XML Schema Mappings: Data Exchange and Metadata Management

XML Schema Mappings: Data Exchange and Metadata Management

Shun'ichi Amano
and
Claire David
and
Leonid Libkin
and
Filip Murlak

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 study both data exchange with such mappings and metadata management problems. Specifically, we concentrate on four types of problems: complexity of mappings, query answering, consistency issues, and composition.

We first analyze the complexity of mappings, 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. Next, we chart the tractability frontier for the query answering problem. We show that the problem is tractable for expressive schema mappings and simple queries, but not vice versa. Then we move to static analysis. We study the complexity of the consistency problem, i.e., deciding whether it is possible to map some document of a source schema into a document of the target schema. Finally, we look at composition of XML schema mappings. We analyze 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, in addition to being closed under composition, have good complexity properties with respect to the main data management tasks. Due to its good properties, we suggest this class as the class to use in applications of XML schema mappings.

Categories and Subject Descriptors: H.2.5 [**Database Management**]: Heterogeneous Databases—*data translation*; H.2.8 [**Database Management**]: Database Applications

General Terms: Theory, Languages, Algorithms

Additional Key Words and Phrases: XML, incomplete information, query answering, certain answers, consistency, membership

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 recent surveys [Arenas et al. 2010; Barceló 2009; Bernstein and Melnik 2007; Kolaitis 2005]). Several advanced prototypes for specifying and managing mappings have been developed [Fagin et al. 2009; Popa et al. 2002; Marnette et al. 2011], and some have been incorporated into commercial systems. There are techniques for using such mappings in data integration and exchange, and metadata management tools (i.e., tools for handling mappings themselves), see, e.g., [Nash et al. 2007; Bernstein and Melnik 2007; Chiticariu and Tan 2006; Fagin et al. 2004; Kolaitis 2005; Madhavan and Halevy 2003] as well as many papers referenced in the surveys mentioned above.

However, 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 the research literature, most XML schema mappings are obtained by various matching tools (see, e.g., [Melnik et al. 2002; Milo and Zohar 1998]) 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., [Barbosa et al. 2005]) or in XML-to-XML mappings, where simple navigational queries were used in addition to relationships between attributes [Fan and Bohannon 2008]. One extra step was made in [Arenas and Libkin 2008] 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 [Arenas and Libkin 2008] 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 [Arenas et al. 2010; Fagin et al. 2003; Fagin et al. 2009].

Our main goal is to develop a theory of XML schema mappings. We would like to introduce a formalism that will be a proper analog of the commonly accepted formalism of source-to-target dependencies used in relational schema mappings [Arenas et al. 2010; Barceló 2009; Fagin et al. 2003; Fagin et al. 2004]. We would like to understand the basic properties of such mappings, such as their complexity, operations on them, and their static analysis. We would like to explore the complexity of query answering in data exchange settings given by XML schema mappings.

At the end of the study, we would like to understand which features of XML schema mappings make handling them hard, and which lend themselves to efficient algorithms. Based on the results of the study, we would like to propose a class of mappings that can be used in practice due to its good properties.

Examples of mappings. To understand features needed in XML schema mappings, we now present some examples of transformations that need to be modeled. Consider two schemas, given, in this example, by the DTDs below. The first DTD

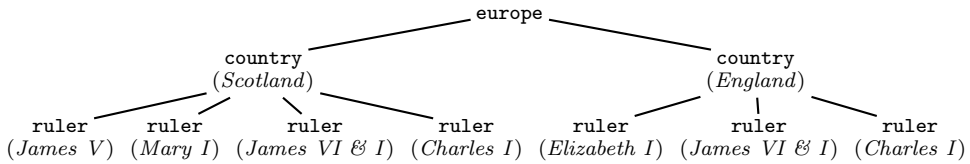


Fig. 1. Tree T_1 conforming to DTD D_1

D_1 describes a set of countries and their rulers. The rules of D_1 are

$$\texttt{europe} \to \texttt{country}^*$$
$$\texttt{country} \to \texttt{ruler}^*$$

Element type `ruler` has no sub-elements. We assume that `ruler` and `country` have one attribute each (name), and `ruler` nodes are ordered chronologically, i.e., in the order of succession. A document could look like the tree in Fig. 1.

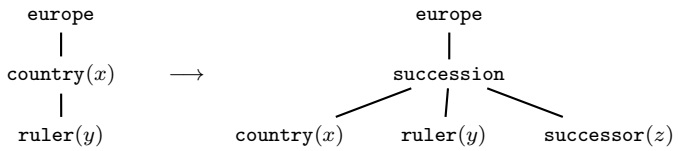
Now suppose that data stored according to D_1 needs to be restructured according to the DTD D_2 that stores successions that took place in various countries.

$$\texttt{europe} \to \texttt{succession}^*$$
$$\texttt{succession} \to \texttt{country ruler successor}$$

Here we assume that `country`, `ruler`, and `successor` have no sub-elements but they have one attribute (name).

Simple attribute-to-attribute mappings can establish correspondence between rulers in D_1 and in D_2 for example. But for more complex relationships (e.g., if a D_1-document says that country x was ruled by y, then a D_2-document should contain a node `succession` with country x and ruler y), we need to define structural correspondences between schemas that simple mappings between attributes (or even paths, as in [Fan and Bohannon 2008]) cannot express.

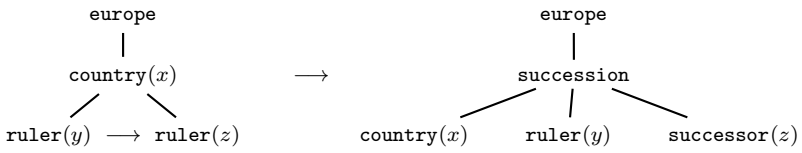
Standard source-to-target dependencies in relational databases specify how source patterns are translated into target patterns. It is thus natural to extend this idea to XML, as indeed was done even for limited mappings we mentioned earlier. Specifically, we shall use *tree patterns* that collect attribute values. For example, we can specify the following mapping between D_1 and D_2:



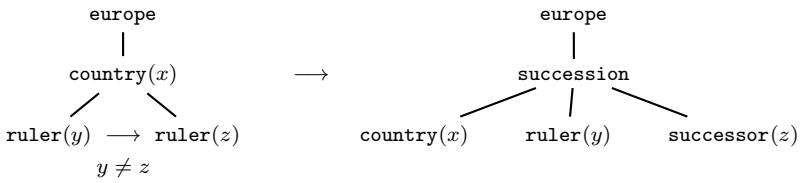
It shows how to restructure information about countries and rulers under D_2. Note that some nodes of tree patterns carry variables. The semantics of such a mapping is that if we have a document T that conforms to D_1 and a match for the pattern on the left, then we collect values x, y that match, and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

Such pattern-based mappings are more expressive than attribute correspondences and even path-based constraints. They have been explored in [Arenas and Libkin 2008], but still in a quite limited way. To see one set of features that the mappings

of [Arenas and Libkin 2008] lacked, note that in the mapping above we should specify the successor's name z. To do so we need to express that two `ruler` nodes are subsequent siblings, which requires horizontal navigation. If we use the $\longrightarrow$ edge to express that the two `ruler` nodes are subsequent siblings, the intended mapping can be formulated as follows:



Another limitation of the formalism introduced in [Arenas and Libkin 2008] is that, unlike relational mapping constraints, it cannot take any joins over the source document. In fact, it cannot even test attribute values for equality or inequality. For example, the following constraint, that takes into account that a ruler can be in power for several subsequent periods (e.g., some presidents), is not allowed in existing XML mapping formalisms:

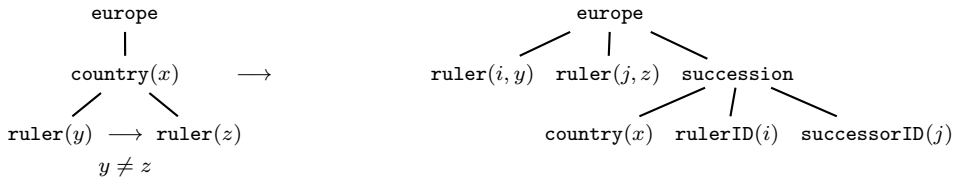


Note that this mapping uses two new features: an inequality comparison ($y \neq z$) and horizontal navigation.

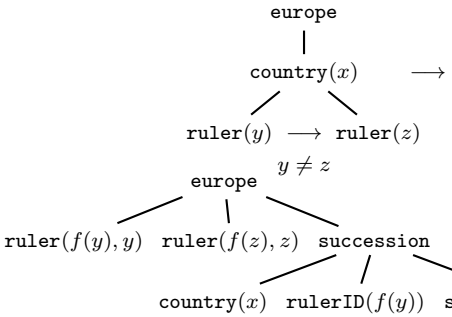
Finally, a useful feature is the ability to express the fact that some new value introduced on the target side does *not* depend on some of the values collected on the source side. Suppose that we change the DTD D_2 to introduce IDs for rulers:

$$\texttt{europe} \to \texttt{succession}^* \ \texttt{ruler}^*$$
$$\texttt{succession} \to \texttt{country rulerID successorID}$$

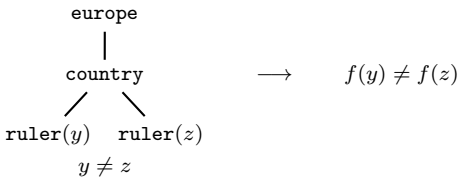
`ruler` has two attributes: ID and name, `country`, `rulerID`, and `successorID` have one attribute. If we try to express the mapping as



we do not reflect the fact, that there is a single ID for each ruler. This problem shows up in relational mappings too and is usually solved by using Skolem functions [Fagin et al. 2004]. The constraint shown below expresses functional dependency between the ruler name and the id:



To make sure that the IDs are unique, we would need another constraint:



Key problems and outline of the paper. As mentioned earlier, our goal is to study general and flexible XML schema mappings that have a variety of features shown above. We would like to understand which features can easily be used in mappings without incurring high computational costs, and which features need to be omitted due to their inherent complexity. Towards that goal, we concentrate on the following standard data exchange problems (well explored in the relational context).

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. We refer to this problem as the *membership problem*. There are two flavors of the membership problem: for data complexity, the mapping is fixed; for combined complexity, it is a part of the input.

Query answering. In the problem of data exchange one needs to materialize a target instance, that is a solution for a given source instance with respect to a given schema mapping, and use it to answer queries over the target. As there could be many possible solutions, the standard semantics of query answering is that of *certain answers* which are independent of a chosen solution. So for good mappings one should be able to construct a single solution that permits finding certain answers within reasonable complexity, at least for simple query languages.

Static analysis of schema mappings. Consider the mapping in the first example we had and change the DTD D_2 to `europe` $\rightarrow$ `successions`; `successions` $\rightarrow$ `succession`*; `succession` $\rightarrow$ `country ruler successor`. Then the mapping becomes inconsistent: it attempts to make `succession` nodes children of the root, while they must be grandchildren. We obviously want to disallow such mappings, so we study the issue of consistency: whether the mappings make sense.

Schema evolution. Evolution of schemas is usually described by means of the *composition* operator: a mapping $\mathcal{M} \circ \mathcal{M}'$ has the effect of first applying $\mathcal{M}$ followed by an application of $\mathcal{M}'$. 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 [Fagin et al. 2004]. These issues will arise in the XML context too.

We now outline the main contributions of the paper. We define a very general class of XML schema mappings that, in addition to the only feature considered previously (vertical navigation in documents), has other features presented in our examples:

—horizontal navigation;

—data-value comparisons (e.g., joins); and

—Skolem functions.

We also look at a restriction of schemas, in which DTDs are non-recursive and regular expressions are of rather simple form, generalizing nested relations. Specifically, they are expressions of the form $\hat{a}_1 \dots \hat{a}_n$, where all the a_i's are distinct, and $\hat{a}$ can stand for a itself, or a^*, or a^+, or $a|\epsilon$. All the schemas we saw in the introduction are such. We call such schemas *nested-relational*.

We then provide a detailed study of the effect of these features on the main computational tasks we consider. Below we give a quick summary.

Complexity of schema mappings. We do a full classification of the complexity of schema mappings. In particular, we show that the membership problem is NP-complete with respect to data complexity, and NEXPTIME-complete with respect to combined complexity in the presence of Skolem functions; without them, the complexity drops to LOGSPACE-complete and Π_2^p-complete. Thus, as far as the complexity of mappings is concerned, the XML case matches the relational one [Fagin et al. 2004; Gottlob and Senellart 2010; Pichler and Skritek 2011].

Query answering. While in the relational case conjunctive queries behave particularly well in data exchange [Fagin et al. 2003], it was already shown in [Arenas and Libkin 2008] that even for mappings using just vertical navigation their analogs could be coNP-hard. Thus, [Arenas and Libkin 2008] isolated a subclass of child-based schema mappings admitting a polynomial algorithm for query answering. We start with that class (which includes mappings with nested-relational schemas) and see how far we can extend it by adding features to mappings and to queries (to match the expressiveness of the mappings).

Again we do a systematic investigation of the complexity of query answering. The main lesson of our study is that we cannot extend both mappings and queries simultaneously: the basic class of queries remains tractable under relatively expressive mappings; but adding new features to the query languages quickly leads to intractability, even for very simple mappings that behave well with basic queries.

Static analysis of schema mappings. The consistency problem was looked at in the case of the simplest mappings based on vertical navigation [Arenas and Libkin 2008] and shown to be EXPTIME-complete. Here we fully analyze it for expressive mappings that use all forms of navigation and joins.

We show that it is the combination of horizontal navigation and joins (equality tests) that determines the complexity. Having arbitrary joins very quickly leads to

undecidability, while without them consistency stays in EXPTIME in the presence of horizontal navigation and Skolem functions. When schemas are nested-relational, the bounds come down: without joins consistency is polynomial without horizontal navigation, and PSPACE-complete with it. With joins and Skolem functions it is NEXPTIME-complete, but adding next-sibling makes it undecidable even for the simplest schemas.

Composition of schema mappings. 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 [Popa et al. 2002] extended with Skolem functions.

At the end, we come up with a class of mappings having many desirable properties: they admit efficient algorithms for both static analysis and query answering, and are closed under composition.

Note. This paper combines and expands three conference papers: Sections 3, 4, 6, 7 come essentially from [Amano et al. 2009], except Prop. 6.9 and Thm. 7.6, originally published in [David et al. 2010], Section 5 comes from [Amano et al. 2010]. New results include: Thm. 4.5, Thm. 4.6, Prop. 6.5 (upper bound), Thm. 6.7 (lower bound without inequality, undecidability without equality, and without inequality), Prop. 6.8, Prop. 6.9 (inequality), Thm. 7.2 (2-EXPTIME lower bound), Thm. 7.6 (inequality, descendant, and horizontal order).

Organization. Notations are given in Section 2. The schema mapping language is described in Section 3. In Section 4 we study the membership problem. Complexity of query answering is studied in Section 5. Static analysis problems are studied in Section 6. Composition related problems are studied in Section 7. Concluding remarks are given in Section 8.

2. PRELIMINARIES

2.1 XML documents and DTDs

We view XML documents over a labeling alphabet Γ of *element types* and a set of attributes Att as structures $T = (\mathrm{dom}(T), \downarrow, \rightarrow, lab, (\rho_a)_{a \in Att})$ where

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

By $|T|$ we shall denote $|\mathrm{dom}(T)|$, i.e., the number of elements (nodes) of the underlying tree domain $\mathrm{dom}(T)$.

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

$A_D : \Gamma \to 2^{Att}$ 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 $\ell(a_1, \ldots, a_n)$. Note that arbitrary number of attributes can be modeled by trees with one attribute per node, by using multiple children. Thus, the number of attributes will not play any special role in our complexity results.

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 write $\|D\|$ for the total size of D, or, in other words, the memory needed to store a natural representation of D. We shall extend this notation to other complex objects (sets of patterns, schema mappings, automata, etc.) as needed.

2.2 Relational schema mappings

We review the standard definitions of relational schema mappings, see [Bernstein and Melnik 2007; Fagin et al. 2003; Kolaitis 2005]. Given two disjoint relational schemas $\mathbf{S}$ (source) and $\mathbf{T}$ (target), a *source-to-target dependency* is an expression of the form $\varphi_s(\bar{x}, \bar{y}) \longrightarrow \psi_t(\bar{x}, \bar{z})$, 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 dependency if $(S, T) \models \forall\bar{x}\forall\bar{y}\ (\varphi_s(\bar{x}, \bar{y}) \longrightarrow \exists\bar{z}\ \psi_t(\bar{x}, \bar{z}))$. 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 dependencies. We define $[\![\mathcal{M}]\!]$ as the set of all pairs S, T of source and target instances that satisfy every dependency 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 [Kolaitis et al. 2006].

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 [Arenas et al. 2009; Fagin et al. 2007], the notion of composition is much better understood [Nash et al. 2007; Chiticariu and Tan 2006; Fagin et al. 2004]. 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 $\mathbf{W}$ such that $[\![\mathcal{M} \circ \mathcal{M}']\!] = [\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$. A positive answer was provided in [Fagin

et al. 2004] for mappings that introduced Skolem functions, i.e., used rules like $\varphi_s(\bar{x}) \longrightarrow \psi_t\big(f_1(\bar{x}_1), \ldots, f_k(\bar{x}_k)\big)$ where the f_i's are Skolem functions and $\bar{x}_i$'s are subtuples of $\bar{x}$. For example, $R(x_1, x_2) \longrightarrow T(x_1, f(x_2))$ 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.

2.3 Child-based schema mappings

The key idea of XML schema mappings defined in [Arenas and Libkin 2008] 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 [Arenas and Libkin 2008], 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 relational mappings joins are very common. For example, in $S_1(x, y) \wedge S_2(y, z) \longrightarrow T(x, z)$ we compute a join of two source relations by means of reusing the variable y. In the setting of [Arenas and Libkin 2008] this was disallowed.

To avoid the syntactically unpleasant formalism of two-sorted structures, [Arenas and Libkin 2008] 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:

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

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 use abbreviations

$$\ell(\bar{x}) = \ell(\bar{x})[\varepsilon]\,, \qquad \ell(\bar{x})/\ell'(\bar{y}) = \ell(\bar{x})[\ell'(\bar{y})]\,, \qquad \ell(\bar{x})//\ell'(\bar{y}) = \ell(\bar{x})[//\ell'(\bar{y})]$$

and 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

$$\pi_1(x, y) = \texttt{europe}/\texttt{country}(x)/\texttt{ruler}(\texttt{y})\,.$$

Schema mappings were defined in [Arenas and Libkin 2008] by means of constraints $\pi_1(\bar{x}, \bar{y}) \longrightarrow \pi_2(\bar{x}, \bar{z})$ 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 is

$$\pi_2(x, y, z) = \texttt{europe}/\texttt{succession}[\texttt{country}(x), \texttt{ruler}(y), \texttt{successor}(z)]\,.$$

But no other mapping from the Introduction can be expressed in this syntax as they use data comparison or 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 Skolem functions, equality/inequality comparisons, and additional axes (next- and following-sibling) in schema mappings.

We now extend patterns (1) to accommodate these additions. The first addition is that we allow arbitrary terms constructed of variables function symbols. Terms are defined inductively: each variable is a term, and if f is a function symbol of arity k and $t_1, t_2, \ldots, t_k$ are terms, then $f(t_1, t_2, \ldots, t_k)$ is also a term. The second addition is that we allow explicit equalities and inequalities between terms in the patterns. The third addition is that we allow next- and following-sibling axes. To do so, in the definition of lists of subtrees we replace occurrences of single trees by sequences specifying precise next- and following-sibling relationship.

Extended patterns are given by the grammar

$$\begin{array}{lll} \varphi & := & \pi, \alpha & \text{patterns} \\ \pi & := & \ell(\bar{t})[\lambda] & \text{pure patterns} \\ \lambda & := & \varepsilon \mid \mu \mid //\pi \mid \lambda, \lambda & \text{sets} \\ \mu & := & \pi \mid \pi \to \mu \mid \pi \to^+ \mu & \text{sequences} \end{array} \tag{2}$$

where α is a conjunction of equalities and inequalities on terms over the set of variables Var and a set of function symbols Fun, and $\bar{t}$ is a tuple of terms. We denote the set of variables used in φ by $\operatorname{Var}\varphi$, and write $\varphi(\bar{x})$ to indicate that $\bar{x}$ is the list of all variables in $\operatorname{Var}\varphi$. Terms set aside, the main difference from (1) is that we replaced π by μ (sequence) in the definition of λ, and μ specifies a sequence of pure 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 is

$$\texttt{europe}/\texttt{country}(x)\,[\texttt{ruler}(y) \to \texttt{ruler}(z)]\,,\ y \neq z$$

and the right-hand side is

$$\begin{array}{l} \texttt{europe}[\,\texttt{ruler}(f(y), y),\ \texttt{ruler}(f(z), z), \\ \qquad \texttt{succession}[\texttt{country}(x),\ \texttt{rulerID}(f(y)),\ \texttt{successorID}(f(z))\,]\,]\,. \end{array}$$

The formal semantics of patterns is defined by means of the relation $(T, s, F) \models \varphi(\bar{a})$, saying that $\varphi(\bar{x})$ is satisfied in a node s of a tree T when its variables $\bar{x}$ are interpreted as $\bar{a}$ and the function symbols are interpreted according to the valuation F, assigning to each function symbol f of arity k a function $F(f)\colon V^k \to V$ (i.e., the value of $f(t_1, t_2, \ldots, t_k)$ is $F(f)(b_1, b_2, \ldots, b_k)$, where b_i is the value of t_i). The relation is defined inductively as follows:

$(T, s, F) \models \ell(\bar{t})$	if $lab(s) = \ell$ or $\ell = \_$, and $\bar{t}$ interpreted under F is the tuple of attributes of s;
$(T, s, F) \models \ell(\bar{t})[\lambda_1, \lambda_2]$	if $(T, s, F) \models \ell(\bar{t})[\lambda_1]$ and $(T, s, F) \models \ell(\bar{t})[\lambda_2]$;
$(T, s, F) \models \ell(\bar{t})[\mu]$	if $(T, s, F) \models \ell(\bar{t})$ and $(T, s', F) \models \mu$ for some s' with $s \downarrow s'$;
$(T, s, F) \models \ell(\bar{t})[//\pi]$	if $(T, s, F) \models \ell(\bar{t})$ and $(T, s', F) \models \pi$ for some s' with $s \downarrow^+ s'$;
$(T, s, F) \models \pi \to \mu$	if $(T, s, F) \models \pi$ and $(T, s', F) \models \mu$ for some s' with $s \to s'$;
$(T, s, F) \models \pi \to^+ \mu$	if $(T, s, F) \models \pi$ and $(T, s', F) \models \mu$ for some s' with $s \to^+ s'$;
$(T, s, F) \models \pi, \alpha$	if $(T, s, F) \models \pi$ and α holds under the interpretation F;

where $\downarrow^+$ and $\to^+$ are transitive closures of $\downarrow$ and $\to$.

Observe that semantically "sets" in tree patterns are literally sets: for a node satisfying $\ell(\bar{t})[\lambda_1, \lambda_2]$, the nodes witnessing λ_1 and λ_2 are not necessarily distinct.

For a tree T, a valuation F, and a pattern φ, we write $(T, F) \models \varphi(\bar{a})$ to denote $(T, \varepsilon, F) \models \varphi(\bar{a})$, 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 write $(T, F) \models \varphi(\bar{x})$ if $(T, F) \models \varphi(\bar{a})$ for some $\bar{a}$. If there are no function symbols in φ, or the valuation is clear from the context, we write $T \models \varphi(\bar{a})$ and $T \models \varphi(\bar{x})$.

Note that patterns are closed under conjunction: $(\ell_1(\bar{s})[\lambda_1], \alpha_1) \wedge (\ell_2(\bar{t})[\lambda_1], \alpha_2)$ can be expressed as $\ell_1(\bar{t})[\lambda_1, \lambda_2], \alpha_1 \wedge \alpha_2 \wedge s_1 = t_1 \wedge s_2 = t_2 \wedge \cdots \wedge s_n = t_n$, if only $\ell_1(\bar{s})$ and $\ell_2(\bar{t})$ are *compatible*, i.e., $\ell_1 = \ell_2$ or $\ell_1 = \_$ or $\ell_2 = \_$, and $\bar{s} = s_1, s_2, \ldots, s_n$, $\bar{t} = t_1, t_2, \ldots, t_n$. If $\ell_1(\bar{s})$ and $\ell_2(\bar{t})$ are not compatible, the conjunction is always false. To express this we allow a special pattern $\bot$, which can be seen as a new element type, never allowed by the schema.

DEFINITION 3.1. Source-to-target dependencies *are expressions of the form*

$$\varphi(\bar{x}, \bar{y}) \;\longrightarrow\; \psi(\bar{x}, \bar{z})\,,$$

where φ and ψ are patterns, the pure pattern π underlying φ does not contain function symbols nor repetitions of variables, and each variable ξ_0 in $\bar{x}, \bar{y}$ satisfies the safety condition: either ξ_0 is used in π or φ contains a sequence of equalities $\xi_0 = \xi_1$, $\xi_1 = \xi_2$, $\ldots$, $\xi_{k-1} = \xi_k$ where $\xi_1, \xi_2, \ldots, \xi_{k-1}$ are terms and ξ_k is a variable used in π.

Given trees T and T' we say that they satisfy the above dependency with respect to a valuation F if for all tuples of values $\bar{a}, \bar{b}$ such that $(T, F) \models \varphi(\bar{a}, \bar{b})$, there exists a tuple of values $\bar{c}$ so that $(T', F) \models \psi(\bar{a}, \bar{c})$.

The restriction introduced in Definition 3.1 that π does not contain function symbols nor repetitions of variables 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 [Arenas and Libkin 2008]). With equality formulae, this is not a restriction at all: for example, a dependency $r(x, x) \longrightarrow r'(x, x)$ can be represented as

$$r(x, x'), x = x' \;\longrightarrow\; r'(x, x')\,.$$

For fragments where equality is allowed we shall just reuse variables.

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 dependencies.*

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 there exists a valuation F of function symbols such that (T, T') satisfy all the dependencies from Σ with respect to F. We denote the set of all solutions under $\mathcal{M}$ for T by $\textsc{Sol}_{\mathcal{M}}(T)$.*

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

$$[\![\mathcal{M}]\!] \;=\; \big\{(T, T') \;\big|\; T \models D_s,\; T' \models D_t,\; T' \in \textsc{Sol}_{\mathcal{M}}(T)\big\}.$$

In the presence of function symbols, there is no need to introduce new variables on the target sides of dependencies. They can all be removed by means of a procedure

called *skolemization*: for each variable z_i occurring only on the target side, a fresh function symbol f_{z_i} is introduced, and each occurrence of z_i is replaced with $f_{z_i}(\bar{x})$.

After this has been done, it is also possible to eliminate inequalities from the target side, using equality on the source side and an incompatible pattern $\bot$, e.g.,

$$\varphi(\bar{x},\bar{y}) \longrightarrow \psi(\bar{x}), t_1 \neq s_1, t_2 \neq s_2$$

is equivalent to the following three dependencies

$$\varphi(\bar{x},\bar{y}) \longrightarrow \psi(\bar{x})\,; \qquad \varphi(\bar{x},\bar{y}), t_1 = s_1 \longrightarrow \bot\,; \qquad \varphi(\bar{x},\bar{y}), t_2 = s_2 \longrightarrow \bot\,.$$

The mappings introduced above 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 (by taking $r[s_1[t_1(x,y)],\ s_2[t_2(y,z)]]$) 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.

Classification of schema mappings. Dependencies used in schema mappings can use four different axes for tree navigation – child, descendant, next and following sibling – as well as equality, inequality, function symbols, and wildcard.

We denote classes of schema mappings by $\mathrm{SM}(\sigma)$, where σ is a signature indicating which of the above features are present in dependencies; i.e.,

$$\sigma \ \subseteq\ \{\downarrow, \downarrow^+, \rightarrow, \rightarrow^+, =, \neq, \_, \mathsf{Fun}\}.$$

We refer to the usual navigational axes as $\downarrow$ (child), $\downarrow^+$ (descendant), $\rightarrow$ (next-sibling), $\rightarrow^+$ (following-sibling). Having $=$ in σ means that we can use equalities in patterns (and reuse variables); having $\neq$ in σ means that we can use inequalities. The use of wildcard is allowed if σ contains $\_$, and Skolem functions are allowed if σ contains Fun.

To simplify notations, we use abbreviations:

- $\Downarrow$ for $\{\downarrow, \downarrow^+, \_\}$ (vertical navigation and wildcard);
- $\Rightarrow$ for $\{\rightarrow, \rightarrow^+\}$ (horizontal navigation);
- $\sim$ for $\{=, \neq\}$ (data value comparisons).

Under these notations, $\mathrm{SM}(\Downarrow)$ is precisely the class of mappings studied in [Arenas and Libkin 2008] (as in [Arenas and Libkin 2008], we do not restrict variable reuse in target patterns). In this paper we shall look at other classes, for instance, $\mathrm{SM}(\Downarrow, \Rightarrow)$, $\mathrm{SM}(\Downarrow, \sim)$, and the largest class $\mathrm{SM}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$.

For reasons to be explained shortly, we work extensively with *nested relational schema mappings*, i.e., schema mappings whose target schemas are nested relational

DTDs: non-recursive DTDs with productions of the form $b \to \hat{a}_1 \dots \hat{a}_n$, where all the a_i's are distinct, and $\hat{a}$ can stand for a itself, or a^*, or a^+, or $a? = (a|\epsilon)$. By $\mathrm{SM}^{\mathrm{nr}}(\sigma)$ we denote the class of nested relational schema mappings in $\mathrm{SM}(\sigma)$.

If we use the standard XML encoding of relational databases, then relational schema mappings fall into the class $\mathrm{SM}^{\mathrm{nr}}(\downarrow, =)$.

4. COMPLEXITY OF SCHEMA MAPPINGS

We look now at the membership problem and consider two flavors of it. *Combined complexity of schema mappings* is the complexity of MEMBERSHIP:

PROBLEM:	MEMBERSHIP
INPUT:	a mapping $\mathcal{M}$, trees T, T'
QUESTION:	$(T, T') \in [\![\mathcal{M}]\!]$?

Data complexity of schema mappings is the complexity of MEMBERSHIP with $\mathcal{M}$ fixed:

PROBLEM:	MEMBERSHIP($\mathcal{M}$)
INPUT:	trees T, T'
QUESTION:	$(T, T') \in [\![\mathcal{M}]\!]$?

We start our complexity analysis by looking at two other problems, related with *patterns*. The first problem is the satisfiability for tree patterns. Its input consists of a DTD D and a pattern $\varphi(\bar{x})$ without function symbols; the problem is to check whether there is a tree T that conforms to D and has a match for φ (i.e., $T \models \varphi(\bar{x})$). 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., [Amer-Yahia et al. 2002; Benedikt et al. 2008; Björklund et al. 2008; Hidders 2003]). For the sake of completeness we include in the Appendix a simple proof, that applies to patterns in the way they are defined here.

PROPOSITION 4.1. *The satisfiability problem for tree patterns is* NP*-complete.*

The proof of this fact uses the notion of homomorphism, which we now recall, as it will be useful in many arguments.

DEFINITION 4.2. *Fix a pattern $\varphi(\bar{x}, \bar{y}, \bar{z}) = \pi(\bar{x}, \bar{y}), \alpha(\bar{x}, \bar{z})$ and tuples $\bar{a}, \bar{b}, \bar{c}$. An interpretation of the function symbols F is* admissible *if it makes $\alpha(\bar{a}, \bar{c})$ hold. Fix an admissible interpretation F. A* homomorphism *h from $\varphi(\bar{a}, \bar{b}, \bar{c})$ into a tree T, denoted $h\colon \varphi(\bar{a}, \bar{b}, \bar{c}) \to T$, is a function assigning to each sub-pattern of $\pi(\bar{a}, \bar{b})$ a node of T so that*

(1) $h(//\pi_1)$ is an ancestor of $h(\pi_1)$;

(2) if $h(\ell(\bar{t})[\mu_1, \mu_2, \dots, \mu_m]) = v$, then
- *(a) either $\ell = \_$ or v is labeled with ℓ,*
- *(b) the tuple $\bar{t}$ interpreted under F is the tuple of attributes of v,*
- *(c) if $\mu_i = \pi_1 \rightsquigarrow_1 \pi_2 \rightsquigarrow_2 \cdots \rightsquigarrow_{k-1} \pi_k$ for some $\rightsquigarrow_i \in \{\to, \to^+\}$, then $h(\pi_j)$ are children of v and $h(\pi_1) \rightsquigarrow_1 h(\pi_2) \rightsquigarrow_2 \cdots \rightsquigarrow_{k-1} h(\pi_k)$ in T.*

Observe that for a fixed admissible interpretation F,

$$T \models \varphi(\bar{a}, \bar{b}, \bar{c}) \quad \iff \quad \text{there exists a homomorphism } h\colon \varphi(\bar{a}, \bar{b}, \bar{c}) \to T.$$

The second problem is data and combined complexity of evaluating tree patterns. For data complexity, we fix a pattern φ, and we want to check for a given tree T and a tuple $\bar{a}$ whether $T \models \varphi(\bar{a})$. For combined complexity, the question is the same, but the input includes $T, \bar{a}$ and φ.

Since patterns are essentially conjunctive queries over trees, the data complexity is in LOGSPACE (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.

PROPOSITION 4.3. *The data complexity of tree patterns evaluation is* LOGSPACE*-complete, and the combined complexity is in* PTIME.

Let us go back to mappings. If an interpretation of function symbols is fixed, the membership problem amounts to evaluating a single pattern. Fix a tree S, a mapping $\mathcal{M} = (D_s, D_t, \Sigma)$ with no variables introduced on the target side, and a valuation F of function symbols in $\mathcal{M}$. Consider the following valuated tree pattern

$$\delta_{S,\mathcal{M},F} = \bigwedge \left\{ \psi(\bar{a}) \;\middle|\; \varphi(\bar{x}, \bar{y}) \longrightarrow \psi(\bar{x}) \;\in \Sigma \text{ and } (S, F) \models \varphi(\bar{a}, \bar{b}) \right\}. \tag{3}$$

Its size is bounded by $\|\mathcal{M}\| \cdot |S|^{\|\mathcal{M}\|}$. A straightforward check gives the following.

LEMMA 4.4. *T is a solution for S under $\mathcal{M}$ with the witnessing valuation F if and only if $T \models D_t$ and $(T, F) \models \delta_{S,\mathcal{M},F}$.*

The complexity of XML mappings is quite high, but it matches that of relational mappings with Skolem functions [Fagin et al. 2004; Pichler and Skritek 2011]. We strengthen the lower bounds a bit by showing hardness for mappings without joins.

THEOREM 4.5. *For schema mappings from* $\mathrm{SM}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$, MEMBERSHIP *is* NEXPTIME*-complete, and* MEMBERSHIP$(\mathcal{M})$ *is always in* NP*. Moreover, we get matching lower bounds in both cases already for relational mappings with Skolem functions, but without* $\sim$*.*

PROOF. Let us first see that MEMBERSHIP is in NEXPTIME. Let $\mathcal{M}$ be a mapping, and let S and T be the source and target trees. Checking conformance to DTDs can be done in PTIME. Let us concentrate on dependencies. Denote by A the set of data values used in S or in T. As usually, we can assume that no variables are introduced on the target side of the constraints. By Lemma 4.4, T is a solution for S if there exists a valuation F of function symbols such that $(T, F) \models \delta_{S,\mathcal{M},F}$. In order to compute and evaluate $\delta_{S,\mathcal{M},F}$, it suffices to know the values of all sub-terms occurring in the definition (3). Their number can be bounded by $N = \|\mathcal{M}\| \cdot |S|^{\|\mathcal{M}\|}$. The algorithm non-deterministically chooses for each of those sub-terms a value from $A \cup \{\bot_1, \bot_2, \ldots, \bot_N\}$, where $\bot_i$ are distinct fresh values, checks that the obtained valuation is consistent with the structure of the terms, computes $\delta_{S,\mathcal{M},F}$ and evaluates it on T. By Proposition 4.3, all this can be done in time polynomial in N and the size of the input.

To show hardness, we will provide a reduction from the following NEXPTIME-hard tiling problem [Papadimitriou 1994]: Given a finite list of tile types $\mathcal{T} =$

$\langle t_0, t_1, \ldots, t_k\rangle$ together with horizontal and vertical adjacency relations $H, V \subseteq \mathcal{T}^2$ and a number n in unary, decide if there exists a tiling of the $2^n \times 2^n$-grid such that a t_0 tile occurs in the top left position, a t_k tile occurs in the bottom right position, and all adjacency relationships are respected.

We give a reduction to the relational case. Hardness for XML follows via the standard reduction. We make additional effort to avoid using joins in the mappings.

Take an instance of the tiling problem. The source instance is

$$\big\{\mathit{False}(0), \mathit{True}(1), \mathit{Eq}(0,0), \mathit{Eq}(1,1)\} \cup \{N_i(0, \underbrace{1, \ldots, 1}_{i-1}, 1, \underbrace{0, \ldots, 0}_{i-1}) \;\big|\; i = 1, 2, \ldots, n\big\}$$

and the target instance contains the adjacency relations H and V together with $\{B(t_0), E(t_k)\}$. Let us now define the set of dependencies. Intuitively, we encode tiles as values of a function f. The tile on position (i, j) is encoded as $f(\langle i\rangle_{bin}\langle j\rangle_{bin})$. We need to check the adjacency relations. This can be done because we can express incrementation on n-bit words. For each $j = 1, 2, \ldots n$ add dependencies

$$\begin{aligned}
\mathit{Eq}(x_1,u_1),\ldots,\mathit{Eq}(x_j,u_j),\mathit{Eq}(y_1,v_1),\ldots,\mathit{Eq}(y_n,v_n)\\
N_{n-j}(x_{j+1},\ldots,x_n,u_{j+1},\ldots,u_n)
\end{aligned} \longrightarrow H(f(\bar{x},\bar{y}), f(\bar{u},\bar{v}))\,,$$

$$\begin{aligned}
\mathit{Eq}(x_1,u_1),\ldots,\mathit{Eq}(x_n,u_n),\mathit{Eq}(y_1,v_1),\ldots,\mathit{Eq}(y_j,v_j)\\
N_{n-j}(y_{j+1},\ldots,y_n,v_{j+1},\ldots,v_n)
\end{aligned} \longrightarrow V(f(\bar{x},\bar{y}), f(\bar{u},\bar{v}))\,.$$

We also need to check that we begin and end properly:

$$\begin{aligned}
\mathit{False}(x_1),\ldots,\mathit{False}(x_n),\mathit{False}(y_1),\ldots,\mathit{False}(y_n) &\longrightarrow B(f(\bar{x},\bar{y}))\,,\\
\mathit{True}(x_1),\ldots,\mathit{True}(x_n),\mathit{True}(y_1),\ldots,\mathit{True}(y_n) &\longrightarrow E(f(\bar{x},\bar{y}))\,.
\end{aligned}$$

In terms of data complexity, the algorithm described above is in NP: the size of $\delta_{S,\mathcal{M},F}$ is polynomial for every fixed mapping. Let us now prove hardness. Recall that if we replace the $2^n \times 2^n$-grid with the $n \times n$-grid in the tiling problem considered above, we end up with an NP-complete problem. The idea of the reduction from this new problem to MEMBERSHIP($\mathcal{M}$) is the same as in the reduction above, and the implementation is even simpler. As the source instance we take $\{B(1), E(n), \mathit{Eq}(1,1), \mathit{Eq}(2,2), \ldots, \mathit{Eq}(n,n), N(1,2), N(2,3), \ldots, N(n-1,n)\}$. The target instance is just like before. The dependencies are

$$\begin{aligned}
N(x,u), \mathit{Eq}(y,v) &\longrightarrow H(f(x,y), f(u,v))\,, & B(x), B(y) &\longrightarrow B(f(x,y))\,,\\
N(x,u), \mathit{Eq}(y,v) &\longrightarrow V(f(y,x), f(v,u))\,, & E(x), E(y) &\longrightarrow E(f(x,y))\,.
\end{aligned}$$

It is routine to verify the correctness of the reduction. □

The main source of hardness are Skolem functions. The first reduction above uses function symbols of unbounded arity, but they can be easily encoded with binary functions by using either nested terms or equality. If we bound the arity of terms (the total number of occurrences of variables) and forbid explicit equalities between terms (repetition of variables is allowed), the combined complexity drops to the third level of the polynomial hierarchy. The data complexity is not affected, since the mappings used in the second reduction above satisfy these restrictions. Complete proof of the following result is in the Appendix.

THEOREM 4.6. *For schema mappings from* $\mathrm{SM}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$ *with bounded arity of terms and no explicit equality,* MEMBERSHIP *is* Σ_3^p*-complete, and the lower bound holds already for relational mappings without* $\sim$*.*

If we forbid Skolem functions altogether, the data complexity of mappings drops very low, but combined complexity is still complete for the second level of the polynomial hierarchy, like in the relational case [Gottlob and Senellart 2010]. If we additionally bound the number of variables in dependencies, even combined complexity is polynomial.

THEOREM 4.7. *For schema mappings from* $\mathrm{SM}(\Downarrow, \Rightarrow, \sim)$*,* MEMBERSHIP$(\mathcal{M})$ *is* LOGSPACE*-complete, while* MEMBERSHIP *is* Π_2^p*-complete in general and in* PTIME *if the maximum number of variables per pattern is fixed.*

PROOF. (1) Conformance to a fixed DTD can be checked in LOGSPACE. It remains to show that we can check LOGSPACE if S and T satisfy a single constraint $\varphi(\bar{x}, \bar{y}) \longrightarrow \psi(\bar{x}, \bar{z})$. Let $\bar{x} = x_1, x_2, \ldots, x_k$, $\bar{y} = y_1, y_2, \ldots, y_\ell$, and $\bar{z} = z_1, z_2, \ldots, z_m$. Let A be the set of data values used in S or T. We need to check that for each $\bar{a} \in A^k$ and each $\bar{b} \in A^\ell$ such that $S \models \varphi(\bar{a}, \bar{b})$ there exists $\bar{c} \in A^m$ such that $T \models \psi(\bar{a}, \bar{c})$. Since the numbers k, ℓ, m are fixed (as parts of the fixed mapping), the space needed for storing all three valuations is logarithmic in the size of S and T. Using Proposition 4.3 we obtain a LOGSPACE algorithm by simply iterating over all possible valuations $\bar{a}$, $\bar{b}$, and $\bar{c}$. LOGSPACE-hardness follows from Proposition 4.3.

(2) First let us see that the problem is in Π_2^p. Consider the following algorithm for the complementary problem: guess a constraint $\varphi(\bar{x}, \bar{y}) \longrightarrow \psi(\bar{x}, \bar{z})$ and a valuation $\bar{a}, \bar{b}$ of variables used in φ, and check that $S \models \varphi(\bar{a}, \bar{b})$ and $T \not\models \psi(\bar{a}, \bar{z})$. By Proposition 4.3, the first check is polynomial. The second check however involves a tree pattern possibly containing free variables, so it can only be done in coNP. Altogether the algorithm is in Π_2^p. The Π_2^p lower bound for relational mappings using neither Skolem functions nor equality [Gottlob and Senellart 2010] carries over to $\mathrm{SM}(\downarrow)$ via the standard encoding. The original reduction can be obtained as a natural modification of the from Theorem 4.6.

(3) Proceed just like in (1). The numbers of variables per pattern is bounded, so there are only polynomially many possible valuations. Hence, we may iterate over all of them using algorithm from Proposition 4.3 to check $S \models \varphi(\bar{a}, \bar{b})$ and $T \models \psi(\bar{a}, \bar{c})$. □

5. QUERY ANSWERING

5.1 Query answering problem

The fundamental problem of data exchange is answering queries over target data in a way consistent with the source data. Inspired by the research on the relational case [Barceló 2009; Fagin et al. 2003; Kolaitis 2005], we study query answering for conjunctive queries and their unions. Conjunctive queries over trees are normally represented with tree patterns [Gottlob et al. 2006; Björklund et al. 2007; 2008]. Thus, for querying XML documents we use the same language as for the dependencies: tree patterns augmented with equalities as well as inequalities, to capture the

analog of relational conjunctive queries with inequalities. And, of course, we allow projection.

That is, a query is an expression of the form

$$\exists \bar{x}\ \varphi\,,$$

where φ is a pattern using no function symbols, such that each free variable satisfies the safety condition (see Definition 3.1). The semantics is defined in the standard way. The output of the query is the set of those valuations of free variables that make the query hold true. This class of queries is denoted by **CTQ** (conjunctive tree queries). Note that **CTQ** is indeed closed under conjunctions, due to the semantics of λ, λ' in patterns.

We also consider unions of such queries: **UCTQ** denotes the class of queries of the form $Q_1(\bar{x}) \cup \cdots \cup Q_m(\bar{x})$, where each Q_i is a query from **CTQ**. Like for schema mappings, we write $\mathbf{CTQ}(\sigma)$ and $\mathbf{UCTQ}(\sigma)$ for $\sigma \subseteq \{\_\ \downarrow, \downarrow^+, \rightarrow, \rightarrow^+, =, \neq\}$ to denote the subclass of queries using only the symbols from σ.

Consider a source DTD D_s

$$\begin{array}{rl@{\qquad}rl} \texttt{europe} & \rightarrow \texttt{country}^* & \texttt{country} & : @\texttt{name} \\ \texttt{country} & \rightarrow \texttt{ruler}^* & \texttt{ruler} & : @\texttt{name} \end{array}$$

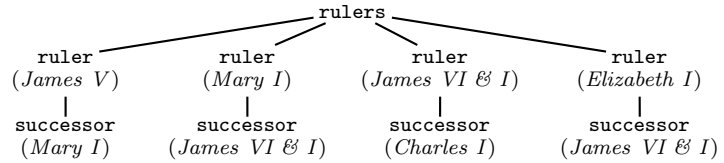
and a target DTD D_t

$$\begin{array}{rl@{\qquad}rl} \texttt{rulers} & \rightarrow \texttt{ruler}^* & \texttt{ruler} & : @\texttt{name} \\ \texttt{ruler} & \rightarrow \texttt{successor} & \texttt{successor} & : @\texttt{name} \end{array}$$

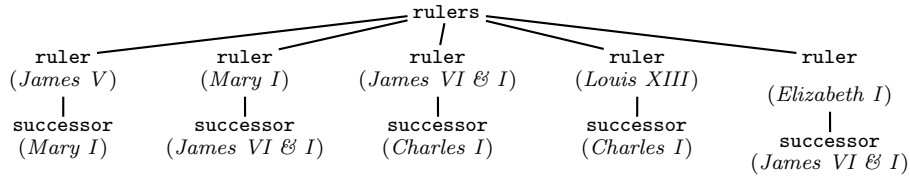
Assuming the rulers are stored in the chronological order on the source side, a natural schema mapping $\mathcal{M}$ might be defined with the following dependency:

$$\texttt{europe/country}[\texttt{ruler}(x) \rightarrow \texttt{ruler}(y)] \longrightarrow \texttt{rulers/ruler}(x)/\texttt{successor}(y)\,.$$

Suppose that the source tree T_1 is the tree in Fig. 1 given in the introduction. A natural solution for T_1 is the following tree T_2:



As we already know, every tree obtained from T_2 by adding new children with arbitrary data values, or by permuting the existing children, is also a solution for T_1. For instance, T_3, shown below, is also a solution for T_1:



A query one might ask over the target database is to list the rulers who were successors to more that one ruler. This would be expressed by the following conjunctive

query MultiSucc:

$$\exists x\ \exists y\ \left(\texttt{rulers}\begin{bmatrix}\texttt{ruler}(x)/\texttt{successor}(z),\\ \texttt{ruler}(y)/\texttt{successor}(z)\end{bmatrix},\ x \neq y\right)$$

On T_2 the query MultiSucc would return $\{$*"James VI & I"*$\}$, and on T_3 the answer would be $\{$*"James VI & I"*, *"Charles I"*$\}$. What is the right answer then? Following [Arenas and Libkin 2008; Fagin et al. 2003], we adapt the *certain answers semantics*.

For a mapping $\mathcal{M}$, a query Q, and a source tree T conforming to D_s, we return the tuples which would be returned for every possible solution:

$$\mathit{certain}_{\mathcal{M}}(Q,T) = \bigcap \big\{ Q(T') \;\big|\; T' \text{ is a solution for } T \text{ under } \mathcal{M} \big\}.$$

The subscript $\mathcal{M}$ is omitted when it is clear from the context. (Note that our queries output sets of tuples rather than trees, so we can define certain answers by taking the intersection of the answers over all solutions). In our running example,

$$\mathit{certain}_{\mathcal{M}}(\text{MultiSucc}, T_1) = \{\text{ "James VI \& I"}\}.$$

Note that when Q is a Boolean query, $\mathit{certain}_{\mathcal{M}}(Q,T)$ is true if and only if Q is true for all the solutions.

Fix an XML schema mapping $\mathcal{M}$ and a query Q. We are interested in the following decision problem.

PROBLEM:	CERTAIN$_{\mathcal{M}}(Q)$
INPUT:	a tree T, a tuple $\bar{s}$
QUESTION:	$\bar{s} \in \mathit{certain}_{\mathcal{M}}(Q,T)$?

We now recall what is already known about simple settings based on downward navigation [Arenas and Libkin 2008], i.e., mappings from SM($\Downarrow, =$) and queries from **UCTQ**($\Downarrow, =$). The problem is in coNP, and could be coNP-hard. To reduce the complexity, one can vary three parameters of the problem: DTDs, dependencies, and queries. It turns out that in order to get tractability we have to restrict the first two parameters simultaneously.

The general idea behind the restrictions is to avoid any need for guessing where patterns could be put in a target tree. For that, the mapping has to be as specific as possible. In terms of DTDs this restriction is well captured by the notion of nested relational DTDs: there is no explicit disjunction, and each node has either at most one a-labeled child or arbitrary many a-labeled children for each a. But guessing is also involved whenever wildcard or descendant is used. The following restricts their use.

DEFINITION 5.1. (see [Arenas and Libkin 2008]) *We say that a schema mapping is* fully specified *if it uses neither* _ *nor* $\downarrow^+$ *in target patterns in dependencies.*

The following summarizes known results on simple mappings that only use vertical navigation.

FACT 5.2. (see [Arenas and Libkin 2008]) *For every schema mapping* $\mathcal{M}$ *in* SM($\Downarrow, =$) *and every query* $Q \in$ **CTQ**($\Downarrow, =$)

(1) $\text{CERTAIN}_{\mathcal{M}}(Q)$ *is in coNP,*

(2) $\text{CERTAIN}_{\mathcal{M}}(Q)$ *is in PTIME, if* $\mathcal{M}$ *is nested-relational and fully specified.*

Moreover, if any of the hypotheses in (2) is dropped, one can find a mapping $\mathcal{M}$ *and a query* Q *such that* $\text{CERTAIN}_{\mathcal{M}}(Q)$ *is coNP-complete.*

Note that item (2) includes, as a special case, the tractability of the certain answers problem for conjunctive queries in relational data exchange. Indeed, it says that answering queries from $\mathbf{CTQ}(\Downarrow, =)$ (and even unions of those) is tractable for mappings from $\text{SM}^{\text{nr}}(\downarrow, =)$, and as we remarked earlier, relational schema mappings fall into this class under the natural representation of relations as flat trees.

The result of [Arenas and Libkin 2008] is actually more precise. For mappings from $\text{SM}^{\text{nr}}(\downarrow, =)$ there is a dichotomy in the first parameter: if DTDs allow enough disjunction, the problem is coNP-hard, otherwise it is polynomial. The exact class of tractable DTDs is the one using so called univocal regular expressions (see [Arenas and Libkin 2008] for a rather involved definition). Intuitively, it extends nested-relational DTDs with a very week form of disjunction. Query answering in this case is based on constructing a specific instance using a chase procedure, and the use of disjunction in DTDs is limited so as to keep the chase polynomial.

Our goal Given the results of [Arenas and Libkin 2008], we must stay with a restricted class of DTDs and fully specified dependencies to have any hope of getting tractability of query answering. Hence, our questions are:

(1) How bad could the complexity of $\text{CERTAIN}_{\mathcal{M}}(Q)$ be if we extend the classes $\text{SM}^{\text{nr}}(\Downarrow, =)$ of mappings and $\mathbf{CTQ}(\Downarrow, =)$ of queries?

(2) Can we extend the classes $\text{SM}^{\text{nr}}(\Downarrow, =)$ of mappings and $\mathbf{CTQ}(\Downarrow, =)$ of queries with new features while retaining tractable query answering?

In the next section, we show that the coNP bound is not broken by adding new features as long as inequality is not allowed in queries. With inequality allowed, the problem quickly becomes undecidable.

5.2 General upper bound

In the previous section we have sketched the tractability frontier for simple mappings and simple queries. Now, we would like to see what can be done to extend the tractable case with horizontal navigation and data comparisons. But first, we need to verify whether the upper bound remains the same with all the new features.

It is known that containment for conjunctive queries with data equalities and inequalities is undecidable over trees [Björklund et al. 2008]. From this it already follows that $\text{CERTAIN}_{\mathcal{M}}(Q)$ cannot be uniformly decidable if inequality is allowed. In fact, it can be undecidable already for a fixed $\mathcal{M}$ and Q that use either child/next-sibling or child/descendant navigation (see Appendix).

PROPOSITION 5.3. *If* σ *contains either* $\downarrow, \downarrow^+$ *or* $\downarrow, \rightarrow$*, then there exist a mapping* $\mathcal{M} \in \text{SM}(\sigma)$ *and a query* $Q \in \mathbf{CTQ}(\sigma, \sim)$ *such that* $\text{CERTAIN}_{\mathcal{M}}(Q)$ *is undecidable. In the case of* $\downarrow, \rightarrow$*, this holds even under restriction to nested-relational DTDs.*

Unlike in some other cases (e.g., relational queries under the closed world semantics [Abiteboul et al. 1991]), the coNP upper bound on certain answers is nontrivial

even in the case of simple child-based mappings (see page 9). Now we show that we can recover the upper bound for much more expressive mappings. This can be done by casting the problem as a special case of query answering over incomplete XML documents, for which the coNP bound was recently proved [Barceló et al. 2010]. We point out that the combined complexity of the procedure proposed in [Barceló et al. 2010] is bad (worse then exponential), but probably can be improved.

PROPOSITION 5.4. *For every schema mapping* $\mathcal{M}$ *from* $\mathrm{SM}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$ *and every query* Q *from* $\mathbf{UCTQ}(\Downarrow, \Rightarrow, =)$*, the complexity of* $\textsc{certain}_{\mathcal{M}}(Q)$ *is in coNP.*

PROOF. Take a query $Q \in \mathbf{UCTQ}(\Downarrow, \Rightarrow, \sim)$, a mapping $\mathcal{M} = (D_s, D_t, \Sigma)$, and a source tree S conforming to D_s. Without loss of generality we can assume that Q is Boolean and that Σ does not introduce fresh variables on the target side. By Lemma 4.4, the certain answer to Q is *false* iff there exists a tree T such that $T \not\models Q$, $T \models D_t$, $T \models \delta_{S,\mathcal{M},F}$ for some witnessing valuation F, where

$$\delta_{S,\mathcal{M},F} = \bigwedge \left\{ \psi(\bar{a}) \;\middle|\; \varphi(\bar{x}, \bar{y}) \longrightarrow \psi(\bar{x}) \ \in \Sigma \text{ and } (S, F) \models \varphi(\bar{a}, \bar{b}) \right\} .$$

Like in the proof of Theorem 4.5, for all sub-terms in the definition of $\delta_{S,\mathcal{M},F}$ we can guess consistent values from the set of data values used in S or a set of nulls $\{\bot_1, \bot_2, \ldots, \bot_N\}$, where $N = \|\mathcal{M}\| \cdot |S|^{\|\mathcal{M}\|}$, and compute $\delta_{S,\mathcal{M},F}$ in polynomial time. Assume that all the equalities and inequalities in $\delta_{S,\mathcal{M},F}$ hold (if not, reject), and let π be the pure pattern underlying $\delta_{S,\mathcal{M},F}$. Note that π contains no variables, only data values. Now, it remains to see if there exists a tree T such that $T \models D_t$, $T \models \pi$ and $T \not\models Q$, which is exactly an instance of the complement of the certain answers problem in the incomplete information scenario considered in [Barceló et al. 2010]. Hence, it can be done in NP. □

Having seen that the upper bound is not affected as long as queries do not use inequality, we can move on to the second question: Can we find $\sigma_1 \supseteq \{\Downarrow, =\}$ and $\sigma_2 \supseteq \{\Downarrow, =\}$ such that $\textsc{certain}_{\mathcal{M}}(Q)$ is tractable for all fully specified $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\sigma_1)$ and $Q \in \mathbf{UCTQ}(\sigma_2)$? In what follows we show that it is almost impossible to extend the query language, but schema mappings can be extended with new features—under certain restrictions.

5.3 Extending the query language

We shall now see that even for simple mappings, $\mathrm{SM}^{\mathrm{nr}}(\downarrow, =)$, the query language cannot be extended beyond $\mathbf{UCTQ}(\Downarrow, =)$.

First, we note that inequality cannot be allowed. We have already seen that it can lead to undecidability for relatively modest mapping languages. But already in the relational case there are conjunctive queries with just two inequalities for which the problem is coNP-hard (with one inequality the problem is tractable) [Fagin et al. 2003; Mądry 2005]. Since the usual translation from the relational setting to the XML setting produces mappings from $\mathrm{SM}^{\mathrm{nr}}(\downarrow, =)$, we have the following result.

COROLLARY 5.5. *There exist a schema mapping* $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\downarrow, =)$ *and a query* Q *in* $\mathbf{CTQ}(\downarrow, =, \neq)$ *such that* $\textsc{certain}_{\mathcal{M}}(Q)$ *is coNP-complete.*

Similarly, extending the query language with any form of horizontal navigation leads to intractability even for the simplest mappings, $\mathrm{SM}^{\mathrm{nr}}(\downarrow)$.

PROPOSITION 5.6. *There exist a schema mapping* $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\downarrow)$*, a query* $Q_1 \in \mathbf{CTQ}(\downarrow, \rightarrow, =)$*, and a query* $Q_2 \in \mathbf{CTQ}(\downarrow, \rightarrow^+, =)$ *such that both* $\textsc{certain}_{\mathcal{M}}(Q_1)$ *and* $\textsc{certain}_{\mathcal{M}}(Q_2)$ *are coNP-complete.*

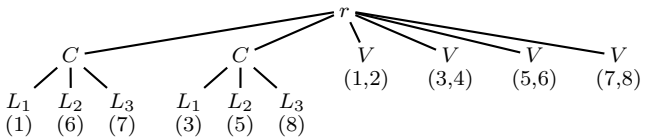
PROOF. The coNP upper bound follows from Proposition 5.4. For the lower bound we only give the proof for $\rightarrow$; the proof for $\rightarrow^+$ can be obtained by replacing $\rightarrow$ with $\rightarrow^+$ in our argument.

We give an XML schema mapping $\mathcal{M}$ and a Boolean query Q such that 3SAT is reducible to the complement of $\textsc{certain}_{\mathcal{M}}(Q)$, i.e., for each 3SAT instance θ

$$\mathit{certain}_{\mathcal{M}}(Q, T_\theta) \text{ is } \mathit{false} \text{ iff } \theta \text{ is satisfiable,}$$

where T_θ is a tree encoding of θ described below.

Suppose we are given a 3-CNF formula $\theta = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \ell_{ij}$, where ℓ_{ij} is a literal, and in each clause all three literals are different. The tree encoding, T_θ, is best explained on a concrete example. A formula $(x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$ is encoded as



Each V node has two attribute values encoding a variable and its negation with two different values. For example, the node $V(1, 2)$ indicates that x_1 is encoded by the data value 1 and $\neg x_1$ by 2. Also for each clause in the formula we have a C node that has three children labeled L_1, L_2, L_3. Each L_i holds the data value encoding the ith literal in the clause. In the example above, the second literal of the first clause is $\neg x_3$ and hence the data value of L_2 under the first C node is 6.

Let us now describe the mapping. In accordance with the encoding, let D_s be

$$\begin{aligned} r &\to C^*V^* & V &: @a_1, @a_2 \\ C &\to L_1L_2L_3 & L_i &: @b \end{aligned}$$

where $i = 1, 2, 3$. The target DTD D_t is defined as

$$\begin{aligned} r &\to C^*V^* & V &: @a_1, @a_2 \\ C &\to L^* & L &: @b \end{aligned}$$

The dependencies Σ essentially copy T_θ in the target, but allow the reordering of children under each C node with the use of ‘,’(comma). This reordering corresponds to ‘choosing one literal per clause’ mentioned earlier. Intuitively, a literal is chosen if its copy has more than two following siblings. Since each C node has three L-children with different data values, at least one literal is chosen for each clause.

$$r[C[L_1(x), L_2(y), L_3(z)]] \to r[C[L(x), L(y), L(z)]]$$
$$r[V(x, y)] \to r[V(x, y)]$$

Thus, a solution gives a (partial) valuation satisfying θ, provided that the choices are consistent. This is taken care of by the query: it is true if a variable and its negation are contained among the chosen literals. The query is

$$\exists x \exists y \ \big(r\big[V(x, y), C[L(x) \to L \to L], C[L(y) \to L \to L]\big]\big) \ .$$

One easily proves that $\text{certain}_{\mathcal{M}}(Q, T_\theta) = \mathit{false}$ if and only if θ is satisfiable. □

We have seen that even if we stick to child-based mappings, we cannot extend the query language. But perhaps we can find a more suitable class of mappings? Observe that mappings in the reductions violated the idea behind the principle of being fully specified (although not the formal definition), as queries used horizontal navigation, and yet mappings did not specify it completely, by allowing the set constructor λ, λ' in patterns (see page 10). Such non-determinism in placing patterns in target trees leads to intractability. So it seems natural to restrict the use of this non-determinism and require that the mappings specify the relative ordering of each two sub-patterns which start at children of the same node. An even stronger restriction could demand that the $\to$ relation among the siblings be specified completely. Unfortunately, unlike in [Arenas and Libkin 2008], such restrictions do *not* lead to tractability. Concrete examples can be found in the Appendix.

5.4 Extending the mapping language

In this section we show that one can allow restricted use of horizontal ordering, function symbols, and data value comparisons in the mappings without losing tractability, provided that we stick to the basic query language.

Let us first see that with next-sibling query answering is intractable.

PROPOSITION 5.7. *There exists a mapping* $\mathcal{M} \in \text{SM}^{\text{nr}}(\downarrow, \to)$ *and a Boolean query* $Q \in \mathbf{CTQ}(\downarrow, =)$ *such that* $\text{CERTAIN}_{\mathcal{M}}(Q)$ *is* coNP*-complete.*

PROOF. The upper bound follows from Proposition 5.4. To obtain the lower bound, we modify the reduction from 3SAT is to the complement of $\text{CERTAIN}_{\mathcal{M}}(Q)$ given in Proposition 5.6. The source DTD D_s and the tree encoding the propositional formula are identical. The target DTD D_t is similar to D_s:

$$\begin{aligned} &r \to C^*V^* && V\colon @a_1, @a_2\\ &C \to ML^*N && L\colon @b\\ &L \to K? \end{aligned}$$

The dependencies Σ copy the tree T_θ in the target, transforming L_i labels into L, adding M and N nodes w.r.t. the target DTD and adding a K-node under at least one L-node for each C-node:

$$\begin{aligned} r/C[L_1(x), L_2(y), L_3(z)] &\longrightarrow r/C[M \to L(x) \to L(y) \to L(z) \to N, L/K]\,,\\ r/V(x,y) &\longrightarrow r/V(x,y)\,. \end{aligned}$$

Intuitively, K means that the literal is set to *true*. As the mapping ensures that at least one literal is chosen for each clause, a solution gives a (partial) valuation satisfying θ, provided that we have chosen consistently. This is verified by the query

$$\exists x \exists y\; r\big[V(x,y), C/L(x)/K, C/L(y)/K\big]\,.$$

Clearly, the query is true if a variable and its negation are chosen. □

The mechanism leading to intractability is actually the one motivating the definition of nested-relational DTDs: by using the pattern $r/C[M \to L(x) \to L(y) \to L(z) \to N, L/K]$ in the mapping above we bound the number of L-children and

thus enforce that the location of the K-node is guessed. We will eliminate this kind of behavior. We say that a sequence $\kappa = \pi_1 \to \pi_2 \to \cdots \to \pi_n$ is a *segment* of a sequence μ, if μ is of the form $\mu' \to^+ \kappa \to^+ \mu''$, $\mu' \to^+ \kappa$, $\kappa \to^+ \mu''$, or κ. Each $\pi_j \to \pi_{j+1} \to \cdots \to \pi_{j'}$ is called a *sub-segment* of μ. The sequence μ is *bounding* for a production $\sigma \to \hat{\ell}_1\, \hat{\ell}_2\, \ldots\, \hat{\ell}_m$ if it has a sub-segment κ such that

$$\mathit{head}(\kappa) = \tau \to \underbrace{\ell_i \to \ell_i \to \cdots \to \ell_i}_{k} \to \tau'$$

for $k > 1$, $\ell_i \notin \{\tau, \tau'\}$ and $\hat{\ell}_i = \ell_i^*$ or $\hat{\ell}_i = \ell_i^+$, where $\mathit{head}(\kappa)$ is defined as

$$\begin{aligned} \mathit{head}(\tau(\bar{s})[\lambda]) &= \tau\,, \\ \mathit{head}(\pi_1 \rightsquigarrow_1 \cdots \rightsquigarrow_{m-1} \pi_m) &= \mathit{head}(\pi_1) \rightsquigarrow_1 \cdots \rightsquigarrow_{m-1} \mathit{head}(\pi_m)\,. \end{aligned}$$

A mapping is *non-bounding* if none of its target patterns contains a sub-pattern of the form $\sigma(\bar{s})[\lambda, \mu, \lambda']$ where μ is a bounding sequence for the production for σ in the target DTD. We will use this restriction later.

With inequality and function symbols query answering is also intractable even for the simplest query language.

PROPOSITION 5.8. *There exists a mapping* $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\downarrow, =, \neq, \mathsf{Fun})$ *and a Boolean query* $Q \in \mathbf{CTQ}(\downarrow)$ *such that* $\textsc{certain}_{\mathcal{M}}(Q)$ *is* coNP*-complete.*

PROOF. In fact, we can prove the coNP-hardness already for relational mappings. From this the claim follows via standard reduction.

Take a 3-CNF formula $\bigwedge_{i=1}^{m} X_i^1 \vee X_i^2 \vee X_i^3$, with $X_i^\ell \in \{x_j, \bar{x}_j \mid j = 1, \ldots, n\}$. We encode each pair of literals $x_i, \bar{x}_i$ as i and $n+i$. Let the source instance be $\{\mathit{var}(1, n+1), \mathit{var}(2, n+2), \ldots, \mathit{var}(n, 2n)\} \cup \{\mathit{clause}(k_i^1, k_i^2, k_i^3) \mid i = 1, \ldots, m\}$, where $k_i^\ell = p$ if $X_i^\ell = x_p$, and $k_i^\ell = n + p$ if $X_i^\ell = \bar{x}_p$. The target schema only contains a unary relation R.

The mapping guesses truth values for the literals by means of two function symbols, f and g. If $f(d) = g(d)$, then the literal encoded by d is *true*, otherwise it is *false*. If the guess is inconsistent, or leaves some clause false, the mapping demands an R tuple in the target. The dependencies are

$$\begin{aligned} \mathit{var}(x, y), f(x) = g(x), f(y) = g(y) &\longrightarrow R(u)\,, \\ \mathit{var}(x, y), f(x) \neq g(x), f(y) \neq g(y) &\longrightarrow R(u)\,, \\ \mathit{clause}(x, y, z), f(x) \neq g(x), f(y) \neq g(y), f(z) \neq g(z) &\longrightarrow R(u)\,. \end{aligned}$$

The query Q is simply $\exists x\, R(x)$. Clearly, there exists a valuation satisfying the formula iff there exist functions f, g such that $\emptyset$ is a valid solution. Hence, the certain answer is *true* iff the formula is not satisfiable. □

To regain tractability we restrict the conditions put on the guessed values. A mapping $\mathcal{M}$ is $\sim$*-monotonic* if it does not contain inequalities between nontrivial terms on the source side, i.e., dependencies are of the form

$$\pi(\bar{x}), \alpha_=(\bar{x}), \alpha_\neq(\bar{x}) \longrightarrow \pi'(\bar{x}), \alpha'(\bar{x})$$

where $\alpha_=$ is a conjunction of *equalities among terms* over the variables in $\bar{x}$, and $\alpha_\neq$ is a conjunction of *inequalities among variables* in $\bar{x}$. Roughly speaking, this

restriction makes dependencies monotonic with respect to equalities between nulls. Note that this setting extends both: mappings with function symbols but without inequalities, and mappings with inequalities but without function symbols.

Throughout the rest of this section we work exclusively with fully specified (i.e., using neither _ nor $\downarrow^+$ on the target side), non-bounding, and $\sim$-monotonic mappings in $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$. We are aiming at the following tractability result.

THEOREM 5.9. *Let $\mathcal{M}$ be a fully specified, non-bounding, and $\sim$-monotonic mapping in $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$. Then $\textsc{certain}_{\mathcal{M}}(Q)$ is in* PTIME *for every $Q \in \mathbf{UCTQ}(\Downarrow, =)$.*

Towards this end we extend the techniques from [Arenas and Libkin 2008] based on the notion of universal solutions, a standard concept in data exchange [Fagin et al. 2003]. As a first step, observe that queries in $\mathbf{UCTQ}(\Downarrow, =)$ are completely ignorant to the sibling ordering: if a solution T' is obtained from a solution T by a permutation of siblings, then $Q(T) = Q(T')$ for each $Q \in \mathbf{UCTQ}(\Downarrow, =)$. We now generalize and strengthen this property in terms of homomorphisms that ignore sibling order, and reduce query answering to the problem of constructing "unordered" universal solutions.

As usually, it is convenient to distinguish between *constants*, i.e., data values appearing on the source side, and *nulls*, representing the unknown values invented to fill in missing values on the target side. We write Const and Nulls for these sets.

An *unordered homomorphism* $h : T \to T'$ consists of a function $h_{\mathrm{Node}} \colon \mathrm{dom}(T) \to \mathrm{dom}(T')$ preserving the root, child relation and labeling, and a function $h_{\mathrm{Null}} \colon \mathsf{Nulls} \to \mathsf{Nulls} \cup \mathsf{Const}$, extended to Const as identity, such that if v stores a tuple $\bar{t}$ of entries from $\mathsf{Nulls} \cup \mathsf{Const}$, $h_{\mathrm{Node}}(v)$ stores $h_{\mathrm{Null}}(\bar{t})$.

LEMMA 5.10. *If there exists an unordered homomorphism $h \colon T \to T'$ between solutions T and T', then $Q(T) \subseteq Q(T')$ for each $Q \in \mathbf{UCTQ}(\Downarrow, =)$.*

U is an *unordered universal solution* for S under $\mathcal{M}$ if it is a solution for S, and for each other solution T there is an unordered homomorphism from U to T.

LEMMA 5.11. *Let $\mathcal{M} \in \mathrm{SM}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$ and let U be an unordered universal solution for a source tree S. For each $Q \in \mathbf{UCTQ}(\Downarrow, =)$ and each tuple $\bar{a}$,*

$$\bar{a} \in \mathit{certain}_{\mathcal{M}}(Q, S) \quad \mathit{iff} \quad \bar{a} \in Q(U)\,.$$

What Lemma 5.11 means is that certain answers to queries from $\mathbf{UCTQ}(\Downarrow, =)$ can be computed by naïve evaluation on any unordered universal solution. Thus, in order to prove Theorem 5.9, it suffices to show how to construct unordered universal solutions in PTIME. The rest of this section describes a solution to this problem.

By Lemma 4.4, T is a solution for S under $\mathcal{M} = (D_s, D_t, \Sigma)$ with a witnessing valuation F iff $T \models D_t$ and $T \models \delta_{S,\mathcal{M},F}$, where

$$\delta_{S,\mathcal{M},F} = \bigwedge \left\{ \psi(\bar{a}) \;\middle|\; \varphi(\bar{x}, \bar{y}) \longrightarrow \psi(\bar{x}) \;\in \Sigma \text{ and } (S, F) \models \varphi(\bar{a}, \bar{b}) \right\}\,.$$

Consequently, constructing an unordered universal solution amounts to finding a "universal" tree satisfying a certain pattern, as soon as we have the correct witnessing valuation. We first show how to build such a tree for a given pattern, and then how to find the witnessing valuation.

We start by adjusting patterns to DTDs: for a fully specified non-bounding pattern φ and a nested-relational DTD D, we construct a pattern φ' such that

—φ' is implied by φ on trees conforming to D,
—φ' seen as a tree conforms to D and satisfies φ.

We turn φ into φ' by means of two operations, called *completion* and *merging*.

We say that a pattern φ is *complete* with respect to a nested relational DTD D if each of its nodes has all the children required by the DTD. More precisely, a label τ is *missing* in a sub-pattern $\sigma(\bar{t})[\lambda]$ if τ occurs in the production for σ as τ or τ^+, but λ cannot be presented as $\lambda_1, \mu_1 \rightsquigarrow \tau(\bar{s})[\lambda'] \rightsquigarrow \mu_2, \lambda_2$. A pattern is complete if no label is missing in its sub-patterns.

Completion simply extends the pattern with all missing labels and thus makes it complete. For a DTD D and a tuple of variables $\bar{x}$, the operation $\mathrm{cpl}_D^{\bar{x}}$ is defined inductively as follows:

$$\begin{aligned}
\mathrm{cpl}_D^{\bar{x}}(\lambda, \lambda') &= \mathrm{cpl}_D^{\bar{x}}(\lambda), \mathrm{cpl}_D^{\bar{x}}(\lambda')\,,\\
\mathrm{cpl}_D^{\bar{x}}(\mu \rightsquigarrow \mu') &= \mathrm{cpl}_D^{\bar{x}}(\mu) \rightsquigarrow \mathrm{cpl}_D^{\bar{x}}(\mu')\,,\\
\mathrm{cpl}_D^{\bar{x}}\big(\sigma(\bar{t})[\lambda]\big) &= \sigma(\bar{t})\big[\mathrm{cpl}_D^{\bar{x}}\big(\lambda\big), \mathrm{cpl}_D^{\bar{x}}\big(\tau_1(\bar{t}_1)\big), \mathrm{cpl}_D^{\bar{x}}\big(\tau_2(\bar{t}_2)\big), \ldots, \mathrm{cpl}_D^{\bar{x}}\big(\tau_m(\bar{t}_m)\big)\big]\,,
\end{aligned}$$

where $\tau_1, \ldots, \tau_m$ are the missing labels and $\bar{t}_i = f_1^i(\bar{x}), f_2^i(\bar{x}), \ldots, f_{p_i}^i(\bar{x})$ where p_i is the number of attributes of τ_i and f_j^i are fresh function symbols. Since D is non-recursive, the operation terminates and returns a pattern at most single exponential in the size of D. We write $\mathrm{cpl}_D(\pi)$ for $\mathrm{cpl}_D^{\mathrm{Var}\,\pi}(\pi)$, and $\mathrm{cpl}_D(\pi, \alpha)$ for $\mathrm{cpl}_D(\pi), \alpha$.

It is easy to see that the result of completion is equivalent to the original pattern, up to the valuation of the new function symbols.

Lemma 5.12. *Let D be a nested relational DTD, and let $\varphi(\bar{x})$ be a fully specified tree pattern. For each $T \models D$ and each valuation F there exists a valuation G of the new function symbols such that for all $\bar{a}$*

$$(T, F) \models \varphi(\bar{a}) \quad \textit{iff} \quad (T, F \cup G) \models (\mathrm{cpl}_D\varphi)(\bar{a})\,.$$

The aim of *merging* is to merge all sub-patterns that are always mapped to the same node in trees conforming to the given DTD. More precisely, for a given φ it produces a pattern φ' such that

—φ' admits an injective homomorphism into a tree conforming to D,
—a match for φ implies a match for φ', and an injective match for φ' implies a match for φ (over trees conforming to D).

Fix a nested-relational DTD D. The pattern $\mathrm{mrg}_D(\varphi)$ is built inductively, with new equalities (induced by merging nodes) added to the global set E along the way. In the beginning the set E is empty.

If there is no sibling ordering, the construction is easy. To obtain $\mathrm{mrg}_D(\sigma(\bar{u})[\pi_1, \ldots, \pi_m])$ proceed as follows ($\bar{s} = \bar{t}$ stands for $s_1 = t_1, \ldots, s_d = t_d$).

(1) Return $\perp$ whenever $\mathit{head}(\pi_i) = \tau$ for some τ not used in the production for σ.
(2) For each τ with $\sigma \to \ldots\tau\ldots$ or $\sigma \to \ldots\tau?\ldots$ merge all π_i's with $\mathit{head}(\pi_i) = \tau$:
  (a) remove all π_i's with $\mathit{head}(\pi_i) = \tau$, say, $\pi_{i_j} = \tau(\bar{t}_j)[\lambda_j]$ for $1 \le j \le k$,
  (b) add a single pattern $\mathrm{mrg}_D(\tau(\bar{t}_1)[\lambda_1, \ldots, \lambda_k])$,

(c) add to E equalities $\bar{t}_1 = \bar{t}_j$ for all $2 \le j \le k$.

(3) Replace all the remaining π_i with $\mathrm{mrg}_D(\pi_i)$.

(4) Return the obtained pattern and the equalities from E.

The resulting pattern is clearly equivalent to the original one (over trees conforming to D), and it admits injective homomorphisms into some trees conforming to D (or is not satisfiable at all).

If we allow sibling order, some labels occurring under $*$ or $+$ might need merging. For instance, under $r \to a^*b^*$; $b \to c?d?$ all a's in $r\big[a \to b \to^+ b[c],\, a \to b \to^+ b[d]\big]$ need to be mapped to the same a-node (the rightmost one), which means they should be merged. Similarly, the subsequent b's should be merged. More generally, for a pattern $\sigma(\bar{u})[\mu_1, \ldots, \mu_k]$ and a production $\sigma \to \hat{\tau}_1\hat{\tau}_2 \ldots \hat{\tau}_n$ in D, we say that a sub-segment κ of μ_i is *critical* if

$$\mathit{head}(\kappa) = \tau_j \;\text{ with }\; \hat{\tau}_j \in \{\tau_j, \tau_j?\} \quad \text{or} \quad \mathit{head}(\kappa) \;=\; \tau_j \to \tau_{j'} \;\text{ with }\; j < j'\,.$$

Since D is nested relational, each critical sub-segment corresponds to a unique node or a pair of nodes in every sequence of children of a σ-node (or cannot be matched at all). Therefore, identical critical sub-segments need to be merged into one, possibly enforcing further merging of the segments containing them.

It is not always possible to guarantee equivalence when merging in the presence of sibling order. In the example above, how does one express that $b[c]$ and $b[d]$ come after $a \to b$ in *some* order? This cannot be done with tree patterns. The solution is to skip $\to^+$ connecting $a \to b$ to $b[c]$ and $b[d]$. The resulting pattern $r\big[a \to b, b[c], b[d]\big]$ is not equivalent, but it satisfies the postulated properties: it is implied by the original pattern, and if it is mapped injectively into a tree conforming to D, the tree satisfies the original pattern. This is because in every injective match this particular $\to^+$ relation is enforced by the DTD. We exploit this idea further in the procedure below.

To obtain $\mathrm{mrg}_D(\sigma[\mu_1, \ldots, \mu_m])$ perform the following steps, collecting equalities in a set E, initially empty.

(1) *Check consistency.* Assume that $\sigma \to \hat{\tau}_1\hat{\tau}_2 \ldots \hat{\tau}_n$ in D. Let ω be obtained from $\hat{\tau}_1\hat{\tau}_2 \ldots \hat{\tau}_n$ as follows:
   (a) set $\hat{\tau}_j$ to τ_j whenever $\hat{\tau}_j = \tau_j?$ and some $\mathit{head}(\mu_i)$ contains τ_j,
   (b) set $\hat{\tau}_j$ to τ_j whenever $\hat{\tau}_j = \tau_j^+$ or $\hat{\tau}_j = \tau_j^*$ and some $\mathit{head}(\mu_i)$ contains a sub-segment $\tau_{j_1} \to \tau_j \to \tau_{j_2}$ with $j_1 < j < j_2$,
   (c) replace $\hat{\tau}_j$ with ε whenever $\hat{\tau}_j = \tau_j?$ or $\hat{\tau}_j = \tau_j^*$ and some $\mathit{head}(\mu_i)$ contains a sub-segment $\tau_{j_1} \to \tau_{j_2}$ with $j_1 < j < j_2$.

   If some $\mathit{head}(\mu_i)$ cannot be satisfied in a word generated by ω, return $\bot$.

(2) *Remove implied $\to^+$.* Remove each occurrence of $\to^+$ between two segments of μ_i, say $\kappa_1 \to^+ \kappa_2$, unless $\mathit{head}(\kappa_1)$ and $\mathit{head}(\kappa_2)$ are both of the form $\tau \to \tau \to \ldots \to \tau$ for the same label τ occurring as τ^+ or τ^* in ω.

(3) *Merge segments according to critical sub-segments.* As long as there are two segments that contain critical sub-segments with the same head, merge them as follows. Let the segments be

$$\pi_{-j} \to \ldots \to \pi_0 \to \pi_1 \to \ldots \to \pi_{k'} \to \pi_{k'+1} \to \ldots \to \pi_k\,,$$
$$\pi'_{-j'} \to \ldots \to \pi'_{-j-1} \to \pi'_{-j} \to \ldots \to \pi'_0 \to \pi'_1 \to \ldots \to \pi'_{k'}\,,$$

where $\pi_i = \tau_i(\bar{u}_i)[\lambda_i]$, $\pi'_i = \tau_i(\bar{u}'_i)[\lambda'_i]$ and either π_0, π'_0 or $\pi_0 \to \pi_1$, $\pi'_0 \to \pi'_1$ are critical sub-segments with the same head (we assume $j < j'$ and $k' < k$, other cases are analogous). If $\tau_i = \tau'_i$ for $i = -j, \ldots, k'$, merge the segments to

$$\underbrace{\pi''_{-j'} \to \ldots \to \pi''_{-j-1}}_{\pi''_i = \mathrm{mrg}_D(\tau'_i(\bar{u}'_i)[\lambda'_i])} \to \underbrace{\pi''_{-j} \to \ldots \to \pi''_0 \to \pi''_1 \to \ldots \to \pi''_{k'}}_{\pi''_i = \mathrm{mrg}_D(\tau_i(\bar{u}_i)[\lambda_i, \lambda'_i])} \to \underbrace{\pi''_{k'+1} \to \ldots \to \pi''_k}_{\pi''_i = \mathrm{mrg}_D(\tau_i(\bar{u}_i)[\lambda_i])}$$

with equalities $\bar{u}_{-j} = \bar{u}'_{-j}, \ldots, \bar{u}_{k'} = \bar{u}'_{k'}$ added to E (if $\pi''_i = \bot$ for some i, return $\bot$). If $\tau_i \neq \tau'_i$ for some $i \in \{-j, \ldots, k'\}$, return $\bot$.

(4) *Return the obtained tree pattern and the equalities from E.*

For generalized patterns, $\mathrm{mrg}_D(\pi, \alpha) = (\pi', \alpha' \wedge \alpha)$, where $(\pi', \alpha') = \mathrm{mrg}_D(\pi)$.

Note that for all tuples of terms $\bar{t}$ we have

$$(\mathrm{mrg}_D\varphi)(\bar{t}) = \mathrm{mrg}_D(\varphi(\bar{t}))\,. \tag{4}$$

Indeed, the merging procedure never looks at the terms, even when it adds equalities in step (3), it just matches corresponding attributes.

The described procedure is clearly polynomial and the size of $\mathrm{mrg}_D\varphi$ is linear in the size of φ. A simple inductive argument gives the lemma below, showing that $\mathrm{mrg}_D\varphi$ satisfies the required properties as long as φ is fully specified and non-bounding (see Appendix). A tree T is *consistent* with a DTD D if it conforms to the DTD obtained from D by setting the root symbol to the label of the root of T. By φ° we denote the pattern obtained from φ by replacing each $\tau(\bar{t})$ with τ, and dropping all equalities and inequalities.

LEMMA 5.13. *Let D be a nested relational DTD and let $\varphi(\bar{x})$ be a fully specified non-bounding tree pattern. Then, $\mathrm{mrg}_D\varphi$ is fully specified and non-bounding, and for every T consistent with D and every $F, \bar{a}$*

(1) $(T, F) \models \varphi(\bar{a})$ implies $(T, F) \models (\mathrm{mrg}_D\varphi)(\bar{a})$;

(2) if $(T, F) \models (\mathrm{mrg}_D\varphi)(\bar{a})$ and the witnessing homomorphism is injective, then $(T, F) \models \varphi(\bar{a})$;

(3) if $\mathrm{mrg}_D\varphi \neq \bot$ and F is admissible, $(\mathrm{mrg}_D\varphi)(\bar{a})$ admits an injective homomorphism into a tree consistent with D.

In particular, $(\mathrm{mrg}_D\varphi)(\bar{a})$ is satisfiable w.r.t. D iff $\varphi(\bar{a})$ is satisfiable w.r.t. D, and $\mathrm{mrg}_D\varphi = \bot$ iff φ° is not satisfiable in a tree consistent with D.

Combining the two operations we build a tree conforming to D and satisfying φ.

COROLLARY 5.14. *Let $\varphi(\bar{x})$ be a fully specified non-bounding pattern, complete with respect to a nested-relational DTD D. Assume that $\mathrm{mrg}_D\varphi \neq \bot$ and let T be the tree obtained from $\mathrm{mrg}_D\varphi$ by evaluating terms according to some admissible $F, \bar{a}$. Then T is consistent with D and $(T, F) \models \varphi(\bar{a})$.*

PROOF. By Lemma 5.13(3), $(\mathrm{mrg}_D\varphi)(\bar{a})$ can be matched injectively in some T' consistent with D. Since the merging operation never removes children, and φ is complete, so is $(\mathrm{mrg}_D\varphi)(\bar{a})$. Hence, its image in T' is a tree consistent with D. Since the matching is injective, the image is isomorphic to T. The second claim follows from $(T, F) \models (\mathrm{mrg}_D\varphi)(\bar{a})$ by Lemma 5.13(1). □

Let us now return to the construction of an unordered universal solution. Suppose that the mapping contains no function symbols, nor variables introduced on the target side, and all its target patterns are complete. Recall the pattern $\delta_{S,\mathcal{M},\emptyset}$ combining all target sides of dependencies for the empty valuation $\emptyset$ of function symbols (the unique partial mapping from Fun to functions over data values, that has empty domain). It is complete as well, and has no free variables. If it is not satisfiable with respect to D_t, there is no solution for S at all. Consider $\rho, \alpha = \mathrm{mrg}_{D_t}(\delta_{S,\mathcal{M},\emptyset})$. By Lemma 5.13, $\rho \neq \bot$ and α is satisfied. Let U be ρ viewed as a tree. By Corollary 5.14 we conclude that U is a solution. To show that U is universal, take some solution T. By Lemma 4.4 and Lemma 5.13(1), $T \models \rho$, so there exists a homomorphism from ρ to T. As U and ρ are isomorphic, this gives a homomorphism from U to T, and proves universality of U. In the general case we need to find the witnessing valuation, which is done by means of a chase-like procedure described in the proof below.

THEOREM 5.15. *Let $\mathcal{M} = (D_s, D_t, \Sigma)$ be a fully specified, non-bounding and $\sim$-monotonic mapping in $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$ with at most d variables in each source side pattern. For each source tree T an unordered universal solution can be computed in time polynomial in $\|\mathcal{M}\| + |T|^d + \|D_t\|^{\|D_t\|}$.*

PROOF. Let Fun be the set of function symbols used in $\mathcal{M}$, and let Const be the set of data values used in T. Also, let Nulls be the set of nulls. Let $\mathrm{GT}(\mathsf{Const}, \mathsf{Fun})$ denote the algebra of ground terms over $\mathsf{Const} \cup \mathsf{Fun}$, where elements of Const are treated as constants. In this algebra, we are using the term interpretation of the function symbols. Under the term interpretation, the value of $f(\text{“4”}, \text{“1”}, \text{“7”})$ is the term $\text{“}f(4,1,7)\text{”}$.

Let α be a conjunction of equalities of terms from $\mathrm{GT}(\mathsf{Const}, \mathsf{Fun})$. Define $\approx_\alpha$ as the least congruence on $\mathrm{GT}(\mathsf{Const}, \mathsf{Fun})$ extending

$$\big\{(t,s) \;\big|\; t = s \text{ or } s = t \text{ is a conjunct of } \alpha\big\}\ . \tag{5}$$

Let $[t]_\alpha$ denote the $\approx_\alpha$-class of t. If $d \approx_\alpha d' \implies d = d'$ for all $d, d' \in \mathsf{Const}$, we say that α is *consistent*. Otherwise it is *inconsistent*.

Assume α is consistent. Let $\nu_\alpha : \mathrm{GT}(\mathsf{Const}, \mathsf{Fun})/_{\approx_\alpha} \to \mathsf{Const} \cup \mathsf{Nulls}$ be a fixed injection preserving constants, i.e., satisfying $\nu_\alpha([c]_\alpha) = c$ for $c \in \mathsf{Const}$. Let $\tilde{\nu}_\alpha$ be the lifting of ν_α to $\mathrm{GT}(\mathsf{Const}, \mathsf{Fun})$, i.e., $\tilde{\nu}_\alpha(t) = \nu_\alpha([t]_\alpha)$. Note that $\tilde{\nu}_\alpha(t) = c \in \mathsf{Const}$ iff $t \approx_\alpha c$.

We define a valuation such that each term t evaluates to $\tilde{\nu}_\alpha(t)$. Let $\mathrm{Rg}\,\nu$ stand for the range of function ν. Let F_α assign to every k-ary function symbol $f \in \mathsf{Fun}$, the function $F_\alpha(f) : (\mathrm{Rg}\,\tilde{\nu}_\alpha)^k \to \mathrm{Rg}\,\tilde{\nu}_\alpha$ given as $F_\alpha(f)(\tilde{\nu}_\alpha(t_1), \tilde{\nu}_\alpha(t_2), \ldots, \tilde{\nu}_\alpha(t_k)) = \tilde{\nu}_\alpha(f(t_1, t_2, \ldots, t_k))$. To see that $F_\alpha(f)$ is well defined, take $s_1, s_2, \ldots s_k$ such that $\tilde{\nu}_\alpha(s_i) = \tilde{\nu}_\alpha(t_i)$ for all i. By the definition of $\tilde{\nu}_\alpha$, it means that $\nu_\alpha([s_i]_\alpha) = \nu_\alpha([t_i]_\alpha)$. As ν_α is injective, we have $[s_i]_\alpha = [t_i]_\alpha$. Since $\approx_\alpha$ is a congruence, $f(t_1, t_2, \ldots, t_k) \approx_\alpha f(s_1, s_2, \ldots, s_k)$, and consequently $\tilde{\nu}_\alpha(f(t_1, t_2, \ldots, t_k)) = \tilde{\nu}_\alpha(f(s_1, s_2, \ldots, s_k))$.

Since under F_α each term t evaluates to $\tilde{\nu}_\alpha(t)$, we have $t = s$ under F_α iff $t \approx_\alpha s$. The latter can be checked in PTIME for given α, s, t.

For a valuation F of function symbols and a pattern π using no variables, we write $F(\pi)$ for the pattern obtained by evaluating the terms in π according to F.

In the following we say that a dependency $\varphi(\bar{x},\bar{y}) \longrightarrow \psi(\bar{x})$ is *pending* for a pattern π,α and tuples $\bar{a},\bar{b} \in \mathsf{Const}$ if $(T,F_\alpha) \models \varphi(\bar{a},\bar{b})$ but $F_\alpha(\pi)$ viewed as a tree does not satisfy $\psi(\bar{a})$ under the valuation F_α. The algorithm iteratively refines a pattern π,α (describing the solution), processing pending dependencies and merging into π,α the missing target constraints. In the last step of the algorithm, the pattern is converted into a solution.

Recall that we can eliminate all inequalities on the target side by means of equality on the source side and an inconsistent pattern $\bot$ on the target side. By Lemma 5.12, we can assume that the target side patterns are complete. Initially, let $\pi = r$ (the trivial pattern), and let $\alpha = \mathit{true}$. Repeat the following until there is no change in π,α:

—Find $\bar{a},\bar{b}$ and a dependency $\varphi(\bar{x},\bar{y}) \longrightarrow \psi(\bar{x})$ pending for π,α and $\bar{a},\bar{b}$.

—Let $\pi',\alpha' = \mathrm{mrg}_{D_t}((\pi,\alpha) \wedge \psi(\bar{a}))$.

—If $\pi' = \bot$ or α' is inconsistent, return "no solution".

—Replace π,α with π',α'.

When the loop terminates, let U be the (unordered) tree obtained from π by ignoring $\rightarrow$, $\rightarrow^+$ and evaluating the terms according to F_α (admissible, as α is consistent).

Clearly, the pair (T,U) satisfies all dependencies with the witnessing valuation F_α. By Corollary 5.14, U is consistent with D_t. Since U has label r in the root, it conforms to D_t. Hence, U is a solution. It remains to prove that it is universal, and that if the algorithm returns "no solution", there is indeed no solution at all.

We shall prove by induction that after every iteration, for each solution T' with a witnessing valuation F, $(T',F) \models \pi,\alpha$. For the initial values $\pi = r$, $\alpha = \mathit{true}$, the claim obviously holds. Assume the claim for π,α and let $\varphi(\bar{x},\bar{y}) \longrightarrow \psi(\bar{x})$ be a dependency pending for π,α and $\bar{a},\bar{b}$; in particular, $(T,F_\alpha) \models \varphi(\bar{a},\bar{b})$. Suppose that T' is a solution with a witnessing valuation F. By the induction hypothesis, α holds under F. Hence, the equality of terms under F is a congruence extending (5). Since $\approx_\alpha$ is the least such congruence, by the definition of F_α, whenever $s = t$ under F_α, it also holds that $s = t$ under F. Since $\mathcal{M}$ is $\sim$-monotonic, φ does not contain inequalities between nontrivial terms (only between variables). Consequently, $(T,F_\alpha) \models \varphi(\bar{a},\bar{b})$ implies $(T,F) \models \varphi(\bar{a},\bar{b})$, and so $(T',F) \models \psi(\bar{a})$. Since $(T',F) \models \psi(\bar{a}) \wedge (\pi,\alpha)$ and $\psi(\bar{a}) \wedge (\pi,\alpha)$ is fully specified and non-bounding, we conclude by Lemma 5.13 (1) that $(T',F) \models (\pi',\alpha')$ where $(\pi',\alpha') = \mathrm{mrg}_{D_t}((\pi,\alpha) \wedge \psi(\bar{a}))$, and the claim follows.

The algorithm returns "no solution" only when $\pi' = \bot$ or α' is inconsistent. Then, by the claim above, no solution exists. If the loop terminates with a positive result, the claim holds for the obtained π,α. In particular, if T' is a solution with a witnessing valuation F, there exists a homomorphism $h : F(\pi) \to T'$. If we manage to define a homomorphism $g : F_\alpha(\pi) \to F(\pi)$, we will be done because U is $F_\alpha(\pi)$ viewed as a tree, and so $g \circ h$ induces a homomorphism from U to T'. A natural candidate for g is obtained by taking identity on nodes of π; we only need to check if g is well defined on nulls and preserves constants. Since α holds under F, like before we conclude that $s = t$ under F_α implies $s = t$ under F. In particular, if $t = c \in \mathsf{Const}$ under F_α, then $t = c$ under F.

As for the complexity, note that the main loop is executed at most $\|\mathcal{M}\| \cdot |T|^d$

	$\mathbf{CTQ}(\downarrow,=)$	$\mathbf{CTQ}(\Downarrow,=)$	$\mathbf{CTQ}(\downarrow,\Rightarrow,=)$
fully specified $\mathrm{SM}^{\mathrm{nr}}(\Downarrow,=)$	PTIME	PTIME	coNP
restricted $\mathrm{SM}^{\mathrm{nr}}(\Downarrow,\Rightarrow,\sim,\mathsf{Fun})$	PTIME	PTIME	coNP
$\mathrm{SM}(\Downarrow,=)$	coNP	coNP	coNP

'restricted' means 'fully specified, non-bounding, $\sim$-monotonic'.
Results on $\mathbf{CTQ}(\Downarrow,=)$ for $\mathrm{SM}(\Downarrow,=)$ and fully specified $\mathrm{SM}^{\mathrm{nr}}(\Downarrow,=)$ are from [Arenas and Libkin 2008].

Fig. 2. The complexity of $\textsc{certain}_{\mathcal{M}}(Q)$

times. The size of the candidate solution grows by at most $\|D_t\|^{\|D_t\|} \cdot \|\mathcal{M}\|$ (the size of the target side pattern after completion) in each execution of the loop. Hence, the size of the computed solution can be bounded by $|T|^d \cdot \|D_t\|^{\|D_t\|} \cdot \|\mathcal{M}\|^2$.

The cost of a single execution of the main loop is polynomial in $\|\mathcal{M}\| \cdot |T|^d + \|D_t\|^{\|D_t\|} \cdot \|\mathcal{M}\|$. Altogether, the data complexity of the algorithm is polynomial, and the combined complexity is polynomial in $\|\mathcal{M}\| + |T|^d + \|D_t\|^{\|D_t\|}$. □

Figure 2 presents the summary of the main results of this section. When we write coNP, we mean that the problem could be coNP-complete for some choice of a mapping and a query from the relevant classes (and is in coNP for all such choices). The last line says that beyond the class of fully specified mappings, there is no hope to get tractability. Within the class of fully specified mappings, it is clear that we have certain freedom to increase the expressiveness of the mappings, but not the queries.

The conclusion, therefore, is that one must restrict the usage of sibling order and inequality to the mappings. What sense does it make to use sibling order in the mapping if we cannot ask queries about it? The example in Section 5.1 shows how one can meaningfully use sibling order on the source side, and store the result on the target side as labeled tuples. In fact, the semantics of the mappings makes it impossible to copy from the source to the target ordered sequences of children of arbitrary length. Hence, whatever we encode on the target side with sibling order, we can equally well encode using labeled tuples, provided we have a little influence on the target DTD. Thus, forbidding horizontal navigation in the target database and queries we do not lose much in terms of expressiveness.

6. CONSISTENCY OF SCHEMA MAPPINGS

As we mentioned in the introduction, XML schema mappings may be inconsistent: there are mappings $\mathcal{M}$ such that no tree has a solution. Formally, a mapping is *consistent* if $[\![\mathcal{M}]\!] \neq \emptyset$; that is, if $\textsc{Sol}_{\mathcal{M}}(T) \neq \emptyset$ for some $T \models D_s$. The decision problem we consider is the following:

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

The complexity of recognizing consistent mappings in $\mathrm{SM}(\Downarrow)$ and $\mathrm{SM}^{\mathrm{nr}}(\Downarrow)$ was established in [Arenas and Libkin 2008]. (Recall that nr indicates restriction to nested-relational DTDs, i.e., non-recursive DTDs with 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^+.)

FACT 6.1. (see [Arenas and Libkin 2008]) CONS($\Downarrow$) *is* EXPTIME*-complete. If we further restrict to nested-relational DTDs in schema mappings, then* CONS($\Downarrow$) *is in polynomial (cubic) time.*

Here we analyze how the complexity of the problem depends on other features used in the mappings: horizontal axes and data comparisons.

More refined consistency questions are whether a given source instance has a solution, and whether *each* source instance has a solution [Amano et al. 2009]. As shown recently in [Bojanczyk et al. 2013], these questions are very different from the original consistency problem: they are both decidable even for the most general mappings considered here, albeit with high combined complexity.

6.1 Adding horizontal axes

In this section we check how the complexity of consistency changes after horizontal axes are introduced. Our first result shows that in the absence of data comparisons, the complexity stays the same.

THEOREM 6.2. *The problem* CONS($\Downarrow, \Rightarrow, \mathsf{Fun}$) *is in* EXPTIME *(and thus* EXPTIME*-complete).*

The key observation is that without data comparisons, CONS($\Downarrow, \Rightarrow$) amounts to solving the problem for mappings that do not mention data values at all, which can be done by tree automata techniques. Recall that φ° denotes the tree pattern obtained from φ by replacing $\ell(\bar{t})$ with ℓ (ℓ can be a label or _). Let $\Sigma^\circ = \{\varphi^\circ \longrightarrow \psi^\circ \mid (\varphi \longrightarrow \psi) \in \Sigma\}$. It is easy to check that (D_s, D_t, Σ) is consistent iff (D_s, D_t, Σ°) is consistent: obviously $[\![(D_s, D_t, \Sigma)]\!] \subseteq [\![(D_s, D_t, \Sigma^\circ)]\!]$ and if $(S, T) \in [\![(D_s, D_t, \Sigma^\circ)]\!]$, then trees S' and T', obtained by replacing all data values with a single data value, satisfy $(S', T') \in [\![(D_s, D_t, \Sigma)]\!]$. Thus we can assume that all tree patterns are variable-free. When data values are ignored, XML trees become ordinary trees, and we can use tree automata to reason about them.

Let us recall the notion of "first child, next sibling" tree automaton. Such an automaton can be presented as a tuple $\mathcal{A} = (\Gamma, Q, q_0, Q_F, \delta)$, where Γ is the labeling alphabet (the set of element types in our case), Q is the state space with the initial state $q_0 \in Q$ and final states $Q_F \subseteq Q$, and $\delta \subseteq Q \times \Gamma \times Q \times Q$ is the transition relation. A run of $\mathcal{A}$ over a tree T is a labeling ρ of the nodes of T with the states of $\mathcal{A}$ that satisfies the following conditions for each node v:

—$(\rho(v), lab_T(v), q_0, q_0) \in \delta$ if v has no children and no next sibling;

—$(\rho(v), lab_T(v), q_0, \rho(w)) \in \delta$ if v has no children but has the next sibling w;

—$(\rho(v), lab_T(v), \rho(v \cdot 0), q_0) \in \delta$ if v has children $v{\cdot}0, \ldots, v{\cdot}k$, but no next sibling;

—$(\rho(v), lab_T(v), \rho(v \cdot 0), \rho(w)) \in \delta$ if v has children $v{\cdot}0, \ldots, v{\cdot}k$ and next sibling w.

The languages of trees *recognized* by $\mathcal{A}$, denoted by $L(\mathcal{A})$, consists all trees admitting an *accepting* run of $\mathcal{A}$, i.e. a run that assigns one of the final states to the root.

It is well known that there is a polynomial translation from DTDs to automata.

LEMMA 6.3. *For each DTD D there exists a polynomial tree automaton recognizing* $\{T \mid T \models D\}$.

The next step is to translate patterns to automata. Since we need to express negations of patterns as well, it is convenient to translate directly into deterministic automata. Recall that a tree automaton is *(bottom-up) deterministic*, if for all $q_1, q_2 \in Q$ and $a \in \Sigma$ there is exactly one state q such that $(q, a, q_1, q_2) \in \delta$. For a deterministic automaton $\mathcal{A}$ one obtains an automaton recognizing the complement of $L(\mathcal{A})$ by replacing the set of final states Q_F with its complement $Q - Q_F$.

LEMMA 6.4. *For every variable-free pattern π, there exists a deterministic tree automaton recognizing $\{T \mid T \models \pi\}$ that can be computed by a* PSPACE *transducer (a Turing machine with polynomial working space and a write-only output tape).*

PROOF. Let $\operatorname{Seq}\pi$ denote the set of *sequence sub-patterns* of π, that is patterns of the form $\pi_1 \rightsquigarrow_1 \pi_2 \rightsquigarrow_2 \cdots \rightsquigarrow_k \pi_{k+1}$ with $\rightsquigarrow_1, \rightsquigarrow_2, \ldots, \rightsquigarrow_k \in \{\to, \to^+\}$ and $k \geq 0$ that are sub-terms of the term constituting π.

To recognize $\{T \mid T \models \pi\}$ the automaton simply computes for each node v the set of sequence sub-patterns that are satisfied in v and those that are satisfied to the right from v. The state space is $Q = 2^{\operatorname{Seq}\pi} \times 2^{\operatorname{Seq}\pi}$, with the initial state $(\emptyset, \emptyset)$ and final states $Q_F = \{(\Phi, \Phi') \mid \pi \in \Phi\}$. The transition relation simply updates the two sets based on their values in the first child and the next sibling. Formally, for a label a we set $\delta(a, (\Phi, \Phi'), (\Psi, \Psi')) = (\Delta, \Psi \cup \Psi')$, where Δ consists of all sequence sub-patterns of π of the form $\pi_1 \to \mu$ or $\pi_1 \to^+ \mu'$ or π_1 such that

—$\mu \in \Psi$, $\mu' \in \Psi \cup \Psi'$;
—if $\pi_1 = \ell[\mu_1, \ldots, \mu_n]$ then $\ell \in \{\_, a\}$, $\mu_1, \ldots, \mu_n \in \Phi \cup \Phi'$;
—if $\pi_1 = //\ell[\mu_1, \ldots, \mu_n]$ then either $\pi_1 \in \Phi \cup \Phi'$ or $\ell \in \{\_, a\}$, $\mu_1, \ldots, \mu_n \in \Phi \cup \Phi'$.

Observe that each state is represented as an object of size polynomial in the size of π and checking $(\Phi, \Phi') \in Q_F$ and $((\Delta, \Delta'), a, (\Phi, \Phi'), (\Psi, \Psi')) \in \delta$ can be done in PTIME. It follows easily that the whole automaton can be computed by a PSPACE transducer. □

PROOF OF THEOREM 6.2. As we have observed, it is enough to consider mappings that do not mention data values. By Lemma 6.4 we can construct in exponential time a deterministic tree automaton $\mathcal{A}(\varphi)$ that accepts the set of trees satisfying φ. Since the automaton is deterministic, the automaton $\overline{\mathcal{A}(\varphi)}$ accepting the complement of $\mathcal{A}(\varphi)$ is also computable in exponential time.

For a mapping (D_s, D_t, Σ) to be consistent, there must exist a pair (S, T) such that for all $\varphi \longrightarrow \psi \in \Sigma$ it holds that $S \models \varphi$ implies $T \models \psi$. Suppose $\Sigma = \{\varphi_i \longrightarrow \psi_i \mid i = 1, 2, \ldots, n\}$. Then the existence of such a pair is equivalent to the existence of a subset $I \subseteq \{1, 2, \ldots, n\}$ satisfying

—there exists $S \models D_s$ such that $S \not\models \varphi_j$ for all $j \notin I$,
—there exists $T \models D_t$ such that $T \models \psi_i$ for all $i \in I$.

This amounts to non-emptiness of the automata

$$\mathcal{A}_{D_s} \times \prod_{j \notin I} \overline{\mathcal{A}(\varphi_j)} \qquad \text{and} \qquad \mathcal{A}_{D_t} \times \prod_{j \in I} \mathcal{A}(\psi_j)\,,$$

where $\mathcal{A}_{D_s}$ and $\mathcal{A}_{D_t}$ are automata recognizing trees conforming to D_s and D_t respectively (Lemma 6.3). It is known that testing non-emptiness of $\mathcal{A}_1 \times \cdots \times \mathcal{A}_k$

can be done in time $\mathcal{O}(|\mathcal{A}_1| \times \cdots \times |\mathcal{A}_k|)$. Since the construction of each $\mathcal{A}(\varphi)$ takes exponential time, the overall complexity is EXPTIME. □

Unlike the case of mappings SM($\Downarrow$) with downward navigation only, once we add even the simplest form of horizontal navigation, consistency is intractable even over nested-relational DTDs.

PROPOSITION 6.5. *Over nested-relational DTDs,* CONS($\Downarrow, \Rightarrow$) *and* CONS($\Downarrow, \rightarrow$) *are* PSPACE*-complete.*

The proof of the upper bound is similar to that of Theorem 6.2, but this time, instead of computing the automata explicitly, we generate them on-the-fly during the non-emptiness check, using the full power of Lemma 6.4. We also rely on the fact that testing non-emptiness of an automaton $\mathcal{A}$ on trees of depth d can be done in space polynomial in $d \cdot \log \|\mathcal{A}\|$. Details can be found in the Appendix.

We note that the upper bounds of this section could also be obtained via a reduction to XPath satisfiability [Benedikt et al. 2008].

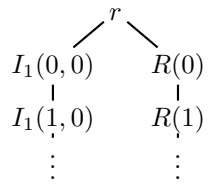
6.2 Adding data comparisons

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 [Bojanczyk et al. 2009; Björklund et al. 2008; David 2008; Fan and Libkin 2002; Segoufin 2006]. 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 [Bojanczyk et al. 2009] or Boolean combinations of patterns with data value comparisons [David 2008]. The reason is the very "positive" nature of dependencies in schema mappings: the use of negation is limited to the implication in dependencies, while known undecidable formalisms can use negation freely.

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

THEOREM 6.6. *Problems* CONS($\downarrow, \downarrow^+, =$), CONS($\downarrow, \downarrow^+, \neq$), CONS($\downarrow, \rightarrow, =$), *and* CONS($\downarrow, \rightarrow, \neq$) *are all undecidable; the latter two already under the restriction to nested-relational DTDs. In particular,* CONS($\Downarrow, \Rightarrow, \sim$) *is undecidable.*

PROOF. (1) Consider first CONS($\downarrow, \downarrow^+, =$). We give a reduction from the halting problem of 2-register machine, known to be undecidable. Given a 2-register machine (defined below), we construct a schema mapping that is consistent iff the machine halts. Trees encoding runs of a 2-register machine will be of the form:



Intuitively, the left branch is meant to represent sequence of states with data values representing registers while the right one is a sequence to represent natural numbers. We do not have any equality test against a constant (say, a natural number). So,

what we really do is simulate values by the depth from the root. More concretely 0 and 1 above might as well be 7 and 3. Whatever they are, we simply take the value at the 0th level as 0 and the 1st level as 1, and so on. The above tree can be easily described by a DTD. To make sure it is a proper run of the given machine, we use dependencies to check that the registers change their values according to legal transitions.

Let us now describe the reduction in detail. A 2-register machine M consists of a set of states $Q = \{1, 2, \ldots, f\}$, a list of instructions $\mathcal{I} = (I_i)_{i \in Q \setminus \{f\}}$ (one instruction for each state apart from the last state f), and two registers r_1 and r_2, each containing a natural number. An instantaneous description (ID) of M is a triple (i, m, n) where $i \in Q$ and $m, n \in \mathbb{N}$ are natural numbers stored in r_1 and r_2, respectively.

An instruction of 2-register machine is either *increment* or *decrement*, and defines the transition relation $\to_M$ between IDs.

Increment. $I_i = (\mathsf{r}, j)$, where $i, j \in Q$ and r is one of r_1 and r_2. This means that M in state i increments r and goes to state j:

$$(i, m, n) \to_M \begin{cases} (j, m+1, n) & \text{if } \mathsf{r} = \mathsf{r}_1\,, \\ (j, m, n+1) & \text{if } \mathsf{r} = \mathsf{r}_2\,. \end{cases}$$

Decrement. $I_i = (\mathsf{r}, j, k)$, where $i, j, k \in Q$ and r is one of the two registers. This means that M in state i can test whether r is 0, and go to state j if it is, or decrement r and go to k if it is not. In symbols,

$$(i, m, n) \to_M \begin{cases} (j, 0, n) & \text{if } \mathsf{r} = \mathsf{r}_1 \text{ and } m = 0\,, \\ (k, m-1, n) & \text{if } \mathsf{r} = \mathsf{r}_1 \text{ and } m \neq 0\,, \\ (j, m, 0) & \text{if } \mathsf{r} = \mathsf{r}_2 \text{ and } n = 0\,, \\ (k, m, n-1) & \text{if } \mathsf{r} = \mathsf{r}_2 \text{ and } n \neq 0\,. \end{cases}$$

The initial ID is $(1, 0, 0)$ and the final ID is $(f, 0, 0)$. The halting problem for 2-register machine is to decide, given a 2-register machine M, whether $(1, 0, 0) \to_M^* (f, 0, 0)$.

Let us now describe how to construct a mapping that is consistent iff the given machine halts. The source DTD D_s over $\{r, I_1, I_2, \ldots, I_f, R, \sharp\}$ is given by

$$\begin{aligned} r &\to I_1 R \\ I_i &\to I_j && \text{for all } i \text{ such that } I_i = (\mathsf{r}, j) \\ I_i &\to I_j | I_k && \text{for all } i \text{ such that } I_i = (\mathsf{r}, j, k) \\ R &\to R | \sharp \\ I_f, \sharp &\to \varepsilon \end{aligned}$$

where each I_i has two attributes corresponding to the values of the registers, and R has one attribute. The target DTD D_t is just $r \to \varepsilon$.

As mentioned above, the sequence of R's is meant to be that of natural numbers, but what represents a number is the depth in the tree instead of a value itself. In other words, the data values are used as indices, so they must be unique. The

following dependency disallows two values to appear more than once.

$$//R(x)//R(x) \longrightarrow \bot$$

Let us now deal with the left branch, which is meant to encode the run itself. We have assumed that the initial ID is $(1, 0, 0)$; it is enforced by the constraint below.

$$r[I_1(x, y), R(z)] \longrightarrow x = y, z = y$$

Now, let us check that we proceed correctly. For each i such that $I_i = (\mathsf{r}_1, j)$, we need to enforce that there is a number in the R-branch to set the value of r_1 to, and that the next configuration is indeed obtained by increasing r_1.

$$r[//I_i(x, y), //R(x)/\sharp] \longrightarrow \bot$$
$$r[//I_i(x, y)/I_j(x', y'), //R(x)/R(x'')] \longrightarrow x' = x'', y' = y$$

For each i such that $I_i = (\mathsf{r}_1, j, k)$, we need to say: if the next state is k, then r_1 stores 0, and both registers stay the same; if the next state is j, then r_1 does not store 0, the register r_1 gets decreased, and r_2 stays the same.

$$r[//I_i(x, y)/I_k(x', y'), R(x'')] \longrightarrow x = x'', x' = x, y' = y$$
$$r[//I_i(x, y)/I_j, R(x)] \longrightarrow \bot$$
$$r[//I_i(x, y)/I_j(x', y'), //R(x'')/R(x)] \longrightarrow x' = x'', y' = y$$

For each i such that $I_i = (\mathsf{r}_2, j)$ or $I_i = (\mathsf{r}_2, j, k)$ we add analogous dependencies.

Finally, we have to make sure that we end properly. In each source tree, the left branch must end with I_f, so we do not need to check that. It is enough to say that both registers are set to 0.

$$r[//I_f(x, y), R(z)] \longrightarrow x = z, y = z$$

The obtained mapping (D_s, D_t, Σ) is consistent iff there is a halting run of the given 2-register machine. Thus we have proved that $\textsc{Cons}(\downarrow, \downarrow^+, =)$ is undecidable.

(2) For $\textsc{Cons}(\downarrow, \rightarrow, =)$, essentially, we rotate the encoding above by 90 degrees. To make the source DTD nested-relational, in the encoding of the run we use a single label I with $i \in Q$ stored as an attribute. It requires a bit of hassle to make sure the attributes store values from a fixed set representing Q and to check which state a given value represents.

Let the source DTD be

$$\begin{aligned} r &\to \sharp\, R^+ \natural\, I^+ \flat\, I_1\, I_2\, \ldots\, I_f \\ R, I_j &: @attr \qquad\qquad 1 \le j \le f \\ I &: @state\ @reg_1\ @reg_2 \end{aligned}$$

and let the target DTD be $r \to \varepsilon$.

The dependency $//R(x)//R(x) \longrightarrow \bot$ ensuring that the values representing register values do not repeat can be rewritten as $r[R(x) \to^+ R(x)] \longrightarrow \bot$, but in order to get undecidability of $\textsc{Cons}(\downarrow, \rightarrow, =)$, we need to enforce this property without $\to^+$. Due to non-injective semantics of tree patterns, we cannot distinguish between something happening once and twice, which means we need a trick. The idea is to

disallow switching from one data value to two different ones (the first dependency), or switching to a data value and the ending marker (the second dependency).

$$r\,[R(x) \to R(y), R(x) \to R(y')] \longrightarrow y = y'$$
$$r\,[R(x) \to R, R(x) \to \natural] \longrightarrow \bot$$

These two conditions imply that if a value repeats in two nodes, the sequence of children between these nodes repeats forever. The latter obviously cannot happen in a finite tree, which means that data values stored in R-nodes never repeat.

We want to think of the data value stored in I_i as the name of the state i of our 2-register machine. For that purpose it is enough to say that no two are equal:

$$r\,[I_i(x), I_j(x)] \longrightarrow \bot \qquad \text{for all } i \neq j$$

To enforce correctness of the encoded computation, we exploit the fact that our 2-register machine is deterministic: we express that each configuration is followed by the configuration determined by the instructions of the machine and the content of the counters. We have assumed that the initial ID is $(1, 0, 0)$:

$$r[\natural \to I(q, x, y),\ \sharp \to R(z),\ I_1(q')] \longrightarrow q = q', x = z, y = z\,.$$

Now, let us check that we proceed correctly. For each i such that $I_i = (\mathsf{r}_1, j)$, we need to enforce that there is a number in the R-branch to set the value of r_1 to, and that the next configuration is indeed obtained by increasing r_1:

$$r[I(q, x, y),\ I_i(q),\ R(x) \to \natural] \longrightarrow \bot\,,$$
$$r[I(q, x, y) \to I(q', x', y'),\ I_i(q),\ I_j(q''),\ R(x) \to R(x'')] \longrightarrow x' = x'', y' = y, q' = q''.$$

For each i such that $I_i = (\mathsf{r}_1, j, k)$, we need to say: if r_1 stores 0, then the next state is k, and both registers stay the same; if r_1 does not store 0, then the next state is j, r_1 gets decreased, and r_2 stays the same.

$$r[I(q, x, y) \to I(q', x', y'),\ I_i(q),\ I_k(q''),\ \sharp \to R(x)] \longrightarrow x' = x,\ y' = y,\ q' = q''$$
$$r[I(q, x, y) \to I(q', x', y'),\ I_i(q),\ I_j(q''),\ R(x'') \to R(x)] \longrightarrow x' = x'',\ y' = y,\ q' = q''$$

For each i such that $I_i = (\mathsf{r}_2, j)$ or $I_i = (\mathsf{r}_2, j, k)$ we add analogous dependencies.

Finally, we make sure that we end properly, i.e., that the last ID is $(f, 0, 0)$.

$$r[I(q, x, y) \to \flat, \sharp \to R(z), I_f(q')] \longrightarrow x = z,\ y = z,\ q = q'$$

The obtained mapping (D_s, D_t, Σ) is consistent iff there is a halting run of the given 2-register machine. Thus we have proved that $\textsc{Cons}(\downarrow, \to, =)$ is undecidable.

(3) It is not difficult to rewrite all the dependencies in the vertical and horizontal version with $\neq$ instead of $=$. To eliminate $=$ from the target side, simply replace all dependencies of the form $\varphi \longrightarrow x = y, z = w, \ldots,$ with $\varphi, x \neq y \longrightarrow \bot$, $\varphi, z \neq w \longrightarrow \bot$, etc. On the source side, we use equality in the form of repeated variables. In dependencies like $r[I(q, x, y),\ I_i(q),\ R(x) \to \natural] \longrightarrow \bot$ we use two repetitions, and we cannot replace them with inequalities on the target side without disjunction. In such dependencies, the repeated use of q expresses the fact that q represents state i. Using inequality we can express this in a different way: we can say that q does not represent any other state. This is done by replacing the term $I_i(q)$ with $I_1(q_1), \ldots, I_{i-1}(q_{i-1}), I_{i+1}(q_{i+1}), \ldots, I_f(f_f)$ and adding inequalities

$q \neq q_1, \ldots, q \neq q_{i-1}, q \neq q_{i+1}, \ldots, q \neq g_f$ to the source side. Separately, we ensure that the first attribute of each I-node stores a data value representing some state:

$$r[I(q,x,y), I_1(q_1), \ldots, I_f(f_f)], q \neq q_1, \ldots, q \neq g_f \longrightarrow \bot\,.$$

This gives an equivalent set of dependencies with at most one repetition of variable. It remains to replace the two occurrences of the repeated variable x with x_1 and x_2, add $x_1 \neq x_2$ on the target side and remove $\bot$. □

This result raises the question whether there is any useful decidable restriction of SM($\Downarrow, \Rightarrow, \sim$). We know from papers such as [Bojanczyk et al. 2009; Fan and Libkin 2002] 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 6.7. *Under the restriction to nested-relational DTDs, both* CONS($\Downarrow, \sim, \mathsf{Fun}$) *and* CONS($\downarrow, \_, =$) *are* NEXPTIME*-complete.*

PROOF. (1) Let us start with the upper bound for CONS($\Downarrow, \sim, \mathsf{Fun}$). Let $\mathcal{M} = \langle D_s, D_t, \Sigma\rangle$ be a schema mapping with D_s and D_t nested-relational. First recall that tree pattern formulae (even with $=$ and $\neq$) are monotone in the following sense: for any tree pattern formula φ, if T' is obtained from T by erasing some subtrees and $T' \models \varphi$, then $T \models \varphi$. Roughly speaking, this allows us to consider the smallest tree possible with regard to a DTD. More precisely, for a nested-relational DTD D, D° is obtained by turning each ℓ^* and $\ell?$ into ε and each ℓ^+ into ℓ [Arenas and Libkin 2008]. The mapping $\mathcal{M}$ is consistent iff (D_s°, D_t, Σ) is consistent. Since disjunctions are not allowed in productions, we have only one tree conforming to D_s° (up to data values stored in the attributes). The size of this tree is at most $N_s = \|D_s\|^{\|D_s\|}$. Let us guess a tree S like this, with attributes taken from $B_s = \{1, 2, \ldots, N_s\}$. By Lemma 4.4, a tree T, conforming to D_t, is a solution for S iff it satisfies

$$\delta_{S,\mathcal{M},F} = \bigwedge \big\{ \psi(\bar{a}) \;\big|\; \varphi(\bar{x},\bar{y}) \longrightarrow \psi(\bar{x}) \;\in \Sigma \text{ and } (S,F) \models \varphi(\bar{a},\bar{b}) \big\}\,.$$

for some valuation F of function symbols. Like in the proof of Theorem 4.5, for all sub-terms in the definition of $\delta_{S,\mathcal{M},F}$ we can guess consistent values from $B_s \cup \{\bot_1, \bot_2, \ldots, \bot_N\}$, where $N = \|\mathcal{M}\| \cdot |S|^{\|\mathcal{M}\|} = \|\mathcal{M}\| \cdot \|D_s\|^{\|D_s\| \cdot \|\mathcal{M}\|}$, and compute $\delta_{S,\mathcal{M},F}$ (in exponential time). Assume that all the equalities and inequalities in $\delta_{S,\mathcal{M},F}$ hold (if not, reject), and let π be the pure pattern underlying $\delta_{S,\mathcal{M},F}$. Note that π contains no variables. It remains to see if there exists a tree T such that $T \models D_t$, $T \models \pi$. By Lemma 4.1, this can be done nondeterministically in time polynomial in the size of π and D_t. Hence, the whole procedure is in NEXPTIME.

(2) The lower bound for CONS($\downarrow, \_, =$) is proved by a reduction from the following NEXPTIME-complete problem: given a non-deterministic Turing machine M and $n \in \mathbb{N}$, does M accept the empty word in at most 2^n steps? The idea of the reduction is to encode in the target tree an accepting run of M (a sequence of 2^n configurations of length 2^n). The machine M is stored in the source tree, except the (specially preprocessed) transition relation, which is encoded in the target tree. The source tree is also used to address the configurations and their cells.

Let M have the tape alphabet A with the blank symbol $\flat \in A$ and the states $q_0, q_1, \ldots, q_f$. W.l.o.g. we assume that q_f is the only final accepting state. The

extended transition relation of M, denoted $\hat{\delta}$, describes possible transitions in a window of three consecutive tape cells. Formally, $\hat{\delta} \subseteq (\{q_0, q_1, \ldots, q_f, \natural\} \times \hat{A})^6$, where $\hat{A}$ is the set of *decorated tape symbols*, defined as $\hat{A} = \{s, s^{\triangleright}, s^{\triangleleft} \mid s \in A\}$. The symbol $\natural$ means "the head is elsewhere", $\triangleright$ marks the beginning of the tape, $\triangleleft$ marks the end of the tape (at position 2^n), and $(p_1, \sigma_1, p_2, \sigma_2, \ldots, p_6, \sigma_6) \in \hat{\delta}$ iff at most one of p_1, p_2, p_3 is not equal to $\natural$, and $p_4\sigma_4 p_5\sigma_5 p_6\sigma_6$ is obtained from $p_1\sigma_1 p_2\sigma_2 p_3\sigma_3$ by performing a transition of M. Note that $p_1 = p_2 = p_3 = \natural$ and $p_4 \neq \natural$ is possible as long as σ_4 is not marked with $\triangleright$. Note that $\hat{\delta}$ can be computed in PTIME. The constant d used in the target DTD is equal to $|\hat{\delta}|$.

The source DTD is given as

$$\begin{aligned} r &\to \mathit{zero}\ \mathit{one}\ q_0\, q_1 \cdots q_f\, \natural\, \tau_1\, \tau_2 \cdots \tau_m \\ \mathit{zero}, \mathit{one} &\to \mathit{bit} \end{aligned}$$

where $\hat{A} = \{\tau_1, \tau_2, \ldots, \tau_m\}$ and each label except r, *zero*, *one* has a single attribute. The target DTD is given as

$$\begin{aligned} r &\to a_1\, b_1\, \mathit{tr}_1\, \mathit{tr}_2 \cdots \mathit{tr}_d \\ \mathit{tr}_i &\to \mathit{tr} && \mathit{tr}\colon @\mathit{st}_1\ @\mathit{sym}_1\ @\mathit{st}_2\ @\mathit{sym}_2 \cdots @\mathit{st}_6\ @\mathit{sym}_6 \\ a_j, b_j &\to a_{j+1}\, b_{j+1} \\ a_{2n}, b_{2n} &\to \mathit{cell} && \mathit{cell}\colon @a_1\ @a_2 \cdots @a_{2n}\ @\mathit{st}\ @\mathit{sym} \end{aligned}$$

for $i = 1, 2, \ldots, d$, $j = 1, 2, \ldots, 2n-1$, and $d = |\hat{\delta}|$. The *cell* nodes store in binary a configuration number and a cell number, as well as a state and a decorated tape symbol. The *tr* nodes store $\hat{\delta}$.

First we enforce that each source tree admitting a solution stores different data value in each node. This is done by means of a set of dependencies of the form

$$r[\pi(X), \pi'(X)] \longrightarrow \bot$$

where $\pi(X)$ and $\pi'(X)$ range over pairs of different patterns from among $\mathit{zero}/\mathit{bit}(X)$, $\mathit{one}/\mathit{bit}(X)$, $q_i(X)$, $\natural(X)$, $\tau_j(X)$.

The children of the root store data values encoding the states and tape symbols used in $\hat{\delta}$. To ensure that $\hat{\delta}$ is stored properly on the target side, for each $(p_1, \sigma_1, p_2, \sigma_2, \ldots, p_6, \sigma_6) \in \hat{\delta}$ add a dependency

$$\begin{aligned} r[p_1(u_1), \sigma_1(v_1), p_2(u_2), \sigma_2(v_2), \ldots, p_6(u_6), \sigma_6(v_6)] &\longrightarrow \\ &\longrightarrow r/\_/\mathit{tr}(u_1, v_1, u_2, v_2, \ldots, u_6, v_6)\,. \end{aligned}$$

Note that d different dependencies are introduced, so each *tr* node in the target tree contains a tuple from $\hat{\delta}$.

The data values stored in two *bit* nodes are used to address the configurations and their cells. This is done by means of three auxiliary patterns, $\mathit{First}(\bar{x})$, $\mathit{Last}(\bar{x})$, and $\mathit{Succ}(\bar{x}, \bar{y})$ with $\bar{x} = x_1, x_2, \ldots, x_n$, $\bar{y} = y_1, y_2, \ldots, y_n$, which roughly speaking implement a binary counter over n bits. In the auxiliary patterns we use disjunction, but since we are only going to apply them on the source side of dependencies,

disjunction can be easily eliminated at the cost of multiplying dependencies. Let

$$\begin{aligned}
\mathit{First}(\bar{x}) &= \mathit{zero}\big[\mathit{bit}(x_1), \mathit{bit}(x_2), \ldots, \mathit{bit}(x_n)\big]\,,\\
\mathit{Last}(\bar{x}) &= \mathit{one}\big[\mathit{bit}(x_1), \mathit{bit}(x_2), \ldots, \mathit{bit}(x_n)\big]\,,\\
\mathit{Succ}(\bar{x},\bar{y}) &= \bigvee_{i=1}^{n}\left(\bigwedge_{j=1}^{i-1} _\big[\mathit{bit}(x_j), \mathit{bit}(y_j)\big], \mathit{zero}/\mathit{bit}(x_i), \mathit{one}/\mathit{bit}(y_i),\right.\\
&\qquad\left.\bigwedge_{j=i+1}^{n} \mathit{one}/\mathit{bit}(x_j) \wedge \mathit{zero}/\mathit{bit}(y_j)\right).
\end{aligned}$$

Using the auxiliary patterns we ensure that the target tree encodes an accepting run of M. In the first configuration the tape is empty and the head state q_0 is over the first cell,

$$\begin{aligned}
r[\mathit{First}(\bar{x}), \mathit{First}(\bar{y}), q_0(u), \flat^{\triangleright}(v)] &\longrightarrow r//\mathit{cell}(\bar{x},\bar{y},u,v)\,,\\
r[\mathit{First}(\bar{x}), \mathit{Succ}(\bar{z}_1,\bar{y}), \mathit{Succ}(\bar{y},\bar{z}_2), \natural(u), \flat(v)] &\longrightarrow r//\mathit{cell}(\bar{x},\bar{y},u,v)\,,\\
r[\mathit{First}(\bar{x}), \mathit{Last}(\bar{y}), \natural(u), \flat^{\triangleleft}(v)] &\longrightarrow r//\mathit{cell}(\bar{x},\bar{y},u,v)\,.
\end{aligned}$$

Note that the valuations of $\bar{y}$ correspond to all numbers from 0 to 2^n-1, and thus the content of each cell of the tape is specified.

The correctness of transitions is ensured by

$$\begin{aligned}
&r\big[\mathit{Succ}(\bar{x}_0,\bar{x}_1),\ \mathit{Succ}(\bar{y}_1,\bar{y}_2),\ \mathit{Succ}(\bar{y}_2,\bar{y}_3)\big] \longrightarrow\\
&\qquad\longrightarrow\ r\Big[_/\mathit{tr}(u_1,v_1,u_2,v_2,\ldots,u_6,v_6), \bigwedge_{i,j} //\mathit{cell}(\bar{x}_i,\bar{y}_j,u_{3i+j},v_{3i+j})\Big].
\end{aligned}$$

Again, $\bar{x}_0,\bar{x}_1$ range over $k, k+1$ with $0 \le k < 2^n-1$, and y_1,y_2,y_3 range over $\ell,\ell+1,\ell+2$ with $0 \le \ell < 2^n-2$, which means that the evolution of each three consecutive cells is checked for every step.

Finally, the machine needs to reach the accepting state (w.l.o.g. we assume that the accepting state is looping),

$$r\big[q_f(u)\big] \longrightarrow r//\mathit{cell}(\bar{x},\bar{y},u,v)\,.$$

Note that each occurrence of // above can be replaced with a sequence of _ and / symbols of suitable length.

It is straightforward to verify that M has an accepting computation of length at most 2^n if and only if there exists a tree which admits a solution with respect to the mapping defined above. □

6.3 Well-behaved classes

In this section we apply the solution building techniques developed in Section 5.4 to identify classes of mappings that admit more practical algorithms for the consistency problem. Unfortunately, classes with polynomial complexity are as rare as they are welcome. On the other hand, as the consistency problem only deals with syntactical objects (mappings), single exponential complexity might be acceptable in certain applications. To cut the complexity down to single exponential, we need to restrict the setting of Theorem 6.7 to fully specified mappings and forbid inequality on the source side. The proof of the result below is in the Appendix.

Horizontal navigation / Equality tests	NO	YES
NO	$\textsc{Cons}(\Downarrow)$ EXPTIME-complete	$\textsc{Cons}(\Downarrow, \Rightarrow, \mathsf{Fun})$ EXPTIME-complete
YES	$\textsc{Cons}(\downarrow, \downarrow^+, =)$ undecidable	$\textsc{Cons}(\downarrow, \rightarrow, =)$ undecidable

Fig. 3. Summary of consistency results

Horizontal navigation / Equality tests	NO	YES
NO	$\textsc{Cons}(\Downarrow)$ in PTIME	$\textsc{Cons}(\Downarrow, \rightarrow)$, $\textsc{Cons}(\Downarrow, \Rightarrow)$ PSPACE-complete
YES	$\textsc{Cons}(\Downarrow, \sim, \mathsf{Fun})$ NEXPTIME-complete	$\textsc{Cons}(\downarrow, \rightarrow, =)$ undecidable

Fig. 4. Summary of consistency results for nested-relational mappings

PROPOSITION 6.8. *For schema mappings from* $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, \sim, \mathsf{Fun})$ *which are fully specified and do not use* $\neq$ *on the source side,* CONS *is* EXPTIME*-complete. If either restriction is lifted, the problem becomes* NEXPTIME*-hard.*

The EXPTIME-algorithm from Proposition 6.8 can be easily extended to mappings using horizontal ordering under some restrictions. First, we need to make sure that the source side patterns are monotonic. This is guaranteed by forbidding $\rightarrow$. Also, our construction builds upon the algorithm constructing universal solutions, which needs the mapping to be non-bounding.

Using the same technique we can also extend the tractability of consistency for $\mathrm{SM}^{\mathrm{nr}}(\downarrow, =)$ [Arenas and Libkin 2008] to $\mathrm{SM}^{\mathrm{nr}}(\downarrow, \sim, \mathsf{Fun})$ under certain restrictions (see Appendix for details).

PROPOSITION 6.9. *For mappings from* $\mathrm{SM}^{\mathrm{nr}}(\downarrow, \sim, \mathsf{Fun})$ *which do not use* $\neq$ *on the source side,* CONS *is in* PTIME.

Again, the result above can be extended to mappings using horizontal ordering, provided that they do not use $\rightarrow$ on the source side, and are non-bounding. If $\rightarrow$ is not allowed on both sides, the argument in the proof of Proposition 6.9 carries over. In order to cover the full case of non-bounding mappings, instead of modifying the target DTD one should skip the completion step in the solution building algorithm. The resulting tree may not be a solution, but it will be a witness for the existence of one. The witness can be turned into a solution by the completion procedure.

To summarize our results, we see that it is the interplay between using equality and horizontal navigation that affects the complexity of consistency, with equality tests having more dramatic effect. These are shown in Figure 3 for arbitrary schema mappings, and in Figure 4 for nested-relational schema mappings.

7. COMPOSITION

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 [Fagin et al. 2004], 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:

—Complexity of composition;
—Consistency of composition: is $[\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$ empty?
—Syntactic definability of composition: can we find a mapping $\mathcal{M}_{13}$ such that $[\![\mathcal{M}_{13}]\!] = [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$?

The first and the last problem have been studied in the relational case; the second problem is motivated by the consistency problem for XML schema mappings themselves.

7.1 Complexity of composition

By analogy with the complexity of schema mappings, we define complexity of composition. *Combined complexity of composition* is the complexity of CompMembership:

Problem:	CompMembership
Input:	mappings $\mathcal{M}$, $\mathcal{M}'$, trees T, T'
Question:	$(T, T') \in [\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$?

Data complexity of composition is the complexity of CompMembership with $\mathcal{M}$ and $\mathcal{M}'$ fixed:

Problem:	CompMembership$(\mathcal{M}, \mathcal{M}')$
Input:	trees T, T'
Question:	$(T, T') \in [\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$?

Data complexity of relational composition is known to be in NP, and could be NP-complete for some mappings [Fagin et al. 2004]. For XML mappings, the problem becomes undecidable once data value comparisons are allowed.

Theorem 7.1. *There exist mappings* $\mathcal{M}, \mathcal{M}' \in \mathrm{SM}(\Downarrow, =)$ *for which* CompMembership$(\mathcal{M}, \mathcal{M}')$ *is undecidable. The same holds for* $\mathrm{SM}(\Downarrow, \neq)$, $\mathrm{SM}(\downarrow, \rightarrow, =)$, *and* $\mathrm{SM}(\downarrow, \rightarrow, \neq)$.

Proof. Our argument relies upon a particular property of the reductions used in the proof of Theorem 6.6: there, halting of a 2-register machine M is reduced to consistency of a mapping $\mathcal{M}_M = (D_s, D_t, \Sigma)$ whose each dependency had only a conjunction of equalities or inequalities on the target side. For such dependencies, the target sides' satisfaction does not depend on the target tree.

Now we need two fixed mappings for which the composition problem is undecidable. It is well known that there exists a universal 2-register machine U, such that it is undecidable whether for a given n the machine accepts with registers initialized to $(n, 0)$. We shall construct mappings $\mathcal{M}$ and $\mathcal{M}'$ and trees T_n so that $(T_n, r) \in [\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$ if and only if U accepts with registers initialized to $(n, 0)$.

For $\mathcal{M}'$ we take $\mathcal{M}_U$, except that we use

$$r[I_1(x, y), R(z)] \longrightarrow y = z$$

instead of $r[I_1(x, y), R(z)] \longrightarrow x = z, y = z$, to allow arbitrary initial values of the first register.

The mapping $\mathcal{M}$ ensures that the first register is initialized with the number encoded in the source tree T_n. Recall that $\mathcal{M}_U$ uses the data values stored in subsequent R-nodes as values of the registers: n is encoded as the n-th data value in this path, counting from the root. Therefore, to encode the value of the first register properly in T_n, we enforce concrete values in the initial n nodes of this path. The source DTD of $\mathcal{M}$ is $\{r \to R;\ R \to R \,|\, \sharp\}$ where R has a single attribute. For T_n we take the tree given by the pattern $r/R(a_1)/R(a_2)/\ldots/R(a_n)/\sharp$ where $a_1, a_2, \ldots, a_n$ are arbitrary pairwise different data values. The dependencies say that the initial segment of register values is copied correctly,

$$r/R(x) \longrightarrow r/R(x)\,,$$
$$r//R(x)/R(y) \longrightarrow r//R(x)/R(y)\,,$$

and that the first register is initialized to the nth value of this segment,

$$r//R(x)/\sharp \longrightarrow r/I_1(x, y)\,.$$

To see that the copying dependencies work as intended, recall that $\mathcal{M}_U$ contains a dependency $r//R(x)//R(x) \longrightarrow \bot$, which ensures that the values copied from T_n are stored in the initial segment of the path.

This shows the undecidability of $\mathrm{SM}(\downarrow, \downarrow^+, =)$ and $\mathrm{SM}(\downarrow, \downarrow^+, \neq)$. Analogous modification gives the undecidability of $\mathrm{SM}(\downarrow, \rightarrow, =)$ and $\mathrm{SM}(\downarrow, \rightarrow, \neq)$, even under the restriction to nested-relational DTDs. □

Without data value comparisons and Skolem functions, CompMembership 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 7.2. *For schema mappings from* $\mathrm{SM}(\Downarrow, \Rightarrow)$, CompMembership *is* 2-EXPTIME*-complete,* CompMembership$(\mathcal{M}, \mathcal{M}')$ *is in* EXPTIME*, and there exist* $\mathcal{M}, \mathcal{M}'$ *for which it is* EXPTIME*-complete.*

Proof. (1) First we show CompMembership is in 2-EXPTIME. Let $\mathcal{M} = (D_1, D_2, \Sigma_{12})$ and $\mathcal{M}' = (D_2, D_3, \Sigma_{23})$, with $\Sigma_{12} = \{\varphi_i \longrightarrow \psi_i \mid i = 1, 2, \ldots, n\}$ and $\Sigma_{23} = \{\gamma_j \longrightarrow \delta_j \mid j = 1, 2, \ldots, m\}$. Given trees $T_1 \models D_1$ and $T_3 \models D_3$, we have to check if there is an interpolating tree T_2.

For a start, consider a variable and attribute free situation. What are the constraints on the interpolating tree? Clearly, what is imposed by T_1, is that some of the target sides of dependencies from Σ_{12} have to be satisfied in T_2. In contrast, what is imposed by T_3, is that some source sides of dependencies from Σ_{23} are *not*

satisfied in T_2: indeed, $T_2 \models \gamma \implies T_3 \models \delta$ is equivalent to $T_3 \not\models \delta \implies T_2 \not\models \gamma$. Therefore, an interpolating tree T_2 exists iff the following automaton is non-empty:

$$\mathcal{A}_{D_2} \times \prod_{i\colon T_1 \models \varphi_i} \mathcal{A}(\psi_i) \times \prod_{j\colon T_3 \not\models \delta_j} \overline{\mathcal{A}(\gamma_j)}\,.$$

This gives an EXPTIME algorithm for the case without data values.

Let us now consider the general case, with variables. Still, neither equality nor inequality is allowed. In particular, no variable can be used more than once on the source side. The main idea is simply to throw in every data value used in T_1 to the alphabet. A tentative set of positive constraints imposed on T_2 by T_1 would be

$$\Psi = \left\{ \psi(\bar{a}, \bar{z}) \;\middle|\; \varphi(\bar{x}, \bar{y}) \longrightarrow \psi(\bar{x}, \bar{z}) \in \Sigma_{12}\,,\ T_1 \models \varphi(\bar{a}, \bar{b}) \right\}\,.$$

The problem is that $\psi(\bar{a}, \bar{z})$ still contains free variables which makes it impossible to use automata.

The solution is to guess valuations for free variables. What should the domain of our guess be? Let A be the set of data values used in T_1, and B the set of data values used in T_3. In an intermediate tree T_2 each data value not in $A \cup B$ can be safely replaced by any fixed data value, in particular, by a fixed data value from A. Indeed, such modification does not interfere with $\mathcal{M}$, since it does not use $\sim$ on the target side. To see that $\mathcal{M}'$ is not affected, note that as we do not have $\sim$ on the source side, the modification will not make any additional source side patterns true. Moreover, since we do not have $\sim$ on the target side, the only restrictions on the attribute values in T_2 are that they must fit in T_3. Attribute values in T_2 that do not come from $A \cup B$ do not fit in T_3 anyhow, which means they are never transferred to the target side, and their value is irrelevant. Hence, we can assume that T_2 only uses data values from $A \cup B$, and guess the values for the outstanding variables in Ψ from $A \cup B$. Let Ψ' denote the set Ψ with guessed data values for free variables (independently for each element of Ψ) and similarly let

$$\Gamma = \left\{ \gamma(\bar{a}, \bar{b}) \;\middle|\; \gamma(\bar{x}, \bar{y}) \longrightarrow \delta(\bar{x}, \bar{z}) \in \Sigma_{23}, \bar{a} \in (A \cup B)^{|\bar{x}|}, \bar{b} \in (A \cup B)^{|\bar{y}|}, T_3 \not\models \delta(\bar{a}, \bar{z}) \right\}\,.$$

The algorithm for CompMembership should construct the set Ψ (at most $|\Sigma_{12}| \cdot |A|^{\|\Sigma_{12}\|}$ polynomial checks), construct Γ (at most $|\Sigma_{23}| \cdot (|A| + |B|)^{\|\Sigma_{23}\|}$ single exponential checks), and then test non-emptiness of

$$\mathcal{A}_{D_2} \times \prod_{\psi \in \Psi'} \mathcal{A}(\psi) \times \prod_{\gamma \in \Gamma} \overline{\mathcal{A}(\gamma)}\,,$$

iterating over all possible Ψ' (at most $(|A|+|B|)^{|\Sigma_{12}| \cdot |A|^{\|\Sigma_{12}\|}}$ iterations, each taking time exponential in the size of Ψ' and Γ, i.e., double exponential in the size of the input). Altogether we obtain a 2-EXPTIME algorithm.

(2) The 2-EXPTIME lower bound can be obtained via a reduction from the membership problem for alternating EXPSPACE Turing machines, which is known to be 2-EXPTIME-hard. The reduction is similar to that from Theorem 6.7, but much more technical. The idea is that the intermediate tree encodes an accepting computation tree of the machine, whose correctness is enforced by the second mapping with the help of a suitable encoding of the transition relation of the machine stored in the target tree. The main difficulty is that we are not allowed to use

Problem	Data complexity	Combined complexity
Pattern evaluation	LOGSPACE-complete	PTIME
Membership SM($\downarrow$, Fun)	NP-complete	NEXPTIME-complete
Membership SM($\Downarrow, \Rightarrow, \sim$, Fun)	NP-complete	NEXPTIME-complete
Membership SM($\Downarrow, \Rightarrow, \sim$)	LOGSPACE-complete	Π_2^p-complete
Composition membership SM($\Downarrow, \Rightarrow$)	EXPTIME-complete	2EXPTIME-complete
Composition membership SM($\Downarrow, \sim$) and SM($\Rightarrow, \sim$)	undecidable	undecidable

Fig. 5. Summary of complexity results

data comparisons in patterns. We circumvent this obstacle by explicitly storing each needed equality/inequality relation on data tuples in the target tree. The first mapping and the source tree are trivial. Details can be found in the Appendix.

(3) Let us now move to CompMembership($\mathcal{M}, \mathcal{M}'$). If the mappings $\mathcal{M}, \mathcal{M}'$ are fixed, the size of the sets Ψ and Γ in the algorithm above is polynomial, so each iteration takes single exponential time. The number of iterations is also single exponential for fixed Σ_{12}. In consequence, CompMembership($\mathcal{M}, \mathcal{M}'$) is in EXPTIME for all $\mathcal{M}, \mathcal{M}' \in \mathrm{SM}(\Downarrow, \Rightarrow)$. The EXPTIME lower bound is shown by a reduction from non-universality problem for bottom-up non-deterministic automata on binary trees. The proof is a straightforward adaptation of the reduction from [Arenas and Libkin 2008]. Details can be found in the Appendix. □

In the EXPTIME lower bound we use both $\Downarrow$ and $\Rightarrow$. From the relational case [Fagin et al. 2004] we have NP lower bound for SM($\Downarrow$) and SM($\Rightarrow$). As the upper bound is still EXPTIME, this leaves a substantial gap. It would be also interesting to see if the upper bounds can be extended to SM($\Downarrow, \Rightarrow$, Fun).

Fig. 5 presents a summary of complexity results. 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 CompMembership($\mathcal{M}, \mathcal{M}'$) is EXPTIME-hard. By putting "undecidable" for data complexity we mean that there are mappings $\mathcal{M}, \mathcal{M}'$ such that CompMembership($\mathcal{M}, \mathcal{M}'$) is undecidable.

7.2 Consistency of composition

We say that the composition of $\mathcal{M}$ and $\mathcal{M}'$ is *consistent* if $[\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!] \neq \emptyset$. The consistency of composition problem comes in two flavors. 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:

Problem:	ConsComp(σ)
Input:	Two consistent mappings $\mathcal{M}, \mathcal{M}' \in \mathrm{SM}(\sigma)$
Question:	Is the composition of $\mathcal{M}$ and $\mathcal{M}'$ consistent?

It turns out that the complexity of ConsComp(σ) and Cons(σ) is the same.

THEOREM 7.3.

—CONSCOMP($\Downarrow$) *and* CONSCOMP($\Downarrow, \Rightarrow, \mathsf{Fun}$) *are* EXPTIME*-complete.*
—CONSCOMP($\downarrow, \downarrow^+, \sim$) *and* CONSCOMP($\downarrow, \rightarrow, \sim$) *are undecidable.*

The decidability result carries over to an arbitrary number of mappings, where the composition of $\mathcal{M}_1, \ldots, \mathcal{M}_n$ is defined as the composition of binary relations $[\![\mathcal{M}_i]\!]$.

PROPOSITION 7.4. *The problem of checking whether the composition of n mappings $\mathcal{M}_1, \ldots, \mathcal{M}_n$ from* SM($\Downarrow, \Rightarrow, \mathsf{Fun}$) *is consistent is in* EXPTIME.

7.3 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 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 [Fagin et al. 2009; Popa et al. 2002].

From Theorem 7.1 it follows immediately that finding a mapping expressing the composition is not always possible for any of the classes SM($\Downarrow, =$), SM($\Downarrow, \neq$), SM($\downarrow, \rightarrow, =$), and SM($\downarrow, \rightarrow, \neq$). The following examples give a more direct illustration of problems with composing XML schema mappings.

Let $D_1 = \{r \to \varepsilon\}, D_2 = \{r \to b_1|b_2;\ b_1, b_2 \to b_3\}$, and $D_3 = \{r \to c_1?c_2?c_3?\}$; no attributes are present. Let $\Sigma_{12} = \{r \longrightarrow r/\_/b_3\}$, $\Sigma_{23} = \{r/b_i \longrightarrow r/c_i \mid i = 1, 2\}$. 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 $r \to c_1?c_2?$ the composition would be definable by $r \longrightarrow r/\_$). 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 example, look at $D_1 = \{r \to a^*\}, D_2 = \{r \to b\, b\}$, and $D_3 = \{r \to \varepsilon\}$, with a and b having an attribute each, and mappings $r/a(x) \longrightarrow r/b(x)$ for Σ_{12} and $r \longrightarrow r$ 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., $r[a(x), a(y), a(z)] \longrightarrow (x = y \vee y = z \vee x = z)$.

In fact a variety of features such as wildcards or sibling order in patterns take us out of our usual classes of schema mappings. We now summarize what causes problems with composition.

PROPOSITION 7.5. *Consider DTDs $D_1 = \{r \to \varepsilon\}$ and $D_3 = \{r \to c_1?c_2?c_3?\}$ with no attributes. Then we can find schema mappings so that their composition, as a mapping between D_1 and D_3, contains exactly pairs (r, T), where T matches r/c_1 or r/c_2, if we allow the first mapping to be not fully specified or bounding, or the second mapping to use $\rightarrow$ or $\rightarrow^+$ or $\neq$.*

PROOF. For each of the mentioned features we provide an intermediate DTD D_2 and two sets of dependencies Σ_{12}, Σ_{23} such that the semantics of the composition

of (D_1, D_2, Σ_{12}) and (D_2, D_3, Σ_{23}) is

$$\{r\} \times \{r[c_1], r[c_2], r[c_1, c_2], r[c_1, c_3], r[c_2, c_3], r[c_1, c_2, c_3]\}\,.$$

Let us start with _ on the target side of Σ_{12}:

$$D_2 = \{r \to a_1^* a_2^*;\ a_1 \to b;\ a_2 \to b\}\,,$$

$$\Sigma_{12} = \{r \longrightarrow r/\_/b\}\,, \qquad \Sigma_{23} = \{r/a_1 \longrightarrow r/c_1;\ r/a_2 \longrightarrow r/c_2\}\,.$$

For an example for $\downarrow^+$ replace _ with // in the above, and for bounding consider

$$D_2 = \{r \to a\, b^*\, c;\ b \to b_1?\, b_2?\, b_3?\}\,,$$

$$\Sigma_{12} = \{r \longrightarrow r[a \to b/b_1 \to b/b_2 \to c, b/b_3]\}\,,$$

$$\Sigma_{23} = \{r/b[b_1, b_3] \longrightarrow r/c_1;\ r/b[b_2, b_3] \longrightarrow r/c_2\}\,.$$

An example for $\to$ on the source side of Σ_{23} is similar:

$$D_2 = \{r \to ab?c\}$$

$$\Sigma_{12} = \emptyset\,, \qquad \Sigma_{23} = \big\{r[a \to c] \longrightarrow r/c_1;\ r[b \to c] \longrightarrow r/c_2\big\}\,.$$

and for $\to^+$ it is slightly more complicated:

$$D_2 = \{r \to a^*;\ a \longrightarrow b_1?b_2?\}\,,$$

$$\Sigma_{12} = \big\{r \longrightarrow r\big[a/b_1, a/b_2\big]\big\}\,, \quad \Sigma_{23} = \big\{r[a/b_1 \to^+ a/b_2] \longrightarrow r/c_1;\ r[a/b_2 \to^+ a/b_1] \longrightarrow r/c_2;\ r/a[b_1, b_2] \longrightarrow r/c_2\big\}\,.$$

For inequality we make use of the fact that $x \neq y \implies x \neq z \vee y \neq z$ for all x, y, z (the a, b, and c nodes store a single attribute):

$$D_2 = \{r \to a^* b^* c^*\}\,,$$

$$\Sigma_{12} = \big\{r \longrightarrow r[a(C_1), b(C_2), c(C_3)]\,;\ C_2 = C_3 \longrightarrow \bot\big\} \quad \Sigma_{23} = \big\{r[a(x), c(z)], x \neq z \longrightarrow r/c_1;\ r[b(y), c(z)], y \neq z \longrightarrow r/c_2\big\}\,.$$

□

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

—the presence of disjunctions in DTDs;
—wildcard and descendant on the target side of the first mapping;
—next-sibling, following-sibling, and inequality on the source side of the second mapping.

When these features are forbidden, syntactic representation of the composition is guaranteed.

THEOREM 7.6. *For all* $\mathcal{M}_1, \mathcal{M}_2 \in \mathrm{SM}^{\mathrm{nr}}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$ *such that*

—$\mathcal{M}_1$ *is fully specified and non-bounding, and*

—$\mathcal{M}_2$ *uses neither* $\neq$ *nor* $\Rightarrow$ *on the source side,*

their composition belongs to $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$ *and can be computed in* EXPTIME.

One last ingredient we need to prove Theorem 7.6 is the notion of homomorphisms between patterns. A *homomorphism* h from a $\Downarrow$-pattern π into a $\downarrow, \Rightarrow$-pattern π' is a substitution of π's variables h_{Var} and a function $h_{\mathrm{Sub}} : \mathrm{Sub}(\pi) \to \mathrm{Sub}(\pi')$, mapping sub-patterns of π to sub-patterns of π', preserving

—the labeling and structure of patterns: $h_{\mathrm{Sub}}(\ell[\pi_1, \ldots, \pi_m]) = \ell[\mu_1, \ldots, \mu_n]$ and $h_{\mathrm{Sub}}(\pi_i) \in \{\pi' \mid \mu_j = \mu' \rightsquigarrow_1 \pi' \rightsquigarrow_2 \mu''$ for some $j, \rightsquigarrow_1, \rightsquigarrow_2, \mu', \mu''\}$ for all i (μ' and μ'' may be empty);

—the structure of terms: $h_{\mathrm{Sub}}(\ell(t)) = \ell(h_{\mathrm{Var}}(t))$.

For a formula φ with $\mathrm{Var}\,\varphi \subseteq \bar{x}$, and a tuple $\bar{a}$ of length equal to $\bar{x}$, we write $\bar{a}|_\varphi$ for the projection of $\bar{a}$ on the coordinates corresponding to $\mathrm{Var}\,\varphi$.

LEMMA 7.7. *If h is a homomorphism from π into π', then for all trees T, valuations of function symbols F, and tuples $\bar{a}$, if $T \models \pi'(\bar{a})$ then $T \models h_{\mathrm{Var}}(\pi)(\bar{a}|_{h_{\mathrm{Var}}(\pi)})$.*

PROOF. The composition of h with any homomorphism from $\pi'(\bar{a})$ to T gives a witnessing homomorphism from $h_{\mathrm{Var}}(\pi)(\bar{a}|_{h_{\mathrm{Var}}(\pi)})$ to T. □

Algorithm 1 lists the composition procedure. It uses the notation $X \hookleftarrow Y$ for $X := X \cup Y$. Recall also that $head(\sigma(\bar{t})[\lambda]) = \sigma$. The first instruction of the

Algorithm 1 Composing schema mappings.

Input: $\mathcal{M}_{12} = (D_1, D_2, \Sigma_{12})$, $\mathcal{M}_{23} = (D_2, D_3, \Sigma_{23}) \in \mathrm{SM}^{\mathrm{nr}}(\downarrow, =, \mathsf{Fun})$,
no variables introduced on target sides, source side patterns use only variables
Output: $\mathcal{M}_{13} = (D_1, D_3, \Sigma_{13}) \in \mathrm{SM}^{\mathrm{nr}}(\downarrow, =, \mathsf{Fun})$ s.t. $[\![\mathcal{M}_{13}]\!] = [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$

```
Σ̂12 := {φ ⟶ cpl_{D2}ψ | φ ⟶ ψ ∈ Σ12}
m := max{|Σ12|, 2} ∪ {‖φ‖ | φ → ψ ∈ Σ23}
Σ13 := ∅
for all φ1 ⟶ ψ1, φ2 ⟶ ψ2, …, φk ⟶ ψk ∈ Σ̂12, k ≤ m do
  {in case of repetitions, rename variables}
  ρ, η := mrg_{D2}(ψ1 ∧ ⋯ ∧ ψk)
  if ρ = ⊥ or head(ρ) ≠ r then Σ13 ↩ {φ1 ∧ ⋯ ∧ φk ⟶ ⊥} end if
  Σ13 ↩ {φ1 ∧ ⋯ ∧ φk ⟶ η}
  for all π, α ⟶ ψ ∈ Σ23 and all homomorphisms h from π to ρ do
    Σ13 ↩ {φ1 ∧ ⋯ ∧ φk ∧ h_Var(α) ⟶ h_Var(ψ)}
  end for
end for
return (D1, D3, Σ13)
```

algorithm simply ensures that in $\widehat{\Sigma}_{12}$ the target side patterns are complete with respect to D_2 (see page 25). By Lemma 5.12 the resulting set of dependencies is equivalent to the original. Hence, we can assume that $\widehat{\Sigma}_{12} = \Sigma_{12}$ and that the algorithm does not introduce any new function symbols. Then, the algorithm

essentially composes implications: if T_1 is such that patterns $\psi_1, \ldots, \psi_k$ have to be accommodated in T_2 because of Σ_{12}, and their presence in T_2 fires some rule of Σ_{23} enforcing ψ in T_3, then ψ should be also enforced by Σ_{13}. Additionally, if $\psi_1, \ldots, \psi_k$ are not satisfiable with respect to D_2, this information has to be reflected in Σ_{13}. Similarly, if $\psi_1, \ldots, \psi_k$ enforce some equalities η, they should also be enforced by Σ_{13}. Correctness of thus produced dependencies, given by Lemma 7.8 below, is quite intuitive. Completeness, shown in Lemma 7.9, relies on the fact that if a dependency of Σ_{23} is fired by the presence of some patterns in T_2, then there is a bounded size subset of these patterns, which already fires this dependency.

LEMMA 7.8. *If $(T_1, T_2) \in [\![\mathcal{M}_{12}]\!]$ and $(T_2, T_3) \in [\![\mathcal{M}_{23}]\!]$ with a witnessing valuation F, then $(T_1, T_3) \in [\![\mathcal{M}_{13}]\!]$ with the same witnessing valuation F.*

PROOF. Pick a constraint from Σ_{13}. First, suppose that it is of the form

$$\varphi_1 \wedge \cdots \wedge \varphi_k \longrightarrow \bot\,,$$

where $\varphi_1 \longrightarrow \psi_1,\ \varphi_2 \longrightarrow \psi_2,\ \ldots,\ \varphi_k \longrightarrow \psi_k \in \Sigma_{12}$ and $\mathrm{mrg}_{D_2}(\psi_1 \wedge \cdots \wedge \psi_k) = \bot$ or $\mathrm{mrg}_{D_2}(\psi_1 \wedge \cdots \wedge \psi_k) = (\rho, \alpha)$ with $\mathit{head}(\rho) \neq r$. Assuming that $T_1 \models \varphi_1 \wedge \cdots \wedge \varphi_k(\bar{c})$ for some $\bar{c}$, we get $T_2 \models \psi_1 \wedge \cdots \wedge \psi_k(\bar{c}|_{\psi_1 \wedge \cdots \wedge \psi_k})$, and by Lemma 5.13 (1) we obtain a contradiction, $T_2 \models \mathrm{mrg}_{D_2}(\psi_1 \wedge \cdots \wedge \psi_k)(\bar{c}|_{\psi_1 \wedge \cdots \wedge \psi_k})$.

Next, suppose the constraint is of the form

$$\varphi_1 \wedge \cdots \wedge \varphi_k \longrightarrow \eta\,,$$

where $\varphi_1 \longrightarrow \psi_1,\ \varphi_2 \longrightarrow \psi_2,\ \ldots,\ \varphi_k \longrightarrow \psi_k \in \Sigma_{12}$ and $(\rho, \eta) = \mathrm{mrg}_{D_2}(\psi_1 \wedge \cdots \wedge \psi_k)$. Pick $\bar{c}$ such that $T_1 \models \varphi_1 \wedge \cdots \wedge \varphi_k(\bar{c})$. Then $T_2 \models \psi_1 \wedge \cdots \wedge \psi_k(\bar{c}|_{\psi_1 \wedge \cdots \wedge \psi_k})$, and by Lemma 5.13 (1), $\eta(\bar{c}|_\eta)$ holds.

Finally, assume the constraint is of the form

$$\varphi_1 \wedge \cdots \wedge \varphi_k \wedge h_{\mathrm{Var}}(\alpha) \longrightarrow h_{\mathrm{Var}}(\psi)\,,$$

where $\varphi_i \longrightarrow \psi_i$ and (ρ, η) are like above, Σ_{23} contains dependency $\pi, \alpha \longrightarrow \psi$ and h is a homomorphism from π into ρ. Pick $\bar{c}$ such that $T_1 \models \varphi_1 \wedge \cdots \wedge \varphi_k \wedge h_{\mathrm{Var}}(\alpha)(\bar{c})$. Then $T_2 \models \psi_1 \wedge \cdots \wedge \psi_k(\bar{c}|_{\psi_1 \wedge \cdots \wedge \psi_k})$ and by Lemma 5.13 (1) we get $T_2 \models \rho(\bar{c}|_\rho)$. Hence $T_2 \models h_{\mathrm{Var}}(\pi)(\bar{c}|_{h_{\mathrm{Var}}(\pi)})$. As $h_{\mathrm{Var}}(\alpha)(\bar{c}|_{h_{\mathrm{Var}}(\alpha)})$ holds, we have $T_2 \models h_{\mathrm{Var}}(\pi, \alpha)(\bar{c}_{h_{\mathrm{Var}}(\pi,\alpha)})$ and hence $T_3 \models h_{\mathrm{Var}}(\psi)(\bar{c}|_{h_{\mathrm{Var}}(\psi)})$. □

LEMMA 7.9. *If $(T_1, T_3) \in [\![\mathcal{M}_{13}]\!]$ with a witnessing valuation F, then there exists an intermediate tree T_2 such that $(T_1, T_2) \in [\![\mathcal{M}_{12}]\!]$ and $(T_2, T_3) \in [\![\mathcal{M}_{23}]\!]$ with the same witnessing valuation F.*

PROOF. Let $\Delta = \big\{\psi(\bar{a}) \;\big|\; \varphi(\bar{x}, \bar{y}) \longrightarrow \psi(\bar{x}) \in \Sigma_{12}, T_1 \models \varphi(\bar{a}, \bar{b})\big\}$. By Lemma 4.4, a tree $T_2 \models D_2$ is a solution for T_1 if and only if $T_2 \models (\bigwedge_{\delta\in\Delta} \delta)$. We shall construct an intermediate tree T_2 from $\mathrm{mrg}_{D_2}(\bigwedge_{\delta\in\Delta} \delta)$.

First, we need to assert $\mathrm{mrg}_{D_2}(\bigwedge_{\delta\in\Delta} \delta) \neq \bot$. Assume $\mathrm{mrg}_{D_2}(\bigwedge_{\delta\in\Delta} \delta) = \bot$. By Lemma 5.13, $\big(\bigwedge_{\delta\in\Delta} \delta\big)^\circ = \bigwedge_{\delta\in\Delta} \delta^\circ$ is not satisfiable in a tree consistent with D_2. Whether $\bigwedge_{\delta\in\Delta} \delta^\circ$ is satisfiable or not depends only on $\big\{\delta^\circ \;\big|\; \delta \in \Delta\big\}$; multiplicities do not count. Consequently, there are distinct $\varphi_1 \longrightarrow \psi_1, \varphi_2 \longrightarrow \psi_2, \ldots, \varphi_\ell \longrightarrow \psi_\ell$ in Σ_{12} such that $T_1 \models \varphi_i(\bar{a}_i, \bar{b}_i)$ and $\psi_i(\bar{a}_i) \in \Delta$ for some $\bar{a}_i, \bar{b}_i$, and $\bigwedge_i (\psi_i(\bar{a}_i))^\circ = \bigwedge_i \psi_i^\circ = (\bigwedge_i \psi_i)^\circ$ is not satisfiable. At some iteration of the main loop the algorithm will process this set of dependencies. By Lemma 5.13, $\mathrm{mrg}_{D_2}(\bigwedge_i \psi_i) = \bot$, so

$\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_\ell \longrightarrow \bot$ will be added to Σ_{13}. Since $T_1 \models \varphi_i(\bar{a}_i, \bar{b}_i)$ for all i, $T_3 \models \bot$, which is a contradiction. Thus, $\mathrm{mrg}_{D_2}(\bigwedge_{\delta\in\Delta} \delta) \neq \bot$.

Next, we check that F is admissible for $\mathrm{mrg}_{D_2}(\bigwedge_{\delta\in\Delta} \delta)$, i.e., that each equality in $\mathrm{mrg}_{D_2}(\bigwedge_{\delta\in\Delta} \delta)$ holds under F. Pick an equality $t = t'$. Where does it come from? Either it was present already in some $\delta \in \Delta$, or it was introduced in step (3) of the merging procedure (page 26), when some two nodes v, v' of $\bigwedge_{\delta\in\Delta} \delta$ where merged. The nodes v, v' come either from the same $\delta \in \Delta$ or from two different $\delta, \delta' \in \Delta$. Let $\gamma = \delta$ or $\gamma = \delta \wedge \delta'$, accordingly. Observe that v and v' must also be merged in $\mathrm{mrg}_{D_2}\gamma$. Indeed, whether they are merged or not depends only on their ancestors and the nodes connected by a sequence of $\to$ to the ancestors, and these are identical in $\bigwedge_{\delta\in\Delta} \delta$ and in γ. Since v and v' are merged in $\mathrm{mrg}_{D_2}\gamma$, the equality $t = t'$ is entailed by $\mathrm{mrg}_{D_2}\gamma$, though not necessarily explicitly present.

Summing up, there are $\psi_1(\bar{a}_1), \ldots, \psi_k(\bar{a}_k) \in \Delta$, $k \leq 2$, such that $t = t'$ is entailed by the the equalities in $\mathrm{mrg}_{D_2}\left(\bigwedge_i \psi_i(\bar{a}_i)\right) = \left(\mathrm{mrg}_{D_2}\left(\bigwedge_i \psi_i(\bar{x}_i)\right)\right)(\bar{a}, \ldots, \bar{a}_k)$ (if $t = t'$ is present in $\delta \in \Delta$, it is also present in $\mathrm{mrg}_{D_2}\delta$). By the definition of Δ, for $i = 1, \ldots, k$, there is $\varphi_i \longrightarrow \psi_i \in \Sigma_{12}$ such that $T_1 \models \varphi_i(\bar{a}_i, \bar{b}_i)$ for some $\bar{b}_i$. At some iteration of the main loop the algorithm processes these k dependencies and adds to Σ_{13} dependency $\varphi_1 \wedge \cdots \wedge \varphi_k \longrightarrow \eta$, where $(\rho, \eta) = \mathrm{mrg}_{D_2}\left(\bigwedge_i \psi_i(\bar{x}_i)\right)$. Since $T_1 \models \varphi_i(\bar{a}_i, \bar{b}_i)$ for all i, we have $T_3 \models \eta((\bar{a}_1, \ldots, \bar{a}_k)|_\eta)$, which implies $T_3 \models t = t'$.

Thus F is admissible for $\mathrm{mrg}_{D_2}(\bigwedge_{\delta\in\Delta} \delta)$ and we can construct a tree T_2 from $\mathrm{mrg}_{D_2}(\bigwedge_{\delta\in\Delta} \delta)$ in the usual way by evaluating the terms. Since each $\delta \in \Delta$ is complete, so is $\bigwedge_{\delta\in\Delta} \delta$. Hence, by Corollary 5.14, $T_2 \models (\bigwedge_{\delta\in\Delta} \delta)$ and T_2 is consistent with D_2. The root of T_2 must have label r. Indeed, if it has label $\sigma \neq r$, then there is $\varphi \longrightarrow \psi$ in Σ_{12} such that $T_1 \models \varphi(\bar{a}, \bar{b})$ and $\psi(\bar{a}) \in \Delta$ for some $\bar{a}, \bar{b}$, and the root of ψ has label σ. At some iteration the algorithm processes $\varphi \longrightarrow \psi$, notices that the root of $\mathrm{mrg}_{D_2}\psi$ has label $\sigma \neq r$, and adds $\varphi \longrightarrow \bot$ to Σ_{13}. Since $T_1 \models \varphi(\bar{a}, \bar{b})$, we have $T_3 \models \bot$, which is a contradiction. Thus, $T_2 \models D_2$ and $(T_1, T_2) \in [\![\mathcal{M}_{12}]\!]$.

It remains to verify that $(T_2, T_3) \in [\![\mathcal{M}_{23}]\!]$. Pick $(\pi, \alpha)(\bar{x}, \bar{y}) \longrightarrow \psi(\bar{x}) \in \Sigma_{23}$. Suppose that $T_2 \models (\pi, \alpha)(\bar{a}, \bar{b})$ and let $g\colon \pi \to T_2$ be a witnessing homomorphism. Then, g can be interpreted as a homomorphism from π to the pure pattern ρ_Δ underlying $\mathrm{mrg}_{D_2}(\bigwedge_{\delta\in\Delta} \delta)$, such that $g_{\mathrm{Var}}(\bar{x}) = \bar{a}$ and $g_{\mathrm{Var}}(\bar{y}) = \bar{b}$. Let $v_1, \ldots, v_k$ be the nodes of ρ_Δ that are in the image of g, $k \leq \|\pi\|$. Each v_i can be traced back to a pattern $\psi_i(\bar{a}_i) \in \Delta$: when a node is obtained by merging two nodes in segments κ and κ' in step (3) of the merging procedure, choose κ. Let $\rho, \eta = \mathrm{mrg}_{D_2}\left(\bigwedge_i \psi_i(\bar{a}_i)\right)$. Since each two nodes are merged in $\mathrm{mrg}_{D_2}\left(\bigwedge_i \psi_i(\bar{a}_i)\right)$ iff they are merged in $\mathrm{mrg}_{D_2}(\bigwedge_{\delta\in\Delta} \delta)$, ρ is contained in ρ_Δ (more precisely, ρ can be mapped injectively into ρ_Δ). Since ρ also contains the image of g, we can see g as a homomorphism $g\colon \pi \to \rho$. By the definition of Δ, for $i = 1, \ldots, k$, there is $\varphi_i \longrightarrow \psi_i \in \Sigma_{12}$ such that $T_1 \models \varphi_i(\bar{a}_i, \bar{b}_i)$ for some $\bar{b}_i$. By equation (4) on page 27, we have $\rho = \hat{\rho}((\bar{a}_1, \ldots, \bar{a}_k)|_{\hat{\rho}})$ for $(\hat{\rho}, \hat{\eta}) = \mathrm{mrg}_{D_2}(\psi_1 \wedge \cdots \wedge \psi_k)$. Recall that π uses each variable exactly once, and it does not use function symbols. This allows us to lift $g\colon \pi \to \rho$ to a homomorphism $\hat{g}\colon \pi \to \hat{\rho}$ such that

$$(\hat{g}_{\mathrm{Var}}(\bar{x}))\left((\bar{a}_1, \ldots, \bar{a}_k)|_{g_{\mathrm{Var}}(\bar{x})}\right) = \bar{a}\,, \quad (\hat{g}_{\mathrm{Var}}(\bar{y}))\left((\bar{a}_1, \ldots, \bar{a}_k)|_{g_{\mathrm{Var}}(\bar{y})}\right) = \bar{b}\,. \qquad (6)$$

Consequently, Σ_{13} contains $\varphi_1 \wedge \cdots \wedge \varphi_k \wedge \hat{g}_{\mathrm{Var}}(\alpha) \to \hat{g}_{\mathrm{Var}}(\psi)$. Recall that $T_1 \models \varphi_i(\bar{a}_i, \bar{b}_i)$ for all i. By (6), $\hat{g}_{\mathrm{Var}}(\alpha)((\bar{a}_1, \ldots, \bar{a}_k)|_{\hat{g}_{\mathrm{Var}}(\alpha)})$ is identical to $\alpha\left((\bar{a}, \bar{b})|_\alpha\right)$, and thus guaranteed by $T_2 \models (\pi, \alpha)(\bar{a}, \bar{b})$. Hence, $T_3 \models \hat{g}_{\mathrm{Var}}(\psi)((\bar{a}_1, \ldots, \bar{a}_k)|_{\hat{g}_{\mathrm{Var}}(\psi)})$,

which is the same as $T_3 \models \psi(\bar{a})$. □

Combining the necessary restrictions on the target sides of the first mapping and the source sides of the second mapping, we obtain that the class of fully specified non-bounding mappings from $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, \Rightarrow, =, \mathsf{Fun})$ without $\Rightarrow$ on the source side is closed under composition. The use of sibling order in this class seems marginal; eliminating it we obtain a more elegant result.

COROLLARY 7.10. *The class of fully specified mappings from* $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, =, \mathsf{Fun})$ *is closed under composition.*

8. CONCLUSIONS

This paper has offered a detailed investigation of various features of XML schema mappings and analyzed their effect on the complexity of the main computational tasks associated with data exchange and metadata management.

One outcome of this analysis we would like to highlight is that the class of fully specified mapping from $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, =, \mathsf{Fun})$ has particularly good properties. It subsumes nested relations (and non-relational extensions of data exchange systems such as those in [Popa et al. 2002; Yu and Popa 2004]) and all the key problems related to this class are tractable: polynomial in the size of data, and single exponential in the size of syntactic objects (mappings, queries). The class is closed under composition (Theorem 7.10) and the composition can be computed in EXPTIME. The consistency problem is in EXPTIME (Proposition 6.8). Universal solutions exist for all source trees and can be computed in PTIME (Theorem 5.15). In consequence, query answering is in PTIME (Theorem 5.9). Combined complexity of both these algorithms is single-exponential, matching the relational case [Fagin et al. 2003].

Therefore, this class of mappings might be a good starting point for real-life applications. Moreover, for some of the mappings from this class, algorithms for implementing data exchange tasks using relational data exchange systems have been worked out [Chirkova et al. 2012], which eliminates the need for building specially tailored systems to handle them.

Future work. We would like to extend this work in several directions. So far, we have concentrated on data complexity of the query answering problem. We would also like to look at combined complexity in the future in order to have a better understanding of query answering in XML schema mappings. We have obtained some of the upper bounds, in particularly for query answering, by using results from [Barceló et al. 2010], which do not even yield an elementary bound for combined complexity. However, it was shown recently [Gheerbrant et al. 2012; David et al. 2013] that for some query answering problems over incomplete XML documents, an alternative construction reduces combined complexity to single exponential, thus opening up a possibility of finding good combined complexity bounds.

Although we have shown that it is rather difficult to extend the query language, there might still be some hope to extend it in a limited way, as was done for queries with inequalities in relational data exchange [Arenas et al. 2009].

Yet another dimension that has not been investigated is the distinction between open world assumption (OWA) and closed world assumption (CWA). Here, we have worked under OWA. In the relational case, an anomaly is observed when the query

involves negation [Arenas et al. 2004; Fagin et al. 2003]. As a remedy to such unintuitive behavior, the notion of solutions under CWA was proposed in [Libkin 2006], further extended in [Afrati and Kolaitis 2008; Hernich et al. 2011; Libkin and Sirangelo 2011]. This direction is hardly explored for XML: it is not even clear how to define the notion of CWA in the XML context.

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 [Arenas et al. 2009; Fagin et al. 2007] or merge [Bernstein and Melnik 2007]. And we would like to extend structural results of [ten Cate and Kolaitis 2009] from relational to XML mappings.

Acknowledgments. The authors thank the anonymous reviewers for their helpful comments. Most of this work was done when the authors were at the University of Edinburgh. The authors were supported by EPSRC grants G049165 and J015377, and the FET-Open Project FoX (grant agreement 233599). A part of the work was done within the *Querying and Managing Navigational Databases* project realized within the Homing Plus program of the Foundation for Polish Science, co-financed by the European Union from the Regional Development Fund within the Operational Programme Innovative Economy ("Grants for Innovation").

REFERENCES

Abiteboul, S., Kanellakis, P., and Grahne, G. 1991. On the representation and querying of sets of possible worlds. *Theoretical Computer Science 78,* 1, 158–187.

Afrati, F. and Kolaitis, P. 2008. Answering aggregate queries in data exchange. In *ACM Symposium on Principles of Database Systems (PODS)*. 129–138.

Amano, S., David, C., Libkin, L., and Murlak, F. 2010. On the tradeoff between mapping and querying power in XML data exchange. In *International Conference on Database Theory (ICDT)*. 155–164.

Amano, S., Libkin, L., and Murlak, F. 2009. XML schema mappings. In *ACM Symposium on Principles of Database Systems (PODS)*. 33–42.

Amer-Yahia, S., Cho, S., Lakshmanan, L., and Srivastava, D. 2002. Tree pattern query minimization. *VLDBJ 11*, 315–331.

Arenas, M., Barceló, P., Fagin, R., and Libkin, L. 2004. Locally Consistent Transformations and Query Answering in Data Exchange. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems, PODS'04*. 229–240.

Arenas, M., Barceló, P., Libkin, L., and Murlak, F. 2010. *Relational and XML Data Exchange*. Morgan&Claypool Publishers.

Arenas, M., Barceló, P., and Reutter, J. 2009. Query languages for data exchange: beyond unions of conjunctive queries. In *International Conference on Database Theory (ICDT)*. 73–83.

Arenas, M. and Libkin, L. 2008. XML data exchange: Consistency and query answering. *J. ACM 55,* 2.

Arenas, M., Pérez, J., and Riveros, C. 2009. The recovery of a schema mapping: Bringing exchanged data back. *ACM Transactions on Database Systems 34,* 4, 22:1–22:48.

Barbosa, D., Freire, J., and Mendelzon, A. 2005. Designing information-preserving mapping schemes for xml. In *Very Large Data Bases (VLDB)*. 109–120.

Barceló, P. 2009. Logical Foundations of Relational Data Exchange. *SIGMOD Record 38,* 1, 49–58.

Barceló, P., Libkin, L., Poggi, A., and Sirangelo, C. 2010. XML with incomplete information. *Journal of the ACM 58,* 1.

Benedikt, M., Fan, W., and Geerts, F. 2008. XPath satisfiability in the presence of DTDs. *Journal of the ACM 55,* 2.

Bernstein, P. A. and Melnik, S. 2007. Model management 2.0: manipulating richer mappings. In *ACM SIGMOD Conference*. 1–12.

Björklund, H., Martens, W., and Schwentick, T. 2007. Conjunctive query containment over trees. In *DBPL*. 66–80.

Björklund, H., Martens, W., and Schwentick, T. 2008. Optimizing Conjunctive Queries over Trees Using Schema Information. In *MFCS*. 132–143.

Bojanczyk, M., Kolodziejczyk, L. A., and Murlak, F. 2013. Solutions in xml data exchange. *J. Comput. Syst. Sci. 79,* 6, 785–815.

Bojanczyk, M., Muscholl, A., Schwentick, T., and Segoufin, L. 2009. Two-variable logic on data trees and XML reasoning. *J. ACM 56,* 3.

Chirkova, R., Libkin, L., and Reutter, J. 2012. Tractable XML data exchange via relations. *Frontiers of Computer Science 6,* 3, 243–263.

Chiticariu, L. and Tan, W.-C. 2006. Debugging schema mappings with routes. In *International Conference on Very Large Data Bases (VLDB)*. 79–90.

David, C. 2008. Complexity of Data Tree Patterns over XML Documents. In *MFCS*. 278–289.

David, C., Gheerbrant, A., Libkin, L., and Martens, W. 2013. Containment of pattern-based queries over data trees. In *International Conference on Database Theory (ICDT)*. 201–212.

David, C., Libkin, L., and Murlak, F. 2010. Certain answers for XML queries. In *ACM Symposium on Principles of Database Systems (PODS)*, J. Paredaens and D. V. Gucht, Eds. ACM, 191–202.

Fagin, R., Haas, L., Hernandez, M., Miller, R., Popa, L., and Velegrakis, Y. 2009. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications, Essays in Honor of John Mylopoulos*, A. Borgida, V. Chaudhri, P. Giorgini, and E. Yu, Eds. Lecture Notes in Computer Science, vol. 5600. Springer-Verlag, 198–236.

Fagin, R., Kolaitis, P., Miller, R., and Popa, L. 2003. Data exchange: semantics and query answering. In *Proceedings Ninth International Conference on Database Theory, ICDT'03. Full version in Theoretical Computer Science*. 207–224.

Fagin, R., Kolaitis, P. G., Popa, L., and Tan, W. C. 2004. Composing Schema Mappings: Second-Order Dependencies to the Rescue. In *ACM Symposium on Principles of Database Systems (PODS)*. 83–94.

Fagin, R., Kolaitis, P. G., Popa, L., and Tan, W. C. 2007. Quasi-inverses of schema mappings. In *Proceedings of the 26th ACM Symposium on Principles of Database Systems (PODS)*. 123–132.

Fan, W. and Bohannon, P. 2008. Information preserving xml schema embedding. *ACM Transactions on Database Systems 33,* 1.

Fan, W. and Libkin, L. 2002. On XML integrity constraints in the presence of DTDs. *Journal of the ACM 49,* 3, 368–406.

Gheerbrant, A., Libkin, L., and Tan, T. 2012. On the complexity of query answering over incomplete XML documents. In *International Conference on Database Theory (ICDT)*. 169–181.

Gottlob, G., Koch, C., and Schulz, K. 2006. Conjunctive queries over trees. *Journal of the ACM 53,* 2, 238–272.

Gottlob, G. and Senellart, P. 2010. Schema mapping discovery from data instances. *J. ACM 57,* 2.

Hernich, A., Libkin, L., and Schweikardt, N. 2011. Closed world data exchange. *ACM Transactions on Database Systems 36,* 2.

Hidders, J. 2003. Satisfiability of XPath expressions. In *DBPL03*. 21–36.

Kolaitis, P. G. 2005. Schema Mappings, Data Exchange, and Metadata Management. In *ACM Symposium on Principles of Database Systems (PODS)*. 61–75.

Kolaitis, P. G., Panttaja, J., and Tan, W. C. 2006. The complexity of data exchange. In *ACM Symposium on Principles of Database Systems (PODS)*. 30–39.

Lewis, H. 1980. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences 21*, 317–353.

Libkin, L. 2006. Data Exchange and Incomplete Information. In *ACM Symposium on Principles of Database Systems (PODS)*. 60–69.

Libkin, L. and Sirangelo, C. 2011. Data exchange and schema mappings in open and closed worlds. *Journal of Computer and System Sciences 77,* 3, 542–571.

Madhavan, J. and Halevy, A. Y. 2003. Composing Mappings Among Data Sources. In *International Conference on Very Large Data Bases (VLDB)*. 572–583.

Marnette, B., Mecca, G., Papotti, P., Raunich, S., and Santoro, D. 2011. ++Spicy: an opensource tool for second-generation schema mapping and data exchange. *PVLDB 4,* 12, 1438–1441.

Mądry, A. 2005. Data exchange: On the complexity of answering queries with inequalities. *Information Processing Letters 94,* 6, 253–257.

Melnik, S., Garcia-Molina, H., and Rahm, E. 2002. Similarity flooding: a versatile graph matching algorithm. In *ICDE*. 117–128.

Miklau, G. and Suciu, D. 2004. Containment and equivalence for a fragment of xpath. *J. ACM 51,* 1, 2–45.

Milo, T. and Zohar, S. 1998. Using schema matching to simplify heterogeneous data translation. In *Very Large Data Bases (VLDB)*. 122–133.

Nash, A., Bernstein, P. A., and Melnik, S. 2007. Composition of mappings given by embedded dependencies. *ACM Trans. Database Syst. 32,* 1, 4:1–4:51.

Papadimitriou, C. 1994. *Computational Complexity*. Addison-Wesley.

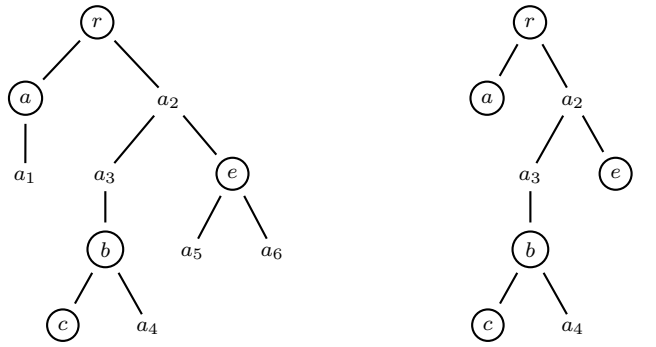
Pichler, R. and Skritek, S. 2011. The complexity of evaluating tuple generating dependencies. In *ICDT*. 244–255.

Popa, L., Velegrakis, Y., Miller, R., Hernández, M., and Fagin, R. 2002. Translating web data. In *Very Large Databases (VLDB)*. 598–609.

Segoufin, L. 2006. Automata and Logics for Words and Trees over an Infinite Alphabet. In *Computer Science Logic*. 41–57.

ten Cate, B. and Kolaitis, P. 2009. Structural Characterizations of Schema-Mapping Languages. In *International Conference on Database Theory (ICDT)*. 63–72.

Yu, C. and Popa, L. 2004. Constraint-based XML query rewriting for data integration. In *ACM SIGMOD Conference*. 371–382.



(a) the image in the tree (b) the image in the support

Fig. 6. A homomorphism from $r[//a, //b/c, //e]$ into a tree and its support

A. COMPLEXITY OF SCHEMA MAPPINGS

Let the *support* of a homomorphism h from φ into T, denoted $\operatorname{supp} h$, be the subtree of T obtained by keeping only nodes that have a descendant or a sibling in the range of h. For example, consider a tree pattern $r[//a, //b/c, //e]$. This pattern is satisfied by a tree T given in Figure 6(a) with the obvious homomorphism h which appropriately assigns sub-formulae to the encircled nodes. To obtain $\operatorname{supp} h$, we remove all nodes except from ancestors of the nodes from the range of h, and their siblings. The result is shown in Figure 6(b).

PROPOSITION 4.1. *The satisfiability problem for tree patterns is* NP*-complete.*

PROOF. We show that for each pattern φ satisfiable with respect to a DTD D over Γ, there exists a homomorphism from φ to some T conforming to D with the support of size $\mathcal{O}(\|\varphi\| \cdot \|D\|)$. Take an arbitrary T conforming to D and satisfying φ. Let h be a homomorphism from φ to T. Divide the nodes of $\operatorname{supp} h$ into four categories: the nodes from the image of h are *red*, the nodes that are not red and have more than one child that is an ancestor of a red node (or is red itself) are *green*, the others are *yellow* if they are ancestors of red nodes, and *blue* otherwise. For example, in Figure 6(b) the encircled nodes are red, a_2 is green, a_3 is yellow, and a_4 is blue. Let N_{red}, N_{green}, N_{yellow}, and N_{blue} be the numbers of red, green, yellow, and blue nodes.

By definition, $N_{\text{red}} \leq \|\varphi\|$. Also $N_{\text{green}} \leq \|\varphi\|$: when going bottom-up, each green node decreases the number of subtrees containing a red node by at least one, and since in the root we arrive with one subtree containing a red node, $N_{\text{green}} \leq N_{\text{red}}$. By a pumping argument we may assume that all yellow paths in $\operatorname{supp} h$ are not longer than the number of labels in Γ. Similarly, all blue sequences of siblings in $\operatorname{supp} h$ are not longer than the size of the regular expressions in the productions of D, which can be bounded by $\|D\|$. The number of (maximal) yellow paths is at most $N_{\text{red}} + N_{\text{green}}$. Hence there are at most $2\|\varphi\| \cdot \|D\|$ yellow nodes. Since all blue nodes are siblings of nodes of other colors, the number of (maximal) blue sequences of siblings is at most $2(N_{\text{red}} + N_{\text{green}} + N_{\text{yellow}}) \leq 4\|\varphi\| \cdot (\|D\| + 1)$ and so

$N_{\text{blue}} \leq 4\|\varphi\|\cdot(\|D\|+1)\|D\|$. Altogether we have at most $2\|\varphi\|(\|D\|+1)(2\|D\|+1) \leq 12\|\varphi\| \cdot \|D\|^2$ nodes.

Now, to decide satisfiability, first guess a polynomial support and a homomorphism. Verifying the homomorphism is polynomial in the size of the formula and the support, hence it is polynomial. Verifying that the support is actually a restriction of a tree conforming to D amounts to checking if a given word is in the language defined by a regular expression, and checking if there exist subtrees to be rooted in the yellow nodes. Both these checks can be done in polynomial time.

To get NP-completeness we do a standard 3CNF SAT reduction. In fact, we only use $\downarrow, \_$ and we do not need variables. Take a formula $\theta = \bigwedge_{j=1}^{k} Z_j^1 \vee Z_j^2 \vee Z_j^3$ with $Z_j^i \in \{x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$. Consider a DTD D (without attributes)

$$r \to x_1 x_2 \cdots x_n$$
$$x_i \to \left\{C_j \;\middle|\; \exists \ell Z_j^\ell = x_i\right\}|\left\{C_j \;\middle|\; \exists \ell Z_j^\ell = \bar{x}_i\right\} \qquad 1 \leq i \leq n$$

over the alphabet $\{x_1, x_2, \ldots, x_n, C_1, C_2, \ldots, C_k\}$. In the second rule, interpret each set as a concatenation of all its elements.

The labels C_j are intended to correspond to $Z_j^1 \vee Z_j^2 \vee Z_j^3$. Each tree conforming to D encodes a valuation of all variables x_i: for each x_i it stores either all conjuncts made true by assigning 1 to x_i, or all conjuncts made true by assigning 0 to x_i.

The satisfiability of θ is equivalent to the satisfiability of the pattern $r[\_/C_1, \_/C_2, \ldots, \_/C_k]$ with respect to D. □

PROPOSITION 4.3. *The data complexity of tree patterns evaluation is* LOGSPACE*-complete, and the combined complexity is in* PTIME.

PROOF. Take a tree pattern $\varphi = \pi, \alpha$, a valuation $\bar{a}$ and a tree T. Checking that $T \models \varphi(\bar{a})$ can be done in PTIME by checking if $\alpha(\bar{a})$ and evaluating the sub-patterns of $\pi(\bar{a})$ bottom-up. Annotate each node v with a set $\Phi(v)$ containing those sub-patterns of $\pi(\bar{a})$ which are satisfied in v. If v is a leaf labeled with σ and storing a tuple $\bar{b}$, let $\Phi(v)$ contain all sub-patterns of $\pi(\bar{a})$ of the form $\sigma(\bar{b})$ and $\_(\bar{b})$. If v is an internal node labeled with σ, having children $v_1, v_2, \ldots, v_k$, and storing a tuple $\bar{b}$, let $\Phi(v)$ contain all sub-patterns of $\pi(\bar{a})$ of the form $\sigma'(\bar{b})[\lambda_1, \lambda_2, \ldots, \lambda_p]$ satisfying

—$\sigma' \in \{\sigma, \_\}$,
—for each $\lambda_i = //\pi_1$ there exists a node v_j such that $//\pi_1 \in \Phi(v_j)$ or $\pi_1 \in \Phi(v_j)$,
—for each $\lambda_i = \pi_1 \rightsquigarrow_1 \pi_2 \rightsquigarrow_2 \ldots \rightsquigarrow_{r-1} \pi_r$ there exists a sequence $1 \leq n_1 < n_2 < \ldots < n_r \leq k$ such that $\pi_j \in \Phi(v_{n_j})$, and if $\rightsquigarrow_j = \to$ then $n_{j+1} = n_j + 1$ for all j,

and all sub-patterns of $\varphi(\bar{a})$ of the form $//\pi_1$ satisfying $\pi_1 \in \Phi(v_j)$ or $//\pi_1 \in \Phi(v_j)$ for some j. Clearly, $T \models \pi(\bar{a})$ iff $\pi(\bar{a}) \in \Phi(\varepsilon)$.

Let us now consider the data complexity, i.e., suppose we are given T and $\bar{a}$ and are to check if $T \models \varphi(\bar{a})$ for some fixed pattern $\varphi = \pi, \alpha$. Equalities and inequalities in $\alpha(\bar{a})$ can be verified in LOGSPACE. In order to store a homomorphism from $\pi(\bar{a})$ to T, we need only logarithmic space: a fixed number of "pointers" to the tree. Verifying a given homomorphism amounts to reachability tests in the tree plus local consistency checks, which can be done in LOGSPACE. Hence, if only we can do the reachability test in LOGSPACE, we can get a LOGSPACE algorithm

by simply iterating over all possible homomorphisms until we get a correct one, or find out that none such exists.

To see that we can do reachability tests, notice that one can compute the descendant and the following-sibling relations in a tree in LOGSPACE since reachability is known to be in LOGSPACE over graphs in which every node has at most one outgoing edge. This is the case for the next-sibling relation; for the child relation, we simply invert it, obtaining the parent relation that satisfies the property, compute its transitive closure, which gives us the ancestor relation, and then invert it to get descendant.

Finally, LOGSPACE-hardness follows from the hardness of reachability over successor-relations; hence even evaluating $r[a \to^+ b]$ over a tree of depth 1 is LOGSPACE-hard. □

THEOREM 4.6. *For schema mappings from* $\mathrm{SM}(\Downarrow, \Rightarrow, \sim, \mathsf{Fun})$ *with bounded arity of terms and no explicit equality,* MEMBERSHIP *is* Σ_3^p*-complete, and the lower bound holds already for relational mappings without* $\sim$*.*

PROOF. We say that a term $s(y_1, y_2, \dots, y_k)$ is a *fragment* of a term t if there are terms $s_1, s_2, \dots, s_k$ such that $s(s_1, s_2, \dots, s_k)$ is a subterm of t and each s_i contains at least one variable. Note that the arity of s is then bounded by the arity of t. The algorithm to test if $(S, T) \in [\![M]\!]$ is as follows:

—Assume that the data domain is partitioned into the set A of values used in S or in T, and the infinite set of nulls $N = \{\bot_1, \bot_2, \dots\}$.

—Guess consistent values from $A \cup N$ for all ground instances of fragments of terms used in $\mathcal{M}$ that can be obtained by substituting variables with values from A.

—Universally choose a constraint $\varphi(\bar{x}, \bar{y}) \longrightarrow \psi(\bar{x}, \bar{z})$ from Σ and tuples $\bar{a}, \bar{b}$ with entries from A such that $S \models \varphi(\bar{a}, \bar{b})$.

—Guess a tuple $\bar{c}$ with entries from $A \cup N'$ (a local set of fresh nulls $N' = \{\bot'_1, \bot'_2, \dots, \}$) and consistent values from $A \cup N \cup N'$ for all subterms in $\psi(\bar{a}, \bar{c})$ such that: if at least one of the terms t_i gets a value from N' then $f(t_1, \dots, t_k)$ gets a value from $A \cup N'$ (†).

—Accept if $T \models \psi(\bar{a}, \bar{c})$ under the guessed partial valuation.

Since the arity of terms is bounded, we have polynomially many ground instances and the first guess of the algorithm is polynomial. When the tests $S \models \varphi(\bar{a}, \bar{b})$ and $T \models \psi(\bar{a}, \bar{c})$ are performed, all relevant terms have fixed values, so by Proposition 4.3 we can do them in polynomial time. Thus, the algorithm is in Σ_3^p.

Let us move to correctness. Assume that the algorithm assigns to $f(t_1, t_2, \dots, t_k)$ value d, and to t_i value d_i for all i. This enforces constraint $f(d_1, d_2, \dots, d_k) = d$ on the valuation of f. We claim that either $d_i \in N'$ for some i, or this constraint is already enforced by the initial guess. From this claim it follows immediately that the guesses made by all the universal branches of the algorithm are consistent: constraints of the first kind coming from different branches do not interfere with each other because each branch has its own local N', constraints of the second kind are respected by all branches. To see that the claim holds, suppose that $d_1, \dots, d_k \in A \cup N$. By the condition (†), it follows that for each i, $t_i = s_i(s_i^1, s_i^2, \dots, s_i^{\ell_i})$ such that for each j the value of s_i^j was not fixed by the initial guess, but is now

$e_i^j \in A$, and $s_i(e_i^1, e_i^2, \ldots, e_i^{\ell_i})$ is a ground term with the value d_i fixed by the initial guess. Hence, the value d of $f(s_1(e_1^1, e_1^2, \ldots, e_1^{\ell_1}), \ldots, s^k(e_1^k, e_2^k, \ldots, e_{\ell_k}^k))$ was also fixed by the initial guess, which implies constraint $f(d_1, d_2, \ldots, d_k) = d$. Thus, if the algorithm accepts, all guesses are consistent and there is a valuation F of function symbols witnessing $(S, T) \in [\![\mathcal{M}]\!]$.

Conversely, suppose that there is a witnessing valuation F over domain $A \cup N$. We can assume that in the first step the algorithm guesses values consistent with F. Take $\varphi(\bar{x}, \bar{y}) \longrightarrow \psi(\bar{x}, \bar{z})$ and $\bar{a}, \bar{b}$ such that $S \models \varphi(\bar{a}, \bar{b})$. There exists a tuple $\bar{d}$ with entries in $A \cup N$ such that $T \models \psi(\bar{a}, \bar{d})$ under valuation F. To show completeness of the algorithm it suffices to find $\bar{c}$ with entries in $A \cup N'$ and modify F to meet condition (†). Let $\bar{c}$ be obtained from $\bar{d}$ by replacing each $\perp_i$ with $\perp_i'$. We extend valuation F to N' as follows: $F(f)$ treats $\perp_i'$ just like $\perp_i$, except that it adds a prime to the result if it is a null, e.g., if $F(f)(a, \perp_1, \perp_2) = \perp_3$, then $F(f)(a, \perp_1', \perp_2) = F(f)(a, \perp_1, \perp_2') = F(f)(a, \perp_1', \perp_2') = \perp_3'$. The extended F obviously meets condition (†). To see that $T \models \psi(\bar{a}, \bar{d})$ holds under extended F, recall that there are no explicit equalities in ψ, and note that our modifications do not change values of terms that where originally in A and do not equate terms that were not equal originally. This concludes the correctness proof.

To prove that MEMBERSHIP is Σ_3^p-hard, we reduce validity of Σ_3 quantified Boolean formulae, building upon the Π_2^p-hardness for relational mappings without Skolem functions [Gottlob and Senellart 2010]. Satisfiability of

$$\exists x_1 \cdots \exists x_\ell \forall x_{\ell+1} \cdots \forall x_m \exists x_{m+1} \cdots \exists x_n \bigwedge_{j=1}^{k} Z_j^1 \vee Z_j^2 \vee Z_j^3$$

is equivalent to the following membership problem: the source instance is

$$S = \{\mathit{False}(0), \mathit{True}(1), V(0), V(1)\};$$

the target instance is

$$T = \left\{ C(a_1, a_2, a_3, b_1, b_2, b_3) \;\middle|\; \begin{array}{c} a_1, a_2, a_3, b_1, b_2, b_3 \in \{0, 1\}, \\ (a_1 \oplus b_1) \vee (a_2 \oplus b_2) \vee (a_3 \oplus b_3) = 1 \end{array} \right\},$$

where $\oplus$ is the xor operator and $\vee$ is the disjunction operator; and the set Σ contains a single constraint

$$\mathit{False}(f) \wedge \mathit{True}(t) \wedge \bigwedge_{i=\ell+1}^{m} V(x_i) \longrightarrow \bigwedge_{j=1}^{k} C(X_j^1, X_j^2, X_j^3, Y_j^1, Y_j^2, Y_j^3),$$

where $(X_j^p, Y_j^p) = \begin{cases} (x_i, f) & \text{for } Z_j^p = x_i \\ (x_i, t) & \text{for } Z_j^p = \bar{x}_i \end{cases}$, and—against the convention—symbols $x_1, \ldots, x_\ell$ are Skolem functions of arity 0 (constants). To see that the reduction is correct, note that according to the semantics of mappings the three blocks of x_i's are quantified like in the formula, and for a given valuation the target side atoms hold in T if and only if the corresponding clauses in the formula hold. □

B. QUERY ANSWERING

PROPOSITION 5.3. *There exist* $\mathcal{M}_1 \in \mathrm{SM}(\downarrow,\downarrow^+)$, $Q_1 \in \mathbf{CTQ}(\downarrow,\downarrow^+,\sim)$ *and* $\mathcal{M}_2 \in \mathrm{SM}^{\mathrm{nr}}(\downarrow,\rightarrow)$, $Q_2 \in \mathbf{CTQ}(\downarrow,\rightarrow,\sim)$ *such that* $\textsc{certain}_{\mathcal{M}_1}(Q_1)$ *and* $\textsc{certain}_{\mathcal{M}_2}(Q_2)$ *are undecidable.*

PROOF. First, we prove the result for **UCTQ**. We modify the reductions from the proof of Theorem 7.1. There, $\mathcal{M}_1 \in \mathrm{SM}(\downarrow,\downarrow^+), \mathcal{M}_1' \in \mathrm{SM}(\downarrow,\downarrow^+,\neq)$ and $\mathcal{M}_2 \in \mathrm{SM}^{\mathrm{nr}}(\downarrow,\rightarrow), \mathcal{M}_2' \in \mathrm{SM}^{\mathrm{nr}}(\downarrow,\rightarrow,\neq)$ are constructed such that the sets $[\![\mathcal{M}_1]\!] \circ [\![\mathcal{M}_1']\!]$ and $[\![\mathcal{M}_2]\!] \circ [\![\mathcal{M}_2']\!]$ are undecidable. The mappings $\mathcal{M}_1', \mathcal{M}_2'$ have an additional property: the target sides of their dependencies are either single inequality, or $\bot$. Hence, they can be rewritten easily so that all dependencies have $\bot$ on the target side: simply replace $z \neq z'$ on the target side with $\bot$ and add $z = z'$ to the source side. Now, let Q_i be the union of the source side patterns of dependencies in $\mathcal{M}_i'$. It is clear that $(S,T) \in [\![\mathcal{M}_i]\!] \circ [\![\mathcal{M}_i']\!]$ iff $\mathit{certain}_{\mathcal{M}_i}(Q_i, S) = \mathit{false}$. It follows immediately that $\textsc{certain}_{\mathcal{M}_i}(Q_i)$ is undecidable.

Next, we use a technique from [Miklau and Suciu 2004] to carry over the undecidability results to **CTQ**. Let us start from the case with $\rightarrow$. Assume that

$$Q_2 = \bigcup_{j=1}^{k} \exists \bar{x}_j \, (\pi_j, \alpha_j)\,,$$

where $\bar{x}_j$ are pairwise disjoint tuples of variables from $\{y_1, y_2, \ldots, y_n\}$ such that the set of entries of $\bar{x}_j$ coincides with $\mathrm{Var}(\pi_j, \alpha_j)$. Let $\widetilde{\mathcal{M}}_2$ be defined as follows. The source DTD is obtained from source DTD of $\mathcal{M}_2$ by replacing the root symbol r by a fresh label p with attributes $@attr_1, @attr_2, \ldots, @attr_n$ and production $p \to r$. The target DTD is obtained from the target DTD of $\mathcal{M}_2$ by replacing the root symbol r with p and adding productions

$$p \to \sharp\, q^*\, \natural$$
$$q \to \sharp\, r^*\, \natural$$

for fresh labels p, q. The set of dependencies of $\widetilde{\mathcal{M}}_2$ contains

$$p(\bar{y}) \;\longrightarrow\; p\big[\sharp \to \underbrace{\pi_{Q_2} \to \ldots \to \pi_{Q_2}}_{k-1} \to q[\sharp \to r \to \natural] \to \underbrace{\pi_{Q_2} \to \ldots \to \pi_{Q_2}}_{k-1} \to \natural\big] \quad (7)$$

with $\pi_{Q_2} = q\,[\pi_1, \ldots, \pi_k]$ and for each dependency $\varphi \longrightarrow \pi, \alpha$ from $\mathcal{M}_2$ it contains

$$p[\varphi] \;\longrightarrow\; p\big[\sharp \to \underbrace{q \to \ldots \to q}_{k-1} \to q[\pi]\big],\ \alpha\,. \quad (8)$$

Finally, let

$$\widetilde{Q}_2 = \exists \bar{x}_1 \ \ldots\ \exists \bar{x}_k \; p[q/\pi_1 \to \ldots \to q/\pi_k],\ \alpha_1 \wedge \cdots \wedge \alpha_k\,.$$

Let $\bar{a}$ be chosen in such a way that each π_j, α_j is satisfiable in a tree conforming to the target DTD of $\mathcal{M}_2$ under the valuation $\bar{y} = \bar{a}$ (without loss of generality we can assume that each π_j, α_j is satisfiable). Then, for every tree S we have

$$\mathit{certain}_{\mathcal{M}_2}(Q_2, S) = \mathit{false} \iff \mathit{certain}_{\widetilde{\mathcal{M}}_2}\Big(\widetilde{Q}_2, p(\bar{a})[S]\Big) = \mathit{false}\,.$$

Indeed, suppose that T is a solution for S falsifying Q_2. Consider T' defined as

$$p\big[\underbrace{q[T_1,\dots,T_k]\,,\,\dots\,,\,q[T_1,\dots,T_k]}_{k-1}\,,\,q[T]\,,\,\underbrace{q[T_1,\dots,T_k]\,,\,\dots\,,\,q[T_1,\dots,T_k]}_{k-1}\big]$$

where T_j is any tree conforming to the target DTD of $\mathcal{M}_2$ that satisfies π_j, α_j with variables evaluated according to $\bar{y} = \bar{a}$ (such T_j exists by the choice of $\bar{a}$). It is clearly a solution for $p(\bar{a})[S]$ under $\widetilde{\mathcal{M}}_2$. If $T' \models \widetilde{Q}_2$, then for some valuation of $\bar{y}$, $\alpha_1, \dots, \alpha_k$ are satisfied and some π_j is matched in T. That means that $T \models \exists \bar{x}_j\ \pi_j, \alpha_j$, which is a contradiction. Hence, $T' \not\models \widetilde{Q}_2$.

Conversely, assume that T is a solution for $p(\bar{a})[S]$ that falsifies $\widetilde{Q}_2$. By (7), the root of T has exactly $2k-1$ children labelled with q, and the kth child has exactly one child labelled with r. Let T' be the subtree rooted at this child. By (8), T' is a solution for S under $\mathcal{M}$. If T' satisfies π_j, α_j with $\bar{x}_j = \bar{b}$, then, by (7), T satisfies $\widetilde{Q}_2$ with $\bar{x}_j = \bar{b}$ and $\bar{x}_i$ evaluated according to $\bar{y} = \bar{a}$ for $i \neq j$. Hence, $T' \not\models Q_2$.

Thus, $\textsc{certain}_{\mathcal{M}_2}(Q_2)$ reduces to $\textsc{certain}_{\widetilde{\mathcal{M}}_2}(\widetilde{Q}_2)$, which shows that $\textsc{certain}_{\widetilde{\mathcal{M}}_2}(\widetilde{Q}_2)$ is undecidable.

In the case with $\downarrow^+$ the source DTD, like above, is obtained from the source DTD of $\mathcal{M}_2$ by replacing the root symbol r by a fresh label p with attributes $@attr_1, @attr_2, \dots, @attr_n$ and production $p \to r$. The target DTD is obtained from the target DTD of $\mathcal{M}_2$ by replacing the root symbol r with p and adding productions

$$\begin{aligned} p &\to q \\ q &\to (q \,|\, \sharp)\, s \\ s &\to (s \,|\, \sharp)\, r \end{aligned}$$

for fresh labels p, q, s. For the sake of readability we write $\ell[\lambda]/\pi$ for $\ell[\lambda, \pi]$. The dependencies are

$$p(\bar{y}) \;\longrightarrow\; p\underbrace{/\pi_{Q_2}/\dots/\pi_{Q_2}}_{k-1}/q[s/\sharp]\underbrace{/\pi_{Q_2}/\dots/\pi_{Q_2}}_{k-1}/\sharp\,,$$

where $\pi_{Q_2} = q\big[s[\pi_1]/\dots/s[\pi_k]\big]$, and

$$p[\varphi] \;\longrightarrow\; p\underbrace{/q/\dots/q}_{k-1}/q/s/\pi,\ \alpha$$

for each dependency $\varphi \longrightarrow \pi, \alpha$ in $\mathcal{M}_2$. The query is

$$\widetilde{Q}_1 = \exists \bar{x}_1\ \dots\ \exists \bar{x}_q\ p//q[s//\pi_1]/\dots/q[s//\pi_k],\ \alpha_1 \wedge \dots \wedge \alpha_k\,.$$

The rest of the argument is entirely analogous. □

Proposition 5.6. *There exist a schema mapping $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\downarrow)$, a query $Q_1 \in \mathbf{CTQ}(\downarrow, \to, =)$, and a query $Q_2 \in \mathbf{CTQ}(\downarrow, \to^+, =)$ such that both $\textsc{certain}_{\mathcal{M}}(Q_1)$ and $\textsc{certain}_{\mathcal{M}}(Q_2)$ are coNP-complete.*

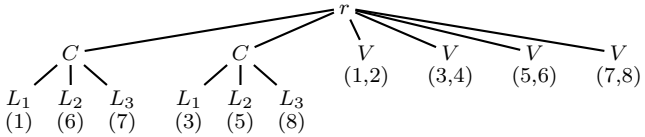
Proof. The coNP upper bound follows from Proposition 5.4. For the lower bound we only give the proof for $\to$; the proof for $\to^+$ can be obtained by replacing $\to$ with $\to^+$ in our argument.

We give an XML schema mapping $\mathcal{M}$ and a Boolean query Q such that 3SAT is reducible to the complement of $\textsc{certain}_{\mathcal{M}}(Q)$, i.e., for each 3SAT instance θ

$$\mathit{certain}_{\mathcal{M}}(Q, T_\theta) \text{ is } \mathit{false} \text{ iff } \theta \text{ is satisfiable,}$$

where T_θ is a tree encoding of θ described below.

Suppose we are given a 3-CNF formula $\theta = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \ell_{ij}$, where ℓ_{ij} is a literal, and in each clause all three literals are different. The tree encoding, T_θ, is best explained on a concrete example. A formula $(x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$ is encoded as



Each V node has two attribute values encoding a variable and its negation with two different values. For example, the node $V(1,2)$ indicates that x_1 is encoded by the data value 1 and $\neg x_1$ by 2. Also for each clause in the formula we have a C node that has three children labeled L_1, L_2, L_3. Each L_i holds the data value encoding the ith literal in the clause. In the example above, the second literal of the first clause is $\neg x_3$ and hence the data value of L_2 under the first C node is 6.

Let us now describe the mapping. In accordance with the encoding, let D_s be

$$\begin{array}{ll} r \to C^* V^* & V\colon @a_1, @a_2 \\ C \to L_1 L_2 L_3 & L_i\colon @b \end{array}$$

where $i = 1, 2, 3$. The target DTD D_t is defined as

$$\begin{array}{ll} r \to C^* V^* & V\colon @a_1, @a_2 \\ C \to L^* & L\colon @b \end{array}$$

The dependencies Σ essentially copy T_θ in the target, but allow the reordering of children under each C node with the use of ‘,’(comma). This reordering corresponds to ‘choosing one literal per clause’ mentioned earlier. Intuitively, a literal is chosen if its copy has more than two following siblings. Since each C node has three L-children with different data values, at least one literal is chosen for each clause.

$$r[C[L_1(x), L_2(y), L_3(z)]] \to r[C[L(x), L(y), L(z)]]$$
$$r[V(x,y)] \to r[V(x,y)]$$

Thus, a solution gives a (partial) valuation satisfying θ, provided that the choices are consistent. This is taken care of by the query: it is true if a variable and its negation are contained among the chosen literals. The query is

$$\exists x \exists y \ \big(r\big[V(x,y), C[L(x) \to L \to L], C[L(y) \to L \to L]\big]\big) \ .$$

Let us see that $\text{certain}_{\mathcal{M}}(Q, T_\theta) = \mathit{false}$ if and only if a 3-CNF formula θ is satisfiable.

($\Rightarrow$) Suppose $\text{certain}_{\mathcal{M}}(Q, T_\theta) = \mathit{false}$. Then there exists a tree T' that is a solution for the tree encoding of θ and falsifies the query. We extract an assignment v from T' as follows. For each fragment matching $r[C[L(x) \to L \to L]]$, v assigns

true to the literal encoded by x (if the value of x does not encode a literal, just ignore it). This assignment is consistent since the query is false over T'. Finally the assignment satisfies θ. The dependency requires that for each clause there is a C node with L nodes storing values encoding the three literals. Since the literals are different, the three nodes are different as well, and at least one of them has two following siblings. Hence, the corresponding literal is true by the assignment.

($\Leftarrow$) Suppose that θ is satisfiable. Assume v is a satisfying assignment. To construct a solution for T_θ that falsifies the query, we simply change the order of L's under each C node so that, for each clause, some literal assigned *true* has at least two following siblings. More specifically, for each tree fragment of the form $r[C[L_1(d_1), L_2(d_2), L_3(d_3)]]$, the corresponding clause has at least one literal that is assigned *true* by v. We choose the data encoding one such literal (we choose the one corresponding to a literal with the smallest index if there are more than one literal to which v assigns *true*). For example, suppose it is d_2. Then we make the tree fragment $r[C[L(d_2) \to L(d_1) \to L(d_3)]]$. After we have processed all the fragments corresponding to clauses, we simply copy all the L nodes in the target. Since v never assigns *true* to a variable and its negation, the query is false over the constructed tree. □

Mappings with fully specified sibling order. There are two possible ways to define the notion of being fully specified with respect to the horizontal ordering. In the more relaxed notion, called $\to^+$*-fully specified*, we insist that for every two subpatterns which start at children of the same node, we know their relative ordering. In the stronger notion, called $\to$*-fully specified*, we completely specify the $\to$ relation among the siblings.

More precisely, $\to^+$-fully specified patterns exclude, in addition to $//\pi$ and wildcard, the ability to take union (i.e., the λ, λ' construct) and are given by:

$$\begin{array}{lcl} \pi & := & \ell(\bar{x})[\mu] \\ \mu & := & \varepsilon \mid \pi \mid \pi \to \mu \mid \pi \to^+ \mu \end{array}$$

The $\to$-fully specified patterns in addition exclude $\to^+$ and are given by:

$$\pi \ := \ \ell(\bar{x})[\mu] \qquad\qquad \mu \ := \ \varepsilon \mid \pi \mid \pi \to \mu$$

For example, $a[b, c]$ is neither $\to$-fully specified nor $\to^+$-fully specified; $a[b \to^+ c]$ is $\to^+$-fully specified, but not $\to$-fully specified, and $a[b \to c \to d]$ is $\to$-fully specified.

We say that a mapping is $\to$-fully specified if each dependency has a $\to$-fully specified pattern on the target side; analogously for $\to^+$-fully specified.

Proposition B.1.

(1) *There exist a $\to$-fully specified schema mapping $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\downarrow, \to)$ and a query $Q \in \mathbf{CTQ}(\downarrow, \to, =)$ such that* certain(Q) *is coNP-complete.*

(2) *There exist a $\to^+$-fully specified schema mapping $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\downarrow, \to^+)$ and a query Q from the class $\mathbf{CTQ}(\downarrow, \to^+, =)$ such that* certain(Q) *is coNP-complete.*

Proof. (1) As in the previous proof, we will provide an XML schema mapping $\mathcal{M} = (D_s, D_t, \Sigma)$ and a query Q such that we can reduce 3SAT to the complement

of certain$_{\mathcal{M}}(Q, T_\theta)$, where T_θ is the same encoding of formulae as in the previous proof.

The idea of the reduction is the following: we transform a 3CNF formula θ into a source tree T_θ. The mapping is defined so that a solution of T_θ corresponds to a selection of (at least) one literal for each clause in the formula. Finally we provide a query that is true when such a selection contains a variable and its negation. Thus the existence of a solution falsifying the query means the existence of a well-defined (partial) assignment that satisfies the formula θ. The difference from the previous proof is how we "choose" literals.

The source DTD D_s is just like before, and the target DTD D_t is again almost the same as D_s, except that it has M_i nodes. With these extra nodes, we 'choose' a literal from each clause. Intuitively we select L_i's that are 'two step to the right of M_1'.

$$\begin{array}{ll} r \to C^* V^* & V : @a_1, @a_2 \\ C \to M_1 M_2 ? M_3 ? L_1 L_2 L_3 & L_1, L_2, L_3 : @b. \end{array}$$

The dependencies are simply copying precisely the source to the target.

$$r[C[L_1(x) \to L_2(y) \to L_3(z)]] \longrightarrow r[C[L_1(x) \to L_2(y) \to L_3(z)]]$$
$$r[V(x,y)] \longrightarrow r[V(x,y)]$$

The query Q is true if a variable and its negation are both 'chosen', just as in the previous proof:

$$\exists x \exists y (r[V(x,y), C[M_1 \to \_ \to \_ \to \_(x)], C[M_1 \to \_ \to \_ \to \_(x)])$$

In order to formally prove the correctness of the reduction, we have to prove that certain(Q, T_θ) is *false* iff θ is satisfiable.

($\Rightarrow$) To prove the left-to-right implication, suppose certain(Q, T_θ) is *false*, which means there is a target tree T' that is a solution for T and falsifies Q. Then we define the truth assignment from T' as follows: for each C node corresponding to a clause, find the node below it that is two steps away from M_1 (whose element type must be one of L_i), say, $L_i(p)$. By the first dependency, p encodes a literal. If p encodes x_j (resp. $\neg x_j$), then assign *true* (resp. *false*) to x_j. By the target DTD each clause contains a true literal. Furthermore, since the query is false over T', the truth assignment does not assign *true* to a variable and its negation, hence the assignment is well-defined.

($\Leftarrow$) For the other direction, suppose the formula is satisfiable with an assignment v. We shall construct a solution of the mapping for T_θ base on v. First we copy every L node. Next for each fragment of the form $r[C[L_1(d_1) \to L_2(d_2) \to L_3(d_3)]]$, we will copy it and put M_1 to the left of L_1 with possibly putting M_2 and M_3 in-between. If v assigns *true* to the first literal in the corresponding clause, then we put both M_2 and M_3 between M_1 and L_1 (to make L_1 "two steps to the left of M_1); otherwise if it is the second literal that is assigned *true* by v, then we put M_2 alone; otherwise we put neither M_2 nor M_3. Since a truth assignment does not assign *true* to both a variable and its negation, it will not happen that the values paired in V, i.e. those encoding a variable and its negation, are both two steps away from M_1. Thus the query is false in the constructed tree, as desired.

(2) We provide $\mathcal{M} = (D_s, D_t, \Sigma)$ and a query Q to which 3SAT is reducible to $\text{CERTAIN}_{\mathcal{M}}(Q)$.

The idea of the reduction is the following: we transform a 3CNF formula θ into a source tree T_θ. The mapping is defined so that a solution of T_θ corresponds to a selection of (at least) one literal for each clause in the formula. Finally we provide a query that is true when such a selection contains a variable and its negation. Thus the existence of a solution falsifying the query means the existence of a well-defined (partial) assignment that satisfies the formula θ. The difference from the previous proofs is how we "choose" literals.

D_s is quite similar as before:

$$\begin{array}{lll} r \to C^*V^* & C\colon @a_1 & V\colon @a_2, @a_3 \\ C \to L^* & L\colon @b & \end{array}$$

Note that the subscript in L is dropped and the C-nodes now store one attribute. We encode a given 3CNF formula θ as T_θ, by dropping the subscript in the previous encoding and inserting a fresh data value in the attribute of each C-node.

The target DTD is the following very simple one.

$$r \to A^*V^* \qquad A\colon @b_1, @b_2 \qquad V\colon @a_2, @a_3$$

The constraint is the following: it "flattens" the structure using two attributes in A: the first indicates a clause and the second encodes a literal. Formally the dependencies are:

$$\begin{aligned} r/C(x)/L(y) &\longrightarrow r/A(x,y) \\ r/V(x,y) &\longrightarrow r/V(x,y) \end{aligned}$$

In a target tree, we choose a literal that has at least two following siblings in each clause (i.e., with the same first attribute value).

Finally, let us define the query Q. Like in the previous reductions, Q is true when the set of selected literals contains a variable and its negation.

$$\begin{aligned} \exists xyuvz_1z_2z_3z_4\ r\big[V(x,y),\, A(u,x) \to^+ A(u,z_1) \to^+ A(u,z_2), \\ A(v,y) \to^+ A(v,z_3) \to^+ A(v,z_4)\big]\,. \end{aligned}$$

We need to show that $\text{certain}_{\mathcal{M}}(Q, T_\theta)$ is *false* iff θ is satisfiable.

($\Rightarrow$) Suppose $\text{certain}(Q, T_\theta)$ is *false*. We have a solution T' for T_θ that falsifies the query Q above. We extract a truth assignment as follows. For any fragment matching $r[A(d,d_1) \to^+ A(d,d_2) \to^+ A(d,d_3)]$, where each d_i is a data value encoding a literal (i.e, appears in some V node in the source) and d is a data value encoding a clause (i.e., appears in C in the source), the variable encoded by d_1 is assigned *true*. Since T' falsifies Q, our assignment is consistent. Also it satisfies θ. Each clause $C(d)$ in the source has three L-labelled children storing different data values (in each clause all three literals are different), so we have at least three nodes of the form $A(d,d')$ in the target. Hence at least one literal in the clause will be assigned *true*.

($\Leftarrow$) Assume θ is satisfiable, with a satisfying assignment v. We need to construct a solution of the mapping for the source tree T_θ. First we copy all the V nodes from the source. For each fragment of the form $r[C(d)[L_1(d_1), L_2(d_2), L_3(d_3)]]$, a

solution must contain the three fragments $r[A(d,d_1)]$, $r[A(d,d_2)]$, and $r[A(d,d_2)]$ in some order. The order of the above three fragments is decided according to which literal is assigned *true* by v, as follows. If it is the first literal, we put $r[A(d,d_1) \to A(d,d_2) \to A(d,d_3)]$; otherwise if it is the second literal, we put $r[A(d,d_2) \to A(d,d_1) \to A(d,d_3)]$; otherwise we put $r[A(d,d_3) \to A(d,d_1) \to A(d,d_2)]$ Repeating this for each fragment encoding a clause, we obtain a tree that conforms to the target DTD and is a solution for T_θ. Since v does not assign *true* to both a variable and its negation, the constructed tree falsifies the query.

Note that we cannot replace $\to^+$ with $\to$ here because the above constraints do not guarantee all the a's with the same first coordinate (identifier for clause) appear consecutively in the target.

Note also that this gives another proof of coNP-hardness of $\mathbf{CTQ}(\downarrow, \to^+, =)$ (Proposition 5.6). □

LEMMA 5.13. *Let D be a nested relational DTD and let $\varphi(\bar{x})$ be a fully specified non-bounding tree pattern. Then, $\mathrm{mrg}_D\varphi$ is fully specified and non-bounding, and for every T consistent with D and every $F, \bar{a}$*

(1) $(T,F) \models \varphi(\bar{a})$ *implies* $(T,F) \models (\mathrm{mrg}_D\varphi)(\bar{a})$*;*

(2) if $(T,F) \models (\mathrm{mrg}_D\varphi)(\bar{a})$ *and the witnessing homomorphism is injective, then* $(T,F) \models \varphi(\bar{a})$*;*

(3) if $\mathrm{mrg}_D\varphi \neq \bot$ *and F is admissible,* $(\mathrm{mrg}_D\varphi)(\bar{a})$ *admits an injective homomorphism into a tree consistent with D.*

In particular, $\varphi(\bar{a})$ is satisfiable w.r.t. D iff $(\mathrm{mrg}_D\varphi)(\bar{a})$ is satisfiable w.r.t. D, and $\mathrm{mrg}_D\varphi = \bot$ iff φ° is not satisfiable in a tree consistent with D.

PROOF. Since $\mathrm{mrg}_D(\pi,\alpha) = (\pi', \alpha' \wedge \alpha)$ with $(\pi',\alpha') = \mathrm{mrg}_D\pi$, it is enough to prove the lemma for $\varphi = \pi$. The algorithm never introduces // or _, so $\mathrm{mrg}_D\pi$ is fully specified. The only place where a bounding sequence could possibly be introduced is when $\pi_{-j} \to \ldots \to \pi_k$ and $\pi'_{-j'} \to \ldots \to \pi'_{k'}$ are merged in step (3) of the merging algorithm. Let the result of the merging be $\pi''_{-j''} \to \ldots \to \pi''_{k''}$ with $j'' = \max(j,j')$, $k'' = \max(k,k')$. Note that either π''_0 or $\pi''_0 \to \pi''_1$ is a critical subsegment. In either case, each bounding sub-segment in $\pi''_{-j''} \to \ldots \to \pi''_{k''}$ must be contained in $\pi''_{-j''} \to \ldots \to \pi''_1$ or in $\pi''_0 \to \ldots \to \pi''_{k''}$. But then there would be a bounding sub-segment in one of the original patterns, which we assumed not to happen. Thus, $\mathrm{mrg}_D\pi$ is non-bounding.

Let us move to the main claim of the lemma. Since $\mathrm{mrg}_D(\pi(\bar{t})) = (\mathrm{mrg}_D\pi)(\bar{t})$, we can work with $\mathrm{mrg}_D(\pi(\bar{a}))$ instead of $(\mathrm{mrg}_D\pi)(\bar{a})$ and assume that the merging procedure works directly on $\pi(\bar{a})$. Let us fix an interpretation F. By induction on the depth of π we prove the following:

(1') for each homomorphism $h\colon \pi(\bar{a}) \to T$ witnessing that $(T,F) \models \pi(\bar{a})$ for some T consistent with D, if some two nodes of $\pi(\bar{a})$ are merged in $\mathrm{mrg}_D(\pi(\bar{a}))$ then their images under h coincide, and the function $\mathrm{mrg}_D(\pi(\bar{a})) \to T$ thus induced by h is a homomorphism witnessing that $(T,F) \models \mathrm{mrg}_D(\pi(\bar{a}))$;

(2') for each injective homomorphism $h\colon \mathrm{mrg}(\pi(\bar{a})) \to T$ witnessing that $(T,F) \models \mathrm{mrg}_D(\pi(\bar{a}))$ for some T consistent with D, the function $\pi(\bar{a}) \to T$ induced by h is a homomorphism;

(3) if $\mathrm{mrg}_D\varphi \neq \bot$ *and* F *is admissible,* $(\mathrm{mrg}_D\varphi)(\bar{a})$ *admits an injective homomorphism into a tree consistent with* D*.*

If $\pi(\bar{a}) = \sigma(\bar{t})$, then $\mathrm{mrg}_D(\pi(\bar{a})) = \sigma(\bar{t})$ and the whole claim is obvious. Let us assume that $\pi(\bar{a}) = \sigma(\bar{t})[\mu_1, \ldots, \mu_m]$.

(1') Let $h\colon \pi(\bar{a}) \to T$ be a homomorphism witnessing that $(T, F) \models \pi(\bar{a})$. It is straightforward to see that the sequence of labels of the root's children in T is a word generated by the regular expression ω, computed in step (1) of the merging algorithm. Consequently, $\mathit{head}(\mu_i)$ is satisfiable in a word generated by ω for all i, and step (1) does not return $\bot$. Removing $\to^+$ in step (2) does not affect the homomorphism h. Let us look at step (3). Take segments $\pi_{-j} \to \ldots \to \pi_k$ and $\pi'_{-j'} \to \ldots \to \pi'_{k'}$ such that either π_0, π'_0 or $\pi_0 \to \pi_1, \pi'_0 \to \pi'_1$ are critical subsegments with identical heads. It follows immediately that the roots of π_0 and π'_0 are mapped by h to the same node in T: in the first case there can be just one node with an appropriate label, and in the second case there are only two subsequent siblings with appropriate labels. Hence, h agrees on the roots of π_i and π'_i for $i = -j', \ldots, k$. From this it follows that $\tau_i = \tau'_i$ for $i = -j', \ldots, k$, and the algorithm merges the two segments. It also follows that the introduced equalities are satisfied and that the arguments of the recursive calls of mrg_D are satisfied in the corresponding subtrees of T, witnessed by h. By the inductive hypothesis, the recursive calls do not return $\bot$, and h induces homomorphisms witnessing that the returned patterns are satisfied in these subtrees as well. Repeating this reasoning for all merges performed in step (3), we end up with an induced homomorphism witnessing that $(T, F) \models \mathrm{mrg}_D(\pi(\bar{a}))$.

(2') Any homomorphism $h\colon \mathrm{mrg}_D(\pi(\bar{a})) \to T$ naturally induces a function $\hat{h}\colon \pi(\bar{a}) \to T$. The function $\hat{h}$ clearly satisfies all the conditions to be a homomorphism, except preserving relations $\to^+$: in step (2) we remove some occurrences of $\to^+$ from $\pi(\bar{a})$. We claim that each removed $\to^+$ is imposed by the structure of D—provided that h is injective.

Suppose that $\to^+$ is removed from $\kappa \to^+ \kappa'$, a subsequence of μ_i. By step (1) of the algorithm, $\mathit{head}(\mu_i)$ is satisfiable in a word generated by ω, and so is $\kappa \to^+ \kappa'$. Recall that $\omega = \hat{\sigma}_1 \ldots \hat{\sigma}_k$ for distinct σ_i. If $\mathit{head}(\kappa)$ and $\mathit{head}(\kappa')$ share no labels, the missing $\to^+$ is always satisfied when κ, κ' are matched in a word generated by ω. If they do share a label, from the shape of ω it follows that $\mathit{head}(\kappa) = \kappa_0 \to \sigma_i \to \sigma_i \to \ldots \to \sigma_i$ and $\mathit{head}(\kappa') = \sigma_i \to \sigma_i \to \ldots \to \sigma_i \to \kappa'_0$, where κ_0 and κ'_0 share no labels. Since $\to^+$ between κ and κ' is removed, either κ_0 or κ'_0 contains a label different from σ_i. Hence, if $\mathit{head}(\kappa)$ and $\mathit{head}(\kappa')$ are matched disjointly in a word generated by ω, then the missing $\to^+$ is also satisfied. Note also that in either case κ and κ' cannot contain critical subsegment with identical heads, so they will not be merged in $\mathrm{mrg}_D(\pi(\bar{a}))$.

Now, assume that $h\colon \mathrm{mrg}(\pi(\bar{a})) \to T$ is an injective homomorphism witnessing that $(T, F) \models \mathrm{mrg}_D(\pi(\bar{a}))$. Then, $\hat{h}$ matches the heads of all segments of μ_i in the root's children. Examining step (1), we see that the word w given by the sequence of labels of the root's children can be generated by ω. If $\to^+$ is removed from $\kappa \to^+ \kappa'$, we know that κ and κ' are not merged in $\mathrm{mrg}_D(\pi(\bar{a}))$. Consequently, $\hat{h}$ matches their heads disjointly in w and $\to^+$ is respected. The claim follows by the induction hypothesis.

(3) Assume that $\mathrm{mrg}_D(\pi(\bar{a})) \neq \bot$ and F is admissible. Let $\omega = \hat{\sigma}_1 \dots \hat{\sigma}_k$ be the regular expression computed in the step (1). Since $\pi(\bar{a})$ is non-bounding, and each $\mathit{head}(\mu_i)$ is satisfiable in a word generated by ω, after step (2) we have a collection of sequences of two kinds: segments with critical sub-segments, with heads of the forms

$$\underbrace{\sigma_p \to \ldots \to \sigma_p}_{\hat{\sigma}_p=\sigma_p^* \text{ or } \hat{\sigma}_p=\sigma_p^+} \to \underbrace{\sigma_{p+1} \to \sigma_{p+2} \to \ldots \to \sigma_{q-1}}_{\hat{\sigma}_j=\sigma_j \text{ for } p<j<q} \to \underbrace{\sigma_q \to \ldots \to \sigma_q}_{\hat{\sigma}_q=\sigma_q^* \text{ or } \hat{\sigma}_q=\sigma_q^+},$$

$$\underbrace{\sigma_p \to \sigma_{p+1} \to \ldots \to \sigma_{q-1}}_{\hat{\sigma}_j=\sigma_j \text{ for } p\leq j<q} \to \underbrace{\sigma_q \to \ldots \to \sigma_q}_{\hat{\sigma}_q=\sigma_q^* \text{ or } \hat{\sigma}_q=\sigma_q^+},$$

$$\underbrace{\sigma_p \to \ldots \to \sigma_p}_{\hat{\sigma}_p=\sigma_p^* \text{ or } \hat{\sigma}_p=\sigma_p^+} \to \underbrace{\sigma_{p+1} \to \sigma_{p+2} \to \ldots \to \sigma_q}_{\hat{\sigma}_j=\sigma_j \text{ for } p<j\leq q},$$

$$\underbrace{\sigma_p \to \sigma_{p+1} \to \ldots \to \sigma_q}_{\hat{\sigma}_j=\sigma_j \text{ for } p\leq j\leq q},$$

$$\underbrace{\sigma_p \to \ldots \to \sigma_p}_{\hat{\sigma}_p=\sigma_p^* \text{ or } \hat{\sigma}_p=\sigma_p^+} \to \underbrace{\sigma_q \to \ldots \to \sigma_q}_{\hat{\sigma}_q=\sigma_q^* \text{ or } \hat{\sigma}_q=\sigma_q^+} \quad \text{with } q = p+1$$

and sequences without critical sub-segments, with heads of the form

$$(\sigma_p \to \ldots \to \sigma_p) \to^+ (\sigma_p \to \ldots \to \sigma_p) \to^+ \ldots \to^+ (\sigma_p \to \ldots \to \sigma_p)$$

with $\hat{\sigma}_p = \sigma_p^*$ or $\hat{\sigma}_p = \sigma_p^+$.

This property is not affected by the merging in step (3): only sequences of the first kind get merged, and the result is also of the first kind (we assumed $\bot$ is never returned). Let $\mathrm{mrg}_D(\pi(\bar{s})) = \sigma(\bar{t})[\kappa_1, \dots, \kappa_r], \beta$. We know that no two sequences κ_i have critical sub-segments with identical heads. It follows that for $i \neq i'$, $\mathit{head}(\kappa_i)$ and $\mathit{head}(\kappa_{i'})$ share at most one label σ_j and moreover $\hat{\sigma}_j = \sigma_j^*$ or $\hat{\sigma}_j = \sigma_j^+$. Together with the special form of the heads described above, this guarantees that we can order κ_i so that there exist $1 \leq p_1 \leq q_1 \leq p_2 \leq q_2 \leq \cdots \leq p_r \leq q_r \leq k$ such that $\mathit{head}(\kappa_i)$ uses exactly labels $\sigma_{p_i}, \sigma_{p_i+1}, \dots, \sigma_{q_i}$ and moreover if $q_i = p_{i+1}$ then $\hat{\sigma}_{q_i} = (\sigma_{q_i})^*$ or $\hat{\sigma}_{q_i} = (\sigma_{q_i})^+$. Let

$$w = w_0\, \mathit{word}(\kappa_1)\, w_1\, \mathit{word}(\kappa_2)\, w_2\, \cdots\, \mathit{word}(\kappa_k)\, w_k$$

where $\mathit{word}(\kappa_i)$ is the word obtained from $\mathit{head}(\kappa_2)$ by skipping all occurrences of $\to$ and $\to^+$, and w_i is the shortest word generated by $\hat{\sigma}_{q_i+1}\hat{\sigma}_{q_i+2}\cdots\hat{\sigma}_{p_{i+1}-1}$ ($q_0 = -1$, $p_{r+1} = k+1$). Obviously, the heads of κ_i can be matched pairwise disjointly in w and w is generated by ω. By the construction of ω, w is also generated by the regular expression in the production for σ in D. To extend this to an injective match of $\mathrm{mrg}_D(\pi(\bar{a}))$ in a tree consistent with D, we invoke the induction hypothesis for the arguments of the recursive calls made in step (3). This concludes the inductive proof.

The additional claims follow easily from the main claim. By part (1), if $\varphi(\bar{a})$ is satisfiable, so is $(\mathrm{mrg}_D\varphi)(\bar{a})$. Conversely, if $(\mathrm{mrg}_D\varphi)(\bar{a})$ is satisfiable, then $\mathrm{mrg}_D\varphi \neq \bot$, $\mathrm{mrg}_D\varphi$ has label r in its root node, and F is admissible. By part (3),

$\mathrm{mrg}_D\varphi$ can be matched injectively in a tree T conforming to D. By part (2), φ is satisfiable.

Let π be a pattern that uses no function symbols, constants, nor repetitions of variables, such that $\varphi = \pi(\bar{t}) \wedge \alpha$. Clearly, φ° is satisfiable in a tree consistent with D iff so is π. Like in the previous paragraph, π is satisfiable in a tree consistent with D iff so is $\mathrm{mrg}_D\pi$. Since π uses no function symbols or constants, by part (3), $\mathrm{mrg}_D\pi$ is satisfiable in a tree consistent with D iff $\mathrm{mrg}_D\pi \neq \bot$. Finally, by equation (4) on page 27, $\mathrm{mrg}_D\pi \neq \bot$ iff $\mathrm{mrg}_D\varphi = (\mathrm{mrg}_D\pi)(\bar{t}) \neq \bot$. □

C. CONSISTENCY OF SCHEMA MAPPINGS

LEMMA 6.3. *For every DTD D there exists a polynomial tree automaton recognizing* $\{T \mid T \models D\}$.

PROOF. Let $\mathcal{A}_a = (\Gamma, Q_a, i_a, F_a, \delta_a)$ be the NFA equivalent to the regular expression in the production for a in D, obtained by the classical linear translation. Without loss of generality we assume that all Q_a's are disjoint.

The automaton recognizing $\{T \mid T \models D\}$ simply runs the $\mathcal{A}_a$ on the sequence of children of each a-node in the tree, with a ranging over the set of element types Γ. More precisely, the state space is $Q = \bigcup_{a\in\Gamma} Q_a$, with $I = \bigcup_{a\in\Gamma} F_a$ and $F = \{i_r\}$ where r is the root symbol of the DTD. The transition relation δ consists of tuples (q, b, i_b, q') such that $(q, b, q') \in \delta_a$ for some $a \in \Gamma$. □

PROPOSITION 6.5. *Over nested relational DTDs both* $\textsc{Cons}(\Downarrow, \Rightarrow)$ *and* $\textsc{Cons}(\Downarrow, \rightarrow)$ *are* PSPACE*-complete.*

PROOF. To solve the consistency problem in PSPACE we follow the approach taken in the proof of Theorem 6.2. Like before, we reduce consistency of the mapping

$$\big(D_s, D_t, \{\varphi_i \longrightarrow \psi_i \mid i = 1, 2, \ldots, n\}\big)$$

to non-emptiness of the automata

$$\mathcal{A}_s = \mathcal{A}_{D_s} \times \prod_{j\notin I} \overline{\mathcal{A}(\varphi_j)} \qquad \text{and} \qquad \mathcal{A}_t = \mathcal{A}_{D_t} \times \prod_{j\in I} \mathcal{A}(\psi_j)$$

for some subset $I \subseteq \{1, 2, \ldots, n\}$ (since we are working in PSPACE, we simply guess I). But this time, instead of computing the automata and then deciding non-emptiness, we will be generating them on-the-fly during the non-emptiness check, using the full power of Lemma 6.4.

We also rely on the fact that testing non-emptiness of an automaton $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ on trees of depth d can be done in space polynomial in $d \cdot \log \|\mathcal{A}\|$. Let $\mathrm{NonEmpty}(q, d)$ check if there exist $a_1, a_2, \ldots, a_n \in \Gamma$, $q_0, q_1, \ldots, q_n \in Q$ and $p_1, p_2, \ldots, p_n \in Q$, $n \leq |Q|$, such that $q_0 = q$, $q_n \in I$, $(q_{i-1}, a_i, p_i, q_i) \in \delta$ and $\mathrm{NonEmpty}(p_i, d-1)$ for all $1 \leq i \leq n$. This can be done by guessing the subsequent a_i, p_i, q_i while i stays below $|Q|$. At every step we refer to the automaton to check if the guessed values are consistent with δ, and make a recursive call. $\mathrm{NonEmpty}(q, 0)$ simply checks if $q \in I$. Local memory at each call is $\mathcal{O}(\|\mathcal{A}\| + \log d)$ and the depth of the recursion is d, hence the overall space complexity is $\mathcal{O}(d \cdot \log \|\mathcal{A}\| + d \cdot \log d)$.

To check non-emptiness of $\mathcal{A}$ on trees of depth d simply run NonEmpty$(q, d-1)$ for a guessed state q such that (p, a, q, p') for some $p \in F$, $a \in \Sigma$ and $p' \in I$.

In order to decide consistency, we run the above non-emptiness test for the product automata $\mathcal{A}_s$ and $\mathcal{A}_t$ and d equal to the depth of the corresponding schema (can be bounded by the number of element types). The non-emptiness algorithm guesses labels and states of the product automaton and whenever it needs to refer to the transition relation, we simply compute it from scratch using the PSPACE-transducer given by Lemma 6.4. Instead of outputting the whole automaton, which would take exponential space, we simply enumerate the transition relation, tuple by tuple, within polynomial space. As $\log \|\mathcal{A}_s\|$ and $\log \|\mathcal{A}_t\|$ are polynomial in the size of the mapping, we obtain a PSPACE algorithm.

Let us now turn to the lower bound. We show PSPACE-hardness of $\textsc{Cons}(\Downarrow, \rightarrow)$ using reduction from Q3SAT. Suppose we are given a formula $Q_1 x_1 \cdots Q_n x_n\, C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where $Q_i \in \{\forall, \exists\}$ and each conjunct C_i consists of 3 atomic disjuncts, e.g., $(x_1 \vee \bar{x}_3 \vee x_7)$.

Define the source DTD D_s over alphabet $\{r, \sharp, \natural, t_1, t_2, \ldots, t_n, f_1, f_2, \ldots, f_n\}$ as

$$\begin{array}{rl}
r \to \sharp t_1 f_1 \natural & \text{if } Q_1 = \forall \\
r \to \sharp t_1 ? f_1 ? \natural & \text{if } Q_1 = \exists \\
t_i, f_i \to \sharp t_{i+1} f_{i+1} \natural & \text{for all } i < n \text{ such that } Q_{i+1} = \forall \\
t_i, f_i \to \sharp t_{i+1} ? f_{i+1} ? \natural & \text{for all } i < n \text{ such that } Q_{i+1} = \exists \\
t_n, f_n \to \varepsilon &
\end{array}$$

The target DTD D_t is simply $r \to \varepsilon$. Note that both DTDs are nested-relational.

Intuitively, t_i means that x_i is assigned *true*, and f_i means it is assigned *false*. For universally quantified variables the DTD requires branching, which corresponds to two values of the variable. For existentially quantified variables we have to choose one truth value, but it does not prevent us from choosing none or both. This will be taken care of by the constraints.

The key observation is that in the presence of sibling order we can enforce the existence of a node. Consider the following constraints:

$$\begin{array}{l}
//_[\sharp \to \natural] \longrightarrow \bot\,, \\
//_[t_i, f_i] \longrightarrow \bot \quad \text{for all } i \le n \text{ such that } Q_i = \exists\,.
\end{array}$$

The first one says that at least one of t_i and f_i appear. The second one says that if $Q_i = \exists$, at most one of t_i and f_i appear.

Finally, for each C_i we add a constraint enforcing that it is satisfied, e.g., for a conjunct $(x_i \vee \bar{x}_j \vee x_k)$, we add

$$//f_i//t_j//f_k \longrightarrow \bot\,.$$

It is straightforward to see that consistency of this mapping is equivalent to satisfiability of the given Q3SAT formula. □

PROPOSITION 6.8. *For schema mappings from* $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, \sim, \mathsf{Fun})$ *which are fully specified and do not use* $\neq$ *on the source side,* CONS *is* EXPTIME*-complete. If either restriction is lifted, the problem becomes* NEXPTIME*-hard.*

PROOF. (1) As usual, assume that all variables are used exactly once in source side tree patterns. By the monotonicity of patterns, it is enough to consider trees having as few nodes as possible. Thus, if a node is not enforced by the DTD, we will not put it in the tree. Hence, we may safely remove from the productions all expressions of the form σ^*, $\sigma?$, and replace σ^+ with σ.

After this modification, there is exactly one tree conforming to the source DTD – up to the values stored in the attributes. Let us call this tree with missing data values T. Now, we need to fill in those values. More precisely, we need the right pattern of equalities between the values in different nodes. We reduce this problem to the solution building problem.

Let us associate a constant c_v with each node v of T. Replace each dependency in $\mathcal{M}$ with a set of dependencies obtained as follows: for each possible homomorphism from the source side pattern to T produce a new dependency by removing the pattern from the source side (keep the equalities) and replacing each variable mapped to v by the constant c_v. Replace the source DTD with $r \to \varepsilon$, and call the new mapping $\mathcal{M}'$. As all variables were used in the source side patterns, the new dependencies contain no variables at all. Clearly, $\mathcal{M}$ is consistent iff $\mathcal{M}'$ is consistent. But $\mathcal{M}'$ is consistent iff the trivial tree r has a solution. Thus we can use the solution building algorithm from Theorem 5.15 for the source tree r and the mapping $\mathcal{M}'$. Let us examine its complexity. As T is single exponential in the size of the source DTD and each original dependency from $\mathcal{M}$ was replaced with at most $|T|^{\|\mathcal{M}\|}$ new dependencies, the number of dependencies in $\mathcal{M}'$ is single exponential as well. The dependencies use no variables and the target DTDs is polynomial. Hence, the whole algorithm runs in EXPTIME.

(2) To obtain the EXPTIME lower bound for each deterministic Turing machine M and each n we construct a mapping $\mathcal{M}$ that is consistent if and only if M reaches an accepting state in at most 2^n steps starting from the empty input word. The construction is a modification of the one in Theorem 6.7. We are going to store the run of the machine in the source tree. The target schema will be trivial and the target sides of dependencies will be conjunctions of equalities. Let M be a deterministic Turing machine with the tape alphabet A including the blank symbol $\flat$, states $q_0, q_1, \ldots, q_f$, the initial state q_0, and the transition function δ. W.l.o.g. we assume that q_f is the only final accepting state and that the machine moves the head back to the first cell of the tape before it terminates. For technical reasons it is also convenient to assume that the machine loops in the accepting state as soon as it is reached.

The source DTD is given as

$$\begin{aligned}
r &\to a_1\, b_1\, q_0\, q_1\, \ldots\, q_f\, \natural\, \tau_1\, \tau_2\, \cdots\, \tau_m \\
a_j, b_j &\to a_{j+1}\, b_{j+1} && 1 \le i < n \\
a_n, b_n &\to 0_1\, 1_1 \\
0_i, 1_i &\to 0_{i+1}\, 1_{i+1} && 1 \le i < n \\
0_n, 1_n &\to \mathit{cell} \\
\mathit{cell}&\colon @\mathit{state}\ @\mathit{symbol} \\
0_i, 1_i, \mathit{zero}, \mathit{one}&\colon @\mathit{attr} && 1 \le i \le n \\
\natural, q_i, \tau_j&\colon @\mathit{attr} && 0 \le i \le f, 1 \le j \le m
\end{aligned}$$

where the set of decorated tape symbols $\hat{A} = \{s, s^{\triangleright}, s^{\triangleleft} \mid s \in A\}$ consists of symbols $\tau_1, \tau_2, \ldots, \tau_m$. The subtrees rooted in nodes labeled a_n or b_n represent subsequent configurations of M. Under each such node we store the content of the tape in the leaves of the exponential subtree consisting of nodes with labels $0_i, 1_i$. The address of the cell is the sequence of labels on the path leading to the leaf. We will make sure that for each node on the path, its label is mirrored in the data value it stores. For each cell we remember the tape symbol and the state (or the information that the head is elsewhere, encoded by $\natural$).

The target DTD is just

$$\begin{aligned}
r &\to q_0\, q_1\, \ldots\, q_f\, \natural\, \tau_1\, \tau_2\, \cdots\, \tau_m\, \mathit{zero}\, \mathit{one} \\
\mathit{zero}, \mathit{one}, \natural, q_i, \tau_j&\colon @\mathit{attr} && 0 \le i \le f, 1 \le j \le m\,.
\end{aligned}$$

Let us now describe the dependencies. First we enforce that each each state and each tape symbol is encoded with a unique data value. This is done by means of a set of dependencies of the form

$$r\big[\pi(x), \pi'(x)\big] \longrightarrow \bot$$

where $\pi(x)$ and $\pi'(x)$ range over pairs of different patterns from among $q_i(x)$, $\natural(x)$, $\tau_j(x)$.

We make sure that the data values reflect labels in the 0_i and 1_i nodes by introducing the following dependencies for $i = 1, 2, \ldots, n$ (the last dependency enforces that the values encoding directions are different)

$$\begin{aligned}
r//0_i(x) &\longrightarrow r/\mathit{zero}(x)\,, \\
r//1_i(x) &\longrightarrow r/\mathit{one}(x)\,, \\
r[//0_1(x), //1_1(x)] &\longrightarrow \bot\,.
\end{aligned}$$

To ensure that states and tape symbols are encoded consistently we add

$$\begin{aligned}
r\big[q_0(x_0), q_1(x_1), \ldots, q_f(x_f), \natural(y)\big] &\longrightarrow r\big[q_0(x_0), q_1(x_1), \ldots, q_f(x_f), \natural(y)\big]\,, \\
r\big[\tau_1(z_1), \tau_2(z_2), \ldots, \tau_m(z_m)\big] &\longrightarrow r\big[\tau_1(z_1), \tau_2(z_2), \ldots, \tau_m(z_m)\big].
\end{aligned}$$

Correctness of the computation tree is enforced by the dependencies. The initial

configuration is checked by

$$\begin{aligned}
r/a_1/a_2/\ldots/a_n/0_1/0_2/\ldots/0_n/\mathit{cell}(u,v) &\longrightarrow r\big[q_0(u), \flat^{\triangleright}(v)\big]\,,\\
r/a_1/a_2/\ldots/a_n/0_1/0_2/\ldots/0_j/1_{j+1}//\mathit{cell}(u,v) &\longrightarrow r\big[\natural(u), \flat(v)\big]\,,\\
r/a_1/a_2/\ldots/a_n/1_1/1_2/\ldots/1_j/0_{j+1}//\mathit{cell}(u,v) &\longrightarrow r\big[\natural(u), \flat(v)\big]\,,\\
r/a_1/a_2/\ldots/a_n/1_1/1_2/\ldots/1_n/\mathit{cell}(u,v) &\longrightarrow r\big[\natural(u), \flat^{\triangleleft}(v)\big]\,,
\end{aligned}$$

where $j = 2, 3, \ldots, n-1$; transitions are checked by

$$r\begin{bmatrix}\mathit{Step}(u_1,u_2,\ldots,u_6,v_1,v_2,\ldots,v_6),\\ p_1(u_1),p_2(u_2),p_3(u_3),\sigma_1(v_1),\sigma_2(v_2),\sigma_3(v_3)\end{bmatrix} \longrightarrow r\begin{bmatrix}p_4(u_4),p_5(u_5),p_6(u_6),\\ \sigma_4(v_4),\sigma_5(v_5),\sigma_6(v_6)\end{bmatrix},$$

where $(p_1,\sigma_1,p_2,\sigma_2,\ldots,p_6,\sigma_6) \in \hat{\delta}$; and the final configuration is checked by

$$r/b_1/b_2/\ldots/b_n/0_1/0_2/\ldots/0_n/\mathit{cell}(u,v) \longrightarrow r\big[q_f(u)\big]\,.$$

The auxiliary pattern $\mathit{Step}(\bar{u},\bar{v})$ is defined as

$$\mathit{Step}(\bar{u},\bar{v}) = \bigvee_{i=1}^{n} //\_\begin{bmatrix} a_i/b_{i+1}/b_{i+2}/\ldots/b_n/\mathit{ThreeCells}(\bar{x},u_1,u_2,u_3,v_1,v_2,v_3),\\ b_i/a_{i+1}/a_{i+2}/\ldots/a_n/\mathit{ThreeCells}(\bar{x},u_4,u_5,u_6,v_4,v_5,v_6)\end{bmatrix},$$

$$\mathit{ThreeCells}(\bar{x},\bar{u},\bar{v}) = \bigvee_{i=1}^{n-1} \mathit{ThreeCells}_i(\bar{x},\bar{u},\bar{v}) \;\vee\; \bigvee_{i=1}^{n-1} \mathit{ThreeCells}'_i(\bar{x},\bar{u},\bar{v})$$

and $\mathit{ThreeCells}_i(\bar{x},\bar{u},\bar{v})$, $\mathit{ThreeCells}'_i(\bar{x},\bar{u},\bar{v})$ are shown in Figure 7.

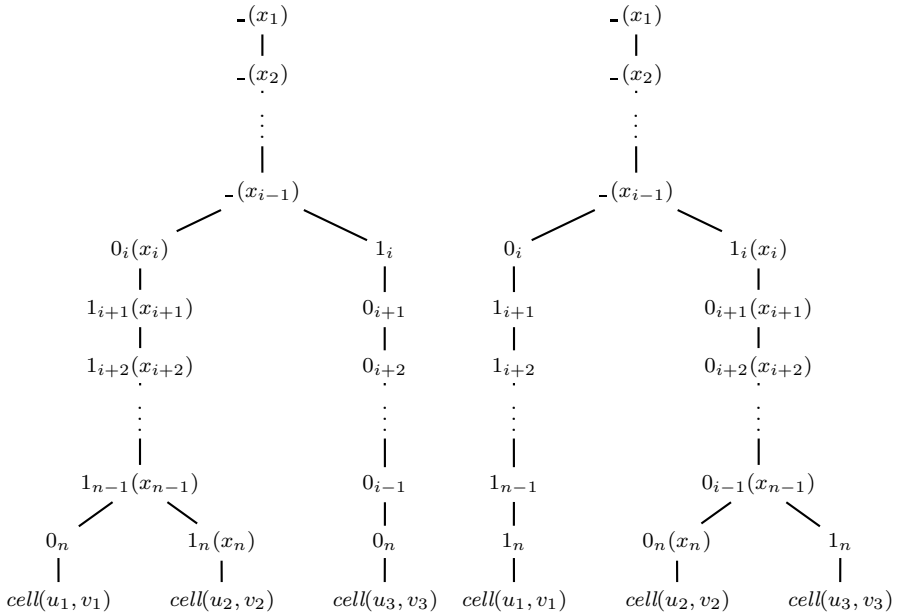
Proving correctness of the construction above poses no difficulties. Each occurrence of // can be replaced with a disjunction of sequences of / and _ symbols. Disjunction is only used at the source side, and therefore can be eliminated at the cost of multiplying dependencies.

(3) In Theorem 6.7 the NEXPTIME lower bound was proved for $\mathrm{SM}^{\mathrm{nr}}(\downarrow,\_,=)$. To obtain the lower bound for fully specified mappings from $\mathrm{SM}^{\mathrm{nr}}(\downarrow,\_,\sim)$, we give a reduction from satisfiability of Bernays-Schoenfinkel formulae, which is known to be NEXPTIME-hard [Lewis 1980]. Let the given formula be

$$\exists x_1 \cdots \exists x_m \forall x_{m+1} \cdots \forall x_n \bigwedge_{i=1}^{k} \bigvee_{j=1}^{\ell} C_{i,j},$$

where C_{ij} is an atom or a negated atom over a relational signature $\mathcal{R}$. It is known that if a Bernays-Schoenfinkel formula has a model at all, it has a model of size $N = m + \max_{R\in\mathcal{R}} \mathrm{ar}(R)$. The idea is to guess a model in the source tree, with assistance of the target tree.

The target DTD is simply $D_t = \{r \to \varepsilon\}$ and the source DTD D_s is defined as



(a) $\mathit{ThreeCells}_i(\bar{x}, \bar{u}, \bar{v})$ (b) $\mathit{ThreeCells}'_i(\bar{x}, \bar{u}, \bar{v})$

Fig. 7. Auxiliary patterns used in $\mathit{ThreeCells}(\bar{x}, \bar{u}, \bar{v})$

follows:

$$
\begin{array}{rll}
r \to t\, f\, v_1\, v_2\, \cdots\, v_N\, x_1 & & \\
x_i \to v\, x_{i+1} & & 1 \le i < m \\
x_m \to v\, x_{m+1,1}\, x_{m+1,2}\, \cdots\, x_{m+1,N} & & \\
x_{i,j} \to v\, x_{i+1,1}\, x_{i+1,2}\, \cdots\, x_{i+1,N} & & m+1 \le i < n,\ 1 \le j \le N \\
x_{n,i} \to v \wedge & & 1 \le i \le N \\
\wedge \to \vee_1\, \vee_2\, \cdots\, \vee_k & & \\
\vee_i \to C_{i,1}\, C_{i,2}\, \cdots\, C_{i,\ell} & & \\
C_{i,j} \colon @attr & & 1 \le i \le k,\ 1 \le j \le \ell \\
v_i, v, t, f \colon @attr & & 1 \le i \le N
\end{array}
$$

The idea is that each x_i and $x_{i,j}$ holds a value (as an attribute). The initial sequence of x_i's encodes the existential guesses. Below, each of the branching $x_{i,j}$ paths corresponds to the universal guesses. Thus each path from x_1 to $x_{n,j}$ is a valuation of all variables. At the bottom of each path we store truth values of the literals $C_{i,j}$ with respect to this valuation. One remaining problem is that we have to make sure that all the values come from a fixed N-element universe. For that purpose we encode the universe in source trees in the sequence of nodes labeled $v_1, v_2, \ldots, v_N$ and ensure that each value appearing in the valuations comes from the universe. Apart from the the encoding of the model described above, we store

sample truth values in the nodes t (*true*) and f (*false*). Note that each "variable" node has a child v which is meant to hold the assigned value. The attribute of each node labeled with $C_{i,j}$ will be used to store the truth value w.r.t. the valuation encoded by the path leading to this node.

Let us now describe the dependencies. To avoid confusion with the Bernays-Schoenfinkel formula's variables, we will be using capital letters for variables in dependencies. For notational simplicity we use //, but it can be easily replaced with a sequence of / and _ of suitable length. We also use an auxiliary tree pattern $\mathrm{path}_i(X_1, \ldots, X_i)[\psi]$ defined as

$$\mathrm{path}_1(X_1)[\psi] = \_[v(X_1), \psi]\,,$$
$$\mathrm{path}_{i+1}(X_1, X_2, \ldots, X_{i+1})[\psi] = \_[v(X_1), \mathrm{path}_i(X_2, X_3, \ldots, X_{i+1})[\psi]]\,.$$

We omit the subscript i, since it is clear from the number of arguments, e.g., $\mathrm{path}(X, Y, Z)[\psi] = \_[v(X), \_[v(Y), \_[v(Z), \psi]]]$. To enhance appropriate intuitions we slightly abuse the notation and write $\mathrm{path}(X_1, X_2, \ldots, X_n)/\psi$ instead of $\mathrm{path}(X_1, X_2, \ldots, X_n)[\psi]$ and $\mathrm{path}(X_1, X_2, \ldots, X_n)//\psi$ instead of $\mathrm{path}(X_1, X_2, \ldots, X_n)[//\psi]$.

First, make sure that the sample truth values are not equal, and that each value stored in a $C_{i,j}$ node is one of the sample truth values:

$$r[t(X), f(X)] \longrightarrow \bot$$
$$r[t(Y), f(Z), //C_{i,j}(X)], X \neq Y, X \neq Z \longrightarrow \bot \qquad \text{for all } i, j\,.$$

Second, check that all the "variable" values are taken from the universe: for all $1 \leq i \leq N$

$$r[\mathrm{path}(X_1, \ldots, X_n), v_1(Y_1), \ldots, v_N(Y_N)], X_i \neq Y_1, X_i \neq Y_2, \ldots, X_i \neq Y_N \longrightarrow \bot.$$

Since each branching for x_i with $i > m$ is meant to be a universal quantification, all the data values must be different:

$$//\_[x_{i,p}[v(X)], x_{i,q}[v(X)]] \longrightarrow \bot \qquad \text{for all } i > m \text{ and } p \neq q\,.$$

Next, we need to make sure that the truth values of relations are consistent. Take $C_{i,j} = K(x_{i_1}, x_{i_2}, \ldots, x_{i_s})$ and $C_{i',j'} = K'(x_{i'_1}, x_{i'_2}, \ldots, x_{i'_s})$, with $K, K' \in \{R, \bar{R}\}$ for some $R \in \mathcal{R}$. We have to check that for every two $\bar{a}$ and $\bar{b}$ such that $a_{i_p} = b_{i'_p}$ for $p = 1, 2, \ldots, s$, the values of $C_{i,j}[\bar{a}]$ and $C_{i',j'}[\bar{b}]$ coincide. For $K = K'$ this is expressed by

$$r\left[\mathrm{path}(X_1, X_2, \ldots, X_n)//C_{i,j}(V),\ \mathrm{path}(X'_1, X'_2, \ldots, X'_n)//C_{i',j'}(V')\right],$$
$$X_{i_1} = X'_{i'_1}\ ,\ \ldots\ ,\ X_{i_s} = X'_{i'_s} \ \longrightarrow\ V = V'$$

and for $K \neq K'$ replace $V = V'$ with $V \neq V'$.

Finally, we have to ensure the truth values assigned make our formula true:

$$r[f(X), //\vee_i[C_{i,1}(X), \ldots, C_{i,\ell}(X)] \longrightarrow \bot \qquad \text{for all } 1 \leq i \leq k\,.$$

It is straightforward to see that the mapping is consistent iff the given formula is satisfiable. □

PROPOSITION 6.9. *For mappings from* $\mathrm{SM}^{\mathrm{nr}}(\downarrow, \sim, \mathsf{Fun})$ *which do not use* $\neq$ *on the source side,* CONS *is in* PTIME.

PROOF. Proceed like in Proposition 6.8: remove from the productions all expressions of the form σ^*, $\sigma?$, replace σ^+ with σ, and consider the unique tree T conforming to the modified source DTD. Again, we need to fill in the data values. This time we cannot work directly with all the nodes of T, because T is potentially exponential. Instead, we will use a trick based on the fact that only polynomial part of the tree matters for the source side tree patterns.

As the source DTD is nested relational, siblings have different labels. Hence, each node of the source tree is determined by the sequence of labels of the nodes forming the unique path from the root to this node. Moreover, for fully specified patterns there exists at most one homomorphism into T. Hence, each source side pattern that can be mapped assigns each of its variables to exactly one node, determined by the sequence of labels on the path in the pattern leading from the root to the variable.

Make sure that no variable is used twice in any source side pattern and then modify the mapping as follows. For each dependency whose source side pattern can be mapped into T, replace in each variable with a constant C_ρ associated with the path ρ from root to the variable in the source side pattern, and remove the pattern from the source side (keeping the equalities), e.g., $r[a(x), a/b(y)], x = y \longrightarrow r/d(x, y)$ should be replaced with $C_{ra} = C_{rab} \longrightarrow r/d(C_{ra}, C_{rab})$.

It suffices to check if the trivial tree r has a solution with respect to the modified mapping. Notice however, that applying the solution building procedure would still lead to an exponential algorithm, since the target tree might need to be exponential in the size of the target DTD.

This time we also need to modify the target DTD. Observe that we can harmlessly relax the target schema by replacing in the productions all occurrences of ℓ with $\ell?$, and ℓ^+ with ℓ^*. Indeed. each solution conforming to the relaxed DTD can be easily extended so as to conform to the original DTD by simply adding the required nodes, and since we disallow $\rightarrow$, adding nodes will not violate the target side patterns. For the modified DTD each pattern is complete (see page 25). In consequence, the completion phase of the solution building algorithm can be skipped, and the algorithm becomes polynomial in $|T|^d + \|\mathcal{M}'\|$, which gives polynomial procedure as $\mathcal{M}'$ has no variables. □

D. COMPLEXITY OF COMPOSITION

THEOREM 7.2. *For schema mappings from* SM($\Downarrow, \Rightarrow$), COMPMEMBERSHIP *is* 2-EXPTIME-*complete,* COMPMEMBERSHIP($\mathcal{M}, \mathcal{M}'$) *is in* EXPTIME*, and there exist* $\mathcal{M}, \mathcal{M}'$ *for which it is* EXPTIME-*complete.*

PROOF. It remains to show the lower bounds.

(1) To obtain the 2-EXPTIME lower bound for COMPMEMBERSHIP we give a reduction from the membership problem for alternating EXPSPACE Turing machines, which is known to be 2-EXPTIME-hard. For a machine M and an input word w we will build a mapping $\mathcal{M} = (D_s, D_t, \Sigma)$ and a tree T conforming to D_t such that $(S, T) \in [\![\mathcal{M}]\!]$ for some S if and only if M accepts w. We show hardness of COMPMEMBERSHIP by asking whether $(r, T) \in [\![\mathcal{M}_1]\!] \circ [\![\mathcal{M}]\!]$, where $\mathcal{M}_1 = (\{r \to \varepsilon\}, D_s, \emptyset)$.

The idea of the reduction is that the tree S encodes an accepting computation

tree of M on w, whose correctness is enforced by the mapping $\mathcal{M}$ with the help of a suitable encoding of the transition relation of M stored in T. The main difficulty is that we are not allowed to use data comparisons in patterns. We circumvent this obstacle by storing each needed equality/inequality relation on data tuples in T.

Let p be a polynomial and let M be an alternating Turing machine running in time $2^{p(|w|)}$ with the tape alphabet A including the blank symbol $\flat$, the state space $Q = Q_\forall \cup Q_\exists$, the initial state q_0, and the transition relation δ. Without loss of generality we assume that $q_0 \in Q_\forall$, all final states are existential, each transition leads from $Q_\forall$ to $Q_\exists$ or from $Q_\exists$ to $Q_\forall$, and for each $q \in Q$ and each $\sigma \in A$, there are exactly two different transitions of the form $(q, \sigma, \dots)$. Fix also an input word w and let $n = p(|w|)$.

The source DTD is given as

$$\begin{aligned}
r &\to \mathit{conf} \\
\mathit{conf} &\to \mathit{conf}_1\ \mathit{conf}_2\ 0_1\ 1_1 \\
\mathit{conf}_1, \mathit{conf}_2 &\to (\mathit{conf} \mid \mathit{leaf})\ 0_1\ 1_1 \\
0_i, 1_i &\to 0_{i+1}\ 1_{i+1} && 1 \le i < n \\
0_n, 1_n &\to \mathit{cell} \\
\mathit{cell}&\colon @\mathit{state}\ @\mathit{symbol}
\end{aligned}$$

and the labels $\mathit{conf}, \mathit{conf}_1, \mathit{conf}_2$ and $0_i, 1_i$ have a single attribute. The *conf* nodes represent configurations with universal states and have two children labeled $\mathit{conf}_1, \mathit{conf}_2$ representing configurations with existential states. Each of these has a single child labeled *conf*. Under each node representing a configuration one stores the content of the tape in the leaves of the exponential subtree consisting of nodes with labels $0_i, 1_i$. The address of the cell is the sequence of labels on the path leading to the leaf. We will make sure that for each node on the path, its label is mirrored in the data value it stores. For each cell we remember the tape symbol and the state (or the information that the head is elsewhere).

The target DTD is given as

$$\begin{aligned}
r &\to \mathit{forall}^*\ \mathit{exists}^*\ \mathit{init}^*\ \mathit{step}^*\ \mathit{final}^*\ \mathit{zero}\ \mathit{one} \\
\mathit{step} &\to a_1^* a_2^* \dots a_n^* \\
\mathit{init} &\to b_1^* b_2^* \dots b_n^* \\
\mathit{forall}&\colon @\mathit{st}\ @\mathit{sym}\ @\mathit{tr}_1\ @\mathit{tr}_2 \\
\mathit{exists}&\colon @\mathit{st}\ @\mathit{sym}\ @\mathit{tr} \\
\mathit{step}&\colon @\mathit{tr}\ @\mathit{st}_1\ @\mathit{sym}_1\ @\mathit{st}_2\ @\mathit{sym}_2\ \cdots\ @\mathit{st}_6\ @\mathit{sym}_6 \\
\mathit{init}&\colon @\mathit{st}\ @\mathit{sym} \\
\mathit{final}&\colon @\mathit{st} \\
b_i&\colon @\mathit{attr}_1\ @\mathit{attr}_2 && 1 \le i \le n \\
\mathit{zero}, \mathit{one}, a_i&\colon @\mathit{attr} && 1 \le i \le n
\end{aligned}$$

The nodes *zero* and *one* are meant to store the values encoding the addresses of tape cells. The *exist* nodes store transitions allowed by δ for a given existential state and tape symbol. The *forall* nodes store both transitions allowed by δ for a

given universal state and tape symbol. The *final* nodes store the accepting states of the machine.

The subtrees rooted at *init* nodes store the initial configuration of the machine for the input word w. The target tree we are going to construct will satisfy a pattern

$$r/\mathit{init}(q,\sigma)\big[b_1(j_1), b_2(j_2), \ldots, b_n(j_n)\big]$$

if and only if the content of the cell number $\langle j_1 j_2 \ldots j_n\rangle_2$ (we are counting from zero) is described by the state q and symbol σ (possibly decorated with $\triangleleft$ or $\triangleright$ to denote the beginning and the end of the tape). We use a mock state $\natural \notin Q$ to say that the head is elsewhere. For simplicity we assume that the data domain contains $Q \cup \{\natural\}$ and A, as well as 0 and 1. Formally, one can think of an encoding with distinct data values.

The *step* nodes store the relation $\hat{\delta}$ extended with the information on the transition that leads to this evolution of the tape. Again, we are assuming that each tuple from δ is contained in the data domain as a single value. The target tree is going to satisfy the pattern

$$r/\mathit{step}(t, q_1, q_2, \ldots q_6, \sigma_1, \sigma_2, \ldots, \sigma_6)\big[a_1(j_1,k_1), a_2(j_2,k_2), \ldots, a_n(j_n,k_n)\big]$$

if either $\langle j_1 j_2 \ldots j_n\rangle_2 \neq \langle k_1 k_2 \ldots k_n\rangle_2$ or the three consecutive cells described by $q_1\sigma_1 q_2\sigma_2 q_3\sigma_3$ can evolve into $q_4\sigma_4 q_5\sigma_5 q_6\sigma_6$ after the machine takes transition t.

Let us now encode these relations in a polynomial-size target tree T. Let k be the smallest number such that $|w| \leq 2^k$. Note that 2^k is polynomial in the length of w. We construct T by arranging the following nodes and subtrees according to the target schema:

—$\mathit{zero}(0)$ and $\mathit{one}(1)$;

—$\mathit{final}(q)$ for each final accepting state q, and for $q = \natural$;

—$\mathit{exists}(q,\sigma,t)$ for all $q \in Q_\exists \cup \{\natural\}$, $\sigma \in \hat{A}$, $t \in \delta$ such that $q = \natural$ or $t = (q, \tau, \ldots)$ and $\sigma \in \{\tau, \tau^\triangleleft, \tau^\triangleright\}$;

—$\mathit{forall}(q,\sigma,t_1,t_2)$ for all $q \in Q_\exists \cup \{\natural\}$, $\sigma \in \hat{A}$, $t_1, t_2 \in \delta$ such that $q = \natural$ or t_1 and t_2 are precisely the two transitions of the form $(q, \tau, \ldots)$ where $\sigma \in \{\tau, \tau^\triangleleft, \tau^\triangleright\}$;

—$\mathit{init}(q,\sigma)\big[b_1(0), b_2(0), \ldots, b_{n-k}(0), b_{n-k+1}(j_1), b_{n-k+2}(j_2), \ldots, b_n(j_k)\big]$ for all $j_1, j_2, \ldots, j_k \in \{0,1\}$ such that $\ell = \langle j_1 j_2 \ldots j_k\rangle_2$ and q, σ describe the content of the ℓth cell in the initial configuration, i.e.,

$$q = \begin{cases} q_0 & \text{for } \ell = 0, \\ \natural & \text{for } \ell > 0, \end{cases} \qquad \sigma = \begin{cases} \tau_0^\triangleright & \text{for } \ell = 0, \\ \tau_\ell & \text{for } 0 < \ell < |w|, \\ \flat & \text{for } \ell \geq |w|, \end{cases}$$

where $w = \tau_0 \tau_1 \ldots \tau_{|w|-1}$;

—$\mathit{init}(\natural,\flat)\begin{bmatrix} b_1(0), \ldots, b_{j-1}(0), b_{j+1}(0), \ldots, b_n(0), \\ b_1(1), \ldots, b_{i-1}(1), b_{i+1}(1), \ldots, b_n(1) \end{bmatrix}$ for all $1 \leq j \leq n-k$, $1 \leq i \leq n$, $i \neq j$;

—$\mathit{init}(\natural,\flat^\triangleleft)\big[b_1(1), b_2(1), \ldots, b_n(1)\big]$;

—$\mathit{step}(t, q_1, \ldots q_6, \sigma_1, \ldots, \sigma_6)\begin{bmatrix} a_1(0,0), \ldots, a_n(0,0), \\ a_1(1,1), \ldots, a_n(1,1) \end{bmatrix}$ for all $q_1, \ldots, q_6 \in Q \cup \{\natural\}$, $\sigma_1, \ldots, \sigma_6 \in \hat{A}$, $t \in \delta$ such that $(q_1, \sigma_1, \ldots, q_6, \sigma_6) \in \hat{\delta}$ and the step of the computation is consistent with the transition t;

$$\text{—}step(t, q_1, \ldots q_6, \sigma_1, \ldots, \sigma_6) \begin{bmatrix} a_1(0,0), \ldots, a_{i-1}(0,0), a_{i+1}(0,0), \ldots, a_n(0,0), \\ a_1(1,1), \ldots, a_{i-1}(1,1), a_{i+1}(1,1), \ldots, a_n(1,1), \\ a_1(0,1), \ldots, a_n(0,1), \\ a_1(1,0), \ldots, a_n(1,0) \end{bmatrix} \text{ for}$$

all $1 \le i \le n$, $q_1, \ldots, q_6 \in Q \cup \{\natural\}$, $\sigma_1, \ldots, \sigma_6 \in \hat{A}$, $t \in \delta$.

Let us now describe the dependencies. We make sure that data values reflect labels in the 0_i and 1_i nodes by introducing the following dependencies for $i = 1, 2, \ldots, n$

$$r//0_i(x) \longrightarrow r/zero(x)\,,$$
$$r//1_i(x) \longrightarrow r/one(x)\,.$$

Correctness of the computation tree is enforced by the dependencies

$$r/\_/Tape(\bar{x}, u, v) \longrightarrow r/init(u, v)[a_1(x_1), a_1(x_2), \ldots, a_n(x_n)]\,,$$
$$r//\_\begin{bmatrix} Tape(\bar{x}, u, v), \\ conf_1(z_1), conf_2(z_2) \end{bmatrix} \longrightarrow r/forall(u, v, z_1, z_2)\,,$$
$$r//\_[Tape(\bar{x}, u, v), conf(z)] \longrightarrow r/exists(u, v, z)\,,$$
$$r//\_\begin{bmatrix} ThreeCells(\bar{x}, \bar{t}, \bar{u}), \\ \_(z)/ThreeCells(\bar{y}, \bar{v}, \bar{w}) \end{bmatrix} \longrightarrow r/step(z, \bar{t}, \bar{u}, \bar{v}, \bar{w})\,[b_1(x_1, y_1), \ldots, b_n(x_n, y_n)]\,,$$
$$r//\_[Tape(\bar{x}, u, v), leaf] \longrightarrow r/final(u)\,,$$

where the auxiliary patterns are defined as

$$Tape(\bar{x}, u, v) = \_(x_1)/\_(x_2)/\ldots/\_(x_n)/cell(u, v)$$
$$ThreeCells(\bar{x}, \bar{u}, \bar{v}) = \bigvee_{i=1}^{n-1} ThreeCells_i(\bar{x}, \bar{u}, \bar{v}) \;\vee\; \bigvee_{i=1}^{n-1} ThreeCells'_i(\bar{x}, \bar{u}, \bar{v})$$

and $ThreeCells_i(\bar{x}, \bar{u}, \bar{v})$, $ThreeCells'_i(\bar{x}, \bar{u}, \bar{v})$ are shown in Figure 7 on page 72.

Proving correctness of the construction above poses no difficulties. Each occurrence of // can be replaced with a disjunction of sequences of / and _ symbols. Disjunction is only used at the source side, and therefore can be eliminated at the cost of multiplying dependencies.

(2) Now, we shall construct mappings $\mathcal{M} = (D_1, D_2, \emptyset)$ and $\mathcal{M}' = (D_2, D_3, \Sigma_{23})$ in SM($\Downarrow, \Rightarrow$) such that CompMembership($\mathcal{M}, \mathcal{M}'$) is EXPTIME-hard. We will reduce from the non-universality problem for bottom-up non-deterministic automata on binary trees, modifying a proof from [Arenas and Libkin 2008].

First, define D_3 over $\{r, label, state, nontr, rejecting\}$ as

$$r \to \sharp\, state^*\, \natural\, label^*\, \flat\, nontr^*$$
$$state \to rejecting?$$
$$rejecting, label, nontr \to \varepsilon$$
$$label, state : @attr$$
$$nontr : @left, @right, @label, @up$$

A tree conforming to D_3 is meant to encode an automaton. It stores the alphabet in the *label*-nodes, state space in the *state*-nodes (we assume that the initial state

is stored as first, just after ♯), and the *complement* of the transition relation. The reason we store the complement is that we do not have negation. We do not have to enforce anything on such a tree, since we will be constructing it ourselves based on a given automaton, when preparing input for composition membership algorithm. In particular, we will make sure, that all states, labels, and non-transitions are stored correctly.

Next, let D_2 over $\{r, node, label, state, leaf, yes, no\}$ be given by

$$\begin{aligned} r &\to node \\ node &\to label\,\sharp\, state^*\,\natural\,(node\; node \mid leaf) \\ state &\to yes \mid no \\ leaf, label &\to \varepsilon \\ state, label &: \; @attr \end{aligned}$$

A tree conforming to D_2 is meant to encode a rejecting run of the corresponding power set automaton. This time we will need to ensure that it really is a correct rejecting run with the dependencies, since this is precisely the tree that will be produced by the composition membership algorithm.

Finally, we define D_1 simply as $r \to \varepsilon$. The only tree conforming to D_1 will be used as a stub.

Let us now describe Σ_{23}, which will enforce the correctness of the run. First, we make sure that *label*-nodes store labels:

$$//label(x) \longrightarrow r/label(x)\,.$$

Second, we need to check that for each *node*-node, each state is stored in exactly one *state*-node, and that nothing else is stored there. We do this using the switching trick from the proof of Theorem 6.6 again:

$$\begin{aligned} //node[\sharp \to state(x)] &\longrightarrow r[\sharp \to state(x)]\,, \\ //node[state(x) \to state(y)] &\longrightarrow r[state(x) \to state(y)]\,, \\ //node[state(x) \to \natural] &\longrightarrow r[state(x) \to \natural]\,, \\ //node[state(x) \to^+ state(y)] &\longrightarrow r[state(x) \to^+ state(y)]\,. \end{aligned}$$

The last dependency guarantees that we do not store a state twice, which ensures that for each state we either have a *yes*-node, or a *no*-node.

Next, we make sure that the *yes*/*no* nodes are properly assigned in the leaves, and then properly propagated up the tree:

$$//node[state(x)/no,\, label(u),\, leaf] \longrightarrow r[\sharp \to state(y), nontr(y, y, u, x)]\,,$$

$$//node[state(x)/no,\, label(u),\, node/state(y)/yes \to node/state(z)/yes] \longrightarrow \\ \longrightarrow r/nontr(y, z, u, x)\,.$$

Finally, check that the run is rejecting:

$$r/node/state(x)/yes \longrightarrow r/state(x)/rejecting\,.$$

Let us see that $\textsc{CompMembership}\big((D_1, D_2, \emptyset), (D_2, D_3, \Sigma_{23})\big)$ is indeed EXPTIME-hard. Take an automaton $\mathcal{A} = (\Gamma, Q, \delta, q_0, Q_F)$ with $\Gamma =$

$\{a_1, a_2, \ldots, a_m\}$ and $Q = \{q_0, q_1, \ldots, q_n\}$. Without loss of generality we may assume that $Q_F = \{q_k, q_{k+1}, \ldots, q_m\}$. Let

$$Q \times Q \times \Gamma \times Q \setminus \delta = \{(p_1, r_1, b_1, s_1), (p_2, r_2, b_2, s_2), \ldots, (p_\ell, r_\ell, b_\ell, s_\ell)\}\,.$$

Encode A as a tree T_A defined as

$$\begin{aligned} r[\sharp,\ &\mathit{state}(q_0)/\mathit{rejecting},\ \mathit{state}(q_1)/\mathit{rejecting},\ \ldots,\ \mathit{state}(q_{k-1})/\mathit{rejecting},\\ &\mathit{state}(q_k),\ \mathit{state}(q_{k+1}),\ \ldots,\ \mathit{state}(q_n),\ \natural,\\ &\mathit{label}(a_1),\ \mathit{label}(a_2),\ \ldots,\ \mathit{label}(a_m),\ \flat,\\ &\mathit{nontr}(p_1, r_1, b_1, s_1), \mathit{nontr}(p_2, r_2, b_2, s_2), \ldots, \mathit{nontr}(p_\ell, r_\ell, b_\ell, s_\ell)]\,. \end{aligned}$$

Proving that A rejects some tree iff $(r, T_A) \in [\![(D_1, D_2, \emptyset)]\!] \circ [\![(D_2, D_3, \Sigma_{23})]\!]$ is straightforward. □

D.1 Consistency of composition

THEOREM 7.3.

—CONSCOMP($\Downarrow$) *and* CONSCOMP($\Downarrow, \Rightarrow, \mathsf{Fun}$) *are* EXPTIME*-complete.*
—CONSCOMP($\downarrow, \downarrow^+, \sim$) *and* CONSCOMP($\downarrow, \rightarrow, \sim$) *are undecidable.*

PROOF. (1) We first prove that CONSCOMP($\Downarrow, \Rightarrow, \mathsf{Fun}$) is in EXPTIME. The idea is the same as for CONS($\Downarrow, \Rightarrow, \mathsf{Fun}$) (Theorem 6.2). The composition of (D_1, D_2, Σ_{12}) and (D_2, D_3, Σ_{23}) is consistent iff the composition of $(D_1, D_2, \Sigma_{12}^\circ)$ and $(D_2, D_3, \Sigma_{23}^\circ)$ is consistent, so we may assume that DTDs have no arguments, and dependencies use no variables. Suppose $\Sigma_{12} = \{\varphi_i \longrightarrow \psi_i \,|\, i \in 1, 2, \ldots, n\}$ and $\Sigma_{23} = \{\varphi'_j \longrightarrow \psi'_j \,|\, j = 1, 2, \ldots, m\}$. The composition is consistent iff there exist $I \subseteq \{1, 2, \ldots, n\}$ and $J \subseteq \{1, 2, \ldots, m\}$ such that there are XML trees T_1, T_2, T_3 with

$$T_1 \models D_1 \wedge \bigwedge_{i \notin I} \neg\varphi_i\,, \qquad T_2 \models D_2 \wedge \bigwedge_{i \in I} \psi_i \wedge \bigwedge_{j \notin J} \neg\varphi'_j\,, \qquad T_3 \models D_3 \wedge \bigwedge_{j \in J} \psi'_j\,.$$

Equivalently, we need to check non-emptiness of the following automata for all possible I and J:

$$\mathcal{A}_{D_1} \times \prod_{i \notin I} \bar{\mathcal{A}}(\varphi_i)\,, \qquad \mathcal{A}_{D_2} \times \prod_{i \in I} \mathcal{A}(\psi_i) \times \prod_{j \notin J} \bar{\mathcal{A}}(\psi'_j)\,, \qquad \mathcal{A}_{D_3} \times \prod_{j \in J} \mathcal{A}(\psi'_j)\,.$$

Like for CONS($\Downarrow, \Rightarrow, \mathsf{Fun}$), this gives an exponential algorithm.

(2) Next we show how to reduce consistency of a single mapping to consistency of a composition of two consistent mappings. Suppose we are given $\mathcal{M} = (D_s, D_t, \Sigma)$. We provide two consistent mappings, $\mathcal{M}_1$ and $\mathcal{M}_2$, such that $\mathcal{M}$ is consistent iff the composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ is consistent.

Let $\natural$ be a fresh symbol, not used in $\mathcal{M}$. Let $D^\natural$ be the DTD obtained from D by extending the alphabet with $\natural$, and replacing the production for the root, $r \to \omega$, with the production $r \to \omega \,|\, \natural$. Define

$$\mathcal{M}_1 = (D_s^\natural, D_t^\natural, \Sigma \cup \{r/\natural \longrightarrow r/\natural\})\,, \qquad \mathcal{M}_2 = (D_t^\natural, D, \{r/\natural \longrightarrow r/\natural\})\,,$$

where D is just $r \to \varepsilon$. Observe that both mappings are consistent: $(r[\natural], r[\natural]) \in [\![\mathcal{M}_1]\!]$, and $(T, r) \in [\![\mathcal{M}_2]\!]$ for any $T \models D_t$.

Let us check that their composition is consistent iff $\mathcal{M}$ is consistent. If $(S,T) \in [\![\mathcal{M}]\!]$, then $(S,T) \in [\![\mathcal{M}_1]\!]$, $(T,r) \in [\![\mathcal{M}_2]\!]$, and so $(S,r) \in [\![\mathcal{M}_1 \circ \mathcal{M}_2]\!]$. Conversely, suppose $(T_1,T_2) \in [\![\mathcal{M}_1]\!]$ and $(T_2,T_3) \in [\![\mathcal{M}_2]\!]$. Note that $\natural$ gets propagated to T_3 if it occurs in T_1 or T_2. In consequence, T_1 and T_2 do not contain $\natural$. But then, $T_1 \models D_s$ and $T_2 \models D_t$. Since the pair (T_1,T_2) satisfies all the constraints from $\Sigma \cup \{r/\natural \longrightarrow r/\natural\}$, we conclude that $(T_1,T_2) \in [\![\mathcal{M}]\!]$.

Observe that if $\mathcal{M}$ uses neither $=$ nor $\neq$, the same is true for $\mathcal{M}_1$ and $\mathcal{M}_2$. Hence, the reduction proves that consistency of composition is EXPTIME-hard in the absence of $=$ and $\neq$, and is undecidable in their presence. □

PROPOSITION 7.4. *The problem of checking whether the composition of n mappings $\mathcal{M}_1,\dots,\mathcal{M}_n$ from* $\mathrm{SM}(\Downarrow,\Rightarrow,\mathsf{Fun})$ *is consistent is in* EXPTIME.

PROOF. The idea used in the proof of Theorem 7.3 can be extended to consistency of multifold composition. Suppose we are given $(D_1,D_2,\Sigma_1),\dots,(D_n,D_{n+1},\Sigma_n)$, where $\Sigma_i = \{\varphi_j^i \longrightarrow \psi_j^i \mid j = 1,2,\dots,n_i\}$ for $i = 1,2,\dots,n$ and we want to know if there are trees $T_1,\dots,T_n$ such that $(T_i,T_{i+1}) \in [\![(D_i,D_{i+1},\Sigma_i)]\!]$ for $i = 1,2,\dots,n$.

Like before, without loss of generality we can assume that the mappings use no attributes and no variables, and the problem becomes reducible to automata non-emptiness. For every n-tuple $(I_1,\dots,I_n) \subseteq \prod_{i=1}^n \{1,2,\dots,n_i\}$, test non-emptiness of the following automata:

$$\mathcal{A}_{D_1} \times \prod_{i \notin I_1} \bar{\mathcal{A}}(\varphi_i^1)\,,\ \dots,\ \mathcal{A}_{D_{j+1}} \times \prod_{i \in I_j} \mathcal{A}(\psi_i^j) \times \prod_{i \notin I_{j+1}} \bar{\mathcal{A}}(\varphi_i^{j+1})\,,\ \dots,\ \mathcal{A}_{D_{n+1}} \times \prod_{i \in I_n} \mathcal{A}(\psi_i^n)\,.$$

The composition of the given n mappings is consistent iff these products are nonempty for some $(I_1,\dots,I_n)$. □