CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema ⋆

Dongwon Lee, Wesley W. Chu

University of California at Los Angeles, Department of Computer Science, Los Angeles, CA 90095, USA

Abstract

As Extensible Markup Language (XML) is emerging as *the* data format of the Internet era, there are increasing needs to efficiently store and query XML data. One path to this goal is transforming XML data into relational format in order to use relational database technology. Although several transformation algorithms exist, they are incomplete in the sense that they focus only on *structural* aspects and ignore *semantic* aspects. In this paper, we present the semantic knowledge that needs to be captured during transformation to ensure a correct relational schema. Further, we show an algorithm that can 1) derive such semantic knowledge from a given XML Document Type Definition (DTD) and 2) preserve the knowledge by representing it as *semantic constraints* in relational database terms. By combining existing transformation algorithms and our *constraints-preserving* algorithm, one can transform XML DTD to relational schema where correct semantics and behaviors are guaranteed by the preserved constraints. Experimental results are also presented.

Key words: XML, DTD, Schema Transformation, Relational Schema, Constraints

1 Introduction

As the World-Wide Web becomes a major means of disseminating and sharing information, Extensible Markup Language (XML) (Bray et al., 2000) is emerging as a possible candidate data format because it is simpler than SGML, and

⋆ This research is supported in part by DARPA contract No. N66001-97-C-8601.

Email addresses: dongwon@cs.ucla.edu (Dongwon Lee), wwc@cs.ucla.edu (Wesley W. Chu).

more powerful than HTML. One way to query XML data is to reuse the established relational database techniques by converting and storing XML data in relational storage. Since the hierarchical XML and the flat relational data models are not fully compatible, the transformation is not a straightforward task.

To this end, several XML-to-relational transformation algorithms have been proposed (Deutsch et al., 1998; Florescu and Kossmann, 1999; Shanmugasundaram et al., 1999). For instance, Shanmugasundaram et al. (1999) presents 3 algorithms that focus on the table level of the schema while Florescu and Kossmann (1999) studies different performance issues among 8 algorithms that focus on the attribute and value level of the schema. They all transform the given XML Document Type Definition (DTD) to relational schema. Similarly, Deutsch et al. (1998) presents a data mining-based algorithm that instead uses XML documents directly without a DTD.

Although they work well for the given applications, they miss one important point. That is, the transformation algorithms only capture the *structure* of a DTD and ignore the hidden *semantic* constraints. Consider the following example.

Example 1. Consider a DTD modeling 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 (explained in Section 3), 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 for the DTD, it does not force 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 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 transformation algorithms only capture the *structure* of the DTD and ignore the hidden *semantic* constraints. Via our *constraints-preserving inlining (CPI)* algorithm, we show the kinds of semantic constraints

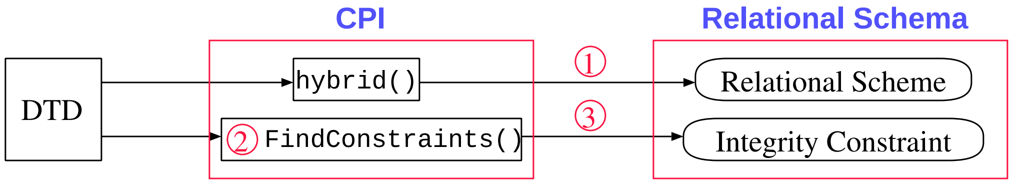

Fig. 1. Overview of our approach. Numbers 1) to 3) specify: 1) transforming schema, 2) discovering constraints via `findConstraints()`, and 3) preserving constraints via `rewriteConstraints()`.

that can be derived from DTDs during transformation, and illustrate how to preserve them by rewriting them in an output schema notation. Since our algorithm to capture and preserve semantic constraints from DTDs is independent of the transformation algorithms, our algorithm can be applied to various transformation processes such as Deutsch et al. (1998); Florescu and Kossmann (1999); Shanmugasundaram et al. (1999) with little change. Figure 1 presents an overview of our approach. First, given a DTD, we transform it to a corresponding relational scheme using an existing algorithm. Second, during the transformation, we discover various semantic constraints in XML notation. Third, we rewrite the discovered constraints to conform to relational notation.

This paper is organized as follows. Section 2 gives background information and related work. In Section 3, the transformation algorithm is discussed in detail. Section 4 presents various semantic constraints that are hidden in DTDs. Section 5 proposes our algorithm to preserve such constraints during transformation. Section 6 reports some experimental results and Section 7 illustrates two example applications where the discovered semantic constraints are further utilized. Finally, Sections 8 and 9 discuss our vision on future work and concluding remarks.

2 Background and Related Work

Relational Schema: We define a relational schema $\mathbb{R}$ to be composed of a *relational scheme* ($\mathbb{S}$) and *semantic constraints* (Δ). That is, $\mathbb{R} = (\mathbb{S}, \Delta)$. In turn, the relational scheme $\mathbb{S}$ is a collection of table schemes such as $r(a_1, ..., a_k)$, where a_i is the i-th attribute in the table r and the semantic constraints Δ is a collection of semantic knowledge such as domain constraints, inclusion dependency, equality-generating dependency, tuple-generating dependency, etc.

XML and DTD: XML is a textual representation of the hierarchical data model defined by the World-Wide Web Consortium (W3C) (Bray et al., 2000). The meaningful piece of the XML document is bounded by matching start-

Table 1
A DTD for `Conference`.

```
<!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>
```

ing and ending *tags* such as `<name>` and `</name>`. In XML, tags are defined by users, while in HTML, permitted tags are pre-defined. Thus, XML is a meta-language that can be used for defining other customized languages. Using DTDs, users can define the structure of the XML document of particular interest. Conceptually, a DTD in XML is very similar to a schema in a relational database. The main building blocks of DTD are *elements* and *attributes*, which are defined by the keywords `<!ELEMENT>` and `<!ATTLIST>`, respectively. In general, components in DTD are specified by the following BNF syntax:

```
<!ELEMENT> <element-name> <element-type>
<!ATTLIST> <attr-name> <attr-type> <attr-option>
```

For a detailed description of a DTD model, refer to Lee and Chu (2000a). Table 1 shows a DTD for `Conference` which states that a `conf` element can have four sub-elements: `title`, `date`, `editor` and `paper` in that order. As common in regular expressions, 0 or 1 occurrence (i.e., *optional*) is represented by the symbol "?", 0 or more occurrences is represented by the symbol "*", and 1 or more occurrences is represented by the symbol "+". A sub-element without any such symbols (e.g., `title`) represents a *mandatory* one.

Keywords `#PCDATA` and `CDATA` are used as *string* types for elements and attributes, respectively. For instance, the type of `title` element is defined as `#PCDATA` so that `title` element can be arbitrary character data. `<attr-option>` can be `#REQUIRED` or `#IMPLIED` among others. An attribute with a `#REQUIRED` option is a *mandatory* one, while an attribute with an `#IMPLIED` option is an *optional* one. `<attr-type>` keywords `ID` and `IDREF` are used for the pointed

Table 2
A valid XML document conforming to the DTD for `Conference` of Table 1.

```
<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>
```

and pointing attributes, respectively. `IDREFS` is a plural form of `IDREF`. For instance, the `author` element must have a mandatory `id` attribute and this attribute is used when other attributes point to this attribute. On the other hand, the `contact` element has a mandatory `aid` attribute that must point to the `id` attribute of the contacting `author` of the current paper. One interesting definition in Table 1 is the `cite` element; it can have zero or more `paper` elements as sub-elements, thus creating a cyclic definition. Table 2 shows a valid XML document conforming to the DTD for `Conference`. The document represents a portion of the fictional ER conference to be held in 2005. The first two `paper` elements are described with `id="p1"` and `id="p2"`, respectively. The `paper` element with `id="p2"` further has a `cite` element that describes the references in the paper. The `paper` element with `id="p7"` shows an example of the valid XML document that is *not* rooted at `conf` element. Note that when a root element is not specified in a DTD (i.e., no `<!DOCTYPE root>` clause is given), a valid XML document can be rooted at any level of the DTD hierarchy as long as their sub-elements and attributes are valid.

XML-Schema: XML-Schema[1] (Fallside (Eds), 2001; Thompson et al., 2001; Biron and Malhotra (Eds), 2001) is an ongoing effort of W3C as a next generation XML schema language. XML-Schema aims to be more expressive than DTD and more usable by a wider variety of applications. It has many novel mechanisms such as inheritance for attributes and elements, user-defined datatypes, more expressive constraints, etc.

XML-Schema supports some features that cannot be easily captured in relational schema. For instance, in XML-Schema, one can define arbitrary combinations of elements and/or attributes as a key. The following snippet defines a key `ekey` for an element `student` that consists of an attribute `Sname` and two sub-elements `Advisor` and `Course`.

```
<key name="ekey">
 <selector xpath="//student"/>
 <field xpath="@Sname"/><field xpath="Advisor"/><field xpath="Course"/>
</key>
```

If a `student` is allowed to have multiple `Advisors` and to take multiple `Courses`, typical XML-to-relational conversion algorithms would store `Advisor` and `Course` in separate tables t_2 and t_3 to avoid violating 1NF, while storing remaining attributes and sub-elements of `student` in a table t_1. Thus, when `Sname`, `Advisor`, and `Course` are all stored in separate tables t_1, t_2, and t_3, respectively, it is not clear how to preserve the key constraint `ekey`.

There are more intriguing features of XML-Schema that one cannot easily capture in relational schema such as user-defined types, namespace, `<any>` types, etc. For the rest of this paper, we restrict ourselves to the case of DTDs only and leave the support for XML-Schema as a future work.

Assumptions: Without loss of generality, to simplify our presentation, we assume that: 1) the input DTD has been already simplified using a technique in Shanmugasundaram et al. (1999), 2) the input XML documents are all *valid*, and 3) the physical XML features such as *entities* or *notations* are not discussed but handled in the implementation.

2.1 Related Work

Conversion between different models has been extensively investigated. For instance, Christophides et al. (1994) deals with transformation problems in the OODB area; since OODB is a richer environment than RDB, their work is

[1] We differentiate two terms in this paper – XML schema and XML-Schema. The former refers to a general term for schema in the XML model, while the latter refers to one particular kind of XML schema language proposed by W3C.

Table 3
Classification of schema conversion between XML and relational models.

Conversion Methods	Structure-oriented	Constraints-oriented
XML to Relational	Deutsch et al. (1998); Shanmugasundaram et al. (1999) Florescu and Kossmann (1999); Bourret (1999) Kappel et al. (2000); Schmidt et al. (2000) Klettke and Meyer (2000); Cheng and Xu (2000), etc	Lee and Chu (2000b)
Relational to XML	Turau (1999); Fernandez et al. (2000) Shanmugasundaram et al. (2000); Banerjee et al. (2000) Carey et al. (2000), etc	Lee et al. (2001)

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, Batini et al. (1992) 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, Bernstein et al. (2000) studies a generic mapping between arbitrary models with the focus of developing a framework for model management. Apart from conversion approaches, it is worthwhile to note that there have been also recent investigations on native XML storage systems such as Kanne and Moerkotte (2000).

Towards conversion between XML and relational models, an array of research has addressed the particular issues lately. On the commercial side, database vendors are busily extending their databases to adopt XML types. Typically, they can handle XML data using BLOB/CLOB formats along with a limited keyword searching or using some object-relational features (Cheng and Xu, 2000; Banerjee et al., 2000), but not many details have been revealed. On the research side, Table 3 shows the classification of such related work.

- **Structure-oriented XML to Relational conversion**: Work done in STORED (Deutsch et al., 1998) is one of the first significant and concrete attempts to this end and deals with non-valid XML documents. STORED uses a data mining technique to find a representative DTD whose support exceeds the pre-defined threshold and convert XML documents to relational format using the DTD. Bourret (1999) discusses template language-based transformation from DTD to relational schema which requires human experts to write an XML-based transformation rule. Shanmugasundaram et al. (1999) presents three inlining algorithms that focus on the table level of the schema conversions. On the contrary, Florescu and Kossmann (1999) studies different performance issues among eight algorithms that focus on the attribute and value level of the schema. Shimura et al. (1999) proposes a DTD-independent mapping algorithm. While ignoring specific characteristics hidden in each DTD, Shimura et al. (1999) decomposes XML documents into element, attribute, text and path tables, so that the changes of DTDs of the XML documents do not necessarily result in invalid mapping as found in examples (Deutsch et al., 1998; Shanmugasundaram et al., 1999). Since our CPI algorithm provides a systematic way of finding and preserving constraints from a DTD, ours is an improvement to the existing transformation algorithms. Recent work in Kappel et al. (2000) attempts a conversion approach based on the notion of meta schema between XML and relational models, but mainly focuses on the structural mapping unlike ours.

- **Constraints-oriented XML to Relational conversion**: Lee and Chu (2000b) proposes a method where the hidden semantic constraints in DTD are systematically found and translated into relational formats. Since the method is orthogonal to the structure-oriented conversion methods, it can be used along with algorithms (Deutsch et al., 1998; Bourret, 1999; Shanmugasundaram et al., 1999; Florescu and Kossmann, 1999) with little change. We are not aware of any other work on this problem. This paper is an extended work of Lee and Chu (2000b).

- **Structure-oriented Relational to XML conversion**: Some primitive work has been done in Turau (1999) dealing with the transformation from relational tables to XML documents. SilkRoute (Fernandez et al., 2000) provides a declarative query language (RXL) for viewing relational data in XML. Applications express the answer data as a query over the view and SilkRoute dynamically materializes the fragment of an XML view. Shanmugasundaram et al. (2000) extends SQL to specify the conversion process declaratively, whereas SilkRoute proposes a new language RXL and describes an extensive study on the issues of efficiently implementing the algorithms. Similar to SilkRoute, XPERANTO (Carey et al., 2000) aims to provide a uniform XML interface to underlying ORDB, transparently providing an XML-to-SQL query rewriter and a table-to-XML answer converter. Its output XML view is, however, mainly specified by the user's input XML queries.

  In addition, there have been other DTD inference algorithms that take as "input" a set of XML documents (Garofalakis et al., 2000) or a view description (Papakonstantinou and Velikhov, 1999).

- **Constraints-oriented Relational to XML conversion**: *Path constraints* on a semi-structured model (Buneman et al., 1998) or XML model (Fan and Siméon, 2000) have been studied, mostly with respect to their implication problems. However, to our best knowledge, there has not been much work on this direction of conversion problem. For instance, Fan and Siméon (2000) proposes three languages to capture the semantics of XML model and presents implication results, but does not deal with issues on converting constraints from RDB to XML model. Recently, the authors proposed to convert relational schema to XML schema using the hidden data semantics found by the *nest* operator in Lee et al. (2001).

3 Transforming DTD to Relational Schema

Transforming a hierarchical XML model to a flat relational model is not a trivial task. There are several difficulties including non 1-to-1 mapping, set values, recursion, and fragmentation issues (Shanmugasundaram et al., 1999). For a better presentation, we chose one particular transformation algorithm, called the *hybrid inlining algorithm* (Shanmugasundaram et al., 1999) among many algorithms (Bourret, 1999; Deutsch et al., 1998; Florescu and Kossmann, 1999; Shanmugasundaram et al., 1999). It is chosen since it exhibits the pros of the other two competing algorithms in Shanmugasundaram et al. (1999) without severe side effects and it is a more generic algorithm than those in Bourret (1999); Deutsch et al. (1998). Since issues of discovering and preserving semantic constraints in this paper is independent of that of transformation algorithms, our technique can be applied to other transformation algorithms easily.

3.1 Choice Elimination Algorithm

Before describing the *hybrid* algorithm, let us first discuss an algorithm that eliminates the choice operators (|) from the content models of a DTD while trying to maintain the same semantics. Shanmugasundaram et al. (1999) does not provide any details on this subtle but important issue and simply assumes that such pre-processing has been already done.

The choice operators are heavily used in XML model, but are not natively supported in relational model. For instance <!ELEMENT r (a|b)> in XML model implies that "r can have either a or b but not both at the same time". Translating this to relational model, the closest mapping with the same semantics would be having a table "r" with two nullable columns "a" and "b", (i.e., (a?,b?)) with a constraint enforcing one of the two columns must be null at all times as follows:

```
CREATE TABLE r (
 a     VARCHAR(20),
 b     VARCHAR(20),
 CHECK ((a is NOT NULL AND b is NULL) OR (a is NULL AND b is NOT NULL))
);
```

Hence, when there is no nested content models, any arbitrary long content models with | operators <!ELEMENT r (a_1 | ... | a_n)> can be treated as if it were <!ELEMENT r (a_1?, ..., a_n?)> with an additional constraint like CHECK ((a_1 is NOT NULL AND a_2 is NULL AND ... a_n is NULL) OR ... OR (a_1 is NULL AND a_2 is NULL AND ... a_n is NOT NULL)). Let us call this

Algorithm 1. `migrateChoice`
Input : Regular expression r
Output: Regular expression $(r_1|r_2|\ldots|r_n)$ equivalent to r

switch r **do**
  case r does not contain "|" operator
    return r;
  case $r = (r_1)^*$
    migrateChoice(r_1) $= (a_1|a_2|\ldots|a_n)$;
    return $(a_1^*, a_2^*, \ldots, a_n^*)^*$;
  case $r = (r_1|r_2)$
    migrateChoice(r_1) $= (a_1|a_2|\ldots|a_n)$;
    migrateChoice(r_2) $= (b_1|b_2|\ldots|b_n)$;
    return $(a_1|a_2|\ldots|a_n|b_1|b_2|\ldots|b_n)$;
  case $r = (r_1, r_2)$
    migrateChoice(r_1) $= (a_1|a_2|\ldots|a_n)$;
    migrateChoice(r_2) $= (b_1|b_2|\ldots|b_n)$;
    return $((a_1, b_1)|(a_1, b_2)|\ldots|(a_1, b_n)|(a_2, b_1)|(a_2, b_2)|\ldots|(a_2, b_n)|$
    $\ldots|(a_n, b_1)|(a_n, b_2)|(a_n, b_n))$;

mapping heuristics as `convertChoice()`.

Now consider a general case where a content model can in turn contain further nested content models in it and all use | operators in a complex manner. From a basic regular expression algebraic law (Hopcroft et al. (2001), page 118), the following equality holds: `(a | b)* = (a*, b*)*`. Using the law, the shown Algorithm `migrateChoice()` determines an equivalent regular expression of the form $(r_1|r_2|\ldots|r_n)$, where no r_i $(1 \leq i \leq n)$ contains | operator (i.e., remove | in inner groups except ones in the outermost group)

Once we have a content model returned from `migrateChoice()`, then all | operators have migrated from inside to outside. Next step is to flatten content models out. For instance, Shanmugasundaram et al. (1999) describes various heuristics such as `a*? = a*` or `(a*,a*) = (a*)`. Let us call such steps as `flatten()`.

As a conclusion, content models of DTDs using the choice operator can be in general converted to relational schema by going through 1) migrateChoice(r) for each content model r and 2) successively convertChoice(r) and 3) flatten(r). For further details of the algorithm, refer to Mani et al. (2001).

Example 2. Consider `<!ELEMENT r ((a|b)*|c)>`. First, migrateChoice(r) is converted to migrateChoice($(a|b)^*|c)$) and is in turn converted to two calls: migrateChoice($(a|b)^*$) and migrateChoice(c). Further, migrateChoice($(a|b)^*$)

returns migrateChoice($(a^*, b^*)^*$) while migrateChoice(c) remains intact. Hence, eventually migrateChoice(r) returns migrateChoice($(a^*, b^*)^*|c$) At second stage, $(a^*, b^*)^*|c$ is simulated by $((a^*, b^*)^*?, c?)$ by convertChoice() and in turn simplified to $((a^*, b^*)^*, c?)$ by flatten(), generating <!ELEMENT r (a*,b*,c?)> with a proper constraint at the end. The new content model is free of the choice operator and can be fed into the hybrid algorithm in the next section. □

3.2 Hybrid Inlining Algorithm

The *hybrid* algorithm (Shanmugasundaram et al., 1999) essentially does the following[2]:

(1) Create 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.
(2) Identify *top nodes* in a DTD graph. A top node satisfies any of the following conditions: 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 a top node T, *inline* all the elements and attributes at *leaf nodes* reachable from T unless they are other top nodes.
(3) Attribute names are composed from the concatenated path from the top node to the leaf node using "_" as a delimiter. Use an attribute with `ID` type as a key if provided. Otherwise, add a system-generated integer key[3].
(4) If a table corresponds to the shared element with indegree > 1 in the DTD, then add a field `parent_elm` to denote the parent element to which the current tuple belongs. Further, for each shared element, a new field `fk_$X$` is added as a *foreign key* to record the key values of parent element X. If X is inlined into another element Y, then record the Y's key value in the `fk_$Y$` field.
(5) Inlining an element Y into a table r, corresponding to another element X (i.e., top node), creates a problem when an XML document is rooted

[2] We have made a few changes to the hybrid algorithm for a better presentation (e.g., renaming, supporting "|" operator), but the crux of the algorithm remains intact.
[3] In practice, even if there is an attribute with `ID` type, one may decide to have a system-generated key for better performance.

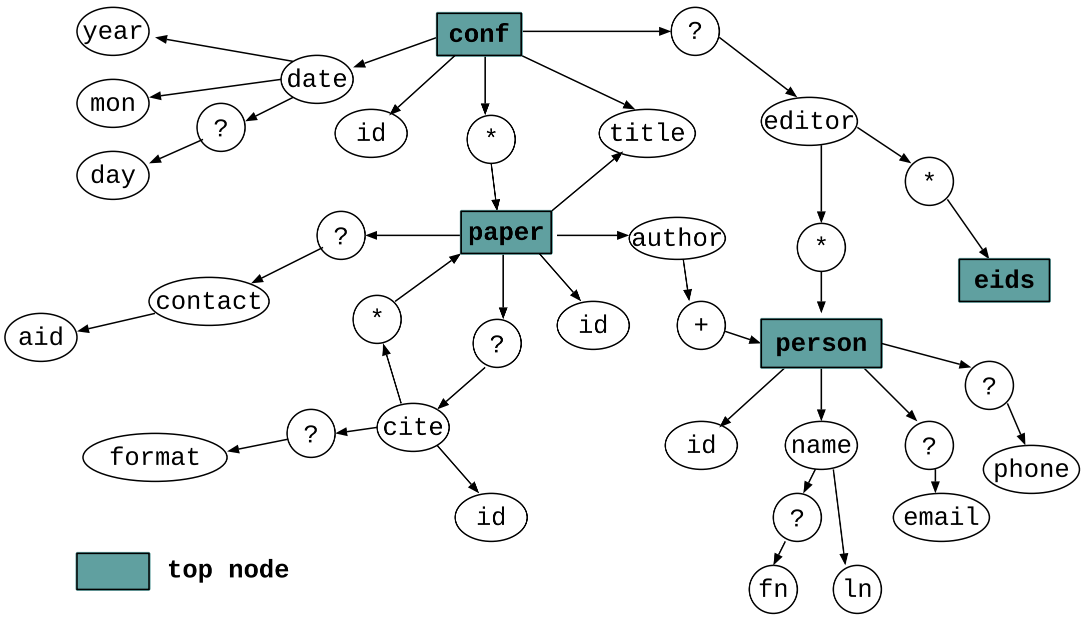

Fig. 2. A DTD graph for the DTD in Table 1.

at the element Y. To facilitate queries on such elements, a new field `root_elm` is added to a table r.

(6) If an *ordered* DTD model is used, a field `ordinal` is added to record position information of sub-elements in the element. (For simplification, the `ordinal` field is not shown in this paper.)

For further details of the algorithm, refer to Shanmugasundaram et al. (1999). Figure 2 illustrates a DTD graph that is created from the DTD of Table 1. Table 4 shows the output of the transformation by the hybrid algorithm.

Among eleven elements in the DTD of Table 1, four elements – `conf`, `paper`, `person`, and `eids` – are top nodes and thus, chosen to be mapped to the different tables. For the top node `conf`, the elements `date`, `title`, and `editor` are reachable and thus inlined. Then, the `id` attribute is used as a key and the `root_elm` field is added. For the top node `paper`, the elements `title`, `contact_aid`, `author`, `cite_format` and `cite_id` are reachable and inlined. Since the `paper` element is shared by the `conf` and `cite` elements (two incoming edges in a DTD graph), new fields `parent_elm`, `fk_conf` and `fk_cite` are added to record who and where the parent node was. Note that in the `paper` table (Table 4), a tuple with `id="p7"` has the value `"paper"` for the `root_elm` field. This is because the element `<paper id="p7">` is rooted in the DTD (Table 2) without being embedded in other elements. Consequently, its `parent_elm`, `fk_conf` and `fk_cite` fields are null. For the top node `person`, the elements `name_fn`, `name_ln` and `email` are reachable and inlined. Since the `person` is shared by the `author` and `editor` elements, again, the `parent_elm` is added. Note that in the `person` table (Table 4), a tuple with `id="klavans"`

Table 4
A relational scheme ($\mathbb{S}$) along with the associated data that are converted from the DTD of Table 1 and XML document of Table 2 by the hybrid algorithm. Note that the hybrid algorithm does not generate *semantic constraints* (Δ).

conf

id	root_elm	title	date_year	date_mon	date_day
er05	conf	ER	2005	May	20

conf_editor_eids

id	root_elm	fk_conf	eids
100001	conf	er05	sheth
100002	conf	er05	bossy

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

person

id	root_elm	parent_elm	fk_conf	fk_paper	name_fn	name_ln	email	phone
klavans	conf	editor	er05	–	Judith	Klavans	klavans...	–
dao	conf	paper	–	p1	Tuong	Dao	–	–
shah	conf	paper	–	p2	Kshitij	Shah	–	–
sheth	conf	paper	–	p2	Amit	Sheth	amit@cs...	–
bossy	conf	paper	–	p3	Marcia	Bossy	–	391.4337
lee	paper	paper	–	p7	Dongwon	Lee	dongwon...	–

has the value `"editor"`, not `"paper"`, for the `parent_elm` field. This implies that "klavans" is in fact an `editor`, not an `author` of the paper.

4 Semantic Constraints in DTDs

4.1 Domain Constraints

When the domain of the attributes is restricted to a certain specified set of values, it is called *Domain Constraints*. For instance, in the following DTD, the domain of the attributes `gender` and `married` are restricted.

```
<!ATTLIST author gender  (male|female) #REQUIRED
                 married (yes|no)      #IMPLIED>
```

In transforming such DTD into relational schema, we can enforce the domain constraints using SQL CHECK clause as follows:

```
CREATE DOMAIN gender VARCHAR(10) CHECK (VALUE IN ("male", "female"))
CREATE DOMAIN married VARCHAR(10) CHECK (VALUE IN ("yes", "no"))
```

When the mandatory attribute is defined by the #REQUIRED keyword in the DTD, it needs to be forced in the transformed relational schema as well. That is, the attribute ln cannot be omitted below.

```
<!ELEMENT person  EMPTY>
<!ATTLIST person  fn CDATA  #IMPLIED   ln CDATA  #REQUIRED>
```

We use the notation "$X \nrightarrow \emptyset$" to denote that an attribute X cannot be null. This kind of domain constraint can be best expressed by using the NOT NULL clause in SQL as follows:

```
CREATE TABLE person (fn VARCHAR(20), ln VARCHAR(20) NOT NULL)
```

4.2 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+, reference*, price?)>
```

(0,1). ("at most" semantics): An element can have either zero or one sub-element. (e.g., sub-element `price`)
(1,1). ("only" semantics): An element must have one and only one sub-element. (e.g., sub-element `title`)
(0,N). ("any" semantics): An element can have zero or more sub-elements. (e.g., sub-element `reference`)
(1,N). ("at least" semantics): An element can have one or more sub-elements. (e.g., sub-element `author`)

Following the notations in Batini et al. (1992), 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, whether or not the sub-element can be null. Similar to the attribute case, 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. Second, whether or not more than one sub-element can occur. This is also known as *singleton constraint* in Wood (1999) 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.

4.3 Inclusion Dependencies (IDs)

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 IDs 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 DTD, the following IDs 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 IDs 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 DTD in Table 1 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[4]. Further, the `eids` attribute can only point to multiple `id` attributes of the `person` element. As a result, the following IDs can be derived: {`editor.eids` $\subseteq$ `person.id`, `contact.aid` $\subseteq$ `person.id` }. IDs can be best enforced by the "foreign key" concept if the attribute being referenced is a primary key. Otherwise, it needs to use the `CHECK`, `ASSERTION`, or `TRIGGERS` facility of SQL.

4.4 Equality-Generating Dependencies (EGDs)

The *Singleton Constraint* (Wood, 1999) 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` $\rightarrow$ `conf.title`, `conf` $\rightarrow$ `conf.date`} can be derived from the `conf` element

[4] 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-Schama and DSD can express where the reference actually points to (Lee and Chu, 2000a).

of 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.

4.5 Tuple-Generating Dependencies (TGDs)

Tuple-Generating Dependencies (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* (Wood, 1999).

(1) **Child constraint:** "`Parent` $\twoheadrightarrow$ `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` $\twoheadrightarrow$ {`title`, `date`} holds.
(2) **Parent constraint:** "`Child` $\twoheadrightarrow$ `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 elements without necessarily specifying its parent element, when a root element is not specified by `<!DOCTYPE root>`. In the DTD of 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`} $\twoheadrightarrow$ `conf` holds. Note that the `title` $\twoheadrightarrow$ `conf` does not hold since the `title` element can be a sub-element of either the `conf` or `paper` element.

5 Discovering and Preserving Semantic Constraints

To help find semantic constraints, we use the following data structure:

Definition 1. An **annotated DTD graph (ADG)** $\mathbb{G}$ is a pair ($\mathbb{V}$, $\mathbb{E}$), where $\mathbb{V}$ is a finite set and $\mathbb{E}$ is a binary relation on $\mathbb{V}$. The set $\mathbb{V}$ consists of element and attributes in a DTD. Each edge $e \in \mathbb{E}$ is labeled with the cardinality relationship types as defined in Section 4.2. In addition, each vertex $v \in \mathbb{V}$ carries the following information:

(1) `indegree` stores the number of incoming edges.

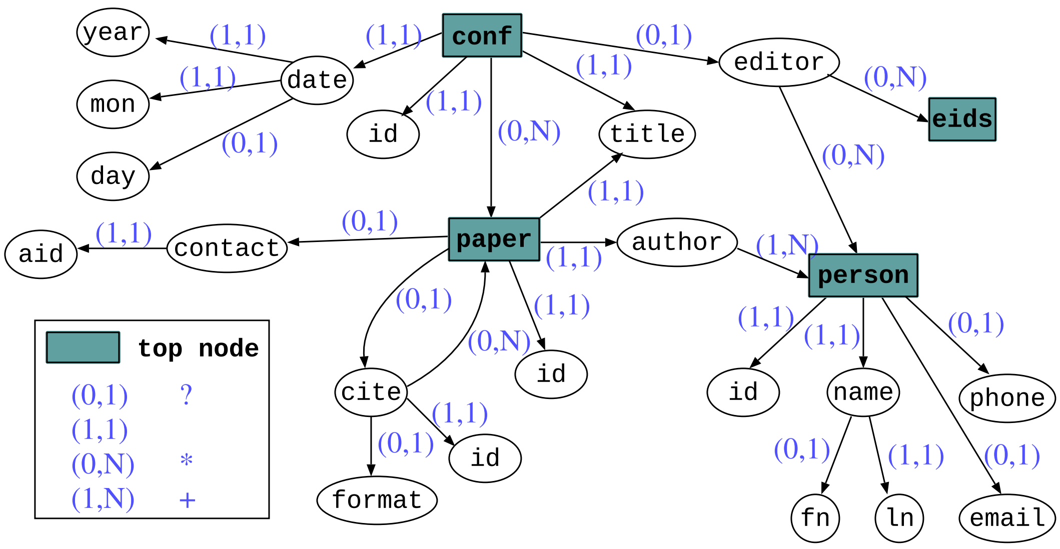

Fig. 3. An *Annotated DTD graph* for the `Conference` DTD of Table 1. The associated values of the nodes (i.e., indegree, type, tag, and status) are not shown.

Table 5
Cardinality relationships and their corresponding semantic constraints.

Relationship	Symbol	Semantics	not null	EGDs	TGDs
(0,1)	?	at most	no	yes	no
(1,1)		only	yes	yes	yes
(0,N)	*	any	no	no	no
(1,N)	+	at least	yes	no	yes

(2) `type` contains the element type name in the content model of the DTD (e.g., `conf` or `paper`).
(3) `tag` stores a flag value whether the node is an element or attribute (if attribute, it contains the attribute keyword like `ID` or `IDREF`, etc.).
(4) `status` contains "visited" flag if the node was visited in a depth-first search or "not-visited". ■

Note that the cardinality relationship types in ADG considers not only element vs. sub-element relationships but also element vs. attribute relationships. For instance, from the DTD `<!ATTLIST X Y #IMPLIED Z #REQUIRED>`, two types of cardinality relationships (i.e., type (0,1) between element X and attribute Y, and type (1,1) between element X and attribute Z) can be derived. Figure 3 illustrates an example of ADG for the `Conference` DTD of Table 1. Then, the cardinality relationships can be used to find semantic constraints in a *systematic* fashion. Table 5 summarizes 3 main semantic constraints that can be derived from and the `findConstraints()` algorithm below is immediately derived from Table 5.

Algorithm 2. `findConstraints`
Input : Node v and w

switch `edge`(v,w) **do**
- **case** *type (0,1)*
  - $v \rightarrow w$
- **case** *type (1,1)*
  - w is not null; $v \rightarrow w$; $v \twoheadrightarrow w$
- **case** *type (0,N)*
  - `/* empty */`
- **case** *type (1,N)*
  - w is not null; $v \twoheadrightarrow w$

Semantic constraints discovered by `findConstraints()` have additional usage as we further discuss in Section 7. However, to enforce correct semantics in the newly generated relational schema, the semantic constraints in XML terms need to be rewritten in relational terms. This is done by the algorithm `rewriteConstraints()`.

5.1 CPI: Constraints-preserving Inlining Algorithm

We shall now describe our complete DTD-to-relational schema transformation algorithm: *CPI (Constraints-preserving Inlining) algorithm* is a combination of the hybrid inlining, `findConstraints()` and `rewriteConstraints()` algorithms. The CPI algorithm is illustrated in `CPI()` and `hybrid()`.

The algorithm first identifies all the top nodes from the ADG. This can be done using algorithms to find sources or strongly-connected components in a graph (Shanmugasundaram et al., 1999). Then, for each top node, the algorithm generates a corresponding table scheme using `hybrid()`. The associated constraints are found and rewritten in relational terms using `findConstraints()` and `rewriteConstraints()`, respectively. The `hybrid()` algorithm scans an ADG in a depth-first search while finding constraints. The final output schema is the union of all the table schemes and semantic constraints.

Table 6 contains the semantic constraints that are rewritten from XML terms to relational terms. As an example, the CPI algorithm will eventually spit out the following SQL `CREATE` statement for the `paper` table. Note that not only is the relational scheme provided, but the semantic constraints are also ensured by use of the `NOT NULL`, `KEY`, `UNIQUE` or `CHECK` constructs.

Algorithm 3. `rewriteConstraints`
Input : Constraints Δ' in XML notation
Output: Constraints Δ in relational notation

switch Δ' **do**

- **case** $X \nrightarrow \emptyset$
  - If X is mapped to attribute X' in table scheme A, then $A[X']$ cannot be null. (i.e., "`CREATE TABLE` A (...X' `NOT NULL`...)")
- **case** $X \subseteq Y$
  - If X and Y are mapped to attributes X' and Y' in table scheme A and B, respectively, then rewrite it as $A[X'] \subseteq B[Y']$. (i.e., If Y' is a primary key of B, then "`CREATE TABLE` A (...`FOREIGN KEY` (X') `REFERENCES` $B(Y')$...)". Else "`CREATE TABLE` A (...(X') `CHECK` (X' IN (SELECT Y' FROM B))...")
- **case** $X \rightarrow X.Y$
  - If element X and Y are mapped to the same table scheme A (i.e., since Y is not a top node, Y becomes an attribute of table A) and Z is the key attribute of A, then rewrite it as $A[Z] \rightarrow A[Y]$. (i.e., "`CREATE TABLE` A (...`UNIQUE` (Y), `PRIMARY KEY` (Z)...)")
- **case** $X \twoheadrightarrow X.Y$
  - **if** (element X and Y are mapped to the same table) **then**
    - Let A be the table and Z be the key attribute of A. Then rewrite it as $A[Z] \twoheadrightarrow A[Y]$. (i.e., "`CREATE TABLE` A (...Y `NOT NULL`, `PRIMARY KEY` (Z)...)")
  - **else**
    - Let the tables be A and B, respectively and Z be the key attribute of A. Then rewrite it as $B[fk\_A] \subseteq A[Z]$. (i.e., "`CREATE TABLE` B (...`FOREIGN KEY` (fk_A) `REFERENCES` $A(Z)$...)")

return Δ

```
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)
);
```

Algorithm 4. `CPI`
Input : Annotated DTD Graph $\mathbb{G} = (V, E)$
Output: Relational Schema $\mathbb{R}$

$\mathbb{V} \leftarrow$ `topnode`$(\mathbb{G})$
for *each* $v \in \mathbb{V}$ **do**
  $table\_def \leftarrow \{\}$
  if $v.tag =$ *'element'* **then**
    `add`('root_elm', $table\_def$); `/* start where? */`
  if $v.indegree > 1$ **then**
    `add`('parent_elm', $table\_def$); `/* shared elements case */`
    `add`(concat('fk_', parent(v)), $table\_def$)
  $\mathbb{W} \leftarrow Adj[v]$; w $\in \mathbb{W}$
  if *any* $w.tag =$ `'ID'` **then** `add`($w.type$, $table\_def$);
  else `add`('id', $table\_def$); `/* system-generated primary key */`
  $\mathbb{R} \leftarrow \mathbb{R} +$ `hybrid`(v, $table\_def$, $\emptyset$)
return $\mathbb{R}$

Algorithm 5. `hybrid`
Input : Vertex v, TableDef $table\_def$, string $attr\_name$
Output: Relational Schema $\mathbb{R}$

$v.status \leftarrow$ 'visited'
for *each* $w \in Adj[v]$ **do**
  if $w.status =$ *'not-visited'* **then**
    $\Delta' \leftarrow$ `findConstraints`(v, w); $\Delta \leftarrow$ `rewriteConstraints`(Δ')
    `hybrid`(w, $table\_def$, `concat`($attr\_name$, '_', $w.type$))
`add`($attr\_name$, $table\_def$); $\mathbb{R} \leftarrow table\_def + \Delta$
return $\mathbb{R}$

6 Experimental Results

We have implemented the CPI algorithm in Java using the IBM XML4J package. Table 7 shows a summary of our experimentation. We gathered test DTDs from "http://www.oasis-open.org/cover/xml.html" and Sahuguet (2000). Since some DTDs had syntactic errors caught by the XML4J, we had to modify them manually. Note that people seldom used the `ID` and `IDREF(S)` constructs in their DTDs except the `XMI` and `BSML` cases. The number of tables generated in the relational schema was usually smaller than that of elements/attributes in DTDs due to the inlining effect. The only exception to this phenomenon was the `XMI` case, where extensive use of types (0,N) and (1,N) cardinality relationships resulted in many top nodes in the ADG.

The number of semantic constraints had a close relationship with the design

Table 6
The semantic constraints in relational notation for the `Conference` DTD of Table 1.

Type	Semantic constraints in relational notation
ID	conf_editor_eids[eids] $\subseteq$ person[id], paper[contact_aid] $\subseteq$ person[id]
EGD	conf[id] $\rightarrow$ conf[title,date_year,date_mon,date_day] paper[id] $\rightarrow$ conf[title,contact_aid,cite_id,cite_format] person[id] $\rightarrow$ conf[name_fn,name_ln,email]
TGD	conf[id] $\twoheadrightarrow$ conf[title,date_year,date_mon,date_day] paper[id] $\twoheadrightarrow$ conf[title,contact_aid,cite_id,cite_format] person[id] $\twoheadrightarrow$ conf[name_fn,name_ln,email] conf_editor_eids[fk_conf] $\subseteq$ conf[id] paper[fk_conf] $\subseteq$ conf[id], paper[fk_cite] $\subseteq$ paper[cite_id] person[fk_conf] $\subseteq$ conf[id], person[fk_paper] $\subseteq$ paper[id]
not null	conf[id,title,date_year,date_mon,root_elm] $\nrightarrow \emptyset$ conf_editor_eids[id,root_elm] $\nrightarrow \emptyset$ paper[id,title,root_elm] $\nrightarrow \emptyset$, person[id,name_ln,root_elm] $\nrightarrow \emptyset$

Table 7
Experimental results of the CPI algorithm.

DTD Semantics		DTD Schema		Relational Schema			
Name	Domain	Elm/Attr	ID/IDREF(S)	Table/Attr	$\rightarrow$	$\twoheadrightarrow$	$\nrightarrow \emptyset$
`novel`	literature	10/1	1/0	5/13	6	9	9
`play`	Shakespeare	21/0	0/0	14/46	17	30	30
`tstmt`	religious text	28/0	0/0	17/52	17	22	22
`vCard`	business card	23/1	0/0	8/19	18	13	13
`ICE`	content synd.	47/157	0/0	27/283	43	60	60
`MusicML`	music desc.	12/17	0/0	8/34	9	12	12
`OSD`	s/w desc.	16/15	0/0	15/37	2	2	2
`PML`	web portal	46/293	0/0	41/355	29	36	36
`Xbel`	bookmark	9/13	3/1	9/36	9	1	1
`XMI`	metadata	94/633	31/102	129/3013	10	7	7
`BSML`	DNA seq.	112/2495	84/97	104/2685	99	33	33

of the DTD hierarchy and the type of cardinality relationship used in the DTD. For instance, the `XMI` DTD had many type (0,N) cardinality relationships, which do not contribute to the semantic constraints. As a result, the number of semantic constraints at the end was small, compared to that of elements/attributes in the DTD. This was also true for the `OSD` case. On the other hand, in the `ICE` case, since it used many type (1,1) cardinality relationships, it resulted in many semantic constraints.

7 Application of the Semantic Constraints

The constraints that are discovered during the transformation are useful to ensure correct semantics of the resulting relational schema. Additionally, they can be used as *semantic knowledge* in a variety of areas (Abiteboul et al., 1999; Böhm et al., 1998; Lee and Chu, 1999; Wood, 1999). Since the focus of this paper is not on the application of the constraints, in this section, we will only illustrate a few motivating examples for the possible applications.

7.1 Semantic Query Optimization

The most common use of the constraints occurs in semantic query optimization where a user's query is typically rewritten using constraints to a simpler form to minimize the processing cost of the query. For instance, consider the following query Q_1: "Find titles of the paper that has at least an author with a non-null last name". In XQL (Robie et al., 1998) notation, this query can be written against `Conference` DTD of Table 1 as follows:

```
XQL:    /paper[author/person/name/ln]/title
```

The [ ] notation in XQL is called the *filter expression*. That is, the given query Q_1 finds all `paper` elements that have at least one sub-element `author` x, such that x has a sub-element `person` y as a child, such that y has a sub-element `name` z as a child, such that z has a sub-element `ln` as a child. When Q_1 is translated to SQL based on the relational schema of Table 4, it will be as follows:

```
SQL:    SELECT  P.title
        FROM    paper P, person Q
        WHERE   P.id = Q.fk_paper        AND
                Q.parent_elm = 'paper'   AND
                Q.name_ln is NOT NULL
```

Note that a filter expression in XQL had to be translated to a join expression between the `paper` and `person` tables in SQL. However, if we had used the semantic constraints in the query formation stage, we could have first created the following XQL query:

```
XQL:    /paper/title
```

Intuitively, this makes sense since the filter expression in Q_1 is satisfied by all the `paper` elements. That is, according to the DTD, all papers must have at least one `author` sub-element (paper $\twoheadrightarrow$ paper.author), `author` must have at least one `person` sub-element (author $\twoheadrightarrow$ author.person), `person` must have one `name` sub-element (person $\twoheadrightarrow$ person.name), and `name` must have one `ln` attribute (name $\twoheadrightarrow$ name.ln). Therefore, the filter expression is redundant and does not have to enforced in the translated SQL, resulting in the following SQL at the end:

```
SQL:    SELECT  title
        FROM    paper
```

7.2 Semantic Caching

In a client and server architecture, client caching is commonly used to speed up query response time and to prepare for unexpected network partition. When such a client caching uses the user's query description as a key value to local cache, it is called *semantic caching*. To maximize the usage of such a client caching, we recently proposed a technique called *query matching* in Lee and Chu (1999). In the query matching technique, a user's query is examined to determine if it can be answered from any of the locally stored answers with the help of semantic knowledge to avoid unnecessary access to the server. If so, it is beneficial since the user's query does not have to be shipped to the server side to get answers.

Suppose a client cache stores the following query Q_1 that selects `person` elements that are directly or indirectly related to ER `conf` element:

```
XQL:    /conf[title='ER']/*/person
```

This query can be translated to the following SQL query based on the relational schema of Table 4:

```
SQL:    SELECT P2.id, P2.name_fn, P2.name_ln, P2.name_email
        FROM   conf C, conf_editor_eids C2, paper P, person P2
        WHERE  C.title = 'ER'       AND
               (C.id = C2.fk_conf   AND  C2.eids = P2.id)
        OR     (C.id = P.fk_conf    AND  P.id = P2.fk_paper)
```

Now the user asks the second query Q_2 that selects the editor's names of the ER `conf` element:

```
XQL:    /conf[title='ER']/editor/person/name
```

Then, Q_2 does not have to be shipped to the server to find answers since $Q_2 \subseteq Q_1$. This is intuitively true since in both XQL queries Q_1 and Q_2, `person.name` $\subseteq$ `person` and `editor` $\subseteq$ `*`. Therefore, given the cached answer A_1 to the query Q_1, answers A_2 to the query Q_2 can be obtained by computing "$A_2 = A_1 \wedge Q_2$" on the client side, which is more efficient than sending Q_2 to and receiving answers from the server.

8 Future Work

Due to many benefits from using relational databases as storage systems for XML data, the need for efficient and effective conversion between relational and XML models will significantly grow in a foreseeable future. We believe the following directions of research are very important.

First, as we move to more expressive next generation XML schema languages such as XML-Schema (Fallside (Eds), 2001) or RELAX (Murata, 2000), the degree of complexities captured in a XML schema is far greater than that in a DTD. For instance, XML-Schema supports an extensive set of features to specify structural and semantic constraints. However, all existing XML to relational conversion algorithms (discussed in Section 2.1) focus only on the DTD case, which is the simpliest and least expressive XML schema language according to Lee and Chu (2000a); Lee et al. (2000). Therefore, there is an immediate need to modify and extend the current conversion algorithms to support more complex schema languages.

Second, 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 to grow substantially as well. Although commercial database vendors already support tools that generate XML documents out of relational data, the types of XML documents generated are very simple in their structure and consequently cannot capture all semantics in the original relational schema. For instance, a majority of tools can only convert the so-called "flat translation" where a table t and columns c_i of relational model is mapped to an element e and its attributes a_i of XML model. We have proposed the "nesting-based translation" to capture certain semantics in the original relational schema (Lee et al., 2001), however, more research efforts in that direction are needed.

9 Conclusion

This paper presents a method to transform XML DTD to relational schema both in *structural* and *semantic* aspects. After discussing the semantic constraints hidden in DTDs, two algorithms are presented for: 1) discovering the semantic constraints using the hybrid inlining algorithm, and 2) rewriting the semantic constraints in relational notation. Our experimental results reveal that constraints can be systematically preserved during the conversion from XML to relational schema. Such constraints can also be used for semantic query optimization or semantic caching.

Despite the obstacles in converting from XML to relational models and vice versa, there are several practical benefits:

- Considering the present market that is mostly dominated by RDB products, it is not easy nor practical to abandon RDB to support XML. It is very likely that industries would be reluctant to adopt the new technology if it does not support the existing RDB techniques as they were reluctant towards object-oriented database in the past.
- By using RDB as an underlying storage system, the mature RDB techniques can be leveraged. That is, a vast number of sophisticated techniques (e.g., OLAP, Data Mining, Data Warehousing, etc.) developed for RDB can be applied to XML data with minimal changes.
- The integration of a large amount of XML data on the Web with the legacy data in relational format is possible.

We strongly believe that devising more accurate and efficient conversion methodologies between XML and relational models is very important and our CPI algorithm can serve as an enhancement for such conversion algorithms. The prototype of CPI algorithm is available at:

`http://www.cobase.cs.ucla.edu/projects/xpress/`

The interested readers are welcome to experiment, improve and extend further.

References

Abiteboul, S., Buneman, P., Suciu, D., 1999. "Data on the Web: From Relations to Semistructured Data and XML". Morgan Kaufmann Pub.

Banerjee, S., Krishnamurthy, V., Krishnaprasad, M., Murthy, R., Feb. 2000. "Oracle8i - The XML Enabled Data Management System.". In: IEEE ICDE. San Diego, CA.

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

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

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

Böhm, K., Aberer, K., Öszu, M. T., Gayer, K., Apr. 1998. "Query Optimization for Structured Documents Based on Knowledge on the Document Type Definition". In: IEEE Advances in Digital Libraries (ADL). Los Alamitos, CA.

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

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

Buneman, P., Fan, W., Weinstein, S., Jun. 1998. "Path Constraints in Semistructured and Structured Databases". In: ACM PODS. Seattle, WA.

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

Cheng, J. M., Xu, J., Feb. 2000. "XML and DB2". In: IEEE ICDE. San Diego, CA.

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

Deutsch, A., Fernandez, M. F., Suciu, D., Jun. 1998. "Storing Semistructured Data with STORED". In: ACM SIGMOD. Philadephia, PA.

Fallside (Eds), D. C., May 2001. "XML Schema Part 0: Primer". W3C Recommendation, http://www.w3.org/TR/xmlschema-0.

Fan, W., Siméon, J., May 2000. "Integrity Constraints for XML". In: ACM PODS. Dallas, TX.

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

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

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

Hopcroft, J. E., Motwani, R., Ullman, J. D., 2001. "Introduction to Automata Theory, Language, and Computation", 2nd Edition. Addison–Wesley.

Kanne, C.-C., Moerkotte, G., Feb. 2000. "Efficient Storage of XML Data". In: IEEE ICDE. San Diego, CA.

Kappel, G., Kapsammer, E., Rausch-Schott, S., Retschitzegger, W., Oct. 2000. "X-Ray - Towards Integrating XML and Relational Database Systems.". In: Int'l Conf. on Conceptual Modeling (ER). Springer, LNCS 1920, Salt Lake City, UT.

Klettke, M., Meyer, H., May 2000. "XML and Object-Relational Database Systems - Enhancing Structural Mappings Based on Statistics". In: Int'l Workshop on the Web and Databases (WebDB). Dallas, TX.

Lee, D., Chu, W. W., Nov. 1999. "Semantic Caching via Query Matching for Web Sources". In: ACM CIKM. Kansas City, MO.

Lee, D., Chu, W. W., Sep. 2000a. "Comparative Analysis of Six XML Schema Languages". ACM SIGMOD Record 29 (3), 76–87.

Lee, D., Chu, W. W., Oct. 2000b. "Constraints-preserving Transformation from XML Document Type Definition to Relational Schema". In: Int'l Conf.

on Conceptual Modeling (ER). Springer, LNCS 1920, Salt Lake City, UT.

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

Lee, D., Mani, M., Murata, M., Nov. 2000. "Reasoning about XML Schema Languages using Formal Language Theory". Technical Report, IBM Almaden Research Center, RJ# 10197, Log# 95071, http://www.cs.ucla.edu/~dongwon/paper/.

Mani, M., Lee, D., Murata, M., 2001. "Normal Forms for Regular Tree Grammars", Unpublished Manuscript, UCLA-CS.

Murata, M., Aug. 2000. "RELAX (REgular LAnguage description for XML)". Web page, http://www.xml.gr.jp/relax/.

Papakonstantinou, Y., Velikhov, P., Mar. 1999. "Enhancing Semistructured Data Mediators with Document Type Definitions". In: IEEE ICDE. Sydney, Austrialia.

Robie, J., Lapp, J., Schach, D., Dec. 1998. "XML Query Language (XQL)". WWW The Query Language Workshop (QL), Cambridge, MA, http://www.w3.org/TandS/QL/QL98/pp/xql.html.

Sahuguet, A., May 2000. "Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask". In: Int'l Workshop on the Web and Databases (WebDB). Dallas, TX.

Schmidt, A., Kersten, M. L., Windhouwer, M., Waas, F., May 2000. "Efficient Relational Storage and Retrieval of XML Documents". In: Int'l Workshop on the Web and Databases (WebDB). Dallas, TX.

Shanmugasundaram, J., Shekita, E. J., Barr, R., Carey, M. J., Lindsay, B. G., Pirahesh, H., Reinwald, B., Sep. 2000. "Efficiently Publishing Relational Data as XML Documents". In: VLDB. Cairo, Egypt.

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

Shimura, T., Yoshikawa, M., Uemura, S., Aug. 1999. "Storage and Retrieval of XML Documents using Object-Relational Databases". In: Int'l Conf. on Database and Expert Systems Applications (DEXA). Springer, LNCS 1677, Florence, Italy.

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

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

Wood, P. T., Nov. 1999. "Optimizing Web Queries Using Document Type Definitions". In: Int'l Workshop on Web Information and Data Management (WIDM). Kansas City, MO.