NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints

Dongwon Lee
Penn State Univ.
dongwon@psu.edu

Murali Mani *
UCLA
mani@cs.ucla.edu

Frank Chiu
UCLA
frankchiu@cs.ucla.edu

Wesley W. Chu
UCLA
wwc@cs.ucla.edu

ABSTRACT

Two algorithms, called NeT and CoT, to translate relational schemas to XML schemas using various semantic constraints are presented. The XML schema representation we use is a language-independent formalism named *XSchema*, that is both precise and concise. A given *XSchema* can be mapped to a schema in any of the existing XML schema language proposals. Our proposed algorithms have the following characteristics: (1) NeT derives a nested structure from a flat relational model by repeatedly applying the *nest* operator on each table so that the resulting XML schema becomes hierarchical, and (2) CoT considers not only the structure of relational schemas, but also semantic constraints such as inclusion dependencies during the translation. It takes as input a relational schema where multiple tables are interconnected through inclusion dependencies and converts it into a *good XSchema*. To validate our proposals, we present experimental results using both real schemas from the UCI repository and synthetic schemas from TPC-H.

Categories and Subject Descriptors

H.2.1 [**Logical Design**]: [Schema and subschema]; H.2.3 [**Languages**]: [Data description languages (DDL)]; H.2.5 [**Heterogeneous Databases**]: [Data translation]

General Terms

Algorithms

Keywords

XML, Schema Translation, Semantic Constraints

1. INTRODUCTION

XML [3] is rapidly becoming one of the most widely adopted technologies for information exchange and representation on the World Wide Web. With XML emerging as *the* data format of the Internet era, there is a substantial increase in the amount of data encoded in XML. However, the majority of everyday data is still stored and maintained in relational databases. Therefore, we expect the needs to convert such relational data into XML documents will grow substantially as well. In this paper, we study the problems in this conversion. Especially, we are interested in finding XML schema[1] (e.g., DTD [3], RELAX-NG [5], XML-Schema [22]) that *best* describes the existing relational schema. Having an XML schema that precisely describes the semantics and structures of the original relational data is important to further maintain the converted XML documents in future.

At present, there exist several tools that enable the composition of XML documents from relational data, such as XML Extender from IBM[2], XML-DBMS[3], DB2XML [23], SilkRoute [8], and XPERANTO [4]. In these tools, the success of the conversion is closely related with the quality of the target XML schema onto which a given input relational schema is mapped. However, the mapping from the relational schema to the XML schema is specified by human experts. Therefore, when large amount of relational schemas and data need to be translated into XML documents, a significant investment of human effort is required to initially design target schemas. To make matters worse, in the context of merging legacy relational data to existing XML documents, devising a good XML schema that does not violate existing structures and constraints is a non-trivial task. Being able to *automatically* infer a precise XML schema out of relational schema would be very useful in such settings.

In this paper, therefore, we are interested in finding a method that can infer the *best* XML schema from the given relational schema automatically. We particularly focus on two aspects of the translation: (1) **Structural aspect**: We want to find the most intuitive and precise XML schema structure from the given relational schema. We especially try to use the hidden characteristics of data using *nest* operator, and (2) **Semantic aspect**: During the translation, we want to use semantic constraints that could be either acquired from database directly or provided by human experts explicitly.

We first present a straightforward relational to XML trans-

*partially supported by NSF grants 0086116, 0085773, 9817773.

CIKM'02, November 4–9, 2002, McLean, Virginia, USA.

[1]We differentiate two terms – XML schema(s) and XML-Schema. The former refers to a general term for schema in XML model while the latter refers to one particular kind of XML schema language proposed by W3C [22].
[2]http://www-4.ibm.com/software/data/db2/extenders/xmlext/
[3]http://www.rpbourret.com/xmldbms/index.htm

lation algorithm, called *Flat Translation* (FT). Since FT maps the flat relational model to the flat XML model in a one-to-one manner, it does not utilize the regular expression operators (e.g., "*", "+") supported in the content models of XML. Then, we present our first proposal called *Nesting-based Translation* (NeT), to remedy the problems found in FT. NeT derives nested structures from a flat relational model by the use of the *nest* operator so that the resulting XML schema is more intuitive and precise than otherwise. Although NeT infers hidden characteristics of data by nesting, it is only applicable to a single table at a time. Therefore, it is unable to capture a correct "big picture" of relational schema where many tables are interconnected. To remedy this problem, we present the second proposal called *Constraints-based Translation* (CoT); CoT considers inclusion dependencies during the translation. Such constraints can be acquired from database through ODBC/JDBC interface or provided by human experts who are familiar with the semantics of the relational schema being translated. CoT is capable of generating a more intuitive XML schema than what NeT. Figure 1 illustrates the overview of our approach.

Related Work: There have been different approaches for the conversion from relational model to XML model, such as XML Extender from IBM, XML-DBMS, SilkRoute [8], XPERANTO [4], DB2XML [23] and NeT [14]. All the above tools (except NeT) require the user to specify the mapping from the given relational model to the XML model. In XML Extender, the user specifies the mapping through a language such as DAD or XML Extender Transform Language. In XML-DBMS, a template-driven mapping language is provided to specify the mappings. SilkRoute provides a declarative query language (RXL) for viewing relational data in XML. XPERANTO uses XML query language for viewing relational data in XML. Note that in SilkRoute and XPERANTO, the user has to specify the query in the appropriate query language. DB2XML uses an algorithm similar to FT (and hence suffers from similar problems). NeT does not require user-input for mapping the relational model to XML model, however it does not use semantic constraints specified in the relational model.

There also have been work in mapping from non-relational models to XML model, and XML to relational and other models. In [20, 21], the authors study the conversion from XML to relational models. [15] studies the conversion from XML to ER model and vice versa. Generation of an XML schema from a UML model is studied in [18]. Given a set of XML documents, generating an XML schema for them is studied in [10].

Roadmap: In Section 2, we first present formalisms to represent relational as well as XML schemas in a language independent notation. In Section 3, we propose NeT algorithm that uses the *nest* operator developed in the nested relational model community. In Section 4, we propose an improved CoT algorithm that considers various semantic constraints during the translation to generate a better XML schema, in addition to applying *nest* operations for each table. In Section 5, we discuss some issues related to correctness and goodness of the schema that NeT and CoT generate. In Section 6, we report the results from our experimentations. Finally, concluding remarks and future directions are discussed in Section 7.

2. INPUT & OUTPUT MODELS

We first briefly define the input and output models for the translation. In relational databases, schema is typically created by SQL DDL (e.g., `CREATE`) statements. Therefore, by examining such DDL statements, one can find out the original schema information. Even if such DDL statements are not available, one can still infer the schema information - table and column names, key and foreign key information, etc - by querying the database through an ODBC/JDBC interface or by examining the database directly. In this paper, regardless of how one acquired the schema information, we assume that the schema information is encoded in a vector $\mathbb{R}$ defined below.

Let us assume the existence of a set $\widehat{T}$ of table names, a set $\widehat{C}$ of column names and a set $\widehat{b}$ of atomic base types defined in the standard SQL (e.g., integer, char, string). When name collision occurs, a column name $c \in \widehat{C}$ is qualified by a table name $t \in \widehat{T}$ using the "[ ]" notation (e.g., $t[c]$).

Definition 1 (Relational Schema) *A* relational schema *is denoted by 4-tuple* $\mathbb{R} = (T, C, P, \Delta)$*, where:*

- T *is a finite set of table names in* $\widehat{T}$*;* C *is a function from a table name* $t \in T$ *to a set of column names* $c \in \widehat{C}$*,*
- P *is a function from a column name* c *to its* column type definition*: i.e.,* $P(c) = \alpha$*, where* α *is a 5-tuple* (τ, u, n, d, f)*, where* $\tau \in \widehat{b}$*,* u *is either "*υ*" (unique) or "*$\neg\upsilon$*" (not unique),* n *is either "?" (nullable) or "*$\neg$*?" (not nullable),* d *is a finite set of valid domain values of* c *or* ϵ *if not known, and* f *is a default value of* c *or* ϵ *if not known, and*
- Δ *is a finite set of relational integrity constraints that can be either retrieved from databases directly or provided by human experts.* □

Example 1. Consider two tables `student(`<u>`Sname`</u>`, `<u>`Advisor`</u>`, `<u>`Course`</u>`)` and `professor(`<u>`Pname`</u>`, Office)` where keys are underlined, and `Advisor` is a foreign key referencing `Pname` column. The column `Office` is an integer type, while the rest of the columns are string types. Also `Office` may be null. When student's advisor has not yet been decided, professor "Prof. Smith" will be the initial advisor. Student can have many advisors and take zero or more courses. The corresponding relational schema and data fragment are given in Table 1. □

Next, let us define the output model. Lately, there have been about a dozen competing XML schema language proposals. Although XML-Schema is being shaped by W3C and will replace DTD soon, it is likely that different applications will choose different XML schema languages that best suit their particular purposes. Therefore, instead of choosing one language proposal, we formalize a core set of important features into a new notion of *XSchema* and use it as our output modeling language. The benefits of such formalization is that it is both concise and precise. More importantly, it breaks the tie between the translation algorithm that we are developing and the final schema language notations. Informally, *XSchema* borrows structural features from DTD and RELAX-NG, and data types and constraint

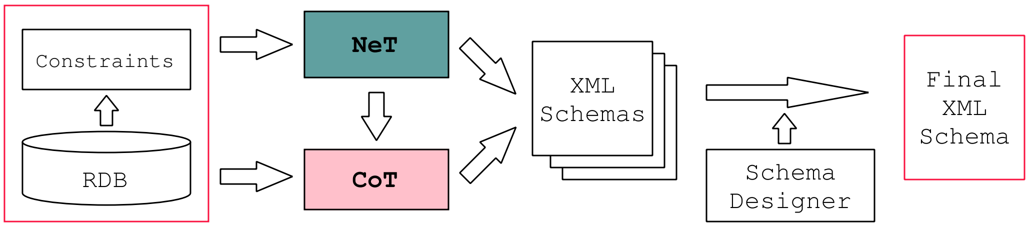

Figure 1: Overview of our approach. Two algorithms, NeT and CoT, can be used independently or jointly; (1) $\mathbb{R} \rightarrow$ NeT $\rightarrow \mathbb{X}$, (2) $\mathbb{R} \rightarrow$ CoT $\rightarrow \mathbb{X}$, or (3) $\mathbb{R} \rightarrow$ NeT $\rightarrow$ CoT $\rightarrow \mathbb{X}$.

$$
\begin{aligned}
T &= \{student, professor\} \\
C(student) &= \{Sname, Advisor, Course\} \\
C(professor) &= \{Pname, Office\} \\
P(Sname) &= (string, \neg v, \neg ?, \epsilon, \epsilon) \\
P(Advisor) &= (string, \neg v, \neg ?, \epsilon, \text{"}Prof.Smith''\text{)} \\
P(Course) &= (string, \neg v, \neg ?, \epsilon, \epsilon) \\
P(Pname) &= (string, v, \neg ?, \epsilon, \epsilon) \\
P(Office) &= (integer, \neg v, ?, \epsilon, \epsilon) \\
\Delta &= \{\{Sname, Advisor, Course\} \stackrel{key}{\rightarrow} student, \\
&\quad Pname \stackrel{key}{\rightarrow} professor, \ Advisor \subseteq Pname
\end{aligned}
$$

student:

Sname	Advisor	Course
John	Prof. Muntz	Multimedia
John	Prof. Zaniolo	Logic
John	Prof. Zaniolo	Data Mining
Tom	Prof. Muntz	Queueing Theory
Tom	Prof. Chu	Database Systems
Tom	Prof. Chu	Distributed Databases

professor:

Pname	Office
Prof. Muntz	600
Prof. Chu	550
Prof. Zaniolo	_
Prof. Parker	490

Table 1: Example relational schema and data.

specification features from XML-Schema. From a formal language and database perspective [17], *XSchema* is a local tree grammar extended with attribute, datatype and constraint specifications.

Starting from the notations in [7], we define *XSchema* below. We first assume the existence of a set $\widehat{E}$ of element names, a set $\widehat{A}$ of attribute names and a set $\widehat{\tau}$ of atomic data types defined in [1] (e.g., ID, IDREF, string, integer, date, etc). When needed, an attribute name $a \in \widehat{A}$ is qualified by the element names using the *path expression* notation $e_1.e_2 \cdots e_n.a$, where $e_i \in \widehat{E}, 1 \leq i \leq n$).

Definition 2 (*XSchema*) *An XSchema is denoted by 6-tuple* $\mathbb{X} = (E, A, M, P, r, \Sigma)$, *where:*

- *E is a finite set of element names in $\widehat{E}$; A is a function from an element name $e \in E$ to a set of attribute names $a \in \widehat{A}$,*
- *M is a function from an element name $e \in E$ to its element type definition: i.e., $M(e) = \alpha$, where α is a regular expression: $\alpha ::= \epsilon \mid \tau \mid \alpha + \alpha \mid \alpha, \alpha \mid \alpha^? \mid \alpha^* \mid \alpha^+$, where ϵ denotes the empty element, $\tau \in \widehat{\tau}$, "+" for the union, "," for the concatenation, "$\alpha^?$" for zero or one occurrence, "α^*" for the Kleene star, and "α^+" for "α, α^*",*
- *P is a function from an attribute name a to its attribute type definition: i.e., $P(a) = \beta$, where β is a 4-tuple (τ, n, d, f), where $\tau \in \widehat{\tau}$, n is either "?" (nullable) or "¬?" (not nullable), d is a finite set of valid domain values of a or ϵ if not known, and f is a default value of a or ϵ if not known, and*
- *$r \subseteq E$ is a finite set of root elements; Σ is a finite set of integrity constraints for XML model* □

Translation from *XSchema* to the actual XML schema language notations is relatively straightforward and not discussed further in this paper. It is worthwhile to note, however, that depending on the chosen XML schema language, some of the features specifiable in *XSchema* might not be translatable at the end. For instance, any "non-trivial type" or composite key information would be lost if one decides to use DTD as the final XML schema language.

3. FLAT TRANSLATION AND NESTING-BASED TRANSLATION

XML model uses two basic building blocks to construct XML documents – attribute and element. A few basic characteristics inherited from XML model include: (1) the attributes of a node are not ordered, while the child elements of a node are ordered, (2) both support data types as specified in [1], and (3) elements can express multiple occurrences better than attributes. The detailed capabilities of those, however, vary depending on the chosen XML schema language. In translating $\mathbb{R}$ to $\mathbb{X}$, therefore, one can either use attribute or element in $\mathbb{X}$ to represent the same entity in $\mathbb{R}$ (e.g, a column with string type in $\mathbb{R}$ can be translated to either attribute or element with string type in $\mathbb{X}$).

To increase the flexibility of the algorithms, we assume that there are two modes – **attribute-oriented** and **element-oriented**. Depending on the mode, an algorithm can selectively translate an entity in $\mathbb{R}$ to either attribute or element if both can capture the entity correctly. However, if the chosen XML schema language requires attribute or element for an entity (e.g., a key column in $\mathbb{R}$ needs to be translated to an attribute with type `ID` in $\mathbb{X}$), we assume that the algorithm follows the limitations.

3.1 Flat Translation

The simplest translation method is to translate (1) tables in $\mathbb{R}$ to elements in $\mathbb{X}$ and (2) columns in $\mathbb{R}$ to attributes (in attribute-oriented mode) or elements (in element-oriented mode) in $\mathbb{X}$. These two modes are analogous except that element-oriented mode adds additional order semantics to the resulting schema. Since $\mathbb{X}$ represents the "flat" relational tuples faithfully, this method is called *Flat Translation* (FT). The general procedure of the **Flat Translation** is omitted in the interest of space and can be found in [14].

FT is a simple and effective translation algorithm, but it has some problems. As the name implies, FT translates the "flat" relational model to a "flat" XML model in a one-to-one manner. The drawback of FT is that it does not utilize several basic "non-flat" features provided by XML for data modeling such as representing *repeating sub-elements* through regular expression operators (e.g., "*", "+"). We remedy this problem in the NeT algorithm below.

3.2 Nesting-based Translation

To remedy the problems of FT, one needs to utilize various *element content models* of XML. Towards this goal, we propose to use the *nest* operator [12]. Our idea is to find a "best" element content model that uses α^* or α^+ using the *nest* operator. First, let us define the *nest* operator. Informally, for a table t with a set of columns C, *nesting* on a non-empty column $X \in C$ collects all tuples that agree on the remaining columns $C - X$ into a set[4]. Formally,

Definition 3 (Nest) *[12]. Let t be a n-ary table with column set C, and $X \in C$ and $\overline{X} = C - X$. For each $(n-1)$-tuple $\gamma \in \Pi_{\overline{X}}(t)$, we define an n-tuple γ^* as follows: $\gamma^*[\overline{X}] = \gamma$, and $\gamma^*[X] = \{\kappa[X] \mid \kappa \in t \wedge \kappa[\overline{X}] = \gamma$. Then, $nest_X(t) = \{\gamma^* \mid \gamma \in \Pi_{\overline{X}}(t)\}$.* □

After $nest_X(t)$, if column X has only a set with "single" value $\{v\}$ for all the tuples, then we say that **nesting failed** and we treat $\{v\}$ and v interchangeably (i.e., $\{v\} = v$). Thus when nesting failed, the following is true: $nest_X(t) = t$. Otherwise, if column X has a set with "multiple" values $\{v_1, ..., v_k\}$ with $k \geq 2$ for at least one tuple, then we say that **nesting succeeded**. The general procedure for nesting is given in [14].

Example 2. Consider a table R in Table 2. Here we assume that the columns A, B, C are non-nullable. In computing $nest_A(R)$ at (b), the first, third, and fourth tuples of R agree on their values in columns (B, C) as (a, 10), while their values of the column A are all different. Therefore, these different values are grouped (i.e., nested) into a set {1,2,3}. The result is the first tuple of the table $nest_A(R)$ – ({1,2,3}, a, 10). Similarly, since the sixth and seventh tuples of R agree on their values as (b, 20), they are grouped to a set {4,5}. In computing $nest_B(R)$ at (c), there are no tuples in R that agree on the values of the columns (A, C). Therefore, $nest_B(R) = R$. In computing $nest_C(R)$ at (d), since the first two tuples of R – (1, a, 10) and (1, a, 20) – agree on the values of the columns (A, B), they are grouped to (1, a, {10,20}). Nested tables (e) through (j) are constructed similarly. □

Since the *nest* operator requires scanning of the entire set of tuples in a given table, it can be quite expensive. In addition, as shown in Example 2, there are various ways to nest the given table. Therefore, it is important to find an efficient way (that uses the *nest* operator minimum number of times) of obtaining an acceptable element content model.

[4]Here, we only consider single attribute nesting.

First, to find out the total number of ways to nest, let us use the following two properties [12]:

$$
\begin{array}{llcl}
P1: & nest_A(nest_B(t)) & \neq & nest_B(nest_A(t)) \\
P2: & nest_X(nestAll_L(t)) & = & nestAll_L(t), \text{ if } X \in L
\end{array}
$$

Here, $nestAll_L(t)$ represents performing nesting on a list of columns as indicated by L. If $L = < c_1, c_2, \ldots, c_n >$, then $nestAll_L(t)$ is given by
$nestAll_{L=<c_1,c_2,\ldots,c_n>}(t) = nest_{c_1}(nest_{c_2}(\ldots(nest_{c_n}(t))))$

P1 states that "commutativity" of nesting does not hold in general and P2 states that nesting along the same column repeatedly has the property of "idempotency". Using the two properties, the number of permutations to nest tables can be described as follows:

Remark 1 *Using the* falling factorial power *notation "x to the m falling" as $x^{\underline{m}}$ in [11], the total number of different nestings N for a table with n columns is given by: $N = \sum_{k=1}^{n} n^{\underline{k}}$* □

According to Remark 1, there are 15 meaningful ways of nesting along the columns A, B, C in Table 2. Then, the next questions are (1) how to decrease N by avoiding unnecessary nesting, and (2) which nesting should be chosen as *the* translation. To answer these questions, let us first describe a few useful properties of the *nest* operator as follows:

Lemma 1. *Consider a table t with column set C, and candidate keys, $K_1, K_2, \ldots, K_n \subseteq C$. Applying the nest operator on a column $X \notin (K_1 \cap K_2 \cap \ldots \cap K_n)$ yields no changes.* ■

COROLLARY 1. For any nested table $nest_X(t)$, $\overline{X} \rightarrow X$ holds. (q.e.d)

Corollary 1 states that after applying the nest operator of column X, the remaining columns $\overline{X}$ become a super key. Fischer et al. [9] have proved that functional dependencies are preserved against nesting as follows:

Lemma 2. [9] *If X, Y, Z are columns of t, then: $t: X \rightarrow Y \Longrightarrow nest_Z(t): X \rightarrow Y$* ■

Now, we arrive at the following useful property:

Theorem 1. *Consider a table t with column set C, candidate keys, $K_1, K_2, \ldots, K_n \subseteq C$, and column set K such that $K = K_1 \cap K_2 \cap \ldots \cap K_n$. Further, let $|C| = n$ and $|K| = m$ $(n \geq m)$. Then, the number of necessary nestings, N, is bounded by $N \leq \sum_{k=1}^{m} m^{\underline{k}}$* ■

Note that in general m is much smaller than n in Theorem 1, thus reducing the number of necessary nesting significantly.

Example 3. Consider a table R in Table 2 again. Suppose attributes A and C constitute a key for R. Since nesting on the same column repeatedly is not useful by property P2 there is no need to construct, for instance, $nest_A(nest_A(R))$. Since nesting on a non-key column is not useful by Lemma 1, nesting along column B (e.g., $nest_B(R)$

	A	B	C
#1	1	a	10
#2	1	a	20
#3	2	a	10
#4	3	a	10
#5	4	b	10
#6	4	b	20
#7	5	b	20

(a) R

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(b) $nest_A(R)$

A	B	C
1	a	10
1	a	20
2	a	10
3	a	10
4	b	10
4	b	20
5	b	20

(c) $nest_B(R) = R$

A	B	C^+
1	a	{10,20}
2	a	10
3	a	10
4	b	{10,20}
5	b	20

(d) $nest_C(R)$

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(e) $nest_B(nest_A(R)) = nest_C(nest_A(R))$

A^+	B	C^+
1	a	{10,20}
{2,3}	a	10
4	b	{10,20}
5	b	20

(f) $nest_A(nest_C(R))$

A	B	C^+
1	a	{10,20}
2	a	10
3	a	10
4	b	{10,20}
5	b	20

(g) $nest_B(nest_C(R))$

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(h) $nest_C(nest_B(nest_A(R))) = nest_B(nest_C(nest_A(R)))$

A^+	B	C^+
1	a	{10,20}
{2,3}	a	10
4	b	{10,20}
5	b	20

(i) $nest_B(nest_A(nest_C(R))) = nest_A(nest_B(nest_C(R)))$

Table 2: A relational table R and its various nested forms. Column names containing a set after nesting (i.e., nesting succeeded) are appended by "+" symbol.

at (c)) can be avoided. Furthermore, the functional dependency (i.e., $AC \overset{key}{\to} R = AC \to \overline{AC} = AC \to B$) persists after nesting on either column A or C by Lemma 2. Consequently, one needs to construct only the following nested tables: $nest_A(R)$ at (b), $nest_C(R)$ at (d), $nest_C(nest_A(R))$ at (e), $nest_A(nest_C(R))$ at (f). □

As we have shown, when candidate key information is available, the number of nestings to be performed can be reduced. However, when such information is not known, the *nest* operator must be applied for all possible combinations in Remark 1. After applying the *nest* operator to the given table repeatedly, there can be still several nested tables where nesting succeeded. In general, the choice of the final schema should take into consideration the semantics and usages of the underlying data or application and this is where user intervention is beneficial. By default, without further input from users, NeT chooses as the final schema the nested table where the most number of nestings succeeded; this is a schema which provides low "data redundancy".

Example 4. Using NeT with the element-oriented mode, $\mathbb{R}_1$ in Example 1 would be translated to $\mathbb{X}_4 = (E, A, M, P, r, \Sigma)$, where

$$\begin{aligned} E &= \{student, professor\} \\ A(professor) &= \{Pname\} \\ M(student) &= (Sname, Advisor^+, Course^+) \\ M(professor) &= (Age^?) \\ P(Pname) &= (ID, \neg?, \epsilon, \epsilon) \\ r &= \{student, professor\} \end{aligned}$$

$$\Sigma = \{\{Sname, Advisor, Course\} \overset{key}{\to} student, \quad Pname \overset{key}{\to} professor, Advisor \subseteq Pname\}$$

□

We expect that the NeT algorithm will be especially useful in two scenarios, outlined below.

- The given relation is in 3NF (or BCNF) but not in 4NF. Non-fully normalized relations occur quite commonly in legacy databases, and they exhibit data redundancy. The NeT algorithm helps to decrease the data redundancy in such cases. As an example, consider the relation `ctx(Course, Teacher, Text)`, which gives the set of teachers and the set of text books for each course. Assume that the following multivalued dependencies hold, `Course` ↠ `Teacher`, and `Course`↠`Text`. Suppose the relation *ctx* is represented as such (i.e., `ctx` is not in 4NF). The key for this relation is given by {`Course, Teacher, Text`}$\overset{key}{\to}$`ctx`. When we do nesting on `ctx`, we will get the following table `ctx'(Course, Teacher`$^+$`, Text`$^+$`)`. Thus NeT helps in removing data redundancies arising from multivalued dependencies.
- It is sometimes possible to represent the given relation "more intuitively" as a nested table by performing grouping on one or more of the attributes. As an example, consider the relation `emp(empNum, branch)` where the key is given by `empNum` $\overset{key}{\to}$ `emp`. This relation gives the employees and the branch where they work. When NeT is applied on the above relation, we might get the new nested relation as `emp'(empNum`$^+$`, branch)`. This relation has grouped the list of employees by their branch.

Thus we observe that NeT is useful for decreasing data redundancy and obtaining a "more intuitive" schema by (1) removing redundancies caused by multivalued dependencies and (2) performing grouping on attributes. However NeT considers tables one by one, and *cannot* obtain a *big picture* of the relational schema where many tables are interconnected with each other through various other dependencies such as inclusion dependencies. To remedy this problem, we propose the second conversion algorithm below.

4. TRANSLATION USING INCLUSION DEPENDENCIES

In this section, we consider one kind of semantic constraints called *Inclusion Dependency (IND)* in database theory. Considering other constraints such as *Functional Dependency (FD)* or *Multi-Valued Dependency (MVD)* is also possible, but we leave it as a future work. General forms of INDs are difficult to acquire from the database automatically. However, we shall consider the most pervasive form of INDs – foreign key constraints – which can be queried through ODBC/JDBC interface. We study the translation of inclusion dependencies incrementally in three steps. In the first step, we consider the simplest case – one foreign key constraint defined between two tables. In the second step, we consider the case when there exist two foreign key constraints among three tables. In the third step, we consider the general case of any number of inclusion dependencies in a schema.

4.1 One Foreign Key between two Tables

Foreign key constraints are a special kind of INDs where the attributes being referenced form the *primary key* of the referenced relation. For two distinct tables s and t with lists of columns X and Y, respectively, suppose we have a foreign key constraint $s[\alpha] \subseteq t[\beta]$, where $\alpha \subseteq X$ and $\beta \subseteq Y$. Also suppose that $K_s \subseteq X$ is the key for s. Then, rewriting this in $\mathbb{R}$ notation, we have: $T = \{s, t\}, C(s) = \{X\}, C(t) = \{Y\}, \Delta = \{s[\alpha] \subseteq t[\beta], \beta \stackrel{key}{\rightarrow} t, K_s \stackrel{key}{\rightarrow} s\}$.

Different cardinality binary relationships between s and t can be expressed in the relational model by a combination of the following: (1) α is unique/not-unique (2) α is nullable/non-nullable. Then, the translation of two tables s, t with a foreign key constraint into *XSchema*, summarized in Table 3, works as follows:

- If α is non-nullable (i.e., none of the columns of α can take null values), then:
  - If α is unique, then there is a 1 : 1 relationship between s and t. This can be captured as a sub-element $M(t) = (Y, s?)$.
  - If α is not-unique, then there is a 1 : n relationship between s and t, and this is captured as a sub-element $M(t) = (Y, s^*)$.

  If s is represented as a sub-element of t, then the key for s will change from K_s to $(K_s - \alpha)$. The key for t will remain the same.
- If α is nullable, then the IND is represented as such in *XSchema*. Here we do flat translation on s, and copy the IND $s[\alpha] \subseteq t[\beta]$ to Σ.

Let us study the case when α is nullable more closely with the following example. Consider the relation $t(w_1, w_2, w_3)$ with key (w_1, w_2). Let t have the following tuples: $\{(1, 1, 1)\}$. Now consider $s(v_1, v_2, v_3)$ with key (v_2, v_3), and IND $s[v_1, v_2] \subseteq t[w_1, w_2]$. Let s have the following tuples: $\{(null, 1, 1), (null, 1, 2), (null, 2, 1), (1, 1, 3)\}$. We can observe that we *cannot* represent s as $s(v_3)$, and obtain the values of (v_1, v_2) for an s tuple by representing this s tuple as a child of a t tuple, or by having an IDREF attribute for the s tuple that refers to a t tuple. This is because v_1 is nullable. In such a case, we represent the IND as such in *XSchema*. In this paper, we are concerned mostly with the usage of sub-elements and IDREF attribute for translation, and therefore, we will focus on the case when α is non-nullable, unless stated otherwise.

Example 5. Consider two tables `student` and `professor` of Example 1 again. There is a foreign key *Advisor* $\subseteq$ *Pname* and *Advisor* is not unique. Using the above rules, the schema will be mapped to the following XML schema in DTD notation:

α	$s : t$	*XSchema*
υ, $\neg ?$	$(1,1):(0,1)$	$M(t) = (Y, s?)$, $M(s) = (X - \alpha)$, $\Sigma = \{(K_s - \alpha) \stackrel{key}{\rightarrow} s, \beta \stackrel{key}{\rightarrow} t\}$
υ, $?$	$(0,1):(0,1)$	$M(t) = (Y)$, $M(s) = (X)$, $\Sigma = \{s[\alpha] \subseteq t[\beta]$, $K_s \stackrel{key}{\rightarrow} s, \beta \stackrel{key}{\rightarrow} t\}$
$\neg\upsilon$, $\neg ?$	$(1,1):(0,n)$	$M(t) = (Y, s^*)$, $M(s) = (X - \alpha)$, $\Sigma = \{(K_s - \alpha) \stackrel{key}{\rightarrow} s, \beta \stackrel{key}{\rightarrow} t\}$
$\neg\upsilon$, $?$	$(0,1):(0,n)$	$M(t) = (Y), M(s) = (X)$, $\Sigma = \{s[\alpha] \subseteq t[\beta]$, $K_s \stackrel{key}{\rightarrow} s, \beta \stackrel{key}{\rightarrow} t\}$

Table 3: Different values taken by α, the corresponding cardinality of the binary relationship between s and t, and the corresponding translation to *XSchema*. υ and ? denote "unique" and "nullable", respectively.

```
<!ELEMENT professor  (Pname,Age,student*)>
<!ELEMENT student    (Sname,Course)>
```

Note the usage of `*` attached to the sub-element *student*. Note further that to identify a unique `student` element for a given professor, one needs now only *Sname* and *Course* pair (*Advisor* attribute was removed from the original key attribute list). □

4.2 Two Foreign Keys among three Tables

Now consider the case where two foreign key constraints exist among three tables s, t_1, t_2 with a list of columns X, Y_1, Y_2, respectively, such that $s[\alpha] \subseteq t_1[\beta_1]$ and $s[\gamma] \subseteq t_2[\beta_2]$, where $\alpha, \gamma \subseteq X$ and are non-nullable, $\beta_1 \subseteq Y_1$ and $\beta_2 \subseteq Y_2$. If one applies the mapping rules for the case of a foreign key between two tables in Section 4.1 one at a time, then one will have a combination of the following depending on whether α and γ are unique or not: (1) $M(t_1) = (Y_1, s?)$ or $M(t_1) = (Y_1, s^*)$, (2) $M(t_2) = (Y_2, s?)$ or $M(t_2) = (Y_2, s^*)$.

The above translation has redundancy, and it exhibits the phenomenon known in database theory as "update anomaly" for s. That is, when one wants to update data for s, he/she needs to update s in two different places – fragment of s data under both t_1 and t_2. On the contrary, the original relational schema is "better" because one needs to update tuples of s in a single place. The same problem occurs for the case of "delete" as well. To avoid these anomalies, one of the two foreign key constraints should be captured either using INDs or using IDREF attributes. For example, let us assume that the first foreign key constraint $s[\alpha] \subseteq t_1[\beta_1]$ is represented as $M(t_1) = (Y_1, s^*)$, $M(s) = (X - \alpha)$. Then the second foreign key constraint $s[\gamma] \subseteq t_2[\beta_2]$ can be represented using IDREF attribute as follows:

$$A(t_2) = \{ID\_t_2\}, \qquad P(ID\_t_2) = (ID, \neg ?, \epsilon, \epsilon)$$
$$A(s) = \{Ref\_t_2\}, \qquad P(Ref\_t_2) = (IDREF, \neg ?, \epsilon, \epsilon)$$
$$M(t_2) = (Y_2), \qquad M(s) = (X - \alpha - \gamma)$$

Let us denote the old and new keys for s as K_s and K'_s, respectively. Then, K'_s is determined as follows: (1) if $\alpha \cap K_s = \phi$, then $K'_s = K_s$, and (2) if $\alpha \cap K_s \neq \phi$, then $K'_s = (K_s - \alpha) \cup Ref\_t_2$

Example 6. In addition to two tables `student` and `professor` of Example 5, consider a third table `class(`<u>`Cname`</u>`, Room)` with a second foreign key $student[Course] \subseteq class[Cname]$. Then, using the above rules, the schema will be mapped to the following XML schema in DTD notation:

```
<!ELEMENT professor  (Pname,Age,student*)>
<!ELEMENT student    (Sname)>
<!ATTLIST student     Ref_class IDREF>
<!ELEMENT class      (Cname,Room)>
<!ATTLIST class       ID_class ID>
```

Note the addition of two new attributes – `Ref_class` of type $IDREF$ and `ID_class` of type ID. The new key for *student* is given by $\{Sname, Ref\_class \stackrel{key}{\rightarrow} student\}$, which cannot be represented in DTD. □

Note that between two foreign keys, deciding which one is represented as sub-element and which one is represented as IDREF attribute can best be done based on further semantics.

4.3 A General Relational Schema

Now let us consider the most general case with set of tables $\{t_1, ..., t_n\}$ and INDs $t_i[\alpha_i] \subseteq t_j[\beta_j]$, where $i, j \le n$. We consider only those INDs where α_i is non-nullable. The relationships among tables can be captured by a graph representation, termed as IND-Graph.

Definition 4 (IND-Graph) *An* IND-Graph $G = (V, E)$ *consists of a node set* V *and a directed edge set* E*, such that for each table* t_i*, there exists a node in* V*, and for each distinct IND* $t_i[\alpha] \subseteq t_j[\beta]$, $t_j \rightarrow t_i$ *exists in* G. □

Note the edge direction is reversed from the IND direction for convenience. Given a set of INDs, such IND-Graph can be easily constructed. Once IND-Graph is constructed, one needs to decide the starting point to apply translation rules. For that purpose, we use the notion of **top nodes** similar to the one in [20, 13], where an element is a top node if it *cannot* be represented as a sub-element of any other element. Such top nodes can be identified as follows:

1. An element s is a top node, if there exists no other element t, $t \neq s$, where there is a IND of the form $s[\alpha] \subseteq t[\beta]$, and α is non-nullable.

2. Consider a set of elements $S = s_1, s_2, \ldots, s_k$ that form a cyclic set of INDs and none of the elements in S is a top node by 1. Suppose there exists no element $t \notin S$, such that there is a IND of the form $s_j[\alpha] \subseteq t[\beta]$, and α is non-nullable. In this case, choose any one of the elements in S as a top node.

Let T denote the set of top nodes. After identifying the top nodes, we traverse G, using say Breadth-First Search (BFS), until we traverse all the nodes and edges, and represent the INDs as sub-elements or IDREF attributes. The algorithm for **Constraint-based Translation** (CoT) is as follows:

1. CoT: $\mathbb{R} = (T, C, P, \Delta) \Longrightarrow \mathbb{X} = (E, A, M, P, r, \Sigma)$
2. Construct IND-Graph $G = (V, E)$ from the given INDs; Identify T, the set of top nodes. Define $S = T$ to keep track of top-nodes and nodes that are represented as sub-elements.
3. For each top-node $t \in T$, do BFS. Suppose we reach a node w from v (i.e., IND: $w[\alpha] \subseteq v[\beta]$); Let $C(w) = C_w$, and $C(v) = C_v$.
   (a) If $w \notin S$ (i.e., w is *not* yet a sub-element of some other node), translate the IND as in Section 4.1.
      i. If α is unique, then $M(v) = (C_v, w?)$.
      ii. If α is not-unique, then $M(v) = (C_v, w^*)$.
      iii. $M(w) = (C_w - \alpha)$.
      iv. $S = S \cup w$.
   (b) If $w \in S$ (i.e., w is already a sub-element of some other node), translate the IND as IDREF attribute as in Section 4.2.
      i. $A(v) = \{ID\_v\}$, $A(w) = \{Ref\_v\}$, $M(v) = (C_v)$, $M(w) = (C_w - \alpha)$, $\Sigma = K'_w \stackrel{key}{\rightarrow} w$.
4. Copy the remaining integrity constraints in Δ to Σ. Also set $r = T$.

`student(Sid, Name, Advisor)`
`emp(Eid, Name, ProjName)`
`prof(Eid, Name, Teach)`
`course(Cid, Title, Room)`
`dept(Dno, Mgr)`
`proj(Pname, Pmgr)`
`student(Advisor)` ⊆ `prof(Eid)`
`emp(ProjName)` ⊆ `proj(Pname)`
`prof(Teach)` ⊆ `course(Cid)`
`prof(Eid, Name)` ⊆ `emp(Eid, Name)`
`dept(Mgr)` ⊆ `emp(Eid)`
`proj(Pmgr)` ⊆ `emp(Eid)`

Table 4: An example schema with associated INDs.

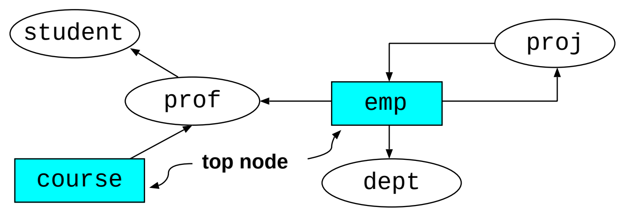

Figure 2: The IND-Graph representation of the schema of Table 4.

Example 7. Consider an example schema and its associated INDs in Figure 4. Two top nodes are identified (1) `course`: There is no node t, where there is an IND of the form `course` $[\alpha] \subseteq t[\beta]$, and (2) `emp`: There is a cyclic set of INDs between `emp` and `proj`, and there exists no node t such that there is an IND of the form `emp` $[\alpha] \subseteq t[\beta]$ or `proj` $[\alpha] \subseteq t[\beta]$. Therefore of `emp` and `proj` we decided to choose `emp` arbitrarily. Following list shows one of the possible orders in which the different INDs are visited, the choice made to represent the IND (either sub-element or IDREF attribute), and the resulting changes in *XSchema*.

1. `prof(Teach)` ⊆ `course(Cid)`: M (`course`) = (`Cid`, `Title`, `Room`, `prof`*), M (`prof`) = (`Eid`, `Name`)

2. `student(Advisor)` ⊆ `prof(Eid)`: M(`prof`) = (`Eid`, `Name`, `student`*), M(`student`) = (`Sid`, `Name`)

3. `dept(Mgr)` ⊆ `emp(Eid)`: M(`emp`) = (`Eid`, `Name`, `ProjName`, `dept`*), M(`dept`) = (`Dno`)

4. `proj(Pmgr)` ⊆ `emp(Eid)`: M(`emp`) = (`Eid`, `Name`, `ProjName`, `dept`*, `proj`*), M(`proj`) = (`Pname`)

5. `emp(ProjName)` ⊆ `proj(Pname)`: M(`emp`) = (`Eid`, `Name`, `dept`*, `proj`*), A(`proj`) = {`ID_proj`}, A(`emp`) = {`Ref_proj`}

6. `prof(Eid, Name)` $\subseteq$ `emp(Eid, Name)`: $M(\texttt{prof}) = (\texttt{student}^*)$, $A(\texttt{emp}) = \{\texttt{ID\_emp}\}$, $A(\texttt{prof}) = \{\texttt{Ref\_emp}\}$, $\Sigma = \{\text{Ref\_emp} \stackrel{key}{\to} prof\}$ □

It is worthwhile to point out that there are several places in CoT where human experts can determine better mapping based on the semantics and usages of the underlying data or application.

- The CoT algorithm identifies a minimal set of top-nodes, breaking any ties (when there are cyclic INDs) arbitrarily. A better mapping might have more top-nodes than this minimal set, or might choose to break a tie in a particular manner.
- Given a set of foreign-key constraints on one table, CoT chooses one foreign-key constraint to be represented as a sub-element, and represents the remaining using IDREF attributes. Human experts might be able to provide better input as to which constraint should be represented as sub-element, and which as IDREF attributes.

Examples so far have shown the conversion flow of $\mathbb{X} \to$ CoT $\to$ DTD. We can also have the conversion flow $\mathbb{X} \to$ NeT $\to$ CoT $\to$ DTD (omitted due to space constraint). However this imposes a restriction; when NeT followed by CoT are applied, nesting can be done only on attributes that do not participate in any IND.

5. DISCUSSION

All three algorithms – FT, NeT, and CoT – are "correct" in the sense that they all have preserved the original information of relational schema. We can show that any valid (or invalid) update against the original relational schema is valid (or invalid) against our resulting XML schemas, and these updates can be translated "easily". Our algorithms support schema evolution - addition of new attributes to relations, addition of new INDs etc can be mapped in a straight forward manner to our XML schemas. Further, using the notion of information capacity [16], a theoretical analysis for the correctness of our translation procedures is possible; we can actually show that NeT and CoT algorithms are *equivalence preserving transformations*. However, we defer this detailed analysis to a later version.

With respect to the "goodness" of XML schema that the proposed algorithms generate, it is not obvious to bluntly state whether or not they are good, since there has not been any unanimous normalization theory for XML model yet. Some early work for nested relational model (e.g., [19]) is related, but more recently a few proposals have been made for normal forms of XML model (e.g., [6, 24]). To a greater or lesser extent, the crux of such normal forms is an attempt to reduce data redundancy so that various anomalies can be avoided. Although the output schema from NeT or CoT does not exactly fit into normal forms defined by [6, 24], they share similar properties. For instance, identifying multivalued attributes and making them repeating sub-elements in NeT is essentially a necessary step towards "object class normal form" in [24]. The use of reference attributes in CoT for handling multiple foreign key constraints defined on one table (Section 4.2) can be explained similarly. Therefore, we would like to point out that although it is early to formally prove the goodness of our proposals, it is evident that our proposals lead to less *redundant* yet *correct* XML schema.

In the area of data modeling using XML schemas and for benchmarking, there is a great deal of interest in understanding the characteristics of XML schemas that occur in practical scenarios. Our algorithms show practical XML schemas resulting from translation from relational sources. The characteristics of the XML schemas generated by NeT or CoT algorithms are: (1) they belong to local tree grammar [17] and (2) they are free from any recursion.

6. EXPERIMENTAL RESULTS

6.1 NeT Results

We implemented the NeT and CoT algorithms[5]. For the NeT implementation, we used two additional optimization rules: (1) if $nest_X(t) = t$, then $nest_X(nestAll_L(t)) = nestAll_L(t)$ for any list of columns, L, and (2) if $nest_X(nestAll_L(t)) = nestAll_L(t)$ for any column X and all possible list of columns L of length l, then $nest_X(nestAll_M(t)) = nestAll_M(t)$ for any column X and all possible list of columns M of length m, where $m \geq l$.

Our preliminary results comparing the goodness of the *XSchema* obtained from NeT, and FT with that obtained from DB2XML v 1.3 [23] appeared in [14]. We further applied our NeT algorithm on several test sets drawn from UCI KDD[6] / ML[7] repositories, which contain a multitude of single-table relational schemas and data. Sample results are shown in Table 5. Two metrics are used as follows:

$$\text{NestRatio} = \frac{\#\text{ of successful nesting}}{\#\text{ of total nesting}}$$

$$\text{ValueRatio} = \frac{\#\text{ of original data values}}{\#\text{ of data values D in the nested table}}$$

where D is the number of individual data values present in the table. For example, the D in the row $(\{1, 2, 3\}, a, 10)$ of a nested table is 5. High value for *NestRatio* shows that we did not perform unnecessary nesting and high value for *ValueRatio* shows that the nesting removed a lot of redundancy.

In our experimentation, we observed that most of the attempted nestings are successful. In Table 5, we see that nesting was useful for all the data sets except for the `Bupa` data set. Also nesting was *especially* useful for the `Car` data set, where the size of the nested table is only 6% of the original data set. Time required for nesting is an important parameter, and it jointly depends on the number of attempted nestings and the number of tuples. The number of attempted nestings depends on the number of attributes, and increases drastically as the number of attributes increases. This is observed for the `Flare` data set, where we have to do nesting on 13 attributes.

6.2 CoT Results

For testing CoT, we need some well-designed relational schema where tables are interconnected via inclusion dependencies. For this purpose, we use the TPC-H schema v 1.3.0[8], which is an ad-hoc, decision support benchmark and

[5] Available at http://www.cs.ucla.edu/∼mani/xml
[6] http://kdd.ics.uci.edu/
[7] http://www.ics.uci.edu/∼mlearn/MLRepository.html
[8] http://www.tpc.org/tpch/spec/h130.pdf

Test Set	# of attr. / # of tuple	NestRatio	ValueRatio	Size before / after	# of nested attr.	Time (sec.)
`Balloons1`	5 / 16	42 / 64	80 / 22	0.455 / 0.152	3	1.08
`Balloons2`	5 / 16	42 / 64	80 / 22	0.455 / 0.150	3	1.07
`Balloons3`	5 / 16	40 / 64	80 / 42	0.455 / 0.260	3	1.14
`Balloons4`	5 / 16	42 / 64	80 / 22	0.455 / 0.149	3	1.07
`Hayes`	6 / 132	1 / 6	792 / 522	1.758 / 1.219	1	1.01
`Bupa`	7 / 345	0 / 7	2387 / 2387	7.234 / 7.234	0	4.40
`Balance`	5 / 625	56 / 65	3125 / 1120	6.265 / 2.259	4	21.48
`TA_Eval`	6 / 110	253 / 326	660 / 534	1.559 / 1.281	5	24.83
`Car`	7 / 1728	1870 / 1957	12096 / 779	51.867 / 3.157	6	469.47
`Flare`	13 / 365	11651 / 13345	4745 / 2834	9.533 / 5.715	4	6693.41

Table 5: Summary of NeT experimentations.

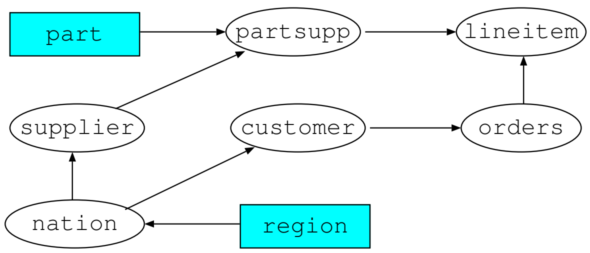

Figure 3: The IND-Graph representation of the TPC-H schema.

has 8 tables and 8 inclusion dependencies. The IND-Graph for the TPC-H schema is shown in Figure 3.

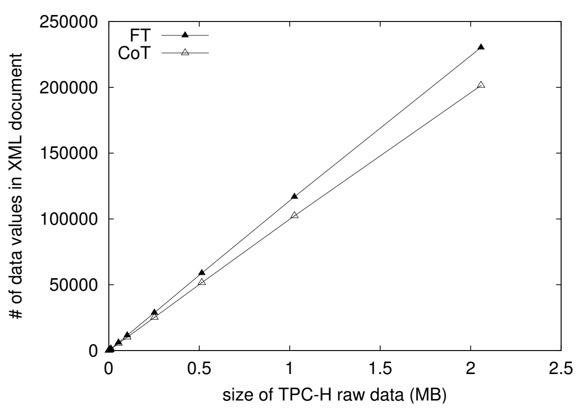

Figure 4: Comparison of XML documents generated by FT and CoT algorithms for TPC-H data.

CoT identifies two top-nodes - `part` and `region`. Suppose we start the scan of the top-nodes from `region`, then, six of the eight inclusion dependencies are mapped using subelement, and the remaining two are mapped using `IDREF` attributes. We believe that the *XSchema* produced by CoT is "more intuitive" than the relational schema we started with.

Figure 4 shows a comparison of the number of data values originally present in the database, and the number of data values in the XML document generated by FT and CoT. Because FT is a flat translation, the number of data values in the XML document generated by FT is the same as the number of data values in the original data. However, CoT is able to decrease the number of data values in the generated XML document by more than 12%.

7. CONCLUSION

We have presented two relational-to-XML conversion algorithms – NeT and CoT. The naive translation algorithm FT translates the "flat" relational model to "flat" XML model in a one-to-one manner. Thus FT does *not* use the non-flat features of the XML model, possible through regular expression operators such as "*" and "+". To remedy this problem, we first presented NeT, which uses the *nest* operator to generate a more precise and intuitive XML Schema from relational inputs. When poorly designed or legacy relational schema needs to be converted to XML format, NeT can suggest a fairly intuitive XML schema. However NeT is only applicable to a single table at a time, and *cannot* obtain a big picture of a relational schema where many tables are interconnected with each other. Our next algorithm CoT addresses this problem; CoT uses *semantic constraints* (especially inclusion dependencies) to come up with a more intuitive XML Schema for the entire relational schema.

Thus our approaches have the following properties: (1) automatically infer a "good" XML Schema from a given relational schema, (2) remove redundancies that might be present in poorly designed or legacy relational schema (3) maintain semantic constraints during translation. With a rapid adoption of XML standards among industries and majority of data still stored in relational databases, the need to correctly and effectively convert relational data into XML format is imminent. We believe our proposed methods are good additions to such a practical problem.

There are several interesting issues to be studied. Implementation issues (e.g., I/O cost, tagging strategy, nesting strategy) are very important. Early investigation on these issues is done in [2]. Since our work in this paper proposes algorithms which can result in a fairly complex target XML schema as an output, studying an efficient implementation of our NeT and CoT algorithms is an important direction. Another direction of future research is studying the normalization theory of XML schema. By formally defining what is a "good" XML schema, one can devise better relational-to-XML conversion algorithms that result in normalized XML schema. Our resulting XML schemas support data updates as well as schema evolution by addition of new attributes or semantic constraints easily. However schema evolution by deletion of attributes or semantic constraints are more difficult in the XML model, and should be studied. Also NeT performed only single attribute nesting. Multiple attribute nesting is another interesting research direction.

8. REFERENCES

[1] P. V. Biron and A. Malhotra (Eds). "XML Schema Part 2: Datatypes". W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-2/.

[2] R. Bourret. "Data Transfer Strategies: Transferring Data between XML Documents and Relational Databases". Web page, 2000. http://www.rpbourret.com/xml/DataTransfer.htm.

[3] T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds). "Extensible Markup Language (XML) 1.0 (2nd Edition)". W3C Recommendation, Oct. 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[4] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, and S. Subramanian. "XPERANTO: Publishing Object-Relational Data as XML". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, May 2000.

[5] J. Clark and M. Murata (Eds). "RELAX NG Tutorial". OASIS Working Draft, Jun. 2001. http://www.oasis-open.org/committees/relax-ng/tutorial.html.

[6] D. W. Embley and W. Y. Mok. "Developing XML Documents with Guranteed "Good" Properties". In *Int'l Conf. on Conceptual Modeling (ER)*, Yokohama, Japan, Nov. 2001.

[7] W. Fan and J. Siméon. "Integrity Constraints for XML". In *ACM PODS*, Dallas, TX, May 2000.

[8] M. F. Fernandez, W.-C. Tan, and D. Suciu. "SilkRoute: Trading between Relations and XML". In *Int'l World Wide Web Conf. (WWW)*, Amsterdam, Netherlands, May 2000.

[9] P. C. Fischer, L. V. Saxton, S. J. Thomas, and D. V. Gucht. "Interactions between Dependencies and Nested Relational Structures". *J. Computer and System Sciences (JCSS)*, 31(3):343–354, Dec. 1985.

[10] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. "XTRACT: A System for Extracting Document Type Descriptors from XML Documents". In *ACM SIGMOD*, Dallas, TX, May 2000.

[11] R. L. Graham, D. E. Knuth, and O. Patashnik. *"Concrete Mathematics: A Foundation for Computer Science"*. Addison-Wesley Pub., 1994.

[12] G. Jaeschke and H.-J. Schek. "Remarks on the Algebra of Non First Normal Form Relations". In *ACM PODS*, Los Angeles, CA, Mar. 1982.

[13] D. Lee and W. W. Chu. "CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema". *J. Data & Knowledge Engineering (DKE)*, 39(1):3–25, Oct. 2001.

[14] D. Lee, M. Mani, F. Chiu, and W. W. Chu. "Nesting-based Relational-to-XML Schema Translation". In *Int'l Workshop on the Web and Databases (WebDB)*, Santa Barbara, CA, May 2001.

[15] M. Mani, D. Lee, and R. D. Muntz. "Semantic Data Modeling using XML Schemas". In *Int'l Conf. on Conceptual Modeling (ER)*, Yokohama, Japan, Nov. 2001.

[16] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. "Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice (Extended Abstract)". In *EDBT*, Cambridge, UK, Mar. 1994.

[17] M. Murata, D. Lee, and M. Mani. "Taxonomy of XML Schema Languages using Formal Language Theory". In *Extreme Markup Languages*, Montreal, Canada, Aug. 2001. http://www.cs.ucla.edu/~dongwon/paper/.

[18] C. Nentwich, W. Emmerich, A. Finkelstein, and A. Zisman. "BOX: Browsing Objects in XML". *Software Practice and Experience*, 30(15):1661–1676, 2000. http://www.cs.ucl.ac.uk/staff/c.nentwich/Box/.

[19] Z. M. Özsoyoglu and L. Y. Yuan. "A New Normal Form for Nested Relations". *ACM Trans. on Database Systems (TODS)*, 12(1):111–136, Mar. 1987.

[20] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. "Relational Databases for Querying XML Documents: Limitations and Opportunities". In *VLDB*, Edinburgh, Scotland, Sep. 1999.

[21] T. Shimura, M. Yoshikawa, and S. Uemura. "Storage and Retrieval of XML Documents using Object-Relational Databases". In *Int'l Conf. on Database and Expert Systems Applications (DEXA)*, pages 206–217, Florence, Italy, Aug. 1999.

[22] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn (Eds). "XML Schema Part 1: Structures". W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-1/.

[23] V. Turau. "Making Legacy Data Accessible for XML Applications". Web page, 1999. http://www.informatik.fh-wiesbaden.de/~turau/veroeff.html.

[24] X. Wu, T. W. Ling, M. L. Lee, and G. Dobbie. "Designing Semistructured Database Using ORA-SS Model". Unpublished Manuscript, 2001.

APPENDIX

A. PROOFS

PROOF. **(Remark 1)** The number of the *first* nesting along n columns is the same as the number of 1-element sequences: n. The number of the *second* nesting along n columns is again the same as the number of 2-elements sequences by P2: $n(n-1)$. Continuing this, the number of the *last* nesting along n columns is again the same as the number of n-elements sequences: $n+n(n-1)+\cdots+n(n-1)\ldots(2)(1) = n^{\underline{1}}+n^{\underline{2}}+\cdots+n^{\underline{n}} = \sum_{k=1}^{n} n^{\underline{k}}$ (q.e.d)

PROOF. **(Lemma 1)** Consider a table t with column set C, and candidate keys, $K_1, K_2, \ldots, K_n \subseteq C$. Consider a column $X \in C$, such that X is not an attribute of at least one of the candidate keys, say $X \notin K_i$. Now $\overline{X} \supseteq K_i$, and hence $\overline{X}$ is unique. Thus, no two tuples can agree on $\overline{X}$. Therefore, by the definition of the *nest* operator, nesting on X will fail. (q.e.d)

PROOF. **(Theorem 1)** The first column to be nested, say X, is chosen such that $X \in K$ by Lemma 1, in one of the m ways. Now after the first nesting, by Corollary 1, we have a new candidate key $\overline{X}$. The next column to be nested is chosen from $K \cap \overline{X}$, where $|K \cap \overline{X}| = m-1$. Thus we have $m-1$ ways of choosing the second column for nesting. Continuing this, we have total number of nesting is $m + m(m-1)+\ldots+m(m-1)\ldots(2)(1) = \sum_{k=1}^{m} m^{\underline{k}}$. (q.e.d)