XML Data Exchange: Consistency and Query Answering

Marcelo Arenas
University of Toronto
marenas@cs.toronto.edu

Leonid Libkin
University of Toronto
libkin@cs.toronto.edu

ABSTRACT

Data exchange is the problem of finding an instance of a target schema, given an instance of a source schema and a specification of the relationship between the source and the target. Theoretical foundations of data exchange have recently been investigated for relational data.

In this paper, we start looking into the basic properties of XML data exchange, that is, restructuring of XML documents that conform to a source DTD under a target DTD, and answering queries written over the target schema. We define XML data exchange settings in which source-to-target dependencies refer to the hierarchical structure of the data. Combining DTDs and dependencies makes some XML data exchange settings inconsistent. We investigate the consistency problem and determine its exact complexity.

We then move to query answering, and prove a dichotomy theorem that classifies data exchange settings into those over which query answering is tractable, and those over which it is coNP-complete, depending on classes of regular expressions used in DTDs. Furthermore, for all tractable cases we give polynomial-time algorithms that compute target XML documents over which queries can be answered.

1. Introduction

Data exchange is the problem of finding an instance of a target schema, given an instance of a source schema and a specification of the relationship between the source and the target. Such a target instance should correctly represent information from the source instance under the constraints imposed by the target schema, and should allow one to evaluate queries on the target instance in a way that is semantically consistent with the source data.

Data exchange is an old problem [25] that re-emerged as an active research topic recently due to the increased need for exchange of data in various formats, typically in e-business applications [5]. A system Clio for data exchange was built [19, 23] and partly incorporated into the latest release of IBM's db2 product. At about the same time, papers [10, 11] by Fagin, Kolaitis, Miller, and Popa laid the theoretical foundation of exchange of relational data, and several followup papers studied various issues in data exchange such as schema mapping composition [12] and query rewriting [6, 28].

And even though practical systems such as Clio handle non-relational data (in particular, nested relations [23]), all theoretical investigation so far has concentrated on the relational case.

Our goal is to start the investigation of basic theoretical issues of data exchange for XML documents. We illustrate XML data exchange by the following example. Suppose we have the source document shown in Figure 1 (b) conforming to the DTD shown in Figure 1 (a). This DTD says that the document consists of several $book$ elements, each having a $title$ attribute and several $author$ subelements; each author has attributes $name$ and aff(iliation).

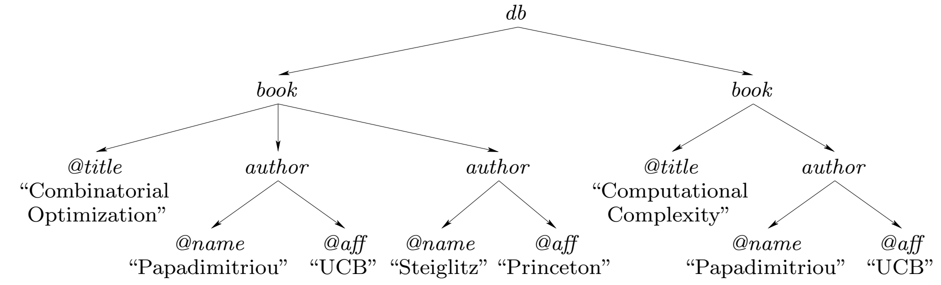
Suppose we want to restructure this document under the target schema shown in Figure 2 (a). This DTD says that a document has several $writer$ elements, each having a $name$ attribute, and several $work$ subelements with attributes $title$ and $year$. Intuitively, a restructured document should look like the XML document shown in Figure 2 (b). Note that the original document provides no data about publication year, and hence we have to invent new values for the document structured under the target schema. In data exchange terminology, these are $nulls$, denoted here by $\perp_1$ and $\perp_2$. The new document forces two of them to be the same, even though their values are not known.

Even in the relational case there could be different target databases that satisfy all the constraints of a data exchange setting [10]. So if we are given source document shown in Figure 1 (b) and a query over the new DTD, shown in Figure 2 (a), how can we answer it? If our query is, for example, $Who\ is\ the\ writer\ of\ the\ work\ named\ “Computational\ Complexity”?$, the answer is $Papadimitriou$

PODS 2005, June 13-15, 2005, Baltimore, Maryland.

```
<!ELEMENT db (book*)>
<!ELEMENT book (author*)>
  <!ATTLIST book
    title CDATA #REQUIRED>
<!ELEMENT author (EMPTY)>
  <!ATTLIST author
    name CDATA #REQUIRED
    aff CDATA #REQUIRED>
```

(a) Source DTD



(b) Source XML document

Figure 1: Source information.

regardless of a particular document that was created for the target DTD. Notice that even though the answer would be the same in every correctly constructed document that conforms to the new DTD, we can deduce this just by looking at a single document shown above. As another example, consider a query *What are the works written in 1994?*. This query cannot be answered with certainty in this scenario.

Our main goals here are the following:

- We propose a formalism for XML data exchange settings, and investigate its basic properties, and
- We study the problem of query answering in data exchange contexts, and analyze its complexity, and develop query evaluation algorithms.

Before we describe the main contributions of the paper, we recall briefly the setting of relational data exchange and query answering [10, 11]. A relational data exchange setting is a triple ($\mathbf{S}$, $\mathbf{T}$, $\Sigma_{\mathbf{ST}}$), where $\mathbf{S}$ is a source schema, $\mathbf{T}$ is a target schema, and $\Sigma_{\mathbf{ST}}$ is a set of *source-to-target dependencies*, or *STDs*, that express the relationship between $\mathbf{S}$ and $\mathbf{T}$. Sometimes a set of constraints on the target schema is also added to the setting. Such a setting gives rise to the *data exchange problem*: given an instance I over the source schema $\mathbf{S}$, find an instance J over the target schema $\mathbf{T}$ such that I together with J satisfy the STDs in $\Sigma_{\mathbf{ST}}$ (when target dependencies are used, J must also satisfy them). Such an instance J is called a *solution* for I. STDs are usually of the form

$$\psi_{\mathbf{T}}(\bar{x}, \bar{z}) \;:\!-\; \varphi_{\mathbf{S}}(\bar{x}, \bar{y}), \qquad (1)$$

where $\varphi_{\mathbf{S}}$ and $\psi_{\mathbf{T}}$ are conjunctions of atomic formulae over $\mathbf{S}$ and $\mathbf{T}$, respectively. The pair $\langle I, J\rangle$ satisfies this dependency if whenever $\varphi_{\mathbf{S}}(\bar{a}, \bar{b})$ is true in I, for some tuple $\bar{c}$, $\psi_{\mathbf{T}}(\bar{a}, \bar{c})$ is true in J.

In general, there may be many different solutions for a given source instance I, and under target constraints, there may be no solutions at all [10, 11]. If one poses a query Q over the target schema, and a source instance I is known, the usual semantics in data exchange uses *certain answers* [2, 14]: we let $\underline{certain}(Q, I)$ be the intersection of all $Q(J)$'s over all possible solutions J. A key problem in data exchange is to find a particular solution J_0 so that $\underline{certain}(Q, I)$ can be obtained by evaluating some query (a rewriting of Q) over J_0.

Some answers to this question were given in [10, 11]: e.g., if Q is a union of conjunctive queries, $\underline{certain}(Q, I)$ can be computed by evaluating Q over a special kind of solution called *canonical* that can be constructed in polynomial time. In general, however, work on query rewriting and incomplete information tells us that the complexity of finding certain answers can be intractable [1, 2].

Coming back to XML data exchange, we have to define XML data exchange settings. By analogy with the relational case, they should have source and target schemas, and source-to-target dependencies. We shall use DTDs as schemas, but it is not immediately clear what formalism to use for STDs, although intuitively they should correspond to conjunctive queries in some relational representation of XML.

This intuition gives rise to a very natural question whether we can "reduce" XML data exchange problem to relational data exchange by using some relational representation of XML documents [15] (for example, as trees with the child and next-sibling relations, as well as attribute values). The problem with this naive approach is that DTDs impose rather expressive constraints on target trees, that can talk about reachability as well as regular expressions. Thus, their expressiveness is well beyond first-order logic, and yet results on relational data exchange have only considered limited first-order constraints on the target so far.

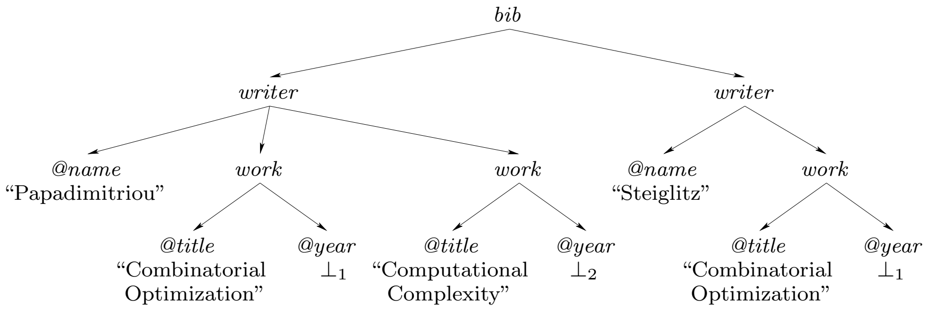
Thus, as is often the case with transferring results from relational databases to XML, we do have to reinvent most basic notions and prove new results.

We now briefly summarize our main results.

- We define data exchange settings based on STDs which show how patterns in the source tree translate into patterns in the target tree.

```
<!ELEMENT bib (writer*)>
<!ELEMENT writer (work*)>
  <!ATTLIST writer
    name CDATA #REQUIRED>
<!ELEMENT work (EMPTY)>
  <!ATTLIST work
    title CDATA #REQUIRED
    year CDATA #REQUIRED>
```

(a) Target DTD



(b) Target XML document

Figure 2: Target information.

- We want to exclude *inconsistent* data exchange settings, in which target instances cannot be constructed. We determine the exact complexity of checking consistency (EXPTIME-complete). We also find restrictions of lower complexity, as well as a practically relevant tractable class, which subsumes non-relational data exchange settings handled by Clio. Notice that the high complexity is in the size of the DTDs and STDs, and *not* XML documents.
- We study querying in the XML data exchange scenario, prove a coNP upper bound on the complexity of query evaluation, and identify a class of source-to-target dependencies, called fully-specified STDs, outside of which finding certain answers is coNP-complete. Within that class, we prove a *dichotomy theorem* which says that depending on the class of regular expressions used in DTDs, query answering is either tractable or coNP-complete. For tractable cases, which subsume nonrelational data exchange handled by Clio [23], we have algorithms for constructing target documents over which queries can be answered.

Organization Section 2 defines some basic XML concepts. In Section 3 we describe XML data exchange settings. In Section 4 we study their consistency, and in Section 5 we present a query language and some necessary restrictions for tractable query answering. In Section 6 we prove the dichotomy theorem. Because of space limitations, full proofs are omitted and only proof sketches are given.

2. Notations

We view XML documents as node-labeled unranked trees. We assume countably infinite sets El of names of element types and Att of attribute names, as well as a domain Str of possible attribute values (normally considered to be strings). Attribute names are preceded by a "@" to distinguish them from element types.

Given finite sets $E \subset El$ and $A \subset Att$, an *XML tree* T over (E, A) is a finite ordered directed tree $(N, <_{\text{child}}, <_{\text{sib}}, root)$ where N is the set of nodes, $<_{\text{child}}$ is the child relation, $<_{\text{sib}}$ is the next-sibling relation (for each node v it orders its children $v_1 <_{\text{sib}} \ldots <_{\text{sib}} v_m$), and $root$ is the root, together with

- a labeling function $\lambda_T : N \to E$ (if $\lambda_T(v) = \ell$, we say that ℓ is the *element type* of v);
- a partial function $\rho_{@a} : N \to \mathsf{Str}$ for every $@a \in A$ assigning some nodes of T values of attribute $@a$.

A DTD (Document Type Definition) over (E, A) is defined as a triple $(P, R, \underline{r})$ where

- P is a function from E to regular expressions over E defined by the grammar

  $$e \;\; ::= \;\; \varepsilon \;\big|\; \ell,\, \ell \in E \;\big|\; e|e \;\big|\; ee \;\big|\; e^*,$$

  (ε is the empty string, and $e|e$, ee and e^* stand for the union, concatenation and the Kleene star);
- $R : E \to 2^A$ associates with each element type a (possibly empty) set of attribute names; and
- $\underline{r} \in E$ is the distinguished element type of the root, which cannot be used in regular expressions $P(\ell)$ and cannot have any attributes ($R(\underline{r}) = \emptyset$).

We also use the standard shorthands e^+ for ee^* and $e?$ for $\varepsilon|e$, and we often write $\ell \to e$ instead of $P(\ell) = e$ as is common for DTDs. Furthermore, we do not consider `PCDATA` elements in XML documents since they can always be represented by attributes.

For example, for the source DTD shown in Figure 1 (a), $E = \{db,\, book,\, author\}$, $A = \{@title,\, @name,\, @aff\}$, P is given by $P(db) = book^*$ (that is, $db \to book^*$), $P(book) = author^*$, $P(author) = \varepsilon$; and $R(db) = \emptyset$, $R(book) = \{@title\}$, and $R(author) = \{@name,\, @aff\}$. Furthermore, db is the element type of the root.

An XML tree T *conforms* to $D = (P, R, \underline{r})$, denoted by $T \models D$, if:

1. for every node v in T with children $v_1, \ldots, v_m$ such that $v_1 <_{\text{sib}} \ldots <_{\text{sib}} v_m$, if $\lambda_T(v) = \ell$, then the string $\lambda_T(v_1) \ldots \lambda_T(v_m)$ is in the language defined by the regular expression $P(\ell)$;
2. for every node v in T with $\lambda_T(v) = \ell$, $\rho_{@a}(v)$ is defined iff $@a \in R(\ell)$;
3. $\lambda_T(root) = \underline{r}$.

We write $\text{SAT}(D)$ for the set of XML trees T that conform to D. It is a folklore result that checking whether $\text{SAT}(D) \neq \emptyset$ can be done in linear time. We say that a DTD D is *consistent* if for every element type ℓ in D, there exists a tree T conforming to D and having a node of type ℓ. From now on, we assume that every DTD is consistent. This can be done without loss of generality due to the following easy observation.

Lemma 2.1. *Given a DTD D with $\text{SAT}(D) \neq \emptyset$, one can construct, in polynomial time, a consistent DTD D' such that $\text{SAT}(D) = \text{SAT}(D')$.*

3. XML Data Exchange Settings

Recall [10, 11] that a relational data exchange setting is a triple $(\mathbf{S}, \mathbf{T}, \Sigma_{\mathbf{ST}})$, where $\mathbf{S}$ and $\mathbf{T}$ are source and target relational schemas, and $\Sigma_{\mathbf{ST}}$ is a family of source-to-target dependencies, that is, expressions of the form[1] $\psi_{\mathbf{T}}(\bar{x}, \bar{z}) :\!-\ \varphi_{\mathbf{S}}(\bar{x}, \bar{y})$, where $\psi_{\mathbf{T}}$ (resp., $\varphi_{\mathbf{S}}$) is a conjunction of atomic formulae over $\mathbf{T}$ (resp., $\mathbf{S}$). Instances I of $\mathbf{S}$ and J of $\mathbf{T}$ satisfy this dependency if whenever $\varphi_{\mathbf{S}}(\bar{a}, \bar{b})$ holds in I, one can find a tuple $\bar{c}$ such that $\psi_{\mathbf{T}}(\bar{a}, \bar{c})$ holds in J.

Now we need to extend this setting to XML data. Instead of source and target schemas $\mathbf{S}$ and $\mathbf{T}$, we shall use source and target DTDs $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$. But what do have in place of relational STDs?

A natural idea is to extend relational source-to-target dependencies to XML trees considered as relational structures. But one needs to add the descendant relation, which is not FO-definable from the child relation and, worse yet, make the logical formalism two-sorted in order to deal with both nodes and values. This would make the formalism rather cumbersome. Instead, we present XML source-to-target dependencies in a formalism that is much closer to XML languages such as tree patterns and XPath [4, 7].

Essentially our STDs say that if a certain pattern occurs in the source, another pattern has to occur in the target. Thus, formulae used in STDs will be very similar to those used, for example, in [4, 7, 21, 9, 27]. One difference though is that while XPath formulae select nodes from a tree, we also need to collect values of attributes that need to be assigned to nodes in the target trees. Thus, as in [9, 21], we shall use variables; in our case, they will range over possible attribute values.

Tree-pattern formulae. The basic component of our language is *attribute formulae*. Assume that _ is a wildcard symbol not included in $El \cup Att$. Given sets $E \subset El$ of element types and $A \subset Att$ of attributes, attribute formulae over (E, A) are defined by

$$\alpha \ := \ \ell \ \mid \ \ell(@a_1 = x_1, \ldots, @a_n = x_n),$$

where $\ell \in E \cup \{\_\}$ and $@a_1, \ldots, @a_n \in A$. In the second case, variables $x_1, \ldots, x_n$ are the free variables of α.

An attribute formula is evaluated in a node of a tree, and values for free variables come from Str. If T is an XML tree over (E, A) and v a node of T, then

- $(T, v) \models \_$;
- $(T, v) \models \ell$ iff $\lambda_T(v) = \ell$, for $\ell \in E$.
- If $\alpha(x_1, \ldots, x_n) = \ell(@a_1 = x_1, \ldots, @a_n = x_n)$, then $(T, v) \models \alpha(s_1, \ldots, s_n)$, where $s_1, \ldots, s_n \in \mathsf{Str}$, iff $(T, v) \models \ell$ and $\rho_{@a_j}(v) = s_j$, for every $j \in [1, n]$.

Tree-pattern formulae over (E, A) are defined by

$$\varphi \ := \ \alpha \ \mid \ \alpha[\varphi, \ldots, \varphi] \ \mid \ //\varphi,$$

where α ranges over attribute formulae over (E, A). The free variables of a tree-pattern formula φ are the free variables in all the attribute formulae that occur in it. For example, the formula $db[book(@title = x)[author(@name = y)]]$ has free variables x and y. We write $\varphi(\bar{x})$ to indicate that free variables of φ are $\bar{x}$.

We evaluate tree-pattern formulae in an XML tree. Given a tree T, a tree-pattern formula $\varphi(\bar{x})$, and a tuple $\bar{s}$ from Str, $\varphi(\bar{s})$ is true in T (written $T \models \varphi(\bar{s})$) if there is a witness node v for $\varphi(\bar{s})$. Intuitively, the witness node is the node at which the pattern is satisfied, with $\bar{s}$ being the values of attributes. Formally, we define v in T to be a witness node for $\varphi(\bar{s})$ as follows:

- v is a witness node for $\alpha(\bar{s})$, where α is an attribute formula, iff $(T, v) \models \alpha(\bar{s})$.
- v is a witness node for $\alpha(\bar{s})[\varphi_1(\bar{s}_1), \ldots, \varphi_k(\bar{s}_k)]$ iff $(T, v) \models \alpha(\bar{s})$ and there are k (not necessarily distinct) children $v_1, \ldots, v_k$ of v such that each v_i is a witness node for $\varphi_i(\bar{s}_i)$, for every $i \leq k$.
- v is a witness node for $//\varphi(\bar{s})$ if there is a descendant v' of v in T which is a witness node for $\varphi(\bar{s})$.

For example, let $\psi(x, y)$ be formula $book(@title = x)[$ $author(@name = y)]$, referring to the example from the introduction (see DTD in Figure 1 (a)). Then $\psi(s, s')$ is true iff s is a title of a book and s' is one of its authors, with the corresponding *book* element being the witness.

[1] In [10, 11], STDs are written as FO sentences but here we prefer a rule-based formalism.

Notice that every tree-pattern formula can be translated into a conjunctive query in a two-sorted logic over XML trees considered as structures in the language of $<_{\text{child}}$ and $<^*_{\text{child}}$ (descendant), being the second sort values of attributes. Thus, we are in principle in the same category of formulae for defining data exchange setting as in the relational case; however, we avoid the two-sorted formalism by using tree-pattern formulae.

Data exchange settings. We now define XML data exchange settings using tree-pattern formulae. Essentially, a data exchange setting consists of source and target DTDs, and source-to-target dependencies, which are rules of the form (1) in which both φ and ψ are tree-pattern formulae.

Definition 3.1. **(Source-to-target dependencies).** *Given finite sets $E_{\mathbf{S}}, E_{\mathbf{T}} \subset El$ of elements types and $A_{\mathbf{S}}, A_{\mathbf{T}} \subset Att$ of attributes, a source DTD $D_{\mathbf{S}}$ over $(E_{\mathbf{S}}, A_{\mathbf{S}})$ and a target DTD $D_{\mathbf{T}}$ over $(E_{\mathbf{T}}, A_{\mathbf{T}})$, a* source-to-target dependency (STD) *between $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$ is an expression of the form:*

$$\psi_{\mathbf{T}}(\bar{x}, \bar{z}) \quad :- \quad \varphi_{\mathbf{S}}(\bar{x}, \bar{y}), \tag{2}$$

where $\varphi_{\mathbf{S}}(\bar{x}, \bar{y})$ and $\psi_{\mathbf{T}}(\bar{x}, \bar{z})$ are tree-pattern formulae over $(E_{\mathbf{S}}, A_{\mathbf{S}})$ and $(E_{\mathbf{T}}, A_{\mathbf{T}})$, respectively, and tuples $\bar{y}$ and $\bar{z}$ have no variables in common.

Given XML trees T and T' conforming to $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$, respectively, we say that the pair $\langle T, T' \rangle$ satisfies this STD if whenever $T \models \varphi_{\mathbf{S}}(\bar{s}, \bar{s}')$, there is a tuple $\bar{s}''$ such that $T' \models \psi_{\mathbf{T}}(\bar{s}, \bar{s}'')$.

Definition 3.2. **(Data Exchange Setting).** *An* XML data exchange setting *is a triple ($D_{\mathbf{S}}$, $D_{\mathbf{T}}$, $\Sigma_{\mathbf{ST}}$), where $D_{\mathbf{S}}$ is a source DTD, $D_{\mathbf{T}}$ is a target DTD, and $\Sigma_{\mathbf{ST}}$ is a set of STDs between $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$.*

Definition 3.3. **(Solutions).** *Given a data exchange setting ($D_{\mathbf{S}}$, $D_{\mathbf{T}}$, $\Sigma_{\mathbf{ST}}$) and an XML tree T conforming to $D_{\mathbf{S}}$, a tree T' conforming to $D_{\mathbf{T}}$ such that $\langle T, T' \rangle$ satisfies all STDs in $\Sigma_{\mathbf{ST}}$ is called a* solution *for T.*

Referring again to the data exchange scenario from the introduction (see Figures 1 and 2), the STD that specifies how to transform *book/author* pairs into *writer/work* pairs is given by $\psi_{\mathbf{T}}(x, y, z) :- \varphi_{\mathbf{S}}(x, y)$ where $\varphi_{\mathbf{S}}(x, y)$ and $\psi_{\mathbf{T}}(x, y, z)$ are

$$db[book(@title = x)[author(@name = y)]] \;\text{ and}$$

$$bib[writer(@name = y)[work(@title = x, @year = z)]],$$

respectively. For example, we know that the source document from the introduction satisfies

$$\varphi_{\mathbf{S}}(Combinatorial\ Optimization, Papadimitriou).$$

Thus, in a solution T' for T, we would have a *writer* child of the root with the *@name* attribute *Papadimitriou*, and a *work* child with two attributes *@title* and *@year*. The value of *@title* is *Combinatorial Optimization*, but the source document provides no information about the value of the *@year* attribute. In a solution therefore one has to invent a null value (shown as $\perp_1$ in the example) for this attribute.

As in other papers on data exchange [10, 11], we assume that the domain Str of attributes is partitioned into two countably infinite sets Const and Var. The set Const contains all values that may occur in source trees, and, following data exchange terminology, we call them *constants*. Elements of Var are called *nulls*, and they are used to populate target trees.

4. Consistent Data Exchange Settings

It is known that even in the relational case some data exchange settings are *inconsistent* due to constraints on the target instance [10, 11]. DTDs, being very close in expressiveness to monadic second-order logic [26], may impose a variety of restrictions on possible solutions, sometimes making data exchange settings inconsistent. For example, consider an STD $\underline{r}[\ell_1[\ell_2(@a = x)]] :- \underline{r}$. If the target DTD is $\underline{r} \to \ell_1|\ell_2,\; \ell_1, \ell_2 \to \varepsilon$, then there is no source XML tree T for which a solution exists. In other words, no matter what the source DTD is, the data exchange setting would be inconsistent.

Thus, we call a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ *inconsistent* if no tree $T \models D_{\mathbf{S}}$ has a solution. Otherwise, the setting is *consistent*.

Obviously one should only work with consistent settings. But how hard is it to test consistency? To answer this, we study the following problem:

PROBLEM:	DATA-EXCHANGE-CONSISTENCY
INPUT:	Data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$.
QUESTION:	Is $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ consistent?

A particular case of this problem is satisfiability of tree-pattern formula which asks whether there exists a tree T that conforms to a DTD D and satisfies a tree pattern formula ψ. Indeed, this happens iff the setting $(D_{\underline{r}}, D, \{\psi :- \underline{r}\})$ is consistent, where $D_{\underline{r}}$ has only one rule $\underline{r} \to \varepsilon$. It is known that satisfiability of tree-patterns may be intractable [16] although precise complexity was not known. Results on XPath containment in the presence of DTDs [21, 27] also suggest high complexity for data exchange consistency. We now determine its exact complexity.

Theorem 4.1. *The problem* DATA-EXCHANGE-CONSISTENCY *is EXPTIME-complete.*

Proof sketch: for membership, we first show that it suffices to consider STDs in which all attribute formulae are of the form ℓ with $\ell \in E \cup \{\_\}$. Then for each STD $\psi :- \varphi$ we construct an unranked tree automaton that accepts a tree whose root has two children iff whenever the subtree rooted at the left child satisfies φ, then the subtree rooted at the right child satisfies ψ. We show that the product of all these automata can be constructed in exponential time. Then we take the product of this automaton with automata defining the DTDs; the setting is consistent iff such an automaton accepts a tree. The latter can be done in polynomial time in the size of the automaton [20].

For hardness, we use reduction from the problem of testing if a nondeterministic bottom-up tree automaton accepts every tree. This problem is known to be EXPTIME-complete [24]. □

The problem Data-Exchange-Consistency remains intractable even under some strong restrictions. Recall that a DTD D is *recursive* if there is a cycle in the graph $G(D)$ defined as $\{(\ell, \ell') \mid \ell' \text{ is mentioned in } P(\ell)\}$, and non-recursive otherwise. We define *path-pattern* formulae as restrictions of tree-pattern formulae given by

$$\varphi \;\; := \;\; \alpha \;\mid\; \alpha[\varphi] \;\mid\; //\varphi.$$

In other words, in such formulae one can talk only of one child or one descendant of a given node. They are closely related to the child-descendant fragment of XPath.

For each fixed DTD $D_{\mathbf{T}}$, we consider the restriction D-E-C$(D_{\mathbf{T}})$ of Data-Exchange-Consistency, whose input is $(D_{\mathbf{S}}, \Sigma_{\mathbf{ST}})$ with all formulae in $\Sigma_{\mathbf{ST}}$ being path-pattern formulae. The question is whether $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent.

The next proposition shows that checking consistency remains intractable even with a fixed target DTD and restricted source DTDs.

Proposition 4.2. *Fix an arbitrary nonrecursive DTD $D_{\mathbf{T}}$ that does not use the the Kleene star. Then:*

a) The problem D-E-C$(D_{\mathbf{T}})$ for non-recursive source DTDs $D_{\mathbf{S}}$ that do not use the Kleene star is PSPACE-complete.

b) The problem D-E-C$(D_{\mathbf{T}})$ for non-recursive source DTDs $D_{\mathbf{S}}$ in which all regular expressions are of the form $\ell \to \ell_1|\ldots|\ell_m$ or $\ell \to \varepsilon$ is NP-complete.

Proof sketch. As before, we can assume that all formulae in STDs have no free variables. Membership in NP is easy by guessing an instance; for membership in PSPACE we transform a DTD such that all regular expressions become either conjunctions or disjunctions, and then use an alternating polynomial-time algorithm. For hardness, we use reductions from QSAT for PSPACE and 3SAT for NP. □

We finally identify a class for which consistency is tractable. This class is relevant in practical applications of data exchange such as those addressed by Clio [19, 23]. One extension of relational data exchange that is enabled by Clio is to nested relational schemas. Nested relations can naturally be represented by XML documents. In that case all the rules in DTDs are of the form $\ell \to \ell_1 \ldots \ell_m \ell_{m+1}^* \ldots \ell_{m+k}^*$, with all the ℓ_i's distinct.

We shall extend this, and consider *nested-relational* DTDs defined as non-recursive DTDs in which all rules are of the form

$$\ell \;\to\; \tilde{\ell}_0 \ldots \tilde{\ell}_m,$$

where all ℓ_i's are distinct, and each $\tilde{\ell}_i$ is one of the following: ℓ_i, or ℓ_i^*, or ℓ_i^+, or $\ell_i? = \ell_i \mid \varepsilon$. Such DTDs have also been looked at in the context of handling partial information in XML [3].

Theorem 4.3. Data-Exchange-Consistency *is solvable in polynomial time if both source and target DTDs are nested-relational.*

Proof sketch. We show how to transform, in linear time, a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ into a setting $(D'_{\mathbf{S}}, D'_{\mathbf{T}}, \Sigma'_{\mathbf{ST}})$ such that $D'_{\mathbf{S}}$ and $D'_{\mathbf{T}}$ are non-recursive DTDs in which all regular expressions are of the form $\ell_1, \ldots, \ell_m$, $m \geq 0$, with all the ℓ_i's distinct, formulae in $\Sigma'_{\mathbf{ST}}$ do not have free variables, and such that $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent iff $(D'_{\mathbf{S}}, D'_{\mathbf{T}}, \Sigma'_{\mathbf{ST}})$ is consistent. The latter can be checked in polynomial time because a DTD D with such regular expressions can have only one tree T_D that conforms to it. Then one evaluates left- and right-hand sides of the rules in $\Sigma'_{\mathbf{ST}}$ on the trees $T_{D'_{\mathbf{S}}}$ and $T_{D'_{\mathbf{T}}}$ conforming to $D'_{\mathbf{S}}$ and $D'_{\mathbf{T}}$, respectively, to verify if the setting is consistent. The trees that conform to $D'_{\mathbf{S}}$ and $D'_{\mathbf{T}}$ need not be of polynomial size, but the verification can be still be done in polynomial time by evaluating inductively all subformulae for each element type, in a manner similar to model-checking for modal and some branching-time temporal logics. □

The algorithm for checking consistency for nested-relational DTDs runs in time $O(nm^2)$ where n is the size of the DTDs and m is the size of the STDs.

5. Query Answering

Our goal is to define the concept of query answering in the XML data exchange scenario. Since we need to compute certain answers (which are defined as intersections of query results over all solutions), we consider queries which return tuples of values as opposed to arbitrary trees.

We already know from results on relational data exchange that answering general FO queries over target instances is problematic [6, 10], and most positive results have been proved for conjunctive or monotone queries [10, 11]. Thus, for our query language, we shall use the closure of tree-pattern formulae under conjunction and existential quantification. This is similar to conjunctive queries over child and descendant as defined in [13], again with the main difference being the use of free variables to collect attribute values, as opposed to outputting nodes of trees.

A query language $\mathcal{CTQ}^{//}$ is defined by

$$Q \;\; := \;\; \varphi \;\mid\; Q \wedge Q \;\mid\; \exists x\, Q,$$

where φ ranges over tree-pattern formulae. The semantics of $\wedge$ and $\exists$, as well as the definition of free variables, is standard. We note that as in the case of tree-pattern formulae, $\mathcal{CTQ}^{//}$-formulae are evaluated in an XML tree.

Notation $\mathcal{CTQ}^{//}$ stands for "conjunctive tree queries with descendant." If we do not allow descendant in queries, we obtain a fragment denoted by $\mathcal{CTQ}$. For example, consider a $\mathcal{CTQ}$ query $\psi(x)$ given by $\exists y\, book(@title = x)[author(@name = y)]$. Then the source document from the introduction, shown in Figure 1 (b), satisfies $\psi(Computational\ Complexity)$.

We shall also consider unions of conjunctive queries. By $\mathcal{CTQ}^{//,\cup}$ we denote the class of queries of the form $Q_1(\bar{x}) \cup \ldots \cup Q_m(\bar{x})$, where each Q_i is a query from $\mathcal{CTQ}^{//}$. By disallowing descendant in the Q_i's we obtain a restriction denoted by $\mathcal{CTQ}^{\cup}$.

5.1 Certain answers.

Assume that we are given a data exchange setting $(D_{\mathbf{S}},\, D_{\mathbf{T}},\, \Sigma_{\mathbf{ST}})$, a source XML tree T that conforms to $D_{\mathbf{S}}$, and a $\mathcal{CTQ}^{//,\cup}$ query $Q(\bar{x})$. What does it mean to answer Q? As in the case of relational data exchange [10, 11], since there may be many possible solutions to the data exchange problem, we define the semantics of Q in terms of *certain answers*:

$$\underline{certain}(Q, T) \quad = \bigcap_{T' \text{ is a solution for } T} Q(T').$$

Thus, a tuple $\bar{s}$ of strings is in $\underline{certain}(Q, T)$ if $\bar{s} \in Q(T')$ for every solution solution T' for T. If Q is a Boolean query (a sentence), then $\underline{certain}(Q, T) = true$ iff for every solution T' for T, we have $T' \models Q$.

Let $(D_{\mathbf{S}},\, D_{\mathbf{T}},\, \Sigma_{\mathbf{ST}})$ be a data exchange setting. The main problem we study is:

PROBLEM:	CERTAIN-ANSWERS(Q).
INPUT:	An XML tree T conforming to $D_{\mathbf{S}}$ and a tuple $\bar{s}$ of strings.
QUESTION:	Is $\bar{s} \in \underline{certain}(Q, T)$?

If Q is a Boolean query ($m = 0$) then the input to the problem is an XML tree T and the problem is to verify whether $\underline{certain}(Q, T) = true$.

Notice that as in the relational case, only tuples from Const could belong to $\underline{certain}(Q, T)$.

5.2 Unordered trees.

Our query answering algorithms take advantage of temporarily "forgetting" about the sibling order. That is, we construct a target tree which does not conform to the target DTD but could be rearranged into one conforming to the DTD simply by imposing a correct sibling order. To capture this, we introduce a class of languages which are permutations of regular languages, and the notion of satisfaction of DTDs by unordered trees.

Given a regular expression r over an alphabet Γ, we let $L(r)$ stand for the language denoted by r. Then we define $\pi(r) \subseteq \Gamma^*$ as the set of all strings w which are permutations of strings in $L(r)$. For example, if $r = (ab)^*$, then $\pi(r)$ has strings in which the number of a's equals the number of b's. Thus, $\pi(r)$ need not be regular; in fact it may not even be context-free because $\pi((abc)^*) \cap L(a^*b^*c^*) = \{a^n b^n c^n \mid n \geq 0\}$.

An *unordered XML tree* is defined as a directed tree $(N,\, <_{\text{child}},\, root)$ (that is, it excludes the sibling order $<_{\text{sib}}$). Given an unordered XML tree T and a DTD D, we say that T conforms to D, denoted by $T \mathrel{|\!\approx} D$, if for every node v in T with children $v_1, \ldots, v_m$ and $\lambda_T(v) = \ell$, the string $\lambda_T(v_1) \ldots \lambda_T(v_m)$ is in $\pi(P(\ell))$, and items 2 and 3 of the definition of $T \models D$ are true. That is, $\lambda_T(v_1) \ldots \lambda_T(v_m)$ is a permutation of some string in the language of $P(\ell)$.

We say that an unordered XML tree T' is a *solution* for an XML tree T in a data exchange setting $(D_{\mathbf{S}},\, D_{\mathbf{T}},\, \Sigma_{\mathbf{ST}})$ if $T' \mathrel{|\!\approx} D_{\mathbf{T}}$ and $\langle T, T' \rangle$ satisfies[2] all the STDs from $\Sigma_{\mathbf{ST}}$. As in the case of ordered trees, we define the semantics of $\mathcal{CTQ}^{//,\cup}$-queries in terms of certain answers, that is, given an XML tree $T \models D_{\mathbf{S}}$, a $\mathcal{CTQ}^{//,\cup}$-query $Q(\bar{x})$ over $D_{\mathbf{T}}$ and and a tuple $\bar{s}$ of strings, we say that $\bar{s} \in \underline{certain}^{\mathsf{un}}(Q, T)$ if and only if $\bar{s} \in Q(T')$ for every unordered solution T' for T.

The following proposition allows one to forget about the sibling ordering while computing certain answers and, in particular, it allows one to use unordered trees when proving lower bounds for this problem.

Proposition 5.1. *Given an XML data exchange setting* $(D_{\mathbf{S}},\, D_{\mathbf{T}},\, \Sigma_{\mathbf{ST}})$*, an XML tree* $T \models D_{\mathbf{S}}$ *and a* $\mathcal{CTQ}^{//,\cup}$*-query* $Q(\bar{x})$*, we have*

$$\underline{certain}(Q, T) = \underline{certain}^{\mathsf{un}}(Q, T).$$

Furthermore, tractable query answering algorithms in this paper will be constructing a certain unordered solution T^* satisfying $\underline{certain}^{\mathsf{un}}(Q, T) = Q(T^*)$. This can be done without loss of generality since every unordered solution can be turned into an ordered solution, and this can be done in polynomial time, as the following result shows.

Given an unordered tree T and a sibling ordering $\prec_{\text{sib}}$, let $T^{\prec_{\text{sib}}}$ be the resulting ordered tree. Then:

Proposition 5.2. *Suppose* $T \mathrel{|\!\approx} D$*. The one can compute, in polynomial time in the size of* T*, a local sibling ordering* $\prec_{\text{sib}}$ *on* T *such that* $T^{\prec_{\text{sib}}} \models D$*.*

We shall also need complexity bounds for checking whether a string w is in $\pi(r)$. Since the Parikh image of a regular language is a semilinear set, this reduces to integer linear programing, which is in NP in general, and in polynomial time if dimension is fixed [17]. This gives us the following.

Proposition 5.3. *The problem of checking whether* w *is in* $\pi(r)$ *for a string* w *and a regular expression* r *is NP-complete. For each fixed* r*, checking whether* w *is in* $\pi(r)$ *can be done in polynomial time.*

[2] The notion of satisfaction of a tree-pattern formula by an unordered tree is defined exactly as in the case of (ordered) XML trees.

5.3 First complexity results: upper bound and some hard cases.

Our goal is to determine the complexity of computing certain answers. A priori it is not even clear if the problem is decidable, but we can prove the following upper bound.

Theorem 5.4. *If Q is a $\mathcal{CTQ}^{//,\cup}$-query, then* $\textsc{Certain-Answers}$*(Q) is in coNP.*

Proof sketch. It suffices to show that if $\bar{a} \notin \underline{certain}(Q, T)$, then one can construct a tree T' of polynomial size (in $\|T\|$) such that $\bar{a} \notin Q(T')$. For this, we take an arbitrary solution T_0 for T for which $\bar{a} \notin Q(T_0)$ and show how to reduce its size, by cutting both long paths and long sibling chains to reduce it to the desired tree T'. While reducing the size, we keep a certain skeleton that witnesses the fact that T_0 is a solution. The main difficulty is in cutting long paths: while our queries are monotone, by removing parts of paths we introduce new "child" edges, and in general it is impossible to conclude that after such a cut $\bar{a}$ is not in the result of the query. □

We would like to identify tractable cases of the $\textsc{Certain-Answers}$ problem. Its complexity is mostly affected by target formulae in STDs and target DTDs, since in the definition of certain answers we take the intersection over all instances satisfying target formulae and the target DTD.

We now identify a necessary restriction for tractability. We define a class of STDs and show that outside of this class we get coNP-hard instances of $\textsc{Certain-Answers}$ even for very simple DTDs.

Definition 5.5. *A source-to-target dependency $\psi_{\mathbf{T}}(\bar{x}, \bar{z})$:– $\varphi_{\mathbf{S}}(\bar{x}, \bar{y})$ is* fully-specified *if $\psi_{\mathbf{T}}$ is of the form $\underline{r}[\varphi_1, \ldots, \varphi_k]$, where $\underline{r}$ is the type of the root and φ_i's do not use descendant // and wildcard _.*

For example, the following source-to-target dependency is fully-specified:

$$bib[writer(@name = y)[work(@title = x)]] \text{ :- }$$
$$book(@title = x)[author(@name = y)].$$

The definition of fully-specified STDs puts three restriction on target formulae: they are witnessed at the root, there is no descendant, and no wildcard. By relaxing those, we can get three classes of STDs, in which target formula satisfy only two of the three restrictions. We denote them by STD(_, //) (wildcard and descendant are forbidden), STD($\underline{r}$, //) (formulae $\underline{r}[\varphi_1, \ldots, \varphi_k]$ in which descendant is forbidden), and STD($\underline{r}$, _) (formulae $\underline{r}[\varphi_1, \ldots, \varphi_k]$ in which wildcard is forbidden).

We call a regular expression r *simple* if either $r = \varepsilon$ or $r = (a_1|a_2|\cdots|a_n)^*$, where $n \geq 1$ and $a_1, a_2, \ldots, a_n$ are pairwise distinct symbols. Simple regular expressions are the simplest expressions that can be used in DTDs, as they impose restrictions neither on the cardinalities nor on the ordering of children, they just specify their types.

Theorem 5.6. *For each of the three classes* STD(_, //), STD($\underline{r}$, //), *and* STD($\underline{r}$, _), *one can find a data exchange setting in which all STDs belong to that class, and a $\mathcal{CTQ}$-query Q such that* $\textsc{Certain-Answers}$*(Q) is coNP-complete, even if all regular expressions used in source and target DTDs are simple.*

Thus, from now one we concentrate on fully-specified STDs. Our goal is to provide a classification of data exchange settings for which computing certain answers is tractable.

6. Computing Certain Answers: Classification and Dichotomy

Proviso: throughout this section, all source-to-target dependencies are fully-specified. As was shown earlier, outside of this class one cannot avoid coNP-hardness even for very simple source and target DTDs.

Our goal now is to classify the complexity of the $\textsc{Certain-Answers}$ problem. As was explained earlier, it depends heavily on target DTDs. We shall classify target DTDs and prove a dichotomy theorem which states that depending on a class of regular languages used in DTDs, computing certain answers is either tractable or coNP-complete.

If $\mathcal{C}$ is a class of regular expressions, we say that a DTD D is a $\mathcal{C}$-DTD if all regular expressions in D belong to $\mathcal{C}$.

Definition 6.1. *Given a class $\mathcal{C}$ of regular expressions, and a class $\mathcal{Q}$ of queries, we say that*

- *$\mathcal{C}$ is* tractable *for $\mathcal{Q}$ if for every data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ with $D_{\mathbf{T}}$ being a $\mathcal{C}$-DTD, and every $Q \in \mathcal{Q}$, the problem* $\textsc{Certain-Answers}(Q)$ *is in PTIME;*
- *$\mathcal{C}$ is* coNP-complete *for $\mathcal{Q}$ if there exists a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ with $D_{\mathbf{T}}$ being a $\mathcal{C}$-DTD, and a query $Q \in \mathcal{Q}$ such that* $\textsc{Certain-Answers}(Q)$ *is coNP-complete;*
- *$\mathcal{C}$ is* strongly coNP-complete *for $\mathcal{Q}$ if the above holds when $D_{\mathbf{S}}$ is simple and Q is a Boolean query.*

We want our classes of regular expressions to have some degree of uniformity: that is, we want to disallow classes that contain just a finite number of regular expressions, or only regular expressions that generate finite languages. We thus impose the constraint that all classes $\mathcal{C}$ contain at least all simple regular expressions (recall that these are of the form $(a_1|a_2|\cdots|a_n)^*$ or ε). Such classes will be called *admissible*.

Theorem 6.2. **(Dichotomy)** *Let $\mathcal{C}$ be an admissible class of regular expressions and $\mathcal{Q}$ be one of $\mathcal{CTQ}$, $\mathcal{CTQ}^{//}$, $\mathcal{CTQ}^{\cup}$ and $\mathcal{CTQ}^{//,\cup}$. Then $\mathcal{C}$ is either tractable, or strongly coNP-complete for $\mathcal{Q}$-queries.*

Furthermore, for each data exchange setting it is decidable if it falls in the tractable case, and in this *case there is a polynomial time algorithm that for each source tree T produces a solution $T^\star$ such that $\bar{s} \in \underline{certain}(Q,T)$ iff $\bar{s} \in Q(T^\star)$ for every tuple $\bar{s}$ from* Const*.*

In the rest of the section, we outline the proof of this result and the polynomial-time algorithm. We introduce a class $\mathcal{C}_U$ of regular expressions that is tractable for $\mathcal{CTQ}^{//,\cup}$-queries (and thus also for $\mathcal{CTQ}$-, $\mathcal{CTQ}^{//}$- and $\mathcal{CTQ}^{\cup}$-queries). Then we show that every admissible class of regular expressions $\mathcal{C} \not\subseteq \mathcal{C}_U$ is strongly coNP-complete for $\mathcal{CTQ}$-queries (and thus also for $\mathcal{CTQ}^{//}$-, $\mathcal{CTQ}^{\cup}$- and $\mathcal{CTQ}^{//,\cup}$-queries).

6.1 The tractable case.

We explain how to compute the canonical tree $T^\star$ over which $\mathcal{CTQ}^{//,\cup}$-queries can be evaluated to produce $\underline{certain}(Q,T)$. The restrictions on the class $\mathcal{C}_U$ guarantee that the construction is done in PTIME.

Fix a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$, where $D_{\mathbf{T}} = (P_{\mathbf{T}}, R_{\mathbf{T}}, \underline{r})$. For every tree-pattern formula $\varphi(\bar{x})$ not mentioning descendant // and wildcard _ and for every tuple $\bar{s}$ of strings, there exists an unordered tree $T_{\varphi(\bar{s})}$ naturally associated with $\varphi(\bar{s})$. It is constructed inductively: if $\varphi(\bar{s}) = \ell(@a_1 = s_1, \ldots, @a_n = s_n)[\varphi_1(\bar{s}_1), \ldots, \varphi_k(\bar{s}_k)]$, then the root of $T_{\varphi(\bar{s})}$ is a node v_0 of type ℓ that has attributes $@a_1$, …, $@a_n$ with values s_1, …, s_n, and k distinct children v_1, …, v_k, with v_i being the root of tree $T_{\varphi_i(\bar{s}_i)}$, for $i \leq k$.

Suppose we are given a source tree T, and a family Σ of STDs. We define the *canonical pre-solution* for T, denoted by $cps(T)$, as an unordered tree T' constructed as follows. For each STD $\underline{r}[\psi_1(\bar{x},\bar{z}), \ldots, \psi_k(\bar{x},\bar{z})] \mathrel{:-} \varphi_{\mathbf{S}}(\bar{x},\bar{y})$ and tuples $\bar{s}, \bar{s}'$ from Const such that $T \models \varphi_{\mathbf{S}}(\bar{s},\bar{s}')$, choose a fresh tuple $\bar{s}''$ of strings from Var and construct trees $T_{\psi_i(\bar{s},\bar{s}'')}$. The tree $cps(T)$ has all such trees $T_{\psi_i(\bar{s},\bar{s}'')}$ as distinct subtrees at the children of its root.

For example, consider DTDs $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$ in Figure 3 (a), (b). In $D_{\mathbf{S}}$ we have rules

$$\underline{r} \to b^*c^*, \;\; b \to \varepsilon, \;\; c \to \varepsilon,$$

and b and c have attribute $@\ell$; in $D_{\mathbf{T}}$ we have

$$\underline{r} \to (cd)^*, \;\; c \to \varepsilon, \;\; d \to e, \;\; e \to \varepsilon,$$

and c, e have attributes $@m$ and $@n$, respectively. If T is the source tree in Figure 3 (c), and $\Sigma_{\mathbf{ST}}$ contains STDs

$$\begin{array}{lcl} \underline{r}[c(@m = x)] & :- & b(@\ell = x) \\ \underline{r}[c(@m = x)] & :- & c(@\ell = x), \end{array}$$

then $cps(T)$ is the tree shown in Figure 3 (d).

Canonical pre-solutions can be computed in PTIME; the problem is that they may not conform to the target DTD (as in the example in Figure 3). We present an algorithm for computing a canonical solution for a tree T from $cps(T)$. The key is to find a "repair" every time we have a violation of constraints imposed by the target DTD. Given a node v of an unordered target tree T', we say that (T',v) violates $D_{\mathbf{T}}$ if v does not have the right attributes or the children of v do not have the right types, that is, if $\{@a \mid \rho_{@a}(v)$ is defined in $T'\} \neq R_{\mathbf{T}}(\lambda_{T'}(v))$ or $\lambda_{T'}(children(v)) \notin \pi(P_{\mathbf{T}}(\lambda_{T'}(v)))$, where $\lambda_{T'}(children(v))$ refers to the string $\lambda_{T'}(v_1) \ldots \lambda_{T'}(v_n)$, and $v_1, \ldots, v_n$ are the children of v.

The "easy" violations are those when nodes do not have the right attributes: if they miss some, we add them and give them fresh values from Var; if they have extra attributes, the repair algorithm fails. More precisely, repairing function $\mathbf{ChangeAtt}$ receives as parameters a target tree T' and a node v such that $\{@a \mid \rho_{@a}(v)$ is defined in $T'\} \neq R_{\mathbf{T}}(\lambda_{T'}(v))$. This function fails if there exists an attribute $@a$ such that $\rho_{@a}(v)$ is defined in T' and $@a \notin R_{\mathbf{T}}(\lambda_{T'}(v))$, since in this case $\Sigma_{\mathbf{ST}}$ forces v to have attribute $@a$ while $D_{\mathbf{T}}$ does not allow v to have such an attribute. Otherwise, for every $@a \in R_{\mathbf{T}}(\lambda_{T'}(v))$ such that $\rho_{@a}(v)$ is not defined, $\mathbf{ChangeAtt}$ assigns a fresh value from Var to $\rho_{@a}(v)$.

The "hard" violations are those when sequences of children do not satisfy the constraints imposed by regular expressions in DTDs. Repairing function $\mathbf{ChangeReg}$ (defined later) tries to repair these violations: It receives as parameters a target tree T' and a node v such that $\lambda_{T'}(children(v)) \notin \pi(P_{\mathbf{T}}(\lambda_{T'}(v)))$, and it either fails or returns a tree T'' such that $\lambda_{T''}(children(v)) \in \pi(P_{\mathbf{T}}(\lambda_{T''}(v)))$.

Functions $\mathbf{ChangeAtt}$ and $\mathbf{ChangeReg}$ are applied to $cps(T)$, in no particular order, until we reach a tree $T^\star$ that either conforms to $D_{\mathbf{T}}$ or is not reparable (that is, the repair algorithm fails). In the first case we say that $T^\star$ is a *canonical solution* for T. For example, Figure 3 (e) shows a canonical solution for the tree in Figure 3 (c).

For the class $\mathcal{C}_U$ (to be defined shortly) we prove:

Lemma 6.3. *If $D_{\mathbf{T}}$ is a $\mathcal{C}_U$-DTD, then for every source tree T:*

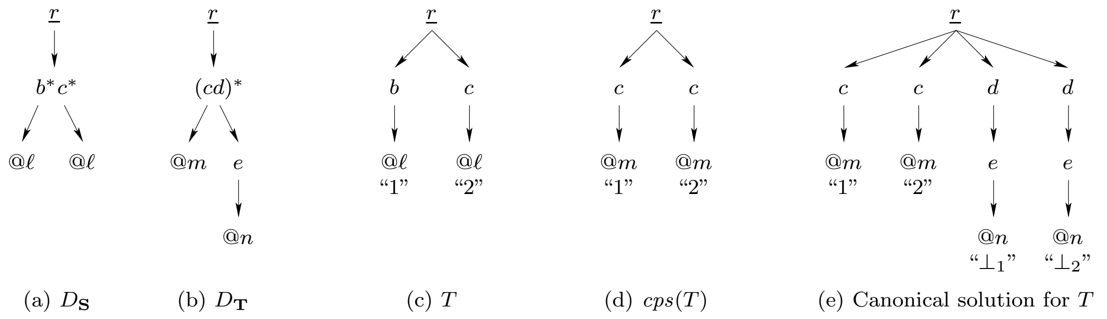
a) There exists a solution for T iff there exists a canonical solution for T.

b) If $T^\star$ is a canonical solution for T, then for every $\mathcal{CTQ}^{//,\cup}$-query $Q(\bar{x})$ and every tuple $\bar{s}$ from Const*, $\bar{s} \in \underline{certain}(Q,T)$ iff $T^\star \models Q(\bar{s})$ (if Q is Boolean, then $\underline{certain}(Q,T) = true$ iff $T^\star \models Q$).*

Furthermore, for $\mathcal{C}_U$-DTDs canonical solutions can be computed efficiently by repeatedly applying $\mathbf{ChangeAtt}$ and $\mathbf{ChangeReg}$.

Lemma 6.4. *If $D_{\mathbf{T}}$ is a $\mathcal{C}_U$-DTD, then it can be checked in polynomial time whether there exists a canonical solution for a given source tree T. Furthermore, if such a solution exists, then it can be computed in polynomial time.*

By putting together these two lemmas we obtain:



(a) $D_{\mathbf{S}}$ (b) $D_{\mathbf{T}}$ (c) T (d) $cps(T)$ (e) Canonical solution for T

Figure 3: Source DTD $D_{\mathbf{S}}$, target DTD $D_{\mathbf{T}}$, source tree T conforming to $D_{\mathbf{S}}$, canonical pre-solution for T and canonical solution for T.

Proposition 6.5. *$\mathcal{C}_U$ is tractable for $\mathcal{CTQ}^{//,\cup}$-queries.*

Now we define the class $\mathcal{C}_U$ and explain how **ChangeReg** works. First, we need some terminology. Let $alph(w)$ (or $alph(r)$) stands for the set of alphabet symbols mentioned in a string w (or a regular expression r). For every $a \in alph(w)$, let $\#_a(w)$ be the number of occurrences of a in w. We write $w \preceq w'$ if $\#_a(w) \leq \#_a(w')$ for every $a \in alph(w)$, and $w \prec w'$ if $w \preceq w'$ and $w' \not\preceq w$.

ChangeReg receives as parameters an unordered tree T' and a node v such that $\lambda_{T'}(children(v)) \notin \pi(P_{\mathbf{T}}(\lambda_{T'}(v)))$. Assume that $\ell = \lambda_{T'}(v)$, $w = \lambda_{T'}(children(v))$ and $r = P_{\mathbf{T}}(\ell)$. To adjust w to make T' conform to $D_{\mathbf{T}}$, **ChangeReg** may need to extend w to a string in the set $min\_ext(w,r)$ of minimal extensions of w that fall into $\pi(r)$:

$$min\_ext(w,r) \quad = \quad \min_{\preceq}\{w' \mid w' \in \pi(r),\ w \preceq w'\}.$$

For example, $min\_ext(b, (bbc)^*) = \{bbc, bcb, cbb\}$. Sometimes **ChangeReg** may need to extend not w itself but a substring of w. For example, $min\_ext(bb, bc^+) = \emptyset$ and, thus, the only way to repair bb is to merge two b's into a single b and then expand to a string in $\pi(r)$. The resulting strings from the process of expanding substrings of w are the strings from which **ChangeReg** will be chosen a candidate to replace w. Formally, the set of possible repairs of w, denoted by $rep(w,r)$, is defined as:

$$rep(w,r) \quad = \bigcup_{w' \preceq w,\ alph(w')=alph(w)} min\_ext(w',r).$$

In this definition, we only consider strings w' such that $alph(w) = alph(w')$, since $\Sigma_{\mathbf{ST}}$ forces v to have at least one child of type b, for every $b \in alph(w)$.

ChangeReg replaces w by a string $w' \in rep(w,r)$. But $rep(w,r)$ may have more than one element; for example, $ccdd$ and cd are in $rep(cc, (cd)^*e^*)$. To choose one, we try to merge as few nodes as possible (so as to avoid attribute clashes) and to add as few new element types as possible (we prefer $ccdd$ to $ccdde$). This is captured by the preference relation $\preceq_w$ defined by $w_1 \preceq_w w_2$ iff (1) $\#_b(w_2) \geq \min\{\#_b(w_1), \#_b(w)\}$ for all $b \in alph(w)$, and (2) $alph(w_2) \setminus alph(w) \subseteq alph(w_1) \setminus alph(w)$. Thus, **ChangeReg** replaces w by $w' \in \max_{\preceq_w} rep(w,r)$.

The canonical solution for a source tree must be unique, no matter which string **ChangeReg** picks from $\max_{\preceq_w} rep(w,r)$ and no matter how **ChangeReg** merges the elements of w. The problem is that for an arbitrary regular expression this does not necessarily holds. Thus, we have to restrict our attention to regular expressions such that (1) $\max_{\preceq_w} rep(w,r)$ has a "best" candidate w' and (2) if $\#_b(w) > \#_b(w')$, then $\#_b(w')$ is equal to 1, so that there is only one way to merge the children of v of type b. We now define these conditions formally.

For a regular expression r and $a \in alph(r)$, let $\mathit{fixed}_a(r)$ be the set of $w \in \pi(r)$ such that $w' \in \pi(r)$ and $w \preceq w'$ imply $\#_a(w) = \#_a(w')$. For example, if $r = a \mid aab^*$, then $aa \in \mathit{fixed}_a(r)$ since every string $w \in \pi(r)$ such that $aa \preceq w$ is a permutation of a string of the form aab^n ($n \geq 0$) and, hence, $\#_a(aa) = \#_a(w) = 2$. On the other hand, $a \notin \mathit{fixed}_a(r)$ since $a \preceq aa \in \pi(r)$ and $\#_a(a) < \#_a(aa)$. If $\mathit{fixed}_a(r) \neq \emptyset$, then define $c_a(r) = \max\{\#_a(w) \mid w \in \mathit{fixed}_a(r)\}$. If $\mathit{fixed}_a(r) = \emptyset$, then $c_a(r) = 0$. Finally,

$$c(r) \quad = \quad \max\{c_a(r) \mid a \in alph(r)\}.$$

For example, $c_a(a \mid aab^*) = 2$ and $c_b(a \mid aab^*) = 0$, and, thus, $c(a \mid aab^*) = 2$.

Lemma 6.6. *$c(r)$ is finite for every r.*

We say that a regular expression r is *univocal* if $c(r) \leq 1$ and for every string w such that $rep(w,r) \neq \emptyset$, the set $rep(w,r)$ has a maximum element with respect to $\preceq_w$: that is, an element $w' \in rep(w,r)$ such that $w'' \preceq_w w'$ for all $w'' \in rep(w,r)$. For example, all of the following are univocal regular expressions: $bc^+d^*e?$, $(b^*|c^*)$ and $(bc)^*(de)^*$. We let $\mathcal{C}_U$ be the class of univocal regular expressions. It is easy to see that all simple regular expressions are univocal, and hence $\mathcal{C}_U$ is an admissible class.

Proposition 6.7. *It is decidable whether a regular expression r is univocal. In fact, for each r one can compute a sentence Φ_r of Presburger Arithmetic which is true iff r is univocal.*

Summing up, if $D_{\mathbf{T}}$ is a $\mathcal{C}_U$-DTD, then **ChangeReg**(T', v) works as follows. Recall that $\ell = \lambda_{T'}(v)$, $w =$

$\lambda_{T'}(children(v))$ and $r = P_{\mathbf{T}}(\ell)$. Initially, $\mathbf{ChangeReg}$ checks whether $rep(w, r)$ is empty. If this is the case, then it fails. Otherwise, $\mathbf{ChangeReg}$ picks an arbitrary string w' from $\max_{\preceq_w} rep(w, r)$, and then it replaces w by w'. More precisely, let $b \in alph(r)$, $p = \#_b(w)$ and $q = \#_b(w')$. If $p < q$, then $\mathbf{ChangeReg}$ adds $(q - p)$ new children to v of type b, each of them having no attributes and no children[3]. If $q < p$, then $q = 1$ (since r is univocal) and, thus, $\mathbf{ChangeReg}$ replaces the sequence $v_1, \ldots, v_p$ of children of v of type b by a single fresh node v' of type b, and then for every subtree T_i of T' rooted at v_i ($i \in [1, p]$), it replaces the root of T_i by v'. At this point $\mathbf{ChangeReg}$ fails if there is an attribute clash, that is, if there is a pair of subtrees of T' rooted at v_i, v_j ($i, j \in [1, p]$) and an attribute $@a$ such that $\rho_{@a}(v_i) \in \mathsf{Const}$, $\rho_{@a}(v_j) \in \mathsf{Const}$ and $\rho_{@a}(v_i) \neq \rho_{@a}(v_j)$.

It is then possible to show that that $\mathbf{ChangeReg}$ runs in polynomial time for every fixed DTD $D_{\mathbf{T}}$.

It is easy to see that all regular expressions used in nested-relational DTDs are univocal. Hence, the following extension of relational data exchange handled by Clio [23] falls in the following large tractable case:

Corollary 6.8. *If* $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ *is a data exchange setting in which* $D_{\mathbf{T}}$ *is nested-relational, and* Q *is a* $\mathcal{CTQ}^{//,\cup}$*-query, then* CERTAIN-ANSWERS(Q) *is in PTIME.*

We finally remark that canonical tree $T^\star$ is unordered and hence may not conform to the target DTD with an arbitrary sibling ordering imposed on it. However, if one needs to materialize the target instance $T^\star$, by Proposition 5.2 one can transform $T^\star$, in polynomial time, into a tree that conforms to the target DTD.

6.2 The intractable case.

The following shows that $\mathcal{C}_U$ is the maximal tractable class, and thus completes the classification of finding certain answers and proves the dichotomy theorem.

Proposition 6.9. *Let* $\mathcal{C}$ *be an admissible class of regular expressions such that* $\mathcal{C} \not\subseteq \mathcal{C}_U$. *Then* $\mathcal{C}$ *is strongly coNP-complete for* $\mathcal{CTQ}$*-queries.*

This result is a consequence of the following lemmas.

Lemma 6.10. *Let* r *be a regular expression such that* $c(r) \geq 2$ *and* $\mathcal{C}$ *an admissible class of regular expressions containing* r. *Then* $\mathcal{C}$ *is strongly coNP-complete for* $\mathcal{CTQ}$*-queries.*

Lemma 6.11. *Let* r *be a non-univocal regular expression such that* $c(r) \leq 1$ *and* $\mathcal{C}$ *an admissible class of regular expressions containing* r. *Then* $\mathcal{C}$ *is strongly coNP-complete for* $\mathcal{CTQ}$*-queries.*

7. Conclusions

We have defined the basic notions of XML data exchange: source-to-target constraints, data exchange settings, consistency and query answering problems. We have seen that transferring relational data exchange results to the XML setting requires considerable effort, even in the fairly simple setting that shows how to translate source patterns into target patterns. We have shown that, while checking consistency is hard in general, it is tractable for a practically relevant class handled by the Clio system at IBM [23]. For query answering, we showed a dichotomy, that separates query answering instances into tractable and coNP-complete ones, depending on properties of DTDs and constraints.

As far as the theoretical foundations of XML data exchange are concerned, this paper uncovered at most the tip of the iceberg. We now briefly list other problems that seem to be worthy a theoretical investigation.

The standard notions of local-as-view and global-as-view from data integration [18] have been adapted in relational data exchange [10, 11] and sometimes they lead to better algorithms or easier analysis of the behavior of data exchange settings and queries. So far we have not made these notions precise in the XML case.

We have concentrated on tree patterns that use the child and descendant axes of XPath; in the future we plan to consider more expressive source-to-target constraints that use other axes such as next sibling. We also would like to consider more expressive schema constraints (for example, ID and IDREF attributes).

Finally, to define the notion of certain answers, we used queries that produce tuples of values. Most XML queries produce trees, but it is not at all clear how to define the certain answers semantics for them. We plan to work on this in the future.

Acknowledgments We are very grateful to Ron Fagin, Phokion Kolaitis, and Lucian Popa for many helpful discussions during the early stages of this project, and to Pablo Barceló and Wenfei Fan for their comments on the draft. The authors were supported by grants from NSERC and CITO, and M. Arenas was supported by a graduate fellowship from IBM and FONDECYT grant 1050701. Part of this work was done while M. Arenas was at IBM Almaden.

8. References

[1] S. Abiteboul, O. Duschka. Complexity of answering queries using materialized views. In *PODS 1998*, pages 254–263.

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

[3] Violations generated by adding b-nodes without attributes or children are repaired later by repeatedly applying $\mathbf{ChangeAtt}$ and $\mathbf{ChangeReg}$.

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

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

[5] S. Amer-Yahia, Y. Kotidis. Web-services architecture for efficient XML data exchange. In *ICDE 2004*, pages 523–534.

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

[7] M. Benedikt, W. Fan, G. Kuper. Structural properties of XPath fragments. In *ICDT 2003*, pages 79–95.

[8] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi. *Tree Automata: Techniques and Applications*. Available at `www.grappa.univ-lille3.fr/tata`. October 2002.

[9] A. Deutsch, V. Tannen. Containment and integrity constraints for XPath. In *KRDB 2001*.

[10] R. Fagin, Ph. Kolaitis, R. Miller, L. Popa. Data exchange: semantics and query answering. In *ICDT'03*, pp. 207–224.

[11] R. Fagin, Ph. Kolaitis, L. Popa. Data exchange: getting to the core. In *PODS'03*, pages 90–101.

[12] R. Fagin, Ph. Kolaitis, L. Popa, W.C. Tan. Composing schema mappings: second-order dependencies to the rescue. *PODS 2004*, pages 83-94

[13] G. Gottlob, C. Koch, K. Schulz. Conjunctive queries over trees. *PODS 2004*, pages 189–200.

[14] T. Imielinski, W. Lipski. Incomplete information in relational databases. *J. ACM* 31 (1984), 761–791.

[15] R. Krishnamurthy, R. Kaushik, J. Naughton. XML-SQL query translation literature: the state of the art and open problems. In *Xsym 2003*, pages 1–18.

[16] L. Lakshmanan, G. Ramesh, H. Wang, Z. Zhao. On testing satisfiability of tree pattern queries. *VLDB 2004*, pages 120–131.

[17] H. W. Lenstra. Integer programming in a fixed number of variables. *Math. Oper. Res.* 8 (1983), 538–548.

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

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

[20] F. Neven. Automata, logic, and XML. In *CSL 2002*, pages 2–26.

[21] F. Neven, T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *ICDT'03*, pages 315–329.

[22] C. H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28 (1981), 765–768.

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

[24] H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.* 19 (1990), 424–437.

[25] N. Shu, B. Housel, R. Taylor, S. Ghosh, V. Lum. EXPRESS: a data extraction, processing, and restructuring system. *TODS* 2 (1977), 134–174.

[26] V. Vianu. A Web Odyssey: from Codd to XML. In *PODS'01*.

[27] P. Wood. Containment for XPath fragments under DTD Constraints. In *ICDT'03*, pages 300–314.

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