, 2006.

[2] M. Arenas. *Design Principles for XML Data*. PhD thesis, University of Toronto, 2005.

[3] M. Arenas, W. Fan, and L. Libkin. Consistency of XML Specifications. In *Inconsistency Tolerance*, pages 15–41, 2005.

[4] M. Arenas and L. Libkin. A Normal Form for XML Documents. *TODS*, 29(1):195–232, 2004.

[5] M. Arenas and L. Libkin. An Information-Theoretic Approach to Normal Forms for Relational and XML Data. *JACM*, 52(2):246–283, 2005.

[6] G. Jan Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: A Practical Study. In *WebDB*, pages 79–84, 2004.

[7] P. Buneman, W. Fan, J. Siméon, and S. Weinstein. Constraints for Semi-structured Data and XML. *SIGMOD Record*, 30(1):47–45, 2001.

[8] B. Choi. What are real DTDs like? In *WebDB*, pages 43–48, 2002.

[9] E. F. Codd. Further Normalization of the Data Base Relational Model. In *Data base systems*, pages 33–64. Englewood Cliffs, N.J. Prentice-Hall, 1972.

[10] E. F. Codd. Recent Investigations in Relational Data Base Systems. In *IFIP Congress*, pages 1017–1021, 1974.

[11] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[12] D. Embley and W. Y. Mok. Developing XML Documents with Guaranteed "Good" Properties. In *ER*, pages 426–441, 2001.

[13] R. Fagin. Multivalued Dependencies and a New Normal Form for Relational Databases. *TODS*, 2(3):262–278, 1977.

[14] R. Fagin. Normal Forms and Relational Database Operators. In *SIGMOD*, pages 153–160, 1979.

[15] R. Fagin. A Normal Form for Relational Databases That Is Based on Domians and Keys. *TODS*, 6(3):387–415, 1981.

[16] W. Fan and J. Siméon. Integrity Constraints for XML. In *PODS*, pages 23–34, 2000.

[17] E. Goldberg and Y. Novikov. BerkMin: A Fast and Robust Sat-Solver. In *DATE*, pages 142–149, 2002.

[18] S. Kolahi. Dependency-Preserving Normalization of Relational and XML Data. In *DBPL*, pages 247–261, 2005.

[19] S. Kolahi and L. Libkin. On Redundancy vs Dependency Preservation in Normalization: An Information-Theoretic Study of 3NF. In *PODS*, pages 114–123, 2006.

[20] M.-L. Lee, T. W. Ling, and W. L. Low. Designing Functional Dependencies for XML. In *EDBT*, pages 124–141, 2002.

[21] M. Levene and M. Vincent. Justification for Inclusion Dependency Normal Form. *TKDE*, 12(2):281–291, 2000.

[22] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *DAC*, pages 530–535, 2001.

[23] D. Suciu. On Database Theory and XML. *SIGMOD Record*, 30(3):39–45, 2001.

[24] I. Tatarinov, Z. Ives, A. Halevy, and D. Weld. Updating XML. In *SIGMOD*, pages 413–424, 2001.

[25] V. Vianu. A Web Odyssey: from Codd to XML. In *PODS*, pages 1–15, 2001.

[26] M. Vincent. Semantic Foundations of 4NF in Relational Database Design. *Acta Informatica*, 36(3):173–213, 1999.

[27] M. Vincent, J. Liu, and C. Liu. A Redundancy Free 4NF for XML. In *XSym*, pages 254–266, 2003.

[28] M. Vincent, J. Liu, and C. Liu. Strong Functional Dependencies and their Application to Normal Forms in XML. *TODS*, 29(3):445–462, 2004.