# XML Data Exchange under Expressive Mappings

Shun'ichi Amano

Doctor of Philosophy

Laboratory for Foundations of Computer Science
School of Informatics
The University of Edinburgh

2009

# Abstract

Data Exchange is the problem of transforming data in one format (the source schema) into data in another format (the target schema). Its core component is a schema mapping, which is a high level specification of how such transformation should be done. Relational data exchange has been extensively studied, but exchanging XML data have been paid much less attention. The goal of this thesis is to develop a theory of XML data exchange with expressive schema mappings, extending a previous work using restricted mappings. Our mapping language is based on tree patterns that can use horizontal navigation and data comparison in addition to downward navigation.

First we look at static analysis problems concerning a single mapping. More specifically, we consider consistency problems with different flavours. One such problem, for instance, asks if any tree has a solution under the given mapping. Then we turn to analyse the complexity of mapping themselves, i.e., recognising pairs of trees such that the one is mapped to the other. For both problems, we provide classifications based on sets of features used in the mappings.

Second we investigate the composition of XML schema mappings. Generally it is hard, or rather simply impossible, to achieve closure under composition in XML settings unlike in relational settings. Nevertheless we identify a class of XML schema mappings that is closed under composition.

Lastly we consider the problem of query answering. It is important to exchange data so that we can feasibly answer queries while it often leads to intractability. We identify the dividing line between tractable and intractable cases: answering queries with extended features is always intractable while tractability of answering simple queries can be retained in extended mappings.

# Declaration

This thesis

- is entirely composed by me;
- contains work that is my own or a result of collaborating with others as specified in the text;
- is not substantially the same as any I have submitted for any other degree or professional qualification.

Shun'ichi Amano
Edinburgh, December 2009

# Acknowledgements

This thesis contains the work of some three years, during which I have benefited from many people.

First of all, I would like to thank my advisor, Professor Leonid Libkin, who taught me many things and introduced me to the subject of this thesis. Without him, this thesis would have been certainly impossible.

Laboratory for Foundations of Computer Science at the University of Edinburgh, and in particular the database group I have belonged to, gave me the opportunities to meet many interesting people. It is impossible to list them all one by one, but I would like to thank Claire and Filip for pleasant collaboration, Anthony for discussing lots of things together, and Tony and Juan for a nice reading group.

Last and probably least, and maybe somewhat strangely, I want to thank the Main Library whose East Asian Collection satisfied my desperate need for reading my native language, without which I would not have survived until the end of my PhD study.

Edinburgh, December 2009

# Contents

# Chapter 1

# Introduction

This thesis is about XML data exchange. In particular we mostly deal with two problems: static analysis of schema mappings and query answering.

In the problem of *data exchange*, source data (conforming to a source schema) must be restructured to form a solution conforming to a target schema. The restructuring must be done according to a *schema mapping*, a specification given by a set of rules (source-to-target dependencies) describing the relationship between the two schemas. Usually, there are many ways of doing this and the goal is to find a solution that would be as general as possible, thus making it possible to answer queries over the target without inventing facts.

While data exchange is an old problem [92], recently it has become even more important due to the proliferation of data in various formats. It has been a topic of active research over the past few years. By now we have a very good understanding of relational data exchange (see e.g., [15, 22, 66]); several advanced prototypes for specifying and managing mappings have been developed and incorporated into commercial systems [78, 88].

Schema mappings are important components in data exchange, and also essential for data integration tasks as well as for peer-to-peer data management. All ETL (extract-transform-load) tools come with languages for specifying mappings. There are techniques for using such mappings in data integration and exchange, and tools for handling mappings themselves, for example, for defining various operations on them [4, 21, 22, 37, 48, 66, 73, 82].

Query answering in relational data exchange is also well understood. Usually, there are many possibilities (solutions) of restructuring a source database. The important notions were universal solutions and cores, which in a way correspond to "most general" solutions. They were first introduced and studied in [45, 47]; a sophisticated method of

feasibly computing cores was developed in [56].

Meanwhile, exchanging XML data and mappings between XML schemas have been paid much less attention. While commercial ETL tools often claim to provide support for XML schema mappings, this is typically done either via relational translations, or by means of very simple mappings that establish connections between attributes in two schemas. Transformation languages of such tools tend to concentrate on manipulating values rather than changing structure. In research literature, most XML schema mappings are obtained by various matching tools (see, e.g., [77, 79]) 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., [14]) or in XML-to-XML mappings, where simple navigational queries were used in addition to relationships between attributes [50]. One extra step was made in [12], which studied extensions of relational data exchange techniques to XML, and introduced XML schema mappings[1] that could use (vertical) navigation as well as bind attribute variables. But even the mappings of [12] 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 [45, 66, 78].

Our goal is to develop the theory of XML data exchange under expressive mappings that use horizontal navigation and data comparisons in addition to simple downward navigation. We would like to introduce a formalism that will be an analog of the commonly accepted formalism of source-to-target dependencies used in relational schema mappings [13, 45, 47, 48, 49, 66]. We would also like to understand the basic properties of such mappings, including their complexity, static analysis questions related to them, as well as operations on XML schema mappings. Furthermore, we study how extending mappings and query language with more expressive tree patterns affect the complexity of query answering problem.

## 1.1 Related Work

### 1.1.1 Relational Data Exchange

The origin of data exchange dates back to 70's [92], and it has revived as a research topic in recent years due to increasing demand for exchange data in a variety of formats, e.g., in e-business applications [9]. As mentioned earlier, one example of advanced prototypes is

[1] which they called XML data exchange settings.

Clio, which has been developed and is now a part of DB2[78, 88].

Motivated by issues surrounding Clio project, a formal study of relational data exchange was initiated by people (mostly) at IBM Almaden [45, 47]. In [45], they defined the formal semantics of relational data exchange and that of query answering, adapting the certain answers semantics. They further studied which solutions are better than others, using a graph-theoretic notion of core in [47].

The complexity issues concerning computing solutions was investigated in [67]. An interesting aside in the undecidability proof of this paper is that it uses the *embedding problem* from universal algebra, which is actually equivalent to the word problem.

The complexity of query answering was already studied in [45], but there was a subtle gap: it is shown there that union of conjunctive queries with at most inequality per disjunct can be answered in polynomial time while those with at least six inequalities leads to coNP-completeness of query answering. They conjectured two inequalities would be sufficient to get coNP-hardness, which is in fact proved to be the case in [74]. A way to accommodate inequalities was considered in [11], with a simplified proof of the intractability of queries with two inequalities.

Although the semantics of [45] is quite natural and widely accepted, some counter-intuitive behaviour was pointed out in [10]. Libkin [70] observed that considering the semantics with Closed World Assumption (CWA) leads to a resolution of the anomaly. The CWA semantics was further extended in [57] to deal with schema mapping with target dependencies; Libkin and Sirangelo [72] considered mixed approach of Open and Closed World Assumption. In [3], answering aggregate queries in data exchange was considered, and was shown to require a strong restriction of CWA semantics.

There has been research on operations on schema mappings, mostly in the framework of meta data management [20, 22]. The most extensively studied is *composition* of schema mappings, i.e., how to define a schema mapping representing the (set-theoretic) composition of two mappings. The problem was first studied in [48], where it was shown that conjunctive query formalism cannot be generally composed and the minimal class of mappings that is closed under composition requires extension with Skolem functions. The *inverse* of mappings is also a well-studied operation. So far it seems that there has been no standard definition. Initial attempt was done in [43]. The problem was that the existence of inverse is rather rare. Then it was remedied in [44] while a different approach was given in [13]. Yet another attempt was made in [49], where source instances are also allowed to have nulls. In [46], optimisation of schema mapping was considered.

Another line of data exchange research is peer data exchange [54, 55], where there are

multiple sources and targets. Also, a closely related area of research is data integration [68], where the problem is not totally transforming data but materialising a view from multiple sources. This connection is explored for example in [55].

### 1.1.2 Theory of XML, with or without Data

#### XML as Node-labelled Trees

Formal language theory, or theory of finite automata, is one of the oldest branches in theoretical computer science, and tree languages were already considered in the 60's ([42, 96]). Since then a lot of results have been established on tree automata, but largely for their own and without much application. Automata theory regained applicability in the 80's through program verification ([100]): Tree languages were used as abstractions of programs. Interests in tree languages got even wider with the appearance of XML (Extensible Markup Language) as a data representation language. Thus from the beginning of XML research, tree automata theory has been one of the most frequently used ([32, 84, 101, 102]).

There have been many aspects of XML research where automata are useful: Schema languages, query languages, transformation of documents, just to name a few. See [89] for a survey. Also automata are inseparably attached to logics since the fundamental result of Büchi that showed the equivalence of finite automata and monadic second order logic ([33, 34]; see also [97, 98]). Thus lots of logics have been considered for trees as well. See [71] for a survey.

#### Towards Infinity

Abstracting XML documents as (node-labelled) trees always requires one thing: neglect of data values (`PCDATA`). Most research had followed this line and focused on analysing structures of XML documents. The problem with data values is that the domain of those data is not finite: For example, it can be natural numbers. Any automata model depends strongly on the finiteness of its alphabet, so the nice tool of automata is no longer usable and analysis of XML with data tends to become much more involved. As such, sparse had been work that does take into account data values in XML: In [51], integrity constraints such as keys in XML databases were studied; In [18], satisfiability of various fragments of XPath were considered, in particular including fragments that allow data comparison; Also [5] investigates typechecking (schema validation) XML documents transformed according to XSLT-like rules.

In recent years, however, XML with data has started to attract much attention. It is

actually not only about data in XML documents, but also is motivated by program verification, where there are many sources of infinity such as variables ranging over natural numbers.

Undecidability and high complexity seems to be a price for considering data values. In [28], the authors showed that FO with three variables is already undecidable over words with data (modelled by the built-in equivalence relation) and studied two variable fragment, showing its decidability. They also considered its tree analogue in[27] with limited decidability results. Temporal logics over data words and data trees were defined in [40, 62]. Extending these, Figueira revisited in [52, 53] the satisfiability problem of XPath with data comparison, taking up where [18] left off. In [29, 87], it was shown that evaluation of XPath with data comparison can be done in linear combined complexity.

Quite naturally automata models over data words/trees were also considered. In fact there are lots of them ([24, 26, 31, 85, 94, 95]; see also [90]). This proliferation seemingly suggests that there is no such thing as "the" automata model for structures with data.

Apart from works mentioned above, logics interpreted over structures with built-in equivalence relations (essentially, first-order relational structure with data in the above terms) are considered for their own sake, for example in [63, 64, 65].

## 1.2 Collaboration

Collaborating with other people has been a great part of the work presented in this thesis. While the whole text is composed by the author, it should be mentioned that the results were obtained through collaboration with Claire David, Leonid Libkin, and Filip Murlak. The author would like to mention in particular that the involved proof of theorem 3.12 is due to Filip Murlak.

## 1.3 Publications

The work contained in this thesis is (to be) published in [7] and [6]. Roughly the chapters 3 and 4 correspond to the former and the chapter 5 to the latter.

## 1.4 Thesis Structure

This thesis consists of six chapters, the first of which is current. In chapter 2, we fix terminology and notation. In particular we quickly summarise the results in relational data

exchange. We also define tree automata and our (standard) abstraction of XML documents. In chapter 3, we study our version of classical static analysis, namely consistency of XML schema mappings. Expressive mappings leads to undecidability, and even with restriction on DTDs complexity of the problem is significantly higher than that of simple mappings. Chapter 4 is concerned with composing XML schema mappings, which had not been considered before. We show that it is rather difficult to compose, even with the help of Skolem functions, unlike the relational case. Nevertheless we identify a composable class of mappings and present an algorithm to compute its syntactic representation. In chapter 5, we turn to the problem that really involves exchanging data – that of query answering. What we observe there is that querying about order in certain answers is rather costly, easily leading to coNP-hardness. On the other hand, extending mapping language alone still allows us to retain tractable query answering. In the final chapter we mention some future directions and draw conclusions.

# Chapter 2

# Preliminaries

After fixing some standard notation, we review *relational* data exchange, tree automata and basic facts about them, and XML documents and DTDs.

## 2.1 Notation

Let us be clear about notation from mathematics and logic. We denote by $[n]$ the set $\{1, \ldots, n\}$. A *tuple* $\bar{a} \in A^n$ from a set $A$ is identified with a function from $[n]$ to $A$. In general we use $\bar{a}$ to mention a tuple of unspecified length. As a convention, $a, b, c, \ldots$ are used for constants, and $x, y, z, \ldots$ are used for variables.

A *relation* $R$ has its associated *arity*; when it is 1, $R$ is a *unary* relation; when it is 2, $R$ is a binary relation; in general when it is $k$, $R$ is a $k$-ary relation. A $k$-ary relation over a set $A$ is a subset of $A^k$. We use notation such as $D(\cdot)$ to mean $D$ is unary, $E(\cdot, \cdot)$ to mean $E$ is binary and so on.

## 2.2 Relational Data Exchange

### 2.2.1 Schema Mappings and Their Semantics

Before we define the formal semantics of relational schema mappings, we first explain what they are. The presentation of this section largely follows the survey by Kolaitis [66].

A *relational schema* $\mathbf{R}$ is a sequence $\langle R_1, \ldots, R_n \rangle$ of relational symbols, where each $R_i$ has its arity $a_i$. An *instance* $I$ of a relational schema $\mathbf{R}$ is a sequence $\langle R_1^I, \ldots, R_n^I \rangle$ of relations, where $R_i^I$ is a (finite) relation of the same arity as $R_i$. Given two mutually disjoint relational schema $S = \langle S_1, \ldots, S_n \rangle$ and $T = \langle T_1, \ldots, T_m \rangle$, we write $K = \langle I, J \rangle$ for the

instance over $\langle S_1, \ldots, S_n, T_1, \ldots, T_m \rangle$ such that $S_i^K = S_i^I$ for $i \in [n]$ and $T_j^K = T_j^J$ for $j \in [m]$. When it clear from the context, we drop superscript in $R^I$ and confuse a relation with the corresponding symbol. Also we sometimes denote by $I$ the domain of $I$, the set of elements appearing in relations in $I$.

The last component of a relational schema mapping is the specification of how instances over two relational schemas should be related. These specifications are called *dependencies*, or *constraints*. Since a relational database can be seen as a logical structure, we can naturally use some logic to specify them. In particular *conjunctive queries* are used and dependencies are classified as follows.

Let **S** and **T** be two disjoint relational schemas.

- A *source-to-target tuple-generating dependency* (s-t tgd) is a first-order formula of the form
$$\forall \bar{x} \forall \bar{y} (\varphi(\bar{x}, \bar{y}) \to \exists \bar{z} \psi(\bar{x}, \bar{z})),$$
where $\varphi(\bar{x}, \bar{y})$ is a conjunction of atoms with relation symbols from **S**[1] and $\psi(\bar{x}, \bar{z})$ a conjunction of atoms with relation symbols from **T**.

- A *target tuple-generating dependency* (target tgd) is a first-order formula of the same form as above, where $\varphi$ and $\psi$ are both conjunctions of atoms with relation symbols from **T**.

- A *target equality-generating dependency* (target egd) is a first-order formula of the form
$$\forall \bar{x} \varphi(\bar{x}) \to (x_i = x_j),$$
where $\varphi$ is a conjunction of atoms with relation symbols from **T**.

**Definition 2.1** A relational schema mapping[2] is a triple $\langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$, where **S** and **T** are relational schemas, called the *source schema* and the *target schema*, respectively, $\Sigma_{st}$ is a set of source-to-target dependencies (stds), and $\Sigma_t$ is the set of target (tuple- or equality-generating) dependencies.

The following is a natural definition of "two instances being related according to a schema mapping". We assume two infinite sets: Const is the set of *constants*; Var is the set of *nulls*. We use $\perp, \perp_1, \ldots$ to denote nulls.

**Definition 2.2** Let $M = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$ be a relational schema mapping.

[1] $\bar{x}$ denotes a sequence of variables $x_1, \ldots, x_n$.
[2] This is also called a *data exchange setting*.

- An *instance* of $M$ is an instance $\langle I, J\rangle$ (written $\langle I, J\rangle \vDash M$) where
  - $I$ is an instance over $\mathbf{S}$, which is called the source instance and whose domain is contained in $\mathsf{Const}$;
  - $J$ is an instance over $\mathbf{T}$, which is called the target instance and whose domain is contained in $\mathsf{Const} \cup \mathsf{Var}$;
  - For any s-t tgd of the form $\forall \bar{x} \forall \bar{y} \exists \bar{z} \varphi(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{z})$, and any tuples $\bar{a}, \bar{b}$ (from $\mathsf{Const}$) such that $I \vDash \varphi(\bar{a}, \bar{b})$, there exist a tuple $\bar{c}$ (from $\mathsf{Const} \cup \mathsf{Var}$) such that $J \vDash \psi(\bar{a}, \bar{c})$;
  - For any target dependency $\delta$, $J \vDash \delta$.
- $[\![M]\!]$ denotes the set of all instances of $M$. In symbols, $[\![M]\!] = \{\langle I, J\rangle \mid \langle I, J\rangle \vDash M\}$;
- Given an instance $I$ over $\mathbf{S}$, we say that an instance $J$ over $\mathbf{T}$ is a *solution for $I$ under $M$* if $\langle I, J\rangle \in [\![M]\!]$;
- For an instance $I$ over $\mathbf{S}$, $\mathrm{Sol}_M(I)$ denotes the set of all solutions for $I$ under $M$. In symbols, $\mathrm{Sol}_M(I) = \{J \mid \langle I, J\rangle \in [\![M]\!]\}$.

**Example** Consider two schemas $\mathbf{S} = \langle E(\cdot,\cdot)\rangle$ and $T = \langle D(\cdot)\rangle$. Let $I = \{E(1,2)\}$ be an instance over $\mathbf{S}$ and $M = \langle \mathbf{S}, \mathbf{T}, \{\forall xy E(x,y) \rightarrow D(x), \forall xy E(x,y) \rightarrow D(y)\}$. Then $J = \{D(1), D(2)\}$ is the solution for $I$ under $M$. Note that any instance containing $J$ (as a substructure) is a solution as well. For example, $J' = \{D(1), D(2), D(3)\}$ or $J'' = \{D(1), D(2), D(\bot)\}$ is just as perfect a solution as $J$ is.

### 2.2.2 Good Solutions and Query Answering

One of the most fundamental problems in data exchange is query answering, in which we have a source instance and a query over the target schema. As seen in the example in the preceding section, a solution is not unique given a source instance. This raises the natural questions: 1) what does it mean to answer a query in data exchange?; 2) which solution is better, or appropriate to materialise? In this section we briefly review the answers to these questions.

#### Universal solutions and cores

We start by answering the second question. Recall the example at the end of the preceding subsection. An instance $J = \{D(1), D(2)\}$ is a solution for $E(1,2)$ under the mapping

$\{\forall xyE(x,y) \to D(x), \forall xyE(x,y) \to D(y)\}$. Also solutions are $J' = \{D(1), D(2), D(3)\}$ and $J'' = \{D(1), D(2), D(\bot)\}$. These may look trivial and $J$ would look better than the others. But why is this? It is natural to say $J$ is in some sense cleaner - or minimal than the others.

In [45], this intuitive notion of "good" solutions are defined through homomorphisms, a standard algebraic notion. Formally:

**Definition 2.3 (homomorphism)** Let $J$ and $J'$ be instances over $\mathbf{T}$ whose domains are contained in $\mathsf{Const} \cup \mathrm{Var}$. A *homomorphism* from $J$ to $J'$ is a mapping $h : J \to J'$ which satisfies the following:

- $h$ is the identity map on $\mathsf{Const}$, i.e., $h(c) = c$ for all $c \in \mathsf{Const}$;
- $h$ preserves relations, i.e., $(a_1, \ldots, a_n) \in R^J$ implies $(h(a_1), \ldots, h(a_n)) \in R^{J'}$ for any $n$-ary relation $R \in \mathbf{T}$.

**Definition 2.4 (universal solution ([45]))** Let $M$ be a relational schema mapping and $S$ a source instance for $M$. A solution $T$ for $S$ under $M$ is a universal solution if for every solution $T' \in \mathrm{Sol}_M(S)$ there exists a homomorphism from $T$ to $T'$.

This explains why $J$ is better than $J'$ in the example above. But note that universal solutions are not unique. Again in the example above, $J''$ is also a universal solution (by a mapping that sends $\bot$ to 1 or 2). Fagin et al. argued that the smallest universal solutions are best ones. They formalised it using a notion from graph theory:

**Definition 2.5 (core ([47, 58]))** Let $T$ be an instance over $\mathbf{T}$. An instance $C \subseteq T$ is a *core* of $T$, if there is a homomorphism $h : T \to C$ and there is no homomorphism from $T$ to any proper substructure $C'$ of $C$.

A core of a graph is unique (up to isomorphism). It is also known that a core of a universal solution is again a universal solution, ensuring that the core of a universal solution is indeed the smallest universal solution.

Before we define the query answering problem, it would be natural to mention the problem asking the existence of a solution. Let $M$ be a relational schema mapping.

| | |
|---|---|
| PROBLEM: | EXISTENCE-OF-SOLUTION$(M)$ |
| INPUT: | A source instance $S$ |
| QUESTION: | Is $\mathrm{Sol}_M(S) = \varnothing$? |

The problem in general is undecidable:

**Theorem 2.6 ([67])** *There is a relational schema mapping $M_u = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t \rangle$ such that* EXISTENCE-OF-SOLUTION$(M_u)$ *is undecidable.*

A closely related problem is that of checking if there exists a universal solution, which is also undecidable ([57]).

### Query Answering

In the query answering problem, we have a (fixed) query over target schema and a (fixed) schema mapping, and are given a source instance. Solutions are not unique, so what does that mean to answer a query? Note that target instance can be seen as incomplete databases, where it is generally agreed that the right semantics of query answering is that of *certain answers* (see [61, 68]). Fagin et al. [45] followed this tradition. Formally we define the set of certain answers of $Q$ with respect to $I$ under $M$, as follows:

$$\text{CERTAIN}_M(Q, I) = \bigcap \{ Q(J) \mid J \in \text{Sol}_M(I) \}.$$

Here $Q(J) = \{ \bar{a} \mid \bar{a} \text{ is a tuple from } J \text{ such that } J \vDash Q(\bar{a}) \}$. Also for a Boolean query $Q$, $\text{CERTAIN}_M(Q, I)$ is true if $Q$ is true for all the solutions $J \in \text{Sol}_M(I)$.

The central tool in computing solutions is the chase procedure, which was originally introduced as a procedure to solve the implication problems of data dependencies ([17, 75]; see also [1]). Since EXISTENCE-OF-SOLUTION is undecidable as mentioned previously, we cannot tell whether the chase procedure will terminate. Fortunately, there is a class of dependencies for which the chase will always terminate, and even better, in polynomial time. This class of is called *weakly acyclic*. We do not go into its definition, but just mention the following result.

**Theorem 2.7 ([47])** *Let $M = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a fixed relational schema mapping such that $\Sigma_t$ consists of a set of target egds and a weakly acyclic set of target tgds. Then there exists an algorithm that, given a source instance I, check if there is a solution for I and when there is, computes the core of the universal solutions for I.*

Any universal solution represent the certain answers in the sense that the following holds for any universal solution $J$ of $I$ under $M$:

$$\text{CERTAIN}_M(Q, I) = Q(J),$$

so that the above theorem provides a feasible algorithm for query answering.

A more sophisticated way to compute a core was investigated in [56], refining the algorithm in [47]. Recently ten Cate et al. introduce a novel way to compute a core with SQL queries [35]. Also a generalisation of weak acyclicity was looked at in [41].

### 2.2.3 Composing Schema Mappings

Say we are given two schema mappings such that the target of the one is the same as the source of the other, and we try to find one single schema mapping from the source of the one to the target of the other. This is what *composing schema mappings* is about. In the following, only s-t tgd is considered.

**Definition 2.8 ([48])** Let $M_{12} = \langle \mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \rangle$ and $M_{23} = \langle \mathbf{S_2}, \mathbf{S_3}, \Sigma_{23} \rangle$ be two schema mappings, where every pair of $S_i$'s has no relation symbol in common.

A schema mapping $M_{13} = \langle \mathbf{S}_1, \mathbf{S}_3, \Sigma_{13} \rangle$ is a *composition of $M_{12}$ and $M_{23}$* if for every $I_1$ over $\mathbf{S}_1$ and $I_3$ over $\mathbf{S}_3$, the following are equivalent:

- $\langle I_1, I_3 \rangle \vDash \Sigma_{13}$.
- There is some instance $I_2$ over $\mathbf{S}_2$ such that $\langle I_1, I_2 \rangle \vDash \Sigma_{12}$ and $\langle I_2, I_3 \rangle \vDash \Sigma_{23}$.

That is, the semantics of composition is set-theoretically defined: $[\![M_{13}]\!] = [\![M_{12}]\!] \circ [\![M_{23}]\!]$.

One computational problem of note is the *composition query* of $M_{12}$ and $M_{23}$:

Given two instances $I_1$ and $I_3$, is $\langle I_1, I_3 \rangle$ in $[\![M_{12}]\!] \circ [\![M_{23}]\!]$?

This was shown to be NP-complete in [48].

A more fundamental question in composing relational mappings is whether a class of relational mappings is *closed under composition*: Given two schema mappings, is their composition always definable? In general, the answer is no as shown in [48]. There is settings such that there is no finite set of dependencies that define their composition. It is also known that when restricting attention to the s-t tgds of the form $\forall \bar{x}(\varphi(\bar{x}) \to \psi(\bar{x}))$ (called *full tgd*), the composition is always definable.

Now we want to find more expressive language so that every composition of two settings is always defined in the language. In [48], *second-order tuple-generating dependencies* (SO tgd) are proposed.

**Definition 2.9** Let $\mathbf{S}$ and $\mathbf{T}$ be two disjoint relational schemas. A *second-order tuple-generating dependency over* $\langle \mathbf{S}, \mathbf{T} \rangle$ is a formula of the form

$$\exists f_1 \cdots \exists f_m (\forall \bar{x_1}(\varphi_1 \to \psi_1) \wedge \cdots \wedge \forall \bar{x_n}(\varphi_n \to \psi_n)),$$

where

- Each $f_i$ is a function symbol;
- Each $\varphi_i$ is a conjunction of
  - atomic formulae $R(y_1, \ldots, y_k)$, where $R$ is a $k$-ary relation symbol in $\mathbf{S}$ and $y_i$'s are variables in $\boldsymbol{x_i}$, not necessarily distinct, and
  - equalities of the form $t = t'$, where $t$ and $t'$ are terms built from $f_i$'s and variables in $\bar{x_j}$;
- Each $\psi_i$ is a conjunction of atomic formulae $S(t_1, \ldots, t_l)$, where $S$ is an $l$-ary relation symbol from $\mathbf{T}$ and $t_i$'s are terms built from $f_i$'s and variables in $x_j$;
- Each variable in $\bar{x_i}$ appears in a relational atomic formula of $\varphi_i$.

The following result from [48] shows that SO tgd is closed under composition:

**Theorem 2.10** *If* $M_{12} = \langle \mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \rangle$ *and* $M_{23} = \langle \mathbf{S}_2, \mathbf{S}_3, \Sigma_{23} \rangle$ *are two schema mappings with* $\Sigma_{12}$ *and* $\Sigma_{23}$ *being SO tgds, then there is an SO tgd* $\Sigma_{13}$ *such that* $[\![M_{12}]\!] \circ [\![M_{23}]\!] = [\![\langle \mathbf{S}_1, \mathbf{S}_3, \Sigma_{13} \rangle]\!]$.

## 2.3 Tree Automata

In this section we recall basic facts about (tree) automata.

We use the following notation for string automata and languages: DFA and NFA stand for deterministic automata and nondeterministic automata, respectively. For an automaton $A$, $L(A)$ denotes the language accepted by $A$; Similarly, $L(r)$ denotes the language matching a regular expression $r$. Recall that given a regular expression $r$, an NFA $A_r$ with $L(A_r) = L(r)$ can be constructed in polynomial time. For general background in (string) language theory, see [60, 93].

A set of trees can be *ranked* or *unraked*. The difference is whether there is a priori bound on the number of children of each node. We first define automata over ranked alphabet. Suppose $\Gamma$ is a finite set of alphabet. For each $\sigma \in \Gamma$, $\mathrm{rank}(\sigma)$ is its *rank*, i.e., the number of children that a node labelled $\sigma$ should have. A $k$-ary tree is a tree the rank of whose alphabet is at most $k$. We say a tree is binary if $k \leq 2$.

A nondeterministic (bottom-up) finite tree automaton (NFTA) on binary trees over alphabet $\Gamma$ is a quadruple $A = \langle Q, q_0, \delta, F \rangle$, where

- $Q$ is the set of states;
- $q_0 \in Q$ is the initial state;
- $\delta : \Gamma \times Q \times Q \to 2^Q$ is the transition function;
- $F \subseteq Q$ is the set of final states.

Given a (binary) tree $T$ whose nodes are labelled alphabets from $\Gamma$, a run of $A$ on $T$ is a function $\rho_A : T \to Q$ that assigns a state to each node in $T$ and satisfies the following:

- For a leaf node $s \in T$ labelled $a$, $\rho_A(s) \in \delta(a, q_0, q_0)$;
- For a non-leaf node $s \in T$ that is labelled $a$ and have children $s_1, s_2$,

$$\rho_A(s) \in \delta(a, \rho_A(s_1), \rho_A(s_2)).$$

A tree $T$ is accepted by $A$ if there is a run $\rho_A$ such that $\rho(r) \in F$ for the root node $r \in T$. Like string languages, $L(A)$ denotes the set of trees, or the tree language, accepted by $A$. In symbols $L(A) = \{T \mid T \text{ is accepted by } A.\}$.

The basic computational problems mentioned in this thesis are the following:

- EMPTINESS: Given an NFTA $A$, is $L(A) = \varnothing$?
- UNIVERSALITY: Given an NFTA $A$, is there a tree that is *not* accepted by $A$?

The EMPTINESS problems is known to be solvable in time linear in the size of $A$ ([38]); The UNIVERSALITY problem is known to be EXPTIME-complete ([91]).

We turn to automata models over unranked trees. Unranked trees over a finite alphabet $\Gamma$ are the same as ranked trees except that each letter in $\Gamma$ does not have rank: this means that a node in an unraked tree can have arbitrary many (albeit finitely many) children. An unranked nondeterministic finite tree automaton (UNFTA) on ordered unranked trees[3] is a triple $A = \langle Q, \delta, F \rangle$ where

- $Q$ is the set of states;
- $\delta : Q \times \Gamma \to 2^{Q^*}$ is the transition function such that $\delta(q, a)$ is a regular (string) language over the alphabet $Q$ for any $q \in Q$ and $a \in \Gamma$;

[3]Note that for unranked tree automata there is no real distinction between being bottom-up and being top-down. That is, it only depends how the set $F$ is construed: if $F$ is seen as "final" then the automaton is bottom-up; if it is seen as "initial" then it is top-down.

- $F \subseteq Q$ is the set of final states.

Given an ordered unranked tree $T$, a run of $A$ on $T$ is a function $\rho_A : T \to Q$ that assigns states to nodes and satisfies the following conditions:

- For a leaf node $s \in T$ labelled $a$, $\epsilon \in \delta(q, a)$;
- For a non-leaf node $s \in T$ that is labelled $a$ and have children $s_1, \ldots, s_n$ (in this order), $\rho_A(s_1) \cdots \rho_A(s_n) \in \delta(q, a)$.

A tree $T$ is accepted by $A$ if there is a run $\rho_A$ such that $\rho_A(r) \in F$ for the root $r$ of $T$. The complexity results on ranked tree automata can be transferred to unranked automata: EMPTINESS is linear and UNIVERSALITY is EXPTIME-complete.

A standard representation of UNFTA uses NFAs for transitions: $\delta$ maps each pair of a state and a letter to an NFA over $Q$. It is known that testing nonemptiness can be done in polynomial time (see [83]).

For more comprehensive information on tree automata see [38].

## 2.4 XML Documents and DTDs

We view XML documents as unranked trees. Each node has an element type and may also have *attribute values* associated with *attribute names*. We assume the following infinite sets are given:

- ELEMENT: a set of infinite *element types*;
- ATTRIBUTE: a set of *attribute names*[4];
- $V$: a set of possible attribute values (which are usually strings).

Throughout the following, unless otherwise specified, we always assume $\Gamma$ and *Att* are finite subsets of ELEMENT and ATTRIBUTE, respectively.

Formally, an *XML document* over a finite labelling alphabet $\Gamma$ (element types) and a finite set of attribute names *Att* is a structure $\mathbf{T} = \langle T, \downarrow, \rightarrow, lab, (\rho_a)_{a \in Att} \rangle$, where

- The set $T$ is an unranked tree domain, i.e., a subset of $\mathbb{N}^*$ such that $n \cdot i \in T$ implies $n \cdot j \in T$ for all $j < i$;

[4] In the following attribute names are prefixed by @.

- The binary relations $\downarrow$ and $\rightarrow$ are child relation ($n \downarrow n \cdot i$) and next-sibling relation ($n \cdot i \rightarrow n \cdot (i+1)$);
- The function *lab* is a labelling from $T$ to $\Gamma$;
- Each $\rho_a$ is a partial function from $T$ to $V$. We say that a node $s \in T$ has the value $v$ for the attribute $@a$ when $\rho_a(s) = v$.

Most often we refer to XML documents as trees and write $T$ instead of $\mathbf{T}$.

A *document type definition* (DTD) over a labelling alphabet $\Gamma$ and a set of attributes *Att* is a triple $D = \langle r, P_D, A_D \rangle$, where

- $r$ is a distinguished root symbol;
- $P_D$ is a function assigning regular expressions over $\Gamma - \{r\}$ to the elements of $\Gamma$, usually written as $\ell \rightarrow e$;
- $A_D$ is a function from $\Gamma$ to $2^{Att}$ which assigns attribute names to each element type.

For notational simplicity we assume that attribute names come in some order and omit attribute names, just as in relational case where attribute names for a relation $R$ is ordered in some way so that we can write $R(a_1, \ldots, a_n)$. Similarly, we describe a node that is labelled $\ell$ and has $n$ attributes with values $a_1, \ldots, a_n$ as $\ell(a_1, \ldots, a_n)$.

A tree $T$ *conforms to* a DTD $D$ if its root is labelled with $r$ and for each node $s \in T$ with $lab(s) = \ell$ it holds that

- $\rho_a(s)$ is defined iff $@a \in A_D(\ell)$,
- the sequence of labels of children of $s$ is in the language $P_D(\ell)$.

DTDs (without attributes) can be naturally represented by tree automata. A DTD $D$ over a set $E$ of element types is represented by a UNFTA $A_D$ where the set of states is $E$, $\delta(\ell, \ell)$ is an NFA for $P_D(\ell)$, and $\delta(\ell, \ell') = \varnothing$ for all $\ell' \neq \ell$, and $F = \{r\}$.

**Remark** What we use here is a common abstraction of DTD. DTD is not the absolute standard of XML schema language, but only one of many. There are classification and characterisations of other XML schema languages like XML Schema and Relax NG [76, 81].

The presence of DTDs in general makes things difficult, as witnessed for example by [16, 18]. We pay special attention to the class of *nested relational* DTDs, a generalisation of

nested relations. A DTD is nested relational if it is non-recursive (i.e., the graph in which we put edges between $\ell$ and the element types in $P_D(\ell)$ does not contain cycles) and all its productions are of the form $\ell \to \hat{\ell}_1 \cdots \hat{\ell}_m$, where $\ell_i$'s are distinct elements of $\Gamma$ and $\hat{\ell}_i$ is one of $\ell_i$, $\ell_i^*$, $\ell_i^+$, $\ell_i?$[5]. Such DTDs are quite common in practice and account for 70% of real-word DTDs according to one study [23].

[5] where we define $\ell^+ == \ell_i \ell_i^*$ and $\ell_i? = \ell_i | \epsilon$

# Chapter 3

# Static Analysis in XML Data Exchange

We now begin to describe what is an XML schema mapping and investigate problems concerning a single mapping. Recall that a relational schema mapping is a quadruple $\langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$, where $\mathbf{S}$ and $\mathbf{T}$ are relational schemas (called the source schema and the target schema, respectively), $\Sigma$ is a set of dependencies.

In XML context, DTDs replace relational schemas naturally. On the other hand, using conjunctive queries as the specification language is cumbersome because we have two sorts of objects in an XML database: tree nodes and data values. Instead, we use *tree patterns*, first introduced in [12].

After defining tree patterns, we explore basic patterns. Then we go on to explore two types of consistency problem, which we call (plain) consistency and absolute consistency. The former is a natural analogue of traditional static analysis in query languages, but since our problem involves two structures, the solution requires significantly different insights. The latter problem is a more restrictive property of a mapping. A motivation comes from the topic of the next chapter: composition.

## 3.1 Tree Patterns

We define *extended tree patterns* as given by the following grammar. The main difference from the original pattern language [12] is that we can specify sequences, i.e., use horizontal ordering.

$$\begin{array}{lcll} \pi & := & \ell(\bar{x})[\lambda] & \text{patterns} \\ \lambda & := & \epsilon \mid \mu \mid //\pi \mid \lambda, \lambda & \text{sets} \\ \mu & := & \pi \mid \pi \to \mu \mid \pi \to^* \mu & \text{sequences} \end{array} \tag{3.1}$$

Here $\ell$ is a label, $\bar{x}$ is a sequence of variables, and $\epsilon$ is the empty sequence. Futhermore we allow the wildcard _, matching any label. We use $\ell/\pi$ as an abbreviation for $\ell[\pi]$. We call $\ell(\bar{x})$ an attribute formula.

A tree $T$ *satisfies a tree pattern* $\varphi$ at a node $s$, written $(T, s) \vDash \varphi$, iff the following conditions hold:

| | | |
|---|---|---|
| $(T,s) \vDash \_$ | iff | true |
| $(T,s) \vDash \ell(\bar{a})$ | iff | $s$ is labelled by $\ell$ and $\bar{a}$ is the tuple of attributes of $s$; |
| $(T,s) \vDash \ell(\bar{a})[\lambda_1, \lambda_2]$ | iff | $(T,s) \vDash \ell(\bar{a})[\lambda_1]$ and $(T,s) \vDash \ell(\bar{a})[\lambda_2]$; |
| $(T,s) \vDash \ell(\bar{a})[\mu]$ | iff | $(T,s) \vDash \ell(\bar{a})$ and $(T,s') \vDash \mu$ for some $s'$ with $s \downarrow s'$; |
| $(T,s) \vDash \ell(\bar{a})[//\pi]$ | iff | $(T,s) \vDash \ell(\bar{a})$ and $(T,s') \vDash \pi$ for some descendant $s'$ of $s$; |
| $(T,s) \vDash \pi \to \mu$ | iff | $(T,s) \vDash \pi$ and $(T,s') \vDash \mu$ for some $s'$ with $s \to s'$; |
| $(T,s) \vDash \pi \to^* \mu$ | iff | $(T,s) \vDash \pi$ and $(T,s') \vDash \mu$ for some younger sibling $s'$ of $s$; |

We write $T \vDash \varphi$ for $(T, \epsilon) \vDash \varphi$.

Observe that semantically 'sets' in tree patterns are literally sets: for a node satisfying $\ell(\bar{a})[\lambda_1, \lambda_2]$, the child witnessing $\lambda_1$ is not necessarily distinct from the one witnessing $\lambda_2$. If we remove the sequences from the definition above, we obtain the language used in [12].

A *source-to-target dependency* (*std*) is an expression of the form

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

where $\pi, \pi'$ are tree patterns in which no variable appears more than once, and $\alpha_{=,\neq}$, $\alpha'_{=,\neq}$ are sets of equalities and inequalities.

A pair of trees $\langle T, T' \rangle$ *satisfies an std* of the form above if for each tuple $\bar{a}, \bar{b}$ such that $T \vDash \pi(\bar{a}, \bar{b})$ and $\alpha(\bar{a}, \bar{b})$ holds, there exists a tuple $\bar{c}$ such that $T' \vDash \pi'(\bar{a}, \bar{c})$ and $\alpha'(\bar{a}, \bar{c})$ holds.

An *XML schema mapping* is a triple $M = \langle D_s, D_t, \Sigma \rangle$, where

- $D_s$ is the source DTD, $D_t$ is the target DTD,
- $\Sigma$ is a set of stds.

Given a tree $T$ conforming to $D_s$, a *solution* for $T$ under $M$ is a tree $T'$ such that

- $T'$ conforms to $D_t$;
- $\langle T, T' \rangle$ satisfies all the stds in $\Sigma$ (written $\langle T, T' \rangle \models \Sigma$).

We also pay special attention to a restricted set of stds, called *fully-specified* stds. Patterns for fully-specified stds are given by the grammar:

$$\begin{array}{lcll} \pi & := & \ell(\bar{x})[\lambda], & \text{where } \ell \in \mathcal{L} \\ \lambda & := & \epsilon \mid \pi \mid \lambda, \lambda & \end{array} \tag{3.2}$$

In other words, (3.2) disallows wildcard and descendant compared to (3.1).

As seen later in this chapter and the following chapters, the restriction of stds to fully-specified ones leads to a tractable subclass of the problem.

### 3.1.1 An Example of XML Schema Mapping

In this subsection we give a concrete example of XML schema mapping, to clarify the definition above.

Consider the following DTD, and a tree $T_1$ conforming to it depicted in Figure 1.

| | |
|---|---|
| $\texttt{novels} \to \texttt{novel}^*$ | |
| $\texttt{novel} \to \texttt{author}\,\texttt{year}$ | $\texttt{novel}\colon \texttt{@name}$ |
| $\texttt{author} \to \epsilon$ | $\texttt{author}\colon \texttt{@name}$ |
| $\texttt{year} \to \epsilon$ | $\texttt{year}\colon \texttt{@year}$ |

Let us consider the following mapping. The dependency takes a pair of novels by the same author, and put the earlier novel (assuming the original database follows the chronological order as in figure 1) with the child node indicating what is the next novel by the author.

$$\texttt{novels}[\texttt{novel}(x)/\texttt{author}(y) \to \texttt{novel}(z)/\texttt{author}(y)]$$

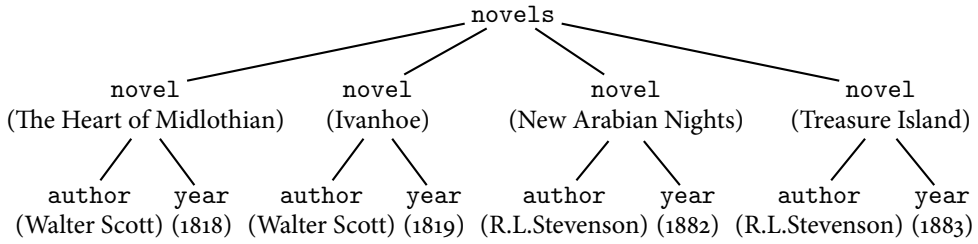



FIGURE 1. a small Scottish literature library

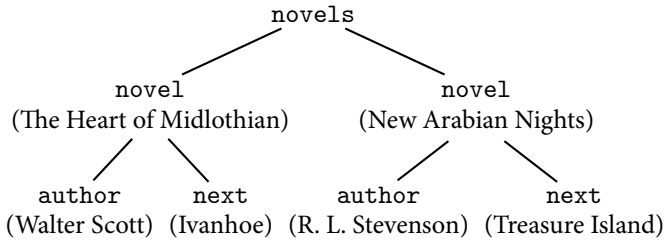



FIGURE 2. a small Scottish literature library, transformed

$$\rightarrow \texttt{novels}/\texttt{novel}(x)[\texttt{author}(y), \texttt{next}(z)]$$

A solution for $T_1$ is depicted in Figure 2.

Note that the semantics of data exchange specifies only the lowerbound and cannot specify the upperbound, in the sense that it specifies what there should be but not what there should *not* be. Thus the tree in Figure 3 is another solution for $T_1$.

### 3.1.2 Classification of schema mappings

In general, mappings can use vertical and horizontal navigation as well as data comparisons. For restricted classes of schema mappings we use the following notation. For a subset $\sigma \subseteq \{\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, =, \neq\}$, we write $\mathrm{SM}(\sigma)$ to denote the class of schema mappings

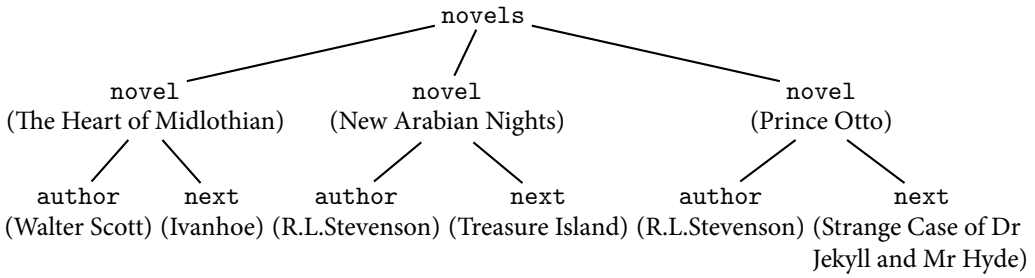



FIGURE 3. a small Scottish literature library: another solution

in which stds use only the operations from $\sigma$.

A special attention is paid to the class of *nested relational schema mappings*, i.e., schema mappings with nested relational DTDs. By $\mathrm{SM}^{nr}(\sigma)$ we denote the class of nested relational schema mappings in $\mathrm{SM}(\sigma)$. The exemplary mapping described above is in $\mathrm{SM}^{nr}(\downarrow, \rightarrow)$.

To simplify notations, we use abbreviations:

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

Under these notations, $\mathrm{SM}(\Downarrow)$ is the precisely the class of mappings studied in [12] (as in [12], we do not restrict variable reuse in target patterns).

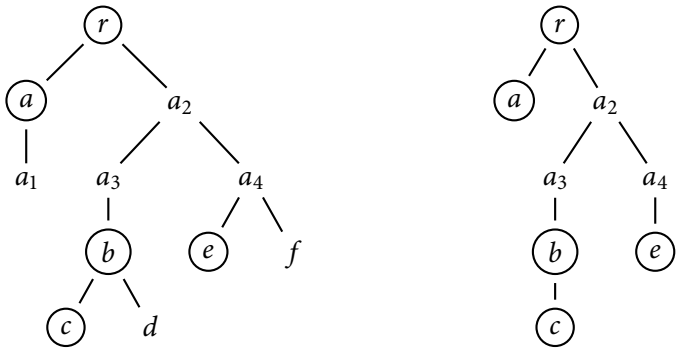
## 3.2 Complexity of Tree Patterns

We first look at some basic properties related to satisfiability of patterns, the complexity of their evaluation, both the data and the combined complexity of it.

The first problem is the satisfiability for tree patterns. Its input consists of a DTD $D$ and a pattern $\pi(\bar{x})$; the problem is to check whether there is a tree $T$ that conforms to $D$ and has a match for $\pi$ (i.e., $\pi(T) \neq \varnothing$). 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.,[8, 19, 25, 59]). For the sake of completeness we give a simple proof, which applies to patterns in the way they are defined here.

**Lemma 3.1** *The satisfiability problem for tree patterns in NP-complete.*

*Proof.* We start by introducing two concepts that will be used in several proofs. By dom $T$ we denote domain of the tree $T$, and by $T(v)$ we understand the label of the node $v$. A *realisation* of a tree pattern $\varphi$ in a tree $T$ is a consistent assignment of a node to each subformula of $\varphi$. Each realisation $\xi$ of $\varphi$ yields exactly one valuation of $\varphi$'s variables; we denote it by $\mathrm{val}(\xi)$. Clearly, $T \vDash \varphi[a]$ iff there exists a realisation $\xi$ of $\varphi$ in $T$ such that $\mathrm{val}(\xi) = a$. Analogously, we can define realisation for a formula and a partial valuation of its variables. Suppose $T \vDash \varphi$ and let $\xi$ be a realisation of $\varphi$ in $T$. Let $Z = \{v \in \mathrm{dom}\, T \colon \exists w\ vw \in \mathrm{im}\, \xi\}$, and let $T'$ be obtained from $T$ by restricting the domain to $Z$. We will call $T'$ the *support* of $\xi$ and denote it $\mathrm{supp}\, \xi$.




(a) A realisation in a tree (b) The support of the realisation

FIGURE 4. A realisation of $r[//a, //b/c, //e]$ in a tree and its support

For example, consider a tree pattern $\varphi = r[//a, //b/c, //e]$. This pattern is satisfied by a tree $T$ given in Figure 3.4(a) with the obvious assignment $\xi$ which appropriately assigns subformulae to the encircled nodes. To obtain supp $\xi$, we remove the nodes that are "beyond" im $\xi$. The result is shown in Figure 3.4(b).

First, let us see that for each $\varphi$ satisfiable with respect to a DTD $D$ over $\Gamma$, there exists a realisation with $\mathcal{O}(\|\varphi\| \cdot |\Gamma|)$-support. Take a tree $T$ conforming to $D$ and satisfying $\varphi$. Let $\xi$ be a realisation of $\varphi$ in $T$. Divide the nodes of supp $\xi$ into three categories: the nodes from the image of $\xi$'s are *red*, the nodes which are not red and have more then one child are *green*, the others are *blue*. For example, in Figure 3.4(b) the encircled nodes are red, $a_2$ is green, $a_3$, $a_4$ are blue. Let $N_{\text{red}}$, $N_{\text{green}}$, and $N_{\text{blue}}$ be the numbers of red, green, 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 blue paths in supp $\xi$ are not longer than $|\Gamma|$. The number of maximal blue paths is at most $N_{\text{red}} + N_{\text{green}}$. Hence there are at most $2\|\varphi\| \cdot |\Gamma|$ blue nodes. All together we have at most $2\|\varphi\| \cdot (|\Gamma| + 1)$ nodes.

Now, to decide satisfiability, first guess a polynomial support and a realisation. Verifying the realisation 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 $D$ requires a consistency check which amounts to deciding if a given word $w$ is a subsequence of a word from the language defined by a given regular expression. The latter can

be done in polynomial time. Translate the regular expression into a non-deterministic automaton $A$. Add epsilon transition from $p$ to $q$ whenever $q$ is reachable from $p$. The modified automaton accepts subsequences of words accepted by $A$.

To get NP-completeness we do a standard 3CNF SAT reduction. In fact, we will only use $\mathrm{SM}^\circ(\downarrow)$. Take a formula $\varphi = \bigwedge_{j=1}^{k} Z_j^1 \vee Z_j^2 \vee Z_j^3$ 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 $D$

$$
\begin{aligned}
& r \to x_1 x_2 \cdots x_n \\
& x_i \to \{C_j \mid \exists_\ell Z_j^\ell = x_i\} | \{C_j \mid \exists_\ell Z_j^\ell = \bar{x}_i\} \qquad\qquad 1 \le i \le n
\end{aligned}
$$

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 the given 3SAT formula $\varphi$ is equivalent to the satisfiability of a formula $r[\_/C_1, \_/C_2, \ldots, \_/C_k]$ with respect to $D$. □

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

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

**Proposition 3.2** *The data complexity of evaluating tree patterns is DLOGSPACE-complete, and the combined complexity is in PTIME.*

*Proof.* Take a tree pattern $\pi$, a valuation $\bar{a}$ and a tree $T$. Checking that $T \vDash \pi[\bar{a}]$ can be done in PTIME by a bottom up evaluation of the sub-patterns of $\pi[\bar{a}]$. Annotate each node $v$ with a set $\Phi(v)$ containing those subformulae of $\pi[\bar{a}]$ which are satisfied in $v$. If $v$ is a leaf labelled with $\sigma$ and storing a tuple $\bar{b}$, let $\Phi(v)$ contain all sub-patterns of $\pi[\bar{a}]$ of the form $\sigma'(\bar{b})$, with $\sigma' \in \{\sigma, \_\}$. If $v$ is an internal node labelled with $\sigma$, having children $v_1, v_2, \ldots, v_k$, and storing a tuple $b$, 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 $\rightsquigarrow_j = \rightarrow$ then $n_{j+1} = n_j + 1$ for all $j$,

and all sub-patterns of $\pi[\bar{a}]$ of the form $//\pi_1$ satisfying $\pi_1 \in \Phi(v_j)$ or $//\pi_1 \in \Phi(v_j)$ for some $j$. $T \vDash \pi[\bar{a}]$ 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 $T \vDash \pi[\bar{a}]$ for some fixed pattern $\pi$. In order to store a realisation of $\pi[\bar{a}]$ in $T$, we need only logarithmic space: a fixed number of "pointers" to the tree. Verifying a given realisation amounts to reachability tests in the tree plus local consistency checks, which can be done in DLOGSPACE. Hence, if only we can do the reachability test in DLOGSPACE, we can get a DLOGSPACE algorithm by simply iterating over all possible realisations 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 younger-sibling relations in a tree in DLOGSPACE since reachability is known to be in DLOGSPACE 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, DLOGSPACE-hardness follows from the hardness of reachability over successor-relations [86]; hence even evaluating $r[a \rightarrow^* b]$ over a tree of depth 1 is DLOGSPACE-hard. □

## 3.3 Consistency

We start by analysing consistency. As its name suggests, this is essentially our version of satisfiability. But the difference from classical versions is that we have to reason about two structures. Also our tree pattern language is fairly positive, not allowing use of negation. Because of these the proof cannot be a simple adaptation of the known results, for example, in XPath or $FO^2$.

We say that a mapping is *consistent* if $\mathrm{Mod}(M) \neq \varnothing$; that is, if $\mathrm{Sol}_M(T) \neq \varnothing$ for some $T \vDash D_s$. The main problem we consider is the following:

| | |
|---|---|
| PROBLEM: | $\text{CONS}(\sigma)$ |
| INPUT: | A mapping $M = (D_s, D_t, \Sigma) \in \text{SM}(\sigma)$ |
| QUESTION: | Is $M$ consistent? |

The important observation made in [12] was that data values can be ignored when dealing with downward mappings. We define $\alpha^\circ$ for an attribute formula $\alpha$ as follows.

- $\_^\circ = \_$;
- $\ell^\circ = \ell$;
- $\ell(x_1, \ldots, x_n)^\circ = \ell$.

We write $\varphi^\circ$ for the expression obtained by substituting $\alpha^\circ$ for every attribute formula $\alpha$ in $\varphi$, and $\Sigma^\circ$ is obtained by every $\psi \coloneq \varphi \in \Sigma$ replaced with $\psi^\circ \coloneq \varphi^\circ$.

**Fact 3.3 ([12])** *Both* $\text{CONS}(\Downarrow)$ *and* $\text{CONS}^\circ(\Downarrow)$ *are EXPTIME-complete. If we restrict to nested-relational DTDs in schema mappings, then* $\text{CONS}(\Downarrow)$ *is solvable in polynomial (cubic) time.*

First we show that in absence of data comparison, the complexity stays the same.

**Theorem 3.4** *The problem* $\text{CONS}(\Downarrow, \Rightarrow)$ *is solvable in EXPTIME (and thus it is EXPTIME-complete).*

Before proving the theorem, we separately state two lemmas. The original proof was for mapping/patterns using only downward navigation, but it can easily extended to accommodate horizontal navigation.

The following result shows that, without data comparisons, $\text{CONS}(\Downarrow, \Rightarrow)$ is no harder than $\text{CONS}^\circ(\Downarrow, \Rightarrow)$.

**Proposition 3.5 ([12])** *An XML schema mapping* $M = \langle D_1, D_2, \Sigma_{12} \rangle \in \text{SM}(\Downarrow, \Rightarrow)$ *is consistent if and only if* $\langle D_1, D_2, \Sigma_{12}^\circ \rangle$ *is consistent.*

Also important is that we can use automata to reason about tree patterns as long as it does not use variables.

**Proposition 3.6 ([12])** *There is an exponential time algorithm that , given a variable-free tree-pattern formula* $\varphi$*, constructs a deterministic tree automaton* $A(\varphi)$ *such that*

$$T \vDash \varphi \text{ iff } A(\varphi) \text{ accepts } T$$

*Note that we can construct the complementary automata in exponential time since it is deterministic.*

*Proof of Theorem 3.4.* For a mapping $\langle D_s, D_t, \Sigma\rangle$ to be consistent, there must exist a pair $\langle T_1, T_2\rangle$ such that for all $\varphi \to \psi \in \Sigma$ it holds that $T_1 \vDash \varphi$ implies $T_2 \vDash \psi$. Suppose $\Sigma = \{\varphi_i \to \psi_i \mid i = 1, 2, \ldots, n\}$. Then the existence of such a pair is equivalent to the existence of a subset $I \subseteq \{1, 2, \ldots, n\}$ satisfying

- there exists $T_1 \vDash D_s$ such that $T_1 \nvDash \varphi_j$ for all $j \notin I$,
- there exists $T_2 \vDash D_t$ such that $T_2 \vDash \psi_i$ for all $i \in I$.

This amounts to nonemptiness of the following automata:

- $A_{D_s} \times \prod_{j \notin I} \bar{A}(\varphi_j)$,
- $A_{D_t} \times \prod_{j \in I} A(\psi_j)$.

It is known (see [38]) that testing nonemptiness of $A_1 \times \cdots \times A_k$ can be done in time $\mathcal{O}(|A_1| \times \cdots \times |A_k|)$. Since the construction of each $A(\varphi)$ takes exponential time, the overall complexity is EXPTIME. □

The impact of horizontal ordering becomes apparent when we consider nested relational mappings. Unlike the downward fragment SM($\Downarrow$), we are not able to check consistency feasibly (unless P=PSPACE).

**Proposition 3.7** CONS($\Downarrow, \rightarrow$) *over nested relational DTDs is PSPACE-hard.*

*Proof.* We show PSPACE-hardness 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 the alphabet $\{r, \sharp, \natural, t_1, t_2, \ldots, t_n, f_1, f_2, \ldots, f_n\}$ as

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

The target DTD $D_t$ is simply $r \to \epsilon$. 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. Let $\perp$ be a tree pattern incompatible with the target DTD, $r[f]$ for instance. Consider the following constraints:

$$r//\_[\sharp \to \natural] \to \perp\,,$$
$$r//\_[t_i, f_i] \to \perp \quad \text{for all } i \le n \text{ such that } Q_i = \exists\,.$$

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

$$r//f_i//t_j//f_k \to \perp\,.$$

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

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 [27, 39, 51, 90]. Here we witness a similar situation. The proofs, however, cannot be simple adaptations of existing proofs which showed undecidability of such formalisms as $FO^3$ [27] or Boolean combinations of patterns with data value comparisons [39]. The reason is the very "positive" nature of stds in schema mappings: the use of negation is limited to the implication in stds, while known undecidable formalisms can use negation freely.

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

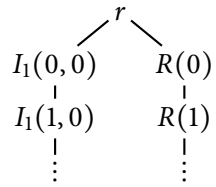
**Theorem 3.8** *The following problems are undecidable:*

- $\mathrm{CONS}(\downarrow^*, =)$;
- $\mathrm{CONS}(\downarrow^*, \neq)$;

- $\textsc{cons}(\rightarrow, =)$;
- $\textsc{cons}(\rightarrow, \neq)$.

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

*Proof.* We describe reduction from halting problem of 2-register machine, which is known to be undecidable ([80]). That is, 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 $\sharp$ and $*$. 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 stds 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 $\mathcal{I} = \langle I_i \mid i \in Q \setminus \{f\} \rangle$ (one instruction for each state apart from the last state $f$), and two registers $\mathsf{r}_1$ and $\mathsf{r}_2$, each containing a natural number. An instantaneous description (ID) of $M$ is a triple $\langle i, m, n \rangle$ where $i \in Q$ and $m, n \in \mathbb{N}$ are natural numbers stored in $\mathsf{r}_1$ and $\mathsf{r}_2$, respectively.

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

**increment** $I_i = \langle \mathsf{r}, j \rangle$, where $i \in Q$ and $\mathsf{r}$ is one of $\mathsf{r}_1$ and $\mathsf{r}_2$. This means that $M$ in state $i$ increments $\mathsf{r}$ and goes to state $j$:

$$\langle i, m, n \rangle \rightarrow_M \begin{cases} \langle j, m+1, n \rangle & \text{if } \mathsf{r} = \mathsf{r}_1\,, \\ \langle j, m, n+1 \rangle & \text{if } \mathsf{r} = \mathsf{r}_2\,. \end{cases}$$

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

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

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

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

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

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 simply $\{r \to \epsilon\}$. The stds $\Sigma$ are described below.

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 disallows two values to appear more than once.

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

Let us now deal with the left branch, which is meant to encode the run itself. We have assumed that the initial ID is $\langle 1, 0, 0\rangle$; the constraints below exclude other situations.

$$\begin{aligned} r[I_1(x, y), //\_/R(x)] &\to \bot \\ r[I_1(x, y), //\_/R(y)] &\to \bot \end{aligned}$$

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

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

For each $i$ such that $I_i = \langle \mathsf{r}_1, j, k\rangle$, we need to say: if the next state is $k$, then $\mathsf{r}_1$ stores 0, and both registers stay the same; if the next state is $j$, then $\mathsf{r}_1$ does not store 0, the register $\mathsf{r}_1$ gets decreased, and $\mathsf{r}_2$ stays the same.

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

For each $i$ such that $I_i = \langle \mathsf{r}_2, j\rangle$ or $I_i = \langle \mathsf{r}_2, j, k\rangle$ we add analogous stds.

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.

$$r[//I_i(x,y)/\sharp, //\_/R(x)] \to \bot$$
$$r[//I_i(x,y)/\sharp, //\_/R(y)] \to \bot$$

The obtained mapping $\langle D_s, D_t, \Sigma\rangle$ is consistent iff there is a halting run of the given 2-register machine.

Rotating the encoding by 90 degrees, we get undecidability of $\textsc{cons}(\to, \to^*, =)$. The source DTD is now defined as

$$r \to \sharp\, R^*\, \natural \{I_1, I_2, \ldots, I_f\}^*\, \flat$$

and the target DTD remains $r \to \varepsilon$. We have lost the restriction on consecutive labels $I_i$, so we need to enforce it with the stds:

$$r[I_i \to I_j] \to \bot \qquad \text{for all } i, j \text{ such that } I_i \notin \{\langle \mathsf{r}, j\rangle, \langle \mathsf{r}, j, k\rangle, \langle \mathsf{r}, k, j\rangle \mid k \in Q,\ \mathsf{r} = \mathsf{r}_1, \mathsf{r}_2\},$$
$$r[\natural \to I_j] \to \bot \qquad \text{for all } j \neq 1,$$
$$r[I_i \to \flat] \to \bot \qquad \text{for all } i \neq f.$$

After rotating, all stds apart from $//R(x)//R(x) \to \bot$ can be written naturally without $\to^*$. For instance, $r[I_1(x,y), //\_/R(x)] \to \bot$ becomes $r[\natural \to I_1(x,y), \_ \to \_ \to R(x)] \to \bot$, and $r[//I_i(x,y)/I_j, R(x)] \to \bot$ becomes $r[I_i(x,y) \to I_j, \sharp \to R(x)] \to \bot$. The std $//R(x)//R(x) \to \bot$ can be rewritten as $r[R(x) \to^* R(x)] \to \bot$.

In order to get undecidability of $\textsc{cons}(\to, =)$, we need to enforce the property above 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 having two consecutive data values equal (the first std), and switching from one data value to two different ones (the second std), or switching to a data value and the ending marker (the third std).

$$r[R(x) \to R(x)] \to \bot$$
$$r\big[R(x) \to R(x'), R(y) \to R(y')\big], x' = y' \to x = y$$
$$r[R(x) \to R, R(x) \to \sharp] \to \bot$$

These three conditions imply that a value cannot repeat in two consecutive nodes, and if it repeats in two distinct nonconsecutive nodes, the sequence of children between these two nodes has to repeat forever. The latter obviously cannot happen in a finite tree, which means that all the data values stored in the $R$ nodes are different.

In is not difficult to rewrite all the stds in the vertical and horizontal version with $\neq$ instead of $=$. First, get rid of equality on the target side: replace all stds of the form $\varphi \to x = y, z = w$ with $\varphi, x \neq y \to \bot$ and $\varphi, z \neq w \to \bot$. On the source side, we only used equality in the form of the variable $x$ repeating twice. To get rid of this, in the stds obtained in the previous stage replace the two occurrences of $x$ with $x_1$ and $x_2$, add $x_1 \neq x_2$ on the target side and remove $\bot$. This proves undecidability of $\textsc{cons}(\downarrow^*, \neq)$ and $\textsc{cons}(\to, \neq)$. □

This result raises the question whether there is any useful decidable restriction of $\mathrm{SM}(\Downarrow, \Rightarrow, \sim)$. We know from papers such as [51] 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.

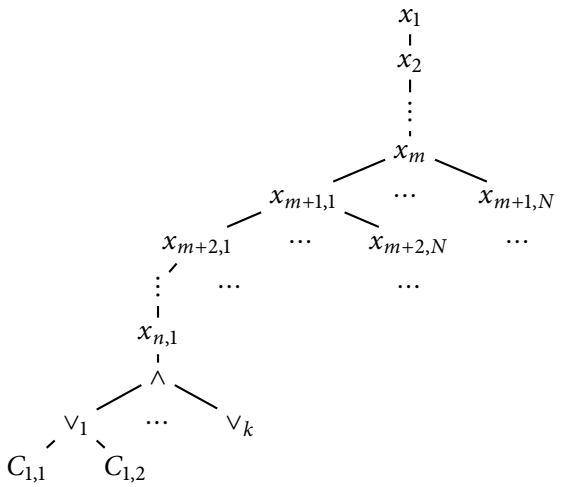
**Theorem 3.9** *Under the restriction to nested-relational DTDs:*

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

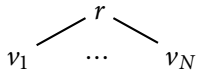
*Proof.* We use reduction from satisfiability of Bernays-Schoenfinkel formulae, which is known to be NEXPTIME-hard ([30, 69, 86]). Let the given formula be

$$\exists x_1 \cdots \exists x_m \forall x_{m+1} \cdots \forall x_n \bigwedge_{i=1}^{k} \bigvee_{j=1}^{\ell} C_{i,j},$$

where $C_{ij}$ is an atom or a negated atom. It is known that if a Bernays-Schoenfinkel formula has a model at all, it has a model of size $N = m + \max_R \mathrm{ar}(R)$. The idea is to guess a model in the source tree, with assistance of the target tree. We encode models as trees of the following form:




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 will encode the universe in target trees of the form




and ensure that each value appearing in the source appears in the target.

Let us now formalise these ideas. Define $D_s$ over $\{r, v, x_1, x_2, \ldots, x_m, \wedge, \vee_1, \vee_2, \ldots, \vee_k\} \cup \{y_{i,j}\}_{i=1\,j=1}^{n\ N} \cup \{C_{i,j}\}_{i=1\,j=1}^{k\ \ell}$ as

$$r \to t\ f\ x_1$$

$$
\begin{aligned}
x_i &\to v\, x_{i+1} && 1 \le i < m \\
x_m &\to v\, y_{1,1}\, y_{1,2} \cdots y_{1,N} && \\
y_{i,j} &\to v\, y_{i+1,1}\, y_{i+1,2} \cdots y_{i+1,N} && 1 \le i < n,\ 1 \le j \le N \\
x_{n,i} &\to v\, \wedge && 1 \le i \le N \\
\wedge &\to \vee_1 \vee_2 \cdots \vee_k && \\
\vee_i &\to C_{i,1}\, C_{i,2} \cdots C_{i,\ell} && \\
C_{i,j} &: @attr && 1 \le i \le k,\ 1 \le j \le \ell \\
v, t, f &: @attr &&
\end{aligned}
$$

and $D_t$ over $\{r, v, v_1, v_2, \ldots, v_N\}$ as

$$
\begin{aligned}
r &\to v_1\, v_2 \cdots v_N && \\
v_i &\to v && 1 \le i \le N \\
v &: @attr &&
\end{aligned}
$$

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 $v$ (not in the picture above), which is meant to hold the assigned value. The attribute of each node labelled with $C_{i,j}$ will be used to store the truth value wrt the valuation encoded by the path leading to this node.

Let us now describe the stds. To avoid confusion with the Bernays-Schoenfinkel formula's variables, we will be using capital letters for variables in stds.

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:

$$
\begin{aligned}
r[t(X), f(X)] &\to \bot \\
r[t(T), f(F), //C_{i,j}(X)], X \neq T, X \neq F &\to \bot \qquad \text{for all } i, j\,.
\end{aligned}
$$

Second, check that all the "variable" values are taken from the universe encoded by the target tree:

$$
//v(X) \to //v(X)\,.
$$

Since each branching for $x_i$ with $i > m$ is meant to be a universal quantification, all the

data values must be different:

$$//_{-}[x_{i,p}[v(X)], x_{i,q}[v(X)]] \vdash \bot \qquad \text{for all } i > m \text{ and } p \neq q\,.$$

Next, we need to make sure that the truth values of relations are consistent. Let us define an auxiliary tree pattern $\mathrm{path}_i(X_1, \ldots, X_i)[\psi]$ as

$$\begin{aligned}\mathrm{path}_1(X_1)[\psi] &= {}_{-}[v(X_1), \psi]\,,\\ \mathrm{path}_{i+1}(X_1, X_2, \ldots, X_{i+1})[\psi] &= {}_{-}[v(X_1), \mathrm{path}_i(X_2, X_3, \ldots, X_{i+1})[\psi]]\,.\end{aligned}$$

We omit the subscript $i$, since it is clear from the number of arguments: for instance, $\mathrm{path}(X, Y, Z)[\psi] = {}_{-}[v(X), {}_{-}[v(Y), {}_{-}[v(Z), \psi]]]$. To enhance appropriate intuitions, we slightly abuse the notation, writing $\mathrm{path}(X_1, \ldots, X_n)/\psi$ instead of $\mathrm{path}(X_1, \ldots, X_n)[\psi]$; and also $\mathrm{path}(X_1, X_2, \ldots, X_n)//\psi$ instead of $\mathrm{path}(X_1, X_2, \ldots, X_n)[//\psi]$.

Take $C_{i,j} = K(x_{i_1}, x_{i_2}, \ldots, x_{i_s})$ and $C_{i',j'} = M(x_{i'_1}, x_{i'_2}, \ldots, x_{i'_s})$, with $K, M \in \{R, \bar{R}\}$ for some relational symbol $R$ used in our formula. We have to check that for every two valuations $\bar{a}$ and $\bar{b}$ for which $a_{i_p} = b_{j_p}$, the values of $C_{i,i'}[\bar{a}]$ and $C_{j,j'}[\bar{b}]$ coincide. For $K = M$ this is expressed by

$$r\left[\mathrm{path}(X_1, X_2, \ldots, X_n)//C_{i,j}(V),\ \mathrm{path}(X'_1, X'_2, \ldots, X'_n)//C_{i',j'}(V')\right],\\ X_{i_1} = X'_{i'_1}, \ldots, X_{i_s} = X'_{i'_s} \ \rightarrow\ V = V'$$

and for $K \neq M$ replace $V = V'$ with $V \neq V'$.

Finally, we have to ensure the truth values assigned make our formula true:

$$r[f(F), //_{\vee i}[C_{i,1}(F), \ldots, C_{i,\ell}(F)] \rightarrow \bot \qquad \text{for all } 1 \leq i \leq k\,.$$

It is straightforward to see that the mapping is consistent iff the given formula is satisfiable.

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 $=$, $\neq$, and $\rightarrow^*$) are monotone in the following sense: for any tree pattern formula $\varphi$, 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. We define an operation $^\circ$ on DTDs as turning every $\ell^*$ and $\ell?$ into $\epsilon$ and $\ell^+$ into $\ell$, as in [12]. The mapping $\mathcal{M}$ is consistent iff $\langle D_s^\circ, D_t, \Sigma\rangle$ is consistent. Since disjunctions are not allowed in productions, we have only one tree conforming to $D_s^\circ$ (up to data values stored in the attributes). The size of this tree is at most $\|D_s\|^{|D_s|}$.

Suppose $S \vDash D_s^\circ$, and $T$ is a solution for $S$. We show that there exists a solution $T'$, single exponential in the size of the mapping. The depth of a tree conforming to $D_t$ is bounded by $|D_t|$, as $D_t$ is non-recursive, so we only need a bound on the branching width. Consider

$$A = \{\psi[\bar{a}] \mid (\varphi, \varphi_{=,\neq} \to \psi, \psi_{=,\neq}) \in \Sigma \text{ and } S \vDash \varphi, \varphi_{=,\neq}[\bar{a}]\}\,.$$

We already know that if $\psi[\bar{a}] \in A$, then $\psi_{=,\neq}[\bar{a}]$ holds, since $T$ is a solution for $S$. Hence, $T'$ is a solution for $S$ iff $T' \vDash \psi[\bar{a}]$ for each $\psi[\bar{a}] \in A$. For each $\psi[\bar{a}] \in A$ fix a realisation $\xi_{\psi[\bar{a}]}$ in $T$. We will now trim unnecessary nodes from $T$. Fix a node $v$ in $T$, and consider its children. Let the child be *red* if it has a descendant used by one of the realisations $\xi_{\psi[\bar{a}]}$, otherwise let the child be *green*. The number of children painted red because of a single realisation $\xi_{\psi[\bar{a}]}$ can be generously bounded by $\|\Sigma\|$. Hence, the number of all red children of $v$ can be bounded by $\|\Sigma\| \cdot |A|$. In the worst case $A$ contains $|S|^{|\text{var}\,\Sigma|}$ instances of the target side of each pattern in $\Sigma$, hence we can bound the number of red children of $v$ by $\|\Sigma\| \cdot |\Sigma| \cdot |S|^{|\Sigma|} \leq \|\Sigma\|^2 \cdot \|D_s\|^{|D_s||\Sigma|}$. Let $T'$ be obtained from $T$ as follows: for each $v$ labelled with $\ell$ cut off those green children, whose label is under $*$ or $+$ in the production for $\ell$ (save for one if the label occurs under $+$). Thus obtained $T'$ conforms to $D_t$. Since no node used by any $\xi_{\psi[\bar{a}]}$ has been removed, $T'$ is a solution for $S$. The number of children of each node in $T'$ can be bounded by $b = \|\Sigma\|^2 \cdot \|D_s\|^{|D_s||\Sigma|} + \|D_t\|$, so the size of $T'$ can be bounded by $(\|\Sigma\|^2 \cdot \|D_s\|^{|D_s||\Sigma|} + \|D_t\|)^{|D_t|}$, which is clearly single exponential in the size of the mapping.

With these bounds in mind, the algorithm for consistency is as follows: guess $S \vDash D_s^\circ$ with the data values in $\{1, 2, \ldots, b_s\}$ and $T \vDash D_t$ with branching bounded by $b_t$ and with data values in $\{1, 2, \ldots, b_t\}$ (both trees are single exponential) and check if $(S, T) \in [\![\Sigma]\!]$. The last check cannot be done in PTIME in general, but a naïve algorithm checking each constraint against each possible valuation requires at most $|\Sigma| \cdot (|S| + |T|)^{|\Sigma|}$ checks polynomial in the size of $S$, $T$, and $\Sigma$. That still gives an algorithm single exponential in the size of the original input. Altogether this works in NEXPTIME.

Actually, we prove that even $\textsc{cons}(\to, =, \neq)$ restricted to nested-relational DTDs is undecidable. The proof is yet another modification of the 2-register machine reduction. Recall the proof for $\textsc{cons}(\to, =)$ . Observe that the reason the mapping used there is not nested-relational is that we let the labels $I_i$ interleave. We will get rid of it by introducing one new label $I$, and storing the state $i$ as an attribute. It will require a bit of hassle to make sure the attributes are valued in the fixed set representing $Q$, and find out which state a given value represents.

Let the source DTD be

$$r \to \sharp\ R^*\ \natural\ I^*\ \flat\ I_1 I_2 \ldots I_f$$

where $I$ has 3 attributes, $R$ and $I_j$ have 1 attribute. The target DTD is still $r \to \varepsilon$.

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:

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

Next, let us make sure that each $I$ node stores a state in its first attribute:

$$r\left[I_1(x_1), I_2(x_2), \ldots, I_f(x_f), I(y, y', y'')\right], y \neq x_1, y \neq x_2, \ldots, y \neq x_f \to \bot$$

Now in the constraints used to prove undecidability of $\textsc{cons}(\to, \to^*, =)$ replace each occurrence of $I_i(x, y)$ with $I(z_i, x, y)$, and add to the highest level of the tree pattern in question a subformula $I_i(z_i)$ (for an occurrence of $I_i$ take fresh variables $z', z''$ and write $I(z_i, z', z'')$). For instance, $r[I_i(x, y) \to I_j, \sharp \to R(x)] \to \bot$ should now be rewritten as $r[I_i(z_i), I_j(z_j), I(z_i, x, y) \to I(z_j, z', z''), \sharp \to R(x)] \to \bot$.

In the above we use both = and ≠. The procedure used in the previous subsection to replace = with ≠ will not work, since in stds like

$$r[I_i(z_i), I_j(z_j), I(z_i, x, y) \to I(z_j, z', z''), \sharp \to R(x)] \to \bot$$

we use two equalities, and we cannot replace them with inequalities on the target side without disjunction.

Slightly modifying the DTDs we could obtain undecidability of $\textsc{cons}(\to, =)$ for nested-relational DTDs. □

## 3.4 Absolute Consistency

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

| | |
|---|---|
| PROBLEM: | ABS-CONS($\sigma$) |
| INPUT: | Mapping $M = (D_s, D_t, \Sigma) \in \mathrm{SM}(\sigma)$ |
| QUESTION: | Is $M$ absolutely consistent? |

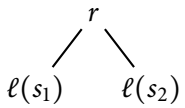
Reasoning about the complexity of absolute consistency is significantly harder than reasoning about the consistency problem. We know that CONS($\Downarrow$) can be easily reduced to CONS$^\circ$($\Downarrow$). However, eliminating data values does not work for absolute consistency.

**Proposition 3.10** *There exists an XML schema mapping $\langle D_S, D_T, \Sigma\rangle$ that is not absolutely consistent while $\langle D_S, D_T, \Sigma^\circ\rangle$ is.*

*Proof.* Consider the following setting.

$$D_S = \langle \{r \to \ell\ell, \ell \to \epsilon\}, \{\langle \ell, a\rangle\}, r\rangle$$
$$D_T = \langle \{r \to \ell', \ell' \to \epsilon\}, \{\langle \ell', a\rangle\}, r\rangle$$
$$\Sigma = \{\ell(x) \to \ell'(x)\}$$

If we consider $\Sigma^\circ$, then this setting is easily seen to be absolutely consistent: For any tree conforming to $D_S$, $r \to \ell'$ is a solution. On the other hand, if we don't drop data value, the tree




with $s_1 \neq s_2$ conforms to $D_S$, but has no solution. This is because the only way to distinguish node with tree-pattern formula is by different data values. Hence the above tree enforces two $\ell'$'s, which violates $D_T$. □

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

Here we settle the problem of absolute consistency in the case of downward navigation, i.e., ABS-CONS($\Downarrow$).

We start with a simpler case of ABS-CONS$^\circ$($\Downarrow$), i.e., checking absolute consistency of mappings $M^\circ$ in which all references to attribute values have been removed. We show that it has lower complexity than CONS$^\circ$($\Downarrow$). For such mappings, $\Sigma$ is of the form $\{\pi_i \to \pi'_i\}_{i\in I}$, where patterns have no variables. To check consistency of such a mapping, we need to

check whether there exists a set $J \subseteq I$ such that $D_t$ and all the $\pi'_j$, $j \in J$ are satisfiable, while $D_s$ together with the *negations* of $\pi_k$, $k \notin J$, are satisfiable. We know that this problem is EXPTIME-complete [12]. On the other hand, for checking absolute consistency, we need to verify that there does not exist $J \subseteq I$ so that $D_s$ and $\pi_j$, $j \in J$, are satisfiable but $D_t$ and $\pi'_j$, $j \in J$, are not. Notice that absolute consistency eliminates the need for checking satisfiability of negations of patterns. In fact, since satisfiability of patterns and DTDs is in NP, the above shows that absolute consistency of mappings $M^\circ$ falls into the 2nd level of the polynomial hierarchy. We can be more precise:

**Proposition 3.11** *Checking whether $M^\circ$ is absolutely consistent is $\Pi_2^p$-complete.*

*Proof.* Let us give a $\Sigma_2^p$ algorithm for the complementary problem. Let $\langle D_s, D_t, \{\varphi_i \to \psi_i \mid i = 1, \ldots, n\}\rangle$ be a mapping. Guess a set $I \subseteq \{1, \ldots, n\}$ and verify that $\{\varphi_i \mid i \in I\}$ is satisfiable with respect to $D_s$ , and $\{\psi_i \mid i \in I\}$ is not satisfiable with respect to $D_t$. Checking satisfiability of a set of tree patterns is easily reduced to single tree pattern satisfiability, e.g., for $\{r[\gamma_1, \gamma_2], r[\delta_1, \delta_2]\}$ an equivalent single formula is $r[\gamma_1, \gamma_2, \delta_1, \delta_2]$. By Lemma 3.1 this can be done in NP. This shows that absolute consistency without data values is in $\Pi_2^p$.

For $\Pi_2^p$-hardness, we give a reduction form the validity of $\Pi_2$ quantified Boolean formulae. For the sake of readability we write $\alpha/\beta/\gamma[\psi, \psi']$ for $\alpha[\beta[\gamma[\psi, \psi']]]$. Satisfiability of $\forall x_1 \cdots x_m \exists x_{m+1} \cdots x_n \bigwedge_{j=1}^k Z_j^1 \vee Z_j^2 \vee Z_j^3$ is equivalent to the absolute consistency of the following mapping over the alphabet $\{t, f, x_1, x_2, \ldots, x_n, C_1, C_2, \ldots, C_k\}$, where $C_j$ is intended to correspond to $Z_j^1 \vee Z_j^2 \vee Z_j^3$:

$$
\begin{array}{lll}
D_s: & r \to x_1 x_2 \cdots x_m & \\
 & x_i \to t|f & 1 \le i \le m \\
 & & \\
D_t: & r \to x_1 x_2 \cdots x_m x_{m+1} \cdots x_n & \\
 & x_i \to t\{C_j \mid \exists_\ell Z_j^\ell = x_i\}|f\{C_j \mid \exists_\ell Z_j^\ell = \bar{x}_i\} & 1 \le i \le n \\
 & & \\
\Sigma: & r/x_i/t \to r/x_i/t & 1 \le i \le n \\
 & r/x_i/f \to r/x_i/f & 1 \le i \le n \\
 & r \to r/\_/C_j & 1 \le j \le k
\end{array}
$$

In the second rule for $D_t$, interpret each set as a concatenation of all its elements. Each source tree encodes a valuation of the universally quantified variables. Each target tree

encodes a valuation of all variables $x_i$ and for each variable $x_i$ it stores all conjuncts made true by assigning this particular value to $x_i$. The first two lines of constraints make sure that the valuations coincide on common variables. The third line checks that each conjunct is true. □

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

**Theorem 3.12** *ABS-CONS($\Downarrow$) is decidable. In fact, the problem is solvable in EXPSPACE and NEXPTIME-hard.*

The decidability proof is quite involved – we only give the main idea of it. Below we first give the sketch of decidability, and then prove its NEXPTIME-hardness, stating them as separate theorems. A slightly more detailed description of the decidability will be given in the next subsection.

**Lemma 3.13** *ABS-CONS($\Downarrow$) is in EXPSPACE.*

*Proof.* The idea of the proof is actually quite simple, and standard. The main goal is to prove that we need a finite number of data values to obtain a counterexample for absolute consistency. To be more concrete, the proof culminates in the following lemma[1]:

**Lemma 3.14** *Given a schema mapping $M = \langle D_s, D_t, \Sigma \rangle$, we can compute $M' = \langle D'_s, D'_t, \Sigma' \rangle$ such that*

- *$D_s, D_t$ are both non-recursive and do not use $^*$ in productions;*
- *$D'_s, D'_t$ generate trees of size at most exponential in the size of $D_s, D_t$, respectively;*
- *if $M'$ is not absolute consistent, then there exists a counterexample using at most $\ell_{D'_s}(\ell_{D'_t} + 1)$ data values, where $\ell_D$ is the maximum number of leaves that $D$ can have so that in particular $\ell_{D'_s}, \ell_{D'_t}$ are at most exponential in the size of original DTDs $D_s, D_t$, respectively.*

Let us see this implies the statement of the theorem. Consider the mapping $\langle D_s, D_t, \Sigma \rangle$ with the data-domain restricted to the set $\{1, 2, \ldots, \ell_{D_s}(\ell_{D_t} + 1)\}$. One can check the absolute consistency of this mapping in exponential space in the following way.

By Savitch's Theorem, it is enough to give a non-deterministic algorithm for the complementary problem. For this simply guess a subset of constraints $\{\psi_1 \to \varphi_1, \ldots, \psi_k \to$

[1] This is a little imprecise: see the next subsection.

$\varphi_k\} \subseteq \Sigma$ together with a set $A$ of valuations of var $\{\psi_1, \ldots, \psi_k\}$ in $\{1, 2, \ldots, \ell_{D_s}(\ell_{D_t} + 1)\}$, and check that $\{\psi_1, \ldots, \psi_k\}[A]$ is satisfiable with respect to $D_s$, and $\{\varphi_1, \ldots, \varphi_k\}[A]$ is *not* satisfiable with respect to $D_t$. This can be done just like ordinary satisfiability (cf. proof of Lemma 3.1)

Aside: We cannot use automata nonemptiness to get the desired bound since the number of patterns (which is essentially the size of an assignment) is exponential. It would give a double exponential algorithm.

□

The algorithm described above is in fact a $\Pi_2^p$ procedure run over exponentially large data. Below we show that the absolute consistency problem is NEXPTIME-hard. This hardly leaves space for improvement.

**Theorem 3.15** *Absolute consistency is NEXPTIME-hard even if we allow only nested relational DTDs and only one of the operation _, //.*

*Proof.* We will reduce satisfiability of Bernays-Schoenfinkel formulae [69] to absolute consistency. Let

$$\varphi = \exists x_1 \exists x_2 \ldots \exists x_m \forall x_{m+1} \forall x_{m+2} \ldots \forall x_n \bigwedge_{i=1}^{k} \bigvee_{j=1}^{\ell} C_{i,j}\,.$$

If $\varphi$ is satisfiable, it has a model with at most $N = m + \max_R \mathrm{ar}(R)$ elements. Define $D_s$ over $\{r, t, f, v, v_1, v_2, \ldots, v_N\}$ as

$$\begin{aligned} r &\to v_1\, v_2 \cdots v_N\, t\, f \\ v_i &\to v && 1 \le i \le N \\ v, t, f &: @attr \end{aligned}$$

and $D_t$ over $\{r, v, y_1, y_2, \ldots, y_m, \wedge, \vee_1, \vee_2, \ldots, \vee_k\} \cup \{x_{i,j}\}_{i=1\,j=1}^{n\quad N} \cup \{C_{i,j}\}_{i=1\,j=1}^{k\quad \ell}$ as

$$\begin{aligned} r &\to x_{1,1}\, x_{1,2} \cdots x_{1,N}\, y_1 \\ x_{i,j} &\to v\, x_{i+1,1}\, x_{i+1,2} \cdots x_{i+1,N} && 1 \le i < n,\ 1 \le j \le \ell \\ x_{n,i} &\to v\, \wedge && 1 \le i \le N \\ \wedge &\to \vee_1\, \vee_2 \cdots \vee_k \\ \vee_i &\to C_{i,1}\, C_{i,2} \cdots C_{i,\ell} && 1 \le i \le k \\ C_{i,j} &\to w\, w_+ \\ w, w_+ &\to u \end{aligned}$$

$$
\begin{aligned}
y_i &\to y_{i+1} && 1 \le i < m \\
y_m, v, u &\to \varepsilon \\
y_i, v, u &: @attr && 1 \le i \le m
\end{aligned}
$$

In the source trees we think of the data values stored in $v$-nodes as elements of the universe. The data values in $t$ and $f$ are intended to mean "true" and "false". In the target tree each $x_{1,i_1}x_{2,i_2}\cdots x_{n,i_n}$ path encodes a valuation of $\varphi$'s variables. At the end of such a path we store the truth values of literals with respect to the valuation given by the path: $w$ stores the truth value of the literal $C_{ij}$ and $w_+$ stores the truth value of the underlying atom. The values in $y_1, y_2, \ldots, y_m$ are meant to witness the existential quantifiers.

Like in the previous reduction from satisfiability of Bernays-Schoenfinkel formulae, we use the abbreviation $\mathrm{path}_{i,j}(X_i, \ldots, X_j)/\psi$ defined on page 41, and write variables as capital letters. Let us describe the constraints. First, make sure that the encoded truth values yield $\varphi$ true.

$$
\begin{aligned}
&r[\_/v(X_{m+1}), \_/v(X_{m+2}), \ldots, \_/v(X_n), t(T)] \\
&\quad \to r[y_1(X_1)/\cdots/y_m(X_m), \mathrm{path}(X_1, \ldots, X_n)/\wedge[\vee_1/\_/w/u(T), \ldots, \vee_k/\_/w/u(T)]]\,.
\end{aligned}
$$

Next, check that the truth values under $w$ and $w_+$ are assigned consistently. For each negative literal $C_{i,j}$ add

$$
\begin{aligned}
&r[\_/v(X_1), \_/v(X_2), \ldots, \_/v(X_n), t(T), f(F)] \\
&\qquad \to\ r/\mathrm{path}(X_1, X_2, \ldots, X_n)/\wedge/\vee_i/C_{i,j}[\_/u(T), \_/u(F)]\,,
\end{aligned}
$$

and for each positive literal $C_{i,j}$ add

$$
\begin{aligned}
&r[\_/v(X_1), \_/v(X_2), \ldots, \_/v(X_n)] \\
&\qquad \to\ \mathrm{path}(X_1, \ldots, X_n)/\wedge/\vee_i/C_{i,j}[w/u(W), w_+/u(W)]\,.
\end{aligned}
$$

Finally, make sure that the truth values of relations are consistent. Take subformulae $C_{i,i'} = K(x_{i_1}, x_{i_2}, \ldots, x_{i_s})$ and $C_{j,j'} = M(x_{j_1}, x_{j_2}, \ldots, x_{j_s})$, with $K, M \in \{R, \bar{R}\}$ for some relational symbol $R$. We have to check that for every two valuations $\bar{a}$ and $\bar{b}$ for which $a_{i_p} = b_{j_p}$, the values of $C_{i,i'}[\bar{a}]$ and $C_{j,j'}[\bar{b}]$ coincide. This is expressed by

$$
r[\_/v(X_1), \_/v(X_2), \ldots, \_/v(X_n),\ \_/v(Y_1), \ldots, \_/v(Y_n),\ \_/v(Z_1), \ldots, \_/v(Z_n)]
$$

$$\to r[\mathrm{path}(\Xi_1,\ldots,\Xi_n)/\wedge/\vee_i/C_{i,i'}/w_+/u(W),\mathrm{path}(\Upsilon_1,\ldots,\Upsilon_n)/\wedge/\vee_j/C_{j,j'}/w_+/u(W)],$$

where the variables $\Xi_p$ and $\Upsilon_p$ are computed as follows. Let

$$G = \langle \{\Xi_p, \Upsilon_p \mid p = 1,2,\ldots,n\}, \{\{\Xi_{i_p}, \Upsilon_{j_p}\} \mid p = 1,2,\ldots,s\}\rangle .$$

For each connected component $S$ of the graph $G$: if $S = \{\Xi_p\}$, set $\Xi_p = X_p$; if $S = \{\Upsilon_p\}$, set $\Upsilon_p = Y_p$; else, set each variable in $S$ to the first so far unused variable $Z_q$. For instance, for $C_{i,i'} = R(x_2,x_2)$, $C_{j,j'} = R(x_1,x_2)$, and $n = 2$, we have two connected components, $\{\Xi_1\}$ and $\{\Xi_2,\Upsilon_1,\Upsilon_2\}$, so we set $\Xi_1 = X_1$, $\Xi_2 = \Upsilon_1 = \Upsilon_2 = Z_1$, and the obtained std is

$$\begin{aligned} r[\_/v(X_1),\_/v(X_2),\ \_/v(Y_1),\_/v(Y_2),\ \_/v(Z_1),\_/v(Z_2)]\ &\to \\ \to r[\mathrm{path}(X_1,Z_1)/\wedge/\vee_i/C_{i,i'}/w_+/u(W),&\mathrm{path}(Z_1,Z_1)/\wedge/\vee_j/C_{j,j'}/w_+/u(W)]. \end{aligned}$$

Let us now see that the reduction works as intended. Suppose that the mapping is absolutely consistent. Take a tree $S$ such that the data value stored under $v_i$ is the natural number $i$, and the values stored under $t$ and $f$ are $\top$ and $\bot$, respectively. Since the mapping is absolutely consistent, there exists a solution $T$ for $S$. It is easy to see that the constraints enforce that $T$ encodes a model for $\varphi$.

Conversely, suppose that $\varphi$ has a model over $\{1,2,\ldots,N\}$. Take a tree $T$ encoding this model, with data values $\top$ and $\bot$ used to denote truth values. Let $S \models D_1$. Denote the value stored under $v_i$ by $a_i$, and the values stored under $t$ and $f$ by $a_t$ and $a_f$. Relabelling data values we may assume that $\{a_1,a_2,\ldots,a_N\} \subseteq \{1,2,\ldots,N\}$. If $a_t \neq a_f$, then we may assume again that $a_t = \top$ and $a_f = \bot$, and it is easy to see that $T$ is a solution for $S$. If $a_t = a_f$, assume $a_t = a_f = \top$ and replace each $\bot$ in $T$ with $\top$ to obtain a solution for $S$.

This shows that absolute consistency is NEXPTIME-hard if we are allowed to use _ in the constraints. For the second case simply observe that replacing all occurrences of _ with // does not change the semantics of the constraints. □

Theorem 3.12 indicates that any algorithm for solving ABS-CONS($\Downarrow$) will run in double-exponential time, and hence will be impractical unless restrictions are imposed. Restrictions to nested-relational DTDs often worked for us, but in this case they alone do not suffice, as we shall see shortly. In addition to nested relational DTDs, we shall need a restriction to *fully-specified stds*, introduced in [12] to obtain tractable algorithms for query answering in data exchange.

The combination of nested-relational DTDs and fully specified stds gives us tractability. Notice that this tractability is essentially tight: due to the preceding theorem, if we allow wildcard or descendant in patterns, the problem become NEXPTIME-hard for nested relational DTDs.

**Theorem 3.16** *With restriction to nested relational DTDs and fully specified stds, the problem* ABS-CONS$(\Downarrow)$ *is solvable in PTIME.*

*Proof.* Nested relational DTDs allows only basic counting – one or many. As previously, we assume, without loss of generality, that (1) attributes only appear leaves, (2) any label has at most one attribute, and (3) no distinct dependencies share variables in the set of stds. A variable $x$ is *1-bounded* with respect to a DTD $D$ and a set of tree patterns $\Phi$ if it appears in (a subformula of a formula in) $\Phi$ as $r/\sigma_1/\cdots/\sigma_n(x)$, where none of $\sigma_i$'s appear with $*$ in $D$. Otherwise it is unbounded. It is clear that any unbounded variable on the source side must also be unbounded, for a setting to be absolutely consistent.

There is another thing we have to check. Recall the example in Proposition 3.10. The problem is that DTDs might require that all the data on the source is the same. To detect this kind of constraints, we need to check if a pair of variable will be "equated" or not. For instance, if we have $r[a(x), a(y)]$ with the DTD $r \to a$, $x$ and $y$ will be necessarily equal. In general we define a binary relation $x \sim y$ on variable appearing in $\Phi$: $x \sim_{D,\Phi} y$ iff $x$ and $y$ appear on the same path in $\Phi$ that is unique with respect to $D$.

Given a setting $\langle D_s, D_t, \Sigma\rangle$, where $\Phi$ is the set of source patterns in $\Sigma$ and $\Psi$ is that of target, we have the setting is absolutely consistent if and only if for all variables,

- $x \sim_{D_t,\Psi} y$ implies $x \sim_{D_s,\Phi} y$;
- and if $x$ is unbounded on the source, then it is unbounded on the target.

Since both conditions are easy to check, we have tractability (quadratic, in fact) for absolute consistency for nested relational DTDs and fully specified stds. □

### 3.4.1 Appendix: a Closer Look at the Proof of Theorem 3.13

This subsection is dedicated to giving an outline of Theorem 3.13. As already mentioned, the key is that we can bound the data domain.

Recall Proposition 3.10. The failure of absolute consistency comes from the fact that the source DTD can (generate a tree that can) hold two data values while the target can only one.

**Storage**

As suggested by this example, checking if a mapping is absolutely consistent amounts to how many tuples the source DTD can hold compared with how many the target can. To do this, we define the notion of *storage*. Without loss of generality, we assume the data domain is $\mathbb{N}$.

Let us start with an example. The storage of a DTD $D$ with respect to a tree pattern $\varphi$ depends on the structure of sets $A$ for which we can provide target trees. Consider a DTD $D$ defined as

$$\begin{aligned} r &\to a\, a \\ a &\to b\, c^* \mid b^*\, c\, c \\ b, c &: @attr \end{aligned}$$

and $\varphi = r/a[b(x), c(y)]$. It is straightforward to see that for arbitrary large natural numbers $M, N$, there exists a tree $T \vDash D$ such that $T \vDash \varphi[a]$ for all assignments (from $\mathrm{var}\varphi$ to $\mathbb{N}$) $a \in \{0\} \times \{0, 1, \ldots, N\} \cup \{0, 1, \ldots, M\} \times \{0, 1\}$. Also note that there does not exist a tree conforming to $D$ that satisfies $\varphi$ for all valuations in $\{0, 1\} \times \{0, 1, 2\}$. Thus intuitively the storage of $D$ with respect to $\varphi$ contains $\{0\} \times \{0, 1, \ldots, N\} \cup \{0, 1, \ldots, M\} \times \{0, 1\}$ for any $M, N$ but not $\{0, 1\} \times \{0, 1, 2\}$.

Formally, for a DTD $D$ and a set of formulae $\Phi$, we define the storage of $D$ with respect to $\Phi$ as

$$\mathrm{val}(D, \Phi) = \{\mathrm{val}(T, \Phi) \mid T \vDash D\},$$

where $\mathrm{val}(T, \Phi)$ is the set of *partial* valuations $a : \mathrm{var}(\Phi) \multimap \mathbb{N}$ such that for each $\varphi \in \Phi$: either $\mathrm{var}\,\varphi \cap \mathrm{dom}\, a = \varnothing$, or $\mathrm{var}\,\varphi \subseteq \mathrm{dom}\, a$ and $T \vDash \varphi[a]$ (no two formulae of $\Phi$ share a free variable). Note that for a singleton $\{\varphi\}$ this reduces to

$$\mathrm{val}(D, \{\varphi\}) = \left\{ \{a \mid a\colon \mathrm{var}\,\varphi \to \mathbb{N},\ T \vDash \varphi[a]\} \mid T \vDash D \right\}.$$

For instance,

$$\mathrm{val}(D, \{\varphi\}) = \left\{ A \times A' \cup B \times B' \mid |A| \le 1;\ |B| \le 2; |A'|, |B'| \text{ are both finite.} \right\}.$$

We define the following order on storage: For storage $S$, $T$,

$$S \le T \iff \forall_{A \in S} \exists_{B \in T} \forall_{a \in A} \exists_{b \in B}\ a \subseteq b\,.$$

We write $S \equiv T$ when $S \leq T$ and $T \leq S$.

This order captures the absolutely consistency as witnessed by the following lemma:

**Lemma 3.17** *A mapping* $\langle D_s, D_t, \{\varphi_i \to \psi_i \mid i = 1, \ldots, n\}\rangle$ *is absolutely consistent iff*

$$\mathrm{val}(D_s, \{\varphi_1, \ldots, \varphi_n\}) \leq \mathrm{val}(D_t, \{\psi_1, \ldots, \psi_n\}) .$$

**Normal Guiding DTDs**

Since our semantics is non-injective, a pattern $r[b(1), b(2)]$ requires two $b$ nodes below $r$ while $r[b(1), b(1)]$ does not. As such we want to distinguish them in such a way that we can construct trees $T_1$, $T_2$ with the following properties:

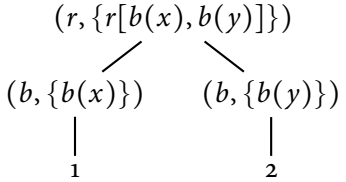
- $T_1 \vDash \varphi[1,2]$, but $T_1 \nvDash \varphi[1,1]$;
- $T_2 \vDash \varphi[1,1]$ and $T_2 \vDash \varphi[2,2]$, but $T_2 \nvDash \varphi[1,2]$.

Since this is impossible in the classical setting we have been working in, we equip a sort of annotation with trees and modify the notion of attribute.

The annotation is done to indicate which patterns are satisfied in a node. Fix two alphabets $\Gamma$ and $\Gamma'$. A *guiding tree* over $\Gamma, \Gamma'$ is an ordinary data tree over the alphabet $\Gamma' \times \mathcal{P}(\Pi_\Gamma)$, where $\Pi_\Gamma$ is the set all tree patterns over the alphabet $\Gamma$. We call $\Gamma$ the *labelling alphabet* and $\Gamma'$ the *auxiliary alphabet*. Even though $\Pi_\Gamma$ is infinite, we only consider finite trees. In fact, we always consider only a finite fragment of $\Pi_\Gamma$, usually a set of subformulae of some given finite set of tree patterns.

The modelling relation for guiding trees is defined just like for ordinary data trees, only to determine if $\varphi$ is satisfied in $v$, instead of looking at the head functor of $\varphi$, we check if $\varphi$ is contained in the second component of $v$'s label. More formally, a realisation $\xi$ of $\varphi$ in a guiding tree $T$ assigns consistently to each subformula $\psi$ of $\varphi$ a node labelled with $(\sigma, \Theta)$, such that $\psi \in \Theta$. Like for ordinary trees, a realisation yields exactly one valuation of $\varphi$'s variables. We say that $T \vDash \varphi[a]$ iff there exists a realisation of $\varphi$ in $T$ yielding the valuation $a$.

We can have $T_1$, a guiding tree, defined as



where 1 and 2 are data values stored in attributes.

DTDs defining sets of guiding trees will be called *guiding DTDs*, or gDTDs for short. Note that the auxiliary alphabet is of no importance for the satisfaction of the formulae. We only use it to be able to define all needed sets of guiding trees by means of gDTDs. Equivalently we could use extended DTDs, but for technical reasons we prefer to include the specification as a part of the label.

We next introduce the notion of multiattribute. This is the opposite of the construct above, so that we can prevent assignment of variables to different values. While a node with an ordinary attribute stores a single data value, a node with a multiattribute stores a set of data values. A multiattribute of multiplicity $m$, $m$-attribute for short, stores $m$ data values, called *instances* of the multiattribute. Intuitively, the semantics of multiattributes say that if we are in a leaf $v$ and have formulae $\sigma_1(x_1), \sigma_2(x_2), \ldots, \sigma_k(x_k)$ to evaluate, we must pick one instance of the multiattribute and assign its value to all the variables $x_1, x_2, \ldots, x_k$. This should be contrasted with the rule we apply for ordinary nodes, where we can chose a different child for each sub-formula.

For instance, we can now define $T_2$ satisfying the postulates above as

$$\begin{array}{c} (r, \{r[b(x), b(y)]\}) \\ | \\ (b, \{b(x), b(y)\}) \\ | \\ \{1, 2\} \end{array}$$

where $\{1, 2\}$ is the content of the 2-attribute of the $(b, \{b(x), b(y)\})$ node. To repeat, we have to pick one of 1, 2 for the $b$ node in the tree, end up assigning the same value to both $x$ and $y$.

The notion of a DTD is extended in a natural way to capture the multiattributes: for each leaf-type we specify the multiplicity of the attribute it has, we are also allowed to use $*$ here, which means that in a tree we may fix any multiplicity of a given multiattribute.

In some sense, guiding trees can be seen trees (in the former sense) coupled with information about (potential) assignments of values to variables.

There are two cases where a pattern can be satisfied arbitrarily many times. One is because of $^*$ in productions of a DTD; the other is because of recursion in a DTD. Basically, we replace both kinds of unboundedness with $*$-attribute, removing recursions and $^*$ in productions.

We need another definition before providing the formal statement of the above. A realisation of a tree pattern in guiding tree is *exhaustive* in a node if the node is assigned

to all formulae from the auxiliary label, or is not assigned to any formula. In other words, a realisation is exhaustive when the auxiliary label completely describe the assignment of formulae to node or is totally irrelevant. Now a gDTD is $\Phi$-normal if it is non-recursive, does not use $^*$ in productions, and only admits realisations of formulae from $\Phi$ which are exhaustive in $*$-attribute leaves.

Finally we can state that recursion and $^*$ in productions can be removed:

**Lemma 3.18** *For each $\Phi$ and each DTD $D$ one can compute in EXPTIME a $\Phi$-normal gDTD $N$ such that*

- *$N$ generates trees of size at most exponential in the size of $D$;*
- $\mathrm{val}(D, \Phi) \equiv \mathrm{val}(N, \Phi)$.

*(Recall $\equiv$ is the equivalence relation induced by $\leq$ on storage.)*

Combining this and the following lemma, the exact statement of Lemma 3.14 can be obtained:

**Lemma 3.19** *Let $\langle D_s, D_t, \{\psi_1 \to \varphi_1, \ \ldots, \ \psi_n \to \varphi_n\}\rangle$ be a mapping such that $D_s$ is a $\Psi$-normal gDTD, $D_t$ is a $\Phi$-normal gDTD, $\Psi = \{\psi_1, \ldots, \psi_n\}$, $\Phi = \{\varphi_1, \ldots, \varphi_n\}$. If the mapping is not absolutely consistent, there exists a counter example using at most $\ell_{D_s}(\ell_{D_t}+1)$ data values.*

The rest of the proof goes as shown in the previous section.

# Chapter 4

# Composing schema mappings

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 [48], where $[\![M]\!]$ is defined as a binary relation. We define the composition of two mappings $M$ and $M'$ simply as $[\![M]\!] \circ [\![M']\!]$. That is, for two mappings $M_{12} = (D_1, D_2, \Sigma_{12})$ and $M_{23} = (D_2, D_3, \Sigma_{23})$, their composition consists of pairs of trees $(T_1, T_3)$ such that:

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

We consider the following problems:

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

## 4.1 Consistency of Composition

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

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

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

| | |
|---|---|
| PROBLEM: | CONSCOMP($\sigma$) |
| INPUT: | Two consistent mappings $M, M' \in \mathrm{SM}(\sigma)$ |
| QUESTION: | Is the composition of $M$ and $M'$ consistent? |

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

**Theorem 4.1** *The complexity of* CONSCOMP($\sigma$) *is:*

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

*Proof.* We first prove that CONSCOMP($\Downarrow, \Rightarrow$) is in EXPTIME. The idea is the same as for CONS($\Downarrow, \Rightarrow$). The composition of $\langle D_1, D_2, \Sigma_{12}\rangle$ and $\langle D_2, D_3, \Sigma_{23}\rangle$ is consistent iff the composition of $\langle D_1, D_2, \Sigma_{12}^\circ\rangle$ and $\langle D_2, D_3, \Sigma_{23}^\circ\rangle$ is consistent, so we may assume that DTDs have no arguments, and tree patterns have no variables. Suppose $\Sigma_{12} = \{\varphi_i \to \psi_i \mid i \in 1, 2, \ldots, n\}$ and $\Sigma_{23} = \{\varphi'_j \to \psi'_j \mid j = 1, 2, \ldots, m\}$. 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

$$
\begin{aligned}
T_1 &\vDash D_1 \wedge \bigwedge_{i \notin I} \neg\varphi_i\,,\\
T_2 &\vDash D_2 \wedge \bigwedge_{i \in I} \psi_i \wedge \bigwedge_{j \notin J} \neg\varphi'_j\,,\\
T_3 &\vDash D_3 \wedge \bigwedge_{j \in J} \psi'_j\,.
\end{aligned}
$$

Equivalently we need to check nonemptiness of the following automata for all possible $I$ and $J$:

$$
\begin{aligned}
&A_{D_1} \times \prod_{i \notin I} \bar{A}(\varphi_i)\,,\\
&A_{D_2} \times \prod_{i \in I} A(\psi_i) \times \prod_{j \notin J} \bar{A}(\psi'_j)\,,\\
&A_{D_3} \times \prod_{j \in J} A(\psi'_j)\,.
\end{aligned}
$$

Like for CONS($\Downarrow, \Rightarrow$), this gives an exponential algorithm.

Next we show how to reduce consistency of a single mapping to consistency of a composition of two consistent mappings. Suppose we are given $M = \langle D_s, D_t, \Sigma \rangle$. We provide two consistent mappings, $M_1$ and $M_2$, such that $M$ is consistent iff the composition of $M_1$ and $M_2$ is consistent.

Let $\perp$ be a fresh symbol, not used in $M$. Let $D^\perp$ be the DTD obtained from $D$ by extending the alphabet with $\perp$, and replacing the production for the root, $r \to \alpha$, with the production $r \to \alpha \mid \perp$. Define

$$M_1 = \langle D_s^\perp, D_t^\perp, \Sigma \cup \{r/\perp \to r/\perp\} \rangle, \qquad M_2 = \langle D_t^\perp, D, \{r/\perp \to r/\perp\} \rangle,$$

where $D$ is just $r \to \varepsilon$. Observe that both mappings are consistent: $(r[\perp], r[\perp]) \in [\![M_1]\!]$, and $(T, r) \in [\![M_2]\!]$ for any $T \vDash D_t$.

Let us check that their composition is consistent iff $M$ is consistent. If $(S, T) \in [\![M]\!]$, then $(S, T) \in [\![\mathcal{M}_1]\!]$, $(T, r) \in [\![M_2]\!]$, and so $(S, r) \in [\![M_1 \circ M_2]\!]$. Conversely, suppose $(T_1, T_2) \in [\![M_1]\!]$ and $(T_2, T_3) \in [\![M_2]\!]$. Note that $\perp$ gets propagated to $T_3$ if it occurs in $T_1$ or $T_2$. In consequence, $T_1$ and $T_2$ do not contain $\perp$. But then, $T_1 \vDash D_s$ and $T_2 \vDash D_t$. Since the pair $(T_1, T_2)$ satisfies all the constraints from $\Sigma \cup \{r/\perp \to r/\perp\}$, we conclude that $(T_1, T_2) \in [\![M]\!]$.

Observe that if $M$ uses neither = nor $\neq$, so do $M_1$ and $M_2$. Hence, the reduction proves that consistency of composition is EXPTIME-hard in the absence of = and $\neq$, and is undecidable in their presence. □

The decidability result carries over to an arbitrary number of mappings. We can define composition of an arbitrary number of mappings $M_1, \ldots, M_n$ simply as the composition of binary relations $[\![M_i]\!]$'s.

**Proposition 4.2** *The problem of checking whether the composition of n schema mappings $M_1, \ldots, M_n$ from* $\mathrm{SM}(\Downarrow, \Rightarrow)$ *is consistent is solvable in EXPTIME.*

*Proof.* The idea used in the proof of Theorem 4.1 can be extended to consistency of multifold composition. Suppose we are given $\langle D_1, D_2, \Sigma_1 \rangle, \ldots, \langle D_n, D_{n+1}, \Sigma_n \rangle$, where $\Sigma_i = \{\varphi_j^i \to \psi_j^i \mid j = 1, 2, \ldots, n_i\}$ for $i = 1, 2, \ldots, n$ and we want to know if there are trees $T_1, \ldots, T_n$ such that $(T_i, T_{i+1}) \in [\![\langle D_i, D_{i+1}, \Sigma_i \rangle]\!]$ for $i = 1, 2, \ldots, n$.

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 nonemptiness. For every $n$-tuple $(I_1, \ldots, I_n) \in \prod_i \{1, 2, \ldots, n_i\}$, test nonemptiness of the following au-

tomata:

$$
\begin{aligned}
& A_{D_1} \times \prod_{i \notin I_1} \bar{A}(\varphi_i^1)\,, \\
& \quad \vdots \\
& A_{D_{j+1}} \times \prod_{i \in I_j} A(\psi_i^j) \times \prod_{i \notin I_{j+1}} \bar{A}(\varphi_i^{j+1})\,, \\
& \quad \vdots \\
& A_{D_{n+1}} \times \prod_{i \in I_n} A(\psi_i^n)\,.
\end{aligned}
$$

The composition of the given $n$ mappings is consistent iff these products are nonempty for some $(I_1, \ldots, I_n)$. □

## 4.2 Data and Combined Complexity

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

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

Data complexity of relational composition is known to be in NP, and could be NP-complete for some mappings [48]. For XML mappings, the problem becomes undecidable once data value comparisons are allowed. Without such comparisons, it is decidable: the data complexity goes a little bit up compared to the relational case, and we have the usual exponential gap between data and combined complexity.

**Theorem 4.3**

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

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

*Proof.* **Combined complexity of composition membership for** $\mathrm{SM}(\Downarrow, \Rightarrow)$

First we show that combined complexity of composition membership is not worse than 2-EXPTIME. Fix two mappings, $\langle D_1, D_2, \Sigma_{12}\rangle$ and $\langle D_2, D_3, \Sigma_{23}\rangle$, with $\Sigma_{12} = \{\varphi_i \to \psi_i \mid i = 1, 2, \ldots, n\}$ and $\Sigma_{23} = \{\gamma_j \to \delta_j \mid j = 1, 2, \ldots, m\}$. Given trees $S \vDash D_1$ and $T \vDash D_3$, we have to check if there is an interpolating tree $U$.

For a start, consider a variable and attribute-free situation. What are the constraints on the interpolating tree? Clearly, what is imposed by $S$, is that some of the target sides of stds from $\Sigma_{12}$ have to be satisfied in $U$. In contrast, what is imposed by $T$, is that some source sides of stds from $\Sigma_{23}$ are *not* satisfied in $U$: indeed, $U \vDash \gamma \implies T \vDash \delta$ is equivalent to $T \nvDash \delta \implies U \nvDash \gamma$. Therefore, the existence of an interpolating tree $U$ is equivalent to nonemptiness of

$$A_{D_2} \times \prod_{i:S\vDash\varphi_i} A(\psi_i) \times \prod_{j:T\nvDash\delta_j} \bar{A}(\gamma_j)\,.$$

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 $S$ to the alphabet. A tentative set of constraints imposed on $U$ by $S$ would then be

$$\Psi = \{\psi(\bar{a}) \mid \varphi \to \psi \in \Sigma_{12}\,,\ \ \bar{a}\colon \operatorname{var}\varphi \to A\,,\ \ S \vDash \varphi(\bar{a})\}\,.$$

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

The solution is guess data values for the remaining free variables. What should be the domain of our guess? Let $A$ be the set of data values used in $S$, and $B$ the set of data values used in $T$. It is not difficult to see that in an intermediate tree $U$ each data value not in $A \cup B$ can be safely replaced by any fixed data value, in particular, by a fixed data value form $B$. This holds because we do not use $=$ nor $\neq$. Hence, we can assume that $U$ only uses data values from $A \cup B$, and guess the values for outstanding variables in $\psi(\bar{a})$ from $A \cup B$. Let $\Psi'$ denote the set $\Psi$ with guessed data values for free variables (independently

for each element of $\Psi$) and let

$$\Gamma = \{\gamma(\bar{a}) \mid \gamma \to \delta \in \Sigma_{23}\,,\ \ \bar{a}\colon \operatorname{var}\gamma \cup \operatorname{var}\delta \to A \cup B\,,\ \ T \not\models \delta(\bar{a})\}\,.$$

The composition membership algorithm should construct the set $\Psi$ (at most $|\Sigma_{12}||S|^{|\Sigma_{12}|}$ polynomial checks), and construct $\Gamma$ (at most $|\Sigma_{23}|(|S|+|T|)^{|\Sigma_{23}|}$ polynomial checks). Then it is enough to check nonemptiness of

$$A_{D_2} \times \prod_{\psi\in\Psi'} A(\psi) \ \times \prod_{\gamma\in\Gamma} \bar{A}(\gamma)\,,$$

iterating over all possible values of $\Psi'$ (at most $(|A|+|B|)^{|\Sigma_{12}||A|^{|\Sigma_{12}|}}$ iterations, each taking time exponential in the size of $\Psi'$ and $\Gamma$, i.e., double exponential in the size of the input). This gives a 2-EXPTIME algorithm.

To show NEXPTIME-hardness, recall the reduction of Bernays-Schoenfinkel satisfiability to absolute consistency of mappings from SM($\Downarrow$) (proof of Theorem 3.15). For a given formula $\varphi$ a mapping $M_\varphi = \langle D_1, D_2, \Sigma\rangle$ was constructed such that $\varphi$ was satisfiable iff $M_\varphi$ was absolutely consistent. In fact, $M_\varphi$ satisfied the following property: $\varphi$ is satisfiable iff there exists $T$ such that $(S,T) \in [\![M_\varphi]\!]$ for

$$S = r[v_1/v(1), v_2/v(2), \ldots, v_N/v(N), t(\top), f(\bot)]\,,$$

where $N$ equals number of existential quantifiers plus maximum of relations' arities. In consequence, $\varphi$ is satisfiable iff $(S,r) \in [\![M_\varphi \circ \langle D_t, \{r \to \varepsilon\}, \varnothing\rangle]\!]$, which reduces Bernays-Schoenfinkel satisfiability to composition membership for SM($\Downarrow$).

**Data complexity of composition membership for** SM($\Downarrow, \Rightarrow$)

Let us examine the algorithm above with assumption that the mappings are fixed. Then, the size of $\Psi'$ and $\Gamma$ is polynomial, so each iteration takes single exponential time. The number of iterations is bounded by $(|A|+|B|)^{|\Sigma_{12}||A|^{|\Sigma_{12}|}}$, which is also single exponential for fixed $\Sigma_{12}$. In consequence, data complexity of composition membership is not worse than EXPTIME for mappings from SM($\Downarrow, \Rightarrow$).

To show EXPTIME-completeness for SM($\Downarrow, \Rightarrow$), we will reduce from non-universality problem for bottom-up nondeterministic automata on binary trees, modifying a proof from [12].

First, define $D_3$ over $\{r, \mathit{label}, \mathit{state}, \mathit{nontransition}, \mathit{rejecting}\}$ as

$$r \to \sharp\ \mathit{state}^*\ \natural\ \mathit{label}^*\ \flat\ \mathit{nontransition}^*$$

$$
\begin{aligned}
\mathit{state} &\to \mathit{rejecting}?\\
\mathit{rejecting}, \mathit{label}, \mathit{nontransition} &\to \varepsilon\\
\mathit{label}, \mathit{state} &: @\mathit{attr}\\
\mathit{nontransition} &: @\mathit{left}, @\mathit{right}, @\mathit{label}, @\mathit{up}
\end{aligned}
$$

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 $\sharp$), 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 $\{r, \mathit{node}, \mathit{label}, \mathit{state}, \mathit{leaf}, \mathit{yes}, \mathit{no}\}$ be given by

$$
\begin{aligned}
r &\to \mathit{node}\\
\mathit{node} &\to \mathit{label}\ \sharp\ \mathit{state}^*\ \natural\ (\mathit{node}\,\mathit{node} \mid \mathit{leaf})\\
\mathit{state} &\to \mathit{yes} \mid \mathit{no}\\
\mathit{leaf}, \mathit{label} &\to \varepsilon\\
\mathit{state}, \mathit{label} &: @\mathit{attr}
\end{aligned}
$$

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 stds, 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 $\Sigma_{23}$, which will enforce the correctness of the run. First, we make sure that *label*-nodes store labels.

$$
//\mathit{label}(x) \to r/\mathit{label}(x)
$$

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 3.8 again.

$$//node[\sharp \to state(x)] \to r[\sharp \to state(x)]$$
$$//node[state(x) \to state(y)] \to r[state(x) \to state(y)]$$
$$//node[state(x) \to \natural] \to r[state(x) \to \natural]$$
$$//node[state(x) \to^* state(y)] \to r[state(x) \to^* state(y)]$$

The last std 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.

$$//node[state(x)/no,\ label(u),\ leaf] \to r[\sharp \to state(y), nontransition(y, y, u, x)]$$
$$//node[state(x)/no,\ label(u),\ node/state(y)/yes \to node/state(z)/yes] \to r/nontransition(y, z, u, x)$$

Finally, check that the run is rejecting.

$$r/node/state(x)/yes \to r/state(x)/rejecting$$

Let us see that membership for $\langle D_1, D_2, \varnothing\rangle \circ \langle D_2, D_3, \Sigma_{23}\rangle$ is indeed EXPTIME-hard. Take an automaton $A = \langle \Gamma, Q, \delta, q_0, F\rangle$ 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 $F = \{q_k, q_{k+1}, \ldots, q_m\}$. 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

$$\begin{aligned} r[\sharp, &state(q_0)/rejecting,\ state(q_1)/rejecting,\ \ldots,\ state(q_{k-1})/rejecting,\\ &state(q_k),\ state(q_{k+1}),\ \ldots,\ state(q_n),\ \natural,\\ &label(a_1),\ label(a_2),\ \ldots,\ label(a_m),\ \flat,\\ &nontransition(p_1, r_1, b_1, s_1), \ldots, nontransition(p_\ell, r_\ell, b_\ell, s_\ell)] \end{aligned}$$

Proving that $A$ rejects some tree iff $(r, T_A) \in [\![\langle D_1, D_2, \varnothing\rangle \circ \langle D_2, D_3, \Sigma_{23}\rangle]\!]$ is straightforward.

In the above, we need both $\Downarrow$ and $\Rightarrow$. The NP lower bound for data complexity of

mappings from SM($\Downarrow$) and SM($\Rightarrow$) comes from the relational case (see [48]). The upper bound being still EXPTIME, this leaves a substantial gap.

**Composition membership for** SM($\Downarrow, \sim$) **and** SM($\Rightarrow, \sim$)

Let us now turn to the undecidability part. Our argument relies upon a particular property of the reductions used to show undecidability in the proof of Theorem 3.8. In all the cases considered there, halting of a 2-register machine $R$ was reduced to consistency of a mapping $M_R = \langle D_s, D_t, \Sigma \rangle$ whose all stds were of the form $\varphi$, $\varphi_{=,\neq} \to \psi_{=,\neq}$. For such stds, the target side's satisfaction does not depend on the target tree! With this in mind we can reduce halting of a 2-register machine to composition membership: for every 2-register machine $M$, $(r, r) \in [\![\langle \{r \to \varepsilon\}, D_s, \varnothing \rangle \circ M_R]\!]$ if and only if $M_R$ is consistent, which in turn is equivalent to halting of $M$. This shows that composition membership is undecidable for mappings from SM($\Downarrow, \sim$) and SM($\Rightarrow, \sim$).

In fact, we obtain undecidability for SM($\downarrow^*, =$), SM($\downarrow^*, \neq$), SM($\to, =$), and SM($\to, \neq$). Using the usual trick of simulating disjunction in the DTDs with ? and $\to$, we can also get undecidability for SM($\to, =$) restricted to nested-relational DTDs. □

## 4.3 Closure under restrictions

Finally we consider the question of whether there is a class of XML schema mappings that is closed under composition. Recall that a class $C$ of schema mappings is said to be closed under composition if the following holds: given two mappings $M_1, M_2 \in C$, there exists $M_3 \in C$ such that $[\![M_3]\!] = [\![M_1]\!] \circ [\![M_2]\!]$.

In relational data exchange, as was explained in Chapter 2, the known results can be summarised in the following two point:

- The class of (mappings with) stds using conjunctive queries is not closed under composition;
- *Skolem function* helps – the class of second-order tgds is closed under composition.

Since the proof of non-closure in the relational case carries over to the XML case, it immediately follows that we need Skolem function. It turns out, however, that the situation is much worse: Various structural features in our tree pattern already provide cases for which we cannot define composition.

We first describe several features of tree patterns that lead to non-closure. Basically the problem is that we can create mappings that require a disjunction for their composition.

Note that these are orthogonal to the non-closure described in [48] so that they cannot be remedied with Skolem functions.

**Proposition 4.4** *The class of schema mappings that uses any one of (a) wildcard; (b) descendant; (c) next-sibling; (d) inequality is not closed under composition.*

*Proof.* For each of $\_$, $//$, $\rightarrow$, $\neq$ we provide three DTDs $D_1, D_2, D_3$ and two sets $\Sigma_{12}, \Sigma_{12}$ of constraints such that the semantics of $\langle D_1, D_2, \Sigma_{12}\rangle \circ \langle D_2, D_3, \Sigma_{23}\rangle$ is

$$\{(r, r[c_1]), (r, r[c_2]), (r, r[c_1, c_2]), (r, r[c_1, c_3]), (r, r[c_2, c_3]), (r, r[c_1, c_2, c_3])\},$$

where we (ab)use tree patterns to describe trees, and thus is not expressible as a single mapping without disjunction[1]. The alphabets of DTDs are shown in parentheses, where bracketed numbers indicate the number of attributes of the label, e.g., $a[1]$ means that elements labelled with $a$ have one attribute; $\ell[0]$ is written as $\ell$.

Let us start with $\_$.

$$\begin{aligned} D_1 &= \{r \to \varepsilon\} && (\{r\}) \\ D_2 &= \{r \to a_1^* a_2^*;\ a_i \to b\} && (\{r, a_1, a_2, b\}) \\ D_3 &= \{r \to c_1?c_2?c_3?\} && (\{r, c_1, c_2, c_3\}) \end{aligned}$$

$$\begin{aligned} \Sigma_{12} &= \{r \to r/\_/b\} \\ \Sigma_{23} &= \{r/a_1 \to r/c_1;\ r/a_2 \to r/c_2\} \end{aligned}$$

Note that $\Sigma_{12}$ requires $b$ in the tree conforming to $D_2$, which in turn require $a_1$ *or* $a_2$ in the 'middle'. And the presence of $a_i$ induces $c_i$ in the tree conforming $D_3$, so that the semantics of the composition is as desired. In the rest we only give the necessary DTDs and stds since the idea is similar.

For an example for // just replace $\_$ with // in the above.

An example for $\rightarrow$ is similar:

$$\begin{aligned} D_1 &= \{r \to \varepsilon\} && (\{r\}) \\ D_2 &= \{r \to ab?c\} && (\{r, a, b, c\}) \\ D_3 &= \{r \to c_1?c_2?c_3?\} && (\{r, c_1, c_2, c_3\}) \end{aligned}$$

[1]Note that the tree $r[c_3]$ is not a solution of the tree $r$ under the mapping $[\![M_1]\!] \circ [\![M_2]\!]$. Our composition is expressed by $r \to r[c_1 \vee c_2]$ if we allow disjunction.

$$\Sigma_{12} = \varnothing$$
$$\Sigma_{23} = \{r[a \to c] \to r/c_1;\ r[b \to c] \to r/c_2\}$$

For inequality we make use of the fact that $x \neq y \implies x \neq z \vee y \neq z$ for all $x, y, z$:

$$D_1 = \{r \to \varepsilon\} \qquad (\{r\})$$
$$D_2 = \{r \to a^* b^* c^*\} \qquad (\{r, a[1], b[1], c[1]\})$$
$$D_3 = \{r \to c_1?c_2?c_3?\} \qquad (\{r, c_1, c_2, c_3\})$$

$$\Sigma_{12} = \{r \to r[a(x), b(y), c(z)], x \neq y\}$$
$$\Sigma_{23} = \{r[a(x), c(z)], x \neq z \to r/c_1;\ r[b(y), c(z)], y \neq z \to r/c_2\}$$

□

Thus we have to restrict the stds in the mappings. The following shows that we have to restrict the DTDs as well since disjunctions in DTDs naturally cause similar problems.

**Proposition 4.5** *The class of schema mappings where (a) disjunction or (b) repeat of the same label is used in the DTDs is not closed under composition, with fully specified stds.*

*Proof.* The idea is the same as above. For disjunction, it is actually easier. Consider

$$D_1 = \{r \to \varepsilon\} \qquad (\{r\})$$
$$D_2 = \{r \to a_1 \mid a_2\} \qquad (\{r, a_1, a_2\})$$
$$D_3 = \{r \to c_1?c_2?c_3?\} \qquad (\{r, c_1, c_2, c_3\})$$

$$\Sigma_{12} = \varnothing$$
$$\Sigma_{23} = \{r/a_1 \to r/c_1;\ r/a_2 \to r/c_2\}$$

It is easy to see that the semantics of the composition is the one required throughout the preceding proof, requiring $c_1$ or $c_2$ in the target.

For the repeat of the same label, consider:

$$D_1 = \{r \to a^*\} \qquad (\{r, a[1\})$$
$$D_2 = \{r \to bb\} \qquad (\{r, b[1]\})$$

$$D_3 = \{r \to \epsilon\} \qquad (\{r\})$$

$$\Sigma_{12} = \{r/a(x) \to r/b(x)\}$$
$$\Sigma_{23} = \varnothing$$

The first mapping states that the values in $a$ must be copied to those in $b$ in the second tree. Due to the DTD $D_2$, this means $a$ can have at most two different values. Again, we need disjunction in the mapping here to express the composition, i.e., $r[a(x), a(y), a(z)] \to x = y \vee y = z \vee z = x$. □

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

- wildcard, descendant, next-sibling, and inequalities in stds (by Proposition 4.4);
- non-nested-relational DTDs[2] (by Proposition 4.5);

We now eliminate them all: we look at mappings with fully-specified stds (given by (3.2), to eliminate wildcard, descendant, and next-sibling) with no inequalities and nested relational DTDs (to eliminate disjunctions and repeating labels).

For this class, a natural extension of stds with Skolem functions gives us closure under composition. We add Skolem functions to XML schema mappings as it was done for relational mappings in [48], by using terms in place of variables. For a valuation of function symbols $\bar{f}$ and a valuation of variables $\bar{a}$, the meaning of $T \vDash \varphi(\bar{t})[\bar{f}, \bar{a}]$ is as usual. For a mapping $M = (D_s, D_t, \Sigma)$ with Skolem functions, $(T, T') \in [\![M]\!]$ iff there exist functions $\bar{f}$ such that for each $(\varphi, \alpha_= \to \psi, \alpha'_=) \in \Sigma$ and each $\bar{a}$, if $T \vDash \varphi, \alpha_=[\bar{f}, \bar{a}]$ then $T' \vDash \psi, \alpha'_=[\bar{f}, \bar{a}]$. Note that we can use the same function symbol in more than one constraint.

**Theorem 4.6** *The class of mappings with Skolem functions and equality, restricted to nested-relational DTDs and fully specified stds, is closed under composition.*

The rest of this section is devoted to the proof of the above.

Like in the proof of Theorem 3.12, without loss of generality we may assume that there is a single attribute *@attr* available, it is only allowed in leaves, and each leaf has it. Since

[2] Strictly speaking, we have not settled the possibility that mappings that use expressions like $(ab)^*$ is closed under composition.

we are using Skolem functions, a valuation now assigns values in $\mathbb{N}$ to all variables and functions $\mathbb{N}^k \to \mathbb{N}$ to all function symbols.

Let us first describe the outline of the composing algorithm:

1. We are given two mappings with sets of dependencies $\Sigma_{12} = \{\varphi_i \to \psi_i | i \in [n]\}$ and $\Sigma_{23} = \{\gamma_i \to \delta_i | i \in [m]\}$;
2. Our goal is to compute $\Sigma_{13}$ that represents composition.
3. We modify dependencies in such a way that "missing" patterns in dependencies are remedied. E.g., if we have $r \to a; a \to b$ in the DTD and $r/a$ in an std, we make it $r/a/b$;
4. We go over all subsets $\{\varphi_{i_k}\}$ of the source patterns of $\Sigma_{12}$, and do the following;
   - Compute the set $\Delta$ of dependencies, a subset of $\{\delta_i | i \in [m]\}$ that are implied by $\{\psi_{i_k}\}$ with respect to $\Sigma_{23}$;
   - Add to $\Sigma_{13}$ a dependency

$$\bigwedge\{\varphi_{i_k}\} \to \bigwedge \Delta.$$

   (Note that we have the comma "," as conjunction.)

We turn to the more detailed description of the algorithm. First we need to formalise the "remedy" patterns, which we call completion. We denote by $\mathrm{Step}_\tau\Phi$ the set of children in $\Phi$ below $\tau$.

**Algorithm 1** Computing $\mathbf{Compl}(\sigma, \Phi, D)$.

**Require:** $\Phi$ is satisfiable wrt $D_\sigma$
  **if** $\sigma \to \sigma_1 \dots \sigma_k \rho_1? \dots \rho_\ell? \tau_1^* \dots \tau_m^*$ **then**
    $\Phi_1 := \{\mathbf{Compl}(\tau, \mathrm{Step}_\tau\Phi, D) \mid \tau = \sigma_1, \dots, \sigma_k\}$
    $\Phi_2 := \{\mathbf{Compl}(\tau, \mathrm{Step}_\tau\Phi, D) \mid \tau = \rho_1, \dots, \rho_\ell \text{ and } \mathrm{Step}_\tau\Phi \neq \varnothing\}$
    $\Phi_3 := \{\mathbf{Compl}(\tau, \mathrm{Step}_\tau\Phi, D) \mid \tau = \tau_1, \dots, \tau_m \text{ and } \eta \in \mathrm{Step}_\tau\Phi\}$
    **return** $\sigma[\Phi_1 \cup \Phi_2 \cup \Phi_3]$
  **else** // $\sigma$ is a leaf, and hence $\Phi$ is a singleton
    let $\{\varphi\} = \Phi$
    **return** $\varphi$
  **end if**

Let $\Phi$ be a set of tree patterns satisfiable with respect to a DTD $D$. Algorithm 1 invoked for $(r, \Phi, D)$ computes a tree pattern $\mathrm{Compl}_D(\Phi)$ called the *completion of* $\Phi$ *with respect*

*to D*. The intuition behind it is that of combining the conditions imposed on a tree by $\Phi$ and by $D$ together. For instance, if $\Phi$ requires the root to have an $a$-node, and DTD requires each $a$-node to have a $b$-node, then the completion should require a path $r/a/b$.

**Lemma 4.7** *For each set of fully specified tree patterns $\Phi$ satisfiable with respect to a nested-relational DTD $D$ it holds that for each $T \vDash D$ and each valuation $(\bar{f}, \bar{a})$, $T \vDash \Phi[\bar{f}, \bar{a}]$ iff $T \vDash \mathrm{Compl}_D(\Phi)[\bar{f}, \bar{a}]$.*

*Proof.* By induction on the depth of $D$ following the steps of Algorithm 1. □

An *embedding* of a tree pattern $\varphi$ in a tree pattern $\psi$ is a substitution of $\varphi$'s variables $\theta$ and mapping $h : \mathrm{Sub}(\varphi) \to \mathrm{Sub}(\psi)$ preserving

- the labelling: $h(\ell[\lambda]) = \ell[\lambda']$),
- the structure of tree patterns: if $h(\ell[\varphi_1, \ldots, \varphi_m]) = \ell[\psi_1, \ldots, \psi_n]$, then $h(\varphi_i) \in \{\psi_1, \ldots, \psi_n\}$ for all $i$,
- and the structure of terms $(h(\ell(t)) = (\ell(t))\theta)$.

**Lemma 4.8** *If $(h, \theta)$ is an embedding of $\varphi$ in $\psi$, then for all $T$ and $(\bar{f}, \bar{a})$, if $T \vDash \psi[\bar{f}, \bar{a}]$ then $T \vDash \varphi\theta[\bar{f}, \bar{a}]$.*

*Proof.* Let $\xi$ be a realisation of $\psi$ in $T$ with the valuation $\bar{f}, \bar{a}$. From the definition of embedding it follows that $\xi \circ h$ is a realisation of $\varphi\theta$ in $T$ with the valuation $\bar{f}, \bar{a}$. □

Before presenting the algorithm to compute the composition of schema mappings, we need one more definition. A fully-specified pattern $\varphi$ can be seen as a conjunction of (fully-specified) *paths*; for example $r[a[b, c]]$ is a conjunction of $r/a/b$ and $r/a/c$. As seen in the proof of Proposition 4.5, DTDs impose some cardinality constraints on the number of different data values in the source document. The important property of nested relational DTDs is that each path is admitted at most once or arbitrarily many. We say a path $\ell_1/\ell_2/\cdots/\ell_k$ is *non-unique* in a DTD $D$ if one of $\ell_i$'s appears as $\ell_i^*$ or $\ell^+$ in $D$; otherwise it is unique. Note that non-unique paths may contain prefix that is a unique path in $D$. We call these prefixes *unique prefixes*.

Now we present the algorithm to compose mappings in Algorithm 2. In the algorithm and the proof below we use notation $r[\Phi]$ for $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ to denote $r[\varphi_1, \varphi_2, \ldots, \varphi_n]$.

**Algorithm 2** Composing schema mappings.

// The goal is to construct $\Sigma_{13}$ that expresses $[\![M_{12}]\!] \circ [\![M_{23}]\!]$
$\Sigma_{13} := \{\varphi, \varphi_= \to \psi_= \mid \varphi, \varphi_= \to \psi, \psi_= \in \Sigma_{12}\}$
**for all** $\varphi(\bar{x}), \varphi_=(\bar{x}) \to \psi(\bar{x}), \psi_=(\bar{x}) \in \Sigma_{12}$ **do**
  **for all** paths $p$ in $\psi$ and unique prefixes of the form $p'/\ell(t)$ of $p$ **do**
    $\bar{x}' = (x'_1, \ldots, x'_p)$// A tuple of fresh variables
    $\Sigma_{13} := \Sigma_{13} \cup \{\varphi(\bar{x}), \varphi(\bar{x}'), \varphi_=(\bar{x}) \cup \varphi_=(\bar{x}') \to t = t[\bar{x}'/\bar{x}]\}$
  **end for**
**end for**
$m := \max_{(\gamma \to \delta) \in \Sigma_{23}} |\mathrm{Sub}\,\gamma|$
$\widehat{\Sigma}_{12} := \varnothing$
**for all** $\varphi(\bar{x}), \varphi_=(\bar{x}) \to \psi(\bar{x}), \psi_=(\bar{x}) \in \Sigma_{12}$ **and** $i = 1, 2, \ldots, m$ **do**
  $\bar{x}_i := (x_{1,i}, x_{2,i}, \ldots, x_{p,i})$
  $\widehat{\Sigma}_{12} := \widehat{\Sigma}_{12} \cup \{\varphi(\bar{x}_i), \varphi_=(\bar{x}_i) \to \psi(\bar{x}_i), \psi_=(\bar{x}_i)\}$
**end for**
// $\widehat{\Sigma}_{12}$ contains $m$ copies of each std from $\Sigma_{12}$
**for all** $\Gamma \subseteq \widehat{\Sigma}_{12}$ **do**
  let $\Gamma = \{r[\Phi_i], \varphi^i_= \to \psi_i, \psi^i_= \mid i = 1, \ldots, k\}$
  $\Psi := \{\psi_1, \ldots, \psi_k\}$
  **if** $\Psi$ is not satisfiable with respect to $D_2$ **then**
    $\Sigma_{13} := \Sigma_{13} \cup \{r[\Phi_1 \cup \ldots \cup \Phi_k], \varphi^1_=, \ldots, \varphi^k_= \to \bot\}$
  **else**
    **for all** $\gamma, \gamma_= \to \delta, \delta_= \in \Sigma_{23}$ **do**
      **for all** embeddings $(\theta, h)$ of $\gamma$ in $\mathbf{Compl}(r, \Psi, D_2)$ **do**
        $\Sigma_{13} := \Sigma_{13} \cup \{r[\Phi_1 \cup \ldots \cup \Phi_k], \varphi^1_=, \ldots, \varphi^k_=, \gamma_=\theta \to \delta\theta, \delta_=\theta\}$
      **end for**
    **end for**
  **end if**
**end for**

*Proof of Theorem 4.6.* Before we proceed with the proof, let us justify three simplifications. First, we would like to enforce that leaf labels are only used in formulae of the form $\sigma(x)$ and never as $\sigma$. One easily transforms each constraint to an equivalent one satisfying this property: simply replace each occurrence of $\sigma$ with a formula $\sigma(z)$ for a fresh variable $z$. Second, we need all the variables to occur on the source side. To transform a constraint to

this form replace each variable $y$ occurring only on the target side with $f_y(x_1, x_2, \ldots, x_n)$ where $f_y$ is a fresh function symbol and $x_1, x_2, \ldots, x_n$ are all the variables used on both sides of the constraint. Finally, it will be convenient to have no functions symbols and no repetitions of variables in the tree pattern on the source side: we replace all nontrivial terms and all repeated occurrences of variables with fresh variables, and add a suitable equality to the source side.

Thus, without loss of generality we may assume that we have $M_{12} = \langle D_1, D_2, \Sigma_{12}\rangle$ and $M_{23} = \langle D_2, D_3, \Sigma_{23}\rangle$ satisfying all three restrictions. Let $\langle D_1, D_3, \Sigma_{13}\rangle$ be the mapping computed by Algorithm 2. Let us prove that indeed $M_{12} \circ M_{23} = M_{13}$. Assume that $T_1, T_2, T_3$ satisfy $(T_1, T_2) \in [\![M_{12}]\!]$ and $(T_2, T_3) \in [\![M_{23}]\!]$. We need to show that $(T_1, T_3) \in [\![M_{13}]\!]$. Observe that the algorithm does not introduce new function symbols. Let $\bar{f}$ be a valuation of the function symbols witnessing $(T_1, T_2) \in [\![M_{12}]\!]$ and $(T_2, T_3) \in [\![M_{23}]\!]$. Pick a constraint from $\Sigma_{13}$. Suppose first that it is of the form

$$\varphi, \varphi_= \to \psi_=$$

with $\varphi, \varphi_= \to \psi, \psi_= \in \Sigma_{12}$. Take a valuation of variables $\bar{a}$ such that $T_1 \vDash \varphi, \varphi_=[\bar{f}, \bar{a}]$. Then $T_2 \vDash \psi_=[\bar{f}, \bar{a}]$, and so $T_3 \vDash \psi_=[\bar{f}, \bar{a}]$, since the satisfaction of equalities depends only on the valuations and not on the trees. We can similarly prove the validity of dependencies of the form

$$\varphi(\bar{x}), \varphi(\bar{x}'), \varphi_=(\bar{x}) \cup \varphi_=(\bar{x}') \to t = t[\bar{x}'/\bar{x}].$$

Next, suppose that the constraint is of the form

$$r[\Phi_1 \cup \ldots \cup \Phi_k], \varphi^1_=, \ldots, \varphi^k_= \to \bot$$

with $\{r[\Phi_i], \varphi^i_= \to \psi_i, \psi^i_= \mid i = 1, \ldots, k\} \subseteq \widehat{\Sigma}_{12}$ and $\Psi = \{\psi_1, \ldots, \psi_k\}$ not satisfiable with respect to $D_2$. Since $T_2$ cannot satisfy $\Psi$ for any valuation, $T_1$ does not satisfy $r[\Phi_1 \cup \ldots \cup \Phi_k], \varphi^1_=, \ldots, \varphi^k_=$ for any valuation, and so $(T_1, T_3)$ satisfies this constraint.

Finally, assume the constraint is of the form

$$r[\Phi_1 \cup \ldots \cup \Phi_k], \varphi^1_=, \ldots, \varphi^k_=, \gamma_=\theta \to \delta\theta$$

where $\{r[\Phi_i], \varphi^i_= \to \psi_i, \psi^i_= \mid i = 1, \ldots, k\} \subseteq \widehat{\Sigma}_{12}$, $\Psi = \{\psi_1, \ldots, \psi_k\}$, $(\gamma, \gamma_= \to \delta, \delta_=) \in \Sigma_{23}$, and $(\theta, h)$ is an embedding of $\gamma$ in $\mathrm{Compl}_{D_2}(\Psi)$. Pick a valuation (of variables) $\bar{a}$ such that $T_1 \vDash r[\Phi_1 \cup \ldots \cup \Phi_k], \varphi^1_=, \ldots, \varphi^k_=, \gamma_=\theta[\bar{f}, \bar{a}]$. Then $T_2 \vDash \Psi[\bar{f}, \bar{a}]$ and in consequence $T_2 \vDash \mathrm{Compl}_{D_2}(\Psi)[\bar{f}, \bar{a}]$. By Lemma 4.8, $T_2 \vDash \gamma\theta[\bar{f}, \bar{a}]$, so $T_2 \vDash \gamma\theta, \gamma_=\theta[\bar{f}, \bar{a}]$. Hence,

$T_3 \vDash \delta\theta[\bar{f}, \bar{a}]$.

Thus we have proved $[\![M_{12} \circ M_{23}]\!] \subseteq [\![M_{12}]\!]$. Let us check that the converse inclusion holds as well. Take $(T_1, T_3) \in [\![M_{13}]\!]$. We need to define an intermediate tree $T_2$ such that $(T_1, T_2) \in [\![M_{12}]\!]$ and $(T_2, T_3) \in [\![M_{23}]\!]$. Let $\bar{f}$ be the witnessing valuation of function symbols. Let

$$\Delta = \{\psi_{\bar{a}} \mid \varphi, \varphi_= \to \psi, \psi_= \in \Sigma_{12} \text{ and } T_1 \vDash \varphi, \varphi_=[\bar{f}, \bar{a}]\},$$

where $\psi_{\bar{a}}$ is obtained by replacing each variable $x$ in $\psi$ with $z_{\bar{a}(x)}$ and $z_1, z_2, \ldots$ is a fixed set of fresh variables. Observe that if $T_1 \vDash \varphi, \varphi_=[\bar{f}, \bar{a}]\}$ for some $\varphi, \varphi_= \to \psi, \psi_= \in \Sigma_{12}$, then by $\Sigma_{13}$ the conjunction of equalities $\psi_=[\bar{f}, \bar{a}]$ is satisfied. Hence, for the valuation $\jmath$ assigning $j$ to $z_j$, a tree $T_2 \vDash D_2$ is a solution for $T_1$ if and only if $T_2 \vDash \Delta[\bar{f}, \jmath]$.

Suppose that $\Delta$ is not satisfiable wrt to $D_2$. Since $D_2$ is nested-relational, there exists a formula $\delta \in \Delta$ which is not satisfiable wrt $D_2$, and further, there exists a constraint $\varphi, \varphi_= \to \psi, \psi_=$ in $\Sigma_{12}$, such that $T_1 \vDash \varphi, \varphi_=[\bar{f}, \bar{a}]$ and $\psi$ is not satisfiable wrt $D_2$. But in such case $\Sigma_{13}$ contains a constraint $\varphi, \varphi_= \to \bot$, and thus $T_3 \vDash \bot$, which is a contradiction.

Hence, $\Delta$ is satisfiable and we can consider $\mathrm{Compl}_{D_2}(\Delta)$. Let $T_2$ be a tree obtained from $\mathrm{Compl}_{D_2}(\Delta)$ in the usual way with the terms evaluated according to $\bar{f}, \jmath$. From the definition of completion it follows that $T_2 \vDash D_2$. It is also straightforward to check that $T_2 \vDash \mathrm{Compl}_{D_2}(\Delta)[\bar{f}, \jmath]$. By Lemma 4.7, $T_2 \vDash \Delta[\bar{f}, \jmath]$ and so $(T_1, T_2) \in [\![M_{12}]\!]$ with the valuation $\bar{f}$ witnessing it.

It remains to verify that $(T_2, T_3) \in [\![M_{23}]\!]$. Pick a constraint $\gamma, \gamma_= \to \delta, \delta_=$ from $\Sigma_{23}$. Suppose that $T_2 \vDash \gamma, \gamma_=[\bar{f}, \bar{a}]$ and let $\xi$ be a witnessing realisation. Since $\gamma$ uses each variable once and contains no function symbols, $\xi$ assigns to each $x$ the data value $\bar{a}(x)$ stored in the attribute of a node $v_x$. Let $\theta(x)$ be the term from $\mathrm{Compl}_{D_2}(\Delta)$ that yielded the data value stored in $v_x$. Obviously, $\theta(x)[\bar{f}, \jmath] = \bar{a}(x)$.

By the definition of $T_2$ we may interpret $(\theta, \xi)$ as an embedding of $\gamma$ in $\mathrm{Compl}_{D_2}(\Delta)$. Examining Algorithm 1 reveals to see that $(\theta, \xi)$ can be also seen as an embedding in $\mathrm{Compl}_{D_2}(\Delta')$, where $\Delta' \subseteq \Delta$ and $|\Delta'| \leq |\mathrm{im}\, h| \leq |\mathrm{Sub}\, \gamma|$. But then in Algorithm 2, for some $\Gamma = \{r[\Phi_i], \varphi^i_= \to \psi_i \mid i = 1, \ldots, k\} \subseteq \widehat{\Sigma}_{12}$ and $\Psi = \{\psi_1, \ldots, \psi_k\}$ we have $\Psi = \Delta'$ (up to a renaming of variables which we omit for the sake of simplicity), which entails that $(\theta, \xi)$ is an embedding of $\gamma$ in $\mathrm{Compl}_{D_2}(\Psi) = \mathrm{Compl}_{D_2}(\Delta')$. By Algorithm 2, $\Sigma_{13}$ contains a constraint $r[\Phi_1 \cup \ldots \cup \Phi_k], \varphi^1_=, \ldots, \varphi^k_=, \gamma_=\theta \to \delta\theta, \delta_=\theta$.

On the other hand, by the definition of $\Delta$, for some valuation $\jmath'$ coinciding with $\jmath$ on the variables used in $\theta$, it holds that

$$T_1 \vDash r[\Phi_1 \cup \ldots \cup \Phi_k], \varphi^1_=, \ldots, \varphi^k_=[\bar{f}, \jmath'].$$

Furthermore, the equalities$\gamma_{=}\theta[\bar{f}, \jmath']$ hold since $\theta(x)[\bar{f}, \jmath'] = \theta(x)[\bar{f}, \jmath] = \bar{a}(x)$. Hence, $T_3 \vDash \delta\theta, \delta_{=}\theta[\bar{f}, \jmath]$, and so $T_3 \vDash \delta, \delta_{=}[\bar{f}, \bar{a}]$.

□

# Chapter 5

# Query Answering

In this chapter we consider the problem of query answering in XML data exchange. Following previous approaches, we adapt the certain answers semantics. The central result in [12] about query answering was the dichotomy on classes of DTDs: they identified a class of DTDs (called univocal) for which query answering is tractable and any extension of which leads to intractable query answering.

It turns out that queries with extended features are almost always intractable while we can feasibly answer simple queries in extended mappings.

After giving formal definition of our query language, we show that the query answering can be done in coNP. We then go on to prove the tractable algorithm for extended mappings. Finally we explore various (im)possibilities, trying to obtain cases where queries with extended features become tractable.

## 5.1 Query Language

For querying the XML documents we use the same language as for the dependencies: tree patterns augmented with equalities and inequalities. Additionally, we allow projection. Thus, a query is an expression of the form $\exists \bar{x}(\pi_1 \wedge \cdots \wedge \pi_n, \alpha_{=,\neq})$, where each $\pi_i$ is a tree pattern and $\alpha_{=,\neq}$ is a set of equalities and inequalities. The semantics is defined in the obvious way. This class of queries is denoted by $\mathbf{CTQ}^{\neq}$ (conjunctive tree queries with inequality).

We also consider unions of such queries: $\mathbf{UCTQ}^{\neq}$ 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 $\mathbf{CTQ}^{\neq}$. 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

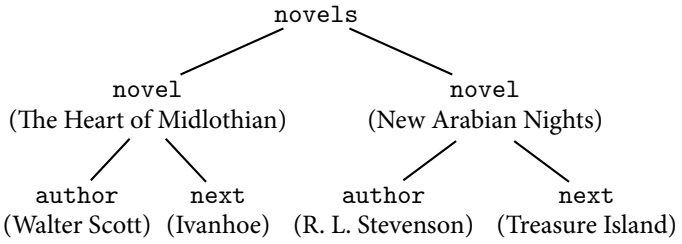



FIGURE 5. a small Scottish literature library, to be queried

queries using only the symbols in $\sigma$.

As an example, let us consider querying the document depicted in 5, the small Scottish Literature Library that appeared in Chapter 3.

The following is a simple query asking who are the authors in the library:

$$\exists x(\texttt{novels}[\texttt{novel}(x)[\texttt{author}(y)]]).$$

Query this over the library returns the set $\{$Walter Scott, R. L. Stevenson$\}$.

Yet another example is

$$\texttt{novels}[\texttt{novel}(y)[\texttt{author}(x)]] \cup \exists z(\texttt{novels}[\texttt{novel}(z)[\texttt{author}(x), \texttt{next}(y)]]),$$

which returns the pairs of author and his work, i.e., the set

$$\{\langle\text{Walter Scott, The Heart of Midlothian}\rangle, \langle\text{Walter Scott, Ivanhoe}\rangle,$$
$$\langle\text{R. L. Stevenson, New Arabian Nights}\rangle, \langle\text{R. L. Stevenson, Treasure Island}\rangle\}.$$

## 5.2 Certain Answers

Data exchange is the problem of transforming data in the source schema into data in the target schema, according to stds. As we have already explained, the result of this transformation may not be unique. The fundamental problem of data exchange is to answer queries over the target data. Suppose we are given a mapping $M$, a query $Q$, and a source tree $T$ conforming to $D_s$. What answer should we return if there is more than one solution for $T$? Following [12, 45], we adapt the *certain answers semantics*, i.e., we return the tuples which would be returned for every possible solution:

$$\text{certain}_M(Q, T) = \bigcap\{Q(T') \mid T' \text{ is a solution for } T \text{ under } M\}.$$

In particular, when $Q$ is a Boolean query, we say that $\text{certain}_M(Q, T)$ is true if and only if $Q$ is true for all the solutions.

Fix an XML schema mapping $M$ and a query $Q$. We are interested in the following computational problem.

| | |
|---|---|
| PROBLEM: | $\text{CERTAIN}_M(Q)$ |
| INPUT: | a tree $T$, a tuple $\bar{s}$ |
| QUESTION: | $\bar{s} \in \text{certain}_M(Q, T)$ ? |

Note that we are working on *data complexity* [99]. In the following, we focus on the fragments of the problem, where "simple" generally means downward mappings/queries.

## 5.3 Complexity

The structure of this section is as follows. We first summarise the known results in the following subsection; Next we prove that the coNP upper bound still holds and then present PTIME algorithm for simple queries in extended settings; Finally we deal with queries with horizontal order, showing that for them tractability cannot be achieved in any reasonable mapping.

### 5.3.1 Simple mappings

First we recall from [12] what is already known about simple settings, i.e., mappings from $\text{SM}(\downarrow, \downarrow^*, =)$ and queries from $\mathbf{UCTQ}(\downarrow, \downarrow^*, =)$. In general the problem is co-NP complete. There are three natural parameters of the problem: DTDs, stds, 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 guessing: 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. But guessing is also involved whenever wildcard and descendant are used in the stds. The mappings which do not use $\downarrow^*$ nor _ in the stds are called *fully specified*. The following theorem summarises the results on simple mappings.

**Theorem 5.1 (Arenas & Libkin [12])** *For a schema mapping $M \in \text{SM}(\downarrow, \downarrow^*, =)$ and a query $Q \in \mathbf{CTQ}(\downarrow, \downarrow^*, =)$*

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

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

In fact, for fully specified mappings there is a dichotomy in the first parameter: if there is enough nondeterminism in the DTDs, 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 [12] for a definition). Intuitively, it extends nested-relational DTDs with just a bit of disjunction.

In addition to restrictions on DTDs and mappings, we must also disallow inequality in the query languages, as it leads to coNP hardness already in the relational case [45, 74]. Since the usual translation form the relational setting to the XML setting produces fully specified nested relational mappings, we have the following result.

**Proposition 5.2** *There exist a nested-relational fully specified schema mapping $M \in \mathrm{SM}(\downarrow)$ and a query $Q \in \mathbf{CTQ}(\downarrow, =, \neq)$ such that* $\textsc{certain}(Q)$ *is coNP-complete.*

Now our question is the following: Can we find $\sigma_1 \supseteq \{\downarrow, =\}$ and $\sigma_2 \supseteq \{\downarrow, \downarrow^*, =\}$ such that $\textsc{certain}_M(Q)$ is tractable for all $M \in \mathrm{SM}^{nr}(\sigma_1)$ and $Q \in \mathbf{UCTQ}(\sigma_2)$. In what follows we show that it is possible to extend schema mappings, but it is almost impossible to extend the query language.

### 5.3.2 Extended mappings

We consider the mapping with horizontal ordering and data comparisons in this section. Let us see that the coNP upper bound carries over to the case where we have the full-fledged mappings and queries. The proof is done by making clear the connection of our problem with query answering in incomplete XML [16]. Also of note is that the proof of the coNP upper bound in [12] contains a bug and that our proof here remedies it.

**Proposition 5.3** *For every mapping $M \in \mathrm{SM}(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, =, \neq)$ and every query $Q \in \mathbf{CTQ}(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, =, \neq)$, the problem* $\textsc{certain}_M(Q)$ *is in coNP.*

*Proof.* We reduce the problem to the problem of computing certain answers over XML with incomplete information.

First we describe the problem of query answering over XML with incomplete information (see [16] for details).

An *XML tree with incomplete information*, or simply incomplete XML tree, is a pair $\langle \pi(\bar{t}), \alpha_{=,\neq}(\bar{t}) \rangle$, where $\pi(\bar{t})$ is a partially evaluated tree pattern and $\alpha_{=,\neq}$ is the set of equalities and inequalities.

Formally, the pattern language for incomplete XML trees, is defined by the grammar[1]:

$$\begin{aligned} \pi &:= \ell(\bar{t})[\lambda] && \text{patterns} \\ \lambda &:= \epsilon \mid \mu \mid //\pi \mid \lambda, \lambda && \text{sets} \\ \mu &:= \pi \mid \pi \to \mu \mid \pi \to^* \mu && \text{sequences} \end{aligned}$$

Here $\bar{t}$ is a sequence of constants and variables.

The notion of certain answers over an incomplete tree $\langle \pi, \alpha \rangle$ is introduced in the natural way. Given a query $Q$ in $\mathbf{UCTQ}^{\neq}$ and a DTD $D$, we define:

$$\text{certain}_D(Q, \langle \pi, \alpha \rangle) = \bigcap \{ Q(T) | T \vDash D \text{ and } T \vDash \pi, \alpha \}.$$

The query answering over incomplete trees can be formulated as:

| | |
|---|---|
| PROBLEM: | INC-CERTAIN$_D(Q)$ |
| INPUT: | an incomplete tree $\langle \pi, \alpha_{=,\neq} \rangle$, tuple $\bar{s}$ |
| OUTPUT: | yes if $\bar{s} \in \text{certain}_D(Q, \langle \pi, \alpha \rangle)$ |

**Proposition 5.4 (Essentially [16])** *For any fixed query $Q \in \mathbf{UCTQ}^{\neq}$ and any fixed DTD $D$, INC-CERTAIN$_D(Q)$ is in coNP.*

*Proof.* The statement of theorem 6.1 in [16] is actually for an incomplete trees without inequalities and queries in $\mathbf{UCTQ}(\downarrow, \downarrow^*, \to, \to^*, =)$, but the proof is valid in the setting above as well.

The idea of the proof is standard: we exhibit a polynomial size counterexample. Without loss of generality, we may consider only Boolean queries. Fix a query $Q$, a DTD $D$, and suppose an incomplete XML tree $\langle \pi, \alpha \rangle$ is given. If $\text{certain}_D(Q, \langle \pi, \alpha \rangle)$ is false, there is a counter example, i.e., a tree $T$ that satisfies $\langle \pi, \alpha \rangle$ and falsifies $Q$. We prove that we can prune $T$ in such a way that the resulting $T'$ of polynomial size is still a counterexample. First we label all the nodes in $T$ witnessing $\langle \pi, \alpha \rangle$. Consider an FO formula equivalent to $Q$, and let $k$ be its quantifier rank. Then, roughly, for any pair $u, v$ of non-witnessing nodes with the same FO rank-$k$ type[2], we cut the nodes in-between and merge $u, v$ (provided that cutting neither removes any witnessing node nor leads to violation of the DTD). Note

[1] The notation is different in [16]. For instance, $\|$ is used in places where we use ','(comma) in patterns (to mean the patterns are unordered). Also they have extra features such as marking node with 'first-child' and so on.

[2] Two nodes having the same FO rank-$k$ types means that they satisfy the same FO formulae with at most quantifier rank $k$.

that the number of FO rank-$k$ types is finite. Since it depends only on the query, it is fixed. By cutting this way, vertically and horizontally, we make sure all the witnesses are not too far apart, and the resulting tree has polynomial size.

Thus guessing a counterexample of polynomial size provides coNP algorithm for the problem $\text{INC-CERTAIN}_D(Q)$. □

We now show how to reduce the query answering in XML data exchange to the query answering in incomplete XML trees. Suppose that we have a query $Q \in \mathbf{UCTQ}^{\neq}$ and an XML schema mapping $M = \langle D_s, D_t, \Sigma \rangle$. Given a tree $T$, we construct an incomplete XML tree $\Psi$ such that $\text{certain}_M(Q, T)$ is equivalent to $\text{certain}_{D_t}(Q, \Psi)$. For each dependency in $\Sigma$ of the form

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

and for each pair of tuples $\bar{s}, \bar{t}$ such that $T \models \varphi(\bar{s}, \bar{t}), \alpha(\bar{s}, \bar{t})$, the constraint (on the target) is a tree pattern $\psi(\bar{s}, \bar{z}')$ and a set of (in)equalities $\alpha'(\bar{s}, \bar{z}')$, where $\bar{z}'$ is a renaming of $\bar{z}$. We collect all such patterns, of which there are at most polynomially many: at most $|T|^{|x|+|y|}$ for each dependency and the number of dependency is fixed.

Now let $\Psi$ be $\langle \eta, \beta \rangle$, where $\eta$ is the result of merging at the root all the patterns that are obtained as above (for all the stds) and $\beta$ is the union of all the (in)equalities (for all the stds). It is straightforward to see that the problem of $\text{certain}_M(Q, T)$ is equivalent to $\text{certain}_{D_t}(Q, \Psi)$. Since $\Psi$ is polynomial in the size of the input, the coNP algorithm for $\text{certain}_{D_t}(Q, \Psi)$ gives a coNP algorithm for $\text{certain}_M(Q, T)$. □

**Theorem 5.5** *Suppose $M$ is a fully specified schema mapping in* $\text{SM}^{nr}(\downarrow, \rightarrow, \rightarrow^*, =, \neq)$ *and* $q \in \mathbf{UCTQ}(\downarrow, \downarrow^*, =)$. *Then* $\text{CERTAIN}(Q)$ *is in PTIME.*

*Proof.* The algorithm follows the same lines with the one in Theorem 6.2 [12]. It constructs a *canonical solution* $T^*$ such that $Q(T) = \text{certain}_M(Q, T)$, working in two stages: First, it constructs a tree called *canonical presolution*. Second, it tries to make it conform to the target DTD. With the extended mapping language, we have to do another check between these two stages, as shown below. Due to the extended mapping language, the merging part becomes more complicated.

First of all, we note an important property of queries in $\mathbf{UCTQ}(\downarrow, \downarrow^*, =)$: Intuitively, it is ignorant to the (horizontal) ordering in trees so that it does not matter whether solutions are ordered.

Before we state the proposition formally, we need some definitions. An unordered XML tree $\mathbf{T}$ is a structure $\langle T, \downarrow, lab_T, (\rho_a)_{@a \in Att} \rangle$, where $T$ is a tree domain, $lab_T$ is a

labeling element types to each node, $\rho_a$ assigns an attribute value for $@a$ as appropriate. We will simply write $T$ instead of $\mathbf{T}$ below.

An unordered XML tree $T'$ is a solution for $T$ if there exists an (ordered) XML tree $\hat{T}$ into which $T'$ can be obtained by ignoring the sibling order relation. For a schema mapping $M$ and a query $Q$, $\text{certain}^*_M(Q, T)$ denotes the set of unordered XML trees that are solutions for $T$ under $M$.

Now we can formalise the property. The following essentially says 'it does not matter whether or not solutions are ordered'.

**Proposition 5.6** *Let $M$ be an XML schema mapping, $T$ an XML tree and $Q$ a query in* $\mathbf{UCTQ}(\downarrow, \downarrow^*, =)$. *Then the following holds:*

$$\text{certain}_M(Q, T) = \text{certain}^*_M(Q, T)$$

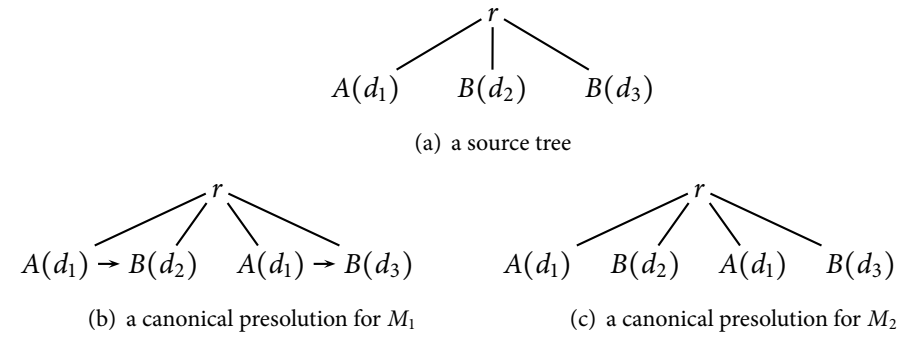
With this property at hand, we describe the algorithm that, given a source tree $T$, constructs an unordered XML tree $T^*$ such that $\text{certain}_M(Q, T) = Q(T^*)$. In the rest of the proof, without loss of generality, we assume that there is no element type that appears in the mapping without associated attributes. That is, if an element type $a$ has an attribute, it always used in the mapping as $a(x)$ and not just as $a$.

As mentioned earlier, the first step is the construction of the *canonical presolution*. The canonical presolution is simply the result of putting together all the target patterns required by the given source tree $T$. Formally for each fully-specified dependency of the form $\varphi(\bar{x}, \bar{y}), \alpha_{=,\neq}(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{z}), \alpha'_{=,\neq}(\bar{x}, \bar{z})$ and for every tuples $\bar{s}, \bar{s}'$ with $T \models \varphi, \alpha_{=,\neq}[\bar{s}, \bar{s}']$, we have $\psi(\bar{s}, \bar{\perp}), \alpha'(\bar{s}, \bar{\perp})$ where $\bar{\perp}$ is a sequence of distinct nulls and $\alpha'_{=,\neq}$ is a set of (in)equalities. In the end we have $\psi(\bar{s}, \bar{t}), \alpha'(\bar{s}, \bar{t})$, where $\bar{s}$ is a tuple of strings and $\bar{t}$ is a tuple of constants and nulls. We have polynomially many of these patterns and polynomially many equalities and inequalities, which we denote by $\boldsymbol{\alpha}$. We further replace nulls in the patterns as specified by $\boldsymbol{\alpha}$, that is, we put the same null if $\boldsymbol{\alpha}$ so specifies by equality, and replace a null with a constant if it is equalised to a constant in $\boldsymbol{\alpha}$. Finally, we obtain a canonical presolution merging all the patterns at the root.

At this stage, if $\boldsymbol{\alpha}$ contains any non-satisfiable (in)equality[3], we return 'no solution'. The size of the canonical presolution is polynomial and the construction can be done in PTIME.

**Example** We describe the computation of canonical presolutions by example. Consider

[3] E.g., if there is a dependency like $x \neq y \rightarrow x = y$, then there could be an equality like $1 = 2$ in $\boldsymbol{\alpha}$.




(a) a source tree

(b) a canonical presolution for $M_1$

(c) a canonical presolution for $M_2$

FIGURE 6. a source tree and its canonical presolutions

two simple mappings, one with sibling-order and the other without,

$$M_1 = \langle D_s, D_t, \{r[a(x), b(y)] \to r[a(x) \to b(y)]\}\rangle;$$
$$M_2 = \langle D_s, D_t, \{r[a(x), b(y)] \to r[a(x), b(y)]\},$$

where $D_s, D_t$ are both $r \to ab^*$.

For the tree depicted in Figure 5.6(a), its canonical presolutions for $M_1$ and $M_2$ will be those in Figures 5.6(b) and 5.6(c), respectively.

We now have a canonical presolution that contains all the constraints induced by the source tree $T$ under the mapping. Obviously the problem is that it might not conform to the target DTD. The second part of the algorithm processes the canonical presolution to enforce the conformation to the target DTD. To do this, we work level by level, from the root to the bottom, trying to 'repair' each level so that it is compatible with the corresponding regular language.

Observe that the sibling-order in target trees imposes the extra constraint on whether or not a solution exists. Again consider the example above. In the tree of Figure 5.6(b) we have to merge the two '$A(d_1)$'s so that the tree conforms the DTD $r \to ab^*$. In the unordered case of 5.6(c) we could just merge them, but this time we have to maintain the next-sibling relation, which is impossible since we cannot merge $B(d_2)$ and $B(d_3)$. Thus we have to do a more careful check for the existence of solution than when we don't have order in the mapping. (Note that this is not complete check for existence of solution: it might turn out that there is no solution later on.)

The repair is done in several steps.

First, we check all the sequences appearing in the target are compatible with (appropriate) regular expressions. For instance, if a sequence $a \to c$ appears in the pattern while

the corresponding regular expression is $abc$, then we exit the algorithm with "no solution".

Second, we put the nodes that are required by the target DTD but not present in the canonical presolution. For example if the regular expression under consideration is $abc$ (or $a^+bc$ or $abc^+$ or $a^+bc^+$) and we have only a node labeled $b$, then we add nodes labeled $a$ and $c$.

Third, we will do the *sequence consistency check*. We have to merge and at the same time check if the all the next-sibling relation can be maintained upon merging. Consider a pair of sequences appearing in the canonical presolution of the form $\sigma(d_1) \to \tau(d_2)$ and $\sigma(d_1) \to \tau(d_3)$, where $\sigma, \tau$ are element types, $d_i$'s are distinct data values. Further assume that $\sigma$ appears in the target DTD as $\sigma$ or $\sigma?$ (i.e., *unstarred*) and that $\tau$ appears as $\tau^*$ or $\tau^+$ (i.e., *starred*). In this case we exit the algorithm with "no solution". The following cases are also treated in the similar way:

- $\sigma \to \tau_1$ and $\sigma \to \tau_2$, where $\tau_1, \tau_2$ are distinct element types;
- the symmetric case where we have $\sigma$ starred and $\tau$ unstarred;
- $\sigma \to \tau(\bot_1)$ and $\sigma \to \tau(\bot_2)$ and $\bot_1 \neq \bot_2$ is in $\boldsymbol{\alpha}$.
- sequences longer than two can be checked in the same way.

We will do the above check for all the pairs of sequences in the canonical presolution. The number of these pairs is at most quadratic in the size of canonical presolution, so that we can do it in polynomial time.

Finally, after we process all the sequences in the presolution, we will remove all the next-/following-sibling orders and replace them with commas. Then we do the *node merging*. For any pair of nodes $a(d_1)$ and $a(d_2)$, we merge if $a$ appears in the current regular expression as $a$ or $a?$. Thus, as before, if $d_1 = d_2$ we successfully merge and go on to the next pair of nodes; otherwise we stop and return 'no solution'. Similarly, if we have $a(\bot_1)$ and $a(\bot_2)$ and have to merge them, we return 'no solution' when $\bot_1 \neq \bot_2 \in \boldsymbol{\alpha}$; otherwise we remove $a(\bot_2)$ and replace $\boldsymbol{\alpha}$ with $\boldsymbol{\alpha}[\bot_1/\bot_2]$, i.e., the set of (in)equalities where each $\bot_2$ in $\boldsymbol{\alpha}$ is replaced with $\bot_1$.

We repeat the repair procedure above for each level. Clearly this algorithm terminates because of the following two facts: First, the target DTD is nonrecursive and hence of constant depth. Second, we add nodes only when it is required by the DTD so that we add only constant number of children under each node in the canonical presolution.

When the algorithm terminates, we have either "no solution", which means there is no solution for the input tree $T$, or a tree $T^*$. $T^*$ is a solution, i.e., satisfying the target DTD

and all the tree patterns imposed by $T$. Also it is a minimal solution, in the sense that for any solution $T'$, there is a homomorphism from $T^*$ to $T'$. The existence of homomorphism can be proved using the fact that we put everything imposed by the mapping and the source tree. From this property we can conclude $\text{certain}_M(Q, T) = Q(T^*)$. □

### 5.3.3 Extended query language

In this section we examine various possibilities of extending the query language. We start with the simple mappings, for which tractability was obtained in [12], and show that extensions of query language lead immediately to intractability. Then we consider two modifications of this simple class, exploring two complementary directions. The first deals with mappings which fully specify the sibling order, the second investigates mappings based on DTDs invariant under sibling permutations.

#### Simple mappings

We first show that the class of mappings for which we can answer $\mathbf{UCTQ}(\downarrow, \downarrow^*, =)$ feasibly [12]. The next theorem shows that use horizontal order in queries easily destroys this tractability.

**Theorem 5.7** *There exist a fully-specified schema mapping $M \in \text{SM}^{nr}(\downarrow)$, and queries $Q_1 \in \mathbf{CTQ}(\downarrow, \rightarrow, =)$ and $Q_2 \in \mathbf{CTQ}(\downarrow, \rightarrow^*, =)$ such that both $\textsc{certain}_M(Q_1)$ and $\textsc{certain}_M(Q_2)$ are coNP-complete. In fact, both $Q_1$ and $Q_2$ do not use even _.*

*Proof.* The coNP upperbound follows from 5.3.

For the lowerbound, we prove the first claim of the theorem, and the proof for the second can be obtained by replacing $\rightarrow^*$ with $\rightarrow$ in the following proof.

We describe an XML schema mapping $M$ and a query $Q \in \mathbf{CTQ}(\downarrow, \rightarrow, =)$ such that 3SAT is reducible to the complement of $\textsc{certain}_M(Q)$. More specifically, there exist an XML schema mapping $M$ and a query $Q$ for which the following holds:

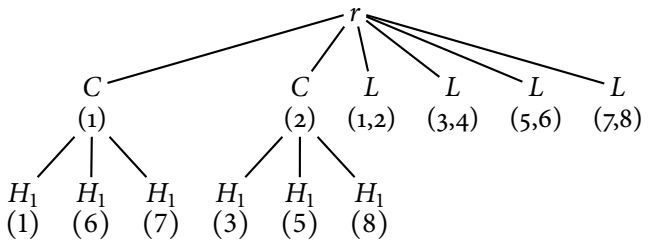
$$\text{certain}_M(Q, T_\varphi) \text{ is false iff } \varphi \text{ is satisfiable.}, \tag{5.1}$$

where $T_\varphi$ is a tree encoding of a formula.

The idea of the reduction is the following: we transform a 3CNF formula $\varphi$ into a source tree $T_\varphi$. The mapping is defined so that a solution of $T_\varphi$ 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 $\varphi$.

Suppose we are given a 3-CNF formula $\varphi = \wedge_{i=1}^{n} \vee_{j=1}^{3} c_{ij}$, where $c_{ij}$ is a literal. We construct a source tree $T_\varphi$ by the following encoding, which we explain with 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 follows:




Each $L$ node has two attribute values encoding a variable and its negation, respectively. For example $L(1, 2)$ indicates that $x_1$ is encoded by the data value '1' and $\neg x_1$ by '2'. In general, for each variable we have an $L$ node encoding it and its negation with distinct values. Also for each clause in the formula we have $C$ node that has three children labeled $H_1, H_2, H_3$, respectively. The data value held at $C$ is an identifier for it[4], and $H_i$ holds the data value encoding $i$-th 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 $H_1$ under the middle $C$ node is '6'.

Formally the source DTD $D_s$ is

$$r \to C^* L^* \qquad C\colon @a_1 \qquad L\colon @a_2, @a_3$$
$$C \to H_1 H_2 H_3 \qquad H_1, H_2, H_3\colon @b_1.$$

The target DTD $D_t$ is quite similar to the source DTD above:

$$r \to C^* L^* \qquad C\colon @a_1 \qquad L\colon @a_2, @a_3$$
$$C \to H^* \qquad H\colon @b_1$$

The last component of schema mapping are the stds $\Sigma$. The idea for the mapping is that, given $T_\varphi$, we essentially copy it 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, we choose a literal having more than two younger-siblings. Since each $C$ node has three $H$ nodes below, clearly at least one literal is chosen for each clause.

[4]This feature is left unused in this proof, but will be useful in later ones.

$$r[C[H_1(x), H_2(y), H_3(z)]] \to r[C[H(x), H(y), H(z)]]$$
$$r[L(x, y)] \to r[L(x, y)]$$

Finally we define the query. It is true if a variable and its negations are contained among the chosen literals.

$$\exists x \exists y (r[L(x, y), C[H(x) \to H \to H], C[H(y) \to H \to H]]) \tag{5.2}$$

Formally, the correctness of reduction can be proved as follows. To be precise, we prove that $\text{certain}_M(Q, T_\varphi)$ is false if and only if a 3-CNF formula $\varphi$ is satisfiable.

($\Rightarrow$) Suppose $\text{certain}_M(Q, T_\varphi)$ is false. Then there exists a tree $T'$ that is a solution for the tree encoding $\varphi$ and falsifies the query. We extract an assignment $v$ from $T'$ as follows. For each fragment matching $r[C[H(x) \to H \to H]]$, $v$ assigns true to the variable encoded by $x$. This assignment is consistent since the query is false over $T'$. Finally the assignment satisfies $\varphi$. The dependency requires that there should be at least three $H$'s (holding values appearing in the source) below a $C$ node. Hence, for each $C$ node, there is at least one $H$ node beneath it that has two younger siblings, so that the corresponding literal is true by the assignment.

($\Leftarrow$) Suppose that $\varphi$ is satisfiable. Assume $v$ is a satisfying assignment. Then we construct a solution of the mapping for $T_\varphi$ that falsies the query in the following way. Basically what we do is to change the order of $H$'s under each $C$ node so that, for each clause, a literal assigned true has at least two younger siblings. More specifically, for each tree fragment of the form $r[C[H_1(d_1), H_2(d_2), H_3(d_3)]]$, the corresponding clause has at least one literal that is assigned true by $v$. We choose the data encoding one such literal [5]. For example, suppose it is $d_2$. Then we make the tree fragment $r[C[H(d_2) \to H(d_1) \to H(d_3)]]$. After we process 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. □

**Even More Fully Specified Mappings**

We have seen that even if we stick to basic downward mappings, we cannot extend the query language. But, can't we find a more suitable class of mappings?

[5] We choose the one corresponding to a literal with the smallest index if there are more than one literals $v$ assigns true.

Observe that using commas in the stds is essential in the proof above, since it induces nondeterminism in the order of nodes. What if we disallow using commas? In other words, what if we always have to specify the order of children? We will call such mappings $\rightarrow^*$-fully specified. An even stronger restriction is that we always have to specify the exact order of children with $\rightarrow$. We call this mappings $\rightarrow$ fully specified. For example, an std using $a[b, c]$ is neither $\rightarrow$- nor $\rightarrow^*$-fully specified; $a[b \rightarrow^* c]$ is $\rightarrow^*$-fully specified, but not $\rightarrow$-fully specified.

Below we consider classes of $\rightarrow$- or $\rightarrow^*$- fully specified mappings and show that these cases are still hard. We deal with the former class first.

**Theorem 5.8** *There exist a $\rightarrow$-fully-specified schema mapping $M \in \mathrm{SM}^{nr}(\downarrow, \rightarrow)$ and a query $Q \in \mathbf{CTQ}(\downarrow, \rightarrow, =)$ such that* CERTAIN$(Q)$ *is coNP-complete.*

*Proof.* As in the previous proof, we will provide an XML schema mapping $M = \langle D_s, D_t, \Sigma \rangle$ and a query $Q$ such that we can reduce 3SAT to the complement of $\mathrm{certain}_M(Q, T_\varphi)$, where $T_\varphi$ is the same encoding of formulae as in the previous proof.

The idea of the reduction is the following: we transform a 3CNF formula $\varphi$ into a source tree $T_\varphi$. The mapping is defined so that a solution of $T_\varphi$ 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 $\varphi$. The difference from the previous proof is how we "choose" literals.

The source DTD $D_s$ is:

$$\begin{array}{lll} r \rightarrow C^* L^* & C\colon @a_1 & L @a_2, @a_3 \\ C \rightarrow H_1 H_2 H_3 & H_1, H_2, H_3\colon @b_1, & \end{array}$$

The target DTD is again almost the same as the source, except that it has $G_i$'s. With these extra element types, we 'choose' a literal from each clause. Intuitively we select $H_i$'s that are 'two step to the right of $G_1$'.

$$\begin{array}{ll} r \rightarrow C^* L^* & L\colon @a_1, @a_2 \\ C \rightarrow G_1 G_2 ? G_3 ? H_1 H_2 H_3 & H_1, H_2, H_3\colon @b_1. \end{array}$$

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

$$r[C[H_1(x) \to H_2(y) \to H_3(z)]] \to r[C[H_1(x) \to H_2(y) \to H_3(z)]]$$
$$r[L(x,y)] \to r[L(x,y)]$$

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

$$\exists x \exists y (r[L(x,y), C[G_1 \to \_ \to \_ \to \_(x)], C[G_1 \to \_ \to \_ \to \_(x)])$$

In order to formally prove the correctness of the reduction, we have to prove that certain$(Q, T_\varphi)$ is false iff $\varphi$ is satisfiable.

($\Rightarrow$) To prove the left-to-right direction, suppose certain$(Q, T_\varphi)$ 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 a clause, find the node below it that is two steps away from $G_1$ (whose element type must be one of $H_i$). Say, it is $H_i(n)$, by the dependency it encodes a literal. If $n$ 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 $T'$, 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 $T_\varphi$ base on $v$. First we copy every $L$ node. Next for each fragment of the form $r[C[H_1(d_1) \to H_2(d_2) \to H_3(d_3)]]$, we will copy it and put $G_1$ to the left of $H_1$ with possibly putting $G_2$ and $G_3$ in-between. If $v$ assigns true to the first literal in the corresponding clause, then we put both $G_2$ and $G_3$ between $G_1$ and $H_1$ (to make $H_1$ "two steps to the left of $G_1$); otherwise if it is the second literal that is assigned true by $v$, then we put $G_2$ alone; otherwise we put neither $G_2$ nor $G_3$. Since a truth assignment doesn't assign true to both a variable and its negation, it will not happen that the values paired in $L$, i.e. those encoding a variable and its negation, are both two steps away from $G_1$. Thus the query is false in the constructed tree, as desired. □

Next we deal with $\to^*$-fully-specifed mappings. Despite the similarity of the proof, it should be noted that it is not a mere translation of the preceding proof.

**Theorem 5.9** *There exist a $\to^*$-fully-specified schema mapping $M \in \mathrm{SM}^{nr}(\downarrow, \to^*)$ and a query $Q \in \mathbf{CTQ}(\downarrow, \to^*, =)$ such that* CERTAIN$(Q)$ *is coNP-complete.*

*Proof.* As before we provide $M = \langle D_s, D_t, \Sigma \rangle$ and a query $Q$ to which 3SAT is reducible to $\textsc{certain}_M(Q)$.

The idea of the reduction is the following: we transform a 3CNF formula $\varphi$ into a source tree $T_\varphi$. The mapping is defined so that a solution of $T_\varphi$ 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 $\varphi$. The difference from the previous proofs is how we "choose" literals.

$D_s$ is quite similar as before:

$$\begin{array}{lll} r \to C^* L^* & C\colon @a_1 & L\colon @a_2, @a_3 \\ C \to H^* & H\colon @b_1 & \end{array}$$

Note that the subscript in $H$ is dropped. We encode a given 3CNF formula $\varphi$ as $T_\varphi$, by simply dropping the subscript in the previous encoding .

The target DTD is the following very simple one.

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

The constraint is the following: it "flattens" the structure using multi-attributes. Each $A$ node contains two attributes, the first of which indicates a clause and the second of which encodes a literal. Formally the stds are:

$$\begin{aligned} r/C(x)/H(y) &\to r/a(x, y) \\ r/L(x, y) &\to r/L(x, y) \end{aligned}$$

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

Lastly we define the query $Q$ as follows. As in the previous reductions, the query is true when the set of selected literals contains a variable and its negation.

$$\begin{aligned} \exists x y v w u_1 u_2 u_3 u_4 (r[L(x, y), r[a(v, x) \to^* a(v, u_1) \to^* a(v, u_2)], \\ a(w, y) \to^* a(w, u_3) \to^* a(w, u_4)]) \end{aligned}$$

To prove formally the reduction, we will show that, given a formula $\varphi$, $\text{certain}_M(Q, T_\varphi)$ is false iff $\varphi$ is satisfiable.

($\Rightarrow$) Suppose $\text{certain}(Q, T_\varphi)$ is false. We have a solution $T'$ for $T_\varphi$ 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 $L$ 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 $\varphi$. For each clause $C(d)$ in the source has three children (labeled $H$), 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 $\varphi$ is satisfiable, with a satisfying assignment $v$. We need to construct a solution of the mapping for the source tree $T_\varphi$. First we copy all the $L$ nodes from the source. For each fragment of the form $r[C(d)[H_1(d_1), H_2(d_2), H_3(d_3)]]$, 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 fist 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 fragments encoding a clause, we obtain a tree that conforms to the target DTD and is a solution for $T_\varphi$. 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.

Also this gives another proof of coNP-hardness of $\mathbf{CTQ}(\downarrow, \to^*, =)$ (Theorem 5.7). □

### Threshold DTDs

Our last attempt stems from the following example. Suppose we have a source-to-target dependency $\varphi(x, y) \to r[a(x) \to b(y)]$ with the target DTD being $r \to a^* b^*$. Once a source tree has more than one pair satisfying $\varphi(x, y)$, it does not have solution since $a(v_1) \to b(v_2)$ and $a(v_3) \to b(v_4)$ can never coexist (assuming $v_1 \neq v_3$ or $v_2 \neq v_4$). Arguably this is rather anomalous and it is more natural, at least for nested relational DTDs, to allow arbitrary permutations of letters. This is precisely what is done in [2]. A *threshold DTD* assigns each element type $\ell$ its *multiplicity atom* $\mu(\ell)$, an expression of the form $\hat{\ell}_1 \cdots \hat{\ell}_m$, where $\hat{\ell}$ is one of $\ell_i$, $\ell_i^*$, $\ell^+$, and $\ell? = \ell_i$, which limits the number of children of

type $\ell_i$ of a node labelled $\ell$ in the obvious way, without imposing any restriction on the order of the children. We write $\mathrm{SM}^{\mathrm{th}}$ and $\mathrm{SM}^{\mathrm{th}}(\sigma)$ for classes of schema mappings using threshold DTDs.

Do such mappings admit a better algorithm for computing certain answers? Again, the answer is (almost) no. For $\mathbf{CTQ}(\downarrow, \rightarrow^*, =)$, the proof carries over. For $\mathbf{CTQ}(\downarrow, \rightarrow, =)$ the original proof breaks down, but we can show hardness for $\mathbf{UCTQ}(\downarrow, \rightarrow, =)$.

**Theorem 5.10** *There exist a $\rightarrow$-fully-specified schema mapping $M \in \mathrm{SM}(\downarrow, \rightarrow)$ using threshold DTDs and a query $Q \in \mathbf{UCTQ}(\downarrow, \rightarrow, =)$ such that $\textsc{certain}(Q)$ is coNP-complete.*

*Proof.* We will describe an XML schema mapping $M = \langle D_s, D_t, \Sigma \rangle$ and a query $Q$ such that 3SAT is reducible to $\textsc{certain}_M(Q)$. The same encoding $T_\varphi$ of a given 3CNF formula $\varphi$ is used as in the proof of 5.7.

The idea of the reduction is the following: we transform a 3CNF formula $\varphi$ into a source tree $T_\varphi$. The mapping is defined so that a solution of $T_\varphi$ 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 $\varphi$. The difference from the previous proofs is how we "choose" literals.

The source DTD $D_s$ is the familiar one:

$$\begin{array}{lll} r \rightarrow C^* L^* & C\colon a_1 & L\colon @a_1, @a_2 \\ C \rightarrow H_1 H_2 H_3 & H_1, H_2, H_3\colon @b_1 & \end{array}$$

The target DTD $D_t$ is the following. The difference from the source DTDs is that each $H_i$ has $A, B$ below. Since we are working with a threshold DTD, $A, B$ can appear in either order. The set of the selected (values encoding) literals having "$A \rightarrow B$" below.

$$\begin{array}{ll} r \rightarrow C^* L^* & L\colon @a_1, @a_2 \\ C \rightarrow H_1 H_2 H_3 & H_1, H_2, H_3\colon @b_1 \\ H_i \rightarrow AB & \end{array}$$

The stds are copying., Note that they are $\rightarrow$-fully-specified.

$$r[C[H_1(x) \rightarrow H_2(y) \rightarrow H_3(z)]] \rightarrow r[C[H_1(x) \rightarrow H_2(y) \rightarrow H_3(z)]]$$
$$r[L(x, y) \rightarrow r[L(x, y)]$$

We define the query that is true when both a variable and its negation are selected or there is a clause where no literal is selected. Formally, it is $q_1 \cup q_2$, where

$$q_1 := \cup_{i,j\in[3]} \exists x y (r[L(x,y), C[H_i(x)[B \to A]], C[H_j(y)[B \to A]]])$$
$$q_2 := r[C[H_1[B \to A] \to H_2[B \to A] \to H_3[B \to A]]]$$

To prove the correctness of reduction, we shall prove that, given a 3-CNF formula $\varphi$, $\text{certain}_M(Q, T_\varphi)$ is false iff $\varphi$ is satisfiable.

($\Rightarrow$) Suppose $\text{certain}_M(Q, T_\varphi)$ is false. Then there exists a solution of the mapping $T'$ for $T_\varphi$ that is a solution and falsifies $Q$, that is, falsifies both of $q_1$ and $q_2$. Extraction of a truth assignment is done as described earlier: A variable $x_i$ is assigned true (resp. false) if there is a node $H(d)$, where $d$ encodes $x_i$, that has $A \to B$ (resp. $B \to A$) below it. Note that some $H(d)$ has $A \to B$ in one place and $B \to A$ in another. In this case the variable encode by $d$ is assigned true. Recall that $T'$ falsifies both $q_1$ and $q_2$. Because of falsifying $q_1$, no variable and its negation are assigned true, whence the assignment is consistent. Also due to falsifying $q_2$, each $C$ node has at least one $H$ having $L \to R$, meaning that each clause has at least one literal assigned true.

($\Leftarrow$) For the other direction, assume we have a satisfying assignment $\nu$ for $\varphi$. We can construct the solution of the mapping for $T_\varphi$ by putting $L \to R$ under $H$'s holding the data encoding literals assigned true and $R \to L$ under those having data encoding false literals. We need to show that $Q = q_1 \cup q_2$ is false over this constructed tree. Since any assignment does not assign true to both a variable and its negation, $q_1$ is falsified. Again by the definition of satisfying assignment, it assigns true to at least one literal in each clause, so that $q_2$ is falsified as well. □

## 5.4 Digression: on querying incomplete DOM-trees

Recall that the coNP upperbound (Proposition 5.3) for query answering was obtained by reducing the problem to query answering over incomplete XML trees. In this section, we show that the above proof leads to a new result in querying answering over incomplete trees, which draws a line between when node ids do make difference and when they don't.

In [16], one of the key factors considered to obtain tractable cases was the notion of *node ids*. Having node ids in patterns essentially corresponds to "injective semantics", as called in [39], of tree patterns.

The difference is whether we may 'collapse' nodes or not. As a very simple example,

consider a tree pattern $\pi = r[a, a]$. The pattern $\pi$ as an incomplete tree without node ids is satisfied by a tree $r[a]$ while $\pi^i$ with node ids is not satisfied by the same tree since a tree must have (at least) two $a$ nodes to satisfy $\pi^i$.

To be more precise, the following change is made to the semantics of "," (comma):

$(T, s) \vDash \ell(\bar{a})[\lambda_1, \lambda_2]$ iff $s$ satisfies $\ell(\bar{a})$ and has *distinct* children $s_1$ and $s_2$ witnessing $\lambda_1$ and $\lambda_2$, respectively.

Formally, an incomplete tree with node ids $\pi^i$, or an *incomplete DOM-tree*, is a tree pattern defined by the same grammar as incomplete trees, with the change above to its semantics

We restate the query answering problem over incomplete trees. Suppose $Q$ is a union of conjunctive tree queries.

| | |
|---|---|
| PROBLEM: | INC-CERTAIN$(Q)$ |
| INPUT: | an incomplete tree $\pi$, tuple $\bar{s}$ |
| QUESTION: | $\bar{s} \in \text{certain}(Q, \pi)$? |

We refer as INC-CERTAIN$^i(Q)$ to the query answering problem where inputs are incomplete DOM-trees.

The following result (Theorem 6.7 [16]) states that for a class of shallow queries node ids make difference. The depth of queries is defined in a natural way, starting from 1 at the root. (Thus trees of depth 2 means essentially strings.)

**Theorem 5.11**

- *For any variable-free query $Q$ in* $\mathbf{CTQ}(\downarrow, \rightarrow)$ *of depth 2,* INC-CERTAIN$^i(Q)$ *is solvable in PTIME.*
- *There exists a variable-free query $Q$ in* $\mathbf{CTQ}(\downarrow, \rightarrow)$ *of depth 2, for which* INC-CERTAIN$(Q)$ *is coNP-complete.*

In the rest of this section we show that the theorem mentioned above cannot go much further: increasing depth by 1 and allowing use of variables leads to intractability even for incomplete DOM-trees.

**Theorem 5.12** *There is a query $Q$ in* $\mathbf{CTQ}(\downarrow, \rightarrow)$ *of depth 3 such that* INC-CERTAIN$^i(Q)$ *is coNP-complete.*

*Proof.* The upperbound follows from Proposition 5.3.

For the lowerbound, we reduce 3SAT to the complement of our problem. Specifically, we describe a (fixed Boolean) query $Q$ such that a given 3SAT formula $\varphi$ is satisfiable iff certain$(Q, \pi^i_\varphi)$ is false, where $\pi^i_\varphi$ is a encoding of $\varphi$ as an incomplete DOM-tree.

We encode a formula in the same way as previous proofs (see the proof of Theorem 5.7). Note that we actually have an incomplete tree that is nearly complete – the only information missing is how the children under each $C$-node is ordered. This means that any realisation of the above incomplete tree materialises one of the possible orders of $H$ nodes under each $C$ node. Also since it contains only values, the semantics of comma does not affect the proof.

Next we define the query.

$$\exists x \exists y (r[L(x, y)] \wedge r[C[H(x) \to \_ \to \_]] \wedge r[C[H(y) \to \_ \to \_]]) \tag{5.3}$$

The intuition behind the reduction is the following. As just mentioned, a realisation of $\pi^i_\varphi$ reorders the children of each $C$ node. Since each $C$ node has three $H$ nodes beneath it, at least one $H$ node has two nodes to the right. We then choose literals (as being true) in each clause by picking those "having two nodes on the right". The above query checks if a variable and its negation are among the selected literals, so that the falsity of the query ensures the consistency of the assignment.

Formally, the correctness of reduction can be proved as follows. To be concrete, we prove that certain$(Q, \pi^i_\varphi)$ is false if and only if a 3-CNF formula $\varphi$ is satisfiable.

($\Rightarrow$) Suppose certain$_M(Q, \pi^i_\varphi)$ is false. Then there exists a tree $T$ that is a realisation of $\pi^i_\varphi$ and falsifies the query. We extract an assignment $\nu$ from $T'$ as follows. For each fragment matching $r[C[H(x) \to \_ \to \_]]$, $\nu$ assigns true to the variable encoded by $x$. This assignment is consistent since the query is false over $T$. Finally the assignment satisfies $\varphi$. The dependency requires that there should be at least three $H$'s (holding values appearing in the source) below each $C$ node. Hence, for each $C$ node, there is at least one $H$ node beneath it that has two younger siblings, so that the corresponding literal is true by the assignment.

($\Leftarrow$) Suppose that $\varphi$ is satisfiable with an assignment $\nu$. Then we construct a solution of the mapping for $\pi^i_\varphi$ that falsies the query in the following way. Basically what we do is to change the order of $H$'s under each $C$ node so that, for each clause, a literal assigned true has at least two younger siblings. More specifically, for each tree fragment of the form $r[C[H(d_1), H(d_2), H(d_3)]]$ in $\pi^i_\varphi$, the corresponding clause has at least one literal that is assigned true by $\nu$. When it is $d_1$, we put $r[C[H(d_1) \to H(d_2) \to H(d_3)]]$; otherwise

when it is $d_2$, we put $r[C[H(d_2) \to H(d_1) \to H(d_3)]]$; otherwise when it is $d_3$, we put $r[C[H(d_3) \to H(d_1) \to H(d_2)]]$. The constructed tree is a realisation of $\pi^i_\varphi$ since we merely reordered the children of each $C$ node, and it falsifies the query since $\nu$ is a valid assignment. □

# Chapter 6

# Conclusion

Let us summarise what is described in this thesis and make a few observations.

We considered the problem of XML data exchange under expressive mappings that use horizontal navigation and data comparison as well as downward navigation. Naturally static analysis and query answering get harder with these features.

The static analysis problem we investigated most thoroughly was the consistency problem. In contrast to what was known about simple mappings, the presence of data comparison makes the problem undecidable. We also observed that non-monotone nature of horizontal ordering leads to intractability where simple mappings admit tractable consistency check. While these gives an overall picture of the problem, there are some technical questions left open. Most of the problem are related to the following problem: given a DTD $D$ and two sets of patterns $P_+$ and $P_-$, can we find a tree $T \models D$ that matches all the patterns in $P_+$ and none in $P_-$? We know that the problem is in EXPTIME and NP-hard; knowing its exact complexity will help us close the complexity gaps. More precisely, what we don't know is if this problem is easier when we have restricted patterns, e.g. the class with only child navigation. Absolute consistency was the other extreme of consistency of which we have decidability for downward fragment without data comparison. With extended features, decidability is left open.

We looked at the most commonly studied operation of composition on mappings. Our finding is that with extended features in tree patterns and non-nested relational DTDs examples showing non-closure can easily be found. The main difficulty is that some sort of disjunction is required. We stripped down these features and found a class of mappings that is closed under composition, with the remedy of Skolem functions.

We have studied query answering for XML data exchange, adopting the certain an-

swers semantics. Earlier work on XML data exchange with a less expressive language showed that query answering is tractable for simple mappings, and coNP-complete for more complex ones. Our main finding is that we can naturally extend the simple mappings with horizontal navigation and inequality, retaining tractability, provided we stick to the basic query language. On the other hand, tractability is lost when extended query languages are considered, even for very simple mappings. This shows that one must restrict the usage of sibling order and inequality to the mappings.

## 6.1 Future Work

There are several directions to extend the current work.

One of the basic questions left untouched is constructing target instances: Given a source tree, how to compute a target tree? This is important in data integration and exchange tasks, but so far good algorithms are lacking, even for very simple mappings already introduced in [12]. 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 [13, 43, 44, 49] or merge [22]. Also, we would like to extend structural results of [36] from relational to XML mappings (in fact we already have results for instance-level justification of classes of mappings). As already mentioned, the decidability of absolute consistency for extended mappings is left open.

As for query answering, so far we have concentrated on data complexity of the problem. We would like to look at combined complexity in future in order to have a better understanding of query answering in XML data exchange.

Concerning another direction, 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 [11].

Yet anther dimension that hasn't been investigated is the distinction between open world assumption (OWA) and closed world assumption (CWA). Here, we have worked exclusively under OWA. In relational case, an anomaly is observed when the query involves negation [10, 45]. As a remedy to such unintuitive behaviour, the notion of solutions under CWA was proposed in [70], further extended in [57, 72]. This direction is hardly explored for XML: it is not even clear how to define the notion of CWA in the XML context.

At more abstract level, we would like to conduct a clearer and more unified investiga-

tion on the difference between ordered and unordered trees. Traditionally they are treated as more or less the same. But from our study it seems that, at least with the presence of data, horizontal ordering in trees has some impact. Apart from our results, the decidability of the vertical fragment of XPath (with data comparison) is still an open problem as stated in [52] while with horizontal ordering it is undecidable as shown in its follow-up [53]. Also it is interesting to see if there is a case where ordering introduces much complexity beyond the area of XML research, say in programming language theory.

# Bibliography


[1] Abiteboul, S., R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[2] Abiteboul, S., L. Segoufin, and V. Vianu, Representing and querying XML with incomplete information. *ACM Transactions on Database Systems*, 31(1), 208–254, 2006.

[3] Afrati, F. N. and P. G. Kolaitis, Answering aggregate queries in data exchange, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, pp. 129–138, 2008.

[4] Alexe, B., L. Chiticariu, and W. C. Tan, Spider: a schema mapping debugger, in *International Conference on Very Large Data Bases (VLDB)*, pp. 1179–1182, 2006.

[5] Alon, N., T. Milo, F. Neven, and V. Vianu, XML with data values: typechecking revisited. *Journal of Computer and System Sciences*, 66(4), 688–727, 2003.

[6] Amano, S., C. David, L. Libkin, and F. Muralk, On the tradeoff between mapping and querying power in XML data exchange, in *International Conference on Database Theory (ICDT)*, 2010.

[7] Amano, S., L. Libkin, and F. Murlak, XML schema mapping, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2009.

[8] Amer-Yahia, S., S. Cho, L. V. S. Lakshmanan, and D. Srivastava, Tree pattern query minimization. *VLDB Journal*, 11(4), 315–331, 2002.

[9] Amer-Yahia, S. and Y. Kotidis, Web-services architecture for efficient XML data exchange, in *International Conference on Data Engineering*, volume 523-534, 2004.

[10] Arenas, M., P. Barceló, R. Fagin, and L. Libkin, Locally consistent transformations and query answering in data exchange, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2004.

[11] Arenas, M., P. Barceló, and J. Reutter, Query languages for data exchange: Beyond unions of conjuctive queries, in *International Conference on Database Theory (ICDT)*, 2009.

[12] Arenas, M. and L. Libkin, XML data exchange: consistency and query answering. *Journal of the ACM*, 55(2), 7:1–72, 2008.

[13] Arenas, M., J. Pérez, and C. Riveros, The recovery of a schema mapping: bringing exchanged data back, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2008.

[14] Barbosa, D., J. Freire, and A. O. Mendelzon, Designing information-preserving mapping schemes for xml, in *International Conference on Very Large Data Bases (VLDB)*, pp. 109–120, 2005.

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

[16] Barceló, P., L. Libkin, A. Poggi, and C. Sirangelo, XML with incomplete information: models, properties, and query answering, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2009.

[17] Beeri, C. and M. Y. Vardi, A proof procedure for data dependencies. *Journal of the ACM*, 31(4), 718–741, 1984, ISSN 0004-5411.

[18] Benedikt, M., W. Fan, and F. Geerts, XPath satisfiability in the presence of DTDs, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2005.

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

[20] Bernstein, P. A., Applying model management to classical meta data problems, in *Conference on Innovative Data Systems Research (CIDR)*, 2003.

[21] Bernstein, P. A., T. J. Green, S. Melnik, and A. Nash, Implementing mapping composition. *VLDB Journal*, 17(2), 333–353, 2008.

[22] Bernstein, P. A. and S. Melnik, Model management 2.0: manipulating richer mappings, in *SIGMOD international conference on Management of data (SIGMOD)*, 2007.

[23] Bex, G. J., F. Neven, and J. van den Bussche, DTDs versus XML schema, in *WebDB*, pp. 79–84, 2004.

[24] Björklund, H. and M. Bołańczyk, Bounded depth data trees, in *International Colloquium on Automata, Languages and Programming (ICALP)*, 2007.

[25] Björklund, H., W. Martens, and T. Schwentick, Optimizing conjunctive queries over trees using schema information, in *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pp. 132–143, 2008.

[26] Björklund, H. and T. Schwentick, On notions of regularity for data languages, in *Fundamental of Computation Theory*, 2007.

[27] Bojańczyk, M., A. Muscholl, T. Schwentick, and L. Segoufin, Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3), 2009.

[28] Bojańczyk, M., A. Muscholl, T. Schwentick, L. Segoufin, and C. David, Two-variable logic on words with data, in *IEEE Symposium on Logic in Computer Science (LICS)*, pp. 7–16, 2006.

[29] Bojańczyk, M. and P. Parys, Xpath evaluation in linear time, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, pp. 241–250, 2008.

[30] Börger, E., E. Grädel, and Y. Gurevich, *Classical Decision Problem*, Springer-Verlag, 1997.

[31] Bouyer, P., A. Petit, and D. Thérien, An algebraic approach to data languages and timed languages. *Information and Computation*, 182, 137–162, 2003.

[32] Brüggemann-Klein, A., M. Murata, and D. Wood, Regular tree and regular hedge languages over unranked alphabets, Technical Report HKUST-TCSC-2001-05, HKUST Theoretical Computer Science Center, 2001.

[33] Büchi, J. R., Weak second order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6, 66–92, 1960.

[34] Büchi, J. R., On a decision method in restricted second order logic, in *Proceedings of International Congress on Logic, Methodology and Philosophy of Science*, pp. 1–11, Stanford University Press, 1962.

[35] ten Cate, B., L. Chiticariu, P. G. Kolaitis, and W. C. Tan, Laconic schema mappings: Computing the core with sql queries. *The Proceedings of the VLDB Endowment (PVLDB)*, 2(1), 1006–1017, 2009.

[36] ten Cate, B. and P. G. Kolaitis, Structural characterizations of schema-mapping languages, in *International Conference on Database Theory (ICDT)*, pp. 63–72, 2009.

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

[38] Comon, H., M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi, Tree Automata Techniques and Applications, Available on: `http://tata.gforge.inria.fr/`, 2007, release 12 October 2007.

[39] David, C., Complexity of data tree patterns over XML documents, in *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2008.

[40] Demri, S. and R. Lazic, Ltl with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009.

[41] Deutsch, A., A. Nash, and J. B. Remmel, The chase revisited, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, pp. 149–158, 2008.

[42] Doner, J., Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4(5), 406–451, 1970.

[43] Fagin, R., Inverting schema mappings. *ACM Transactions on Database Systems*, 32(4), 2007.

[44] Fagin, R., P. G. Kolaitis, Lucian, and W.-C. Tan, Quasi-inverses of schema mappings. *ACM Transactions on Database Systems*, 33(2), 2008.

[45] Fagin, R., P. G. Kolaitis, R. J. Miller, and L. Popa, Date exchange: semantics and query answering. *Theoretical Computer Science*, 336, 89–124, 2005.

[46] Fagin, R., P. G. Kolaitis, A. Nash, and L. Popa, Towards a theory of schema-mapping optimization, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, pp. 33–42, 2008.

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

[48] Fagin, R., P. G. Kolaitis, L. Popa, and W.-C. Tan, Composing schema mappings: second-order dependencies to the rescue. *ACM Transactions on Database Systems*, 30(4), 994–1055, 2005.

[49] Fagin, R., P. G. Kolaitis, L. Popa, and W. C. Tan, Reverse data exchange: coping with nulls, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, pp. 23–32, 2009.

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

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

[52] Figueira, D., Satisfiability of downward XPath with data equality tests, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2009.

[53] Figueira, D. and L. Segoufin, Future-looking logics on data words and trees, in *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2009.

[54] Fuxman, A., P. G. Kolaitis, R. J. Miller, and W. C. Tan, Peer data exchange. *ACM Transactions on Database Systems*, 31(4), 1454–1498, 2006.

[55] Giacomo, G. D., D. Lembo, M. Lenzerini, and R. Rosati, On reconciling data exchange, data integration, and peer data management, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, pp. 133–142, 2007.

[56] Gottlob, G. and A. Nash, Efficient core computation in data exchange. *Journal of the ACM*, 55(2), 2008.

[57] Heinrich, A. and N. Schweikardt, CWA-solutions for data exchange settings with target dependencies, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2007.

[58] Hell, P. and J. Nešetřil, The core of a graph. *Discrete Mathematics*, 109(1-3), 117–126, 1992.

[59] Hidders, J., Satisfiability of XPath expressions, in *International Symposium on Database Programming Languages (DBPL)*, pp. 21–36, 2003.

[60] Hopcroft, J. E. and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[61] Imielinski, T. and W. Lipski, Incomplete information in relational databases. *Journal of the ACM*, 31, 761–791, 1984.

[62] Jurdziński, M. and R. Lazić, Alternation-free modal mu-calculus for data trees, in *IEEE Symposium on Logic in Computer Science (LICS)*, 2007.

[63] Kieroński, E., Results on the guarded fragment with equivalence or transitive relations, in *EACSL Annual Conference on Computer Science Logic (CSL)*, pp. 309–324, 2005.

[64] Kieroński, E. and M. Otto, Small substructures and decidability issues for two-variable first-order logic, in *IEEE Symposium on Logic in Computer Science (LICS)*, 2005.

[65] Kieroński, E. and L. Tendera, On finite satisfiability of the guarded fragment with equivalence or transitive guards, in *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pp. 318–332, 2007.

[66] Kolaitis, P. G., Schema mappings, data exchange, and metadata management, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2005.

[67] Kolaitis, P. G., J. Panttaja, and W.-C. Tan, The complexity of data exchange, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2006.

[68] Lenzerini, M., Data integration: A theoretical perspective, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, pp. 233–246, 2002.

[69] Lewis, H. R., Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, 21(3), 317–353, 1980.

[70] Libkin, L., Data exchange and incomplete information, in S. Vansummeren, ed., *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2006.

[71] Libkin, L., Logics for unranked trees: An overview. *Logical Methods in Computer Science*, 2(3), 2006.

[72] Libkin, L. and C. Sirangelo, Data exchange and schema mappings in open and closed worlds, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, 2008.

[73] Madhavan, J. and A. Y. Halevy, Composing mappings among data sources, in *International Conference on Very Large Data Bases (VLDB)*, pp. 572–583, 2003.

[74] Mądry, A., Data exchange: on the complexity of answering queries with inequalities. *Information Processing Letters*, 94, 253–257, 2005.

[75] Maier, D., A. O. Mendelzon, and Y. Sagiv, Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4), 455–469, 1979.

[76] Martens, W., F. Neven, T. Schwentick, and G. J. Bex, Expressiveness and complexity of xml schema. *ACM Transactions on Database Systems*, 31(3), 770–813, 2006.

[77] Melnik, S., H. Garcia-Molina, and E. Rahm, Similarity flooding: A versatile graph matching algorithm and its application to schema matching, in *International Conference on Data Engineering (ICDE)*, pp. 117–128, 2002.

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

[79] Milo, T. and S. Zohar, Using schema matching to simplify heterogeneous data translation, in *International Conference on Very Large Data Bases (VLDB)*, pp. 122–133, 1998.

[80] Minsky, M. L., *Computation: Finite and Infinite Machines*, Prentice-Hall, 1967.

[81] Murata, M., D. Lee, M. Mani, and K. Kawaguchi, Taxonomy of xml schema languages using formal language theory. *ACM Transactions on Internet Technology*, 5(4), 660–704, 2005.

[82] Nash, A., P. A. Bernstein, and S. Melnik, Composition of mappings given by embedded dependencies. *ACM Transactions on Database Systems*, 32(1), 4, 2007.

[83] Neven, F., Automata, logic, and XML, in *EACSL Annual Conference on Computer Science Logic (CSL)*, pp. 2–26, 2002.

[84] Neven, F., Automata theory for XML researchers. *SIGMOD Record*, 31, 39–46, 2002.

[85] Neven, F., T. Schwentick, and V. Vianu, Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3), 403–435, 2004.

[86] Papadimitriou, C., *Computational Complexity*, Addison-Wesley, 1994.

[87] Parys, P., Xpath evaluation in linear time with polynomial combined complexity, in *ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems (PODS)*, pp. 55–64, 2009.

[88] Popa, L., Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin, Translating web data, in *International Conference on Very Large Data Bases (VLDB)*, pp. 598–609, 2002.

[89] Schwentick, T., Automata for XML - a survery. *Journal of Computer and System Sciences*, 73(3), 289–315, 2007.

[90] Segoufin, L., Automata and logics for words and trees over an infinite alphabet, in *EACSL Annual Conference on Computer Science Logic (CSL)*, 2006.

[91] Seidl, H., Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3), 424–437, 1990.

[92] Shu, N., B. Housel, R. Taylor, S. Ghosh, and V. Lum, EXPRESS: a data extraction, processing, and restructuring system. *ACM Transactions on Database Systems*, 2, 134–174, 1977.

[93] Sipser, M., *Introduction to the Theory of Computation*, PWS Publishing Company, 1997.

[94] Tan, T., Graph reachability and pebble automata over infinite alphabets, in *IEEE Symposium on Logic in Computer Science (LICS)*, 2009.

[95] Tan, T., On pebble automata for data languages with decidable emptiness problem, in *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2009.

[96] Thatcher, J. W. and J. B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1), 57–81, 1968.

[97] Thomas, W., Automata on infinite objects, in J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pp. 133–192, The MIT Press, 1990.

[98] Thomas, W., Languages, automata, logic, in G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages*, pp. 389–455, Springer, 1997.

[99] Vardi, M. Y., The complexity of relational query languages (extended abstract), in *Annual ACM Symposium on Theory of Computing (STOC)*, pp. 137–146, 1982.

[100] Vardi, M. Y., Logic and automata: A match made in heaven, in *International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 64–65, 2003.

[101] Vianu, V., A web odyssey: from Codd to XML. *SIGMOD Record*, 32(2), 68–77, 2003.

[102] Vianu, V., XML: From practice to theory, in A. H. F. Laender, ed., *XVIII Simpósio Brasileiro de Bancos de Dados, 6-8 de Outubro, Manaus, Amazonas, Brasil, Anais/Proceedings*, SBBD, pp. 11–25, UFAM, 2003.