On the Tradeoff between Mapping and Querying Power in XML Data Exchange *

Shun'ichi Amano Claire David Leonid Libkin Filip Murlak

ABSTRACT

In XML data exchange, a schema mapping specifies rules for restructuring a source document under the target schema, and queries over the target document must be answered in a way consistent with the source information. Mapping rules and queries in this scenario are typically based on various kinds of tree patterns. Patterns with downward navigation have been studied, and tractable classes of mappings and queries have been isolated.

In this paper we extend schema mappings and queries with general tree patterns that include horizontal navigation and data-value comparisons, and study their impact on the tractability of the query answering problem. Our main results state that, in the nutshell, extending the tractable cases for downward patterns with expressive schema mappings is harmless, but adding new features to queries quickly leads to intractability even for very simple schema mapping.

1. INTRODUCTION

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 follow a specification, known as a *schema mapping*, which describes the relationship between the two schemas. Normally such specifications are given by *source-to-target dependencies*, which are often formulated as relational calculus queries of special form. Under a schema mapping, there could be many possible solutions to materialize as a target instance. The goal then is to find a solution that would make it possible to answer queries over the chosen target instance in a way that is consistent with the source data.

*Mailling address: School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, UK.
Email addresses: {s-amano,cdavid,libkin}@inf.ed.ac.uk, fmurlak@mimuw.edu.pl

ICDT 2010, March 22–25, 2010, Lausanne, Switzerland.

The problem of data exchange has been actively studied over the past few years (see, e.g., recent surveys [19, 10, 8]). Most research has focused on the relational case, where the complexity of basic problems in relational data exchange is by now quite well understood. Systems for handling relational data exchange have been developed and incorporated into commercial systems (see, e.g., [24]). Although practical systems often claim to handle non-relational data, this is usually done either through relational translations or with very simple mappings whose power is in manipulating data values rather than changing the structure [18].

A systematic study of semi-structured data exchange was initiated in [7] which proposed a simple mapping language for XML data exchange. Apart from basic static analysis questions, [7] showed how to build solutions for answering analogs of conjunctive queries, and isolated a subclass of mappings admitting a polynomial algorithm for query answering. The mappings considered by [7] were very restricted though, as they only admitted downward navigation and a limited form of equality comparisons, but disregarded horizontal navigation, and arbitrary (in)equality comparisons.

A much more expressive language for XML schema mappings was proposed recently [4]. The language added the features that were missing in [7], and it allowed one to restructure documents based on arbitrary forms of navigation and arbitrary comparisons of data values. But while [4] presented a rather complete picture of the complexity of static analysis problems, it did not address the key problem of query answering.

Our goal thus is to study query answering in such expressive XML schema mappings. Once we add new features to mappings, we can also use them in queries, so there is a natural question to what extent we can enrich mapping and query languages while retaining good bounds on the query answering problem. We show that we cannot extend both simultaneously; in fact there is a certain tradeoff between the power of the mapping language and the power of the query language. We show, informally, that:

- the basic tractable class of queries identified in [7] remains tractable under the most expressive mappings; but
- adding new features to the query languages quickly leads to intractability, even for very simple mappings that behave well with simple queries.

The plan of the paper is as follows. After presenting the main definitions in Section 2, we review what is known about query answering in simple settings, based only on downward navigation. This is done in Section 3. The key findings of [7], reviewed there, are that we need to restrict both DTDs and source-to-target constraints to have any hope of getting tractability. We thus use the restrictions of [7] (to so-called nested relational DTDs and fully specified source-to-target constraints) throughout this paper.

In Section 4 we show that throwing in all the new features does not increase the upper bound for the query answering problem – it remains in coNP. We then show that if we add all the new features to schema mappings while keeping the basic tractable language of [7], we retain tractability. This is done in Section 5.

Then, in Section 6, we consider possible extensions of query languages with the same features as the mappings: horizontal navigation and comparisons of data values. We show that any such extension immediately leads to intractability. This remains true even for very simple mappings. We explore further possibilities of restricting mappings, by strengthening the definition of fully specified constraints and relaxing constraints on the ordering of elements, and show that even then the query answering problem remains intractable once the queries are extended beyond the class considered in [7].

2. PRELIMINARIES

2.1 XML documents and DTDs

We view XML documents as unranked trees. Each node has a label indicating its *element type* and may also have *attribute values* associated with *attribute names*. We assume attribute values come from an infinite domain V, and also that attribute names are prefixed by @ so as to be distinguished from element types.

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

- the set T is an unranked tree domain, i.e., a prefix-closed subset of $\mathbb{N}^*$ such that $n \cdot i \in T$ implies $n \cdot j \in T$ for all $j < i$;
- the binary relations $\downarrow$ and $\rightarrow$ are the child relation $(n \downarrow n \cdot i)$ and the next-sibling relation $(n \cdot i \rightarrow n \cdot (i+1))$;
- the function lab is a labeling from T to Γ;
- each ρ_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, when the interpretations of $\downarrow, \rightarrow, lab$, and ρ_a are understood, we write just T to refer to an XML document.

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

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

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

A tree T *conforms to* a DTD D if its root is labeled 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 of $P_D(\ell)$.

For example consider the following DTD D_1:

$$\begin{array}{rcllr} \texttt{europe} & \rightarrow & \texttt{country}^* & & \\ \texttt{country} & \rightarrow & (\texttt{ruler})^* & \texttt{country}: \ @\texttt{name} & (1) \\ \texttt{ruler} & \rightarrow & \varepsilon & \texttt{ruler}: \ @\texttt{name} & \end{array}$$

A tree T_1 that conforms to this DTD is shown in Figure 1.

We shall use a class of nested relational DTDs [2, 4, 7, 11] that generalize nested relations. Such DTDs are common in practice (accounting for more than 50% of DTDs in one empirical study [11]) and they have been shown to reduce the complexity of many XML static analysis problems.

A DTD is *nested relational* if it is non-recursive (i.e., the graph in which we put edges between ℓ and the element types in $P_D(\ell)$ does not contain cycles) and all of its productions are of the form $\ell \rightarrow \hat{\ell}_1 \cdots \hat{\ell}_m$, where the ℓ_i's are distinct elements of Γ and each $\hat{\ell}_i$ is one of ℓ_i, or ℓ_i^*, or $\ell_i^+ = \ell_i \ell_i^*$, or $\ell_i? = \ell_i | \varepsilon$. The DTD D_1 is nested relational. On the other hand, an expression $\ell_1 \ell_1 \ell_2^*$ cannot occur in a nested relational DTD (as ℓ_1 is used twice), nor can $(\ell_1 \ell_2)^*$ and $\ell_1 | \ell_2$.

2.2 XML schema mappings

Recall that a relational schema mapping is a quadruple $\langle \mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t \rangle$, where $\mathbf{S}$ and $\mathbf{T}$ are relational schemas (called the source schema and the target schema, respectively), Σ_{st} is a set of source-to-target dependencies, and Σ_t is the set of target dependencies. The dependencies are of the form $\varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \psi(\bar{x}, \bar{z})$, where φ, ψ are conjunctions of atomic formulae. In the literature, schema mappings without target dependencies (i.e., mappings such that Σ_t is empty) have attracted special attention, e.g., in the study of compositions of schema mappings [15].

In the XML context, various abstractions of XML Schema naturally replace relational schemas. These could be DTDs, or XSDs, or other formalisms (see, e.g., [23]). Following the tradition of papers on data exchange for semi-structured data [7, 4, 25] we use DTDs as our schema formalism, although many results can be easily extended to formalisms capturing the full power of tree automata.

For source-to-target constraints, one can use conjunctive queries as the specification language; however, in the XML

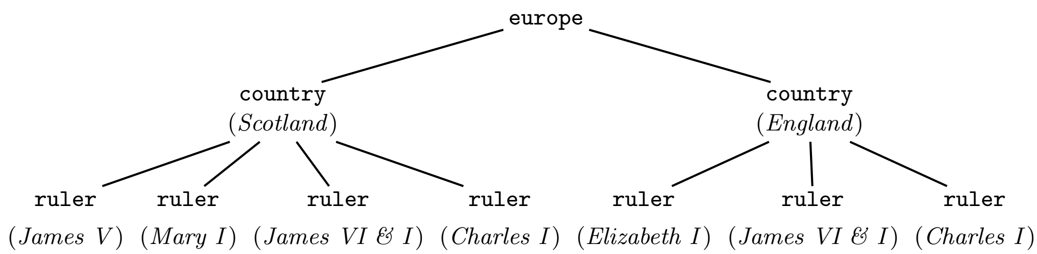


Figure 1: Tree T_1 conforming to DTD D_1

scenario this is quite cumbersome because we have two sorts of objects in an XML database: tree nodes and data values. Instead, following [7, 4], we use *tree patterns*, that capture the expressiveness of two-sorted conjunctive queries but are easier to handle syntactically.

Extended tree patterns, that handle both vertical and horizontal navigation, are given by the grammar below:

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

Here ℓ ranges over the alphabet Γ of labels and the *wildcard* symbol _ that matches every label. We write $\pi(\bar{x})$ to indicate that $\bar{x}$ is the tuple of variables used in π.

A tree T *satisfies* $\varphi(\bar{a})$ at a node s, with variables interpreted as $\bar{a}$, written $(T, s) \models \varphi$, iff the following conditions hold:

$(T,s) \models \ell(\bar{a})$	iff	s is labeled by ℓ and $\bar{a}$ is the tuple of attributes of s;
$(T,s) \models \ell(\bar{a})[\lambda_1, \lambda_2]$	iff	$(T,s) \models \ell(\bar{a})[\lambda_1]$ and $(T,s) \models \ell(\bar{a})[\lambda_2]$;
$(T,s) \models \ell(\bar{a})[\mu]$	iff	$(T,s) \models \ell(\bar{a})$ and $(T,s') \models \mu$ for some s' with $s \downarrow s'$;
$(T,s) \models \ell(\bar{a})[//\pi]$	iff	$(T,s) \models \ell(\bar{a})$ and $(T,s') \models \pi$ for some descendant s' of s;
$(T,s) \models \pi \to \mu$	iff	$(T,s) \models \pi$ and $(T,s') \models \mu$ for some s' with $s \to s'$;
$(T,s) \models \pi \to^* \mu$	iff	$(T,s) \models \pi$ and $(T,s') \models \mu$ for some s' with $s \to^* s'$.

We write $T \models \varphi$ for $(T, \varepsilon) \models \varphi$.

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

Following [4], we define a *source-to-target dependency* (*std*) as 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 π, π' are tree patterns, and $\alpha_{=,\neq}$ and $\alpha'_{=,\neq}$ are sets of equalities and inequalities among variables. Since we can always state equalities explicitly, we can assume without loss of generality that no variable appears more than once in π and π'.

A pair of trees $\langle T, T' \rangle$ *satisfies an std* of the form above if for all tuples $\bar{a}, \bar{b}$ so that $T \models \pi(\bar{a}, \bar{b})$ and $\alpha(\bar{a}, \bar{b})$ holds, there exists a tuple $\bar{c}$ such that $T \models \pi'(\bar{a}, \bar{c})$ 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,
- Σ 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 Σ (written as $\langle T, T' \rangle \models \Sigma$).

For example, let D_s be the DTD D_1 defined in (1), and let D_t be the following DTD:

$$\begin{array}{rcll} \texttt{rulers} & \to & \texttt{ruler}^* & \\ \texttt{ruler} & \to & \texttt{successor} & \texttt{ruler} : @\texttt{name} \\ \texttt{successor} & \to & \varepsilon & \texttt{successor} : @\texttt{name} \end{array}$$

Assuming the rulers are stored in the chronological order, a natural schema mapping M might be defined with the following std:

$$\texttt{europe}[\texttt{ruler}(x) \to \texttt{ruler}(y)] \;\longrightarrow\; \texttt{rulers}/\texttt{ruler}(x)/\texttt{successor}(y)\,,$$

where we use the standard XML abbreviation $\ell(\bar{x})/\pi$ for $\ell(\bar{x})[\pi]$.

A natural solution for the tree T_1 from Figure 1 is a tree T_2 shown in Figure 2. Notice that the solution is not unique. Every tree obtained from T_2 by adding new children with arbitrary data values, or by permuting the existing children, is also a solution for T_1. For instance, a solution T_3 shown in Figure 3 is as good a solution for T_1 as any.

Mappings based on downward navigation The language of tree patterns used in [7] only referred to downward navigation, i.e., it did not use sequences and was given by

$$\begin{array}{lcl} \pi & := & \ell(\bar{x})[\lambda] \\ \lambda & := & \varepsilon \mid \pi \mid //\pi \mid \lambda, \lambda \end{array}$$

The stds from that paper did not use inequalities and only allowed equalities explicitly incorporated into the patterns, i.e., they were of the form $\pi(\bar{x}, \bar{y}) \to \pi'(\bar{x}, \bar{z})$, with the same semantics as we use here.

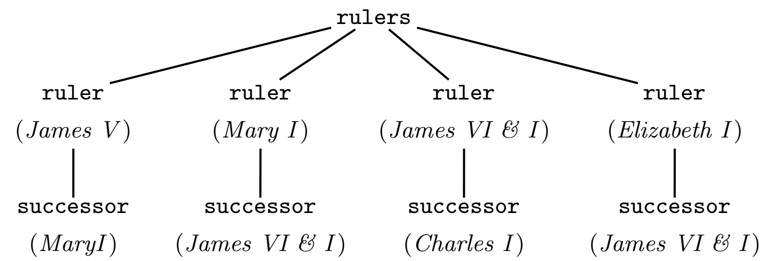


Figure 2: T_2: a solution for T_1

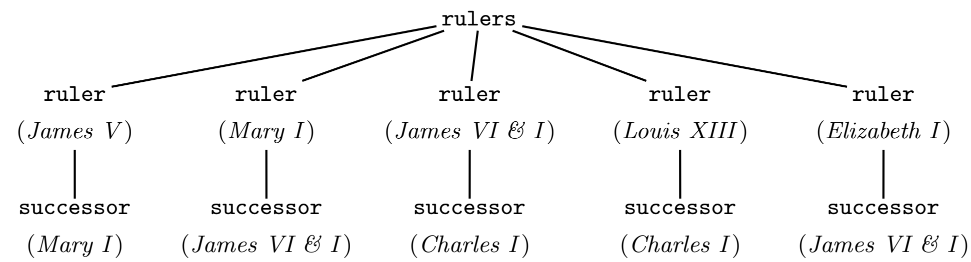


Figure 3: T_3: another possible solution for T_1

Classes of mappings In general, mappings can use vertical and horizontal navigation as well as data comparisons. Restricted classes are obtained by limiting downward navigation (for example, disallowing descendant), horizontal navigation (disallowing $\to$ or $\to^*$ in sequences), and data value comparisons (disallowing equalities or inequalities).

More precisely, we say that $\downarrow^*$, $\to$, and $\to^*$ are used in a mapping if some pattern uses the $//\pi$ construction in 'sets' in (2), or the $\pi \to \mu$ or $\pi \to^* \mu$ construction in 'sequences' in (2). A mapping uses equality if either equalities are present in formulae $\alpha_=$, or if some variables are repeated in patterns (which forces two data values to be equal). And finally, a mapping uses inequality if inequalities are present in formulae $\alpha_{\neq}$.

Thus, for a subset $\sigma \subseteq \{\downarrow, \downarrow^*, \to, \to^*, =, \neq\}$, we write $\mathbf{SM}(\sigma)$ to denote the class of schema mappings in which stds use only the operations from σ. For example, the mapping described above is in $\mathbf{SM}(\downarrow, \to)$. The class (3) of XML schema mappings originally considered in [7] is $\mathbf{SM}(\downarrow, \downarrow^*, =)$ in this notation.

For reasons to be explained shortly, we work extensively with *nested relational schema mappings*, i.e., schema mappings whose target DTDs are nested relational. By $\mathbf{SM}^{\mathrm{nr}}(\sigma)$ we denote the class of nested relational schema mappings in $\mathbf{SM}(\sigma)$. Our example of a mapping above is from the class $\mathbf{SM}^{\mathrm{nr}}(\downarrow, \to)$.

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

2.3 Query language

We follow the relational case [8, 14, 19] and study query answering for conjunctive queries and their unions. Conjunctive queries over trees are normally represented with tree patterns [16, 12, 13]. Thus, for querying XML documents we use the same language as for the dependencies: tree patterns augmented with equalities as well as inequalities, to capture the analog of relational conjunctive queries with inequalities. And, of course, we allow projection.

That is, a query is an expression of the form

$$\exists \bar{x}\ (\pi, \alpha_{=,\neq}),$$

where π is a tree pattern and $\alpha_{=,\neq}$ is a set of equalities and inequalities. The semantics is defined in the standard way. The output of the query is the set of those valuations of free variables that make the query hold true. This class of queries is denoted by **CTQ** (conjunctive tree queries). Note that **CTQ** is indeed closed under conjunctions, due to the semantics of λ, λ' in patterns.

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

Let us get back to our running example. A query one might ask over the target database is to list the rulers who were successors to more that one ruler. This would be expressed by the following conjunctive query `MultiSucc`:

$$\exists x\ \exists y\ \left(\begin{array}{ll} \texttt{rulers}[& \texttt{ruler}(x)/\texttt{successor}(z), \\ & \texttt{ruler}(y)/\texttt{successor}(z)\ ], \\ x \neq y & \end{array} \right)$$

Note that this query uses both inequality and equality, since the variable z is used twice in the tree pattern. Hence, this query is in $\mathbf{CTQ}(\downarrow, =, \neq)$.

Coming back to our example, on the tree T_2 from Figure 2 the query MultiSucc would return {*"James VI & I"*}, and on the tree T_3 from Figure 3 the answer would be {*"James VI & I"*, *"Charles I"*}.

3. DATA EXCHANGE IN SIMPLE SETTINGS

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 [14, 7], we adapt the *certain answers semantics*, i.e., we return the tuples which would be returned for every possible solution:

$$\mathit{certain}_M(Q,T) = \bigcap \left\{ Q(T') \;\middle|\; \begin{array}{l} T' \text{ is a solution for } T \\ \text{under } M \end{array} \right\}.$$

The subscript M is omitted when it is clear from the context.

Note that our queries output sets of tuples rather than trees, so we can define certain answers by taking the intersection of the answers over all solutions.

In our running example,

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

Note that when Q is a Boolean query, $\mathit{certain}_M(Q,T)$ 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 decision problem.

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

We now recall what is already known about simple settings based on downward navigation [7], i.e., mappings from $\mathbf{SM}(\downarrow,\downarrow^*,=)$ and queries from $\mathbf{UCTQ}(\downarrow,\downarrow^*,=)$. The problem is in coNP, and could be coNP-hard. To reduce the complexity, one can vary three 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 any need for guessing where patterns could be put in a target tree. For that, the mapping has to be as specific as possible. In terms of DTDs this restriction is well captured by the notion of nested relational DTDs (for instance, there is no explicit disjunction). But guessing is also involved whenever wildcard and descendant are used in the stds. The mappings which use neither $\downarrow^*$ nor $\_$ in target patterns in stds were called *fully specified*. The following theorem summarizes the results on simple mappings.

THEOREM 3.1. (see [7]) *For every schema mapping $M \in \mathbf{SM}(\downarrow,\downarrow^*,=)$ and every query $Q \in \mathbf{CTQ}(\downarrow,\downarrow^*,=)$*

(1) $\text{CERTAIN}_M(Q)$ is in coNP,

(2) $\text{CERTAIN}_M(Q)$ 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 $\text{CERTAIN}_M(Q)$ is coNP-complete.

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

The result of [7] is actually more precise. For fully specified mappings there is a dichotomy in the first parameter: if DTDs allow enough disjunction, the problem is coNP-hard, otherwise it is polynomial. The exact class of tractable DTDs is the one using so called univocal regular expressions (see [7] for a rather involved definition). Intuitively, it extends nested-relational DTDs with a very week form of disjunction. Query answering in this case is based on constructing a specific instance using a chase procedure, and the use of disjunction in DTDs is limited so as to keep the chase polynomial.

Our goal Given the results of [7], we must stay with a restricted class of DTDs and fully specified mappings to have any hope of getting tractability of query answering. Hence, our questions are:

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

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

In the next section, we show that we do not have to worry about the first question – the coNP bound is not broken by adding new features. We also make an observation about an easy lower bound on the problem that refines question 2 a bit.

4. DATA EXCHANGE IN EXTENDED SETTINGS

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

Note that unlike in some other cases (e.g., relational queries under the closed world semantics [1]), the coNP upper bound on certain answers is nontrivial even in the case of simple downward mappings (3). Now we show that we can recover the upper bound for much more expressive mappings. We do it by casting the problem as a special case of query answering over incomplete XML documents, for which the coNP bound has recently been proved [9].

PROPOSITION 4.1. *For every schema mapping M from $\mathbf{SM}(\downarrow,\downarrow^*,\rightarrow,\rightarrow^*,=,\neq)$ and every query Q from $\mathbf{CTQ}(\downarrow,\downarrow^*,\rightarrow,\rightarrow^*,=,\neq)$, the complexity of $\text{CERTAIN}_M(Q)$ is in coNP.*

Proof sketch. Suppose that the mapping M is of the form $\langle D_s, D_t, \{\varphi_i \to \psi_i \mid i \in [n]\}\rangle$. Given a tree T conforming to D_s, what we have is essentially an incomplete tree. More precisely, we have polynomially many target patterns $\psi_i(\bar{s}_{i,1}), \ldots, \psi_i(\bar{s}_{i,k_i})$, where $i \in [n]$ and k_i is polynomial in the size of T. This set of patterns can be seen as a single XML tree with incomplete information. Thus our problem is equivalent to the following: given an incomplete tree I and a tuple $\bar{s}$, is $\bar{s}$ a certain answer to Q over I? This was shown to be in [9]. $\square$

From the previous section we know that in order to get tractability, we need to confine ourselves to nested relational DTDs and fully specified stds. In addition, we must disallow inequality in the query languages. It is known to lead to coNP-hardness already in the relational case [14, 22]. Since the usual translation from the relational setting to the XML setting produces fully specified nested relational mappings, we have the following result.

COROLLARY 4.2. *There exist a nested-relational fully specified schema mapping M in $\mathbf{SM}^{\mathrm{nr}}(\downarrow,=)$ and a query Q in $\mathbf{CTQ}(\downarrow,=,\neq)$ such that* CERTAIN(Q) *is coNP-complete.*

Our goal – revised Now that we know that inequality in queries immediately leads to intractability, and that we do not have to worry about potentially worse complexity bounds than before, our revised goal is as follows: Can we find $\sigma_1 \supseteq \{\downarrow,=\}$ and $\sigma_2 \supseteq \{\downarrow,\downarrow^*,=\}$ such that $\textsc{certain}_M(Q)$ is tractable for all $M \in \mathbf{SM}^{\mathrm{nr}}(\sigma_1)$ 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. EXTENDING THE MAPPING LANGUAGE

In this section we show that we can extend mappings with horizontal navigation and data value comparisons without losing tractability, provided that we stick to the basic query language. Recall that mappings must be *fully specified* to guarantee tractability, i.e., they use patterns given by the grammar

$$\begin{array}{lcl} \pi & := & \ell(\bar{x})[\lambda] \\ \lambda & := & \varepsilon \mid \mu \mid \lambda,\lambda \\ \mu & := & \pi \mid \pi \to \mu \mid \pi \to^* \mu \end{array} \qquad (3)$$

with $\ell \neq \_$ (i.e., they disallow $\downarrow^*$ and the wildcard), see [7]).

THEOREM 5.1. *Suppose that M is a fully specified schema mapping in $\mathbf{SM}^{\mathrm{nr}}(\downarrow,\to,\to^*,=,\neq)$ and $Q \in \mathbf{UCTQ}(\downarrow,\downarrow^*,=)$. Then* CERTAIN$(Q)$ *is in PTIME.*

Proof sketch. The proof is based on the observation that the chase algorithm of [7] can be extended to handle additional features of schema mappings. Given an source tree T, the algorithm constructs in polynomial time a "minimal" solution T^* for which $Q(T^*) = \mathrm{certain}_M(Q,T)$.

The algorithm roughly works as follows. First, it constructs a *canonical presolution*, which is essentially the result of putting together all the target patterns to be satisfied. That is, for each std $\varphi(\bar{x},\bar{y}), \alpha(\bar{x},\bar{y}) \to \pi'(\bar{x},\bar{z}), \alpha'(\bar{x},\bar{z})$ and tuples $\bar{a},\bar{b}$ so that $\varphi(\bar{a},\bar{b}), \alpha(\bar{a},\bar{b})$ are satisfied in the source tree T, we pick a tuple $\bar{c}$ of fresh nulls so that $\alpha'(\bar{a},\bar{c})$ is satisfied and put the pattern $\pi'(\bar{a},\bar{c})$ into the target tree. This construction only involves evaluating patterns and can be done in polynomial time.

As such a tree need not conform to the target DTD, the algorithm then tries to "repair" the canonical presolution so that it actually is a solution, i.e., conforms to the target DTD. The important property of the class $\mathbf{UCTQ}(\downarrow,\downarrow^*,=)$ is that queries are insensitive to the horizontal order of children in trees. Hence in trying to enforce the conformance to the target DTD, we do not have to reorder children under each node. This repairing procedure might fail, in which case there is no solution for the source tree T. If it successfully terminates, then we have the desired solution T^* and we can compute $Q(T^*)$. $\square$

6. EXTENDING THE QUERY LANGUAGE

We have seen that extending the mapping language is harmless, so the next question is whether we can extend the query language. The answer is exactly the opposite: even very small additions lead to intractability. We start with the simplest mappings and show that extending queries with any form of horizontal navigation leads to intractability. Then, analyzing the causes of this intractability, we consider two modifications of the 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. We show that even under such restrictions, tractability cannot be recovered.

6.1 Simple mappings

We start with the simplest class of mappings, those with only the child-based navigation, with fully specified patterns, and nested relational DTDs. For such mappings, we cannot extend the query language $\mathbf{CTQ}(\downarrow,\downarrow^*,=)$ with any form of horizontal navigation.

THEOREM 6.1. *There exist*

- *a fully specified schema mapping $M \in \mathbf{SM}^{\mathrm{nr}}(\downarrow)$;*
- *a query $Q_1 \in \mathbf{CTQ}(\downarrow,\to,=)$; and*
- *a query $Q_2 \in \mathbf{CTQ}(\downarrow,\to^*,=)$*

such that both $\textsc{certain}_M(Q_1)$ *and* $\textsc{certain}_M(Q_2)$ *are coNP-complete.*

Proof. The coNP upper bound follows from Proposition 4.1. For the lower bound, we prove the first claim of the theorem, and the proof for the second can be obtained by replacing $\to^*$ with $\to$ in the following proof.

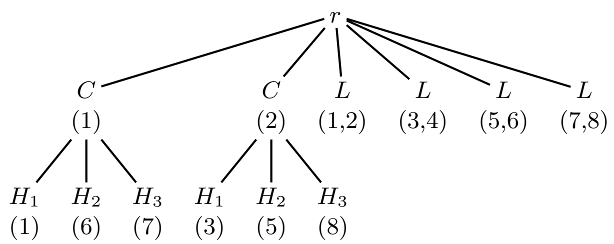
We describe an XML schema mapping M and a query $Q \in \mathbf{CTQ}(\downarrow,\to,=)$ such that 3SAT is reducible to the complement of $\textsc{certain}_M(Q)$. More specifically, there exist an XML schema mapping M and a Boolean query Q such that

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

where T_φ is a tree encoding of the formula φ.

The idea of the reduction is the following: we transform a 3CNF formula φ into a source tree T_φ. The mapping is defined so that a solution of T_φ 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 implies the existence of a well-defined (partial) assignment that satisfies the formula φ.

Suppose we are given a 3-CNF formula $\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} c_{ij}$, where c_{ij} is a literal. We construct a source tree T_φ 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, and H_i holds the data value encoding the 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

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

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

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

The last component of the schema mapping are the stds Σ. The idea for the mapping is that, given T_φ, 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 following siblings. Since each C node has three H nodes below, clearly at least one literal is chosen for each clause.

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

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

$$\exists x \exists y \left(\begin{array}{ll} r\left[L(x,y),\right. & C[H(x) \to H \to H], \\ & \left. C[H(y) \to H \to H]\right] \end{array} \right)$$

Formally, the correctness of the reduction can be proved as follows. We prove that $\text{certain}_M(Q, T_\varphi)$ is false if and only if a 3-CNF formula φ 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 of φ 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 φ. 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 following siblings, so that the corresponding literal is true by the assignment.

($\Leftarrow$) Suppose that φ is satisfiable. Assume v is a satisfying assignment. Then we construct a solution of the mapping for T_φ that falsifies 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 following 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 (we choose the one corresponding to a literal with the smallest index if there are more than one literal to which v assigns true). For example, suppose it is d_2. Then we make the tree fragment $r[C[H(d_2) \to H(d_1) \to H(d_3)]]$. 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. $\square$

6.2 Fully specified sibling order

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

Observe that in the setting of the previous section, we violated the idea behind the principle of being *fully specified* (although not the formal definition), as queries used horizontal navigation, and yet mappings did not specify it completely, by allowing the set constructor λ, λ' in (3). Such nondeterminism in placing patterns in target trees leads to intractability.

So it seems natural to restrict the use of this nondeterminism and properly redefine the notion of being fully specified for horizontal navigation. We do it now, but show that, unlike in the easier case of [7], it does *not* lead to tractability.

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

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

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

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

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

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

We start with the $\to$-fully specified mappings.

Theorem 6.2. *There exist a $\to$-fully specified schema mapping $M \in \mathbf{SM}^{\mathrm{nr}}(\downarrow, \to)$ and a query $Q \in \mathbf{CTQ}(\downarrow, \to, =)$ such that* CERTAIN(Q) *is coNP-complete.*

Proof sketch. 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_φ is the same encoding of formulas as in the previous proof.
The source DTD D_s is:

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

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'.

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

The stds are simply copying :

$$\begin{array}{rcl} r[L(x,y)] & \to & r[L(x,y)] \\ r[C[H_1(x) \to H_2(y) \to H_3(z)]] & \to & \\ & & r[C[H_1(x) \to H_2(y) \to H_3(z)]] \end{array}$$

The query Q is selected so that it is true if a variable and its negation are both 'chosen', just as in the previous proof. It is given by :

$$\exists x \exists y \left(\begin{array}{ll} r\big[L(x,y), & C[G_1 \to \_ \to \_ \to \_(x)], \\ & C[G_1 \to \_ \to \_ \to \_(y)]] \end{array} \right)$$

Correctness is shown in the appendix. □

We now show intractability for $\to^*$-fully specified mappings.

Theorem 6.3. *There exist a $\to^*$-fully specified schema mapping $M \in \mathbf{SM}^{\mathrm{nr}}(\downarrow, \to^*)$ and a query Q from the class $\mathbf{CTQ}(\downarrow, \to^*, =)$ such that* CERTAIN(Q) *is coNP-complete.*

Proof sketch. As before we provide $M = \langle D_s, D_t, \Sigma \rangle$ and a query Q to which 3SAT is reducible to the complement of CERTAIN$_M(Q)$.

The DTD D_s is similar to the one we used 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 φ as T_φ, by simply dropping the subscript in the previous encoding.

The target DTD is the following:

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

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:

$$\begin{array}{c} r/C(x)/H(y) \to r/A(x,y) \\ r/L(x,y) \to r/L(x,y) \end{array}$$

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

Finally 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. It is given by

$$\begin{array}{l} r[L(x,y), r[A(v,x) \to^* A(v,u_1) \to^* A(v,u_2)], \\ \qquad A(w,y) \to^* A(w,u_3) \to^* A(w,u_4)] \end{array}$$

with all the variables $x, y, v, w, u_1, u_2, u_3, u_4$ quantified existentially. Note that the stds in M do not use $\to^*$.

Observe 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. □

6.3 Threshold DTDs

To motivate our last attempt to find a tractable class, consider the following example. Suppose we have a 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.

Such an approach was taken by [2]. They used *threshold DTDs* which assign each element type ℓ its *multiplicity atom* $\mu(\ell)$, an expression of the form $\hat{\ell}_1 \cdots \hat{\ell}_m$, where $\hat{\ell}$ is one of ℓ_i, ℓ_i^*, ℓ^+, and $\ell? = \ell_i$, which limits the number of children of type ℓ_i of a node labeled ℓ in the obvious way, without imposing any restriction on the order of the children. For example, $a^* b^*$ viewed as a multiplicity atom says that there are some (perhaps none) a's and some b's, but not that all a's should precede all b's, as the usual regular expression would say.

We write $\mathbf{SM}^{\mathrm{th}}$ and $\mathbf{SM}^{\mathrm{th}}(\sigma)$ for classes of schema mappings using such threshold DTDs.

Do such mapping admit a better algorithm for computing certain answers? Again, the answer is negative, this time for unions of conjunctive queries (note that previous results for tractable query answering in both relational and XML data exchange work for both conjunctive queries and their unions).

THEOREM 6.4. *There exist*

- *a* $\rightarrow$*-fully specified schema mapping* M *from* $\mathbf{SM}^{\mathrm{th}}(\downarrow, \rightarrow)$, *and a query* Q *from* $\mathbf{UCTQ}(\downarrow, \rightarrow, =)$,
- *a* $\rightarrow^*$*-fully specified schema mapping* M' *from* $\mathbf{SM}^{\mathrm{th}}(\downarrow, \rightarrow)$, *and a query* Q' *from* $\mathbf{CTQ}(\downarrow, \rightarrow^*, =)$,

such that both $\text{CERTAIN}_M(Q)$ *and* $\text{CERTAIN}_{M'}(Q')$ *are coNP-complete.*

Proof sketch. For the second item, the proof of Theorem 6.3 applies verbatim, so we prove only the first item. 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 the complement of $\text{CERTAIN}_M(Q)$. We use the same encoding T_φ of a given 3CNF formula φ as in the proof of Theorem 6.1.

The idea of the reduction is the following: we transform a 3CNF formula φ into a source tree T_φ. The mapping is defined so that a solution of T_φ 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 φ. 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.

$$\begin{array}{l} r[C[H_1(x) \rightarrow H_2(y) \rightarrow H_3(z)]] \;\rightarrow \\ \qquad r[C[H_1(x) \rightarrow H_2(y) \rightarrow H_3(z)]], \\ r[L(x,y) \rightarrow r[L(x,y)]. \end{array}$$

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

$$q_1 = \bigcup_{i,j\in\{1,2,3\}} \exists x \exists y \left(\begin{array}{ll} r\big[L(x,y), & C[H_i(x)[B \rightarrow A]], \\ & C[H_j(y)[B \rightarrow A]]\big] \end{array} \right)$$

and

$$q_2 = r\Big[C\big[H_1[B \rightarrow A] \rightarrow H_2[B \rightarrow A] \rightarrow H_3[B \rightarrow A]\big]\Big]$$

The correctness of the reduction follows from these two observations:

- Due to q_1, the assignment is consistent;
- Due to q_2, each clause has at least one variable assigned true.

$\square$

7. CONCLUSION

We have studied query answering for XML data exchange with the language allowing vertical and horizontal navigation and data comparisons. 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.

Figure 4 presents the summary of the main results. When we write coNP, we mean that the problem could be coNP-complete for some choice of a mapping and a query from the relevant classes (and is in coNP for all such choices). We use 'f.s.' as an abbreviation for 'fully specified'. The last line says that beyond the class of fully specified mappings, there is no hope to get tractability. Within the class of fully specified mappings, it is clear that we have the freedom to increase the expressiveness of the mappings, but not the queries.

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

There are several directions to extend the results of this paper. So far, we have concentrated on data complexity of the problem. We would also like to look at combined complexity in the future in order to have a better understanding of query answering in XML schema mappings.

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 [6].

Yet anther dimension that has not been investigated is the distinction between open world assumption (OWA) and closed world assumption (CWA). Here, we have worked under OWA. In the relational case, an anomaly is observed when the query involves negation [5, 14]. As a remedy to such unintuitive behaviour, the notion of solutions under CWA was proposed in [20], further extended in [17, 21, 3]. This direction is hardly explored for XML: it is not even clear how to define the notion of CWA in the XML context.

Acknowledgments The authors were supported by EPSRC grants E005039 and F028288, and the FET-Open Project FoX (grant agreement 233599).

Queries / Mappings	$\mathbf{CTQ}(\downarrow,=)$	$\mathbf{CTQ}(\downarrow,\downarrow^*,=)$	$\mathbf{CTQ}(\downarrow,\rightarrow,=)$	$\mathbf{CTQ}(\downarrow,\rightarrow,\rightarrow^*,=)$	$\mathbf{CTQ}(\downarrow,=,\neq)$
f.s. $\mathbf{SM}^{\mathrm{nr}}(\downarrow,=)$	PTIME	PTIME [7]	coNP	coNP	coNP [14, 22]
$\rightarrow$-f.s. $\mathbf{SM}^{\mathrm{nr}}(\downarrow,\rightarrow,=)$	PTIME	PTIME	coNP	coNP	coNP
$\rightarrow^*$-f.s. $\mathbf{SM}^{\mathrm{nr}}(\downarrow,\rightarrow^*,=)$	PTIME	PTIME	coNP coNP	coNP (even for queries in $\mathbf{CTQ}(\downarrow,\rightarrow^*,=)$)	coNP
f.s. $\mathbf{SM}^{\mathrm{nr}}(\downarrow,\downarrow^*,\rightarrow,\rightarrow^*,=,\neq)$	PTIME	PTIME	coNP	coNP	coNP
$\mathbf{SM}(\downarrow,=)$	coNP [7]	coNP	coNP	coNP	coNP

Figure 4: The complexity of $\text{CERTAIN}_M(Q)$

8. REFERENCES

[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theor. Comp. Sci.* 78 (1991), 158–187.

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

[3] F. Afrati, Ph. Kolaitis. Answering aggregate queries in data exchange. In *PODS 2008*, pages 129–138.

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

[5] M. Arenas, P. Barceló, R. Fagin, and L. Libkin. Locally consistent transformations and query answering in data exchange. In *PODS 2004*, pages 229–240.

[6] M. Arenas, P. Barceló, and J. Reutter. Query languages for data exchange: Beyond unions of conjuctive queries. In *ICDT 2009*, pages 73–83.

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

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

[9] P. Barceló, L. Libkin, A. Poggi, and C. Sirangelo. XML with incomplete information: models, properties, and query answering. In *PODS 2009*, pages 237–246.

[10] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, 2007.

[11] G. J. Bex, F. Neven, and J. van den Bussche. DTDs versus XML schema. In *WebDB*, pages 79–84, 2004.

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

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

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

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

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

[17] A. Heinrich and N. Schweikardt. CWA-solutions for data exchange settings with target dependencies. In *PODS 2007*, pages 113–122.

[18] Informatica PowerCenter Transformation Language Reference. Version 8.1.1, September 2006.

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

[20] L. Libkin. Data exchange and incomplete information. In *PODS 2006*, pages 60–69.

[21] L. Libkin and C. Sirangelo. Data exchange and schema mappings in open and closed worlds. In *PODS 2008*, pages 139–148.

[22] A. Madry. Data exchange: on the complexity of answering queries with inequalities. *IPL*, 94:253–257, 2005.

[23] W. Martens, F. Neven, Th. Schwentick. Simple off the shelf abstractions for XML Schema *SIGMOD Record* 36(3): 15–22 (2007).

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

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