Certain Answers for XML Queries

Claire David
University of Edinburgh
cdavid@inf.ed.ac.uk

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

Filip Murlak
University of Warsaw
fmurlak@mimuw.edu.pl

ABSTRACT

The notion of certain answers arises when one queries incompletely specified databases, e.g., in data integration and exchange scenarios, or databases with missing information. While in the relational case this notion is well understood, there is no natural analog of it for XML queries that return documents.

We develop an approach to defining certain answers for such XML queries, and apply it in the settings of incomplete information and XML data exchange. We first revisit the relational case, and show how to present the key concepts related to certain answers in a new model-theoretic language. This new approach naturally extends to XML. We prove a number of generic, application-independent results about computability and complexity of certain answers produced by it. We then turn our attention to a pattern-based XML query language with trees as outputs, and present a technique for computing certain answers that relies on the notion of a basis of a set of trees. We show how to compute such bases for documents with nulls and for documents arising in data exchange scenarios, and provide complexity bounds. While in general complexity of query answering in XML data exchange could be high, we exhibit a natural class of XML schema mappings for which not only query answering, but also many static analysis problems can be solved efficiently.

Categories and Subject Descriptors. H.2.4 [**Database Management**]: Systems—*Query processing*; H.2.5 [**Database Management**]: Heterogeneous Databases—*Data translation*; I.7.2 [**Document and Text Processing**]: Document Preparation—*XML*

General Terms. Theory, Languages, Algorithms

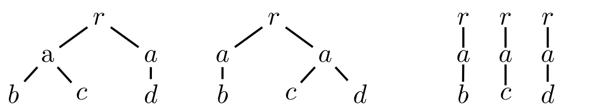
Keywords. Incomplete information, certain answers, data exchange, queries returning trees

1. INTRODUCTION

The notion of *certain answers* is one of the central concepts in database theory, applied when the queried database is not known precisely. Such incompleteness of information often arises in data exchange or integration, when the database we query is only partly defined via source data and mappings between schemas. Semantically, an incomplete description of a database is a set $\mathcal{D}$ of completely described databases that it can represent. If we have a query Q, then the most common (but not the only) way of defining certain answers is to take tuples that belong to the result no matter which complete database in $\mathcal{D}$ is used; that is, $\mathsf{certain}(Q, \mathcal{D}) = \bigcap\{Q(D) \mid D \in \mathcal{D}\}$. But as we move from relational data to more complex formats, such as XML, this definition immediately becomes problematic. Queries in XML languages (e.g., XQuery) return trees, rather than set of tuples. What is a natural analog of certain answers then?

In the literature on XML with incomplete information (e.g., [3, 9, 22, 28, 29]) and on XML data exchange, integration, and query answering using views (e.g., [5, 6, 15, 16, 37]) one usually avoids this question by either considering queries returning tuples of scalar values, or single nodes, or computing query answers on one specific instance. Such simplifications are very useful for initial investigations, as they tell us when tractable evaluation mechanisms are possible for simple queries. Now that we know the answers, it is time to move to proper XML queries. To do so, however, we need to understand the notion of certain answers.

To define $\mathsf{certain}(Q, \mathcal{T})$, for a query Q returning trees and a set $\mathcal{T}$ of XML trees, we need an analog of the intersection operator applied to $Q(\mathcal{T}) = \{Q(T) \mid T \in \mathcal{T}\}$. What could it be? The first idea is to take the maximal tree contained in all of trees in $Q(\mathcal{T})$. But this may fail as such a tree need not be unique: if T_1 and T_2 shown below are in $Q(\mathcal{T})$, there are three maximal trees subsumed by them shown in part (c) of the figure:



(a) T_1 (b) T_2 (c) Maximal trees

One might be tempted to combine these trees in (c) into one, but this is not satisfactory either. We have a choice of merging the roots, or the a-nodes, but either way the resulting tree is not subsumed by T_1 and T_2; in what sense, then, is it a certain answer? And so far we have not considered *data values*. But what if in T_1 and T_2, the left a carries value "1" and the right a carries "2"? Then the middle tree with the c leaf is not among maximal subsumed trees, and without it we lose the certain knowledge about a c-grandchild of the root.

Thus, even in a very simple example, it is not completely clear what the natural notion of certain answers is. To define this notion we (as happens with so many problems related to XML data management), start by re-examining the relational case.

There are two established relational approaches to certain answers. The first, based on the work of Imieliński and Lipski [27], finds a representation of the set $Q(\mathcal{D})$ of answers to Q on a set of databases $\mathcal{D}$. It could be $\bigcap Q(\mathcal{D})$, or more generally, an incomplete database that in some way describes $Q(\mathcal{D})$ – this is the idea of representation systems [1, 27, 24]. The concept of "certainty" is thus decoupled from querying: one finds certain information from an arbitrary collection of databases, which, for the case of certain answers, is $Q(\mathcal{D})$. The second approach originates in the work of Reiter [32], and couples the notion of certainty with querying much closer: it views an incomplete database as a theory L in some logical language, and certain answers to a query Q are those that are implied by L.

To extend these to XML, we use a *hybrid* approach, combining the ideas of [27] and [32]: from [27] we take the decoupling of the notion of certainty and query answering, and from [32], the logical approach to defining certainty. We make the following contributions.

1. We present a generic model-theoretic (rather than Reiter's proof-theoretic) explanation of the notion of certainty in a collection of databases.
2. We show how to apply the same approach to XML, and investigate its basic properties.
3. We use this approach with an XML query language that returns trees, and provide conditions that guarantee computability of certain answers.
4. We do two case studies, for XML documents with incomplete information, and for data exchange guided by XML schema mappings, and show how to compute certain answers in both.
5. The complexity of certain answers in XML data exchange could be quite high, so we find a natural class of schema mappings that admit both tractable static analysis and query answering.

We now elaborate more on these contributions.

(1) The key element of our approach is the notion of a logical theory that presents the certain knowledge about a collection $\mathcal{D}$ of databases. We then introduce the notion of a *max-description* of $\mathcal{D}$; it is a small set of logical formulae providing as much information about it as possible within a given logical language. Max-descriptions can be viewed as structures, and for all cases of interest, they share the same *core* (the smallest substructure which is also a homomorphic image [26]; this notion has many applications in database theory [20, 25, 17, 20]).

(2) We apply the same approach to collections of XML trees. The logical languages in which theories of certain knowledge are defined over relations are usually (ground) conjunctive queries; as their natural XML analogs, we use tree patterns. We then define max-descriptions and prove some of their basic properties, such as existence and computability. All max-descriptions of a family $\mathcal{T}$ of XML trees have the same core, called the core-description of $\mathcal{T}$. We prove characterizations of max- and core-descriptions and establish bounds on their size and complexity when $\mathcal{T}$ is finite.

(3) We define a natural extension of pattern-based languages used previously in the setting of incomplete XML or XML data exchange and integration [6, 9, 11, 12, 16, 28, 29]. This extension has the flavor of FLWR expressions of XQuery, and allows trees as outputs of queries. We define certain answers to Q over $\mathcal{T}$ as max-descriptions of $Q(\mathcal{T})$. To compute them, we introduce the notion of a *basis* of $\mathcal{T}$, and prove certain answers to Q over $\mathcal{T}$ can be computed from a basis of $\mathcal{T}$ itself.

(4) We do two case studies and show how bases, and thus certain answers, can be computed. We first look at XML documents with null values. Our technique immediately implies that for such documents, certain answers to FLWR-like queries can be found by naïve evaluation in polynomial time, extending results in [9].

The second application is about XML data exchange, i.e., restructuring a document conforming to a source schema under a target schema and then answering queries over target instances. The relational case has seen much activity lately (see, e.g., surveys [30, 8, 10]), but for XML data exchange, only the complexity of simple queries is known [6, 4]. We show that for large classes of mappings, bases can be computed, and thus certain answers to FLWR-like queries can be found too. Without further restrictions, however, the complexity could be rather high.

(5) Motivated by the last observation, we find a natural class of XML schema mappings that exhibit good complexity of query answering and static analysis problems. It extends non-relational mappings used in data exchange [31, 37]. Consistency checking, composing mappings, building target instances, and answering FLWR-like queries can all be done efficiently for this class.

Organization The paper follows the main five areas of contribution outlined above, with Sections 2–6 corresponding to items (1)–(5).

2. RELATIONAL CERTAIN ANSWERS

We start with some standard definitions related to tables with incomplete information (cf. [1, 27, 24]). Assume that we have two disjoint countable sets Const of constants that appear in complete databases and Var of variables. A *naïve table* is a table whose entries come from $\mathsf{Const} \cup \mathsf{Var}$. To emphasize that a relation only has entries from Const, we call it a *ground* relation.

A *homomorphism* between tables R and R' of the same arity is a mapping $h : \mathsf{Var} \to \mathsf{Const} \cup \mathsf{Var}$ such that for each tuple $t = (v_1, \ldots, v_m) \in R$, the tuple $h(t) = (h(v_1), \ldots, h(v_m))$ is in R'. We always assume that such mappings are extended to mappings $\mathsf{Const} \cup \mathsf{Var} \to \mathsf{Const} \cup \mathsf{Var}$ by taking $h(c) = c$ for all $c \in \mathsf{Const}$. A ground relation D *represents* R if there is a homomorphism $h : R \to D$. We denote the set of all ground relations representing R by $Rep(R)$.

A naïve table R with tuples $t_1, \ldots, t_m$ and variables $x_1, \ldots, x_n$ defines a Boolean *conjunctive query* (CQ) $Q_R = \exists x_1, \ldots, x_n\ R(t_1) \wedge \ldots \wedge R(t_m)$. That is, the tableau of the CQ Q_R is R itself. If R is a ground table, then Q_R is a quantifier-free CQ. Note that $D \in Rep(R)$ iff Q_D is contained in Q_R, and $Rep(R) = Rep(R')$ iff Q_R and $Q_{R'}$ are equivalent CQs.

The idea of certain answers is to extract the maximum knowledge from answers $Q(D)$, as D ranges over some collection of databases $\mathcal{D}$ (e.g., $Rep(R)$, or instances of a global schema in data integration). In the approach of Imieliński and Lipski [27] based on *representation systems*, this maximum knowledge is defined in such a way that it compactly represents the set of all tuples that appear with certainty in $Q(\mathcal{D})$. What it means precisely depends on the *logical language* we use for specifying sets of tuples. There are two most common cases.

If we use the *language of ground relations*, then $\Box_{\rm gr} Q(\mathcal{D}) = \bigcap Q(\mathcal{D})$ (the set of ground tuples that appear in every query answer $Q(D)$ for $D \in \mathcal{D}$) compactly represents the maximum knowledge we can extract from $Q(\mathcal{D})$. This is the most commonly used (but not the only) notion of certain answers seen in the literature. However, if we use the more expressive *language of naïve tables*, we can extract additional knowledge from $\mathcal{D}$ by means of a naïve table R so that $\bigcap Q(\mathcal{D}) = \bigcap Rep(R)$. This is the idea behind weak representation systems of [27]. Those based on naïve tables are used for evaluating positive relational algebra queries over incomplete databases, or in data integration and exchange [19, 8][1].

The second approach, due to Reiter [32, 33], is to view query answering as *logical implication* by a theory that represents the certain knowledge of a class of databases $\mathcal{D}$. We view $\mathcal{D}$ as as a logical theory $L_{\mathcal{D}}$ such that $\mathcal{D} = \mathrm{Mod}(L_{\mathcal{D}})$, the set of models of $L_{\mathcal{D}}$. For example, if R is a naïve table, then $L_{Rep(R)}$ is the query Q_R. Given a query $Q(\bar{x})$, certain answers are those that are implied by $L_{\mathcal{D}}$. For example, if we are interested in ground facts, then we look for ground tuples $\bar{a}$ so that $L_{\mathcal{D}} \vdash Q(\bar{a})$. More generally, we can look for implication statements $L_{\mathcal{D}} \vdash \exists \bar{x} Q(\bar{a}, \bar{x})$, which correspond to computing naïve tables as query answers.

[1]We shall not be looking at more expressive representation mechanisms such as conditional tables for now, due to their added complexity [2].

Certain answers revisited We now revisit the notion of certain answers, using both approaches. We saw that the notion of certain answers is not unique and depends on a representation language. The approach of Imieliński-Lipski nicely decouples certainty and querying: it could be applied to any collection of databases, not just $Q(\mathcal{D})$. Reiter's approach puts these notions together, but makes an important point that certain information has to faithfully represent the theory of a set of databases. We take the decoupling of querying and certainty from the first approach, and then follow the spirit of Reiter's approach, replacing proof-theoretic concepts (which are problematic in the context of finite structures) by *model-theoretic*.

We define a notion of a *max-description* of a set $\mathcal{D}$ of databases that, in a given language, provides a description of information we can infer with certainty from $\mathcal{D}$. Certain query answers are then a special case of max-descriptions, applied to sets $\{Q(D)\}$ as D ranges over a collection of databases.

To explain this notion, assume that $\mathcal{L}$ is a logical formalism in which we express properties of databases from $\mathcal{D}$ (e.g., CQs or ground facts). To describe $\mathcal{D}$ fully in $\mathcal{L}$ we would need its $\mathcal{L}$-definition: a finite set Φ of $\mathcal{L}$-formulae so that $\mathcal{D} = \mathrm{Mod}(\Phi)$ (the set of all models of Φ). This is not always achievable, so instead we settle for the next best thing, which is an $\mathcal{L}$-definition of the *certain knowledge* about $\mathcal{D}$ that is expressed in $\mathcal{L}$.

This certain $\mathcal{L}$-knowledge of the class $\mathcal{D}$ is $\mathrm{Th}_{\mathcal{L}}(\mathcal{D})$, the $\mathcal{L}$*-theory of* $\mathcal{D}$, i.e., the set of all formulae from $\mathcal{L}$ satisfied in all structures from $\mathcal{D}$. To extract its maximal description, expressed again in $\mathcal{L}$, we look for finite sets Φ of formulae that define $\mathrm{Mod}(\mathrm{Th}_{\mathcal{L}}(\mathcal{D}))$, the models of the certain knowledge of $\mathcal{D}$. This leads to the following.

Definition 2.1 *A finite set of $\mathcal{L}$-formulae Φ is called a* max-$\mathcal{L}$-description *of $\mathcal{D}$ if* $\mathrm{Mod}(\Phi) = \mathrm{Mod}(\mathrm{Th}_{\mathcal{L}}(\mathcal{D}))$. *When $\mathcal{L}$ is clear from the context, we talk about a* max-description *of $\mathcal{D}$.*

It turns out that we reconstruct familiar relational notions of certain answers with this concept. Assume first that $\mathcal{L}$ is the language of ground facts and their conjunctions. For a set $\mathcal{D}$ of complete databases or naïve tables, its $\mathcal{L}$-theory, $\mathrm{Th}_{\mathcal{L}}(\mathcal{D})$ is the set of all ground tuples that occur in all elements of $\mathcal{D}$, i.e., $\bigcap \mathcal{D}$ (as well as conjunctions of these ground facts, since the theory is closed under conjunction). Thus, a max-description of $\mathcal{D}$ contains all ground facts from $\bigcap \mathcal{D}$ and no others; if duplicates are eliminated, we get precisely $\Box_{\rm gr}\mathcal{D} = \bigcap \mathcal{D}$.

Now let $\mathcal{L}$ be the language of CQs, and $\mathcal{D}$ a set of ground relations. A max-description can be viewed as both a CQ and a naïve table. Viewed as a naïve table R, it has the property that $Rep(R) = \bigcap Rep(R')$ as R' ranges over naïve tables satisfying $\mathcal{D} \subseteq Rep(R')$: so it is indeed a naïve table that is closest to describing $\mathcal{D}$. One can also observe that the set of ground tuples in R is exactly $\bigcap \mathcal{D}$, i.e., it defines a weak representation system of [27]. By looking at max-descriptions as CQs, we note that any two of them are equivalent, and hence all share the same minimization, denoted by $\square\mathcal{D}$.

Thus, in both examples, even though max-descriptions need not be unique, they have the same minimizations (obtained by duplicate elimination or by CQ minimization). Technically, these are the *cores* [26]: the smallest substructures of max-descriptions which are also their homomorphic images; this follows from well-known results on CQ minimization [17].

Let us now summarize what we have learned from this re-examination of the relational case.

1. The notion of certain answers depends on the choice of language for specifying tuples. Those used most often in the relational case are (ground) conjunctive queries.
2. A max-description of $\mathcal{D}$ with respect to $\mathcal{L}$ is then defined as an $\mathcal{L}$-formula that in the best way approximates the certain $\mathcal{L}$-knowledge of $\mathcal{D}$.
3. Max-descriptions need not be unique, but they have isomorphic cores.

3. MAX-DESCRIPTIONS FOR XML TREES

We now apply the methodology of the previous section to XML trees. First, we define XML documents themselves. An XML tree, over a labeling alphabet Γ, is a structure $T = \langle U, \downarrow, \ell ab, r, \rho\rangle$, where

- U is a set of nodes, and $r \in U$;
- $\downarrow$ is a binary relation on U so that $\langle U, \downarrow, r\rangle$ is a tree with root r;
- $\ell ab : U \to \Gamma$ is a labeling function;
- function ρ assigns to each node in U a (possibly empty) tuple of values from $\mathsf{Const} \cup \mathsf{Var}$.

Note that for now we disregard the sibling ordering in XML trees (as is common in exchange/integration applications, and as often happens when one deals with incompleteness [6, 15, 3, 9]). Also we do not introduce attribute names to keep notations simple; each node just comes with a tuple of its attribute values, some of which could be variables. We further assume that $\ell ab(r)$, the label of the root, is always a special label designated for the root of a tree.

We now apply the ideas of the previous section to sets $\mathcal{T}$ of XML trees. We shall be making two assumptions about such sets $\mathcal{T}$: (a) all trees in them are labeled by some fixed alphabet, and (b) nodes with the same label have tuples of attributes of the same length. This will be true in all the applications (either enforced by a DTD, or by an incomplete tree). The assumptions are not really required but help keep notations simple.

First, we need analogs of (ground) conjunctive queries. Since relational CQs can be seen as naïve tables (query tableau), trees with variables are a natural analog of CQs for trees. In this role, they are usually presented as tree *patterns* [23, 11, 12, 6], and we use a special syntax for them, that can later be extended when we define our query language. Patterns are given by:

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

where $\ell \in \Gamma$. We use the convention that $\bar{\alpha}$ refers to tuples of elements from $\mathsf{Const}\cup\mathsf{Var}$. Tuples over Const will be denoted by $\bar{a}, \bar{b}, \ldots$ and tuples over Var by $\bar{x}, \bar{y}, \ldots$; sometimes we shall write $(\bar{a}, \bar{x})$ instead of $\bar{\alpha}$ to explicitly name variables and constants. We write $\pi(\bar{\alpha})$ to name all variables and constants used in a pattern. We also abbreviate $\ell(\bar{\alpha})[\varepsilon]$ as $\ell(\bar{\alpha})$.

Let $v : \mathsf{Var} \to \mathsf{Var} \cup \mathsf{Const}$ be a valuation defined on all variables in π. We define the notion of a tree T satisfying π at node s wrt v, written $(T, s, v) \models \pi$:

- $(T, s, v) \models \ell(\bar{\alpha})$ iff $\ell ab(s) = \ell$ and $\rho(s) = v(\bar{\alpha})$;
- $(T, s, v) \models \ell(\bar{\alpha})[\lambda_1, \lambda_2]$ iff $(T, s, v) \models \ell(\bar{\alpha})[\lambda_1]$ and $(T, s, v) \models \ell(\bar{a})[\lambda_2]$;
- $(T, s, v) \models \ell(\bar{\alpha})[\pi]$ iff $(T, s, v) \models \ell(\bar{\alpha})$ and $(T, s', v) \models \pi$ for some s' with $s \downarrow s'$.

We write $T \models_v \pi(\bar{\alpha})$ if $(T, r, v) \models \pi(\bar{\alpha})$, and $T \models \pi(\bar{\alpha})$ if $T \models_v \pi(\bar{\alpha})$ for some valuation v. For a set of patterns B, we define $\mathrm{Mod}(B) = \{T \mid \forall \pi \in B : T \models \pi\}$.

We shall write $\Pi(\mathsf{Const}, \mathsf{Var})$ for the set of all patterns and $\Pi(\mathsf{Const})$ for the set of *ground* patterns that do not use variables; these are natural analogs of CQs and ground CQs. For a set $\mathcal{T}$ of XML trees we then define its theory $\mathrm{Th}(\mathcal{T})$ and its ground theory $\mathrm{Th}_{\mathrm{gr}}(\mathcal{T})$ as

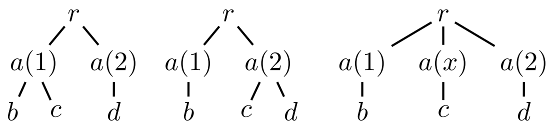
$$\mathrm{Th}(\mathcal{T}) = \{\pi \in \Pi(\mathsf{Const}, \mathsf{Var}) \mid \forall T \in \mathcal{T} : T \models \pi\}$$

$$\mathrm{Th}_{\mathrm{gr}}(\mathcal{T}) = \{\pi \in \Pi(\mathsf{Const}) \mid \forall T \in \mathcal{T} : T \models \pi\}$$

These theories provide the certain knowledge (in the corresponding language) that can be extracted from $\mathcal{T}$. Max-descriptions are then the closest we can get to defining $\mathcal{T}$.

Definition 3.1 *Let $\mathcal{T}$ be a set of XML trees. A pattern $\pi \in \Pi(\mathsf{Const}, \mathsf{Var})$ is a* max-description *of $\mathcal{T}$ if $\mathrm{Mod}(\pi) = \mathrm{Mod}(\mathrm{Th}(\mathcal{T}))$.*
A pattern $\pi \in \Pi(\mathsf{Const})$ is a ground max-description *of $\mathcal{T}$ if $\mathrm{Mod}(\pi) = \mathrm{Mod}(\mathrm{Th}_{\mathrm{gr}}(\mathcal{T}))$.*

We now illustrate the notion of max-descriptions by revisiting the example of trees T_1 and T_2 in the figure shown in the introduction. A max-description for them is obtained by merging all the trees in part (c) of that figure at the root, rather than at the a-nodes, as indeed seemed intuitive. We also asked about a modification of the example where the a-nodes carry data values, as shown in (d) and (e) in the figure below. Then a max-description of $\{T_1', T_2'\}$ is the tree in (f) of the figure. It has one occurrence of a variable and preserves the certain knowledge about the c-node.



(d) T_1' (e) T_2' (f) a max-description

The first easy observation shows that it often suffices to work with max-descriptions. Let ground(π) be a ground pattern obtained from π by dropping all the subpatterns $\ell(\bar{a}, \bar{x})[\ldots]$ with nonempty $\bar{x}$.

Proposition 3.2 *If π is a max-description of $\mathcal{T}$, then* ground(π) *is a ground max-description of $\mathcal{T}$.*

We next show that in many cases (certainly those relevant in the applications we consider), max-descriptions exist and can be computed.

Proposition 3.3 *For every finite nonempty set of trees $\mathcal{T}$, a max-description exists and can be computed. For every nonempty set of ground trees, a ground max-description exists, and can be computed if $\mathcal{T}$ and* Th$(\mathcal{T})$ *are recursive.*

Proof. The finite case is covered by Proposition 3.8. Let us concentrate on the infinite case.

Recall every label in Γ has a fixed number of attributes. Let N be the maximal number of attributes. Let C be a finite set of constants and let Π_C denote the set of ground patterns over C. We will show that there are only finitely many non-equivalent tree patterns from Π_C of depth less then d, and that there is a computable bound on their size.

We proceed by induction. For depth 0, every pattern is of the form $\ell(\bar{c})$ or ℓ. Clearly, there are at most $k_0 = |\Gamma| \cdot |V|^N$ of them, and each has $n_0 = 1$ node. Suppose that there are at most k_i non-equivalent patterns of depth i, and all of them have at most n_i nodes. Let us look at depth $i+1$. A pattern is defined by its root and the set of subpatterns. As we consider only ground patterns, observe that it makes no sense to put two equivalent subpatterns into the pattern. Hence, there is at most $n_0 \cdot 2^{n_i}$ non-equivalent patterns of depth at most $i+1$. Each of that patterns has the size at most $n_i = 1 + k_i n_i$. The claim follows.

Now, we need to show that for a given set of ground trees, there exists a ground max-description. Since $\mathcal{T}$ is nonempty, there exists $T \in \mathcal{T}$. If a pattern is to be satisfied T, its depth cannot be larger that the depth of T, and it cannot use constants not used in T. By the claim above, there are only finitely many ground patterns satisfied in T. Let this set of patterns be called Π_T. Now, let $\Pi_{\mathcal{T}}$ be the set of patterns from Π_T that are satisfied in every tree from $\mathcal{T}$. Let π be the conjunction of all patterns from $\Pi_{\mathcal{T}}$, i.e., the patterns obtained by merging all patterns from $\Pi_{\mathcal{T}}$ at the root (recall that all trees have the same label for the root). It is obvious that π is a max-description of $\mathcal{T}$.

Assume now that both $\mathcal{T}$ and Th$(\mathcal{T})$ are recursive. Let us see that we can compute a ground max-description. First, find a tree $T \in \mathcal{T}$: construct all trees over the alphabet Γ, and Const one by one, and for each check if it is in $\mathcal{T}$. As $\mathcal{T}$ is nonempty, we are guaranteed to find $T \in \mathcal{T}$ finally. Next, construct the set Π_T as follows. Check each pattern over Γ, using only constants used in T, with at most n_h nodes, where h is the height of T. If the pattern is satisfied in T, add it to Π_T. By the claim above this gives all patterns satisfied in T (up to equivalence). Let $\Pi_{\mathcal{T}} = \{\pi \in \Pi_T \mid \pi \in \text{Th}(\mathcal{T})\}$. A ground max-description is obtained by merging all the patterns from $\Pi_{\mathcal{T}}$ at the root. $\square$

Remark. Note that the proof would also work for patterns with variables, provided that each variable is used only once. That is because the subpatterns would still be completely independent. Also the trees in $\mathcal{T}$ need not be ground for the proposition to hold.

To characterize max-descriptions and to find their canonical forms we need a notion of homomorphisms between XML trees $T_1 = \langle U_1, \downarrow, \ell ab_1, r_1, \rho_1\rangle$ and $T_2 = \langle U_2, \downarrow, \ell ab_2, r_2, \rho_2\rangle$. Let C_i and V_i be the sets of data values and variables used as attribute values in T_i, for $i = 1, 2$. A *homomorphism* h between T_1 and T_2 is a function that maps U_1 into U_2 and V_1 into $C_2 \cup V_2$ such that:

1. $h(r_1) = r_2$;
2. $\ell ab_1(s) = \ell ab_2(h(s))$ for all nodes s;
3. if $s \downarrow s'$ in T_1 then $h(s) \downarrow h(s')$ in T_2;
4. $h(\rho_1(s)) = \rho_2(h(s))$ for all nodes s (as usual, h extends to the constant set C_1 as the identity).

As patterns provide a syntax for trees, each pattern π can be viewed as a tree T_π (the tree associated with $\ell(\bar{\alpha})[\pi_1, \ldots, \pi_n]$ has an ℓ-labeled root with attributes $\bar{\alpha}$ and subtrees $T_{\pi_1}, \ldots, T_{\pi_n}$, where T_ε is the empty tree). We can thus talk about homomorphisms between patterns: e.g., we write $h : \pi \to \pi'$ instead of the more formal $h : T_\pi \to T_{\pi'}$.

An immediate observation is that the semantics of tree pattern satisfaction can be stated in terms of homomorphisms, just as the semantics of naïve tables is stated in terms of their homomorphisms into complete databases:

Lemma 3.4 $T \models \pi$ *iff there exists a homomorphism from π to T.*

In particular, Th$(\mathcal{T})$ is the set of patterns π such that there is a homomorphism from π to every tree $T \in \mathcal{T}$. Then we have the following characterization of max-descriptions, which confirms the intuition of max-descriptions as maximum extractable information from the certain knowledge we have about $\mathcal{T}$.

Theorem 3.5 *A pattern π is a max-description of $\mathcal{T}$ iff it belongs to* $\mathrm{Th}(\mathcal{T})$ *and every pattern in* $\mathrm{Th}(\mathcal{T})$ *has a homomorphism into it.*

Let $\mathrm{core}(\pi)$ denote the core of π. Technically speaking, we define the core of the tree T_π, but we can always view it as a pattern. Theorem 3.5 implies that every two max-descriptions π and π' of $\mathcal{T}$ are *homomorphically equivalent*, as there homomorphisms $\pi \to \pi'$ and $\pi' \to \pi$. It is well-known that homomorphically equivalent structures have isomorphic cores [26]; in particular, all max-descriptions have the same core. The same applies to ground max-descriptions and leads to the following.

Definition 3.6 *The core of all max-descriptions of $\mathcal{T}$ is called the* core-description of $\mathcal{T}$ *and denoted by $\Box\mathcal{T}$. The core of all ground max-descriptions of $\mathcal{T}$ is called the* ground core-description of $\mathcal{T}$ *and is denoted by $\Box_{\mathrm{gr}}\mathcal{T}$.*

The following (almost folklore) proposition summarizes the basic results about core-descriptions.

Proposition 3.7 *Let $\mathcal{T}$ be a set of XML trees.*

- *$\Box\mathcal{T}$ is a max-description of $\mathcal{T}$.*
- *$\Box_{\mathrm{gr}}\mathcal{T} = \mathrm{core}(\mathrm{ground}(\Box\mathcal{T}))$.*
- *Checking whether $\pi' = \mathrm{core}(\pi)$ is in* DP*; moreover, there is a fixed pattern π_0 such that checking whether $\pi_0 = \mathrm{core}(\pi)$ is* NP*-complete.*

Complexity What is the complexity of finding a max-description, and how big can it be? In particular, how big can the smallest of them, i.e., the core-description, be? We now give such bounds for finite sets $\mathcal{T}$. Let $|T|$ be the number of nodes in T, and let $\|\mathcal{T}\|$ be $\sum_{T\in\mathcal{T}} |T|$. By $|\mathcal{T}|$ we mean the number of trees in $\mathcal{T}$.

We can compute a max-description of $\mathcal{T}$ in exponential time in general, and in polynomial time if the number of trees in $\mathcal{T}$ is fixed; furthermore, we cannot beat the exponential bound in general. More precisely, we have the following.

Theorem 3.8 *Let $\mathcal{T}$ be a finite set of XML trees.*

1. *A max-description of $\mathcal{T}$ is computable in time polynomial in $\|\mathcal{T}\|$ and exponential in $|\mathcal{T}|$.*
2. *If $|\mathcal{T}| = n$, then $|\Box\mathcal{T}| \le \left(\frac{\|\mathcal{T}\|}{n}\right)^n$.*
3. *There is a family $\{\mathcal{T}_n\}_{n>0}$ of sets of trees so that $|\mathcal{T}_n| = n$ and $|T| > 1$ for each $T \in \mathcal{T}_n$, such that $|\Box\mathcal{T}_n| > \left(\frac{\|\mathcal{T}_n\|}{1.5n}\right)^n$.*

Proof. *1.* For simplicity assume that every node has a single attribute; extension to the general case is straightforward.

Let $\mathcal{T} = \{T_1, \dots, T_n\}$. The pattern we are going to construct is a sort of consistent product of $T_1, \dots, T_n$. Proceed as follows. For the root, labeled with r, take the sequence $(\varepsilon, \dots, \varepsilon)$, i.e., the sequence of roots of $T_1, \dots, T_n$. Then iteratively, under every node $(v_1, \dots, v_n)$ put a new node $(w_1, \dots, w_n)$ for every sequence $w_1, \dots, w_n$ such that w_i is a child of v_i in T_i, and all w_i's are labeled with the same letter σ. Label the node σ and put a fresh variable in the attribute slot. If for some node $\bar v = (v_1, \dots, v_n)$ some v_i is a leaf, $\bar v$ is a leaf as well.

Define $A(v_1, \dots, v_n) = (a_1, \dots, a_n)$ where a_i is the data value attached to the node v_i. For every node $\bar v$ such that $A(\bar v) = (c, \dots, c)$ for a constant c, replace the variable in $\bar v$ with the constant c. For the remaining nodes, whenever $A(\bar v) = A(\bar w)$, replace the variable in $\bar w$ with the variable in $\bar v$.

The constructed formula is clearly satisfied in every T_i. Let us see that it is indeed a max-description. Suppose that π' is satisfied in every T_i. Let h_i be a homomorphism from π' to T_i. It is easy to see that $h = (h_1, \dots, h_n)$ is a homomorphism from π' to π.

The complexity of the algorithm is polynomial in the size of the output pattern, which is bounded by $\prod_{i=1}^n |T_i| \le \left(\frac{\|\mathcal{T}\|}{n}\right)^n$ (by the inequality between the arithmetic and the geometric means for non-negative numbers).

2. The bound on size of core-descriptions follows from the above.

3. For every $n \ge 1$ we define a set $\mathcal{T}_n$ of n trees, each with 3 nodes, such that the core-description of $\mathcal{T}_n$ has the size $2^n + 1$. Let D be the DTD

$$\begin{aligned} &r \to c\,c \\ &c:\ \ @a_1^0, \dots, @a_n^0, @a_1^1, \dots, @a_n^1,\,. \end{aligned}$$

Every tree conforming to D has only three nodes: root ε labeled with r, and two children labeled with c. To distinguish the two c nodes in a tree conforming to D we always call one of them the 0-node and the other one the 1-node.

For $i = 1, \dots, n$ let $T_i \models D$ be the tree whose all attributes store $\flat$, safe for a_i^0 in the 0-node and a_i^1 in the 1-node, that store $\sharp$. For $v = v_1 \dots v_n \in \{0,1\}^n$ let

$$\varphi_v = r[c(x_1^0, \dots, x_n^0, x_1^1, \dots, x_n^1)] \wedge \bigwedge_{i=1}^{n} x_i^{v_i} = \flat\,.$$

Rename the variables so that none is used in two different formulae. Let π be the pattern obtained by taking $\bigwedge\{\varphi_v \colon v \in \{0,1\}^n\}$. Clearly, π holds in every T_i: indeed, φ_v is witnessed by the 0-node if $v_i = 0$, and by the 1-node if $v_i = 1$.

Let us see that π it is a max-description. Take any π', β satisfied in all T_i. Observe that π' has to be of the form $r[c(\bar x_1), c(\bar x_2), \dots, c(\bar x_k)]$. Let h_i be a homomorphism mapping π' to T_i. We need to find a homomorphism mapping π' to π. Fix a c-node u. Recall that $h_i(u)$ is either 0 or 1 (ie the 0-node or the 1-node in T_i). Define $g(u) = h_1(u)h_2(u)\dots h_n(u) \in \{0,1\}^n$. In π every c-node corresponds to a single formula φ_v. Let $h(u)$ be the node of π corresponding to the formula $\varphi_{g(u)}$. Checking that h is a homomorphism is straightforward.

It is easy to see that π does not have nontrivial endomorphisms. Hence, it is a core. $\square$

Remarks. One can easily encode an n-tuple of attributes as a path of n nodes each with a single attribute. Since in our examples the attributes store one of two values, $\flat$ or $\sharp$, each attribute can be replaced with a child storing the value of the attribute in the label. Thus, we can get the exponential lower bound even for trees with no data.

Notice also that the upper bound from the proposition above carries over to the relational case easily. The examples for the lower bound are not hierarchical and can be turned into relational examples as well.

Relations as trees We now demonstrate that our definition fully agrees with the relational one when we naturally encode relations as XML documents. Assume we have a naïve table (or a complete relation) R with n tuples $t_1, \ldots, t_n$. It can be encoded as a tree T_R of depth 1 with n children of the root, where the ith child has, as its attribute values, the values in the tuple t_i.

Proposition 3.9 *For a naïve table R, the max-descriptions of T_R are precisely the tree representations of the max-descriptions of* $\mathrm{Rep}(R)$.

Summary We adapted the relational approach of the previous section, and defined max-descriptions as analogs of certain answers. We looked at analogs of CQs and ground CQs, patterns and ground patterns, as our languages. There could be many different max-descriptions, but they all share the same analog of "minimization" – the core.

4. CERTAIN ANSWERS FOR TREE QUERIES

Now that we understand the notion of a proper representation of certain information contained in a set of trees, we apply this notion to answering XML queries that can produce trees.

Suppose we have a query Q that takes trees as inputs and returns trees. Our goal is to define certain answers to such a query over a set $\mathcal{T}$ of trees. Following the previous section, we define such certain answers as max-descriptions of the set $Q(\mathcal{T}) = \{Q(T) \mid T \in \mathcal{T}\}$, and core certain answers as the smallest elements of this set.

Definition 4.1 *Given a query Q and a set $\mathcal{T}$ of trees, a pattern π is a* certain answer to Q over $\mathcal{T}$ *if it is a max-description of $Q(\mathcal{T})$. We define* core-certain answer *as* $\square Q(\mathcal{T})$.

Of course we can also introduce the ground versions of these: a ground max-description of $Q(\mathcal{T})$ is a ground certain answer, and its core is the ground core certain answer $\square_{\mathrm{gr}} Q(\mathcal{T})$. Note that while there could be many (ground) certain answers, core versions are unique (up to isomorphism). From the results of the previous section we obtain :

Corollary 4.2 *Let π be a certain answer to Q over $\mathcal{T}$. Then:*

1. $\mathrm{ground}(\pi)$ *is a ground certain answer to Q.*
2. $\mathrm{core}(\pi) = \square Q(\mathcal{T})$.
3. $\mathrm{core}(\mathrm{ground}(\pi)) = \square_{\mathrm{gr}} Q(\mathcal{T})$.

In particular, it will suffice for most tasks to find a single certain answer π, as from it we can construct other types of certain answers by taking the core, and dropping subpatterns with variables.

A query language returning trees As in many functional and database query languages [13, 35, 36], including FLWR (for-let-where-return) expressions of XQuery [34], the key construct of the language we use is a comprehension. It is of the form

$$\mathsf{for}\ \pi(\bar{x})\ \mathsf{return}\ q(\bar{x})$$

where $\pi(\bar{x})$ is a pattern and $q(\bar{x})$ defines a forest (we shall formally define forest queries below). Given an input tree T, for each tuple $\bar{a}$ such that $T \models \pi(\bar{a})$, we construct the forest $q(\bar{a})$ and take the union of all such forests as the answer to the query (to make a forest into a tree, we just put a common root above it). Forest expressions can involve subqueries as well, for example $\mathsf{for}\ \pi(\bar{x})\ \mathsf{return}\ \big(\mathsf{for}\ \pi'(\bar{x}, \bar{z})\ \mathsf{return}\ q'(\bar{x}, \bar{z})\big)$. Then, for each $\pi(\bar{a})$ true in an input T, we construct the forest $\mathsf{for}\ \pi'(\bar{a}, \bar{z})\ \mathsf{return}\ q'(\bar{a}, \bar{z})$ and take the union of these as the answer to the query over T.

In the language we allow a more general class of patterns $\Pi^*(\mathsf{Const}, \mathsf{Var})$ that also use descendant and wildcard, defined by:

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

where $\_$ is the wildcard symbol and $//$ stands for the descendant. The definition of the semantics is extended by two clauses: (1) $(T, s, v) \models \_(\bar{\alpha})$ if $\rho(s) = v(\alpha)$, and (2) $(T, s, v) \models //\pi$ if there is a descendant s' of s so that $(T, s', v) \models \pi$.

Definition 4.3 *A $\mathcal{TQL}$ query Q is an expression of the form $r[q]$, where q is a forest query without free variables. The syntax and the semantics of forest queries are given in Figure 1. For a tree T and a query $Q = r[q]$, we define $Q(T)$ as the tree $r[[\![q]\!]_T]$, i.e., the forest $[\![q]\!]_T$ under root r.*

Forest queries include the empty forest (ε), trees ($\ell(\bar{a}, \bar{x}')[q']$), unions of forests ($q', q''$), and comprehensions ($\mathsf{for}\ \pi\ \mathsf{return}\ q'$). By union $F \cup F'$ of forests we mean the forest obtained by putting together F and F' (i.e., it may contain more than one copy of a tree). By $\ell(\bar{a})[F]$ we mean a tree obtained by attaching a common ℓ-labeled root with values $\bar{a}$ on top of forest F. Note that $\mathcal{TQL}$ queries can be applied not only to ground trees, but also to trees over $\mathsf{Const} \cup \mathsf{Var}$, providing an analog of naïve evaluation of queries. Recall that $T \models_v \pi$ means that v is the valuation of variables of π that witnesses the match of π in T. It is used to provide a proper analog of naïve evaluation.

Syntax: $q(\bar{x}) \quad ::= \quad \varepsilon$
$\quad\quad\quad\quad\quad\quad | \quad \ell(\bar{a}, \bar{x}')[q'(\bar{x}'')]$
$\quad\quad\quad\quad\quad\quad | \quad q'(\bar{x}'), q''(\bar{x}'')$
$\quad\quad\quad\quad\quad\quad | \quad \mathsf{for}\ \pi(\bar{a}, \bar{x}, \bar{y})\ \mathsf{return}\ q'(\bar{x}, \bar{y})$

where $\pi \in \Pi^*(\mathsf{Const}, \mathsf{Var})$, $\bar{x}'$ and $\bar{x}''$ are subtuples of $\bar{x}$, and $\bar{y}$ is disjoint from $\bar{x}$.

Semantics: $[\![q(\bar{x})]\!]_{T,v}$
for a tree T, and a valuation $v : \bar{x} \to \mathsf{Const} \cup \mathsf{Var}$

$$\begin{array}{rcl} [\![\varepsilon]\!]_{T,v} & = & \varepsilon \\ [\![\ell(\bar{a}, \bar{x}')[q'(\bar{x}'')]]\!]_{T,v} & = & \ell(\bar{a}, v(\bar{x}'))\big[[\![q']\!]_{T,v}\big] \\ [\![q'(\bar{x}), q''(\bar{x}'')]\!]_{T,v} & = & [\![q']\!]_{T,v} \cup [\![q'']\!]_{T,v} \end{array}$$

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

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

Figure 1: Syntax and semantics of forest queries

Example 4.4 Consider the tree T_{doc} given in Figure 2(a). This tree represents a set of books, with a title attribute and one or more author subelements, each with name and affiliation elements, and their attributes. We want to output a tree T_{bib} in Figure 2(b): it restructures the input by combining, for each author, his/her works with their title and affiliation attributes. The $\mathcal{TQL}$ query for this transformation is $Q = r[q]$, where q is the following forest query:

$$\begin{array}{l} \mathsf{for}\ r//name(x)\ \mathsf{return} \\ \quad person(x)\big[\mathsf{for}\ r/book(y)/author[name(x), aff(z)] \\ \quad\quad\quad\quad\quad \mathsf{return}\ work(y, z)\big] \end{array}$$

We used XPath-like notation / and // above to simplify the syntax of patterns. □

Proposition 4.5 *For each $\mathcal{TQL}$ query Q and a tree T, computing $Q(T)$ can be done in time polynomial in the size of T.*

When we try to compute certain answers to a query Q over a set $\mathcal{T}$ of trees, $\mathcal{T}$ is usually infinite. Hence, we need a finite representation of it. Recall also that finiteness guarantees existence of max-descriptions. Such a representation comes in the form of a *basis* $\mathcal{B}$ of $\mathcal{T}$. Recall that each pattern can be viewed as a tree, so when we write $\mathcal{B} \subseteq \mathcal{T}$, we mean that $T_\pi \in \mathcal{T}$ for every $\pi \in \mathcal{B}$.

Definition 4.6 *A* basis *for a set of trees $\mathcal{T}$ is a set of patterns $\mathcal{B} \subseteq \mathcal{T}$ such that $\mathcal{T} \subseteq \bigcup\{\mathrm{Mod}(\pi) \mid \pi \in \mathcal{B}\}$.*

Bases are preserved by $\mathcal{TQL}$ queries, which can be used for computing max-descriptions.

Lemma 4.7 *Let $\mathcal{B}$ be a basis of $\mathcal{T}$ and let Q be a $\mathcal{TQL}$ query. Then*

1. *the sets of max-descriptions of $\mathcal{T}$ and $\mathcal{B}$ coincide; and*
2. *$Q(\mathcal{B})$ is a basis of $Q(\mathcal{T})$.*

As a direct corollary, we obtain the following result.

Theorem 4.8 *If $\mathcal{T}$ is a set of trees, $\mathcal{B}$ is its basis, and Q is a $\mathcal{TQL}$ query, then certain answers to Q over $\mathcal{T}$ and over $\mathcal{B}$ coincide. In particular, $\Box Q(\mathcal{T}) = \Box Q(\mathcal{B})$.*

This gives a general approach to computing certain answers:

1. Compute a (small) basis $\mathcal{B}$ of $\mathcal{T}$;
2. Compute a max-description π of $Q(\mathcal{B})$.

Such a max-description π is guaranteed to be a certain answer. If core or ground certain answers need to be found, one can compute core(π), or ground(π), or core(ground(π)), according to Corollary 4.2.

5. APPLICATIONS

The previous section showed that the problem of computing certain answers for $\mathcal{TQL}$ queries over collections $\mathcal{T}$ of trees is reduced to the problem of computing small bases of $\mathcal{T}$. We now show how to find such bases for sets $\mathcal{T}$ that arise in applications.

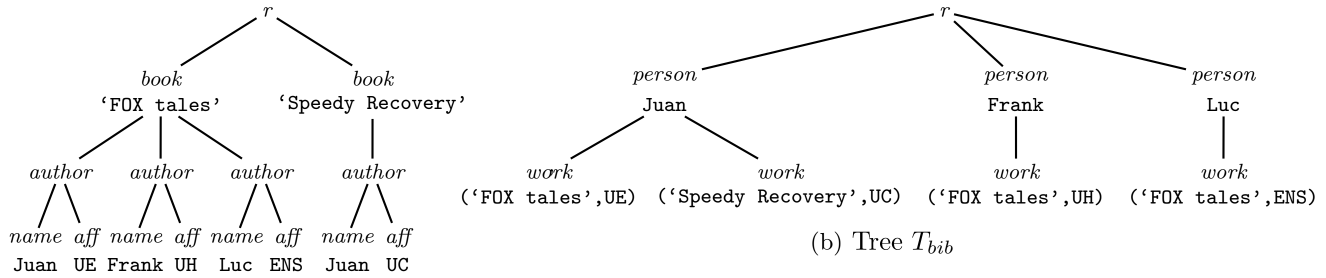
Languages considered for XML data exchange and incomplete information in the past were limited so as to apply the usual relational notion of certain answers. For example, [6, 9, 5, 4] dealt with the set of *flat* queries in $\mathcal{TQL}$, i.e., queries of the form $r[\mathsf{for}\ \pi(\bar{x})\ \mathsf{return}\ \ell(\bar{x}')[\varepsilon]]$. We now show how to extend query answering to a proper class of XML queries, using the machinery of the previous two sections.

5.1 XML with incomplete information

A standard approach to incomplete XML documents is to model them as various types of patterns, with nulls as possible values of attributes [3, 9]. For example, in [9], incomplete XML documents were simply patterns in $\Pi^*(\mathsf{Const}, \mathsf{Var})$, under the usual semantics that such a pattern π represents the set $Rep(\pi)$ of ground trees in $\mathrm{Mod}(\pi)$. The patterns in [9] also considered horizontal navigation, which we do not look at for now.

The language studied in [9] was that of flat $\mathcal{TQL}$ queries. It was shown that every such query can be evaluated in coNP, and that for patterns involving descendant it is very easy to achieve coNP-hardness. Thus, to find tractable classes, for which an analog of naïve evaluation provides certain answers, [9] concentrated on patterns without descendant. It showed that if the horizontal ordering is completely specified, then the naive evaluation computes certain answers in polynomial time. The question of what happens without the horizontal ordering was left open. We now show that the answer easily follows from our techniques, not only for flat, but for all $\mathcal{TQL}$ queries.

Given a pattern $\pi \in \Pi(\mathsf{Const}, \mathsf{Var})$ and a $\mathcal{TQL}$ query Q, a certain answer to Q over π is a max-description of $Q(Rep(\pi))$. Note that π is a basis of $\mathrm{Mod}(\pi)$, and hence, by Theorem 4.8, $Q(T_\pi)$ is a max-description of $Q(\mathrm{Mod}(\pi))$. It is easy show that a max-description



(a) Tree T_{doc}

(b) Tree T_{bib}

Figure 2: A $\mathcal{TQL}$ example: $T_{bib} = Q(T_{doc})$ for query Q in Example 4.4

of $Q(\text{Mod}(\pi))$ is also a max-description of $Q(Rep(\pi))$, which gives us the following.

Theorem 5.1 *If $\pi \in \Pi(\mathsf{Const}, \mathsf{Var})$ and Q is a $\mathcal{TQL}$ query, then $Q(T_\pi)$ is a certain answer to Q over $Rep(\pi)$.*

In particular, naïve evaluation works for $\mathcal{TQL}$ queries over incomplete documents specified as $\Pi(\mathsf{Const}, \mathsf{Var})$ patterns. This happens not only for flat queries (which answers an open question from [9]), but for all $\mathcal{TQL}$ queries.

5.2 XML data exchange

Recall the usual setting of a data exchange problem (cf. [8, 30]): we have source and target schemas, and a *schema mapping* $\mathcal{M}$, which shows how source and target instances are related. Semantically, such a mapping $\mathcal{M}$ is a binary relation that consists of pairs (S, T), where S is a source instance and T is a target instance. For a given source S, a target T that is related to it by $\mathcal{M}$ is called a *solution* (wrt $\mathcal{M}$). Given a query Q over the target schema, and source data S, the standard notion of query answering is *certain answers* for the collection $Q(T)$, as T ranges over solutions. In other words, one looks for answers that would not depend on a particular solution that is materialized.

This problem has been extensively studied in the relational case. For XML, pattern-based schema mappings were proposed in [6] which also provided a full classification of the complexity of *flat* $\mathcal{TQL}$ queries; followups [4, 5] were also restricted to flat queries. We now extend those results to $\mathcal{TQL}$ queries returning trees.

Schema mappings Mappings $\mathcal{M} = \langle D_s, D_t, \Sigma \rangle$ consist of a source schema D_s, a target schema D_t, and a set Σ of *source-to-target dependencies (stds)* that relate source and target instances. Given an source of T of schema D_s, a target T' of schema D_t is called a *solution* for T under $\mathcal{M}$ if (T, T') satisfy all the stds in Σ. We let $\mathcal{M}(T)$ stand for the set of all solutions for T.

We now give precise definitions of schemas and stds. As in [6, 5], schemas will be given by DTDs, and stds will be based on tree patterns.

DTDs A DTD over a labeling alphabet Γ containing a special symbol r for the root is a pair $(\delta_{lab}, \delta_{attr})$ where δ_{lab} maps Γ to regular expressions over $\Gamma - \{r\}$, and $\delta_a : \Gamma \mapsto \mathbb{N}$. A tree T with a root labeled r conforms to $(\delta_{lab}, \delta_{attr})$ if each ℓ-labeled node has $\delta_{attr}(\ell)$ attributes, and the labels of its children can be ordered to form a string in the language $\delta_{lab}(\ell)$. This definition takes into account our notational simplification about attributes (which are identified not by names but by their element type and position) and the fact that we use unordered trees (which can be done in data exchange without loss of generality, as [6] showed).

Source-to-target dependencies (stds) Following [21] (and also [5]), we use Skolem functions in stds, to make them more expressive and achieve composability. Fix an infinite set Fn of function symbols, each with an associated arity. The set of terms over Fn is defined inductively: each $x \in \mathsf{Var}$ is a term of arity 0; and if $t_1, \ldots, t_k$ are terms and f is a k-ary function symbol in Fn, then $f(t_1, \ldots, t_k)$ is a k-ary term. We write $t(\bar{x})$ to indicate the list of variables used in t.

An std is an expression of the form:

$$\pi(\bar{x}, \bar{y}), \theta(\bar{x}, \bar{y}) \;\longrightarrow\; \pi'(\bar{t}(\bar{x})), \theta'(\bar{t}'(\bar{x})), \qquad (1)$$

where π, π' are patterns from $\Pi^*(\mathsf{Const}, \mathsf{Var})$; tuples $\bar{x}$ and $\bar{y}$ contain variables, while $\bar{t}$, $\bar{t}'$ are sequences of terms over Fn, and θ, θ' are conjunctions of equalities among $\bar{x}, \bar{y}$, and terms in $\bar{t}'(\bar{x})$.

For example, assume that we start with tree T_{doc} in Fig. 2, and we want to produce a tree that for each author, would have his/her name and date of birth, as well as books. A natural std for this is

$$r/book(x)//name(z) \longrightarrow r/person(z, f(z))/work(x),$$

where f is a unary Skolem function whose interpretation is the date of birth of writer z. Skolem functions make it possible to specify that attributes such as date of birth depend only on the name z and not the book x.

To define the semantics, we need a valuation v on Fn that associates with each function symbol of arity k a function $f : (\mathsf{Const} \cup \mathsf{Var})^k \to \mathsf{Const} \cup \mathsf{Var}$. Such a valuation naturally extends to terms over Fn. Then a pair of trees (T, T') satisfies the std (1) under a valuation v of Skolem functions if whenever $T \models \pi(\bar{a}, \bar{b})$ and all the qualities in $\theta(\bar{a}, \bar{b})$ hold, T' satisfies $\pi'(\bar{t}(\bar{a}))$ and all the equalities in θ' hold, when all the terms are interpreted by v. We say that (T, T') satisfies a set of stds Σ if there is a valuation of Skolem functions such that (T, T') satisfies every std in Σ under this evaluation.

Classes of schema mappings We denote the class of schema mappings just defined by $\mathrm{SM}^*(\mathsf{Fn})$. It was shown in [6] that answering even very simple queries under $\mathrm{SM}^*(\mathsf{Fn})$ mappings could be intractable. To avoid intractability, one typically imposes restrictions on DTDs. Following [3, 4, 6], we use the class of *nested-relational DTDs*. In them, all regular expressions are of the form $\hat{\ell}_1 \ldots \hat{\ell}_m$, where all ℓ_i's are distinct and $\hat{\ell}_i$ is either ℓ_i or ℓ_i^* or ℓ_i^+ or $\ell_i? = \ell_i|\varepsilon$. Furthermore, these DTDs are not recursive. They generalize nested relations, and it has been shown experimentally that a large fraction of real-life DTDs is of this form [7]. We write $\mathrm{SM}_{\mathrm{nr}}(\mathsf{Fn})$ for the class of schema mappings where all DTDs are nested-relational.

Computing certain answers Given a mapping $\mathcal{M} \in \mathrm{SM}_{\mathrm{nr}}(\mathsf{Fn})$, a source tree T, and a query Q over the target trees, the standard semantics of query answering in data exchange [19, 8, 30] is to compute a *certain answer to Q over all solutions for T*, i.e., over $\mathcal{M}(T)$.

Thus, if Q is a $\mathcal{TQL}$ query, we need to compute a max-description of $Q(\mathcal{M}(T))$, which will provide a certain answer. Theorem 5.2 tells us that for this, we need to (1) compute a basis of $\mathcal{M}(T)$, and (2) evaluate Q over that basis. The following result shows that we can always compute a finite basis in data exchange:

Theorem 5.2 *Given a mapping $\mathcal{M} \in \mathrm{SM}^*_{\mathrm{nr}}(\mathsf{Fn})$ and a source tree T, one can compute a basis $\mathcal{B}$ for $\mathcal{M}(T)$ in time single exponential in the size of T. The size of $\mathcal{B}$ can be exponential, but the size of each pattern $\pi \in \mathcal{B}$ is polynomial in T.*

Theorem 5.2 and the recipe sketched out in Section 4 give us an algorithm for computing certain answers. We now analyze its complexity for $\mathrm{SM}^*_{\mathrm{nr}}(\mathsf{Fn})$ mappings $\mathcal{M}$ and $\mathcal{TQL}$ queries Q. We first look at a functional (rather than decision) problem:

> CERTAINANSWER$(\mathcal{M}, Q)$: Given a source tree T, compute a certain answer to Q over $\mathcal{M}(T)$.

Note that we are talking about data complexity. Even in this case, the size of the output can be rather large.

Proposition 5.3 *There is a mapping $\mathcal{M} \in \mathrm{SM}^*_{\mathrm{nr}}(\mathsf{Fn})$ and a $\mathcal{TQL}$ query Q such that there exists a family of source trees $\{T_n\}_{n>0}$ satisfying $|T_n| = O(n)$ and $|\Box Q(\mathcal{M}(T_n))| = \Omega(2^n)$.*

If we look at the combined complexity, i.e., the problem CERTAINANSWER$(\mathcal{M})$ in which both T and Q are inputs, the complexity jumps by another exponent, which is also unavoidable.

Proposition 5.4 *The problem* CERTAINANSWER$(\mathcal{M})$ *is in* 2EXPTIME. *Moreover, there is a mapping $\mathcal{M} \in \mathrm{SM}^*_{\mathrm{nr}}(\mathsf{Fn})$, and families of source trees $\{T_n\}_{n>0}$ and queries $\{Q_n\}_{n>0}$ of size polynomial in n such $|\Box Q_n(\mathcal{M}(T_n))| = \Omega(2^{2^n})$.*

Before we outline restrictions that avoid high complexity of query answering, we briefly look at certain answers as a *decision problem*:

> VERIFY_CA$(\mathcal{M}, Q)$ Given a source tree T, and a pattern $\pi \in \Pi(\mathsf{Const}, \mathsf{Var})$, is π satisfied by $\Box Q(\mathcal{M}(T))$?

Equivalently, we can ask if π is satisfied by an arbitrary max-description π' of $Q(\mathcal{M}(T))$, since $\mathrm{Mod}(\pi') = \mathrm{Mod}(\Box Q(\mathcal{M}(T)))$. The problem is clearly decidable, but we can lower the straightforward upper bound of Proposition 5.4 by applying Theorem 5.2 directly instead. As usual, when we say that the problem with parameters such as $\mathcal{M}$ and Q is complete for a class $\mathcal{C}$, we mean that it is always in $\mathcal{C}$ and could be hard for $\mathcal{C}$ for some instantiation of the parameters.

Proposition 5.5 *The problem* VERIFY_CA$(\mathcal{M}, Q)$ *is Π_2^p-complete, and* CONP*-complete if the number of variables in the input pattern is fixed.*

6. TRACTABLE XML DATA EXCHANGE

The machinery we developed here allowed us to prove computability of certain answers for a large class of mappings and a proper XML query language. The complexity bounds of the previous section, however, are rather high. The natural question is then whether we can lower them.

The answer is that we can, and what we need is a simple restriction: instead of $\Pi^*(\mathsf{Const}, \mathsf{Var})$ patterns in the definition of stds (1), we use $\Pi(\mathsf{Const}, \mathsf{Var})$ patterns. We call the resulting class of schema mappings $\mathrm{SM}_{\mathrm{nr}}(\mathsf{Fn})$. In other words, we forbid descendant and wildcard.

We claim that this is a good class of XML schema mappings. It subsumes nested relations (and non-relational extensions of data exchange systems such as those in [31, 37]). And all the key problems related to this class are tractable.

The tractability of several problems relies on the fact that for these mappings, one can compute *singleton bases* of $\mathcal{M}(T)$. These are essentially analogs of universal solutions used heavily in relational data exchange [19].

Theorem 6.1 *Given a mapping $\mathcal{M} \in \mathrm{SM}_{\mathrm{nr}}(\mathsf{Fn})$ and a source tree T, there is a single-tree basis of $\mathcal{M}(T)$, which can be constructed in time polynomial in the size of T.*

This immediately gives us tractable algorithms for query answering and for the problem of materializing a target instance:

> SOLUTION$(\mathcal{M})$: Given a source tree T, compute a solution $T' \in \mathcal{M}(T)$.

Since every element of a basis of a set, when viewed as a tree, belongs to the set, we obtain the following.

Corollary 6.2 *Let $\mathcal{M}$ be a mapping in* $\mathrm{SM_{nr}}(\mathsf{Fn})$ *and* Q *a* $\mathcal{TQL}$ *query. Then both* $\mathrm{CertainAnswer}(\mathcal{M}, Q)$ *and* $\mathrm{Solution}(\mathcal{M})$ *are solvable in polynomial time.*

We also note that the algorithms are at most single-exponential in the size of the mapping, matching the relational case [19].

A standard static analysis problem looked at in the context of XML data is *consistency*: Given a mapping $\mathcal{M}$, does there exist an XML tree T such that $\mathcal{M}(T) \neq \emptyset$? In other words, does the mapping make sense? In general, this problem is ExpTime-complete [6]. However, extending results of [6] which analyzed this problem in restricted settings (but without Skolem functions) we obtain the following.

Proposition 6.3 *For mappings from* $\mathrm{SM_{nr}}(\mathsf{Fn})$*, the consistency problem is solvable in polynomial time.*

We next look at the issue of *composability* [21]. Each mapping $\mathcal{M}$ is semantically a binary relation $[\![\mathcal{M}]\!] = \{(T, T') \mid T' \in \mathcal{M}(T)\}$. If we have a mapping $\mathcal{M}$ between DTDs D_1 and D_2, and a mapping $\mathcal{M}'$ between DTDs D_2 and D_3, their *composition* is the mapping $[\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!] = \{(T, T') \mid \exists T_0 : T_0 \in \mathcal{M}(T) \text{ and } T' \in \mathcal{M}'(T_0)\}$. The composability question is the following: given two mappings $\mathcal{M}$ and $\mathcal{M}'$, can we construct a mapping $\mathcal{M}''$ so that $[\![\mathcal{M}'']\!] = [\![\mathcal{M}]\!] \circ [\![\mathcal{M}']\!]$?

This problem was solved for the relational case in [21] with the help of adding Skolem functions to mappings; the composition algorithm of [21] runs in ExpTime. It was also shown in [5] that the composition result extends to a class of XML mappings more restrictive than those that we consider here. But it turns out that our tractable mappings are closed under composition.

Theorem 6.4 *The class of mappings* $\mathrm{SM_{nr}}(\mathsf{Fn})$ *is closed under composition. Moreover the composition of two mappings can be constructed in* ExpTime.

We conclude by an observation about the problem $\mathrm{Verify\_CA}(\mathcal{M}, Q)$ for mappings in $\mathrm{SM_{nr}}(\mathsf{Fn})$. It turns out that we can lower the bounds of Proposition 5.5. We note that the relational analog of this problem, whether a naïve table (an analog of a pattern) is entailed by a solution in relational data exchange, is NP-complete (this is essentially the problem of containment of naïve tables [2]). We get a matching bound for our XML class, and a tractability result for fixed patterns.

Proposition 6.5 $\mathrm{Verify\_CA}(\mathcal{M}, Q)$ *is* NP*-complete for mappings from* $\mathrm{SM_{nr}}(\mathsf{Fn})$*, and in* PTime *if the number of variables in the input pattern is fixed.*

In summary, the class of mappings $\mathrm{SM_{nr}}(\mathsf{Fn})$ is closed under composition, has a tractable consistency problem, and the two key computational problems for it – materializing solutions and answering queries that may return documents – are solvable in polynomial time using the technique of computing bases for sets of trees.

7. FUTURE WORK

Now that we have a methodology for defining certain answers for proper XML queries, we would like to extend it in several ways. One direction has to do with additional power of query languages. Even though we allow trees as outputs and nesting of queries, we do not yet have any form of negation. In relational databases, adding negation requires more expressive representation mechanisms such as conditional tables [1, 27]. We would like to understand their XML analogs.

A second direction is about adding constraints, either on incomplete documents, or over target instances in data exchange. It is known that the complexity of many problems goes up with the addition of constraints [8, 14, 19, 30] in such settings. Query answering mechanisms are likely to change as well; we expect to adapt some form of chase for complex objects [18] to find certain answers over constraint-restricted classes of documents.

There is also a certain duality between the well-studied concept of a universal solution in data exchange [19] and our notion of max-description, which we would like to look at. To define universal solutions, we start with the set of all solutions and find those that have a homomorphism *into* every solution. For max-descriptions, on the other hand, we start with the theory $\mathrm{Th}(\mathcal{T})$, and then find its elements so that there is a homomorphism *from* every other element of $\mathrm{Th}(\mathcal{T})$ into them.

Acknowledgments This work started when the third author was at the University of Edinburgh. We acknowledge support by EPSRC grants E005039 and G049165, and the FET-Open Project FoX (grant agreement 233599).

8. References

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

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

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

[4] S. Amano, Claire David, L. Libkin, and F. Murlak. On the tradeoff between mapping and querying power in XML data exchange. In *ICDT'10*, to appear.

[5] S. Amano, L. Libkin, and F. Murlak. XML schema mappings. In *PODS'09*, pages 33–42.

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

[7] D. Barbosa, L. Mignet, P. Veltri. Studying the XML web: gathering statistics from an XML sample. *WWW* 9(2):187-212 (2006).

[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'09*, pages 237–246.

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

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

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

[13] P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, 1994.

[14] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS'03*, pages 260–271.

[15] D. Calvanese, G. De Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: A description logic approach. *J. Logic & Comput.*, 9(3):295–318, 1999.

[16] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Regular XPath: constraints, query containment and view-based answering for XML documents. In *LID'08*, 2008.

[17] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC'77*, pages 77–90.

[18] A. Deutsch and V. Tannen. XML queries and constraints, containment and reformulation. *TCS*, 336(1): 57–87, 2005.

[19] R. Fagin, Ph. Kolaitis, R. Miller, and L. Popa. Data exchange: Semantics and query answering. *TCS*, 336(1):89–124, 2005.

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

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

[22] S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Repairs and consistent answers for XML data with functional dependencies. In *XSym'03*, pages 238–253.

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

[24] G. Grahne. *The Problem of Incomplete Information in Relational Databases*, Springer, 1991.

[25] C. Gutiérrez, C. Hurtado, and A. Mendelzon. Foundations of semantic web databases. In *PODS'04*, pages 95–106.

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

[27] T. Imieliński and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[28] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic XML. In *VLDB'07*, pages 27–38.

[29] B. Kimelfeld and Y. Sagiv. Modeling and querying probabilistic XML data. *SIGMOD Record*, 37(4):69–77, 2008.

[30] Ph. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS'05*, pages 61–75.

[31] L. Popa, Y. Velegrakis, R. Miller, M. Hernández, and R. Fagin. Translating web data. In *VLDB'02*, pages 598–609.

[32] R. Reiter. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling*, Springer, pages 191–233, 1982.

[33] R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *J. ACM* 33(2):349-370, 1986.

[34] W3C. XQuery, W3C Recommendation. available at http://www.w3.org/TR/xquery, 2007.

[35] P. Wadler. Comprehending monads. *Math. Str. in Comp. Sci.*, 2(4):461–493, 1992.

[36] L. Wong. Kleisli, a functional query system. *J. Funct. Program.*, 10(1):19–56, 2000.

[37] C. Yu and L. Popa. Constraint-based XML query rewriting for data integration. In *SIGMOD'04*, pages 371–382.