Effective Schema Conversions between XML and Relational Models

Dongwon Lee
UCLA / CSD
dongwon@cs.ucla.edu

Murali Mani
UCLA / CSD
mani@cs.ucla.edu

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

Abstract

As Extensible Markup Language (XML) is emerging as the data format of the Internet era, there is an increasing need to efficiently store and query XML data. At the same time, as requirements change, we expect a substantial amount of conventional *relational* data to be converted or published as XML data. One path to accommodate these changes is to transform XML data into relational format (and vice versa) to use the mature relational database technology.

In this paper, we present three semantics-based schema transformation algorithms towards this goal: 1) CPI converts an XML schema to a relational schema while preserving semantic constraints of the original XML schema, 2) NeT derives a nested structured XML schema from a flat relational schema by repeatedly applying the *nest* operator so that the resulting XML schema becomes hierarchical, and 3) CoT takes a relational schema as input, where multiple tables are interconnected through inclusion dependencies and generates an equivalent XML schema as output.

1 Introduction

Recently, XML [4] has emerged as the *de facto* standard for data format on the web. The use of XML as the common format for representing, exchanging, storing, and accessing data poses many new challenges to database systems. Since the majority of everyday data is still stored and maintained in relational database systems, we expect that the needs to convert data format between XML and relational models will grow substantially. To this end, several schema transformation algorithms have been proposed (e.g., [7, 9, 16, 5]). Although they work well for the given applications, the XML-to-Relational or Relational-to-XML transformation algorithms only capture the *structure* of the original schema and largely ignore the hidden *semantic constraints*. Consider the following example for XML-to-Relational conversion case.

Example 1. Consider a DTD that models conference publications:

```
<!ELEMENT conf  (title,society,year,mon?,paper+)>
<!ELEMENT paper (pid,title,abstract?)>
```

Suppose the combination of `title` and `year` uniquely identifies the `conf`. Using the hybrid inlining algorithm [16], the DTD would be transformed to the following relational schema:

```
conf  (title,society,year,mon)
paper (pid,title,conf_title,conf_year,abstract)
```

While the relational schema correctly captures the structural aspect of the DTD, it does not enforce correct semantics. For instance, it cannot prevent a tuple t_1: `paper(100,'DTD...','ER',3000,'...')` from being inserted. However, tuple t_1 is inconsistent with the semantics of the given DTD since the DTD implies that the paper cannot exist without being associated with a conference and there is apparently no conference "ER-3000" yet. In database terms, this kind of violation can be easily prevented by an *inclusion dependency* saying "`paper[conf_title,conf_year]` $\subseteq$ `conf[title,year]`". □

The reason for this inconsistency between the DTD and the transformed relational schema is that most of the proposed transformation algorithms, so far, have largely ignored the hidden *semantic constraints* of the original schema.

1.1 Related Work

Between XML and Non-relational Models: Conversion between different models has been extensively investigated. For instance, [6] deals with transformation problems in OODB area; since OODB is a richer environment than RDB, their work is not readily applicable to our application. The logical database design methods and their associated transformation techniques to other data models have been extensively studied in ER research.

For instance, [1] presents an overview of such techniques. However, due to the differences between ER and XML models, those transformation techniques need to be modified substantially. More recently, [2] studies a generic mapping between arbitrary models with the focus of developing a framework for model management, but is not directly relevant to our problems.

From XML to Relational: From XML to relational schema, several conversion algorithms have been proposed recently. STORED [7] is one of the first significant attempts to store XML data in relational databases. STORED uses a data mining technique to find a representative DTD whose support exceeds the pre-defined threshold and using the DTD, converts XML documents to relational format. Because [3] discusses template language-based transformation from DTD to relational schema, it requires human experts to write an XML-based transformation rule. [16] presents three inlining algorithms that focus on the table level of the schema conversions. On the contrary, [9] studies different performance issues among eight algorithms that focus on the attribute and value level of the schema. Unlike these, we propose a method where the hidden semantic constraints in DTDs are systematically found and translated into relational formats [12]. Since the method is orthogonal to the structure-oriented conversion method, it can be used along with algorithms in [7, 3, 16, 9].

From Relational to XML: There have been different approaches for the conversion from relational model to XML model, such as XML Extender from IBM, XML-DBMS, SilkRoute [8], and XPERANTO [5]. All the above tools require the user to specify the mapping from the given relational schema to XML schema. 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.

2 Overview of Our Schema Translation Algorithms

In this paper, we present three schema transformation algorithms that not only capture the structure, but also the semantics of the original schema. The overview of our proposals is illustrated in Figure 1.

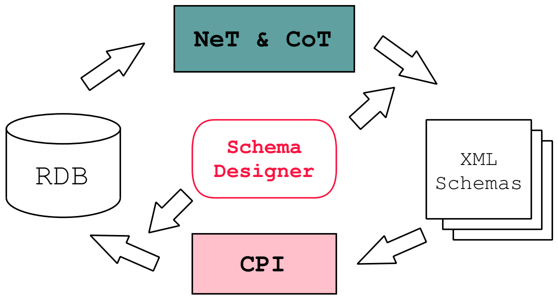

Figure 1: Overview of our schema translation algorithms.

CPI (Constraints-preserving Inlining Algorithm): identifies various semantics constraints in the original XML schema and preserves them by rewriting them in the final relational schema.

NeT (Nesting-based Translation Algorithm): derives a nested structure from a flat relational schema by repeatedly applying the *nest* operator so that the resulting XML schema becomes hierarchical. The main idea is to find a more intuitive element content model of the XML schema that utilizes the regular expression operators provided by the XML schema specification (e.g., “*” or “+”).

CoT (Constraints-based Translation Algorithm): 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 the overall picture of relational schema where multiple tables are interconnected. To remedy this problem, CoT considers inclusion dependencies during the translation, and merges multiple interconnected tables into a coherent and hierarchical parent-child structure in the final XML schema.

3 The CPI Algorithm

Transforming a hierarchical XML model to a flat relational model is not a trivial task due to several inherent difficulties such as non-trivial 1-to-1 mapping, existence of set values, complicated recursion, and/or fragmentation issues [16]. Most XML-to-Relational transformation algorithms (e.g., [3, 7, 9, 16]) have so far mainly focused on the issue of structural conversion, largely ignoring the semantics already existed in the original XML schema. Let us first describe various semantic constraints that one can mine from the DTD. Throughout the discussion, we will use the example DTD and XML document in Tables 1 and 2.

3.1 Semantic Constraints in DTDs

Cardinality Constraints: In a DTD declaration, there are only 4 possible cardinality relationships between an element and its sub-elements as illustrated below:

```
<!ELEMENT article (title, author+, ref*, price?)>
```

```
<!ELEMENT conf    (title,date,editor?,paper*)>
<!ATTLIST conf    id     ID          #REQUIRED>
<!ELEMENT title   (#PCDATA)>
<!ELEMENT date    EMPTY>
<!ATTLIST date    year   CDATA       #REQUIRED
                  mon    CDATA       #REQUIRED
                  day    CDATA       #IMPLIED>
<!ELEMENT editor  (person*)>
<!ATTLIST editor  eids   IDREFS      #IMPLIED>
<!ELEMENT paper   (title,contact?,author,cite?)>
<!ATTLIST paper   id     ID          #REQUIRED>
<!ELEMENT contact EMPTY>
<!ATTLIST contact aid    IDREF       #REQUIRED>
<!ELEMENT author  (person+)>
<!ATTLIST author  id     ID          #REQUIRED>
<!ELEMENT person  (name,(email|phone)?)>
<!ATTLIST person  id     ID          #REQUIRED>
<!ELEMENT name    EMPTY>
<!ATTLIST name    fn     CDATA       #IMPLIED
                  ln     CDATA       #REQUIRED>
<!ELEMENT email   (#PCDATA)>
<!ELEMENT phone   (#PCDATA)>
<!ELEMENT cite    (paper*)>
<!ATTLIST cite    id     ID          #REQUIRED
                  format (ACM|IEEE)  #IMPLIED>
```

Table 1: A DTD for `Conference`.

```
<conf id="er05">
  <title>Int'l Conference on Conceptual Modeling (ER)</title>
  <date>
    <year>2005</year> <mon>May</mon> <day>20</day>
  </date>
  <editor eids="sheth bossy">
    <person id="klavans">
      <name fn="Judith" ln="Klavans"/>
      <email>klavans@cs.columbia.edu</email>
    </person>  </editor>
  <paper id="p1">
    <title>Indexing Model for Structured...</title>
    <contact aid="dao"/>
    <author>
      <person id="dao"><name fn="Tuong" ln="Dao"/></person>
    </author>
  </paper>
  <paper id="p2">
    <title>Logical Information Modeling of...</title>
    <contact aid="shah"/>
    <author>
      <person id="shah">
        <name fn="Kshitij" ln="Shah"/>
      </person>
      <person id="sheth">
        <name fn="Amit" ln="Sheth"/>
        <email>amit@cs.uga.edu</email>
      </person>
    </author>
    <cite id="c100" format="ACM">
      <paper id="p3">
        <title>Making Sense of Scientific...</title>
        <author>
          <person id="bossy">
            <name fn="Marcia" ln="Bossy"/>
            <phone>391.4337</phone>
          </person>
        </author> </paper> </cite> </paper>
</conf>
<paper id="p7">
  <title>Constraints-preserving Transformation from...</title>
  <contact aid="lee"/>
  <author>
    <person id="lee">
      <name fn="Dongwon" ln="Lee"/>
      <email>dongwon@cs.ucla.edu</email>
    </person> </author>
  <cite id="c200" format="IEEE"/>
</paper>
...
```

Table 2: An example XML document conforming to the DTD in Table 1.

1. (0,1): An element can have either zero or one sub-element. (e.g., sub-element `price`)
2. (1,1): An element must have one and only one sub-element. (e.g., sub-element `title`)
3. (0,N): An element can have zero or more sub-elements. (e.g., sub-element `ref`)
4. (1,N): An element can have one or more sub-elements. (e.g., sub-element `author`)

Following the notations in [1], let us call each cardinality relationship as type (0,1), (1,1), (0,N), (1,N), respectively. From these cardinality relationships, mainly three constraints can be inferred. First is whether or not the sub-element can be null. We use the notation "$X \nrightarrow \emptyset$" to denote that an element X cannot be null. This constraint is easily enforced by the `NULL` or `NOT NULL` clause in SQL. Second is whether or not more than one sub-element can occur. This is also known as *singleton constraint* in [17] and is one kind of equality-generating dependencies. Third, given an element, whether or not its sub-element should occur. This is one kind of tuple-generating dependencies. The second and third types will be further discussed below.

Inclusion Dependencies (INDs): An *Inclusion Dependency* assures that values in the columns of one fragment must also appear as values in the columns of other fragments and is a generalization of the notion of *referential integrity*.

Trivial form of INDs found in the DTD is that "given an element X and its sub-element Y, Y must be included in X (i.e., $Y \subseteq X$)". For instance, from the `conf` element and its four sub-elements in the `Conference` DTD, the following INDs can be found as long as `conf` is not null: {`conf.title` $\subseteq$ `conf`, `conf.date` $\subseteq$ `conf`, `conf.editor` $\subseteq$ `conf`, `conf.paper` $\subseteq$ `conf`}. Another form of INDs can be found in the attribute definition part of the DTD with the use of the `IDREF(S)` keyword. For instance, consider the `contact` and `editor` elements in the `Conference` DTD shown below:

```
<!ELEMENT person  (name,(email|phone)?>
<!ATTLIST person  id   ID     #REQUIRED>
<!ELEMENT contact EMPTY>
<!ATTLIST contact aid  IDREF  #REQUIRED>
<!ELEMENT editor  (person*)>
<!ATTLIST editor  eids IDREFS #IMPLIED>
```

The DTD restricts the `aid` attribute of the `contact` element such that it can only point to the `id` attribute of the `person` element[1]. Further, the `eids` attribute can only point to multiple `id` attributes of the `person` element. As a result, the following INDs can be derived: {`editor.eids` $\subseteq$ `person.id`, `contact.aid` $\subseteq$

[1]Precisely, an attribute with `IDREF` type does not specify which element it should point to. This information is available only by human experts. However, new XML schema languages such as XML-Schema and DSD can express where the reference actually points to [11].

`person.id`}. Such INDs can be best enforced by the "foreign key" if the attribute being referenced is a primary key. Otherwise, it needs to use the `CHECK`, `ASSERTION`, or `TRIGGERS` facility of SQL.

Equality-Generating Dependencies (EGDs): The *Singleton Constraint* [17] restricts an element to have "at most" one sub-element. When an element type X satisfies the singleton constraint towards its sub-element type Y, if an element instance x of type X has *two* sub-elements instances y_1 and y_2 of type Y, then y_1 and y_2 must be the same. This property is known as *Equality-Generating Dependencies (EGDs)* and denoted by "$X \rightarrow Y$" in database theory. For instance, two EGDs: {`conf` → `conf.title`, `conf` → `conf.date`} can be derived from the `conf` element in Table 1. This kind of EGDs can be enforced by SQL `UNIQUE` construct. In general, EGDs occur in the case of the (0,1) and (1,1) mappings in the cardinality constraints.

Tuple-Generating Dependencies (TGDs): TGDs in a relational model require that some tuples of a certain form be present in the table and use the "$\twoheadrightarrow$" symbol. Two useful forms of TGDs from DTD are the *child* and *parent constraints* [17].

1. **Child constraint:** `"Parent` ↠ `Child"` states that every element of type *Parent* must have at least one child element of type *Child*. This is the case of the (1,1) and (1,N) mappings in the cardinality constraints. For instance, from the DTD in Table 1, because the `conf` element must contain the `title` and `date` sub-elements, the child constraint `conf` ↠ {`title, date`} holds.

2. **Parent constraint:** `"Child` ↠ `Parent"` states that every element of type *Child* must have a parent element of type *Parent*. According to XML specification, XML documents can start from any level of element without necessarily specifying its parent element, when a root element is not specified by `<!DOCTYPE root>`. In the DTD in Table 1, for instance, the `editor` and `date` elements can have the `conf` element as their parent. Further, if we know that all XML documents were started at the `conf` element level, rather than the `editor` or `date` level, then the parent constraint {`editor, date`} ↠ `conf` holds. Note that the `title` ↠ `conf` does not hold since the `title` element can be a sub-element of either the `conf` or `paper` element.

3.2 Discovering and Preserving Semantic Constraints from DTDs

The CPI algorithm utilizes a structure-based conversion algorithm as a basis and identifies various semantic constraints described in Section 3.1. We will use the *hybrid* algorithm [16] as the basis algorithm. CPI first constructs a *DTD graph* that represents the structure of a given DTD. A DTD graph can be constructed when parsing the given DTD. Its nodes are elements, attributes, or operators in the DTD. Each element appears exactly once in the graph, while attributes and operators appear as many times as they appear in the DTD. CPI then annotates various cardinality relationships (summarized in Table 3) among nodes to each edge of the DTD graph. Note that the cardinality relationship types in the graph consider not only element vs. sub-element relationships but also element vs. attribute relationships. Figure 2 illustrates an example of such annotated DTD graph for the `Conference` DTD in Table 1.

Relationship	Symbol	not null	EGDs	TGDs
(0,1)	?	no	yes	no
(1,1)		yes	yes	yes
(0,N)	*	no	no	no
(1,N)	+	yes	no	yes

Table 3: Cardinality relationships and their corresponding semantic constraints.

Once the annotated DTD graph is constructed, CPI follows the basic navigation method provided by the *hybrid* algorithm; it identifies **top nodes** [16, 12] that are the nodes: 1) not reachable from any nodes (e.g., source node), 2) direct child of "*" or "+" operator node, 3) recursive node with indegree > 1, or 4) one node between two mutually recursive nodes with indegree $= 1$. Then, starting from each top node T, *inline* all the elements and attributes at *leaf nodes* reachable from T unless they are other top nodes. In doing so, each annotated cardinality relationship can be properly converted to its counterpart in SQL syntax as described in Section 3.1. The details of the algorithm is beyond the scope of this paper and interested readers are referred to [12]. For instance, Figure 3 and Table 4 are such output relational schema and data in SQL notation, automatically generated by the CPI algorithm.

4 The NeT Algorithm

The simplest Relational-to-XML translation method, termed as FT (Flat Translation) in [13], is to translate 1) tables in a relational schema to elements in an XML schema and 2) columns in a relational schema to attributes in an XML schema. FT is a simple and effective translation algorithm. However, since FT translates the "flat" relational model to a "flat" XML model in a one-to-one manner, it does not utilize several basic "non-flat"

paper								
id	root_elm	parent_elm	fk_conf	fk_cite	title	contact_aid	cite_id	cite_format
p1	conf	conf	er05	–	Indexing ...	dao	–	–
p2	conf	conf	er05	–	Logical ...	shah	c100	ACM
p3	conf	cite	–	c100	Making ...	–	–	–
p7	paper	–	–	–	Constraints ...	lee	c200	IEEE

Table 4: Final relational "data" for the `paper` element in the `Conference` DTD in Table 1, generated by CPI algorithm.

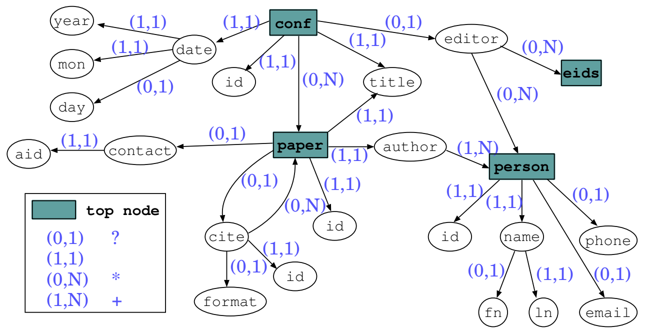

Figure 2: An annotated *DTD graph* for the `Conference` DTD in Table 1.

```
CREATE TABLE paper (
  id          NUMBER       NOT NULL,
  title       VARCHAR(50)  NOT NULL,
  contact_aid VARCHAR(20),
  cite_id     VARCHAR(20),
  cite_format VARCHAR(50)
    CHECK (VALUE IN ("ACM", "IEEE")),
  root_elm    VARCHAR(20)  NOT NULL,
  parent_elm  VARCHAR(20),
  fk_cite     VARCHAR(20)
    CHECK (fk_cite IN (SELECT cite_id FROM paper)),
  fk_conf     VARCHAR(20),
  PRIMARY KEY (id),
  UNIQUE (cite_id),
  FOREIGN KEY (fk_conf)      REFERENCES conf(id),
  FOREIGN KEY (contact_aid)  REFERENCES person(id)
);
```

Figure 3: Final relational "schema" for the `paper` element in the `Conference` DTD in Table 1, generated by CPI algorithm.

features provided by the XML model for data modeling such as representing *repeating sub-elements* through regular expression operators (e.g., "*", "+"). To remedy the shortcomings of FT, we propose the NeT algorithm that utilizes various *element content models* of the XML model. NeT uses the *nest* operator [10] to derive a "good" element content model.

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[2]. Formally,

Definition 1 (Nest) *[10]. 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**.

Example 2. Consider a table R in Table 5. 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. For a detailed description on the various properties of the *nest* operator, the interested are referred to [13, 14].

Lemma 1. *For a table t with n columns, and m ($\leq n$) columns that participate in all candidate keys, the max-*

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

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

imum number of nestings is $(m) + (m)(m-1) + ... + (m)(m-1)...(2)(1)$. ■

Lemma 1 implies that when candidate key information is available, one can avoid unnecessary nestings substantially. For instance, suppose attributes A and C in Table 5 constitute a key for R. Then, one needs to compute only: $nest_A(R)$ at (b), $nest_C(R)$ at (d), $nest_C(nest_A(R))$ at (e), $nest_A(nest_C(R))$ at (f) in Table 5.

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 the nested table where the most number of nestings succeeded as the final schema, since this is a schema which provides low "data redundancy". The outline of the NeT algorithm is as follows:

1. For each table t_i in the input relational schema $\mathbb{R}$, apply the *nest* operator repeatedly until no nesting succeeds.
2. Choose the best nested table based on the selected criteria. Denote this table as $t'_i(c_1, \ldots, c_{k-1}, c_k, \ldots, c_n)$, where nesting succeeded on the columns $\{c_1, \ldots, c_{k-1}\}$.
   (a) If $k = 1$, follow the FT translation.
   (b) If $k > 1$,
      i. For each column c_i $(1 \leq i \leq k-1)$, if c_i was nullable in $\mathbb{R}$, use c_i^* for the element content model, and c_i^+ otherwise.
      ii. For each column c_j $(k \leq j \leq n)$, if c_i was nullable in $\mathbb{R}$, use $c_j^?$ for the element content model, and c_j otherwise.

5 The CoT Algorithm

The NeT algorithm 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 at a time, and cannot obtain a *overall picture* of the relational schema where many tables are interconnected with each other through various other dependencies. To remedy this problem, we propose the CoT algorithm that uses Inclusion Dependencies (INDs) of relational schema. 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.

The basic idea of the CoT is the following: 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, 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, and 2) α is nullable/non-nullable. Then, the translation of two tables s, t with a foreign key constraint works as follows:

1. If α is non-nullable (i.e., none of the columns of α can take null values), then:
   (a) If α is unique, then there is a $1:1$ relationship between s and t, and can be captured as `<!ELEMENT t (Y, s?)>`.
   (b) If α is not-unique, then there is a $1:n$ relationship between s and t, and can be captured as `<!ELEMENT t (Y, s*)>`.

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

Table 6: An example schema with associated INDs.

2. 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.

Extending this to the general case where multiple tables are interconnected via INDs, consider the schema with a set of tables $\{t_1, ..., t_n\}$ and INDs $t_i[\alpha_i] \subseteq t_j[\beta_j]$, where $i, j \leq n$. We consider only those INDs that are foreign key constraints (i.e., β_j constitutes the primary key of the table t_j), and where α_i is non-nullable. The relationships among tables can be captured by a graph representation, termed as IND-Graph.

Definition 2 (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 $V_i \in V$, and for each distinct IND $t_i \subseteq t_j$, there exists an edge $E_{ji} \in E$ from the node V_j to V_i.* □

Note the edge direction is reversed from the IND direction for convenience. Given a set of INDs, the IND-Graph can be easily constructed. Once an IND-Graph G is constructed, `CoT` needs to decide the starting point to apply translation rules. For that purpose, we use the notion of **top nodes**. Intuitively, an element is a top node if it *cannot* be represented as a sub-element of any other element. Let T denote the set of top nodes. Then, `CoT` traverses G, using say Breadth-First Search (BFS), until it traverses all the nodes and edges, while capturing the INDs on edges as either sub-elements (when the node is visited for the first time) or IDREF attributes (when the node was visited already).

Example 3. Consider a schema and its associated INDs in Table 6. The IND-Graph with two top nodes is shown in Figure 4: 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`$[\alpha] \subseteq t[\beta]$. Then,

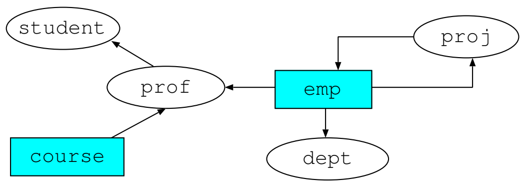

Figure 4: The IND-Graph representation of the schema in Table 6 (*top nodes* denoted by rectangular nodes).

- First, starting from a top node `course`, do BFS scan. Pull up a reachable node `prof` into `course` and make it as sub-element by `<!ELEMENT course (Cid, Title, Room, prof*)>`. Similarly, the node `student` is also pulled up into its parent node `prof` by `<!ELEMENT prof (Eid, Name, student*)>`. Since the node `student` is a leaf, no nodes can be pulled in: `<!ELEMENT student (Sid, Name)>`. Since there is no more unvisited reachable node from `course`, the scan stops.

- Next, starting from another top node `emp`, pull up neighboring node `dept` into `emp` similarly by `<!ELEMENT emp (Eid, Name, ProjName, dept*)>` and `<!ELEMENT dept (Dno, Mgr)>`. Then, visit a neighboring node `prof`, but `prof` was visited already. To avoid data redundancy, an attribute `Ref_prof` is added to `emp` accordingly. Since attributes in the left-hand side of the corresponding IND, $prof(Eid, Name) \subseteq emp(Eid, Name)$, form a super key, the attribute `Ref_prof` is assigned type `IDREF`, and not `IDREFS`: `<!ATTLIST prof Eid ID>` and `<!ATTLIST emp Ref_prof IDREF>`.

- Next, visit a node `proj` and pull it up to `emp` by `<!ELEMENT emp (Eid, Name, ProjName, dept*, proj*)>` and `<!ELEMENT proj (Pname)>`. In next step, visit a node `emp` from `prof`, but since it was already visited, an attribute `Ref_emp` of type `IDREFS` is added to `proj`, and scan stops.

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

6 Conclusion

We have presented a method to transform a relational schema to an XML schema, and two methods to transform an XML schema to a relational schema, both in *structural* and *semantic* aspects. All three algorithms are "correct" in the sense that they all have preserved the original information of relational schema. For instance, using the notion of information capacity [15], a theoretical analysis for the correctness of our translation procedures is possible; we can actually show that CPI, NeT and CoT algorithms are *equivalence preserving transformations*.

CPI was tested against DTDs gathered from OASIS[3]. For all cases, CPI successfully identified hidden semantic constraints from DTDs and correctly preserved them by rewriting them in SQL. We also applied the NeT algorithm on 10 test sets drawn from UCI KDD[4] / ML[5] repositories, which contain a multitude of single-table relational schemas and data. Except the one case, NeT successfully found nested attributes and thus generated a hierarchical XML schema. When CoT was tested against TPC-H schema[6] which is an ad-hoc, decision support benchmark and has 8 tables and 8 inclusion dependencies, it successfully derived a "better" XML schema with less redundant data (about 12% decrease compared to FT). Further details can be found in [12, 14].

Despite the difficulties in conversions between XML and relational models, there are many practical benefits. We strongly believe that devising more accurate and efficient conversion methodologies between XML and relational models is important. The prototypes of our algorithms are available at: `http://www.cobase.cs.ucla.edu/projects/xpress/`

References

[1] C. Batini, S. Ceri, and S. B. Navathe. *"Conceptual Database Design: An Entity-Relationship Approach"*. The Benjamin/Cummings Pub., 1992.

[2] P. Bernstein, A. Halevy, and R. Pottinger. "A Vision of Management of Complex Models ". *ACM SIGMOD Record*, 29(3):55–63, Dec. 2000.

[3] R. Bourret. "XML and Databases". Web page, Sep. 1999. http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[4] 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.

[5] 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.

[6] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. "From Structured Document to Novel Query Facilities". In *ACM SIGMOD*, Minneapolis, MN, Jun. 1994.

[7] A. Deutsch, M. F. Fernandez, and D. Suciu. "Storing Semistructured Data with STORED". In *ACM SIGMOD*, Philadephia, PA, Jun. 1998.

[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] D. Florescu and D. Kossmann. "Storing and Querying XML Data Using an RDBMS". *IEEE Data Eng. Bulletin*, 22(3):27–34, Sep. 1999.

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

[11] D. Lee and W. W. Chu. "Comparative Analysis of Six XML Schema Languages". *ACM SIGMOD Record*, 29(3):76–87, Sep. 2000.

[12] 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.

[13] 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.

[14] D. Lee, M. Mani, F. Chiu, and W. W. Chu. *"NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints"*. UCLA CS Technical Report, Feb. 2002.

[15] 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.

[16] 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.

[17] P. T. Wood. "Optimizing Web Queries Using Document Type Definitions". In *Int'l Workshop on Web Information and Data Management (WIDM)*, pages 28–32, Kansas City, MO, Nov. 1999.

[3]http://www.oasis-open.org/cover/xml.html
[4]http://kdd.ics.uci.edu/
[5]http://www.ics.uci.edu/∼mlearn/MLRepository.html
[6]http://www.tpc.org/tpch/spec/h130.pdf