XML Schema Mappings

Shun'ichi Amano
University of Edinburgh
s-amano@inf.ed.ac.uk

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

Filip Murlak
University of Edinburgh
fmurlak@inf.ed.ac.uk

ABSTRACT

Relational schema mappings have been extensively studied in connection with data integration and exchange problems, but mappings between XML schemas have not received the same amount of attention. Our goal is to develop a theory of expressive XML schema mappings. Such mappings should be able to use various forms of navigation in a document, and specify conditions on data values. We develop a language for XML schema mappings, and concentrate on three types of problems: static analysis of mappings, their complexity, and their composition. We look at static analysis problems related to various flavors of consistency: for example, whether it is possible to map some document of a source schema into a document of the target schema, or whether all documents of a source schema can be mapped. We classify the complexity of these problems. We then move to the complexity of mappings themselves, i.e., recognizing pairs of documents such that one can be mapped into the other, and provide a classification based on sets of features used in mappings. Finally we look at composition of XML schema mappings. We study its complexity and show that it is harder to achieve closure under composition for XML than for relational mappings. Nevertheless, we find a robust class of XML schema mappings that have good complexity properties and are closed under composition.

Categories and Subject Descriptors. H.2.5 [**Database Management**]: Heterogeneous Databases—*Data translation*; H.2.1 [**Database Management**]: Logical Design; F.1.1 [**Computation by abstract devices**]: Models of Computation—*Automata*

General Terms. Theory, Languages, Algorithms

Keywords. schemas, mappings, XML, consistency, composition, complexity

PODS'09, June 29–July 2, 2009, Providence, Rhode Island, USA.

1. Introduction

The study of mappings between schemas has been an active research subject over the past few years. Understanding such mappings is essential for data integration and data exchange tasks as well as for peer-to-peer data management. All ETL (extract-transform-load) tools come with languages for specifying mappings. We have a very good understanding of mappings between relational schemas (see, e.g., recent SIGMOD and PODS keynotes on the subject [9, 23]); several advanced prototypes for specifying and managing mappings have been developed and incorporated into commercial systems [29, 32]. There are techniques for using such mappings in data integration and exchange, and tools for handling mappings themselves, for example, for defining various operations on them [8, 9, 14, 17, 23, 27, 31].

But much less is known about mappings between XML schemas. While commercial ETL tools often claim to provide support for XML schema mappings, this is typically done either via relational translations, or by means of very simple mappings that establish connections between attributes in two schemas. Transformation languages of such tools tend to concentrate on manipulating values rather than changing structure. In research literature, most XML schema mappings are obtained by various matching tools (see, e.g., [28, 30]) and thus are quite simple from the point of view of their transformational power. More complex mappings were used in the study of information preservation in mappings, either in XML-to-relational translations (e.g., [6]) or in XML-to-XML mappings, where simple navigational queries were used in addition to relationships between attributes [19]. One extra step was made in [4] which studied extensions of relational data exchange techniques to XML, and introduced XML schema mappings that could use not only navigational queries but also simple tree patterns binding several attribute values at once. But even the mappings of [4] cannot reason about the full structure of XML documents: for example, they completely disregard horizontal navigation and do not allow even the simplest joins, something that relational mappings use routinely [16, 23, 29].

Our goal is to develop the theory of XML schema mappings. We would like to introduce a formalism that will be an analog of the commonly accepted formalism of source-to-target dependencies used in relational schema mappings [5, 16, 17, 18, 23]. We would also like to understand the ba-

sic properties of such mappings, including their complexity, static analysis questions related to them, as well as operations on XML schema mappings.

To understand the set of features that need to be modeled in XML schema mappings, consider two schemas, given by the DTDs below. The first DTD D_1 describes a set of professors and their teaching and supervision duties. Teaching is organized by year; each year two courses are taught. For supervision duties, students are listed. The rules of D_1 are:

$$\begin{array}{rcl} r & \rightarrow & prof^* \\ prof & \rightarrow & teach, supervise \\ teach & \rightarrow & year \\ year & \rightarrow & course, course \\ supervise & \rightarrow & student^* \end{array}$$

Element types *course* and *student* have no subelements. We assume that *prof*, *student*, *year*, and *course* have one attribute each (name or id for professors and students, course number for courses).

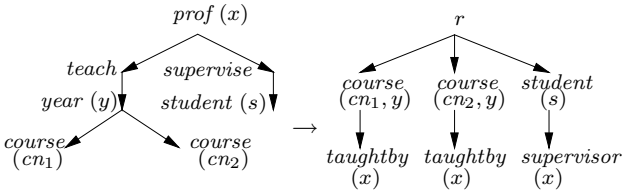
Now suppose that data stored according to D_1 needs to be restructured according to the DTD D_2 that describes courses and students at a university:

$$\begin{array}{rcl} r & \rightarrow & course^*, student^* \\ course & \rightarrow & taughtby \\ student & \rightarrow & supervisor \end{array}$$

Here we assume that *taughtby* and *supervisor* have no subelements but they have one attribute each (for teacher and supervisor names); *student* as before has one attribute, and *course* has two, for course number and year.

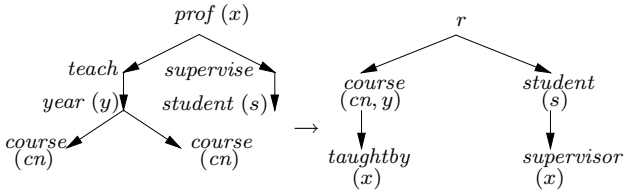
Simple attribute-to-attribute mappings can establish correspondence between professors in D_1 and values of *taughtby* attributes in D_2 for example. But for more complex relationships (e.g., if a D_1-document says that professor x teaches course c, then a D_2-document should say course c has x as the value of *taughtby*), we need to define structural correspondences between DTDs that simple mappings between attributes (or even paths, as in [19]) cannot express.

A proposal made in the study of XML data exchange [4] was to use the standard notion of *tree patterns* in a way that allows us to collect attribute values. For example, we can specify the following mapping between D_1 and D_2:

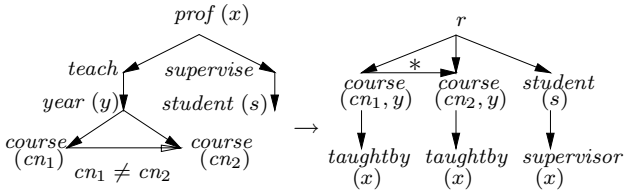


It shows how to restructure information about professors and their teaching and supervision duties under D_2. Note that some nodes of tree patterns carry variables. The semantics of such a mapping is that if we have a document T that conforms to D_1 and a match for the pattern on the left, then we collect values x, y, s, cn_1, cn_2 from that match, and put them in a document that conforms to D_2, structured according to the pattern on the right.

Even though the constraints used in mappings of [4] are more expressive than attribute correspondences and even path-based constraints, they are still quite limited: for example, unlike relational mapping constraints, they cannot take any joins over the source document. In fact, they cannot even test attribute values for equality. For example, the following constraint, that does not replicate a course in the target if the same course was taught by a professor twice, is not allowed in existing XML mapping formalisms:



Another missing feature in existing formalisms is the horizontal navigation. Suppose we know that a professor teaches two different courses. In a source document, they come in a certain order (which may correspond, for example, to the order in which they were taught, or just an order of their numbers). It is reasonable to expect the document obtained as the result of the mapping not to reverse this order. That is, if professor x teaches cn_1 and cn_2 and they come in this order, then in the document that conforms to D_2 the course cn_2 should appear after cn_1 (shown by the $\xrightarrow{*}$ edge below):



Note that this mapping uses two new features: an inequality comparison ($cn_1 \neq cn_2$) and horizontal navigation in both sides of the constraint.

Our goal is to study general and flexible XML schema mappings that have a variety of features shown above. We address the following three problems.

1. Static analysis of schema mappings. We illustrate the problems we consider here by an example. Suppose the DTD D_2 changes to $r \rightarrow courses, students$; $courses \rightarrow course^*$; $students \rightarrow student^*$. Then the mapping given in the example in the first figure is inconsistent: it attempts to make *course* nodes children of the root, while they must be grandchildren. In fact no source tree can be mapped into any target tree. We obviously want to disallow such mappings, so we study the issue of consistency: whether the mapping applies to at least one source tree. This problem was addressed in [4] but only for simple mappings; here we analyze it for expressive mappings that use all forms of navigation and joins.

An even more appealing notion of consistency is that *every* source tree can be mapped into some target tree. This problem has never been addressed previously. In the relational settings, all mappings without target constraints are such, but in the case of XML one can easily generate examples of mappings that apply to some trees but not to all. The

problem – which we call *absolute consistency* – turns out to be much harder than the first version of consistency, and we provide solutions in several common cases.

2. Complexity of schema mappings. We look at the problem of recognizing pairs of trees (T, T') such that T can be mapped into T' by a given mapping. There are two flavors of the problem: for data complexity, the mapping is fixed; for combined complexity, it is a part of the input.

3. Composition of schema mappings. Composition is a crucial operation for modeling schema evolution, and is the most studied operation on relational schema mappings. Key questions addressed in the relational case are the complexity of composition (which is known to be higher than the complexity of the commonly considered mappings) and closure under composition. The latter is normally achieved in the relational case by adding Skolem functions [17]. In addition to studying the complexity of composition, we show that closure is much harder to achieve for XML schema mappings. In fact we identify the set of features that make it impossible to achieve closure under composition without going beyond what is normally considered in relational mappings. We then find a robust class of XML schema mappings that are closed under composition. These are very close to non-relational mappings of the Clio tool [32] extended with Skolem functions.

Organization. Notations are given in Section 2. The schema mapping language is described in Section 3. Basic computational problems related to mappings are addressed in Section 4. Static analysis problems are studied in Sections 5 (consistency) and 6 (absolute consistency). Consistency and complexity of composition are studied in Section 7, and closure under composition in Section 8. Concluding remarks are given in Section 9.

2. Preliminaries

XML documents and DTDs. We view XML documents over a labeling alphabet Γ of *element types* and a set of attributes Att as structures

$$T = \langle U, \downarrow, \rightarrow, lab, (\rho_a)_{a \in Att} \rangle,$$

where

- U is an unranked tree domain (a finite prefix-closed subset of $\mathbb{N}^*$ such that $n \cdot i \in U$ implies $n \cdot j \in U$ for all $j < i$);
- the binary relations $\downarrow$ and $\rightarrow$ are child ($n \downarrow n \cdot i$) and next sibling ($n \cdot i \rightarrow n \cdot (i+1)$);
- $lab : U \rightarrow \Gamma$ is the labeling function; and
- each ρ_a is a partial function from U to V, the domain of attribute values, that gives the values of a for all the nodes in U where it is defined.

A DTD D over Γ with a distinguished symbol r (for the root) and a set of attributes Att consists of a mapping P_D from Γ to regular expressions over $\Gamma - \{r\}$ (one typically writes them as productions $\ell \rightarrow e$ if $P_D(\ell) = e$), and a mapping $A_D : \Gamma \rightarrow 2^{Att}$ that assigns a (possibly empty) set of attributes to each element type. We always assume, for notational convenience, 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 left-to-right, form a string in the language of $P_D(\ell)$.

We shall also refer to a class of *nested-relational DTDs*; as the name suggests, they generalize nested relations. In such DTDs, all productions are of the form $\ell \rightarrow \hat{\ell}_1 \ldots \hat{\ell}_m$, where all ℓ_i's are distinct labels from $\Gamma$and $\hat{\ell}_i$ is either ℓ_i or ℓ_i^* or ℓ_i^+ or $\ell_i? = \ell_i | \varepsilon$. Moreover, such DTDs are not recursive, i.e., the graph in which we put an an edge between ℓ and all the ℓ_i's for each production has no cycles.

Such DTDs are very common in practice (some empirical studies suggest that they cover about 70% of real-world DTDs [10]) and it is known that many computational problems become easier for them [1, 3, 4].

Relational schema mappings. We review the standard definitions of relational schema mappings, see [9, 16, 23]. Given two disjoint relational schemas $\mathbf{S}$ (source) and $\mathbf{T}$ (target), a *source-to-target dependency (std)* is an expression of the form $\varphi_s(\bar{x}, \bar{y}) \rightarrow \psi_t(\bar{x}, \bar{z})$, where φ_s is a conjunction of atoms over $\mathbf{S}$ and ψ_t is a conjunction of atoms over $\mathbf{T}$. If we have a source schema instance S and a target schema instance T, we say that they satisfy the above std if $(S, T) \models \forall \bar{x} \forall \bar{y}\ (\varphi_s(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \psi_t(\bar{x}, \bar{z}))$. That is, we assume that new variables on the right are quantified existentially, and the others are quantified universally. We also omit quantifiers from our shorthand notation. Intuitively, new variables $\bar{z}$ correspond to new values put in the target: every time $\varphi_s(\bar{x}, \bar{y})$ is satisfied, new tuples are put in the target to satisfy $\psi_t(\bar{x}, \bar{z})$ for some $\bar{z}$.

A *schema mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ where $\mathbf{S}$ and $\mathbf{T}$ are source and target relational schemas and Σ is a set of stds. We define $[\![\mathcal{M}]\!]$ as the set of all pairs S, T of source and target instances that satisfy every std from Σ. If $(S, T) \in [\![\mathcal{M}]\!]$, one says that T is a *solution* for S under $\mathcal{M}$.

Sometimes one also adds *target constraints* Σ_t to the mapping; then for $(S, T) \in [\![\mathcal{M}]\!]$ we in addition require that T satisfy Σ_t. In such a case solutions may not exist and it is natural to ask whether solutions exist for some instance, all instances, or a specific instance S. These are essentially various flavors of the consistency problem for schema mappings; in their most general form, they are undecidable, but for some important classes of relational constraints their complexity is well understood [24].

One of the main goals in the study of relational schema mappings is to define various operations on them. Typically these operations correspond to changes that occur in

schemas, i.e., they model schema evolution. The two most important and studied operations are *composition* and *inverse*. While there is still no universally agreed definition of an inverse of a mapping [5, 18], the notion of composition is much better understood [8, 14, 17]. If we have $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and $\mathcal{M}' = (\mathbf{T}, \mathbf{W}, \Sigma')$, the composition is defined as the relational composition $[\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$. A key question then is whether we can have a new mapping, $\mathcal{M} \circ \mathcal{M}'$ between $\mathbf{S}$ and $\mathbf{W}$, so that $[\![\mathcal{M} \circ \mathcal{M}']\!]$ captures exactly the composition $[\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$. A positive answer was provided in [17] for mappings that introduced Skolem functions, i.e., rules like $\varphi_s(\bar{x}) \to \psi_t\big((f_i(\bar{x}_i))_i\big)$ where the f_i's are Skolem functions and $\bar{x}_i$'s are subtuples of $\bar{x}$. For example, $R(x_1, x_2) \to T(x_1, f(x_2))$ says that for each tuple (x_1, x_2) in the source, a tuple containing x_1 and a null needs to be put in the target, but the null value should be the same for all tuples with the same value of x_2.

Child-based schema mappings. The key idea of XML schema mappings defined in [4] was to extend the relational framework by viewing XML trees as databases over two sorts of objects: tree nodes, and data values. Relations in such representations include edges in the tree and relations associating attribute values with nodes. In [4], two restrictions were made. First, only child and descendant edges were considered (essentially it dealt only with unordered trees). A second restriction was that no joins on data values were allowed over the source.

In the case of relational mappings, joins are very common. For example, an std like $S_1(x, y) \wedge S_2(y, z) \to T(x, z)$ computes a join of two source relations by means of reusing the variable y. In the setting of [4] this was disallowed.

To avoid the syntactically unpleasant formalism of two-sorted structures, [4] formalized schema mappings by means of tree patterns with variables for attribute values. Nodes are described by formulae $\ell(\bar{x})$, where ℓ is either a label or the wildcard _, and $\bar{x}$ is a tuple of variables corresponding to the attributes of the node. Patterns are given by:

$$\begin{array}{lcll} \pi & := & \ell(\bar{x})[\lambda] & \text{patterns} \\ \lambda & := & \varepsilon \mid \pi \mid //\pi \mid \lambda, \lambda & \text{lists} \end{array} \qquad (1)$$

That is, a tree pattern is given by its root node and a listing of its subtrees. A subtree can be rooted at a child of the root (corresponding to π in the definition of λ), or its descendant (corresponding to $//\pi$). We shall also abbreviate $\ell(\bar{x})[\varepsilon]$ to just $\ell(\bar{x})$. We write $\pi(\bar{x})$ to indicate that $\bar{x}$ is the list of variables used in π.

For instance, the source pattern in the first example in the Introduction can be expressed as $\pi_1(x, y, cn_1, cn_2, s)$:

$$\begin{array}{l} \mathit{prof}(x)[\mathit{teach}[\mathit{year}(y)[\mathit{course}(cn_1), \mathit{course}(cn_2)]], \\ \qquad \mathit{supervise}[\mathit{student}(s)]] \end{array}$$

Schema mappings were defined in [4] by means of constraints $\pi_1(\bar{x}, \bar{y}) \to \pi_2(\bar{x}, \bar{z})$ so that no variable from $\bar{x}, \bar{y}$ appears in π_1 more than once. For example, the mapping from the Introduction can be expressed in this formalism. The target pattern $\pi_2(x, y, cn_1, cn_2, s)$, which permits the reuse of variables, is:

$$\begin{array}{l} r[\mathit{course}(cn_1, y)[\mathit{taughtby}(x)], \\ \quad \mathit{course}(cn_2, y)[\mathit{taughtby}(x)], \\ \quad \mathit{student}(s)[\mathit{supervisor}(x)]] \end{array}$$

But neither the second nor the third mapping from the Introduction can be expressed in this syntax as they reuse variables and make use of horizontal navigation, and both are prohibited by (1).

3. Schema mapping language

As suggested by our examples (and even translations from relational schema mappings to XML), it is natural to consider both equality/inequality comparisons and additional axes (next- and following-sibling) in schema mappings. We now modify patterns (1) to accommodate these additions. Adding equality is the easiest: we just allow variable reuse in patterns. For next- and following-sibling axes, in the definition of lists of subtrees we replace occurrences of single trees by sequences specifying precise next- and following-sibling relationship. Inequalities will not be added to patterns; rather they will be specified separately in mappings.

Extended patterns will be given by the grammar:

$$\begin{array}{lcll} \pi & := & \ell(\bar{x})[\lambda] & \text{patterns} \\ \lambda & := & \varepsilon \mid \mu \mid //\pi \mid \lambda, \lambda & \text{lists} \\ \mu & := & \pi \mid \pi \to \mu \mid \pi \to^* \mu & \text{sequences} \end{array} \qquad (2)$$

The main difference from (1) is that we replaced π by μ (sequence) in the definition of λ, and μ specifies a sequence of patterns together with their horizontal relationships.

As an example, we consider the last mapping from the Introduction. We now express both left- and right-hand sides in our syntax. The left-hand side $\pi_3(x, y, cn_1, cn_2, s)$ is

$$\begin{array}{r} r[\mathit{prof}(x)\,[\mathit{teach}[\mathit{year}(y)[\mathit{course}(cn_1) \to \\ \mathit{course}(cn_2)]], \\ \mathit{supervise}[\mathit{student}(s)]]] \end{array} \qquad (3)$$

The right-hand side $\pi_4(x, y, cn_1, cn_2, s)$ is

$$\begin{array}{r} r[\mathit{course}(cn_1, y)[\mathit{taughtby}(x)] \to^* \\ \mathit{course}(cn_1, y)[\mathit{taughtby}(x)], \\ \mathit{student}(s)[\mathit{supervisor}(x)]] \end{array} \qquad (4)$$

The formal semantics of patterns is defined by means of the relation $(T, s) \models \pi(\bar{a})$, saying that $\pi(\bar{x})$ is satisfied in a node s of a tree T when its variables $\bar{x}$ are interpreted as $\bar{a}$. It is defined inductively below.

- $(T, s) \models \ell(\bar{a})$ iff s is labeled ℓ and $\bar{a}$ is the tuple of attributes of s.
- $(T, s) \models \ell(\bar{a})[\lambda_1, \lambda_2]$ iff $(T, s) \models \ell(\bar{a})[\lambda_1]$ and $(T, s) \models \ell(\bar{a})[\lambda_2]$.
- $(T, s) \models \ell(\bar{a})[\mu]$ iff $(T, s) \models \ell(\bar{a})$ and there is a child s' of s such that $(T, s') \models \mu$.
- $(T, s) \models \ell(\bar{a})[//\pi]$ iff $(T, s) \models \ell(\bar{a})$ and there is a descendant s' of s such that $(T, s') \models \pi$.

- $(T, s) \models \pi \to \mu$ iff $(T, s) \models \pi$ and $(T, s') \models \mu$ where $s \to s'$.
- $(T, s) \models \pi \to^* \mu$ iff $(T, s) \models \pi$ and $(T, s') \models \mu$ for some following sibling s' of s.

For a tree T and a pattern π, we write $T \models \pi(\bar{a})$ iff $(T, r) \models \pi(\bar{a})$, that is, patterns are witnessed at the root. This is not a restriction since we have descendant // in the language, and can thus express satisfaction of a pattern in an arbitrary node of a tree. We also denote the set $\{\bar{a} \mid T \models \pi(\bar{a})\}$ by $\pi(T)$.

To define source-to-target dependencies for schema mappings, we also add explicit data value comparisons. When we write $\alpha_{=}(\bar{x})$, we mean a formula which is a conjunction of equalities among variables $\bar{x}$. Likewise, $\alpha_{\neq}(\bar{x})$ stands for a conjunction of inequalities and $\alpha_{=,\neq}(\bar{x})$ for a conjunction of equalities and inequalities.

Definition 3.1. Source-to-target dependencies *(abbreviated as* stds*) are expressions of the form*

$$\pi(\bar{x}, \bar{y}), \alpha_{=,\neq}(\bar{x}, \bar{y}) \ \to \ \pi'(\bar{x}, \bar{z}), \alpha'_{=,\neq}(\bar{x}, \bar{z}),$$

where π and π' are patterns such that each variable appears in π exactly once.

Given trees T and T', we say that they satisfy the above stds if for every tuples of values $\bar{a}, \bar{b}$ such that $T \models \pi(\bar{a}, \bar{b})$ and $\alpha_{=,\neq}(\bar{a}, \bar{b})$ is true, it is the case that there exists a tuple of values $\bar{c}$ so that $T' \models \pi'(\bar{a}, \bar{c})$ and $\alpha'_{=,\neq}(\bar{a}, \bar{c})$ is true.

The restriction that each variable appear exactly once in π is only important for our classification, as we would like to look at cases with no equality comparisons between attribute values over the source (such as in [4]). With equality formulae, this is not a restriction at all: for example, an std $r(x, x) \to r'(x, x)$ can be represented as $r(x, x'), x = x' \to r'(x, x')$. For fragments where equality is allowed we shall just reuse variables.

The third example of a mapping from the Introduction can now be expressed as

$$\pi_3(x, y, cn_1, cn_2, s), \ cn_1 \neq cn_2 \ \to \ \pi_4(x, y, cn_1, cn_2, s),$$

where π_3 and π_4 are given by (3) and (4).

Now we can define the notions of schema mappings and their semantics.

Definition 3.2. *An* XML schema mapping *is a triple $\mathcal{M} = (D_s, D_t, \Sigma)$, where D_s is the source DTD, D_t is the target DTD, and Σ is a set of stds.*

Given a tree T that conforms to D_s and a tree T' that conforms to D_t, we say that T' is a solution *for T under $\mathcal{M}$ if (T, T') satisfy all the stds from Σ. We denote the set of all solutions under $\mathcal{M}$ for T by $\text{SOL}_{\mathcal{M}}(T)$.*

The semantics of $\mathcal{M}$ is defined as a binary relation

$$[\![\mathcal{M}]\!] \ = \ \left\{ (T, T') \ \middle| \ \begin{array}{l} T \models D_s, \quad T' \models D_t \\ T' \text{ is a solution for } T \text{ under } \mathcal{M} \end{array} \right\}.$$

These mappings naturally generalize the usual relational mappings. If we have relational schemas $\mathbf{S}$ and $\mathbf{T}$, they can be represented as DTDs $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$: for example, for $\mathbf{S} = \{S_1(A, B), S_2(C, D)\}$, the DTD $D_{\mathbf{S}}$ has rules $r \to s_1, s_2; \ s_1 \to t_1^*; \ s_2 \to t_2^*$, as well as $t_1, t_2 \to \varepsilon$, with t_1 having attributes A, B, and t_2 having attributes C, D. Then each conjunctive query over a schema is easily translated into a pattern over the corresponding DTD together with some equality constraints. For example, $S_1(x, y), S_2(y, z)$ will be translated into

$$r[s_1[t_1(x, y_1)], \ s_2[t_2(y_2, z)]], \ y_1 = y_2.$$

Of course equalities can be incorporated into the pattern (i.e., by $r[s_1[t_1(x, y)], \ s_2[t_2(y, z)]]$) but as we said, we often prefer to list them separately to make classification of different types of schema mappings easier. Note also that these patterns use neither the descendant relation nor the horizontal navigation nor inequalities.

We shall also use abbreviations $\ell(\bar{x})/\ell'(\bar{y})$ for $\ell(\bar{x})[\ell'(\bar{y})]$ and $\ell(\bar{x})//\ell'(\bar{y})$ for $\ell(\bar{x})[//\ell'(\bar{y})]$.

Classification of schema mappings

Source-to-target dependencies used in schema mappings can use four different axes for tree navigation – child, descendant, next and following sibling – as well as equality and inequality.

We denote classes of schema mappings by $\text{SM}(\sigma)$, where σ is a signature indicating which axes and comparisons are present in stds. We refer to the usual navigational axes as $\downarrow$ (child), $\downarrow^*$ (descendant), $\to$ (next-sibling), $\to^*$ (following-sibling). Having $=$ in σ means that we can use formulae $\alpha_{=}, \alpha'_{=}$ in stds (and reuse variables on both sides); having $\neq$ in σ means that we can use formulae $\alpha_{\neq}, \alpha'_{\neq}$, and having both $=$ and $\neq$ means that we can use $\alpha_{=,\neq}$ and $\alpha'_{=,\neq}$ in stds.

To simplify notations, we use abbreviations:

- $\Downarrow$ for $\{\downarrow, \downarrow^*\}$ (vertical navigation);
- $\Rightarrow$ for $\{\to, \to^*\}$ (horizontal navigation);
- $\sim$ for $\{=, \neq\}$ (data value comparisons).

Under these notations, $\text{SM}(\Downarrow)$ is the precisely the class of mappings studied in [4] (as in [4], we do not restrict variable reuse in target patterns). In this paper we shall look at other classes, including $\text{SM}(\Downarrow, \Rightarrow)$, $\text{SM}(\Downarrow, \sim)$, and the largest class $\text{SM}(\Downarrow, \Rightarrow, \sim)$. In a few cases we shall provide more specific information about signatures; then we adopt the convention that the child axis $\downarrow$ is always present.

We also write $\text{SM}^\circ(\sigma)$ for mappings where stds in Σ do not mention attribute values, i.e., where all pattern formulae are of the form $\ell[\lambda]$. These will be useful for establishing hardness results, telling us that structural properties alone make certain problems infeasible.

4. Basic properties of mappings

We first look at some basic properties related to satisfiability of patterns, the complexity of their evaluation, as well as the data and combined complexity of schema mappings.

The first problem is the satisfiability for tree patterns. Its input consists of a DTD D and a pattern $\pi(\bar{x})$; the problem is to check whether there is a tree T that conforms to D and has a match for π (i.e., $\pi(T) \neq \emptyset$). This problem is NP-complete; the result is essentially folklore as it appeared in many incarnations in the literature on tree patterns and XPath satisfiability (see, e.g., [2, 7, 13, 22]). For the sake of completeness, we state the result that applies to patterns in the way they are defined here.

Lemma 4.1. *The satisfiability problem for tree patterns in NP-complete.*

We next look at data and combined complexity of evaluating tree patterns. For data complexity, we fix a pattern π, and we want to check for a given tree T and a tuple $\bar{a}$ whether $T \models \pi(\bar{a})$. For combined complexity, the question is the same, but the input includes $T, \bar{a}$ and π.

Since patterns are essentially conjunctive queries over trees, the data complexity is in DLOGSPACE (and the bound cannot be lowered in general, since transitive closures of $\downarrow$ and $\rightarrow$ may have to be computed). And since they are nicely structured conjunctive queries, the combined complexity is tractable as well. More precisely, we have:

Proposition 4.2. *The data complexity of evaluating tree patterns is DLOGSPACE-complete, and the combined complexity is in PTIME.*

We next move to the complexity of schema mappings and again consider two flavors of it.

- *Data complexity of schema mappings.* For a fixed mapping $\mathcal{M}$, check, for two trees T, T', whether $(T, T') \in [\![\mathcal{M}]\!]$.
- *Combined complexity of schema mappings.* Check, for two trees T, T' and a mapping $\mathcal{M}$, whether $(T, T') \in [\![\mathcal{M}]\!]$.

The data complexity remains low; the combined complexity jumps to the second level of the polynomial hierarchy, but the parameter that makes it jump there is the number of variables in stds. If we fix that number, even the combined complexity is tractable. More precisely, we have:

Theorem 4.3.
- *The data complexity of schema mappings is DLOGSPACE-complete.*
- *The combined complexity of schema mappings is Π_2^p-complete.*
- *The combined complexity of schema mappings is in PTIME if the maximum number of variables per pattern is fixed.*

5. Consistency of schema mappings

As we already mentioned, XML schema mappings may be inconsistent: there are mappings $\mathcal{M}$ so that $[\![\mathcal{M}]\!] = \emptyset$, i.e., no tree has a solution. The problem of recognizing such mappings in $\mathrm{SM}(\Downarrow)$ was shown to be EXPTIME-complete [4]. In addition to consistent mappings, in which *some* trees have solutions, we would like to consider mappings in which *every* tree has a solution. These are very desirable for a variety of reasons: not only are we guaranteed to have possible target documents for every possible source, but the property is also preserved when we compose mappings.

We start by analyzing consistency. We say that a mapping is *consistent* if $[\![\mathcal{M}]\!] \neq \emptyset$; that is, if $\mathrm{SOL}_{\mathcal{M}}(T) \neq \emptyset$ for some $T \models D_s$. The main problem we consider is the following:

PROBEM:	$\mathrm{CONS}(\sigma)$
INPUT:	A mapping $\mathcal{M} = (D_s, D_t, \Sigma) \in \mathrm{SM}(\sigma)$
QUESTION:	Is $\mathcal{M}$ consistent?

If we use $\mathrm{SM}^\circ(\sigma)$ instead of $\mathrm{SM}(\sigma)$ (i.e., if we use mappings in which attribute values are not mentioned at all), we denote the consistency problem by $\mathrm{CONS}^\circ(\sigma)$.

Fact 5.1. (see [4]) *Both* $\mathrm{CONS}(\Downarrow)$ *and* $\mathrm{CONS}^\circ(\Downarrow)$ *are EXPTIME-complete. If we restrict to nested-relational DTDs in schema mappings, then* $\mathrm{CONS}(\Downarrow)$ *is solvable in polynomial (cubic) time.*

Recall that nested-relational DTDs have rules of the form $\ell \to \hat{\ell}_1 \dots \hat{\ell}_m$ for distinct ℓ_i's, where $\hat{\ell}_i$ is ℓ_i or $\ell_i?$ or ℓ_i^* or ℓ_i^+.

Our first result shows that in the absence of data comparisons, the complexity stays the same.

Theorem 5.2. *The problem* $\mathrm{CONS}(\Downarrow, \Rightarrow)$ *is solvable in EXPTIME (and thus it is EXPTIME-complete).*

The key observation is that without data comparisons, $\mathrm{CONS}(\Downarrow, \Rightarrow)$ is no harder than $\mathrm{CONS}^\circ(\Downarrow, \Rightarrow)$, which can be solved by tree automata techniques (more precisely, by non-emptiness of a product of tree automata).

Unlike the case of mappings $\mathrm{SM}(\Downarrow)$ with downward navigation only, once we add even the simplest form of horizontal navigation ($\rightarrow$), we cannot have tractable consistency checking even over nested-relational DTDs:

Proposition 5.3. $\mathrm{CONS}(\Downarrow, \rightarrow)$ *over nested relational DTDs is PSPACE-hard.*

We now move to classes of schema mappings that allow comparisons of attribute values. It is common to lose decidability (or low complexity solutions) of static analysis problems once data values and their comparisons are considered [11, 13, 15, 20, 33]. Here we witness a similar situation. The proofs, however, cannot be simple adaptations of existing proofs which showed undecidability of such formalisms as FO^3 [11] or Boolean combinations of patterns with data value comparisons [15], or implication of conjunctive queries over trees [13]. The reason is the very "positive" and "tree-shaped" nature of stds in schema mappings: the

use of negation is limited to the implication in stds (unlike in [11, 15]) nor can node variables be used in the patterns as in [13].

Nevertheless, we can prove a very strong undecidability result: having either descendant or next-sibling, together with either $=$ or $\neq$, leads to undecidability of consistency.

Theorem 5.4. *The following problems are undecidable:*

- $\textsc{Cons}(\downarrow^*, =)$;
- $\textsc{Cons}(\downarrow^*, \neq)$;
- $\textsc{Cons}(\rightarrow, =)$;
- $\textsc{Cons}(\rightarrow, \neq)$.

In particular, $\textsc{Cons}(\Downarrow, \Rightarrow, \sim)$ *is undecidable.*

This result raises the question whether there is any useful decidable restriction of $\mathrm{SM}(\Downarrow, \Rightarrow, \sim)$. We know from papers such as [11, 20] that getting decidability results for static analysis problems that involve data values is a very nontrivial problem. This time, nested-relational DTDs give us a decidable restriction, if there are no horizontal axes.

Theorem 5.5. *Under the restriction to nested-relational DTDs:*

- *the problem* $\textsc{Cons}(\Downarrow, \sim)$ *is NEXPTIME-complete;*
- *the problem* $\textsc{Cons}(\Downarrow, \Rightarrow, \sim)$ *is undecidable.*

6. Absolute consistency of schema mappings

We now switch to a stronger notion of consistency. Recall that a mapping is consistent if $\textsc{Sol}_{\mathcal{M}}(T) \neq \emptyset$ for some $T \models D_s$. We say that $\mathcal{M}$ is *absolutely consistent* if $\textsc{Sol}_{\mathcal{M}}(T) \neq \emptyset$ for *all* $T \models D_s$. We consider the problem:

PROBEM:	$\textsc{AbsCons}(\sigma)$
INPUT:	Mapping $\mathcal{M} = (D_s, D_t, \Sigma) \in \mathrm{SM}(\sigma)$
QUESTION:	Is $\mathcal{M}$ absolutely consistent?

Reasoning about the complexity of absolute consistency is significantly harder than reasoning about the consistency problem. We know that $\textsc{Cons}(\Downarrow)$ can be easily reduced to $\textsc{Cons}^\circ(\Downarrow)$. However, eliminating data values does not work for absolute consistency. Indeed, consider a mapping with the source DTD $r \to a^*;\ a \to \varepsilon$ and the target DTD $r \to a;\ a \to \varepsilon$, with a having a single attribute. Let the std be $r/a(x) \to r/a(x)$. This mapping $\mathcal{M}$ is not absolutely consistent: take, for example, a source tree with two different values of the attribute. But stripping $\mathcal{M}$ of data values, i.e., replacing the std by $r/a \to r/a$, makes it absolutely consistent.

Thus, we cannot use purely automata-theoretic techniques for reasoning about absolute consistency, even for downward navigation. In fact, the above example indicates that to reason about absolute consistency even in that case, we need to reason about counts of occurrences of different data values.

Here we settle the problem of absolute consistency in the case of downward navigation, i.e., $\textsc{AbsCons}(\Downarrow)$.

We start with a simpler case of $\textsc{AbsCons}^\circ(\Downarrow)$, i.e., checking absolute consistency of mappings $\mathcal{M}^\circ$ in which all references to attribute values have been removed. We show that it has lower complexity than $\textsc{Cons}^\circ(\Downarrow)$. For such mappings, Σ is of the form $\{\pi_i \to \pi'_i\}_{i \in I}$, where patterns have no variables. To check consistency of such a mapping, we need to check whether there exists a set $J \subseteq I$ so that D_t and all the $\pi'_j, j \in J$ are satisfiable, while D_s together with the *negations* of $\pi_k, k \notin J$, are satisfiable. We know that this problem is EXPTIME-complete [4]. On the other hand, for checking absolute consistency, we need to verify that there does not exist $J \subseteq I$ so that D_s and $\pi_j, j \in J$, are satisfiable but D_t and $\pi'_j, j \in J$, are not. Notice that absolute consistency eliminates the need for checking satisfiability of negations of patterns. In fact, since satisfiability of patterns and DTDs is in NP, the above shows that absolute consistency of mappings $\mathcal{M}^\circ$ falls into the 2nd level of the polynomial hierarchy. We can be more precise:

Proposition 6.1. *Checking whether* $\mathcal{M}^\circ$ *is absolutely consistent is* Π^p_2*-complete.*

The main result proves decidability of absolute consistency for schema mappings based on downward navigation:

Theorem 6.2. $\textsc{AbsCons}(\Downarrow)$ *is decidable. In fact the problem is in EXPSPACE and NEXPTIME-hard.*

The proof of the result is quite involved and is based on an analysis of a data structure that counts possible numbers of occurrences of attribute values. Also, it appears to be hard to close the gap between EXPSPACE and NEXPTIME. But the gap is not large: Theorem 6.2 indicates that any algorithm for solving $\textsc{AbsCons}(\Downarrow)$ will run in double-exponential time, and hence will be impractical unless restrictions are imposed. Restrictions to nested-relational DTDs often worked for us, but in this case they alone do not suffice, as we shall see shortly. In addition to nested relational DTDs, we shall need a restriction to *fully-specified stds*, introduced in [4] to obtain tractable algorithms for query answering in data exchange. Patterns for fully-specified stds are given by the grammar:

$$\begin{array}{rcll} \pi & := & \ell(\bar{x})[\lambda], & \text{where } \ell \in \mathcal{L} \\ \lambda & := & \varepsilon \mid \pi \mid \lambda, \lambda & \end{array} \tag{5}$$

In other words, (5) disallows wildcard and descendant compared to (1).

The combination of nested-relational DTDs and fully specified stds gives us tractability, but if we relax these restrictions, the complexity goes back to NEXPTIME-hardness:

Theorem 6.3. *Over nested relational DTDs and fully specified stds, the problem* $\textsc{AbsCons}(\Downarrow)$ *is solvable in PTIME. However, for nested relational DTDs and stds that extend fully specified ones by adding either the wildcard or the descendant, the problem becomes NEXPTIME-hard.*

	CONS(⇓)	CONS(⇓, ⇒)	CONS(⇓, ∼)	CONS(⇓, ⇒, ∼)	ABSCONS(⇓)
arbitrary DTDs	EXPTIME-complete	EXPTIME-complete	undecidable	undecidable	in EXPSPACE; NEXPTIME-hard
nested relational DTDs	PTIME	PSPACE-hard (even for CONS(⇓, →))	NEXPTIME-complete	undecidable	PTIME, with fully specified stds

Figure 1: Summary of consistency results

An abridged summary of the complexity results related to the consistency problem is given in Fig. 1.

7. Composition: consistency and complexity

We now look at the most commonly studied operation on schema mappings: their composition. The definition of the composition is exactly the same as in the relational case [17], since $[\![\mathcal{M}]\!]$ is defined as a binary relation. We define the composition of two mappings $\mathcal{M}$ and $\mathcal{M}'$ simply as $[\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$. That is, for two mappings $\mathcal{M}_{12} = (D_1, D_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (D_2, D_3, \Sigma_{23})$, their composition consists of pairs of trees (T_1, T_3) such that:

1. $T_1 \models D_1$ and $T_3 \models D_3$; and
2. there exists $T_2 \models D_2$ such that (T_1, T_2) satisfy Σ_{12} and (T_2, T_3) satisfy Σ_{23}.

We consider the following problems:

- Consistency of composition: is $[\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$ empty?
- Complexity of composition; and
- Syntactic definability of composition: can we find a mapping $\mathcal{M}_{13} = (D_1, D_3, \Sigma_{13})$ such that $[\![\mathcal{M}_{13}]\!] = [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$?

The last two problems have been studied in the relational case; the first problem is motivated by the consistency problem for XML schema mappings themselves. We study the first two problems in this section, and syntactic definability of composition in Section 8.

7.1 Consistency of composition

We say that the composition of $\mathcal{M}$ and $\mathcal{M}'$ is *consistent* if $[\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!] \neq \emptyset$.

There are two flavors of the consistency of composition problem. One is simply to check whether the composition of two given mappings is consistent. This is not very different from the usual consistency problem: by composing a mapping with a trivial one (e.g., sending the source root to the target root) we can use consistency of composition to test consistency of the mapping itself.

A more interesting version of consistency is when we know that both inputs themselves are consistent:

PROBEM:	CONSCOMP(σ)
INPUT:	Two consistent mappings $\mathcal{M}, \mathcal{M}' \in \text{SM}(\sigma)$
QUESTION:	Is the composition of $\mathcal{M}$ and $\mathcal{M}'$ consistent?

It turns out that the complexity of this problem is the same as the complexity of CONS(σ).

Theorem 7.1. *The complexity of* CONSCOMP(σ) *is:*

- *EXPTIME-complete for* $\sigma = \{\Downarrow\}$ *or* $\{\Downarrow, \Rightarrow\}$.
- *undecidable for* $\sigma = \{\Downarrow, \sim\}$ *or* $\sigma = \{\Rightarrow, \sim\}$.

The decidability result carries over to an arbitrary number of mappings. We can define composition of an arbitrary number of mappings $\mathcal{M}_1, \ldots, \mathcal{M}_n$ simply as the composition of binary relations $[\![\mathcal{M}_i]\!]$'s.

Proposition 7.2. *The problem of checking whether the composition of* n *mappings* $\mathcal{M}_1, \ldots, \mathcal{M}_n$ *from SM*$(\Downarrow, \Rightarrow)$ *is consistent is solvable in EXPTIME.*

7.2 Complexity of composition

By analogy with the complexity of schema mappings, we define data and combined complexity of composition:

- *Data complexity of composition.* For fixed mappings $\mathcal{M}$ and $\mathcal{M}'$, check, for two trees T and T', whether $(T, T') \in [\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$.
- *Combined complexity of composition.* Check, for two mappings $\mathcal{M}$ and $\mathcal{M}'$ and two trees T and T', whether $(T, T') \in [\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$.

Data complexity of relational composition is known to be in NP, and could be NP-complete for some mappings [17]. For XML mappings, the problem becomes undecidable once data value comparisons are allowed. Without such comparisons, it is decidable: the data complexity goes a little bit up compared to the relational case, and we have the usual exponential gap between data and combined complexity.

Theorem 7.3.
- *For mappings from SM*$(\Downarrow, \Rightarrow)$*, the combined complexity of composition is in* 2-*EXPTIME and NEXPTIME-hard, and the data complexity of composition is EXPTIME-complete.*
- *For mappings from both SM*$(\Downarrow, \sim)$ *and SM*$(\Rightarrow, \sim)$*, the combined complexity of composition is undecidable.*

By saying that the data complexity is EXPTIME-complete we mean that it is always in EXPTIME, and there exist mappings $\mathcal{M}, \mathcal{M}'$ such that checking whether the input trees (T, T') belong to $[\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$ is EXPTIME-hard.

Fig. 2 presents a summary of complexity results. By putting "not uniformly decidable" for data complexity of composition over $\mathrm{SM}(\Downarrow, \Rightarrow, \sim)$ we mean that there is no recursive function that maps a pair of mappings $(\mathcal{M}, \mathcal{M}')$ into an algorithm that checks whether $(T, T') \in [\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$.

8. Composition: closure under restrictions

We now address the last issue related to composition of schema mappings: the syntactic representation. The question is whether for two given mappings $\mathcal{M}_{12} = (D_1, D_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (D_2, D_3, \Sigma_{23})$ we can find a mapping $\mathcal{M}_{13} = (D_1, D_3, \Sigma_{13})$ so that $[\![\mathcal{M}_{13}]\!] = [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$. We know that in the relational case getting closure under composition requires adding Skolem functions, i.e., stds of the form $\varphi_s(\bar{x}, \bar{y}) \rightarrow \psi_t(\bar{x}, \bar{z})$ where each element of $\bar{z}$ is either a variable or a term $f(\bar{u})$ for a Skolem function f and some tuple $\bar{u}$ of variables among $\bar{x}, \bar{y}$ [17]. Skolem functions are a natural addition to relational schema mappings. For example, to create a mapping between a source $S(\mathit{empl\_name}, \mathit{project})$ and a target $T(\mathit{empl\_id}, \mathit{empl\_name}, \mathit{office})$, it is natural to use an std $S(x, y) \rightarrow T(f(x), x, z)$ which assigns a unique id to each employee name, rather than $S(x, y) \rightarrow T(z_1, x, z_2)$ which may assign different ids for different projects an employee is involved in.

Formulae in stds can also have equality comparisons among values of Skolem functions, in addition to relational atoms. The semantics of solution is obtained by existentially quantifying such Skolem functions. By Fagin's theorem, this puts data complexity of composed relational mappings in NP, and in fact there are simple examples of NP-completeness of mapping composition [17].

We show here that getting closure for XML schema mappings is harder than for relational mappings, and can only be obtained in limited settings that essentially correspond to nested relations. Such settings constitute an important practical class however; for example, they are used in non-relational extensions of the Clio project [29, 32].

We start with a simple example. Let $D_1 = \{r \rightarrow \varepsilon\}, D_2 = \{r \rightarrow b_1 | b_2;\ b_1, b_2 \rightarrow b_3\}$, and $D_3 = \{r \rightarrow c_1?c_2?c_3?\}$; no attributes are present. Let Σ_{12} contain $r \rightarrow r/\_/b_3$ and Σ_{23} contain $r/b_i \rightarrow r/c_i$ for $i = 1, 2$. Then $[\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$ consists of pairs of trees (r, T), where T matches either r/c_1 or r/c_2. To define such a mapping, we need a disjunction over the target (note that $c_3?$ is necessary in D_3: with the production $r \rightarrow c_1?c_2?$ the composition would be definable by $r \rightarrow r/\_$). Disjunctions in mappings are not well understood even in the relational case, and we certainly do not know how to compose such mappings.

As another illustration of problems with composing XML schema mappings, look at $D_1 = \{r \rightarrow a^*\}, D_2 = \{r \rightarrow b\, b\}$, and $D_3 = \{r \rightarrow \varepsilon\}$, with a and b having an attribute each, and mappings $r/a(x) \rightarrow r/b(x)$ for Σ_{12} and $r \rightarrow r$ for Σ_{23}. In the composition we have pairs (T, r) such that in T at most two different data values are present. This again requires disjunction, e.g., $r[a(x), a(y), a(z)] \rightarrow (x = y \vee y = z \vee x = z)$. In fact a variety of features such as wildcards in patterns and places where attributes appear take us out of our usual classes of schema mappings.

We now summarize what causes problems with composition. We call an element type *starred* if it appears under the scope of a $*$ or a $+$ in a DTD and *unstarred* otherwise.

Proposition 8.1. *Consider DTDs $D = \{r \rightarrow \varepsilon\}$ and $D' = \{r \rightarrow c_1?c_2?c_3?\}$ with no attributes. Then we can find schema mappings so that their composition, as a mapping between D and D', contains exactly pairs (r, T), where T matches r/c_1 or r/c_2, if we allow either*

- *one of the following features in stds: (a) wildcard; (b) descendant; (c) next-sibling; (d) inequality; and only starred element types can have attributes; or*
- *only fully-specified stds but unstarred element types can have attributes.*

In other words, the following features make composition problematic by requiring capabilities (disjunction in mappings) that are not even understood in the relational case:

- the presence of disjunctions and attributes of unstarred element types in DTDs;
- wildcard, descendant, next-sibling, and inequalities in stds.

We now eliminate them all: we look at mappings with fully-specified stds (given by (5); to eliminate wildcard, descendant, and next-sibling), nested relational DTDs (to eliminate disjunctions), no inequalities, and attributes appearing only with starred element types.

For this class, a natural extension of stds with Skolem functions gives us closure under composition. We add Skolem functions to XML schema mappings as it was done for relational mappings in [17], by using terms in place of variables. For a valuation of function symbols $\bar{f}$ and a valuation of variables $\bar{a}$, the meaning of $T \models \varphi(\bar{t})[\bar{f}, \bar{a}]$ is as usual. For a mapping $\mathcal{M} = (D_s, D_t, \Sigma)$ with Skolem functions, $(T, T') \in [\![\mathcal{M}]\!]$ iff there exist functions $\bar{f}$ such that for each $(\varphi, \alpha_= \rightarrow \psi, \alpha'_=) \in \Sigma$ and each $\bar{a}$, if $T \models \varphi, \alpha_=[\bar{f}, \bar{a}]$ then $T' \models \psi, \alpha'_=[\bar{f}, \bar{a}]$. Note that we can use the same function symbol in more than one constraint.

We say that a DTD is *strictly nested-relational* if it is nested-relational and only starred element types can have attributes. Now we can state the closure result.

Theorem 8.2. *The class of mappings with Skolem functions and equality, restricted to strictly nested-relational DTDs and fully specified stds, is closed under composition.*

	tree pattern evaluation	mappings $\mathcal{M}$	composition $[\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$ over SM($\Downarrow, \Rightarrow$)	composition $[\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$ over SM($\Downarrow, \Rightarrow, \sim$)
data complexity	DLOGSPACE-complete	DLOGSPACE-complete	EXPTIME-complete	not uniformly decidable
combined complexity	PTIME	Π_2^p-complete PTIME for fixed-arity stds	in 2-EXPTIME NEXPTIME-hard	undecidable

Figure 2: Summary of complexity results

9. Conclusions

This paper has made several initial steps in the investigation of XML schema mappings, but many questions remain. Of course there are technical questions left open here, such as closing the complexity gaps (although all the gaps we have here are rather small). Most of them are related to the following technical problem: given a DTD D and two sets of patterns P_+ and P_-, can we find a tree $T \models D$ that matches all the patterns in P_+ and none in P_-? We know that the problem is in EXPTIME and NP-hard; knowing its exact complexity will help us close the complexity gaps.

More importantly, we would like to extend this work in several directions. One of them is constructing target instances. This is important in data integration and exchange tasks, but so far good algorithms are lacking, even for very simple mappings already introduced in [4]. We also would like to work further on operations on schema mappings. We have identified a natural class that is closed under composition, but we do not know anything about its maximality, nor do we know anything about other operations such as inverse [5, 18] or merge [9]. We would like to see how the complexity of schema mappings affects the complexity of query answering in integration and exchange scenarios with XML data.

Acknowledgment. The authors were supported by EPSRC grants E005039 and F028288; the second author also by EU grant MEXC-CT-2005-024502.

10. References

[1] S. Abiteboul, L. Segoufin, V. Vianu. Representing and querying XML with incomplete information. In *PODS'01*, pages 150–161.

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

[3] M. Arenas, L. Libkin. A normal form for XML documents. *ACM TODS* 29 (2004), 195–232.

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

[5] M. Arenas, J. Pérez, C. Riveros. The recovery of a schema mapping: bringing exchanged data back. *PODS'08*, pages 13–22.

[6] D. Barbosa, J. Freire, A. Mendelzon. Designing information-preserving mapping schemes for XML. In *VLDB'05*, pages 109–120.

[7] M. Benedikt, W. Fan, F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM* 55(2): (2008).

[8] P. Bernstein, T.Green, S. Melnik, A. Nash. Implementing mapping composition. *VLDB'06*, pages 55–66.

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

[10] G. J. Bex, F. Neven, J. Van den Bussche. DTDs versus XML Schema: a practical study. *WebDB'04*, pages 79–84.

[11] M. Bojanczyk, C. David, A. Muscholl, Th. Schwentick, L. Segoufin. Two-variable logic on data trees and XML reasoning. In *PODS'06*, pages 10–19.

[12] H. Björklund, W. Martens, T. Schwentick. Conjunctive query containment over trees. *DBPL'07*, pages 66–80.

[13] H. Björklund, W. Martens, T. Schwentick. Optimizing conjunctive queries over trees using schema information. *MFCS'08*, pages 132–143.

[14] L. Chiticariu, W.-C. Tan. Debugging schema mappings with routes. In *VLDB'06*, pages 79–90.

[15] C. David. Complexity of data tree patterns over XML documents. In *MFCS'08*, pages 278–289.

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

[17] R. Fagin, Ph. Kolaitis, L. Popa, W.C. Tan. Composing schema mappings: second-order dependencies to the rescue. *ACM TODS* 30(4) 994–1055 (2005).

[18] R. Fagin, Ph. Kolaitis, L. Popa, W.C. Tan. Quasi-inverses of schema mappings. *ACM TODS* 33(2): (2008).

[19] W. Fan, P. Bohannon. Information preserving XML schema embedding. *ACM TODS* 33(1) (2008).

[20] W. Fan, L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM* 49 (2002), 368–406.

[21] G. Gottlob, C. Koch, K. Schulz. Conjunctive queries over trees. *J. ACM* 53 (2006), 238–272.

[22] J. Hidders. Satisfiability of XPath expressions. In *DBPL'03*, pages 21–36.

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

[24] Ph. Kolaitis, J. Panttaja, W.C. Tan. The complexity of data exchange. In *PODS 2006*, pages 30–39.

[25] M. Lenzerini. Data integration: a theoretical perspective. In *PODS'02*, pages 233–246.

[26] H. Lewis. Complexity results for classes of quantificational formulas. *JCSS* 21 (1980), 317–353.

[27] J. Madhavan, A. Halevy. Composing mappings among data sources. In *VLDB'03*, pages 572–583.

[28] S. Melnik, H. Garcia-Molina, E. Rahm. Similarity flooding: a versatile graph matching algorithm. In *ICDE'02*, pages 117–128.

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

[30] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *VLDB'98*, pages 122–133.

[31] A. Nash, P. Bernstein, S. Melnik. Composition of mappings given by embedded dependencies. *ACM TODS* 32(1): 4 (2007).

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

[33] L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL'06*, pages 41–57.