Tractable XML Data Exchange via Relations

Rada Chirkova #1, Leonid Libkin *2, Juan Reutter *3

#NC State University
1chirkova@csc.ncsu.edu

*University of Edinburgh
2libkin@inf.ed.ac.uk
3juan.reutter@ed.ac.uk

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

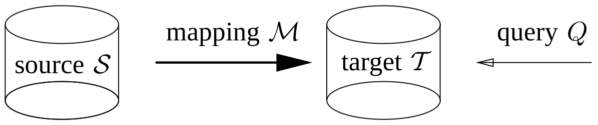
Since many XML documents are stored in relations, it is natural to consider using a relational system for XML data exchange. However, there is a complexity mismatch between query answering in relational and XML data exchange, which indicates that restrictions have to be imposed on XML schemas and mappings,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

I. 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, that is, 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 [22], [9], [8].

The general setting of data exchange is this:



We have fixed source and target schemas, an instance $\mathcal{S}$ of the 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, that is, 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* $\text{CERTAIN}_{\mathcal{M}}(Q, \mathcal{S})$, i.e., answers independent of a particular solution chosen. If Q produces relations, these are usually defined as $\bigcap_i Q(\mathcal{T}_i)$. Certain answers 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 [13], [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 [13], [14], [22], [8] or slight extensions [33], [19]; 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*?

Previous work on algorithms for XML data exchange has tacitly assumed that one uses a native XML DBMS such as [20]. However, this is not the only (and perhaps not even the most common) route: XML documents are often stored in relational DBMSs. In fact, many ETL products claim that they handle XML data simply by producing relational translations (known as *shredding* [23]). 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 [13], [22], [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 [15], interval codings [34], and other numbering schemes [31]), and those that are based on schemas for XML, such as variants of the inlining technique [29], [23]. 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 [13], [22], [8]. Constraints imposed by general XML schema specifications need not in general be even definable in relational calculus, let alone be acyclic [21]. 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 [15] 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], [34] 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 [22], [8] (beyond simple ordering constraints [2]).

One translation scheme however that fits in 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.

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 document T conforming to D, produces 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 $\sigma_D(Q)\big(\sigma_D(T)\big) = Q(T)$ (that is, 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 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 the answer to Q under

[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.

$\mathcal{M}$ over a source tree T is the same as 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.

For the choice of the query language, one has to be careful since the definition of certain answers depends on the output of the queries. We consider two classes of conjunctive queries over trees. The first is tree patterns that output tuples of attribute values. These are the queries most commonly considered in XML data exchange [6], [4], [3] because for them we can define certain answers as the usual intersection $\text{CERTAIN}_{\mathcal{M}}(Q, \mathcal{S}) = \bigcap_i Q(\mathcal{T}_i)$. The second type of queries we use is a simple XML-to-XML query language from whose queries output *trees*. It is essentially the positive fragment of FLWR expressions of XQuery [32]. For outputs which are XML trees, the intersection operator is no longer meaningful for defining certain answers. Instead, we use recent results of [11] that show how to define and compute certain answers for XML-to-XML 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 schema mappings and a shredding mechanism that allows 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 a "DTD-aware" translation [30], [29] or a "schemaless" translation. The latter translations include the edge [15] and the node [34] representations of the data. Indexes could be prebuilt on the data to improve performance in relational query processing, see, e.g., [31], [34]. Constraints arising in the translation are sometimes dealt with explicitly [7], [24]. See [18] for a survey of the relational approach to answering XML queries.

The work on data exchange concentrated primarily on relations, see [8], [22] for surveys and [27], [28] for system descriptions. Mappings for the XML data exchange problem were studied in [6], [4]; it was noticed there 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 [16], [28], 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.

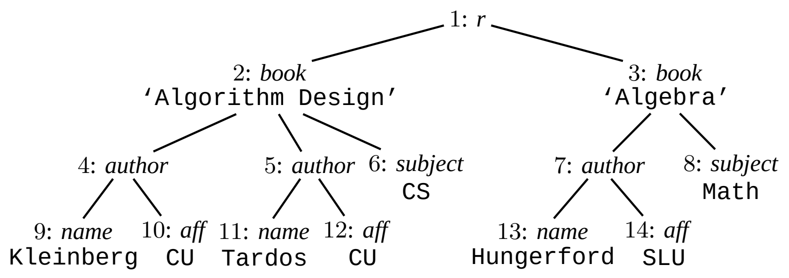
Outline Key definitions are given in Section II. Section III provides translations of schemas and documents and shows that they fulfill Requirements 1 and 2. Section IV provides the main concepts of relational and XML data exchange. Section V provides translations of mappings and queries, and shows that Requirements 3, 4, and 5 are fulfilled. Section VI studies queries that output XML trees. Finally, section VII extends the results to handle target constraints. Formal proofs of correctness of all the algorithms can be found in the full version which is available at www.csc.ncsu.edu/research/tech/index.php, as technical report TR-2010-16.

II. 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 [13], [22], [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 (that is, 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 on 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 acyclicity: 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 @.



(a) Tree T

r	$\rightarrow$	*book**
book	$\rightarrow$	$author^*\ subject$
$author$	$\rightarrow$	$name\ aff$
A_D(*book*)	$=$	$@title$
$A_D(subject)$	$=$	$@sub$
$A_D(name)$	$=$	$@nam$
$A_D(aff)$	$=$	$@aff$

(b) DTD D

Fig. 1. The XML tree T conforms to D

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 $\lambda : N \rightarrow El$;
2) attribute-value assignments, which are partial functions $\rho_{@a} : N \rightarrow Str$ for each $@a \in Att$; and
3) an ordering on the 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 $\ell \rightarrow e$ 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 T conforms to a DTD D (written as $T \models D$) if its root is labeled r, the set of attributes for a node labeled 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 [27]. 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].

A DTD D is *non-recursive* if the graph $G(D)$ defined as $\{(\ell, \ell') \mid \ell'$ is mentioned in $P(\ell)\}$ is acyclic. A non-recursive DTD D is *nested-relational* if all rules of D are of the form $l \rightarrow \tilde{l}_0 \ldots \tilde{l}_m$ 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 DTDs are assumed to be nested-relational. We also assume, without loss of generality, that the graph $G(D)$ is not a directed acyclic graph (DAG) but a tree. (One can always unfold a DAG into a tree by tagging occurrences of element types with the types of their predecessors.)

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. $\square$

III. Translations of schemas and documents

We now review the *inlining* technique [29], provide a precise definition of the translation, and show that it satisfies our **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 "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.

EXAMPLE 3.1. Consider again DTD D in Figure 1(b). The relational schema INLSCHEMA(D) is as follows:

$R_r(\underline{\texttt{rID}})$
$R_{book}(\underline{\texttt{bookID}},\texttt{@title},\texttt{rID},\texttt{subID},\texttt{@sub})$
$R_{author}(\underline{\texttt{authID}},\texttt{bookID},\texttt{nameID},\texttt{afID},\texttt{@nam},\texttt{@aff})$

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

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:
    $attr(R_\ell) = \begin{cases} id_\ell & \\ A_D(\ell) & \\ id_{\mu(\ell)} & | \text{ if } \ell \neq r. \\ id_{\ell'} & | \ \mu(\ell') = \ell,\ \ell' \text{ is not marked,} \\ A_D(\ell') & | \ \mu(\ell') = \ell,\ \ell' \text{ is not marked.} \end{cases}$
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 $(\mathbf{S}_D, \Delta_D)$
```

The following shows that our **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 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. $\square$

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. In the following algorithm, and for the remainder of the paper, we denote by id_n the relational element representing the node n of a tree T.

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

Proposition 3.4: Let D be a DTD, and T an XML tree such that $T \models D$. Then INLDOC(T, D) is an instance of the schema computed by INLSCHEMA(D).

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 contains elements
    $\begin{cases} id_n & \\ \rho_{@a}(n) & | \ @a \in A_D(\ell) \\ id_{\mu(n)} & | \text{ if } \ell \neq r \\ id_{n'} & | \ \mu(n') = n, n' \text{ is not marked.} \\ \rho_{@a}(n') & | \ \mu(n') = n\ ,\ @a \in A_D(\lambda(n')) \text{ and} \\ & \quad n' \text{ is not marked} \end{cases}$
    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 $I$
```

IV. 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 with a set of constraints Δ_T, and Σ is a set of *source-to-target dependencies* that specify how the source and the target are related. Most commonly these are given as source-to-target tuple generating dependencies (st-tgds):

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

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 $(\mathcal{S}, \mathcal{T})$ (that is, 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 $\mathcal{T}$).The set of all $\mathcal{M}$-solutions for $\mathcal{S}$ is denoted by SOL$_{\mathcal{M}}(\mathcal{S})$ (or 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, that is, answers that do not depend on a particular solution. Formally, for a source $\mathcal{S}$ and

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

(a) Relation R_{author} in INLDOC(T, D)

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

(b) Relation R_{book} in INLDOC(T, D)

Fig. 2. Shredding of T into INLSCHEMA(D)

a mapping $\mathcal{M}$, we define CERTAIN$_{\mathcal{M}}(Q, \mathcal{S})$ as $\bigcap\{Q(\mathcal{T}) \mid \mathcal{T} \in \text{SOL}_{\mathcal{M}}(\mathcal{S})\}$.

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

Universal solutions were identified in [13] 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, such as the *canonical solution* CANSOL$_{\mathcal{M}}(\mathcal{S})$, computed by applying the chase procedure with constraints Σ and $\Delta_{\mathbf{T}}$ to the source instance $\mathcal{S}$. If all the constraints in $\Delta_{\mathbf{T}}$ are acyclic (in fact, even a weaker notions suffices), such a chase terminates and computes CANSOL$_{\mathcal{M}}(\mathcal{S})$ in polynomial time [13].

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 [14] or the algorithmic constructions of [26] 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 notion of schema is 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 to 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 $(T, r) \models \pi(\bar{a})$, that is, the pattern is witnessed at the root.

EXAMPLE 4.1. Consider tree T from Figure 1(a), and the tree pattern $\pi(x, y) = r[book(x)[author[name(y)]]]$, which finds books together with the names of their authors. Then it is easy to see that $T \models \pi($'Algorithm Design', Tardos$)$. In fact, 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 labeled as *author*, or an *author*-labeled node with an attribute. □

Remark More general patterns have been considered in the literature [5], [25], [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 restrictions 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* is 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

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

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

As in the relational case, target trees may contain nulls to account for values not specified by mappings. 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') satisfies 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 all $\mathcal{M}$-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 the 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 shown that the XML tree T' in Figure 3(a) is an $\mathcal{M}$-solution for T. $\square$

V. 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. Then we 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: The key ingredient in our algorithms is a translation of patterns π compatible with a DTD D into a *conjunctive query* $\text{INLPATTERN}(\pi, D)$ over the relational schema $\text{INLSCHEMA}(D)$. 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 $\text{INLDOC}(T_\pi, D)$ (which may have variables as attribute values);
3) View $\text{INLDOC}(T_\pi, D)$ as a tableau of a conjunctive query; the resulting query is $\text{INLPATTERN}(\pi, D)$.

The algorithm is actually more complicated because INLDOC cannot be used in Step 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 labeled ℓ and has $\bar{x}$ as attribute values. It also 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.

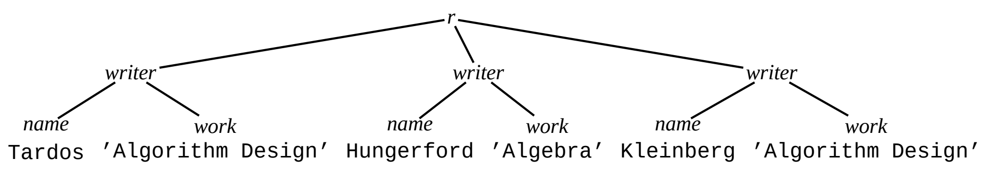
Nevertheless, we can still mark the nodes of $T_{\pi(\bar{x})}$ with respect to D and define the nearest appropriate ancestor exactly as it has been done previously. Intuitively, the procedure INLPATTERN shreds each node of $T_{\pi(\bar{x})}$ into a different predicate, and then joins these predicates using the nearest appropriate ancestor.

```
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
          $Q_v(\bar{x}_v) := \exists id_v \exists id_{\mu(v)} \exists \bar{z} R_\ell(id_v, \bar{x}_v, id_{\mu(v)}, \bar{z})$,
        where $\bar{z}$ is a tuple of fresh variables, and the
        positions of variables $id_v$, $\bar{x}_v$ and $id_{\mu(v)}$ are
        consistent with the attributes $id_\ell$, $A_D(\ell)$ and
        $id_{\mu(\ell)}$ respectively 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 let $Q_v(\bar{x}_v)$ be
          $\exists id_{v'} \exists id_{\mu(v')} \exists id_v \exists \bar{z} R_{\ell'}(id_{v'}, id_{\mu(v')}, id_v, \bar{x}_v, \bar{z})$,
        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 $A_D(\ell)$ respectively 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 INLPATTERN 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 $\pi(T)$.



(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

Fig. 3. Tree T' is an $\mathcal{M}$-solution for T

Conjunctive queries over trees: We use the language that is essentially conjunctive queries over trees [6], [17], [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 formula. 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 recursive algorithm InlQuery below.

```
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($Q_1, D$)
```

Now we show that every query Q in $\mathcal{CTQ}$ 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

$$Q(T) \ = \ \text{InlQuery}(Q, D)\big(\text{InlDoc}(T, D)\big).$$

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.

Correctness While one could be tempted to ask for a translation that preserves all solutions, such a result need 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.

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

This intuition can be formalized by means of the universal solutions, which are the most general among all solutions, and thus do not permit null renaming. Furthermore, one typically materializes a universal solution, as these solutions contain all the information needed to compute certain answers of conjunctive queries. This motivates the restriction of our **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 solutions 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($\mathcal{M}, D_S, D_T$)-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 [13], [22], [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 $\text{certain}_{\mathcal{M}}(Q, T)$, 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, 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 theorem shows that our translation is correct with respect to query answering, that is, our **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 $\textsc{InlQuery}(Q, D_T)$ for $\textsc{InlDoc}(T, D_S)$ over $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$ coincide:

$$\textsc{certain}_{\mathcal{M}}(Q,T) = \textsc{certain}_{\textsc{InlMap}(\mathcal{M})}(\textsc{InlQuery}(Q,D_T), \textsc{InlDoc}(T,D_S)).$$

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 the conditions of Theorem 5.4, $\textsc{certain}_{\mathcal{M}}(Q,T)$ can be obtained by the following procedure:

1) run $\textsc{InlQuery}(Q, D_T)$ on an $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$-universal solution for $\textsc{InlDoc}(T, D_S)$;
2) discard all tuples that contain null values.

VI. XML-to-XML Queries

Up to now, we have only considered XML queries that output tuples of attribute values. In this section, we shall focus on proper XML-to-XML query languages, that is, queries that output XML trees.

Some immediate questions arise when dealing with these formalisms in a data exchange context. Let $\mathcal{M} = (D_S, D_T, \Sigma)$ be an XML schema mapping, T be a tree conforming to D_S, and $\mathcal{Q}$ be an XML-to-XML query. Since the evaluation of $\mathcal{Q}$ over T returns an XML tree, we cannot define certain answers as $\bigcap\{\mathcal{Q}(T')) \mid T'$ is a solution for $T\}$, since the meaning of the intersection operator for XML documents is not clear.

To overcome this problem, we use recent results from [11] which showed how to define certain answers for queries returning XML trees, and how to use them in the data exchange context. The key idea of [11] is to use *tree patterns* to define information contained in documents, and to use them to represent compactly the certain knowledge from the collection $\{\mathcal{Q}(T')) \mid T'$ is a solution for $T\}$. More precisely, if Π is a set of tree patterns which are matched by every tree $Q(T')$, we look for a small set Π_0 of patterns that is equivalent to Π as a description of certain answers. By equivalence we mean that a tree matches every pattern in Π iff it matches every pattern in Π_0. If the set Π_0 is finite, then its patterns can be put together to create a tree with nulls, which we then view as the certain answer.

We shall not need additional details of the construction; instead we shall use a result from [11] that tells us how certain answers can be computed for a specific XML-to-XML query language. The language, called TQL (to be defined shortly), is inspired by XQuery's FLWR expressions, and is restricted to positive features (i.e., no negation). The key result from [11] is the following:

Proposition 6.1 ([11]): Let $\mathcal{M} = (D_S, D_T, \Sigma)$ be an XML schema mapping, Q a TQL query, and T a tree that conforms to D_S. If T' is an $\mathcal{M}$-universal solution for T, then $\textsc{certain}_{\mathcal{M}}(Q,T) = Q(T')$.

Given this result, we now do the following. We provide a formal definition of the TQL language of [11], which can express XML-to-XML analogs of relational conjunctive queries. We then show how to adapt the machinery we have previously developed for evaluating certain asnwers over a universal solution. Note that for this new translation, a TQL query Q returning trees needs to be translated into a *set* of relational queries generating views that define the shredding of the tree $Q(T)$.

A. TQL queries

TQL queries [11] are inspired by the FLWR (for-let-where-return) expressions of XQuery [32], but they only use positive features. The key construct is $\textsf{for}\ \pi(\bar{x})\ \textsf{return}\ q(\bar{x})$, where $\pi(\bar{x})$ is a pattern and $q(\bar{x})$ is a query that defines a forest expression. Formally, the syntax of forest expressions is

$$\begin{array}{rcl} q(\bar{x}) & ::= & \epsilon \\ & | & \ell(\bar{a},\bar{x}')[q'(\bar{x}'')] \\ & | & q'(\bar{x}'), q''(\bar{x}'') \\ & | & \textsf{for}\ \pi(\bar{a},\bar{x},\bar{y})\ \textsf{return}\ q'(\bar{x},\bar{y}) \end{array}$$

where ℓ ranges over node labels, $\bar{a}$ over constant attribute values, and $\bar{x}$ etc are tuples of variables.

A TQL query Q is an expression of the form $r[q]$, where q is a forest expression without variables. To define the semantics of this language, we first define inductively the forest $[\![q(\bar{x})]\!]_{T,v}$, for a valuation v of all variables in $\bar{x}$ as attribute values. We use the notation $\ell(\bar{a})[f]$ for a tree whose root is labeled ℓ and carries a tuple of attributes $\bar{a}$, and f is the forest of subtrees below the root.

$$\begin{array}{rcl} [\![\epsilon]\!]_{T,v} & = & \epsilon \text{ (empty forest)} \\ [\![\ell(\bar{a},\bar{x}')[q'(\bar{x}'')]]\!]_{T,v} & = & \ell(\bar{a},v(\bar{x}'))\left[[\![q']\!]_{T,v}\right] \\ [\![q'(\bar{x}),q''(\bar{x}'')]\!]_{T,v} & = & [\![q']\!]_{T,v} \cup [\![q'']\!]_{T,v} \end{array}$$

$$[\![\textsf{for}\ \pi(\bar{a},\bar{x},\bar{y})\ \textsf{return}\ q'(\bar{x},\bar{y})]\!]_{T,v} =$$

$$\bigcup\{[\![q']\!]_{T,v'} \mid v' \text{ extends } v \text{ and } T \models \pi(\bar{a},v'(\bar{x}),v'(\bar{y}))\}$$

For a tree T and a query $Q = r[q]$, the evaluation $Q(T)$ of Q over T is defined as the tree $r[[\![q]\!]_T]$, i.e., the forest $[\![q]\!]_T$ under root r.

EXAMPLE 6.2. Recall the tree T from figure 1(a). The tree T' from figure 3(a) can also be obtained as the transformation

$Q(T)$ resulting from the evaluation of a TQL query Q over T, where $Q = r[q]$, and q is defined as

$$\text{for } r/book(x)/author/name(y) \text{ return } writer[name(y), work(x)] \quad (3)$$

For the sake of readability, we use the / operator to denote the child axis in tree patterns. □

B. Inlining TQL queries

If Q is a TQL query, then, to be able to define its inlining translation, we need to specify a DTD for trees $Q(T)$. Note that TQL queries define the shape of their outputs, and at the same time do not put restrictions on the number of appearances of labels. Hence it is natural to define the DTD for outputs of Q as a *starred* DTD D_Q, whose shape is determined by Q, and where each element type except the root occurs under the Kleene star.

More precisely, for a forest expression q, we define a forest F_q inductively as follows: F_ε is the empty forest; $F_{\ell[q']}$ is $\ell[F_{q'}]$; $F_{q' \cup q''} = F_{q'} \cup F_{q''}$, and $F_{\text{for } \pi \text{ return } q'} = F_{q'}$. For $Q = r[q]$ we let $T_Q = r[F_q]$.

Then D_Q is a non-recursive DTD that has a rule $p \to c_1^* \cdots c_n^*$ for each node p in T_Q with children labelled $c_1, \ldots, c_n$. As usual, we require that D_Q be acyclic and we assume without loss of generality that $G(D_Q)$ is a tree.

EXAMPLE 6.3.[Example 6.2 continued] Recall query $Q = r[q]$. Then, T_Q is the XML tree given by $r[writer[name, work]]$, and thus D_Q contains productions $r \to writer^*$, $writer \to name^* work^*$, $name \to \epsilon$ and $work \to \epsilon$. □

Before showing the algorithm INLTQL, we need to introduce some features that will be used in the algorithm. Consider again query (3) and DTD D_Q in examples 6.2 and 6.3. For each pair of attributes that satisfy $r/book(x)/author/name(y)$, the query Q creates a subtree $writer[name(y), work(x)]$ in the tree $Q(T)$. Thus, the relational translation would need to create one tuple in the relations corresponding to *writer*, *name* and *work* for each pair of attributes x, y that satisfy the relational translation of the pattern $r/book(x)/author/name(y)$ in the instance INLDOC(T).

Thus, in the relational translation we need a way to associate each particular *writer* wih a particular *name* and *work*. One possible way of doing this is by creating a (Skolem) function f that associates with each pair $(name, work)$ a unique identifier for the corresponding *writer*. Thus, the function f must be defined in such a way that $f(book, name)$ is different for each different pair $(name, work)$. We enforce this requirement by letting each term $f(\bar{a})$ represent a distinct constant $c_{f(\bar{a})}$.

We will define our translation algorithm inductively. The key procedure TQLSTEP for the inductive step is described below. Its inputs, in addition to a query and a DTD, include a conjunctive query corresponding to the conjunction of patterns in the query, and a function term corresponding to the parent in the tree $Q(T)$ (for example, when creating views for relation R_{work}, we would input the identifier $f(x, y)$ of the parent node labelled *writer*). This is illustrated by the example below.

EXAMPLE 6.4.[Example 6.3 continued] Assume that query $Q = r[q]$ of examples 6.2 and 6.3 is posed over T under schema D. The following views define the translation for Q:

$R_r(f_r) := \text{true}$
$R_{writer}(f_{writer}(x, y), f_r) :=$
$\qquad \text{INLQUERY}(r/book(x)/author/name(y), D)$
$R_{name}(f_{name}(x, y), f_{writer}(x, y), y) :=$
$\qquad \text{INLQUERY}(r/book(x)/author/name(y), D)$
$R_{work}(f_{work}(x, y), f_{writer}(x, y), x) :=$
$\qquad \text{INLQUERY}(r/book(x)/author/name(y), D)$

Notice how each tuple in relations R_{name} and R_{work} is set to reference the correct tuple in relation R_{writer}. □

```
Procedure TQLSTEP($Q$, $D$, $\varphi$, $t$)
  Input : A forest expression $q(\bar{x})$, a DTD $D$, a
          conjunctive query $\varphi(\bar{x})$ and a skolem term $t$.
  Output: A set of views over INLSCHEMA($D_Q$).

  if $q(\bar{x}) ::= \epsilon$ then
      return $\emptyset$
  else if $q(\bar{x}) ::= q'(\bar{x}'), q''(\bar{x}'')$ then
      return TQLSTEP($q', D, \varphi, t$) $\cup$ TQLSTEP($q'', D, \varphi, t$)
  else if $q(\bar{x}) ::= \ell(\bar{a}, \bar{x}')[q'(\bar{x}'')]$ then
      Let $f$ be a fresh skolem function. Define view $V$ as
      $R_\ell(f(\bar{x}), t, \bar{a}, \bar{x}') :=$ INLQUERY($\varphi, D$), or just
      $R_\ell(f(), t, \bar{a}) :=$ true if $\varphi = \emptyset$.
      return $\{V\} \cup$ TQLSTEP($q', D, \varphi, f(\bar{x})$)
  else if $q(\bar{x}) ::=$ for $\pi(\bar{a}, \bar{x}, \bar{y})$ return $q'(\bar{x}, \bar{y})$ then
      Let $\varphi'(\bar{a}, \bar{x}, \bar{y}) = \varphi(\bar{x}) \wedge \pi(\bar{a}, \bar{x}, \bar{y})$.
      return TQLSTEP($q', D, \varphi', t$)
```

To define the inlining translation INLTQL, we simply need a Skolem term for the root of the tree, as the basis for the inductive procedure TQLSTEP.

```
Procedure INLTQL($Q$, $D$)
  Input : A TQL query $Q = r[q]$ and a DTD $D$.
  Output: A set of views over INLSCHEMA($D_Q$).

  Create a 0-ary function $f_r$.
  return TQLSTEP($Q, D, \emptyset, f_r()$)
```

A TQL query Q is compatible with a DTD D if all the patterns used in Q are compatible with D. The following proposition shows that INLTQL satisfies an analog of **requirement 3** for queries that outputs trees.

Proposition 6.5: Given a DTD D, a TQL query Q compatible with D, and and a tree T that conforms to D, we have that INLDOC$(Q(T), D_Q) =$ INLTQL$(Q, D)($INLDOC$(T))$, up to renaming of nulls.

That is, the set of views INLTQL(Q, D) applied to the inlining of T yields the same answer as the inlining of $Q(T)$.

Translating relations back into XML

To complete the translation, we need an algorithm to publish back the relational data as an XML document. This is done by means of the algorithm PUBREL. We say that an instance I of INLSCHEMA(D) *D-represents* a tree T that conforms to D if $I = \text{INLDOC}(T, D)$.

```
Procedure PUBREL(D,I)
Input  : A DTD $D$ and an instance $I$ that $D$-represents some tree.
Output: An XML tree $T$ that is $D$-represented by $I$.

for each node $\ell$ of $G(D)$, traversed as Depth-first-search
do
    for each tuple $t$ of $R_\ell$ in $I$ with elements $n$, $\bar{a}$ and $n'$
    corresponding to attributes $id_n$, $A_D(\ell)$ and $id_{\mu(n)}$.
    do
        Add to $T$ a node $n$ labelled $\ell$, with attributes $\bar{a}$,
        whose parent is $n'$ (no parent if $\ell = r$);
        for every non-starred node $\ell'$ of $G(D)$ such that
        $\mu(\ell') = \ell$, and elements $n''$ and $\bar{b}$ in $t$
        corresponding to attributes $id_{\ell'}$ and $A_D(\ell')$ do
            Create a node $n''$ in $T$ labelled $\ell'$, with
            attributes $\bar{b}$, in a parent-child scheme that
            resembles $G(D)$.
        endfor
    endfor
endfor
return $T$
```

This algorithm will only work for relational instances that represent shredded documents. The following proposition shows its correctness.

Proposition 6.6: Given a DTD D and a relational instance I of INLSCHEMA(D), it is the case that $\text{INLDOC}(\text{PUBREL}(D, I)) = I$.

C. TQL queries in XML data exchange

Combining the previously mentioned result in [11] with the correctness of the algorithms we presented we conclude that **requirements 1-5** are satisfied for data exchange with XML-to-XML queries:

Theorem 6.7: 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 TQL query Q, the certain answers of Q for T under $\mathcal{M}$ and the certain answers of $\text{INLTQL}(Q, D_T)$ for $\text{INLDOC}(T, D_S)$ over $\text{INLMAP}(\mathcal{M}, D_S, D_T)$ coincide:

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

Remark: The notion of certain answers naturally (component-wise) extends to queries computing multiple relations.

Theorem 6.7 and Proposition 6.6 give us a way of computing $\text{CERTAIN}_{\mathcal{M}}(Q, T)$. First, compute $\text{CERTAIN}_{\text{INLMAP}(M)}(\text{INLTQL}(Q, D_T), \text{INLDOC}(T, D_S))$ by materializing views $\text{INLTQL}(Q, D_T)$ over the canonical solution for $\text{INLDOC}(T, D_S)$ and then use the procedure PUBREL to output it as the tree $\text{CERTAIN}_{\mathcal{M}}(Q, T)$.

VII. Adding XML constraints

So far, we assumed that target schemas consist of DTDs only; now we extend them with *target constraints*. Constraints have been studied and used extensively in the XML context. Analogs of keys and foreign keys such as `ID` and `IDREF` are very common. Thus, it is natural to ask whether our procedures continue to work when target schemas are augmented with such constraints. Here we look at *keys* and *foreign keys* that naturally extend the functionality of `ID` and `IDREF`:

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

We now show how to translate XML keys and foreign keys into relational integrity constraints in a way that preserves the satisfaction of the key requirements. Recall that we use the 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 $\text{INLCONSTR}(\Delta, D) = \emptyset$
for each key $@a \to \ell$ in $\Delta$:
    add to $\text{INLCONSTR}(\Delta, D)$ the key $@a \to R_\ell$ if $\ell$ is
    marked, or the key $@a \to R_{\mu(\ell)}$ if $\ell$ is not marked.
endfor
for each foreign key $\ell_1[@a] \subseteq_{FK} \ell_2[@b]$ in $\Delta$
    Add to $\text{INLCONSTR}(\Delta, D)$ the foreign key
    $R_{\ell_1}[@a] \subseteq_{FK} R_{\ell_2}[@b]$, replacing $R_{\ell_i}$ for $R_{\mu(\ell_i)}$ if $\ell_1$
    or $\ell_2$ are not marked.
endfor
return $\text{INLCONSTR}(\Delta, D)$
```

Using INLCONSTR, we extend the procedure INLMAP for the case of schema mappings with *target* constraints Δ_T in a way that retains its good properties. This is formalized in the procedure EXTINLMAP below.

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

return $\text{INLMAP}(\mathcal{M}, D_S, D_T)$, and the set of constraints
$\text{INLCONSTR}(\Delta_T, D_T)$
```

Proposition 7.1: For XML data-exchange settings that include a set Δ of XML keys and foreign keys, the extensions of procedure InlMap and InlQuery using $\textsc{InlConstr}(\Delta, D)$ satisfy our **Requirement 4** and **Requirement 5**, respectively.

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 [12].

VIII. 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 $\textsc{InlMap}(\mathcal{M})$ generates a universal solution R'_{univ}. This solution is a shredding, via InlDoc, of a universal XML solution, and also conforms to the shredding of source DTD. Finally, we apply the standard technique [13] for evaluating queries in relational data exchange to the query translation $\textsc{InlQuery}(Q)$ or $\textsc{InlTQL}(Q)$ to produce the correct answers, in the latter case with the possibility of using PubRel to publish back the results into XML.

We finish with a remark about the possibility of allowing operators ? and + in DTDs, as well as a choice operator for representing multiple choices. We say that a non-recursive DTD D is an *extended nested relational* DTD if all rules of D are of the form $\ell \to \tilde{\ell}_0 \dots \tilde{\ell}_m$, or $\ell \to \ell_0 + \dots + \ell_m$, where all the ℓ_i's and $\tilde{\ell}_i$'s are distinct, and each $\tilde{\ell}_i$ is one of ℓ_i, $\ell_i?$, ℓ_i^* or ℓ_i^+ (as usual $\ell?$ stands for $\ell|\epsilon$ and ℓ^+ for $\ell\ell^*$).

The procedure InlSchema can be extended to these DTDs. For each element ℓ that is under the operator ?, the transformation creates a special relation ℓ that references the relation of the nearest appropriate ancestor of ℓ. Furthermore, the transformation for a rule of the form $\ell_1 \to \ell_2^+$ can be defined by including a dependency that ensures that there is at least one tuple in the relation R_{ℓ_2} for each tuple in R_{ℓ_1}. Finally, for the choice operator $\ell \to \ell_0 + \dots + \ell_m$ the transformation would create one relation R_ℓ for each possible choice of $\ell_0, \dots, \ell_m$. Then, it is possible to extend all the procedures in a way that still satisfies **requirements 1-5** under extended nested relational DTDs.

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] C. David, L. Libkin, F. Murlak. Certain answers for XML queries. In *PODS 2010*, pages 191-202.

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

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

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

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

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

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

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

[19] M. Hernández, H. Ho, L. Popa, A. Fuxman, R. Miller, T. Fukuda, P. Papotti. Creating nested mappings with Clio. In *ICDE 2007*.

[20] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. Lakshmanan, A. Nierman, S Paparizos, J. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, C. Yu. TIMBER: A native XML database. *VLDB J.* 11 (2002).

[21] N. Klarlund, T. Schwentick, D. Suciu. XML: model, schemas, types, logics, and queries. In *Logics for Emerging Appl. of Databases 2003*.

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

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

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

[25] L. Lakshmanan, G. Ramesh, H. Wang, Z. Zhao. On testing satisfiability of tree pattern queries. *VLDB 2004*, pages 120–131.

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

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

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

[29] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D.J. DeWitt, J.F. Naughton. Relational databases for querying XML documents: limitations and opportunities. In *VLDB 1999*, pages 302-314.

[30] J. Shanmugasundaram, E. Shekita, J. Kiernan, R. Krishnamurthy, S. Viglas, J. Naughton, and I. Tatarinov. 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] XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery.

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

[34] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman. On supporting containment queries in relational database management systems. In *SIGMOD'01*, pages 425–436.

APPENDIX

A. Preliminary Definitions

Below are some technical definitions that will be used through the remainder of the appendix.

Homomorphisms and tree homomorphisms: Let K_1 and K_2 be instances of the same schema $\mathbf{R}$. A *homomorphism* h from K_1 to K_2 is a function h defined from the domain of K_1 to the domain of K_2 such that: (1) $h(c) = c$ for every constant element c in K_1, and (2) for every $R \in \mathbf{R}$ and every tuple $\bar{a} = (a_1, \ldots, a_k)$ in the relation R in K_1, it holds that $h(\bar{a}) = (h(a_1), \ldots, h(a_k))$ belongs to the relation R in K_2. Notice that this definition of homomorphism slightly differs from the usual one, as the additional constraint that homomorphisms are the identity on the constants is imposed.

Given a conjunctive query $Q(\bar{x})$ over a schema $\mathbf{R}$, we denote by $I_{Q(\bar{x})}$ the instance of $\mathbf{R}$ constructed as follows: for every relational symbol $R \in \mathbf{R}$ and relational atom $R(\bar{b})$ occurring in $Q(\bar{x})$, we include tuple $\bar{b}$ in the relation R of $I_{Q(\bar{x})}$. We define all variables in $\bar{x}$ to be constant elements in $I_{Q(\bar{x})}$, whereas every existentially quantified variable of Q is a null element.

It is now straightforward to prove the following lemma:

Lemma 1.1: Let I be an instance of schema $\mathbf{R}$, and Q a conjunctive query.Then, a tuple $\bar{a}$ of constant elements belongs to the evaluation of Q over I if and only if there is a homomorphism from $I_{Q(\bar{a})}$ to I.

We also need to introduce the equivalent definition of homomorphisms for XML trees, or *tree homomorphism* [6]. Let $T = (N, G)$ and $T' = (N', G')$ be XML trees, let n_r and n'_r be the roots of T and T', respectively, and let $Str(T) = \{s \in Str \mid \text{there exists } n \in N \text{ and } @a \in Att \text{ such that } \rho_{@a}(n) = s\}$, $Str(T')$ defined correspondingly. Then, $h : N \cup Str(T) \to N' \cup Str(T')$ is a homomorphism from T to T', if:

- for every $n \in N$, $h(n) \in N'$;
- for every constant element $s \in Str(T)$, $h(s) = s$, and for every null $s \in Str(T)$, $h(s) \in Str(T')$;
- $h(n_r) = n'_r$;
- for every $n_1, n_2 \in N$, if $G(n_1, n_2)$, then $G'(h(n_1), h(n_2))$;
- for every $n \in N$, $\lambda_T(n) = \lambda_{T'}(h(n))$; and
- for every $n \in N$ and $@a \in Att$ such that $\rho_{@a}(n)$ is defined, $h(\rho_{@a}(n)) = \rho_{@a}(h(n))$.

Given a tree pattern $\pi(\bar{x})$, we construct the tree $T_{\pi(\bar{x})}$ inductively: if $\pi(\bar{x}) = \ell(\bar{x})[\pi_1(\bar{x}_1), \ldots, \pi_k(\bar{x}_k)]$, then the root of $T_{\pi(\bar{x})}$ is a node labelled ℓ, with attributes $\bar{x}$, and k children corresponding to $T_{\pi_1(\bar{x}_1)}, \ldots, T_{\pi_k(\bar{x}_k)}$. As for the relational case, it is easy to prove the following lemma:

Lemma 1.2: Let T be an XML tree, $\pi(\bar{x})$ a tree pattern, and s a tuple of values in Str. Then, $\bar{s} \in \pi(T)$ if and only if there is a homomorphism from $T_{\pi(\bar{s})}$ to T.

Universal Solutions: By means of homomorphisms, we give a precise definition of universal solutions in relational or XML data exchange settings. Formally, let $(\mathbb{S}, \mathbb{T}, \mathcal{M})$ be a relational data exchange setting. Then, given an instance I of $\mathbb{S}$, we say that an $\mathcal{M}$-solution J for I is an $\mathcal{M}$-universal solution for I if for every other $\mathcal{M}$-solution J' for I, there exists an homomorphism from J to J' [13]. The definition for the case of XML data exchange setting is analogously formulated using the notion of tree homomorphism [6].

B. Proof of Proposition 3.2

Let D be a DTD over a set of element types El. Notice that all the foreign key constraints created with the procedure $\text{INLSCHEMA}(D)$ are of the form $R_\ell[id_{\mu(\ell)}] \subseteq_{FK} R_{\mu(\ell)}[id_{\mu(\ell)}]$, for some marked label $\ell \in El$; that is, each relation R_ℓ references the relation $R_{\mu(\ell)}$ that corresponds to the *nearest appropriate ancestor* of ℓ. Thus, the graph associated with the constraints of $\text{INLSCHEMA}(D)$ only contains edges from the attribute $id_{\mu(\ell)}$ of relation R_ℓ to attribute $id_{\mu(\ell)}$ relation $R_{\mu(\ell)}$. The proof then follows from the fact that $G(D)$ is acyclic, and thus the labels of D cannot form a cycle of nearest appropriate ancestors. □

C. Proof of Proposition 3.4

Let D and T as stated in the Proposition, and (S_D, Δ_D) be the output of $\text{INLSCHEMA}(D)$. That $\text{INLDOC}(T, D)$ satisfies the key constraints of Δ_D is trivial since the identifier of each node in T is unique. Same applies for the dependency stating the uniqueness of the root; since T conforms to D, the root of T (and only the root) must be labelled r. Moreover, for each foreign key in Δ of the form $R_\ell[id_{\mu(\ell)}] \subseteq R_{\mu(\ell)}[id_{\mu(\ell)}]$, notice that, since $G(D)$ is a tree, for each $\ell \in El - \{r\}$, there is exactly one element ℓ' such that $\ell' = \mu(\ell)$. Since T conforms to D, every ℓ-labelled node in T must be a descendant of an ℓ' labelled node. This guarantees that the interpretation of relations R_ℓ and $R_{\ell'}$ in $\text{INLDOC}(T, D)$ satisfy the constraint $R_\ell[id_{\mu(\ell)}] \subseteq R_{\mu(\ell)}[id_{\mu(\ell)}]$; each tuple in the interpretation of R_ℓ over $\text{INLDOC}(T, D)$ corresponds to a node n in T that must be a descendant of an ℓ' labelled node n' in T, and thus there must be a tuple in the interpretation of $R_{\ell'}$ identified with the element $id_{n'}$. □

D. Proof of Proposition 5.1

To prove that

$$\pi(T) \subseteq \text{INLPATTERN}(\pi, D)\big(\text{INLDOC}(T, D)\big),$$

let $\pi(\bar{x})$, D and T as defined, so that T conforms to D. Assume now that $\bar{a}$ is a tuple of attribute values such that $\bar{a} \in \pi(T)$, and let h be the homomorphism from $T_{\pi(\bar{a})}$ to T (by lemma 1.2, h is guaranteed to exist).

We now show how to construct a homomorphism g from $I_{\text{INLQUERY}(\pi,D)(\bar{a})}$ to $\text{INLDOC}(T, D)$ (this, by Lemma 1.1, suffices for the proof). Define g as follows:

- For each variable of the form id_v in $\textsc{InlPattern}(\pi, D)(\bar{a})$, where v is a node of $T_{\pi(\bar{a})}$, define $g(id_v) = id_{h(v)}$,
- for each $a \in \bar{a}$, let $g(a) = h(a)$, and
- for each other existentially quantified variable z in $\textsc{InlPattern}(\pi, D)(\bar{a})$ not of form id_v, assume that z belongs to a predicate $R_\ell(\bar{z})$ in $\textsc{InlPattern}(\pi, D)(\bar{a})$. Let id_v be the variable in predicate $R_\ell(\bar{z})$ that corresponds to the position of the attribute id_ℓ of relation R_ℓ, and assume that $h(v) = n$, for some node $n \in T$. Then, as defined in the previous item, $g(id_v) = id_n$. From the definition of the inlining procedure, we know that $\textsc{InlDoc}(T, D)$ contains a fact (and only one, since the attibute id_ℓ is key for the relation R_ℓ) of the form $R_\ell(id_n, \bar{b})$, for some tuple $\bar{b}$ of elements. Define g so that it maps the variable z to the element in the position of $(id_n, \bar{b})$ that corresponds to the position that z occupies in the predicate $R_\ell(\bar{z})$ in $\textsc{InlPattern}(\pi, D)(\bar{a})$.

We first show that g is well defined. First, it is easy to see that g is defined for every element of $I_{\textsc{InlPattern}(\pi,D)(\bar{a})}$. We now prove that there is no element in $I_{\textsc{InlPattern}(\pi,D)(\bar{a})}$ that is mapped by g to two different values in $\textsc{InlDoc}(T, D)$. To see this, assume for the sake of contradiction that there is an element x in $I_{\textsc{InlPattern}(\pi,D)(\bar{a})}$ such that g is defined to map x to two elements of $\textsc{InlDoc}(T)$. Then, there are three cases to consider:

- x cannot be a variable in $\textsc{InlPattern}(\pi, D)(\bar{a})$ of the form id_v for some node v of $T_{\pi(\bar{a})}$, since we have defined x to be mapped to $id_{h(v)}$ only;
- x cannot belong to $\bar{a}$, since we have defined every $a \in \bar{a}$ to be mapped only to $h(a)$;
- then, x is an existentially quantified variable in $\textsc{InlPattern}(\pi, D)(\bar{a})$ that is not of form id_v (that is, it is a fresh variable generated by the procedure $\textsc{InlPattern}$). But notice then that x belongs only to one predicate of $\textsc{InlPattern}(\pi, D)(\bar{a})$. Moreover, as explained in the definition of g, there is only one tuple in $\textsc{InlDoc}(T, D)$ to where x is being mapped.

We now prove that g is indeed a valid homomorphism. First, it is easy to see that for every $a \in \bar{a}$, $g(a) = a$. This follows since we have defined $g(a)$ as $h(a)$, and from the construction of $T_{\pi(\bar{a})}$, every $a \in \bar{a}$ is a constant, and thus $h(a) = a$.

Consider now a fact of the form $R_\ell(\bar{w})$ in $I_{\textsc{InlPattern}(\pi,D)(\bar{a})}$. We need to show that $R_\ell(g(\bar{w}))$ belongs to $\textsc{InlDoc}(T, D)$. We will assume for the sake of readability that $\ell \neq r$. The proof can be easily adapted for the case when $\ell = r$. From the inlining procedure for queries, there must be a node v of $T_{\pi(\bar{a})}$ such that $\textsc{InlPattern}$ adds to $\textsc{InlPattern}(\pi, D)(\bar{a})$ some existential quantification of the predicate $R_\ell(\bar{w})$ in the step that corresponds to v (that is, $R_\ell(\bar{w})$ is part of $Q(\bar{a}_v)$). We have two cases. Assume first that v is marked. Then,

$$Q_v(\bar{a}_v) \quad = \quad \exists id_v \exists id_{\mu(v)} \exists \bar{z} R_\ell(id_v, \bar{a}_v, id_{\mu(v)}, \bar{z}),$$

where $\bar{z}$ is a tuple of fresh variables not used elsewhere in $\textsc{InlPattern}(\pi, D)(\bar{a})$ and the position of the variables id_v, $\bar{a}_v$ and $id_{\mu(v)}$ coincide with the attributes id_ℓ, $A_D(\ell)$ and $id_{\mu(\ell)}$ in *attr*(R_ℓ).

Further, we now that the homomorphism h maps the node v of $T_{\pi(\bar{a})}$ to some node $h(v)$ in T, and thus, from the properties of tree homomorphisms, we also know that $h(v)$ has the element type ℓ, and that for every $a \in a_v$ and $@a \in Att$, if $\rho_{@a}(v) = a$, then $\rho_{@a}(h(v)) = a$. Moreover, since homomorphisms must preserve the child relation, it is easy to see that the nearest appropriate ancestor of $h(v)$ in T must be $h(\mu(v))$. Then, it is clear that $\textsc{InlDoc}(T, D)$ must contain a tuple of the form $R_\ell(id_{h(v)}, \bar{a}_v, id_{h(\mu(v))}, \bar{b})$, for some tuple $\bar{b}$ of elements, and where the positions of $\bar{a}_v$ correspond to the attributes in $A_D(\ell)$ of *attr*(R_ℓ) where $\rho(v)$ is defined. From the definition of g, it is clear that $g(id_v, \bar{a}_v, id_{\mu(v)}, \bar{z})$ is the tuple $id_{h(v)}, id_{h(\mu(v))}, \bar{a}_v, g(\bar{z})$. The proof then follows since $g(\bar{z})$ is defined to be $\bar{b}$.

Second, assume that v is not marked, and that $\lambda(v) = \ell$, $\mu(v)$ in $T_{\pi(\bar{a})}$ is the node v', and $\lambda(v') = \ell'$. Then, as defined, the query $Q_v(\bar{a}_v)$ is of form:

$$Q_v(\bar{a}_v) \quad = \quad \exists id_{v'} \exists id_{\mu(v')} \exists id_v \exists \bar{z} R_{\ell'}(id_{v'}, id_{\mu(v')}, id_v, \bar{a}_v, \bar{z}),$$

where $\bar{z}$ is a tuple of fresh variables not used elsewhere in $\textsc{InlPattern}(\pi, D)(\bar{a})$, and the position of the variables $id_{v'}$ id_v, $id_{\mu(v')}$ and $\bar{a}_v$ is consistent with the attributes $id_{\ell'}$, id_ℓ, $id_{\mu(\ell')}$ and $A_D(\ell)$ in *attr*$(R_{\ell'})$.

Further, we now that the homomorphism h maps the nodes v and v' of $T_{\pi(\bar{x})}$ to some nodes $h(v)$ and $h(v')$ in T. Then, from the properties of tree homomorphisms, we obtain that λ assigns the types ℓ and ℓ' to $h(v)$ and $h(v')$, respectively, and that for every $a \in a_v$ and $@a \in Att$, if $\rho_{@a}(v) = a$, then $\rho_{@a}(h(v) = a$. Moreover, since homomorphisms preserves the child relation, it is easy to see that $h(v')$ must be the nearest appropriate ancestor of $h(v)$ in T, and that the nearest appropriate ancestor of $h(v')$ must be $h(\mu(v'))$. Then, it is clear that the inlining of T must contain a tuple of the form $R_{\ell'}(id_{h(v')}, id_{h(\mu(v'))}, id_{h(v)}, \bar{a}_v, \bar{b})$ for some tuple $\bar{b}$ of elements, where the positions of $\bar{a}_v$ correspond to the attributes in $A_D(\ell)$ such that $\rho(v)$ is defined. Again, the proof follows since we have defined $g(\bar{z})$ as $\bar{b}$.

For the proof that

$$\textsc{InlPattern}(\pi, D)\big(\textsc{InlDoc}(T, D)\big) \subseteq \pi(T),$$

assume that for a tuple $\bar{a}$ of constants there is an homomorphism h from $I_{\textsc{InlPattern}(\pi,D)(\bar{a})}$ to $\textsc{InlDoc}(T, D)$. We construct a homomorphism g from $T_{\pi(\bar{a})}$ to T. By Lemma 1.2, this suffices for the proof.

Define g as follows:

- For every node v of $T_{\pi(\bar{a})}$, consider the variable id_v defined in the procedure $\textsc{InlDoc}$, and assume that $h(id_v) = id_n$, for some element id_n of $\textsc{InlDoc}(T, D)$. Define $g(v) = n$. Notice that this is well defined: from the definition of $\textsc{InlDoc}$, and the properties of homomorphisms, we know that n must be a node of T (both id_v and id_n occur in a position of the predicates that corresponds to the identifiers of the nodes in the schema $\text{InlSchema}(D)$.

- For every $s \in \mathit{Str}(T_{\pi(\bar{a})})$, let v be the node of $T_{\pi(\bar{a})}$ such that $s = \rho_{@a}(v)$. Then, notice that from the definition of the translation of patterns, s must be a free variable of the query Q_v in $\text{InlPattern}(\pi, D)$, and thus $I_{\text{InlPattern}(\pi,D)}$ contains the variable s. Define $g(s) = h(s)$

We now prove that g is a valid homomorphism from $T_{\pi(\bar{a})}$ to T.

First, as mentioned in the definition of g, it is clear that $g(v) \in N$, for every $v \in T_{\pi(\bar{a})}$.

Second, we prove that, if v is the root of $T_{\pi(\bar{a})}$, then $g(v) = n_r$, where n_r is the root of T. This follows since π is fully specified, π must be of form $r(\bar{a})[\lambda]$. Then, the variable id_v must be mentioned in a predicate of R_r of $\text{InlPattern}(\pi, D)$. Since h is an homomorphism, $h(id_v)$ must belong to a tuple in R_r. It follows from the construction of $\text{InlSchema}(D)$ and proposition 3.4 that it must be the (unique) identifier of R_r, and thus the identifier of the root node of T.

Next, we prove that for every node v of $T_{\pi(\bar{a})}$, $\lambda_{T_{\pi(\bar{a})}}(v) = \lambda_T(g(v))$. Assume that for a node v in $T_{\pi(\bar{a})}$ it is the case that $\lambda_{T_{\pi(\bar{a})}}(v) = \ell$. There are two cases. The case when v is marked follows from the fact that there must be a tuple in the interpretation of the relation R_ℓ in $\text{InlDoc}(T, D)$ that contains $h(id_v)$ in its id_ℓ-attribute. Then, since g maps v to the node in T that corresponds to $h(id_v)$ in $\text{InlDoc}(T, D)$, it must be the case that $\lambda_T(g(v)) = \ell$. If v is not marked, let ℓ' be the nearest appropriate ancestor of ℓ, and consider the tuple in the interpretation of relation $R_{\ell'}$ in $\text{InlDoc}(T, D)$ that contains the element id_v in the position that corresponds to the attribute id_ℓ. The proof follows easily using the same argument as for the other case.

Assume now that two nodes v_1, v_2 of $T_{\pi(\bar{a})}$ are such that v_2 is a child of v_1 in $T_{\pi(\bar{a})}$.Let $\ell_1 = \lambda(v_1)$ and $\ell_2 = \lambda(v_2)$, and assume that $h(id_{v_1}) = id_{n_1}$ and $h(id_{v_2}) = id_{n_2}$, for some nodes n_1, n_2 of T. Thus, $g(v_1) = n_1$, and $g(v_2) = n_2$. The proof that $g(v_2)$ is a child of $g(v_1)$ follows easily from the fact that g preserves the labelling of the nodes, the graph $G(D)$ is a tree, π is compatible with D and and T conforms to D: If v_2 is a child of v_1 in $T_{\pi(\bar{a})}$, then it must be that $\ell_1 \in P_D(\ell_1)$, and that ℓ_1 does not appear in the production of any other label in D. Then, since $\lambda_T(n_2) = \ell_2$ and $\lambda_T(n_1) = \ell_1$ and T conforms to D, it must be that n_2 is a child of n_1.

Next, it is easy to see that for every $s \in \mathit{Str}(T_{\pi(\bar{a})})$, $g(s) \in \mathit{Str}(T)$. Moreover, since we have defined $g(s) = h(s)$, we also have that that $g(s) = s$ for every constant s.

Finally, we prove that for every node v of $T_{\pi(\bar{a})}$ and $@a \in \mathit{Att}$ such that $\rho_{@a}(v)$ is defined, $g(\rho_{@a}(v)) = \rho_{@a}(g(v))$. Assume that for a node v of $T_{\pi(\bar{a})}$ and for an attribute $@a \in \mathit{Att}$, it is the case that $\rho_{@a}(v)) = s$. We must prove that $g(s) = \rho_{@a}(g(v))$. But we have defined $g(s) = h(s)$, and thus, we need to prove that $h(s) = \rho_{@a} g(v)$. Assume first that v is marked. Then, notice that s is the variable in the position corresponding to $@a$ in $\mathit{attr}(R_{\lambda(v)})$ in the predicate of $\text{InlPattern}(\pi, D)$ added in the step corresponding to Q_v. Thus, from the properties of relational homomorphisms, s must belong to the tuple in $R_{\lambda(v)}$ in $\text{InlDoc}(T, D)$ that contains $h(id_v)$ in its first position. Since g maps v to the node in T identified by $h(id_v)$, it must be the case that $\rho_{@a}(g(v)) = h(s)$. For the case where v is not marked, consider the nearest appropriate ancestor of v in $T_{\pi(\bar{a})}$, and let v' be such node. Notice that since g preserves the child relation, $g(v')$ is the nearest appropriate ancestor of $g(v)$. The proof then follows by considering the attribute corresponding to $@a$ in $A_D(\ell)$ in the relation $R_{\ell'}$, where $\ell' = \lambda(v')$ and then using the same argument than in the previous case. $\square$

By combining this results with Lemmas 1.1 and 1.2, it is not difficult to obtain the following corollary:

Corollary 1.3: Let D be a DTD, T an XML document that conforms to D, and π a pattern compatible with D. In addition, let $\bar{a}$ be a tuple of elements and variables. Then, there exists an homomorphism from $T_{\pi(\bar{a})}$ to T if and only if there is an homomorphism from $I_{\text{InlPattern}(\pi,D)(\bar{a})}$ to $\text{InlDoc}(T, D)$.

Moreover, it is not difficult to adapt this proof to show the following:

Lemma 1.4: Let D be a DTD, and T_1, T_2 two trees that conform to D. There is a tree homomorphism from T_1 and T_2 if and only if there is a homomorphism from $\text{InlDoc}(T_1, D)$ to $\text{InlDoc}(T_2, D)$

E. Proof of Theorem 5.2

Fix a DTD D and a tree T. The proof is done by induction.

We have already proved the base case with the proof of Proposition 5.1.

For the induction step, assume first that Q is of form $\exists z Q_1(\bar{x}, \bar{z})$, and that $Q_1(T) = \text{InlQuery}(Q_1, D)(\text{InlDoc}(T, D))$. It is now easy to see that $Q(T) = \text{InlQuery}(Q, D)(\text{InlDoc}(T, D))$: Assume first that a tuple $\bar{a}$ belongs to $Q(T)$. Then, there must be a tuple $\bar{z}$ of variables such that $(\bar{a}, \bar{z})$ belongs to $Q_1(T)$. Thus, from the inductive hypothesis, we obtain that $(\bar{a}, \bar{z})$ belong to the evaluation of $\text{InlQuery}(Q_1, D)(\bar{a}, \bar{z})$ over $\text{InlDoc}(T, D)$. It follows that $(\bar{a}, \bar{z})$ belong to the evaluation of $\text{InlQuery}(Q, D)(\bar{a}, \bar{z})$ over $\text{InlDoc}(T, D)$, since the algorithms defines $\text{InlQuery}(Q, D) = \exists \bar{z} \text{InlQuery}(Q_1, D)$. The other direction is analogous.

Next, assume that $Q = Q_1(\bar{x}_1) \wedge Q_2(\bar{x}_2)$, and that $Q_1(T) = \text{InlQuery}(Q_1, D)(\text{InlDoc}(T, D))$ and $Q_2(T) = \text{InlQuery}(Q_2, D)(\text{InlDoc}(T, D))$. The argument is similar to the previous case: assume first that a tuple $\bar{a}$ belongs to $Q(T)$. Then, there must be sub tuples $\bar{a}_1$, $\bar{a}_2$ of $\bar{a}$ such that $(\bar{a}_1)$ and $(\bar{a}_2)$ belong to $Q_1(T)$ and $Q_2(T)$, respectively. We obtain that $(\bar{a}_1)$ and $(\bar{a}_2)$ belong to the evaluation of $\text{InlQuery}(Q_1, D)$ and $\text{InlQuery}(Q_2, D)$ over $\text{InlDoc}(T, D)$, and thus, since $\text{InlQuery}(Q, D) = \text{InlQuery}(Q_1, D) \wedge \text{InlQuery}(Q_2, D)$, $\bar{a}$ belongs to the evaluation of $\text{InlQuery}(Q, D)$ over T. The other direction is also analogous.

F. Proof of Theorem 5.3

For this proof, we first provide a key lemma. Let $\mathcal{M} = (D_S, D_T, \Sigma)$ be an XML schema mapping, T be an XML tree that conforms to D_S, and J an $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$-solution for $\textsc{InlDoc}(T, D)$. For a relation R_ℓ of $\textsc{InlSchema}(D_T)$, we denote all the positions that correspond to an attribute id_ℓ or $id_{\mu(\ell)}$ of R_ℓ as the *identifier positions* of R_ℓ. Moreover, an element a in a tuple t in the interpretation of R_ℓ in J is an *identifier element* if it occupies an identifier position in t. We also define the *attribute* positions of a relation R_ℓ as the positions that correspond to attributes of ℓ or of $\ell' \mid \mu(\ell') = \ell$ in D, and define the notion of an *attribute element* as expected. We now present the lemma:

Lemma 1.5: Let $\mathcal{M} = (D_S, D_T, \Sigma)$ be an XML schema mapping, and T be an XML tree that conforms to D_S. Moreover, let J be an $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$-solution for $\textsc{InlDoc}(T, D)$ such that (1) every identifier element in J does not appear in two identifier positions in two (not necessarily different) tuples, and (2) no identifier element is also an attribute element. Then, there exists a tree T' such that $\textsc{InlDoc}(T', D_T) \subseteq J$, and such that T' is an $\mathcal{M}$-solution for T.

Lemma 1.5 formalizes the intuition that this class of "well behaved" $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$-solutions contains the correct representation of a shredded tree. The proof of this Lemma constructs from J a correct tree representation, in which each identifier element in J represents a node of the tree T' such that $\textsc{InlDoc}(T', D_T) \subseteq J$. We leave the details since the proof is lengthy and straightforward.

We now prove the theorem.

Part a: Let $\mathcal{M} = (D_S, D_T, \Sigma)$ be an XML schema mapping, and T an XML document that conforms to D_S. Consider an arbitrary $\mathcal{M}$-universal solution T' for T. We need to show that $\textsc{InlDoc}(T', D_T)$ is an $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$-universal solution for T. This is split into two parts, proving first that $\textsc{InlDoc}(T', D_T)$ is a solution, and then that it is universal.

As stated, we first prove that $\textsc{InlDoc}(T', D_T)$ is an $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$-solution for $\textsc{InlDoc}(T, D_S)$. From Proposition (3.4), it is clear that $\textsc{InlDoc}(T', D_T)$ satisfies the dependencies in Δ_D. We now show that the pair $\big(\textsc{InlDoc}(D_S, T), \textsc{InlDoc}(D_t, T')\big)$ satisfy all the dependencies in Σ. Assume that for a dependency of the form

$$\textsc{InlPattern}(\pi(\bar{x}), D_S) \rightarrow \exists \bar{z} \textsc{InlPattern}(\pi'(\bar{x}, \bar{z}), D_T)$$

there is a tuple $\bar{t}_x$ such that $\textsc{InlDoc}(D_S, T) \models \textsc{InlPattern}(\pi(\bar{t}_x), D)$. From Proposition 5.1, it must be the case that $T \models \pi(\bar{t}_x)$. Thus, since T' is a solution for T, there must be a tuple $\bar{t}_z$ of constant and/or null elements such that $T' \models \pi'(\bar{t}_x, \bar{t}_z)$. Again, from Proposition 5.1, we obtain that $\textsc{InlDoc}(D_T, T') \models \textsc{InlPattern}(\pi'(\bar{t}_x, \bar{t}_z), D)$. This finishes the proof that $\textsc{InlDoc}(T', D_T)$ is an $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$-solution for $\textsc{InlDoc}(T, D_S)$.

We now prove that $\textsc{InlDoc}(T', D_T)$ is indeed universal. Assume for the sake of contradiction that it is not an universal solution, that is, there exists a solution J such that there does not exists an homomorphism from $\textsc{InlDoc}(T', D_T)$ to J. Construct from J a solution J' as follows: For each identifier position of every relation R_ℓ in $\textsc{InlSchema}(D_T)$, and each tuple in the interpretation of R_ℓ, replace each identifier element a of t with a fresh null element z_a. In this case, replace also each occurrences of a in the position $id_{\mu(\ell')}$ of tuples in the interpretation of relations $R_{\ell'}$ that reference R_ℓ in a constraint in $\textsc{InlSchema}(D_T)$, and replace each occurrence of a in an attribute position with a fresh null. It is easy to see that J' is an $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$-solution for $\textsc{InlDoc}(T, D_S)$ as well. In fact, since we have replaced each of those elements a with nulls in a "cascade" fashion, J' clearly satisfies all dependencies in $\textsc{InlSchema}(D_T)$. Furthermore, since each dependency in $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$ contains a different existentially quantified variable for each relation in its right-hand side, $(\textsc{InlDoc}(T, D_S), J')$ satisfy the dependencies in $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$. Finally, there is a homomorphism from J' to J: map each such z_a to the element a, and map each other element to itself. Thus, by composition of homomorphisms, there cannot exist an homomorphism from $\textsc{InlDoc}(T', D_T)$ to J', as this would imply the existence of an homomorphism from $\textsc{InlDoc}(T', D_T)$ to J. However, notice that J' satisfies the property of Lemma 1.5, since all identifying elements not satisfying it have been replaced by fresh new null elements. Let then $T_{J'}$ be the $\mathcal{M}$-solution for T such that $\textsc{InlDoc}(T_{J'}, D_T) \subseteq J'$ (Lemma 1.5 proves the existence of $T_{J'}$). Notice that, since $\textsc{InlDoc}(T_{J'}, D_T) \subseteq J'$, there also exists a homomorphism from $\textsc{InlDoc}(T_{J'}, D_T)$ to J'. Yet again, by composition of homomorphisms, we conclude that there cannot exist a homomorphism from $\textsc{InlDoc}(T', D_T)$ to $\textsc{InlDoc}(T_{J'}, D_T)$.

On the other hand, the XML tree T' is an $\mathcal{M}$-universal solution, and thus there is an homomorphisms from T' to $T_{J'}$. But then, by Lemma 1.4, there exists an homomorphism from $\textsc{InlDoc}(T', D_T)$ to $\textsc{InlDoc}(T_{J'}, D_T)$. This is a contradiction.

Part b: Assume that R is an $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$-universal solution for $\textsc{InlDoc}(T, D_S)$. For this proof we use the fact that $\textsc{CanSol}(\textsc{InlDoc}(T, D_S))$ satisfies the conditions of Lemma 1.5, that is, that every identifier element in $\textsc{CanSol}(\textsc{InlDoc}(T, D_S))$ does not appear in two tuples in two different identifier positions; this can be easily proved from the properties of the chase procedure (see [13]). Further, since R is universal, there must be an homomorphism from R to $\textsc{CanSol}(\textsc{InlDoc}(T, D_S))$, and thus it also must be the case that R satisfies the conditions of Lemma 1.5. Then, from Lemma 1.5, let T' be an $\mathcal{M}$-solution for T such that $\textsc{InlDoc}(T', D_T) \subseteq R$.

To prove that T' is an $\mathcal{M}$-universal solution for T, let T'' be an $\mathcal{M}$-solution for T, we need to prove that there is a homomorphism from T' to T''. From the part a) of this Theorem, $\textsc{InlDoc}(T'', D_T)$ is an $\textsc{InlMap}(\mathcal{M}, D_S, D_T)$-

solution for $\text{INLDOC}(T, D_S)$, and, since R is universal, there is a homomorphism h from R to $\text{INLDOC}(T'', D_T)$. Moreover, since $\text{INLDOC}(T', D_T) \subseteq R$, h is also a homomorphism from $\text{INLDOC}(T', D_T)$ to $\text{INLDOC}(T', D_T)$. Thus, from Lemma 1.4, there is a homomorphism from T' to T''. This concludes the proof. $\square$

G. Proof of Theorem 5.4

Assume first that a tuple $\bar{t}$ belongs to the certain answers of a query Q over a tree T under a mapping $\mathcal{M} = (D_S, D_T, \Sigma)$. Then, clearly, $\bar{t}$ belongs to the evaluation of Q over the canonical solution $\text{CANSOL}(T)$ for T (that, in this case, is guaranteed to exists [6]) under $\mathcal{M}$. Then, from proposition 5.2, $\bar{t}$ belongs to the evaluation of $\text{INLQUERY}(Q, D_T)$ over $\text{INLDOC}(\text{CANSOL}(T), D_T)$. Moreover, from proposition 5.3, $\text{INLDOC}(\text{CANSOL}(T), D_T)$ is an $\text{INLMAP}(\mathcal{M}, D_S, D_T)$-universal solution for $\text{INLDOC}(T, D_S)$. From results in [13], we obtain that $\bar{t}$ belongs to the certain answers of $\text{INLQUERY}(Q, D_T)$ over $\text{INLDOC}(T, D_S)$ under $\mathcal{M}$. The other direction is symmetrical. $\square$

H. Proof of Proposition 6.5

We begin by proving that $\text{INLTQL}(Q, D)(\text{INLDOC}(T)) \subseteq \text{INLDOC}(Q(T), D_Q)$. Let D_Q be the DTD corresponding to Q. Assume that there exists a tuple t that is part of a view V in $\text{INLTQL}(Q, D)$, with view V of form $R_\ell(f(\bar{x}), g(\bar{x}''), \bar{a}, \bar{x}') := \text{INLQUERY}(\varphi(\bar{x}), D)$ (we do not prove the case when $\ell = r$ since it is very similar). Thus, t must be of form $(c_{f(\bar{b})}, c_{g(\bar{c})}, \bar{a}, \bar{b}')$, where $\bar{c}$ and $\bar{b}'$ are contained in $\bar{b}$, and it must be the case that $\bar{b}$ belongs to $\text{INLQUERY}(\varphi, D)(\text{INLDOC}(T, D)$. By Theorem 5.2, $\bar{b}$ belongs to $\varphi(T)$. Denote by v the valuation that assigns $\bar{x}$ as $\bar{b}$ (and obviously $\bar{c}$ to $\bar{x}''$), and assume that the forest query that created view V in the inlining of Q is of form $\ell(\bar{a}, \bar{x}')[q'(\bar{x}'')]$. It can be proved by induction that $[\![q(\bar{x})]\!]_{T,v}$ must contain a node of form $\ell(\bar{a}, \bar{b}')[[\![q'(\bar{x}'']\!]_{T,v}]$. Thus, the inlining of $Q(T)$ must contain a tuple in R_ℓ of form $(id_n, id_{\mu(n)}, \bar{a}, \bar{b}')$; the proof follows by renaming as $c_{f(\bar{b})}$ and $c_{g(\bar{c})}$ the nulls id_n and $id_{\mu(n)}$, respectively. We only need to show that no null has to be renamed as two different constants. Assume now that there is a node n_1 in $Q(T)$ that requires to be named twice according to the above procedure. We consider all possible cases:

- For two tuples $(c_{f(\bar{b})}, c_{g(\bar{c})}, \bar{a}, \bar{b}')$, $(c_{f'(\bar{d})}, c_{g'(\bar{e})}, \bar{a}', \bar{d}')$, id_n must be renamed as $c_{f(\bar{b})}$ and $c_{f'(\bar{d})}$. But then, since every view V of $\text{INLTQL}(Q, D)$ is created with a different function symbol, it must be the case that $f = f'$ (if not, these two tuples correspond to a different forest query in $Q(T)$). Let now v_b and v_d be the valuation that maps $\bar{x}$ to $\bar{b}$ and $\bar{d}$, respectively. It follows that $\bar{b} = \bar{d}$, because n_1 cannot belong to both $[\![q(\bar{x})]\!]_{T,v_b}$ and $[\![q(\bar{x})]\!]_{T,v_d}$ at the same time.
- For two tuples $(c_{f(\bar{b})}, c_{g(\bar{c})}, \bar{a}, \bar{b}')$, $(c_{f'(\bar{d})}, c_{g'(\bar{e})}, \bar{a}', \bar{d}')$, id_n must be renamed as $c_{g(\bar{c})}$ and $c_{g'(\bar{e})}$. Let $q(\bar{x})$ and $q'(\bar{y})$ the forest queries that gave rise to the creation of functions f and f' in INLTQL. In this case n_1 must be the common ancestor of both $[\![q(\bar{x})]\!]_{T,v_b}$ and $[\![q'(\bar{x})]\!]_{T,v_d}$, and thus it follows that $g = g'$, because the same skolem term must have been passed on by algorithm INLTQL. Let now $q''(\bar{z})$ be the forest query that gave rise to the creation of function g in INLTQL, and thus v be the valuation such that n_1 is the root node of $[\![q''(\bar{z})]\!]_{T,v}$. According to our renaming procedure, both v_b and v_d must be extensions of v, and thus it must be that $\bar{c} = \bar{e}$, as they are both replaced under v
- For two tuples $(c_{f(\bar{b})}, c_{g(\bar{c})}, \bar{a}, \bar{b}')$, $(c_{f'(\bar{d})}, c_{g'(\bar{e})}, \bar{a}', \bar{d}')$, id_n must be renamed as $c_{f(\bar{b})}$ and $c_{g'(\bar{e})}$. Using the same arguments presented in the previous cases, we conclude that $f = g'$, the skolem terms that produced $f(\bar{b})$ and $g'(\bar{e})$ are the same, and that the valuation that assigns $\bar{b}$ to the free variables of the skolem term corresponding to $f(\bar{b})$ must then be an extension of valuation that assigns $\bar{d}$ to the skolem term of $f'(\bar{d})$, that assigns in turn $\bar{e}$ to the term corresponding to $g'(\bar{e})$; it must be that $\bar{b} = \bar{e}$. The last remaining case is completely symmetric.

Next, we show that $\text{INLDOC}(Q(T), D_Q) \subseteq \text{INLTQL}(Q, D)(\text{INLDOC}(T))$.

Since every element of D_Q is under a star, it is easy to see that relation R_ℓ will contain only attributes id_ℓ, $id_{\mu(\ell)}$ and $A_{D_Q}(\ell)$. We first rename all elements that are in a position corresponding to attributes id_ℓ as follows:

Let t be a tuple of relation R_ℓ in $\text{INLDOC}(Q(T), D_Q)$, and assume that id_n is the element that corresponds to attribute id_ℓ of R_ℓ. If $\ell = r$, rename id_n by the 0-ary term $f_r()$ used in procedure INLTQL. For the case when $\ell \neq r$, it is easy to see from the definition of the procedure INLDOC that $Q(T)$ must contain an ℓ-labelled node n. Thus, from the semantics of TQL queries, there must be a sub-forest q of Q of form $q(\bar{x}) = \ell(\bar{a}, v(\bar{x}'))[q'(\bar{x}'')]$ and a valuation v such that n is the top node of forest $[\![q(\bar{x}]\!]_{T,v}$. Let f be the function created by procedure INLTQL in the step corresponding to q. Finally, let $\pi_1(\bar{z}_1), \ldots, \pi_k(\bar{z}_k)$ be the sequence of patterns present in for-return constructs in Q from the root until q, and let $\bar{z}$ be the union of $\bar{z}_1, \ldots, \bar{z}_k$. Then, rename id_n as $c_{f(v(\bar{z}))}$. Notice that this procedure is well defined, since v must apply to each variable of $\bar{z}$.

Let us denote by J the instance resulting of renaming all elements of $\text{INLDOC}(Q(T), D_Q)$ accordingly. We show that $J \subseteq \text{INLTQL}(Q, D)(\text{INLDOC}(T, D))$, up to renaming of nulls in attribute positions (that is, nulls in positions $A_D(\ell)$ in tuples on R_ℓ.

Let t be a tuple of relation R_ℓ in J, and assume that the elements in t corresponding to attributes id_ℓ, $id_{\mu(\ell)}$ and $A_{D_Q}(\ell)$ are $c_{f(\bar{b})}, c_{g(\bar{b}')}, \bar{a}$.

We need to show that such tuple is in fact in $\text{INLTQL}(Q, D)(\text{INLDOC}(T))$. Let n and n' be the nodes in $Q(T)$ such that id_n and $id_{n'}$ where replaced by $c_{f(\bar{b})}$ and $c_{g(\bar{b}')}$, respectively, and $q(\bar{x}), q'(\bar{x}')$ the forest queries that give rise to the creation of f and respectively g by procedure INLTQL. Moreover, let $\varphi(\bar{z}) = \pi_1(\bar{z}_1), \ldots, \pi_k(\bar{z}_k)$ be the sequence of patterns present in for-return constructs in Q from the root until q, where $\bar{z}$ is the union of $\bar{z}_1, \ldots, \bar{z}_k$. In the same fashion, we select $\varphi'(\bar{z}') = \pi'_1, (\bar{z}'_1), \ldots, \pi'_{k'}(\bar{z}_{k'})$ and $\bar{z}'$ for forest query q'. As a remark, since n' is the parent of n, observe that each pattern π'_i corresponds to a pattern π_j, for some $j \leq k$. Finally, it is easy to see that there is no other query of form $\ell(\bar{y}, \bar{a})[q''(\bar{y}')]$ in between q and q'. Thus, the step of INLTQLcorresponding to $q(\bar{x})$ must have received the term $g(\bar{z}')$ as input.

By following these remarks, one notices that procedure INLTQL creates the following view V for the step of $q(\bar{x})$: $R_\ell(f(\bar{z}), g(\bar{z}'), \bar{d}, \bar{x}) := \text{INLQUERY}(\varphi, D)$.

All that remains to see is that, since $(T, v) \models \varphi(\bar{z})$, it must be that $\text{INLDOC}(T, D) \models \text{INLQUERY}(\varphi(v(\bar{z})), D)$. This ensures the existence of a fact of form $R_\ell(c_{f(v(\bar{z}))}, c_{g(v(\bar{z}'))}, \bar{d}, v(\bar{x})) = R_\ell(c_{f(\bar{b})}, c_{g(\bar{b}')}, \bar{a})$ in $\text{INLTQL}(Q, D)(\text{INLDOC}(T, D))$.

I. Proof of Proposition 6.6

Let T be a tree such that $\text{INLDOC}(T, D) = I$. We construct a mapping h between T and $\text{PUBREL}(I)$ as follows:

- For each node n of T that is marked, let ℓ be it's label, and id_n be the identifier of $I = \text{INLDOC}(T, D)$ that belongs to the attribute id_ℓ of the tuple t created by procedure INLDOC from node n. Then, define h so it maps n to the node of $\text{PUBREL}(I)$ created by procedure PUBREL from tuple t of R_ℓ.
- For each node n that is not marked, let $n' = \mu(n)$, and t the corresponding tuple in INLDOC that corresponds to node n'. Let ℓ and ℓ' be the label of n and n', respectively, and assume that id_n, $id_{n'}$ are the identifiers of t in positions id_ℓ and $id_{\ell'}$ of tuple t in R'_ℓ. Then, procedure PUBREL will create from t a node n'_t labelled ℓ' and a node n_t labelled with ℓ, such that $\mu(n_t) = n'_t$ in $\text{PUBREL}(I)$. Define h so it maps n to n_t.

It is clear that this mapping is one to one, since $I = \text{INLDOC}(T, D)$. Furthermore, since $G(D)$ is a tree, it is also clear that this mapping preserves the relation μ of nearest appropriate ancestors, as the way in which procedure PUBREL creates the parent-child relation of nodes is always unique. Finally, from the definition of procedures PUBREL and INLDOC it must be the case that for every n in T labelled ℓ, the set $\{\rho_{@a}(n) \mid @a \in A_D(\ell)\}$ is the same as $\{\rho_{@a}(h(n)) \mid @a \in A_D(\ell)\}$ in $\text{PUBREL}(I)$.

It is now an easy exercise to prove that INLDOC creates the same relations (up to renaming of nulls) for $\text{PUBREL}(I)$ and (T), since for every marked node n of T the procedure creates exactly the same tuple as marked node $h(n)$ of $\text{PUBREL}(I)$.

J. Proof of Theorem 6.7

Fix an M-universal solution T' for T. By theorem 6.1, $\text{CERTAIN}_{\mathcal{M}}(Q, T) = Q(T')$, where T' is an universal solution. Furthermore, by proposition 6.5, $\text{INLDOC}(Q(T'), D_Q) = \text{INLTQL}(Q, D_T)(\text{INLDOC}(T', D_T))$.

Finally, since the views created by the procedure INLTQL are essentially conjunctive queries using skolem terms, and (by theorem 5.3) $\text{INLDOC}(T', D_T)$ is an $\text{INLMAP}(M, D_S, D_T)$-universal solution for INLDOC, it can be proved that $\text{INLTQL}(Q, D_T)(\text{INLDOC}(T', D_T)) = \text{CERTAIN}_{\text{INLMAP}(M)}(\text{INLTQL}(Q, D_T), \text{INLDOC}(T, D_S))$, using standard tools from data exchange literature (see [13], [8]).

K. Proof of Theorem 7.1

In [6], a chase procedure was defined to compute the canonical universal solution for a tree T under an XML mapping $\mathcal{M}$. If we include a set of XML integrity constraints Δ_T in $\mathcal{M}$, it is possible to extend this chase procedure so that it correctly computes the canonical solution for T under the extension of $\mathcal{M}$, assuming that the constraints in Δ_T are acyclic (this restriction, as we have discussed, can be improved to consider more weaker notions of acyclicity [13], [22], [8]). Moreover, the procedure $\text{EXTINLMAP}(\mathcal{M}, D_S, D_T, \Delta_T)$ produces a mapping with acyclic relational constraints in the target schema if and only if Δ_T is acyclic. Thus, using this results, it is possible to adapt the proofs of theorems 5.3, 5.4 and 6.7 for the case stated in this theorem (that is, considering integrity constraints in target schemas).