Tractable XML Data Exchange via Relations

Rada Chirkova
NC State University
chirkova@csc.ncsu.edu

Leonid Libkin
University of Edinburgh
libkin@inf.ed.ac.uk

Juan Reutter
University of Edinburgh
juan.reutter@ed.ac.uk

ABSTRACT

We consider the problem of data exchange for XML documents: given source and target schemas, a mapping between them, and a document that conforms to the source schema, construct a target document and answer target queries in a way that is consistent with the source information. The problem has primarily been studied in the relational context, in which data-exchange systems have also been built.

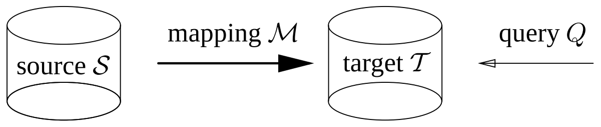
Since many XML documents are stored in relations, it is natural to consider using a relational system for XML data exchange. However, there is a complexity mismatch between query answering in relational and XML data exchange, which indicates that restrictions have to be imposed on XML schemas and mappings, as well as on XML shredding schemes, to make the use of relational systems possible.

We isolate a set of five requirements that must be fulfilled in order to have a faithful representation of the XML data-exchange problem by a relational translation. We then demonstrate that these requirements naturally suggest the inlining technique for data-exchange tasks. Our key contribution is to provide shredding algorithms for schemas, documents, mappings and queries, and demonstrate that they enable us to correctly perform XML data-exchange tasks using a relational system.

1. Introduction

Data exchange is the problem of finding an instance of a target schema, given an instance of a source schema and a schema mapping, i.e., a specification of the relationship between the source and the target. Such a target instance should correctly represent information from the source instance under the constraints imposed by the target schema, and should allow one to evaluate queries on the target instance in a way that is semantically consistent with the source data. The problem has received much attention in the past few years, with several surveys already available [20, 9, 8].

The general setting of data exchange is this:



We have fixed source and target schemas, an instance $\mathcal{S}$ of a source schema, and a mapping $\mathcal{M}$ that specifies the relationship between the source and the target schemas. The goal is to construct an instance $\mathcal{T}$ of the target schema, based on the source and the mapping, and answer queries against the target data in a way consistent with the source data.

The mappings rarely specify the target instance completely, i.e., for each source $\mathcal{S}$ and mapping $\mathcal{M}$, there could be multiple target instances $\mathcal{T}_1, \mathcal{T}_2, \ldots$ that satisfy the conditions of the mapping. Such instances are called *solutions*. The notion of query answering has to account for their non-uniqueness. Typically, one tries to compute *certain answers* $\text{CERTAIN}_{\mathcal{M}}(Q, \mathcal{S}) = \bigcap_i Q(\mathcal{T}_i)$, i.e., answers independent of a particular solution chosen. Such an answer must be produced by evaluating some query – not necessarily Q but perhaps its *rewriting* Q_{rewr} over a particular solution $\mathcal{T}$: so that $Q_{\text{rewr}}(\mathcal{T}) = \text{CERTAIN}_{\mathcal{M}}(Q, \mathcal{S})$.

Thus, the key tasks in data exchange are: (a) choosing a particular solution $\mathcal{T}$ among $\{\mathcal{T}_1, \mathcal{T}_2, \ldots\}$ to materialize, and (b) finding a way of producing query answers over that solution by running a rewritten query Q_{rewr} over it. Usually one builds a so-called *universal* solution [12, 8]; these solutions behave particularly nicely with respect to query answering.

These basics of data exchange are independent of a particular model of data. Most research on data exchange, however, occurred in the relational context [12, 13, 20, 8] or slight extensions [27, 17]; the first paper that attempted to extend relational results to the XML context was [6], and a few followups have since appeared [4, 3]. They all concentrate on the algorithmic aspects of query answering and constructing solutions, with the main goal of isolating tractable cases. The problem these papers do not address is *how XML data exchange can be implemented*?

One possibility is to use a native XML DBMS such as [18], but this is not the most common route: XML data is commonly stored in relational DBMSs. In fact, many ETL products claim that they handle XML data simply by producing relational translations (known as *shredding* [21]). This leads to a two-step approach:

- first shred XML data into relations;
- then apply a relational data-exchange engine (and publish the result back as an XML document).

The approach seems very natural, but the key question is whether it will *work correctly*. That is, are we guaranteed to have the same result as we would have gotten had we implemented a native XML data exchange system?

To state this more precisely, assume that we have a translation $\sigma(\cdot)$ that can be applied to (a) XML schemas, (b) XML documents, (c) XML schema mappings, and (d) XML queries. Then the concept of *correctness* of such a translation is shown below:

$$\begin{tikzcd}
\text{XML :} & \text{source } \mathcal{S} \arrow[r, "\text{mapping } \mathcal{M}"] \arrow[d, "\text{shred}"] & \text{target } \mathcal{T} \arrow[r, "\text{query } Q"] \arrow[d, "\text{shred}"] & \text{answer} \arrow[d, "\text{shred}"] \\
\text{Relations :} & \sigma(\mathcal{S}) \arrow[r, "\sigma(\mathcal{M})"] & \sigma(\mathcal{T}) \arrow[r, "\sigma(Q)"] & \text{answer}
\end{tikzcd}$$

That is, suppose we start with an XML document $\mathcal{S}$ and an XML schema mapping $\mathcal{M}$. In a native system, we would materialize some solution $\mathcal{T}$ over which we could answer queries Q.

But now we want a relational system to do the job. So we shred $\mathcal{S}$ into $\sigma(\mathcal{S})$ and then apply to $\sigma(\mathcal{S})$ the translation of the mapping $\sigma(\mathcal{M})$ to get a solution – which itself is a shredding of an XML solution – so that the answer to Q could be reconstructed from the result of the query $\sigma(Q)$ over that relational solution.

The idea seems simple and natural on the surface, but starts looking challenging once we look deeper into it. Before even attempting to show that the relational translation faithfully represents the XML data-exchange problem, we need to address the following.

Complexity mismatch. Without restrictions, there *cannot be a faithful representation* of XML data exchange by a relational system. Indeed, it is well known that positive relational-algebra queries can be efficiently evaluated in relational data exchange [12, 20, 8], but even for simple XML analogs of conjunctive queries finding query answers can be coNP-hard [6]. So any claim that a relational data-exchange system correctly performs XML data exchange for arbitrary documents and queries is bound to be wrong. We thus need to identify the cases that can be handled by a relational system.

Which shredding scheme to use? There are several, that can roughly be divided into two groups: those that do not take the schema information into account (e.g., the edge-representation [14], interval codings [32], and other numbering schemes [31]), and those that are based on schemas for XML, such as variants of the inlining technique [26, 21]. Since in data-exchange scenarios we start with two schemas, it seems more appropriate to apply schema-based techniques.

Target constraints. In relational data exchange, constraints in target schemas are required to satisfy certain acyclicity conditions; without them, the chase procedure that constructs a target instance does not terminate [12, 20, 8]. Constraints imposed by general XML schema specifications need not in general be even definable in relational calculus, let alone be acyclic [19]. We thus need to find a shredding technique that enables us to encode targets schemas by means of constraints that guarantee chase termination.

As for the complexity issue, the work on the theory of XML data exchange has identified a class of mappings for which efficient query answering is possible [6, 4, 3]. The schemas (say, DTDs), have rules of the form $db \to book^*$, $book \to author^*\ subject$ (we shall give a formal definition later), and the mappings transform patterns satisfied over the source into patterns satisfied over targets.

This restriction suggests a relational representation to use. Going with the edge-representation [14] is problematic: first, each edge in an XML pattern used in a mapping will result in a join in the relational translation, making it inefficient, and second, enforcing even a simple schema structure under that representation takes us out of the class of target constraints that relational data-exchange systems can handle. Verifiably correct translations based on numerical encodings [31, 32] will necessarily involve numerical and/or ordering constraints in relational translations of mappings, and this is something that relational data exchange cannot handle at the moment [20, 8] (beyond simple ordering constraints [2]).

One translation scheme however that fits in very well with restrictions identified in [6, 4, 3] is the *inlining* scheme. It works very well for DTDs of the "right" shape, and its output schemas involve only acyclic constraints, which is perfect for data-exchange scenarios. As for queries, for now we follow [6, 4, 3] and deal with queries whose outputs are relational. (Please see [28] for an overview of native-XML processing of queries whose outputs are sets of XML trees.) The main reason for this, as in those papers, is that for queries with relational outputs the notion of certain answers is well understood (it is the intersection of answers over all possible solutions).

Desiderata for the translation We now formulate some basic requirements for the translation σ, in order to be able to achieve our goals described in the diagram above. We need the following:

Requirement 1: translation of schemas A translation $\sigma(D)$ that, when applied to a DTD of a special form, produces a relational schema that only has acyclic constraints, which can be used in a relational data-exchange setting.

Requirement 2: translation of documents A translation $\sigma_D(\cdot)$ for a DTD D that, when applied to a document T conforming to D, produces a relational database $\sigma_D(T)$ of schema $\sigma(D)$.

Requirement 3: translation of queries For a DTD D, a translation $\sigma_D(Q)$ of (analogs of) conjunctive queries so that $\sigma_D(Q)\big(\sigma_D(T)\big) = Q(T)$ (i.e., the result of $Q(T)$ can be computed by relational translations).

Requirement 4: translation of mappings For a mapping $\mathcal{M}$ between a source DTD D_s and a target DTD D_t, its translation $\sigma(\mathcal{M})$ is a mapping between $\sigma(D_s)$ and $\sigma(D_t)$ that preserves universal solutions. That is:

(a) Each σ_{D_t}-translation of a universal solution for T under $\mathcal{M}$ is a universal solution of for $\sigma_{D_s}(T)$ under $\sigma(\mathcal{M})$; and

(b) Each universal solution for $\sigma_{D_s}(T)$ under $\sigma(\mathcal{M})$ contains[1] a σ_{D_t}-translation of a universal solution of T under $\mathcal{M}$.

Requirement 5: query answering For (analogs of) conjunctive queries over trees, computing $\text{answ}_{\mathcal{M}}(Q,T)$ can be done by computing a $\sigma(\mathcal{M})$-solution of $\sigma(T)$, followed by evaluation of $\sigma(Q)$ over that solution, as is normally done in a relational data-exchange system.

Satisfaction of these five requirements would guarantee that we have a *correct* relational translation of an XML data-exchange problem, which would guarantee correct evaluation of queries.

Contributions Our main contributions are as follows. First, we introduce an architecture for XML data exchange using relational vehicles, with a focus on correct evaluation of (analogs of) conjunctive queries on XML data. Second, we identify a class of XML

[1]We cannot require the equivalence as relational solutions are open to adding new tuples and thus cannot always be translations of trees; we shall discuss this later.

schema mappings and a shredding mechanism that allow us to overcome the complexity mismatch. Third, we provide algorithms for relational translation of schemas, XML documents, schema mappings, and queries in our proposed architecture. Finally, we prove the correctness of the translations: namely, we show that they satisfy the above five requirements, and thus enable us to use relational data exchange systems for XML data exchange tasks.

Related work In recent years, significant effort has been devoted to developing high-performance XML database systems, and to building tools for data exchange. One major direction of the XML effort is the "relational approach", which uses relational DBMSs to store and query XML data. Documents could be translated into relational tuples using either the "DTD-aware" translation [30, 26] or the "schemaless" translation. The latter translation includes the edge [14] and the node [32] representation of the data. Indexes could be prebuilt on the data to improve performance in relational query processing, see, e.g., [31, 32]. Constraints arising in the translation are sometimes dealt with explicitly [7, 29]. See [28] for a survey of the relational approach to answering XML queries.

The work on data exchange concentrated primarily on relations, see [8, 20] for surveys and [24, 25] for system descriptions. Mappings for the XML data exchange problem were studied in [6, 4]; these papers noticed that the complexity of many tasks in XML data exchange in higher than for their relational analogs, which suggests that restrictions must be imposed for a relational implementation. The problem of exchanging XML data was also studied in [15, 25], which give translations of documents and DTDs into nested-relational schemas, and then show how to perform XML data exchange under this translation. Most RDBMSs, however, do not provide support for nested relational schemas, and, thus, specific machinery has to be developed in order to implement this translation under a strictly relational setting. In fact, the results of this paper may aid towards the development of a relational implementation for both XML and nested-relational data exchange.

Outline Key definitions are given in Section 2. Section 3 provides translations of schemas and documents and shows that they fullfill requirements 1 and 2. Section 4 provides the main concepts of relational and XML data exchange. Section 5 provides translations of mappings and queries, and shows that requirements 3, 4, and 5 are fullfilled. Section 6 extends results to handle target constraints and more complex DTDs. Some technical details of algorithms and the proofs of correctness of the translations are in the appendix.

2. Preliminaries

Relational schemas and constraints. A *relational schema*, or just *schema*, is a finite set $\mathbf{R} = \{R_1, \ldots, R_k\}$ of relation symbols, possibly with a set of integrity constraints (*dependencies*). Constraints used most often in data exchange are egd's and tgd's [12, 20, 8] (equality- and tuple-generating dependencies), but for our purposes it will suffice to consider only *keys* and *foreign keys*. If R is a relation over attributes U, and X is a set of attributes, then X is a key of R if no two tuples of R coincide on X-attributes (i.e., for all tuples $t_1, t_2 \in R$ with $t_1 \neq t_2$ we have $\pi_X(t_1) \neq \pi_X(t_2)$). If R_1 and R_2 are relations over sets of attributes U_1 and U_2, respectively, then an inclusion constraint $R_1[X] \subseteq R_2[Y]$, where $X \subseteq U_1$ and $Y \subseteq U_2$ are of the same cardinality, holds when $\pi_X(R_1) \subseteq \pi_Y(R_2)$. We further say that a foreign key the attributes of $R_1[X] \subseteq_{FK} R_2[Y]$ holds if the inclusion constraint $R_1[X] \subseteq R_2[Y]$ holds, and Y is a key of R_2.

With each set of keys and foreign keys, we associate a graph in which we put an edge between attributes A and B if there is a constraint $R_1[X] \subseteq_{FK} R_2[Y]$ with $A \in X$ and $B \in Y$. If this graph is acyclic, we say that the set of constraints is *acyclic*. A schema is acyclic if its constraints are acyclic. In data exchange, one often uses a more technical notion of weak acyclicty: it includes some cyclic schemas for which the chase procedure still terminates. For us, however, the simple concept of acyclicity will suffice, as our translations of schemas only produce acyclic constraints.

XML documents and DTDs Assume that we have the following disjoint countably infinite sets: El of element names, Att of attribute names, and Str of possible values of string-valued attributes. All attribute names start with the symbol @.

An *XML tree* is a finite rooted directed tree $T = (N, G)$, where N is the set of nodes and G is the set of edges, together with

1. a labeling function $\lambda : N \to El$;
2. attribute-value assignments, which are partial functions $\rho_{@a} : N \to Str$ for each $@a \in Att$; and
3. an ordering on children of every node.

A DTD D over El with a distinguished symbol r (for the root) and a set of attributes Att consists of a mapping P_D from El to regular expressions over $El - \{r\}$, usually written as productions $\ell \to e$ if $P_D(\ell) = e$, and a mapping A_D from El to 2^{Att} that assigns a (possibly empty) set of attributes to each element type. For notational convenience, we always assume that attributes come in some order, just like in the relational case: attributes in tuples come in some order so we can write $R(a_1, \ldots, a_n)$. Likewise, we shall describe an ℓ labeled tree node with n attributes as $\ell(a_1, \ldots, a_n)$.

A tree $\mathbf{T}$ conforms to a DTD D (written as $T \models D$) if its root is labelled r, the set of attributes for a node labelled ℓ is $A_D(\ell)$, and the labels of the children of such a node, read from left to right, form a string in the language of $P_D(\ell)$.

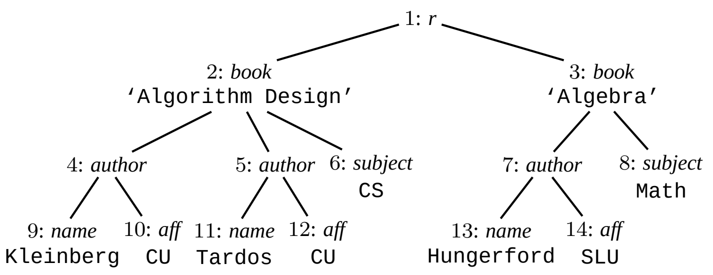
A class of DTDs In this paper we consider a restriction on DTDs called *nested-relational DTDs* [1, 6], a class of DTDs that naturally represent nested relational schemas such as the ones used by the Clio data exchange system [24]. The reason for using them is that outside of this class, it is very easy to construct instances of XML data exchange problems that will exhibit coNP-hardness of answering conjunctive queries (which are known to be tractable in practically all instances of relational data exchange), see [6].

First, a DTD D is *non-recursive* if there is no cycle in the graph $G(D)$ defined as $\{(l, l') \mid l'$ is mentioned in $P(l)\}$. Further, a non-recursive DTD D is a *nested-relational DTD* if all rules of D are of the form $l \to \tilde{l}_0 \ldots \tilde{l}_m$ where all the l_i's are distinct, and each $\tilde{l}_i$ is one of l_i and l_i^*. From now on, unless otherwise noted, all DTD's are assumed to be nested-relational. We also assume, wlog, that the graph $G(D)$ is not a dag but a tree (as one can always unfold a dag into a tree by tagging an element type with the type of its parent).

EXAMPLE 2.1. Figure 1(a) shows an example of an XML tree. In the figure, the node identifiers precede the corresponding labels of each node in T; we omit the attribute names and only show the attribute values of each node. In addition, figure 1(b) shows an example of a nested relational DTD. Moreover, it is easy to see that the tree T of figure 1(a) conforms to D. □

3. Translations of schemas and documents

We now review the *inlining* technique [26], provide a precise definition of the translation, and show that it satisfies **Requirements 1** and **2**. The main idea of inlining is that separate relations are created for the root and each element type that appears under a star, and other element types are inlined in the relations corresponding to their "nearest appropriate ancestor". Each relation for an element type has an ID attribute that is a key, as well as (for non-root)



(a) Tree T

$$
\begin{array}{lcl}
r & \rightarrow & book^{*} \\
book & \rightarrow & author^{*}\ subject \\
author & \rightarrow & name\ aff \\
A_D(book) & = & @title \\
A_D(subject) & = & @sub \\
A_D(name) & = & @nam \\
A_D(aff) & = & @aff
\end{array}
$$

(b) DTD D

Figure 1: The XML tree T conforms to D

a "parent-ID" attribute that is a foreign key pointing to the "nearest appropriate ancestor" of that element in the document. All the attributes of a given element type in the DTD become attributes in the relation corresponding to that element type when such a relation exists, or otherwise become attributes in the relation for the "nearest appropriate ancestor" of the given element type.

We begin with a formal definition of the *nearest appropriate ancestor* for the element types used in D. Given a nested-relational DTD $D = (P_D, A_D, r)$, we "mark" in $G(D)$ each element type that occurs under a star in P_D. In addition, we mark the root element type in $G(D)$. Then, for a given element type ℓ, we define the *nearest appropriate ancestor* of ℓ, denoted by $\mu(\ell)$, as the closest marked element type ℓ' in the path from the root element to ℓ in the graph $G(D)$. The inlining schema generation is formally captured by means of the procedure InlSchema below.

```
Procedure InlSchema($(D)$)
  Input  : A nested relational DTD $D$.
  Output: A relational schema $\mathbf{S}_D$ and a set of integrity
          constraints $\Delta_D$
  Set $\mathbf{S}_D = \emptyset$ and $\Delta_D = \emptyset$
  for each marked element type $\ell$ of $D$:
      add to $\mathbf{S}_D$ a relation $R_\ell$, with attributes:
          $attr(R_\ell) = \begin{cases} id_\ell & \\ A_D(\ell) & \\ id_{\mu(\ell)} & \mid \text{ if } \ell \neq r. \\ id_{\ell'} & \mid \ \mu(\ell') = \ell,\ \ell' \text{ is not marked,} \\ A_D(\ell') & \mid \ \mu(\ell') = \ell,\ \ell' \text{ is not marked.} \end{cases}$
  endfor
  for each relation $R_\ell$ in $\mathbf{S}_D$:
      add to $\Delta_D$ the constraint stating that $id_\ell$ is key of $R_\ell$ and,
      if $\ell \neq r$, the foreign key
          $R_\ell[id_{\mu(\ell)}] \subseteq_{FK} R_{\mu(\ell)}[id_{\mu(\ell)}].$
  endfor
  add to $\Delta_D$ the dependency (stating the uniqueness of the root)
          $\forall \bar{y} \forall \bar{z} R_r(x, \bar{y}) \wedge R_r(x', \bar{z}) \rightarrow x = x'.$
  return $(\mathbf{S}_D, \Delta_D)$
```

Example 3.1. Consider again DTD D in figure 1(b). The relational schema InlSchema(D) is as follows:

R_r(<u>`rID`</u>)
R_{book}(<u>`bookID`</u>,`@title`,`rID`,`subID`,`@sub`)
R_{author}(<u>`authID`</u>,`bookID`,`nameID`,`afID`,`@nam`,`@aff`)

Keys are underlined; we also have the following foreign keys: $R_{book}(\texttt{rID}) \subseteq_{FK} R_r(\texttt{rID})$ and $R_{author}(\texttt{bookID}) \subseteq_{FK} R_{book}(\texttt{bookID})$. □

The following shows that **Requirement 1** is satisfied.

Proposition 3.2. *For every nested relational DTD D, the output of* InlSchema*(D) is an acyclic relational schema.*

Shredding of XML documents. We now move to the shredding procedure. Given the inlining InlSchema(D) $= (\mathbf{S}_D, \Delta_D)$ of a DTD D, and an XML tree T conforming to D, we use the algorithm InlDoc to *shred* T into an instance of the relational schema $\mathbf{S}_D$ that it satisfies the constraints in Δ_D. Let us first explain this translation by means of an example.

Example 3.3. Recall tree T from figure 1(a) and DTD D from figure 1(b). Figure 2 shows relations R_{book} and R_{author} in the shredding of T. □

To present the algorithm, we define the *nearest appropriate ancestor* $\mu(n)$ of a node n of an XML document $T = (N, G)$ that conforms to a DTD D as follows. Mark each node n of T such that $\lambda(n)$ is starred in D, as well as the root of T. Then $\mu(n)$ is the closest marked node n' that belongs to the path from the root to n.

```
Procedure InlDoc($(T, D)$)
  Input  : A nested relational DTD $D$ and an XML tree $T$ that
           conforms to $D$.
  Output: A relational instance of the schema InlSchema($D$).
  for each marked node $n$ of $T$:
      Let $\ell$ be the label of $n$; Add to the relation $R_\ell$ of $I$ a tuple
      that contain elements
          $\begin{cases} id_n & \\ \rho_{@a}(n) & \mid \ @a \in A(\ell) \\ id_{\mu(n)} & \mid \text{ if } \ell \neq r \\ id_{n'} & \mid \ \mu(n') = n, n' \text{ is not marked.} \\ \rho_{@a}(n') & \mid \ \mu(n') = n\ ,\ @a \in A(\lambda(n')) \text{ and} \\ & \quad\ n' \text{ is not marked} \end{cases}$
      where the identifiers and attributes values for each of the
      elements $id_{n'}$, $id_{\mu(n)}$ and $\rho_{@a}(n')$ coincide with the
      position of the attributes for $id_{\lambda(n')}$, $id_{\mu(\ell)}$ and
      $A_D(\lambda(n'))$ of $R_\ell$.
  endfor
  return $I$
```

The following proposition shows **Requirement 2** is satisfied.

Proposition 3.4. *Let D be a DTD, and T an XML tree such that $T \models D$. Let $(\mathbf{S}_D, \Delta_D)$ be the schema computed by* InlSchema*. Then,* InlDoc*$(T, D) \models \Delta_D$.*

bookID	@title	rID	subID	@sub
id_2	'Algorithm Design'	id_1	id_6	CS
id_3	'Algebra'	id_1	id_8	Math

(a) Relation R_{book} in $\textsc{InlDoc}(T, D)$

authID	bookID	nameID	afID	@nam	@af
id_4	id_2	id_9	id_{10}	'Kleinberg'	CU
id_5	id_2	id_{11}	id_{12}	'Tardos'	CU
id_7	id_3	id_{13}	id_{14}	'Hungerford'	SLU

(b) Relation R_{author} in $\textsc{InlDoc}(T, D)$

Figure 2: Shredding of T into $\textsc{InlSchema}(D)$

4. Relational and XML Data Exchange

We now quickly review the basics of relational data exchange and introduce XML schema mappings that guarantee tractable query answering.

Relational Data Exchange A schema mapping $\mathcal{M}$ is a triple $(\mathbb{S}, \mathbb{T}, \Sigma)$, where $\mathbb{S}$ is a source schema, $\mathbb{T} = (\mathbf{T}, \Delta_{\mathbf{T}})$ is a target schema, and Σ is a set of *source-to-target dependencies* that specify how the source and the target are related. Most common these are given as source-to-target tuple generating dependencies (st-tgds):

$$\varphi(\bar{x}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z}), \tag{1}$$

where φ and ψ are conjunctions of relational atoms over $\mathbb{S}$ and $\mathbb{T}$ respectively.

In data-exchange literature, one normally considers instances with two types of values: constants and nulls. Instances $\mathcal{S}$ of the source schema $\mathbb{S}$ consist only of constant values, and nulls are used to populate target instances $\mathcal{T}$ when some values are unknown.

An instance $\mathcal{T}$ of $\mathbb{T}$ (which may contain both constants and nulls) is called a *solution* for an instance $\mathcal{S}$ of $\mathbb{S}$ under $\mathcal{M}$, or an $\mathcal{M}$-*solution* if every st-tgd (1) from Σ is satisfied by $(\mathcal{S}, \mathcal{T})$ (i.e., for each tuple $\bar{a}$ such that $\varphi(\bar{a})$ is true in $\mathcal{S}$, there is a tuple $\bar{b}$ such that $\psi(\bar{a}, \bar{b})$ is true in $\mathcal{T}$), and $\mathcal{T} \models \Delta_{\mathbf{T}}$. The set of all $\mathcal{M}$-solutions for $\mathcal{S}$ is denoted by $\textsc{Sol}_{\mathcal{M}}(\mathcal{S})$ (or $\textsc{Sol}(\mathcal{S})$ is $\mathcal{M}$ is understood).

Certain answers and canonical universal solution The main difficulty in answering a query Q against the target schema is that there could be many possible solutions for a given source. Thus, for query answering in data exchange one normally uses the notion of certain answers, i.e., answers that do not depend on a particular solution. Formally, for a source $\mathcal{S}$ and a mapping $\mathcal{M}$, we define $\textsc{certain}_{\mathcal{M}}(Q, \mathcal{S})$ as $\bigcap\{Q(\mathcal{T}) \mid \mathcal{T} \in \textsc{Sol}_{\mathcal{M}}(\mathcal{S})\}$.

Building all solutions is impractical (or even impossible), so it is important to find a particular solution $\mathcal{T}_0 \in \textsc{Sol}_{\mathcal{M}}(\mathcal{S})$, and a rewriting Q_{rewr} of Q, so that $\textsc{certain}_{\mathcal{M}}(Q, \mathcal{S}) = Q_{\text{rewr}}(\mathcal{T}_0)$.

Universal solutions were identified in [12] as the preferred solutions in data exchange. Over them, every positive query can be answered, with a particularly simple rewriting: after Q is evaluated on a universal solution $\mathcal{T}_0$, tuples containing null values are discarded. Even among universal solutions there are ones that are most commonly materialized in data exchange systems [12, 13, 23]. The one we shall use here is the *canonical solution* $\textsc{CanSol}_{\mathcal{M}}(\mathcal{S})$, computing by applying the chase procedure with constraints Σ and $\Delta_{\mathbf{T}}$ to the source instance $\mathcal{S}$. If all the constraints in $\mathcal{S}$ are acyclic (in fact, even weaker notions suffice), such a chase terminates and computes $\textsc{CanSol}_{\mathcal{M}}(\mathcal{S})$ in polynomial time [12].

Note that our **Requirement 4** relates universal solutions in relational and XML data exchange; in particular, we do not insist on working with the canonical solutions, and others, such as the core [13] or the algorithmic constructions of [23] can be used as well.

Towards XML schema mappings: patterns To define XML schema mappings we need the notions of schemas and source-to-target dependencies. The notions of schema are well understood in the XML context. Our dependencies, as in [6, 4, 3] will be based on *tree patterns*. Patterns are defined inductively as follows:

- $\ell(\bar{x})$ is a pattern, where ℓ is a label, and $\bar{x}$ is a (possibly empty) tuple of variables (listing attributes of a node);
- $\ell(\bar{x})[\pi_1, \ldots, \pi_k]$ is a pattern, where $\pi_1, \ldots, \pi_k$ are patterns, and ℓ and $\bar{x}$ are as above.

We write $\pi(\bar{x})$ to indicate that $\bar{x}$ is the tuple of all the variables used in a pattern. The semantics is defined with respect to a node of a tree and a valuation of all the variables of a pattern as attribute values. Formally, $(T, v) \models \pi(\bar{a})$ means that π is satisfied in node v when $\bar{x}$ is interpreted as $\bar{a}$. It is defined as follows:

- $(T, v) \models \ell(\bar{a})$ if v is labeled ℓ and its tuple of attributes is $\bar{a}$;
- $(T, v) \models \ell(\bar{a})[\pi_1(\bar{a}_1), \ldots, \pi_k(\bar{a}_k)]$ if
  1. $(T, v) \models \ell(\bar{a})$ and
  2. there exist children $v_1, \ldots, v_k$ of v (not necessarily distinct) so that $(T, v_i) \models \pi_i(\bar{a}_i)$ for every $i \leq k$.

We write $T \models \pi(\bar{a})$ if $(T, r) \models \pi(\bar{a})$, i.e., the pattern is witnessed at the root.

EXAMPLE 4.1. Consider tree T from figure 1(a), and the tree pattern $\pi(x, y) = r[book(x)[author[name(y)]]]$, that finds books together with the names of their authors. Then, it is easy to see that $T \models \pi($'Algorithm Design', Tardos$)$. In fact, the evaluation of $\pi(x, y)$ over T returns the tuples ('Algorithm Design', Tardos), ('Algorithm Design', Kleinberg), and ('Algebra', Hungerford). □

Given a DTD D and a tree pattern π, we say that π is *compatible* with D if there exists a tree T that conforms to D and a tuple of attribute values $\bar{a}$ such that $T \models \pi(\bar{a})$. In general checking compatibility of patterns with DTDs is NP-complete [10] but for the DTDs we consider here it can be easily done in polynomial time.

EXAMPLE 4.2.[Example 4.1 continued] The pattern $\pi(x, y)$ is compatible with the DTD D of figure 1(b). On the other hand, the pattern $\pi'(x) = r[author(x)]$ is not, because no tree consistent with D can have a child of r labelled as *author*, or an *author*-labelled node with an attribute. □

Remark More general patterns have been considered in the literature [5, 22, 10, 4]; in particular, they may involve descendant navigation, wild cards for labels, and horizontal axes. However, [6, 4] showed that with these features added, query answering in data exchange becomes intractable even for very simple queries. In fact, the restirctions we use in our definition were identified in [6] as essential for tractability of query answering.

XML schema mappings As our descriptions of XML schemas we shall use DTDs (since for complex schemas, query answering in data exchange is known to be intractable [6], and DTDs will suffice to capture all the known tractable cases). Source-to-target constraints will be given via patterns.

Formally, an *XML schema mapping* a triple $\mathcal{M} = (D_S, D_T, \Sigma)$, where D_S is the source (nested relational) DTD, D_T is the target (nested relational) DTD, and Σ is a set of *XML source-to-target dependencies* [6], or XML stds, that are expressions of the form

$$\pi(\bar{x}) \rightarrow \pi'(\bar{x}, \bar{z}), \tag{2}$$

where π and π' are tree patterns compatible with D_S and D_T, resp.

As in the relational case, target trees may contain nulls to account for values not specified by mappings. That is, given a tree T that conforms to D_S, a tree T' (over constants and nulls) is an $\mathcal{M}$-solution for T if T' conforms to D_T, and the pair (T, T') satisfy all the dependencies (2) from Σ. The latter means that for every tuple $\bar{a}$ of attribute values from T, if T satisfies $\pi(\bar{a})$, then there exists a tuple $\bar{b}$ of attribute values from T' such that T' satisfies $\pi'(\bar{a}, \bar{b})$. The set of al $\mathcal{M}$-solutions for T is denoted by $\text{SOL}_{\mathcal{M}}(T)$.

EXAMPLE 4.3. Consider the data exchange scenario $(D, D_T, \mathcal{M})$ given by de DTDs D and D_T of figures 1(b) and 3(b), respectively, and where $\mathcal{M}$ is specified by the dependency

$$r[book(x)[author[name(y)]]] \rightarrow r[writer[name(y), work(x)]],$$

that restructures book-author pairs as writer-work. It can be seen that the XML tree T' in figure 3(a) is an $\mathcal{M}$-solution for T. $\square$

5. XML data exchange using relations

We now provide algorithms for implementing XML data exchange via relational translations. Since we have already shown how to translate DTDs and documents, we need to present translations of stds of mappings and queries. Both of them are based on translating patterns into relational conjunctive queries. We first concentrate on that translation, and then show how to extend it easily to mappings and queries, and prove the correctness of the translations. This will complete our program of using a relational system for XML data exchange in a semantically correct way.

Inlining tree patterns. They key ingredient in our algorithms is a translation of patterns π compatible with a DTD D into a *conjunctive query* INLPATTERN(π, D) over the relational schema INLSCHEMA(D). Very roughly, it can be viewed as this:

1. View a pattern $\pi(\bar{x})$ as a tree T_π in which some attribute values could be variables;
2. compute the relational database INLDOC(T_π, D) (which may have variables as attribute values);
3. view INLDOC(T_π, D) as a tableau of a conjunctive query; the resulting query is INLPATTERN(π, D).

The algorithm is actually more complicated because INLDOC cannot be used in ste 2; we shall explain shortly why.

Towards defining INLPATTERN, observe that each tree pattern $\pi(\bar{x})$ can be viewed as an XML document $T_{\pi(\bar{x})}$, in which both values and variables can be used as attribute values. It is defined inductively as follows: $T_{\ell(\bar{x})}$ is a single-node tree labeled ℓ, with $\bar{x}$ as attribute values, and if π is $\ell(\bar{x})[\pi_1(\bar{x}_1), \ldots, \pi_k(\bar{x}_k)]$, then the root of T_π is a labeled ℓ and has $\bar{x}$ as attribute values, and it has k children, with the subtrees rooted at them being $T_{\pi_1(\bar{x}_1)}, \ldots, T_{\pi_k(\bar{x}_k)}$.

However, even for a pattern $\pi(\bar{x})$ compatible with a DTD D, we may not be able to define its inlining as the inlining of $T_{\pi(\bar{x})}$, because $T_{\pi(\bar{x})}$ need not conform to D. For example, if a DTD has a rule $r \rightarrow ab$ and we have a pattern $r[a]$, it is compatible with D, but $T_{r[a]}$ does not conform to D, as it is missing a b-node. Hence, the procedure INLDOC cannot be used 'as-is' in our algorithm.

Nevertheless, we can still mark the nodes of $T_{\pi(\bar{x})}$ wrt D and define the nearest appropriate ancestor exactly as it has been done previously. With this, we define the procedure INLPATTERN which, intuitively, shreds each node $T_{\pi(\bar{x})}$ independently, adding existential quantifiers to account for the missing information.

```
Procedure INLPATTERN($\pi$, $D$)
 Input  : A DTD $D$, a tree pattern $\pi(\bar{x})$ compatible with $D$.
 Output: Conjunctive query over INLSCHEMA($D$).
 for each node $v$ of $T_{\pi(\bar{x})}$ of form $\ell(\bar{x}_v)$:
    Construct a query $Q_v(\bar{x}_v)$ as follows:
    if $v$ is marked then
       $Q_v(\bar{x}_v) := \exists id_v \exists id_{\mu(v)} \exists \bar{z} R_\ell(id_v, id_{\mu(v)}, \bar{x}_v, \bar{z})$,
       where $\bar{z}$ is a tuple of fresh variables, and the positions
       of variables $id_v$ $id_{\mu(v)}$ and $\bar{x}_v$ are consistent with the
       attributes $id_\ell$, $id_{\mu(\ell)}$ and $A_D(\ell)$ in attr($R_\ell$).
       If $\ell = r$, then $Q_v$ does not use $id_{\mu(v)}$.
    else ($v$ is not marked):
       set $v' := \mu(v)$,  $\ell' := \lambda(v')$, and
       $Q_v(\bar{x}_v) := \exists id_{v'} \exists id_{\mu(v')} \exists id_v \exists \bar{z} R_{\ell'}(id_{v'}, id_{\mu(v')}, id_v, \bar{x}_v, \bar{z})$,
       where $\bar{z}$ is a tuple of fresh variables, and the positions
       of the variables $id_{v'}$, $id_{\mu(v')}$, $id_v$ and $\bar{x}_v$ are consistent
       with the attributes $id_{\ell'}$, $id_{\mu(\ell')}$, $id_\ell$ and $A_D(\ell)$ in
       attr($R_{\ell'}$). If $\ell' = r$, then $Q_v$ does not use $id_{\mu(v')}$.
 endfor
 return $\bigwedge_{v \in T_{\pi(\bar{x})}} Q_v(\bar{x}_v)$.
```

Note that the compatibility of π with D ensures that the translation INLQUERY is well-defined. That is, (1) every attribute formula of the form $\ell(\bar{x})$ only mentions attributes in $A_D(\ell)$, and (2) for all nodes $v, v' \in T_{\pi(\bar{x})}$, if v' is a child of v, then $\lambda(v') \in P_D(\lambda(v))$.

Correctness. Given a pattern $\pi(\bar{x})$, the evaluation of π on a tree T is $\pi(T) = \{\bar{a} \mid T \models \pi(\bar{a})\}$. The following proposition shows the correctness of INLPATTERN.

PROPOSITION 5.1. *Given a nested relational DTD D, a pattern π compatible with D, and a tree T that conforms to D, we have* $\pi(T) = \text{INLPATTERN}(\pi, D)\big(\text{INLDOC}(T, D)\big)$.

That is, the inlining of π, applied to the inlining of T, returns $\pi(T)$.

Conjunctive queries over trees. We use the language that is essentially conjunctive queries over trees [6, 16, 10] with navigation along the child axis. The language $\mathcal{CTQ}$ is obtained by closing patterns under conjunction and existential quantification:

$$Q \;:=\; \pi \mid Q \wedge Q \mid \exists x\, Q,$$

where π is a fully specified tree-pattern formulae. The semantics is straightforward, given the semantics of patterns defined above: $Q(\bar{a}) \wedge Q'(\bar{b})$ is true iff both $Q(\bar{a})$ and $Q'(\bar{b})$ are true, and $\exists x\, Q(\bar{a}, x)$ is true iff $Q(\bar{a}, c)$ is true for some value c. The output of Q on a tree T is denoted by $Q(T)$.

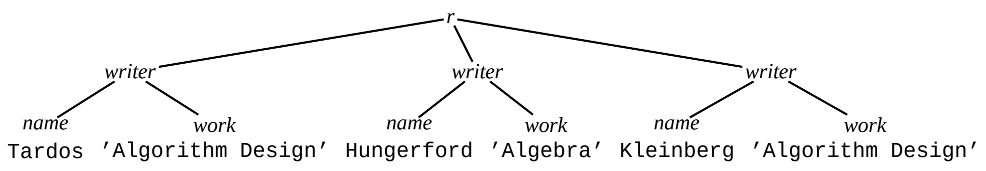
We say that a query Q is compatible with the DTD D if every pattern used in it is compatible with D.

The inlining of queries Q compatible with D is given by the following recursive algorithm.

Now we show that every query Q from $\mathcal{CTQ}$ can be computed by its inlining on the inlining of its input (assuming, of course, compatibility with a DTD). In other words, **Requirement 3** is satisfied.

THEOREM 5.2. *Given a DTD D, a tree T that conforms to it, and a compatible query Q, we have*

$$Q(T) \;=\; \text{INLQUERY}(Q, D)\big(\text{INLDOC}(T, D)\big).$$



(a) Target Tree T'

$$\begin{array}{lcl} r & \rightarrow & writer^* \\ writer & \rightarrow & name\; work^* \\ A_D(name) & = & @nam \\ A_D(work) & = & @title \end{array}$$

(b) Target DTD D_T

Figure 3: Tree T' is an $\mathcal{M}$-solution for T

```
Procedure INLQUERY($Q$, $D$)
  Input  : A DTD $D$, a query $Q$ compatible with $D$.
  Output: A conjunctive query over INLSCHEMA($D$).

  if $Q = \pi$ then
      return INLPATTERN($\pi, D$)
  else if $Q = Q_1 \wedge Q_2$ then
      return INLQUERY($Q_1, D$) $\wedge$ INLQUERY($Q_2, D$)
  else if $Q = \exists x Q_1$ then
      return $\exists x$ INLQUERY($Q_1, D$)
```

Inlining XML schema mappings We use our transformation of tree patterns to define the procedure INLMAP, that, given source and target DTDs D_S and D_T, transforms an XML mapping $\mathcal{M}$ into a relational mapping INLMAP($\mathcal{M}$,D_S,D_T) specified with a set of source-to-target tuple generating dependencies.

```
Procedure INLMAP($\mathcal{M}$, $D_S$, $D_T$ )
  Input  : An XML mapping $\mathcal{M}$ from a source DTD $D_S$ to a
           target DTD $D_T$.
  Output:  A relational mapping from INLSCHEMA($D_S$) to
           INLSCHEMA($D_T$).

  Set INLMAP($\mathcal{M}, D_S, D_T$) := $\emptyset$
  for dependency $\pi(\bar{x}) \rightarrow \exists \bar{z} \pi'(\bar{x}, \bar{z})$ in $\mathcal{M}$ do
    INLMAP($\mathcal{M}, D_S, D_T$) := INLMAP($\mathcal{M}, D_S, D_T$) $\bigcup$
     {INLQUERY($\pi, D_S$)($\bar{x}$) $\rightarrow \exists \bar{z}$ INLQUERY($\pi', D_T$)($\bar{x}, \bar{z}$)}

  end
  return INLMAP($\mathcal{M}, D_S, D_T$)
```

Correctness While one could be tempted to ask for a translation that preserves all solutions such a result need not not hold. The relational mapping INLMAP uses null values to represent the shredded nodes of XML trees, and thus we should only consider solutions whose null values have not been renamed. However, relational solutions are open to renaming of nulls. This intuition can be formalized by means of the universal solutions, which are most general among all solutions, and thus do not permit null renaming. Furthermore, one typically materializes a universal solution, as these solution contain all the information needed to compute certain answers of conjunctive queries. This motivates the restriction of **Requirement 4** to universal solutions.

The theorem below shows that parts (a) and (b) of **Requirement 4** hold. Note that in part (b), relational universal solutions are only required to contain a shredding of an XML universal solution. This is because relational soltions are also open to adding arbitrary tuples, which need not reflect a tree structure of an XML document.

THEOREM 5.3. *a) Let $\mathcal{M} = (D_S, D_T, \Sigma)$ be an XML schema mapping and T an XML document that conforms to D_S. If T' is an $\mathcal{M}$-universal solution for T, then its inlining* INLDOC(T', D_T) *is an* INLMAP($\mathcal{M}$)*-universal solution for* INLDOC(T, D_S).

b) Let $\mathcal{M} = (D_S, D_T, \Sigma)$ be an XML schema mapping, and T an XML document that conforms to D_S. Then, for every INLMAP($\mathcal{M}, D_S, D_T$)*-universal solution R for* INLDOC(T, D_S) *there exists an $\mathcal{M}$-universal solution T' such that* INLDOC(T', D_T) *is contained in R.*

Answering XML queries using relational data exchange. The semantics of query answering in data exchange, both relational and XML [12, 20, 8, 6, 4] is defined by means of certain answers. That is, given a schema mapping $\mathcal{M} = (D_S, D_T, \Sigma)$, a tree T that conforms to D_S and a conjunctive tree query Q that is compatible with D_T, the *certain answers of Q for T under $\mathcal{M}$*, denoted by CERTAIN$_{\mathcal{M}}(Q, T)$, is the set of tuples that belong to the evaluation of Q over every possible $\mathcal{M}$-solution for T, that is, $\bigcap\{Q(T') \mid T'$ is an $\mathcal{M}$-solution for $T\}$. Note that our queries return sets of tuples, so we can talk about the intersection operator.

It was shown in [6, 4] that, for conjunctive tree queries and mappings using nested-relational DTDs, the problem of computing certain answers for a given source tree T is solvable in polynomial time. Thus, for the classes of mappings and queries we consider, there is no complexity mismatch between relational and XML data exchange. The next thorem shows that our translation is correct with respect to query answering, i.e., **Requirement 5** is satisfied.

THEOREM 5.4. *Let $\mathcal{M} = (D_S, D_T, \Sigma)$ be an XML schema mapping. Then, for every XML tree T that satisfies D_S and for every conjunctive tree query Q, the certain answers of Q for T under $\mathcal{M}$ and the certain answers of* INLQUERY(Q, D_T) *for* INLDOC(T, D_S) *over* INLMAP($\mathcal{M}, D_S, D_T$) *coincide:*

$$\text{CERTAIN}_{\mathcal{M}}(Q, T) = \text{CERTAIN}_{\text{INLMAP}(M)}(\text{INLQUERY}(Q, D_T), \text{INLDOC}(T, D_S)).$$

This result, combined with the standard procedure of evaluating conjunctive queries in relational data exchange, also gives us an algorithm for computing certain answers.

COROLLARY 5.5. *Under conditions of Theorem 5.4,* CERTAIN$_{\mathcal{M}}(Q, T)$ *can be obtained by the following procedure:*

1. *evaluate* INLQUERY(Q, D_T) *on an* INLMAP($\mathcal{M}, D_S, D_T$)*-universal solution for* INLDOC(T, D_S)*;*
2. *discard all tuples that contain null values.*

6. Adding XML constraints

So far, we assumed that target schemas consist exclusively of DTDs; now we extend them with *integrity constraints*. Such extensions are very common: for instance, the `ID` and `IDREF` features are somewhat similar to keys and foreign keys. Thus, it is natural to ask whether our procedures continue to work when target schemas are augmented with additional dependencies. Here we look *keys* and *foreign keys* that naturally extend the functionality of `ID` and `IDREF`:

- A *key* $@a \rightarrow \ell$ states that the attribute $@a$ uniquely determines an ℓ-labeled node;
- a *foreign keys* $\ell_1[@a] \subseteq_{FK} \ell_2[@b]$ states that the $@a$ attribute of ℓ_1-nodes must occur as the $@b$ attribute of ℓ_2-nodes, and the latter is a key for ℓ_2.

A natural approach is to to translate XML keys and foreign keys into relational integrity constraints, and then verify that that our requirements continue to be satisfied. We now show how to do this, using our assumption that graphs of DTDs are trees.

```
Procedure InlConstr($\Delta$,$D$)
Input  : A DTD $D$, a set of keys and foreign keys $\Delta$.
Output: A set of relational keys and foreign keys.

Set InlConstr($\Delta, D$) = $\emptyset$
for each key $@a \rightarrow \ell$ in $\Delta$:
    add to InlConstr($\Delta, D$) the key $@a \rightarrow R_\ell$ if $\ell$ is
    marked, or the key $@a \rightarrow R_{\mu(\ell)}$ if $\ell$ is not marked.
endfor
for each foreign key $\ell_1[@a] \rightarrow \ell_2[@b]$ in $\Delta$
    Add to InlConstr($\Delta, D$) the foreign key
    $R_{\ell_1}[@a] \rightarrow R_{\ell_2}[@b]$, replacing $R_{\ell_i}$ for $R_{\mu(\ell_i)}$ if $\ell_1$ or $\ell_2$
    are not marked.
endfor
return InlConstr($\Delta, D$)
```

Using InlConstr, we can extend the procedure InlMap for the case of XML data exchange settings that include a set of *target* constraints Δ_T in a way that retains its good properties (due to space constraints, the precise definition is in the appendix).

Proposition 6.1. *For XML data exchange settings that include a set Δ of XML keys and foreign keys, the extensions of procedure* InlMap *and* InlQuery *using* InlConstr(Δ, D) *satisfy* **Requirement 4** *and* **Requirement 5***, resp.*

Unlike in other results in the paper, the restriction to DTDs whose graphs are trees is essential here: without such a restriction, a foreign key can be translated into a disjunctive tgd, and those are known to lead to intractability in data exchange scenarios [11].

7. Concluding Remarks

Our technique provides a relational approach to solve two of the most important problems of XML data exchange settings: materializing solutions, and answering queries. The diagram below summarizes this. In a pure XML setting, we can start with a document T, and use a mapping $\mathcal{M}$ to find a (universal) solution T'_{univ}, over which we can then answer a query Q to produce certain answers.

$$\begin{tikzcd}
T \arrow[r, "\mathcal{M}"] \arrow[d, "\textsc{InlDoc}"] & T'_{\text{univ}} \arrow[r, "Q"] \arrow[d, "\textsc{InlDoc}"] & \text{certain answer} \arrow[d, equal] \\
R \arrow[r, "\textsc{InlMap}(\mathcal{M})"] & R'_{\text{univ}} \arrow[r, "\textsc{InlQuery}(Q)"] & \text{certain answer}
\end{tikzcd}$$

Using the translation InlDoc of documents, we generate a relational instance R, on which the translation of the mapping InlMap$(\mathcal{M})$ generates a universal solution R'_{univ}. This solution is a shredding of a universal XML solution, and also conforms to the shredding of source DTD. Finally, we apply the standard technique [12] for evaluating queries in relational data exchange to the query translation InlQuery(Q) to produce the correct answers.

8. References

[1] S. Abiteboul, L. Segoufin and V. Vianu. Representing and querying XML with incomplete information. *TODS*, 31(1) (2006), 208-254

[2] F. Afrati, C. Li, V. Pavlaki. Data exchange in the presence of arithmetic comparisons. In *EDBT 2008*, pages 487-498.

[3] S. Amano, C. David, L. Libkin, F. Murlak. On the tradeoff between mapping and querying power in XML data exchange. In *ICDT 2010*.

[4] S. Amano, L. Libkin, F. Murlak. XML schema mappings. In *PODS 2009*, pages 33-42.

[5] S. Amer-Yahia, S. Cho, L. Lakshmanan, D. Srivastava. Tree pattern query minimization. *VLDB J.* 11 (2002), 315–331.

[6] M. Arenas, L. Libkin. XML data exchange: consistency and query answering. *J. ACM* 55(2): (2008).

[7] A. Balmin and Y. Papakonstantinou. Storing and querying XML data using denormalized relational databases. *VLDB J.*, 14:30–49, 2005.

[8] P. Barceló. Logical foundations of relational data exchange. *SIGMOD Record* 38(1): 49–58 (2009).

[9] P. A. Bernstein, S. Melnik. Model management 2.0: manipulating richer mappings. *SIGMOD'07*, pages 1-12

[10] H. Björklund, W. Martens, T. Schwentick. Conjunctive query containment over trees. In *DBPL 2007*, pages 66-80.

[11] A. Deutsch, V. Tannen. Reformulation of XML queries and constraints. In *ICDT'03*, pages 225–241.

[12] R. Fagin, P. G. Kolaitis, R. Miller, L. Popa. Data exchange: semantics and query answering. *TCS* 336(1): 89–124 (2005).

[13] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM TODS* 30(1):174–210, 2005.

[14] D. Florescu, D. Kossman. Storing and querying XML data using a RDBMS *IEEE Data Engineering Bulletin* 22(3): 27–34, 1999.

[15] A. Fuxman, M. Hernández, H. Ho, R. Miller, P. Papotti, L. Popa. Nested mappings: schema mapping reloaded. *VLDB'06*, pages 67-78.

[16] G. Gottlob, C. Koch, K. Schulz. Conjunctive queries over trees. *JACM* 53(2): 238-272, 2006.

[17] M. Hernández, H. Ho, et al. Creating nested mappings with Clio. In *ICDE 2007*, pages 1487-1488.

[18] H. V. Jagadish, et al. TIMBER: A native XML database. *VLDB Journal* 11(4): 274-291, 2002.

[19] N. Klarlund, T. Schwentick, D. Suciu. XML: Model, Schemas, Types, Logics, and Queries. In *Logics for Emerging Applications of Databases 2003*, pages 1-41.

[20] Ph. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS 2005*, pages 61-75.

[21] R. Krishnamurthy, R. Kaushik, J. F. Naughton. XML-to-SQL query translation literature: the state of the art and open problems. In *XSym'03*, pages 1–18.

[22] L. Lakshmanan, et al. On testing satisfiability of tree pattern queries. *VLDB 2004*, pages 120–131.

[23] G. Mecca, P. Papotti, S. Raunich. Core schema mappings. In *SIGMOD 2009*, pages 655-668.

[24] R. Miller, M. Hernández, L. Haas, L. Yan, H. Ho, R. Fagin, L. Popa. The Clio project: managing heterogeneity. *SIGMOD Record* 30(1): 78–83 (2001).

[25] L. Popa, Y. Velegrakis, R. Miller, M. Hernández, R. Fagin. Translating Web data. In *VLDB 2002*, pages 598–609.

[26] J. Shanmugasundaram, et al. Relational databases for querying XML documents: limitations and opportunities. In *VLDB 1999*, pages 302-314.

[27] C. Yu, L. Popa. Constraint-based XML query rewriting for data integration. In *SIGMOD 2004*, pages 371-382.

[28] G. Gou and R. Chirkova. Efficiently querying large XML data repositories: A survey. *IEEE TKDE*, 19:1381–1403, 2007.

[29] R. Krishnamurthy, R. Kaushik, and J. Naughton. XML views as integrity constraints and their use in query translation. In *ICDE'05*, pages 693–704.

[30] J. Shanmugasundaram, et al. A general techniques for querying XML documents using a relational database system. *SIGMOD Record*, 30:20–26, 2001.

[31] I. Tatarinov, et al. Storing and querying ordered XML using a relational database system. In *SIGMOD'02*, pages 204–215.

[32] C. Zhang, et al. On supporting containment queries in relational database management systems. In *SIGMOD'01*, pages 425–436.