XML Data Exchange: Consistency and Query Answering

MARCELO ARENAS
Pontificia Universidad Católica de Chile
and
LEONID LIBKIN
University of Edinburgh

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.

Categories and Subject Descriptors: []:

General Terms:

Additional Key Words and Phrases:

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 [Shu et al. 1977] 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 [Amer-Yahia and Kotidis 2004]. A system Clio for data exchange was built [Miller et al. 2001; Popa et al. 2002] and

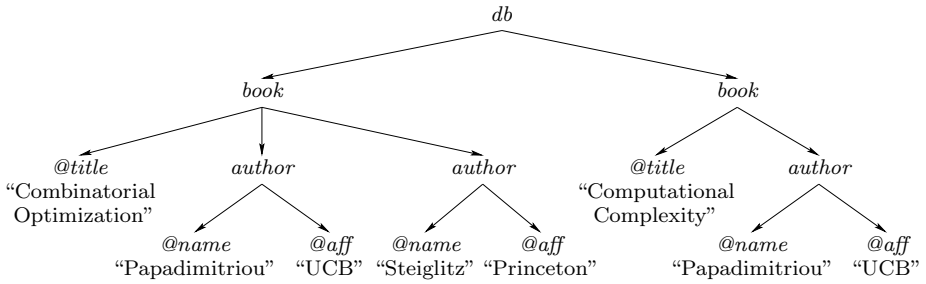
---

This is an expanded version of [Arenas and Libkin 2005].

Authors' email addresses: marenas@ing.puc.cl (Marcelo Arenas) and libkin@inf.ed.ac.uk (Leonid Libkin).

```
<!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

Fig. 1. Source information.

partly incorporated into the latest release of IBM's db2 product. At about the same time, papers [Fagin et al. 2003; Fagin et al. 2003] 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 [Fagin et al. 2005] and query rewriting [Arenas et al. 2004; Yu and Popa 2004]. And even though practical systems such as Clio handle non-relational data (in particular, nested relations [Popa et al. 2002]), 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.

Example 1.1. 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 [Fagin et al. 2005]. 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* 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 XML 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 [Fagin et al. 2005; Fagin et al. 2005]. 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

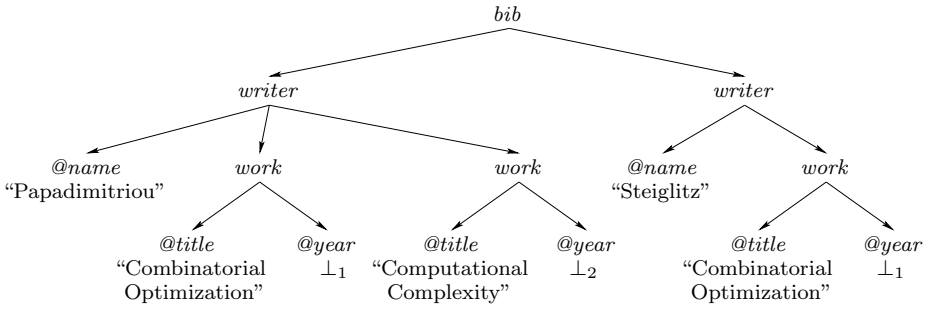
$$\psi_{\mathbf{T}}(\bar{x}, \bar{z}) \ :- \ \varphi_{\mathbf{S}}(\bar{x}, \bar{y}), \tag{1}$$

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 [Fagin et al. 2005; Fagin et al. 2005]. 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* [Abiteboul et al. 1991; Imielinski and Lipski 1984]: 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 [Fagin et al. 2005; Fagin et al. 2005]: 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

```
<!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

Fig. 2. Target information.

certain answers can be intractable [Abiteboul and Duschka 1998; Abiteboul et al. 1991].

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 [Krishnamurthy et al. 2003] (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. Therefore, 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 summarize our main results and outline the structure of the paper.

—Basic definitions (XML documents, DTDs) are given in Section 2.

—In Section 3, we define data exchange settings based on STDs which show how

patterns in the source tree translate into patterns in the target tree.

—In Section 4, we study the consistency problem of XML data exchange. We want to exclude *inconsistent* data exchange settings, in which target instances cannot be constructed. Our first result, Theorem 4.1, shows that the complexity of checking consistency of XML data exchange settings is EXPTIME-complete. In addition, we show that many reasonable restrictions remain computationally hard (complete for PSPACE or NP). However, for a practically relevant class, which subsumes non-relational data exchange settings handled by Clio, consistency can be checked in $O(nm^2)$, where n is the size of the DTDs, and m is the size of the STDs (Theorem 4.5) .

—In Section 5, we study the problem of query answering in XML data exchange. Since for a given source document T there could be many documents T' that satisfy all the STDs (solutions), we define query answers as those that are true in *all* solutions, i.e. certain answers. We look at queries that produce sets of tuples of values so that the notion of certain answers be well-defined.
Our first result (Theorem 5.5) is a coNP upper bound on the complexity of query answering. For XML data exchange, this bound is much harder to achieve than for relational data exchange.
Our next result delineates the boundary of intractability for query answering in data exchange. We show that if tree patterns over the target in STDs use one of three features – the descendant relation, the wildcard, or patterns that do not start at the root – then query answering could be coNP-hard even for very simple DTDs (Theorem 5.11). Thus, from that point on, we deal with target patterns that start at the root, use only the child relation, and do not use the wildcard.

—For such patterns, called fully-specified, in Section 6, 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 (Theorem 6.2). Regular expressions in the class that guarantees tractability are called *univocal*. It is decidable if a regular expression is univocal (Proposition 6.10). For tractable cases, which subsume nonrelational data exchange handled by Clio [Popa et al. 2002], we provide algorithms for constructing target documents over which queries can be answered.

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

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

(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.

Example 2.1. 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 2.2. *Given a DTD D with $\text{SAT}(D) \neq \emptyset$, one can construct, in polynomial time, a consistent DTD D' such that $\text{SAT}(D) = \text{SAT}(D')$.*

Proof. Let A_D be an unranked nondeterministic finite tree automaton (UNFTA) that accepts $\text{SAT}(D)$ (see Appendix A for some basic facts about ranked and unranked tree automata). We assume that each transition is represented by an NFA, that is, $\delta(q, a)$ is an NFA over Q. For each ℓ in E, let A_ℓ be a constant-size UNFTA that tests if ℓ occurs in a tree. Then $L(A_D \times A_\ell)$ is nonempty iff there is a tree that conforms to D in which an ℓ-node occurs. Furthermore, this test can be done in polynomial time in the size of A_D (and hence of D).

When this procedure is applied to all symbols $\ell \in E$, we have them partitioned into two classes $E = E_1 \cup E_2$, where element types from E_1 appear in trees in

SAT(D) and those from E_2 do not. Then, for each regular expression r, construct an expression r' by replacing each $\ell \in E_2$ with the symbol $\emptyset$. Then define the following function ρ on regular expressions expanded with the $\emptyset$ symbol: $\rho(\emptyset) = \emptyset$, $\rho(\varepsilon) = \varepsilon$, $\rho(\ell) = \ell$, and

$$\rho(r_1 r_2) = \begin{cases} \rho(r_1)\rho(r_2) & \text{if } \rho(r_1) \neq \emptyset,\ \rho(r_2) \neq \emptyset \\ \emptyset & \text{otherwise;} \end{cases} \qquad \rho(r^*) = \begin{cases} \rho(r)^* & \text{if } \rho(r) \neq \emptyset \\ \varepsilon & \text{otherwise;} \end{cases}$$

$$\rho(r_1|r_2) = \begin{cases} \rho(r_1)|\rho(r_2) & \text{if } \rho(r_1) \neq \emptyset,\ \rho(r_2) \neq \emptyset \\ \rho(r_1) & \text{if } \rho(r_1) \neq \emptyset,\ \rho(r_2) = \emptyset \\ \rho(r_2) & \text{if } \rho(r_2) \neq \emptyset,\ \rho(r_1) = \emptyset \\ \emptyset & \text{if } \rho(r_1) = \rho(r_2) = \emptyset. \end{cases}$$

Now in D we eliminate all symbols $\ell \in E_2$ and for each $\ell \in E_1$, replace $r = P(\ell)$ with $\rho(r')$. Let D' be the resulting DTD. A straightforward induction shows that SAT(D) = SAT(D'), and the transformation was polynomial-time. Since D' only uses symbols from E_1, for each element type that occurs in D' there is a tree in SAT(D') that uses it, proving consistency. □

3. XML DATA EXCHANGE SETTINGS

Recall [Fagin et al. 2005; Fagin et al. 2005] 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 we 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 [Amer-Yahia et al. 2002; Benedikt et al. 2003].

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 [Amer-Yahia et al. 2002; Benedikt et al. 2003; Neven and Schwentick 2003; Deutsch and Tannen 2001; Wood 2003]. 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 [Deutsch and Tannen 2001; Neven and Schwentick 2003], we shall use variables; in our case, they will range over possible attribute values.

[1] In [Fagin et al. 2005; Fagin et al. 2005], STDs are written as FO sentences but here we prefer a rule-based formalism.

3.1 Tree-pattern formulae

The basic component of our language for source-to-target dependencies 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

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

where $\ell \in E \cup \{\_\}$, $@a_1, \ldots, @a_n \in A$ and $x_1, \ldots, x_n$ is a set of (non necessarily distinct) variables. In the second case, variables $x_1, \ldots, x_n$ are the free variables of α. For example, the formula $\ell(@a = x, @b = y)$ has free variables x and y, while the formula $\ell(@a = z, @b = z)$ has z as its only free variable. 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 $\sigma : \{x_1, \ldots, x_n\} \to \mathsf{Str}$ assigns to each variable x_i a string value ($i \in [1, n]$), then $(T, v) \models \ell(@a_1 = \sigma(x_1), \ldots, @a_n = \sigma(x_n))$ iff $(T, v) \models \ell$ and $\rho_{@a_j}(v) = \sigma(x_j)$, for every $j \in [1, n]$.

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

$$\varphi := \quad \alpha \quad | \quad \alpha[\varphi, \ldots, \varphi] \quad | \quad //\varphi,$$

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 the 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.

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.

3.2 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:*

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

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.*

EXAMPLE 3.4. 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

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

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 [Fagin et al. 2005; Fagin et al. 2005], 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 [Fagin et al. 2005; Fagin et al. 2005]. DTDs, being very close in expressiveness to monadic second-order logic [Vianu 2001], 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.

In this section, we investigate consistency of XML data exchange settings. We impose an additional restriction on tree patterns over the source that all variables used in each tree pattern formula are distinct[2]. The intuition behind this restriction is that we collect values of attributes that occur in patterns that only specify a tree structure, not statements about equality of attributes. For example, a tree pattern formula $\varphi = \ell(@a_1 = x, @a_2 = x)$ is satisfied in a node with attributes $@a_1$ and $@a_2$ that have the same value. Since our goal for source patterns is to collect values, we disallow formulae of this kind over the source, instead using formulae like $\varphi' = \ell(@a_1 = x_1, @a_2 = x_2)$, saying that the value of $@a_1$ should be recorded in x_1, and the value of $@a_2$ in x_2. Patterns like φ are essentially a conjunction of φ' and an equality statement $x_1 = x_2$, and we do not consider those over the source. We do, however, allow them over the target: while our goal is to collect values over the source, in populating target documents we can use values more than once. For example, φ could be used as a target formula: $\ell(@a_1 = x, @a_2 = x) \ :- \ \ell'(@a = x)$ says that for each value x of an attribute $@a$ of element type ℓ' in the source, we must have a node of type ℓ where both attributes $@a_1$ and $@a_2$ are assigned the same value x.

The proviso that variables in source tree-pattern formulae are distinct only applies in this section; we shall not need it when we deal with query answering.

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

[2]To avoid any potential confusion with the conference version of the paper caused by a slight notational difference, we should point out that in the conference version [Arenas and Libkin 2005], the distinct variables assumption was not spelled out as directly as here, and was enforced by using distinct variables in patterns.

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 [Lakshmanan et al. 2004] although precise complexity was not known. Results on XPath containment in the presence of DTDs [Neven and Schwentick 2003; Wood 2003] 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. 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 [Neven 2002].

More precisely, suppose we are given a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$. For each attribute formula α, define a formula α° as follows:

—if $\alpha = \ell$ with $\ell \in El \cup \{\_\}$, then $\alpha^\circ = \alpha$;
—if $\alpha = \ell(@a_1 = x_1, \ldots, @a_n = x_n)$, then $\alpha^\circ = \ell$.

For a tree formula φ, we let φ° denote a formula obtained from φ by replacing each attribute subformula α with α°. We define $\Sigma^\circ_{\mathbf{ST}}$ as the result of replacing each $\psi :- \varphi$ in $\Sigma_{\mathbf{ST}}$ with $\psi^\circ :- \varphi^\circ$. Notice that formulae of the form φ° do not have free variables.

CLAIM 4.2. *A data exchange setting* $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ *is consistent iff* $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma^\circ_{\mathbf{ST}})$ *is consistent.*

PROOF. One direction is immediate from the observation that $T \models \varphi^\circ$ iff $T \models \varphi(\bar{s})$ for some values $\bar{s}$. For the other direction, suppose $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma^\circ_{\mathbf{ST}})$ is consistent, with $T' \models D_{\mathbf{T}}$ being a solution for $T \models D_{\mathbf{S}}$. Fix a string s_0 and assign it as values of all attributes in T and T', resulting in trees $T(s_0)$ and $T'(s_0)$. Clearly, $\langle T(s_0), T'(s_0)\rangle$ witness the consistency of $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$. This proves the claim. □

We remark that the restriction on source tree-pattern formulae is essential here. Consider for example, a source DTD $\underline{r} \to \ell_0$, $\ell_0 \to \ell_1$, $\ell_1 \to \varepsilon$, with ℓ_i's all having an attribute $@a$, a target DTD with just the root $\underline{r}$, and an STD $\underline{r}[b] \;:-\; \underline{r}[\ell_0(@a = x)[\ell_1(@a = x)]]$. This setting is consistent: for example, a source tree with different values of $@a$ at ℓ_0 and ℓ_1 makes a witness, together with the only possible target. But $\Sigma^\circ_{\mathbf{ST}}$ is *not* consistent: it has a single STD $\underline{r}[b] \;:-\; \underline{r}[\ell_0[\ell_1]]$, and since every tree that conforms to the source DTD satisfies the pattern $\underline{r}[\ell_0[\ell_1]]$, the resulting setting is inconsistent.

Since the transformation from $\Sigma_{\mathbf{ST}}$ to $\Sigma^\circ_{\mathbf{ST}}$ is linear-time, we can assume without loss of generality that in our data exchange settings all attribute subformulae are of the form ℓ with $\ell \in El \cup \{\_\}$. To prove membership in EXPTIME, we need

the following result. Recall that an UFTA(DFA) is a unranked deterministic tree automaton where all transitions are represented by deterministic tree automata (see Appendix A for some basic facts about ranked and unranked tree automata).

CLAIM 4.3. *For every tree-pattern formula φ without free variables, one can compute in exponential time a deterministic UFTA(DFA) $\mathcal{A}_\varphi$ with a set F of accepting states such that for every tree T and a node v,*

$$run_{\mathcal{A}_\varphi}(v) \in F \quad \Leftrightarrow \quad v \text{ is a witness for } \varphi \text{ in } T.$$

PROOF. We construct $\mathcal{A}_\varphi$ inductively on the structure of φ. If $\varphi = \_$, the automaton has one state which is accepting. If $\varphi = \ell$, then it has an accepting state q_a and a rejecting state q_r and transitions $\delta(q_a, \ell) = \{q_a, q_r\}^*, \delta(q_r, \ell) = \emptyset, \delta(q_a, \ell') = \emptyset$ and $\delta(q_r, \ell') = \{q_a, q_r\}^*$ for every $\ell' \neq \ell$.

Now assume we have an automaton $\mathcal{A}_\varphi = (Q, \delta, F)$ for φ. The set of states of $\mathcal{A}_{//\varphi}$ will be $Q \times \{q_a^{//\varphi}, q_r^{//\varphi}, q_*^{//\varphi}\}$. The set of accepting states will be $Q \times \{q_a^{//\varphi}\}$. Intuitively, each run of the new automaton simulates a run of $\mathcal{A}_\varphi$ with the extra component being:

—$q_r^{//\varphi}$ until an accepting state of $\mathcal{A}_\varphi$ has been seen;
—$q_*^{//\varphi}$ once the run of $\mathcal{A}_\varphi$ enters an accepting state, assuming that the last component was $q_r^{//\varphi}$;
—$q_a^{//\varphi}$ once the last component $q_*^{//\varphi}$ has been seen.

It is straightforward to define this transition function; since it involves constructing products of given DFAs with a fixed automaton, it is done in polynomial time.

The last case is that of $\psi = \ell[\varphi_1, \ldots, \varphi_m]$. Assume that we have already constructed $\mathcal{A}_i = \mathcal{A}_{\varphi_i} = (Q_i, \delta_i, F_i)$ for each φ_i. The set of states of $\mathcal{A}_\psi$ will be $Q = Q_1 \times \ldots \times Q_m \times \{q_a^\psi, q_r^\psi\}$; the accepting states are those where the last component is q_a^ψ. The transition function of $\mathcal{A}_\psi$ is defined as follows. Let $\mathcal{A}_m$ be a deterministic string automaton over the alphabet Q which accepts strings s such that for each $i \leq m$, there is a letter $(q_1, \ldots, q_m, q) \in s$ with $q_i \in F_i$. We make it deterministic so that we could use both $\mathcal{A}_m$ and its complement. Then we simply let $\delta((\bar{q}, q_a), \ell)$ be given by $\mathcal{A}_m$, $\delta((\bar{q}, q_r), \ell)$ be given by $\mathcal{A}_m$'s complement, $\delta((\bar{q}, q_a), \ell')$ be empty, and $\delta((\bar{q}, q_r), \ell')$ be Q^* for all $\ell' \neq \ell$.

Clearly $\mathcal{A}_\psi$ enters a state $(\bar{q}, q_a)$ iff ψ holds in a given node. Thus, we have to show how to compute $\mathcal{A}_m$. Notice that our inductive construction shows that the number of states of each $\mathcal{A}_\varphi$ is [3] at most $3^{\|\varphi\|}$. The set of states of $\mathcal{A}_m$ is q_Y for Y ranging over subsets of $\{1, \ldots, m\}$. Being in q_Y means that so far φ_i with $i \in Y$, and only them, have been witnessed. If $\mathcal{A}_m$ is in q_Y and reads a letter $\bar{q}$ in which precisely q_j, $j \in Y'$, are in F_j's, then it enters $q_{Y \cup Y'}$. Such a transition table would have size $O(2^m \times 3^m)$ and clearly can be computed in exponential time by simply enumerating appropriate subsets. This concludes the proof of the claim. □

Suppose we have an automaton $\mathcal{A}_\varphi$ for a tree-pattern formula. Then we can straightforwardly transform $\mathcal{A}_\varphi$ into a UFTA(DFA) $\mathcal{A}(\varphi)$ such that $T \models \varphi$ iff

[3] We shall often use notation $\|\cdot\|$ applied to trees, automata, formulae, etc., referring to their size in some reasonable encoding (the exact encoding, up to a linear factor, is irrelevant).

$T \in L(\mathcal{A}(\varphi))$. This is done by letting $\mathcal{A}(\varphi)$ stay in an accepting state once an accepting state of $\mathcal{A}_\varphi$ has been seen; this can be done in polynomial time. Thus, $\mathcal{A}(\varphi)$ can also be constructed in exponential time in the size of φ.

We now show how to solve Data-Exchange-Consistency in EXPTIME. Let $\Sigma_{\mathbf{ST}}$ be $\{\psi_i :- \varphi_i \mid 1 \leq i \leq n\}$. Recall that these formulae are assumed not to have free variables. Let $\mathcal{A}_{\mathbf{S}}$ and $\mathcal{A}_{\mathbf{T}}$ be automata for source and target DTDs; those can be computed in polynomial time (see Appendix A). Then, for every subset $I \subseteq \{1, \ldots, n\}$ we do the following:

(1) Check if the product automaton

$$\mathcal{A}_{\mathbf{S}} \times \prod_{i \in I} \mathcal{A}(\varphi_i) \times \prod_{j \notin I} \bar{\mathcal{A}}(\varphi_j)$$

accepts a tree. Here $\bar{\mathcal{A}}$ refers to the automaton that accepts the complement of $L(\mathcal{A})$. This product automaton accepts a tree T iff $T \models D_{\mathbf{S}}$, $T \models \varphi_i$ for all $i \in I$, and $T \not\models \varphi_j$ for $j \notin I$.

(2) Check if the product automaton

$$\mathcal{A}_{\mathbf{T}} \times \prod_{i \in I} \mathcal{A}(\psi_i)$$

accepts a tree. This product automaton accepts a tree T' iff $T' \models D_{\mathbf{T}}$ and $T' \models \psi_i$ for all $i \in I$.

Then the data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent iff for some I, both of the above automata accept a tree (the pair of trees accepted by them witnesses consistency). Hence, it remains to verify that steps 1 and 2 above can be carried out in exponential time. It is known that checking nonemptiness of a product $\mathcal{A}_1 \times \ldots \times \mathcal{A}_k$ can be done in time $O(\|\mathcal{A}_1\| \times \ldots \times \|\mathcal{A}_k\|)$ [Comon et al. 2007]. We saw that each $\mathcal{A}(\varphi)$ can be constructed in time $O(c^{\|\varphi\|})$ for some constant c. Furthermore, $\bar{\mathcal{A}}(\varphi)$ can also be constructed in time $O(c^{\|\varphi\|})$ because $\mathcal{A}(\varphi)$ is deterministic and we simply have to reverse accepting and rejecting states. Thus, testing nonemptiness of the automaton in 1. can be done in time bounded by

$$c_1 \cdot \|D_{\mathbf{S}}\| \cdot \prod_{i \leq n} c^{\|\varphi_i\|} \quad = \quad O\big(2^{(\|D_{\mathbf{S}}\| + \|\Sigma_{\mathbf{ST}}\|)^k}\big)$$

for some appropriately chosen constants c_1 and k. Likewise, step 2. is also exponential in the size of $D_{\mathbf{T}}$ and $\Sigma_{\mathbf{ST}}$, thus proving membership in EXPTIME.

Next, we move to proving EXPTIME-hardness. For this, it suffice to reduce to the following problem: Given a nondeterministic finite tree automaton (NFTA) $\mathcal{A}$, is there a tree $T \notin L(\mathcal{A})$ [Seidl 1990]. The idea of the encoding is that $D_{\mathbf{S}}$ codes both a tree T and a run. In the target document we copy rejecting states of $\mathcal{A}$. The STDs ensure that our coding is correct, and consistency would tell us that every run on T has to end in a rejecting state.

Let $\Gamma = \{a_1, \ldots, a_k\}$ be the alphabet, and let $\mathcal{A} = (Q, q_0, \delta, F)$ where $Q = \{q_0, \ldots, q_n\}$, and $q_{i_1}, \ldots, q_{i_m}$ enumerate states in $Q \setminus F$ (rejecting states). Source and target DTDs are constructed by using the set of element types

$\Gamma \cup Q \cup \{\underline{r}, \underline{vr}, \underline{label}, \underline{right}, \underline{left}, \underline{leaf}, \underline{yes}, \underline{no}, f\}$. The source DTD $D_{\mathbf{S}}$ has no attributes and the regular expressions are as follows:

$$\begin{aligned}
\underline{r} &\rightarrow \underline{vr} \\
\underline{vr} &\rightarrow \underline{label}\ q_0 \dots q_n\ \underline{left}\ \underline{right} \ \mid\ \underline{label}\ q_0 \dots q_s\ \underline{leaf} \\
\underline{left} &\rightarrow \underline{vr} \\
\underline{right} &\rightarrow \underline{vr} \\
\underline{label} &\rightarrow a_1 \mid a_2 \mid \dots \mid a_k \\
a_i &\rightarrow \varepsilon, \quad i \leq k \\
q_j &\rightarrow \underline{yes} \mid \underline{no}, \quad j \leq n \\
\underline{leaf} &\rightarrow \varepsilon
\end{aligned}$$

The target DTD $D_{\mathbf{T}}$ is simply $\underline{r} \rightarrow q_{i_1}? \cdots q_{i_m}?$, with all the productions $q_{i_j} \rightarrow \varepsilon$. The intuition is that $\underline{vr}$-nodes of a tree that conforms to $D_{\mathbf{S}}$ form a binary tree over which we run $\mathcal{A}$; then q_i for a given node will lead to a $\underline{yes}$ if there is a run in which this node is in q_i. This is ensured by STDs saying that if we have two nodes in which states q_i and q_j respectively have children $\underline{yes}$, and their parent node has label a, and q_k has $\underline{no}$ as its child while $q_k \in \delta(a, q_i, q_j)$, then we enforce $\underline{r}[f]$ in the target which is inconsistent with the target DTD. That is, $\Sigma_{\mathbf{ST}}$ has an STD:

$$\underline{r}[f] \ \ :\!-\ \ \underline{vr}\big[q_k[\underline{no}],\ \underline{label}[a],\ \underline{left}[\underline{vr}[q_i[\underline{yes}]]],\ \underline{right}[\underline{vr}[q_j[\underline{yes}]]]\big].$$

Similarly we handle the case of leaves so that the $\underline{yes}$-states are those in which a leaf with a given label can be in a run of $\mathcal{A}$, that is, for every $q_i \in \delta(a, q_0, q_0)$, $\Sigma_{\mathbf{ST}}$ has an STD:

$$\underline{r}[f] \ \ :\!-\ \ \underline{vr}\big[q_i[\underline{no}],\ \underline{label}[a],\ \underline{leaf}\big].$$

Finally, $\Sigma_{\mathbf{ST}}$ has an STD

$$\underline{r}[q_i] \ \ :\!-\ \ \underline{r}\big[\underline{vr}[q_i[\underline{yes}]]\big].$$

that copies every "yes" state of the root into the target.

Assume that the setting is consistent, and we have a pair $(\mathbf{T}, T')$ that witnesses its consistency. Let T be the binary tree of nodes of type $\underline{vr}$ extracted from $\mathbf{T}$. For each node of T, the corresponding q_i sibling that can be assigned to that node of T in some run of $\mathcal{A}$ will always have a $\underline{yes}$ child (otherwise an inconsistent f child would have been forced in the target). Thus, the $\underline{yes}$-states of the root include all the possible states in which $\mathcal{A}$ is when it reaches the root, and since there is a solution for $\mathbf{T}$, we conclude that they all are rejecting. Hence, every run on T ends in a rejecting state, meaning that $T \notin L(\mathcal{A})$.

Conversely, suppose we have a tree $T \notin L(\mathcal{A})$. Then extend it to a tree $\mathbf{T}$ that conforms to $D_{\mathbf{S}}$ simply by annotating T with all the runs of $\mathcal{A}$ on T. But then such a $\mathbf{T}$ has a solution in the data exchange setting: since $T \notin L(\mathcal{A})$ every run coded in $\mathbf{T}$ ends in a rejecting state and, thus, a target tree that has all the rejecting states as children of the root is a solution for $\mathbf{T}$. Therefore, our data exchange setting, which was constructed in polynomial time from $\mathcal{A}$, is consistent iff there is a tree not accepted by $\mathcal{A}$. This completes the proof of the hardness case. □

The proof of Theorem 4.1 shows that DATA-EXCHANGE-CONSISTENCY remains EXPTIME-complete even if all formulae in all STDs have no free variables. In fact, this problem remains intractable under some other 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 the grammar:

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

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 DATA-EXCHANGE-CONSISTENCY($D_{\mathbf{T}}$) 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 (the proof of this proposition is given in Appendix B.1).

PROPOSITION 4.4. *Fix an arbitrary nonrecursive DTD $D_{\mathbf{T}}$ that does not use the Kleene star. Then:*

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

b) The problem DATA-EXCHANGE-CONSISTENCY($D_{\mathbf{T}}$) *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.*

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 [Miller et al. 2001; Popa et al. 2002]. 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 [Abiteboul et al. 2001].

THEOREM 4.5. DATA-EXCHANGE-CONSISTENCY *is solvable in polynomial time if both source and target DTDs are nested-relational. Specifically, it can be solved in time $O(nm^2)$, where n is the size of the DTDs and m is the size of the source-to-target dependencies.*

PROOF. As in the proofs of Theorem 4.1 and Proposition 4.4, we assume without loss of generality that the DTDs do not have attributes and all formulae in STDs have no free variables (see Claim 4.2). We consider two linear-time transformations of DTDs. Recall that $\tilde{\ell}$ is one of ℓ, or $\ell?$, or ℓ^+, or ℓ^*. For each nested-relational

DTD D, we define DTDs D° and D^* in which each $\tilde{\ell}$ is replaced according to the following rules:

In D $\tilde{\ell}$ is	replace in D° by	replace in D^* by
ℓ	ℓ	ℓ
$\ell?$	ε	ℓ
ℓ^+	ℓ	ℓ
ℓ^*	ε	ℓ

We now need the following result.

CLAIM 4.6. *If $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$ are nested relational, then $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent iff $(D^\circ_{\mathbf{S}}, D^*_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent.*

PROOF. If D is nested relational and $T \models D^\circ$ or $T \models D^*$, then $T \models D$. Hence, if $(D^\circ_{\mathbf{S}}, D^*_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent, then $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent.

For the opposite direction, assume that $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent, and let $\langle T_1, T_2\rangle$ witness it: that is, $T_1 \models D_{\mathbf{S}}, T_2 \models D_{\mathbf{T}}$, and $\langle T_1, T_2\rangle \models \Sigma_{\mathbf{ST}}$. Let $T \models D_{\mathbf{S}}$ and let T° be the tree obtained from T as follows: if v is a node of T labeled ℓ, its parent is labeled ℓ', and ℓ occurs in $P(\ell')$ as $\ell?$ or ℓ^* (one of two cases corresponding to replacement by ε in the definition of D°), we delete the entire subtree rooted at v. This transformation results in a tree T° that conforms to $D^\circ_{\mathbf{S}}$. Furthermore, if $T^\circ \models \varphi$, where φ is a tree formula, then $T \models \varphi$ (tree formulae are monotone which can be shown by a straightforward inductive argument or by translation into conjunctive FO queries). This implies that $\langle T_1^\circ, T_2\rangle \models \Sigma_{\mathbf{ST}}$: if ψ :– φ is an STD from $\Sigma_{\mathbf{ST}}$ and $T_1^\circ \models \varphi$, then $T_1 \models \varphi$ and from $\langle T_1, T_2\rangle \models \Sigma_{\mathbf{ST}}$ we conclude $T_2 \models \psi$. Hence, we proved that $(D^\circ_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent.

Next, let D_ℓ denote the restriction of D to element types reachable from ℓ in $G(D)$; in particular, ℓ becomes the root of D_ℓ. (Recall that $G(D)$ is the graph of D: there is an edge from ℓ to ℓ' iff ℓ' occurs in a string in the language given by $P(\ell)$.) We write $\mathrm{cons}(D, \varphi)$ if there is tree $T \models D$ such that $T \models \varphi$. We now show that if D is nested relational, then for every tree-pattern formula φ and every ℓ,

$$\mathrm{cons}(D_\ell, \varphi) \quad \text{iff} \quad \mathrm{cons}(D^*_\ell, \varphi). \tag{3}$$

Notice that this implies that $(D^\circ_{\mathbf{S}}, D^*_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent since $(D^\circ_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent and there is only one tree conforming to $D^*_{\mathbf{T}}$.

We prove (3) by induction on ℓ starting with those that do not have outgoing edges in $G(D)$, and in each step using another inner induction on φ. Notice that since $T \models D^*$ implies $T \models D$, one has to verify only the implication $\mathrm{cons}(D_\ell, \varphi) \Rightarrow \mathrm{cons}(D^*_\ell, \varphi)$. Suppose ℓ does not have any outgoing edges in $G(D)$; that is, D contains $\ell \to \varepsilon$. In this case $D_\ell = D^*_\ell$. Now assume that we have an element type ℓ such that D contains

$$\ell \to \ell_1^1? \ldots \ell_{n_1}^1? \, (\ell_1^2)^+ \ldots (\ell_{n_2}^2)^+ \, (\ell_1^3)^* \ldots (\ell_{n_3}^3)^* \, \ell_1^4 \ldots \ell_{n_4}^4,$$

such that the equivalence (3) holds for all the ℓ_j^i's. In the DTD D^* we have

$$\ell \to \ell_1^1 \ldots \ell_{n_1}^1 \, \ell_1^2 \ldots \ell_{n_2}^2 \, \ell_1^3 \ldots \ell_{n_3}^3 \, \ell_1^4 \ldots \ell_{n_4}^4,$$

The proof of the equivalence (3) is now by induction on the formulae. For formulae _ and ℓ', where $\ell' \in El$, this is immediate. For formulae $//\varphi$ this follows from

the induction hypothesis and the observation that the regular expressions corresponding to ℓ in D and D^* contain exactly the same element types. Now let φ be $\ell'[\varphi_1, \ldots, \varphi_k]$. Assume that $\ell' \neq \ell$ and $\ell' \neq \_$. Then ℓ' must be one of ℓ^i_j's or a descendant of one of these nodes to ensure consistency and the statement is true by the induction hypothesis. The case of $\ell' = \_$ is simply a disjunction of the cases $\ell' = \ell$ (proved below) and $\ell' = \ell^i_j$ or ℓ' is a descendant of ℓ^i_j. Thus, the remaining case to consider is $\ell' = \ell$. Suppose T is a tree that witnesses $\text{cons}(D_\ell, \varphi)$. Then the root v of T is of type ℓ, and it has children $v_1, \ldots, v_k$ (not necessarily distinct) that witness $\varphi_1, \ldots, \varphi_k$, respectively. Let v_m be labeled by ℓ_m (where ℓ_m is one of $\ell^i_j, i \leq 4, j \leq n_i$). Since we have $\text{cons}(D_{\ell_m}, \varphi_m)$, by the induction hypothesis we have $\text{cons}(D^*_{\ell_m}, \varphi_m)$. But $D^*_{\ell_m}$ has only one tree that conforms to it. Thus, if in T we replace each subtree rooted at a child of v of type ℓ_m by the tree that conforms to $D^*_{\ell_m}$, we obtain a tree T_1 which satisfies φ. But now subtrees of T_1 rooted at any two children of v that are labeled ℓ_m are identical and hence we can keep only one of them for each label, and still satisfy φ. This results in a tree T_2 that satisfies φ. We note that T_2 does not necessarily conform to D^*_ℓ since it could be the case that for some element type ℓ^i_j, the root of T_2 does not have any children of type ℓ^i_j. It is easy to solve this problem to generate from T_2 a tree T_3 conforming to D^*_ℓ and satisfying φ. Hence, we proved $\text{cons}(D^*_\ell, \varphi)$.

This concludes the proof of (3) and thus proves the claim. □

Using this lemma, we can prove the theorem as follows. First, construct $D^\circ_{\mathbf{S}}$ and $D^*_{\mathbf{T}}$ in linear time. In them, all the rules are of the form $\ell \to \ell_1 \ldots \ell_k$, where all ℓ_i's are distinct, or $\ell \to \varepsilon$, and hence each admits only one tree (since they are non-recursive).

If D is an arbitrary DTD of the above form, T is the only tree that conforms to it, and φ is a tree-pattern formula, then one can check $T \models \varphi$ in time $O(\|D\| \cdot \|\varphi\|^2)$. This can be seen by induction on the structure of φ. Specifically, as in the proof of Proposition 4.4, we enumerate all the subformulae φ' of φ and for each element type ℓ that occurs in D we verify that the formula is true in the subtree whose root is that element.

That is, we keep, with each node, an array of m Boolean values, where m is the number of subformulae of φ. For the basis step, formulae ℓ and $\_$ are true in a node labeled ℓ. If we have a formula $//\varphi'$, we search the graph of the DTD $G(D)$ and for each node from which a node with φ' being true is reachable we put a 1 into the array position corresponding to $//\varphi'$. This takes linear time in the size of D. For the formula $\ell[\varphi_1, \ldots, \varphi_k]$, we compute, for each element type ℓ', the Boolean *or* of the arrays associated with the element types $\ell_1 \ldots \ell_s$ where $\ell' \to \ell_1 \ldots \ell_s$ is the rule in the DTD for ℓ'. If the result has 0 in at least one position corresponding to $\varphi_1, \ldots, \varphi_k$, we put 0 in the position corresponding to φ in the array for ℓ'. Otherwise, if the result has 1 in all positions corresponding to $\varphi_1, \ldots, \varphi_k$, we put 1 in the position corresponding to $\ell[\varphi_1, \ldots, \varphi_k]$ in the array associated with ℓ' if $\ell' = \ell$ or $\ell = \_$, and 0 otherwise. Thus, for each subformula of φ of the form $\ell[\varphi_1, \ldots, \varphi_k]$, we have to take a disjunction of k arrays. Hence, the complexity is quadratic in the size of the formula.

Let $T_{\mathbf{S}}$ be the tree that conforms to $D^\circ_{\mathbf{S}}$ and $T_{\mathbf{T}}$ the tree that conforms to $D^*_{\mathbf{T}}$. To verify consistency of $(D^\circ_{\mathbf{S}}, D^*_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$, we check for each STD ψ :– φ in $\Sigma_{\mathbf{ST}}$ if

$T_{\mathbf{S}} \models \varphi$ and $T_{\mathbf{T}} \models \psi$. This is done in time $O\big((\|D_{\mathbf{S}}\| + \|D_{\mathbf{T}}\|) \cdot (\|\varphi\|^2 + \|\psi\|^2)\big)$. The setting is consistent iff there is no STD such that $T_{\mathbf{S}} \models \varphi$ and $T_{\mathbf{T}} \not\models \psi$. Hence consistency is checked in $O\big((\|D_{\mathbf{S}}\| + \|D_{\mathbf{T}}\|) \cdot \|\Sigma_{\mathbf{ST}}\|^2\big)$. □

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 [Arenas et al. 2004; Fagin et al. 2005], and most positive results have been proved for conjunctive or monotone queries [Fagin et al. 2005; Fagin et al. 2005]. 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 [Gottlob et al. 2006], 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 $\exists y\, \mathit{book}(@\mathit{title} = x)[\mathit{author}(@\mathit{name} = y)]$. Then the source document from the introduction, shown in Figure 1 (b), satisfies $\psi(\mathit{Computational\ Complexity})$. 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 [Fagin et al. 2005; Fagin et al. 2005], since there may be many possible solutions to the data exchange problem, we define the semantics of Q in terms of *certain answers*:

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

Thus, a tuple $\bar{s}$ of strings is in $\underline{\mathit{certain}}(Q, T)$ if $\bar{s} \in Q(T')$ for every solution T' for T. If Q is a Boolean query (a sentence), then $\underline{\mathit{certain}}(Q, T) = \mathit{true}$ if and only if 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 $T \not\approx D$, 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 $T \not\models D$ 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 $T' \not\approx D_{\mathbf{T}}$ and $\langle T, T' \rangle$ satisfies[4] 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 $T \not\models D_{\mathbf{S}}$, a $\mathcal{CTQ}^{//,\cup}$-query $Q(\bar{x})$ over $D_{\mathbf{T}}$ 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 $T \not\models D_{\mathbf{S}}$ 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

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

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 $T \not\approx D$. Then one can compute, in polynomial time in the size of T, a local sibling ordering $\prec_{\text{sib}}$ on T such that $T^{\prec_{\text{sib}}} \models D$.*

Before proving this proposition, we need to establish 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 programming, which is in NP in general, and in polynomial time if dimension is fixed [Lenstra 1983]. 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.*

PROOF. Clearly checking if $w \in \pi(r)$ is in NP: we simply guess a permutation w' of w and verify if it is in $L(r)$. For NP-hardness, we reduce from a simplified version of Integer Linear Programming (ILP), which is known to be NP-complete [Garey and Johnson 1979]. The input consists of an $n \times m$ matrix $\mathbf{A}$ and an n-vector $\vec{b}$ of 0s and 1s; the question is whether there is an m-vector $\vec{x}$ of 0s and 1s such that $\mathbf{A}\vec{x} = \vec{b}$. More precisely, let Γ be an n-element alphabet $\sigma_1, \ldots, \sigma_n$. With the ith column $(a_{1,i}, \ldots, a_{n,i})$ of $\mathbf{A}$ we associate a string

$$r_i \;=\; \sigma_{j_1} \cdots \sigma_{j_\ell},$$

where $\{j_1, \ldots, j_\ell\}$ is the set of all indexes $j \in [1, n]$ for which $a_{j,i} = 1$. Then we define

$$r_{\mathbf{A}} \;=\; r_1^* \cdots r_m^*.$$

As the string w, we choose

$$w \;=\; a_{k_1} \cdots a_{k_\ell},$$

where $\{k_1, \ldots, k_\ell\}$ is the set of all indexes $k \in [1, n]$ for which $b_k = 1$. Then $\mathbf{A}\vec{x} = \vec{b}$ implies $w \in \pi(r_1^{x_1} \cdots r_m^{x_m}) \subseteq \pi(r_{\mathbf{A}})$. Conversely, if $w \in \pi(r_{\mathbf{A}})$, then for some $\vec{x}$ we have $w \in \pi(r_1^{x_1} \cdots r_m^{x_m}) \subseteq \pi(r_{\mathbf{A}})$ and thus $\mathbf{A}\vec{x} = \vec{b}$. Hence the instance of ILP has a solution iff $w \in \pi(r)$, proving NP-hardness.

Finally, we show that if r is fixed, then checking whether w is in $\pi(r)$ can be done in polynomial time. To prove this we need to use what is known as Pilling normal form [Kozen 2002]:

LEMMA 5.4. *For every regular expression r over an alphabet Γ, there exist regular expressions s_1, …, s_n such that $\pi(r) = \pi(s_1| \cdots |s_n)$ and each s_i ($i \in [1, n]$) is of the form $w_0(w_1)^* \cdots (w_m)^*$, where each $w_j \in \Gamma^*$ ($j \in [0, m]$).*

If r is fixed, then we can compute in polynomial time the regular expressions s_1, …, s_n mentioned in Lemma 5.4. Thus, if we show that it is possible to check in polynomial time whether $w \in \pi(s)$ with s being of the form $w_0 w_1^* \ldots w_k^*$, then we conclude that it is possible to check in polynomial time whether $w \in \pi(r)$. Let $\#_a(w)$ be the number of occurrences of symbol a in a string w. Then $w \in$

$\pi(w_0 w_1^* \dots w_k^*)$ if there exist nonnegative integers $x_1, \dots, x_k$ such that for every alphabet symbol a:

$$\#_a(w) \;=\; \#_a(w_0) \;+\; \sum_{i=1}^{k} x_i \cdot \#_a(w_i).$$

Thus, to check $w \in \pi(w_0 w_1^* \dots w_k^*)$ we need to solve an instance of ILP, but in this time the dimension k is fixed. But it is well-known that ILP is solvable in PTIME in fixed dimension [Lenstra 1983]. □

Proof of Proposition 5.2. Clearly it suffices to show that for each fixed regular expression r, the following can be solved in polynomial time in the size of string w: assuming that $w \in \pi(r)$, find a permutation w' of w such that $w' \in L(r)$. Let $\Gamma = \{\sigma_1, \dots, \sigma_m\}$ be the alphabet of r. Let $\mathcal{A}_r$ be an NFA for r, with Q being its set of states. Let r_q be the regular expression such that $L(r_q)$ consists of exactly the strings which are accepted by $\mathcal{A}_r$ in which q becomes the new initial state. It follows from Proposition 5.3 that there is a polynomial p such that for every string w' and every $q \in Q$ one can test whether w' is in $\pi(r_q)$ in time $p(|w'|)$.

Let n be the length of w. The algorithm works as follows. At each step i of the algorithm, we have a string w_i of length i, a string w^i of length $n - i$, and a state q_i of $\mathcal{A}$ such that

(1) $w_i w^i$ is a permutation of w;

(2) there is a run of $\mathcal{A}$ on w_i that ends in q_i, and

(3) $\mathcal{A}_{q_i}$ accepts a permutation of w^i.

The algorithm starts with $w_0 = \varepsilon$, $w^0 = w$, and q_0 being the initial state of $\mathcal{A}$. Then $\mathcal{A}_{q_0} = \mathcal{A}$ accepts a permutation of w by the assumption that $w \in \pi(r)$. At the end, we have a permutation w_n of w and a run of $\mathcal{A}$ on it that ends in a state q_n such that $\mathcal{A}_{q_n}$ accepts ε: that is, q_n is a final state of $\mathcal{A}$. In other words, w_n is a permutation of w that belongs to $L(r)$. Thus, an algorithm satisfying 1, 2, and 3 will correctly compute a permutation of w.

At each step of the algorithm, we do the following. For each letter σ_j present in w^i, we check if there is a state q such that $q \in \delta_{\mathcal{A}}(q_i, \sigma_j)$ and a string $w^i_{\sigma_j}$ obtained from w^i by eliminating one occurrence of σ_j belongs to $\pi(r_q)$. By assumption 3, for at least one state q this condition will be satisfied. We then take $w_{i+1} := w_i \sigma_j$, $w^{i+1} = w^i_{\sigma_j}$, and $q_{i+1} = q$. Conditions 1, 2, 3 are clearly satisfied. It remains to check the complexity. For each i, we have to check whether $w^i_{\sigma_j}$ is in $\pi(r_q)$. This by the assumption can be done in time $p(n)$. The number of such tests is at most $|\Gamma| \times |Q|$ and hence does not depend on n. Therefore the overall running time is $O(np(n))$. This proves the proposition. □

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 5.5. *If Q is a $\mathcal{CTQ}^{//,\cup}$-query, then* CERTAIN-ANSWERS*(Q) is in* coNP.

PROOF. Fix a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ and a query Q. To prove that the complexity of finding certain answers is in coNP it suffices to show that there exists a polynomial p that depends only on the data exchange setting and on Q such that the following is true. If for a tree T and a tuple $\bar{s}$ of strings from Const it is the case that $\bar{s} \notin \underline{certain}(Q, T)$, then there exists a solution T' for T such that $\bar{s} \notin Q(T')$ and the size of T' is at most $p(\|T\|)$ where $\|T\|$ is the size of T. Indeed, then checking whether $\bar{s} \notin \underline{certain}(Q, T)$ is in NP, and hence CERTAIN-ANSWERS(Q) is in coNP. In fact, by Proposition 5.1, it suffices to show that there exists an unordered tree T' that weakly conforms to $D_{\mathbf{T}}$ and such that $\langle T, T'\rangle \models \Sigma_{\mathbf{ST}}$, $\|T'\| \leq p(\|T\|)$ and $\bar{s} \notin Q(T')$. We prove the existence of such a polynomial-size tree in two steps: we first show that all the paths in the tree are of polynomial size and then we reduce the size of a tree without long paths.

Now for an arbitrary tree T, a node v, a tree-pattern formula $\varphi(\bar{x})$ and a tuple $\bar{a}$, we inductively define a set $\text{witness}_v(T, \varphi, \bar{a})$ of witnesses to $\varphi(\bar{a})$ starting at v as follows.

—If v does not witness $T \models \varphi(\bar{a})$, then $\text{witness}_v(T, \varphi, \bar{a}) = \emptyset$.

—Otherwise:

—if φ is an attribute formula, then $\text{witness}_v(T, \varphi, \bar{a}) = \{v\}$;

—if $\varphi = //\varphi'$ and v' is an arbitrarily chosen descendant of v such that v' witnesses φ', then $\text{witness}_v(T, \varphi, \bar{a}) = \{v, v', v''\} \cup \text{witness}_{v'}(T, \varphi', \bar{a})$, where v'' is the child of v lying between v and v';

—if $\varphi = \alpha[\varphi_1, \ldots, \varphi_k]$ with each φ_i being witnessed by a child v_i of v, then

$$\text{witness}_v(T, \varphi, \bar{a}) \;=\; \{v\} \;\cup\; \bigcup_{i=1}^{k} \text{witness}_{v_i}(T, \varphi_i, \bar{a}).$$

Intuitively, we collect all the nodes at which φ and its subformula are witnessed, and with each descendant subformula, we also include the child on the way to that descendant. Clearly the cardinality of $\text{witness}_v(T, \varphi, \bar{a})$ is linear in the size of φ. We then define $\text{witness}(T, \varphi, \bar{a})$ as $\text{witness}_v(T, \varphi, \bar{a})$ for an arbitrarily chosen witness v for $T \models \varphi(\bar{a})$ (and $\emptyset$ if no witness exists). Note that the definition of $\text{witness}(T, \varphi, \bar{a})$ is nondeterministic but it suffices to pick an arbitrary one.

Let T' be a solution to T in the data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$. For each STD $\psi(\bar{x}, \bar{z}) \text{ :- } \varphi(\bar{x}, \bar{y})$ in $\Sigma_{\mathbf{ST}}$, and for each $\bar{a}, \bar{b}$ such that $T \models \varphi(\bar{a}, \bar{b})$, we have a tuple $\bar{c}_{\varphi(\bar{a},\bar{b})}$ such that $\psi(\bar{a}, \bar{c}_{\varphi(\bar{a},\bar{b})})$ holds in T'. We then define

$$\text{witness}_{\Sigma_{\mathbf{ST}},T}(T') \;=\; \bigcup_{\psi(\bar{x},\bar{z}) \text{ :- } \varphi(\bar{x},\bar{y}) \,\in\, \Sigma_{\mathbf{ST}}} \;\; \bigcup_{\bar{a},\bar{b} \,:\, T \models \varphi(\bar{a},\bar{b})} \text{witness}(T', \psi, \bar{a}\bar{c}_{\varphi(\bar{a},\bar{b})}).$$

If $\Sigma_{\mathbf{ST}}$ and T are clear from the context we write just $\text{witness}(T')$. This set collects all the witnesses to satisfaction of all the formulae that have to hold in the solution T' due to the source-to-target constraints. We finally define $\text{witness}^*(T')$ to contain $\text{witness}(T')$ and all the greatest lower bounds for all the subsets of $\text{witness}(T')$. Since T' is a tree, the size of $\text{witness}^*(T')$ is at most quadratic in the size of

witness(T'). Furthermore, the size of witness(T') is polynomial in the size of T, and hence the size of witness*(T') is polynomial in the size of T.

Let T and T' be two XML trees, V a subset of nodes of T. We call a map $h : T \to T'$ a *V-preserving embedding* if h is one-to-one map that preserves labels and all attribute values of nodes, preserves the descendant relation, and, furthermore, preserves the child relation restricted to V. That is, if v' is a child of v in T and $v, v' \in V$, then $h(v')$ is a child of $h(v)$ in T'.

From now on, when we say "solution", we mean a weak solution, that is a tree T' that weakly conforms to the target DTD and satisfies all STDs. The following claim is immediate from the definitions and will be used several times in the proof.

CLAIM 5.6. *Suppose T' is a solution for T. Assume that a tree T'' conforms to $D_{\mathbf{T}}$ and has a subset V of nodes and a V-preserving embedding $h : T'' \to T'$ such that $h(V) = \text{witness}^*(T')$. Then T'' is a solution for T.*

Notice that Claim 5.6 holds because V witnesses right-hand sides of all the STDs. In particular, $V = \text{witness}^*(T'')$.

Suppose $\bar{s} \notin \underline{certain}(Q, T)$ for a tuple $\bar{s}$ from Const. Then there is a solution T_0 such that $\bar{s} \notin Q(T_0)$. Our first lemma lets us restrict the length of paths in a solution.

LEMMA 5.7. *There exists a polynomial p_0 that depends on the data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ and the query Q such that, for every tree T and a solution T_0 satisfying $\bar{s} \notin Q(T_0)$, one can construct another solution T_1 satisfying $\bar{s} \notin Q(T_1)$ in which all paths have length at most $p_0(\|T\|)$.*

PROOF. Since $Q(\bar{x})$ is a $\mathcal{CTQ}^{//,\cup}$ query, it is a union of queries of the form $\exists \bar{y} \ \bigwedge_i \psi_i(\bar{x}, \bar{y})$, with all the ψ_i's being tree-pattern formulae. We let $\beta_1(\bar{x}, \bar{y}), \ldots, \beta_m(\bar{x}, \bar{y})$ enumerate all the tree-pattern formulae and their subformulae that occur in Q. Let m' be the number of element types used in the target DTD. We define M to be $2^m \cdot m' + 1$. Let $V_0 = \text{witness}^*_{\Sigma_{\mathbf{ST}}, T}(T_0)$. Construct a tree T_0' in which all nodes except those in V_0 are given new attribute values, which are fresh and distinct values from Var. Clearly T_0' is still a solution because V_0 witnesses all the STDs. Also notice that $\bar{s} \notin Q(T_0')$. Indeed, otherwise we would have $T_0' \models \exists \bar{y} \bigwedge_i \beta_i(\bar{s}, \bar{y})$ for some collection of β_i's and thus $T_0' \models \bigwedge_i \beta_i(\bar{s}, \bar{c}_0')$ for some tuple $\bar{c}_0'$. Then, if $\bar{c}_0$ is a tuple obtained by changing, in $\bar{c}_0'$, newly created attribute values in T_0' to those they replaced, we would have $T_0 \models \bigwedge_i \beta_i(\bar{s}, \bar{c}_0)$, contradicting $\bar{s} \notin Q(T_0)$.

Now consider an arbitrary path $v_1 \ldots v_p$ of length $p \geq M + 5$ in T_0' such that the only descendants of v_1 that belong to V_0 are also descendants of v_p. In other words, all nodes from V_0 in the subtree rooted at v_1 are descendants of the last node on the path, v_p. For each node v_i, let $B(v_i) \subseteq \{1, \ldots, m\}$ be the set of indexes j such that v_i has a child at which $\exists \bar{y} \beta_j(\bar{s}, \bar{y})$ holds. Because of the bound on M, there exist two indexes $2 < i_1 < i_2 < p - 2$ such that:

$$\lambda_{T_0'}(v_{i_1}) = \lambda_{T_0'}(v_{i_2}) \quad \text{and} \quad B(v_{i_1}) = B(v_{i_2}).$$

Let $T_0'(v_{i_1} \leftarrow v_{i_2})$ be the tree that results from replacing the tree rooted at v_{i_1} with the tree rooted at v_{i_2}. Because of our assumption on the path, we have that $V_0 = \text{witness}^*(T_0'(v_{i_1} \leftarrow v_{i_2}))$. Furthermore, $T_0'(v_{i_1} \leftarrow v_{i_2})$ still conforms to $D_{\mathbf{T}}$

since $\lambda_{T_0'}(v_{i_1}) = \lambda_{T_0'}(v_{i_2})$ and, hence, it is a solution for T. Finally, $\bar{s} \notin Q(T_0'(v_{i_1} \leftarrow v_{i_2}))$. Indeed, if v is a witness for some tree-pattern formula $\beta_i(\bar{s}, \bar{y}_0)$, $i \leq m$, in the tree $T_0'(v_{i_1} \leftarrow v_{i_2})$, then v would be a witness for $\beta_i(\bar{s}, \bar{y}_1)$ in T_0', where $\bar{y}_1$ differs from $\bar{y}_0$ only in positions corresponding to values from Var that appear only once in T_0'. A simple induction on formulae then shows that $\bar{s} \in Q(T_0'(v_{i_1} \leftarrow v_{i_2}))$ would imply $\bar{s} \in Q(T_0')$.

Thus, every path $v_1 \dots v_p$ in T_0' satisfying the conditions that $p \geq M + 5$ and the only descendants of v_1 that belong to V_0 are also descendants of v_p can be shortened in such a way that the resulting tree is still a solution and $\bar{s}$ is not in the output of Q on it. Applying this inductively, we obtain a tree in which every path of length $\geq M+5$ has an intermediate node with a descendant in V_0 that is not a descendant of the last node on the path. Since M does not depend on T, this shows that we can have a solution T_1 such that $\bar{s} \notin Q(T_1)$ and all paths in T_1 are of length at most linear in $|V_0|$, and thus polynomial in $\|T\|$. This proves the lemma. □

Now we need the last two ingredients to complete the proof of membership in coNP. Recall that $\#_a(w)$ is the number of occurrences of symbol a in a string w. Given strings w_1 and w_2, we say that $w_1 \preceq w_2$ if $\#_a(w_1) \leq \#_a(w_2)$, for all alphabet symbols a. Furthermore, given a regular expression r, we define $\|r\|$ as follows. If $r = \varepsilon$, then $\|r\| = 0$. If $r = a$, where a is an element type, then $\|r\| = 1$. If either $r = r_1|r_2$ or $r = r_1 r_2$, then $\|r\| = \|r_1\| + \|r_2\|$. Finally, if $r = r_1^*$, then $\|r\| = \|r_1\|$.

LEMMA 5.8. *Let r be a regular expression of the form $r_1|\dots|r_m$, where each r_i ($i \in [1, m]$) is of the form $w_0(w_1)^* \cdots (w_n)^*$, being w_j a string ($j \in [0, n]$), and let $p_r(x)$ be polynomial $\|r\| \cdot (x+1)$. If $w_0 \preceq w$ and $w \in \pi(r)$, then there exists a string w' such that $w_0 \preceq w' \preceq w$ and $|w'| \leq p_r(|w_0|)$.*

PROOF. Assume without loss of generality that $w \in \pi(r_1)$ and $r_1 = u_0 u_1^* \cdots u_n^*$. Since $w \in \pi(r_1)$, there exist natural numbers $\ell_1, \dots, \ell_n$ such that $w = u_0 u_1^{\ell_1} \cdots u_n^{\ell_n}$. Define q_i as $\min\{\ell_i, |w_0|\}$, for every $i \in [1, n]$, and w' as $u_0 u_1^{q_1} \cdots u_n^{q_n}$. It is easy to see that $w_0 \preceq w' \preceq w$. Furthermore, $w' \in \pi(r_1)$ and $|w'| \leq |u_0| + \sum_{i=1}^{n} |u_i| \cdot |w_0| \leq |u_0| \cdot (|w_0| + 1) + \sum_{i=1}^{n} |u_i| \cdot (|w_0| + 1) \leq \|r_1\| \cdot (|w_0| + 1) \leq \|r\| \cdot (|w_0| + 1)$. This concludes the proof of the lemma. □

Since in this proof the target DTD $D_{\mathbf{T}}$ is assumed to be fixed, in what follows we assume, by Lemma 5.4, that every regular expression in $D_{\mathbf{T}}$ is of the form mentioned in the statement of Lemma 5.8.

Notice that in the absence of elements of $V_0 = \text{witness}^*_{\Sigma_{\mathbf{ST}}, T}(T_0)$, the proof of Lemma 5.7 and Lemma 5.8 imply the following statement.

LEMMA 5.9. *Let D be an arbitrary DTD, and $\beta_i(\bar{x}, \bar{y}), i \in I$, a collection of tree-pattern formulae. Then one can find a number N that depends on D and the collection $\{\beta_i\}$ such that for every tuple $\bar{s}'$ from Str, if there is a tree T such that $T \models D$ and $T \not\models \exists \bar{y} \beta_i(\bar{s}', \bar{y})$ for all $i \in I$, then there is a tree T' with this property such that the size of T' is at most N.*

PROOF. Let T be a tree such that $T \models D$ and $T \not\models \exists \bar{y} \beta_i(\bar{s}', \bar{y})$ for all $i \in I$. If we just disregard points in the witness set in the proof of 5.7, then we obtain a tree T'' such that $T'' \models D$, $T'' \not\models \exists \bar{y} \beta_i(\bar{s}', \bar{y})$ for all $i \in I$, and all paths in T'' are of length at most N', where N' is a constant that depends only on D and the collection $\{\beta_i\}$.

Then in T'' we look at every node v to construct tree T'. Let v_1, ..., v_p be the children of v, $w = \lambda_{T''}(v_1) \cdots \lambda_{T''}(v_p)$ and $w_0 = a_1 \cdots a_t$, where $\{a_1, \ldots, a_t\}$ is the set of alphabet symbols mentioned in w. By Lemma 5.8, we can assume, without loss of generality, that there is $t' \leq p_r(t) = \|r\| \cdot (t+1) \leq (\|r\|+1)^2$, where r is the regular expression corresponding to $\lambda_{T''}(v)$, such that

$$w_0 \preceq \lambda_{T''}(v_1) \cdots \lambda_{T''}(v_{t'}) \in \pi(r).$$

Thus, if we remove subtrees rooted at $v_{t'+1}, \ldots, v_p$, the resulting tree still conforms to D. Let T' now denote the tree obtained by applying this procedure to all nodes in T''. It is easy to see that T' conforms to D. Furthermore, by monotonicity of each formula β_i, we conclude that $T' \not\models \exists \bar{y} \beta_i(\bar{s}', \bar{y})$ for all $i \in I$ (otherwise we would have $T'' \models \exists \bar{y} \beta_i(\bar{s}', \bar{y})$).

Given that for each node v in T', the number of children of v is at most $(\|r\|+1)^2$, where r is the regular expression in D corresponding to $\lambda_{T'}(v)$, and given that the length of each path in T' is at most N', we conclude that there exists a constant N that depends only on D and the collection $\{\beta_i\}$ such that the size of T' is at most N. This concludes the proof of the lemma. □

We now prove the existence of a polynomial-size solution (in the size of T) that witnesses $T' \not\models Q(\bar{s})$. Suppose a solution T_0 is given in which $Q(\bar{s})$ does not hold. Then, by Lemma 5.7, we replace it by a solution T_1 in which all paths are of length at most $p_0(\|T\|)$ such that $\bar{s} \notin Q(T_1)$. Let $V_1 = \text{witness}^*(T_1)$. We know that the size of V_1 is polynomial in $\|T\|$. Next, by Lemma 5.9, we replace every subtree rooted at a node v that does not contain an element of V_1 by a fixed-size subtree in such a way that for the resulting tree, say T_2, it is still the case that $\bar{s} \notin Q(T_2)$. This can be done simply by making sure that none of subformulae used in Q becomes true in the new fixed-size tree. Finally, in T_2 we look at every node v such that the subtree rooted at v contains elements of V_1. Let v_1, ..., v_p be the children of v and assume, without loss of generality, that v_1, ..., v_t, $t \leq p$, are the only children of v having as descendants nodes in V_1. Let $w_0 = \lambda_{T_2}(v_1) \cdots \lambda_{T_2}(v_t)$ and $w = \lambda_{T_2}(v_1) \cdots \lambda_{T_2}(v_p)$. By Lemma 5.8, we can assume, without loss of generality, that there is $t' \leq p_r(t)$, where r is the regular expression corresponding to $\lambda_{T_2}(v)$, such that

$$w_0 \preceq \lambda_{T_2}(v_1) \cdots \lambda_{T_2}(v_{t'}) \in \pi(r).$$

Thus, if we remove subtrees rooted at $v_{t'+1}, \ldots, v_p$, the resulting tree still conforms to $D_{\mathbf{T}}$. Let T_3 now denote the tree obtained by applying this procedure to all nodes in T_3 that are in V_1 or have a descendant in V_1.

The set V_1 still belongs to T_3, and T_3 conforms to $D_{\mathbf{T}}$, and thus it is a solution for T. Furthermore, by monotonicity of Q we have $\bar{s} \notin Q(T_3)$ (otherwise we would have $\bar{s} \in Q(T_2)$). Thus, it remains to calculate the size of T_3.

In T_3, the following holds: (1) every path is of length at most $p_0(\|T\|)$; (2) every node that is in V_1 or has a descendant in V_1 has at most $p_{\mathbf{T}}(|V_1|)$ children, where $p_{\mathbf{T}}$ is the maximum of p_r given by Lemma 5.8 over all regular expressions r used in $D_{\mathbf{T}}$, and (3) every subtree rooted at a node that is not in V_1 and does not have a descendant in V_1 has size bounded by a fixed number N. Thus, the size of T_3 is bounded by

$$O\big(|V_1| \cdot p_0(\|T\|) \cdot p_{\mathbf{T}}(|V_1|) \cdot N\big),$$

and hence is polynomial in $\|T\|$ since so is the cardinality of $V_1 = \text{witness}^*_{\Sigma_{\mathbf{ST}},T}(T_3)$. Thus, we have a solution T_3 of polynomial size which witnesses $\bar{s} \notin \underline{certain}(Q,T)$, which shows that CERTAIN-ANSWERS is in coNP. This concludes the proof of Theorem 5.5. □

We would like to identify tractable cases of the CERTAIN-ANSWERS problem. We note that in a a source-to-target dependency $\psi_{\mathbf{T}}(\bar{x},\bar{z}) \text{ :- } \varphi_{\mathbf{S}}(\bar{x},\bar{y})$, the source formula $\varphi_{\mathbf{S}}$ is used to extract data from a source tree T, while the target formula $\psi_{\mathbf{T}}$ shows how to structure data under the target DTD. Hence, the complexity of computing certain answers 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 CERTAIN-ANSWERS even for very simple DTDs.

DEFINITION 5.10. *A source-to-target dependency* $\psi_{\mathbf{T}}(\bar{x},\bar{z}) \text{ :- } \varphi_{\mathbf{S}}(\bar{x},\bar{y})$ *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, …, 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 5.11. *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* CERTAIN-ANSWERS(Q) *is* coNP*-complete, even if all regular expressions used in source and target DTDs are simple.*

PROOF. For the sake of readability, here we consider only the case of STD(_, //), and the other two cases are considered in Section B.2.

We define a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ and a Boolean $\mathcal{CTQ}$-query Q such that both $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$ are simple DTDs, $\Sigma_{\mathbf{ST}}$ is a set of source-to-target dependencies in STD(_, //) and 3SAT can be reduced to the complement of CERTAIN-ANSWERS(Q), that is, for every propositional formula θ in 3-CNF, there exists a PTIME constructible XML tree T_θ conforming to $D_{\mathbf{S}}$ such that θ is satisfiable if and only if $\underline{certain}(Q,T_\theta) = \mathit{false}$. Simple DTD $D_{\mathbf{S}}$ is defined as follows. Let

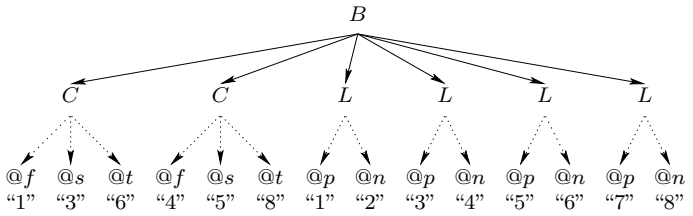


Fig. 3. XML tree T_θ, defined in the proof of Theorem 5.11, representing propositional formula $\theta = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$.

$E_{\mathbf{S}} = \{K, C, L\}$ be a set of element types and $A_{\mathbf{S}} = \{@f, @s, @t, @p, @n\}$ be a set of attributes. Then $D_{\mathbf{S}} = (P_{\mathbf{S}}, R_{\mathbf{S}}, K)$ is a DTD over $(E_{\mathbf{S}}, A_{\mathbf{S}})$, where $P_{\mathbf{S}}$ is defined as:

$$P_{\mathbf{S}}(K) = C^*L^*, \qquad P_{\mathbf{S}}(C) = \varepsilon, \qquad P_{\mathbf{S}}(L) = \varepsilon.$$

and $R_{\mathbf{S}}$ is defined as:

$$R_{\mathbf{S}}(K) = \emptyset, \qquad R_{\mathbf{S}}(C) = \{@f, @s, @t\}, \qquad R_{\mathbf{S}}(L) = \{@p, @n\}.$$

XML trees conforming to $D_{\mathbf{S}}$ are used to represent propositional formulae. Let θ be 3-CNF formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$. To construct tree T_θ, first we assign a distinct natural number to each literal, say

$$\begin{array}{llll} x_1 \mapsto 1, & x_2 \mapsto 3, & x_3 \mapsto 5, & x_4 \mapsto 7, \\ \neg x_1 \mapsto 2, & \neg x_2 \mapsto 4, & \neg x_3 \mapsto 6, & \neg x_4 \mapsto 8. \end{array}$$

Then we represent each clause of θ as a node of type C, being the values of attributes $@f$, $@s$, $@t$ the first, second and third literal of that clause, respectively. For each propositional variable x in θ, we use the attributes $@p$, $@n$ of a node of type L to store the values assigned to x and $\neg x$, respectively. Tree T_θ is shown in Figure 3.

Simple DTD $D_{\mathbf{T}}$ is defined as follows. Let $E_{\mathbf{T}} = \{K, L, G_1, G_2, G_3, H_1, H_2, H_3\}$ be a set of element types and $A_{\mathbf{T}} = \{@\ell, @p, @n\}$ a set of attributes. Then $D_{\mathbf{T}} = (P_{\mathbf{T}}, R_{\mathbf{T}}, K)$ is a DTD over $(E_{\mathbf{T}}, A_{\mathbf{T}})$, where $P_{\mathbf{T}}$ is defined as:

$$\begin{array}{llllll} P_{\mathbf{T}}(K) = G_1^*L^*, & P_{\mathbf{T}}(G_1) = H_1^*G_2^*, & P_{\mathbf{T}}(H_1) = H_2^*, \\ P_{\mathbf{T}}(H_2) = H_3^*, & P_{\mathbf{T}}(H_3) = \varepsilon, & P_{\mathbf{T}}(G_2) = H_1^*G_3^*, \\ P_{\mathbf{T}}(G_3) = H_1^*, & P_{\mathbf{T}}(L) = \varepsilon. \end{array}$$

and $R_{\mathbf{T}}$ is defined as:

$$\begin{array}{llllll} R_{\mathbf{T}}(K) = \emptyset, & R_{\mathbf{T}}(G_1) = \emptyset, & R_{\mathbf{T}}(H_1) = \{@\ell\}, \\ R_{\mathbf{T}}(H_2) = \{@\ell\}, & R_{\mathbf{T}}(H_3) = \{@\ell\}, & R_{\mathbf{T}}(G_2) = \emptyset, \\ R_{\mathbf{T}}(G_3) = \emptyset, & R_{\mathbf{T}}(L) = \{@p, @n\}. \end{array}$$

Finally, $\Sigma_{\mathbf{ST}}$ is defined as follows. The first rule of $\Sigma_{\mathbf{ST}}$ is defined as:

$$K[L(@p = x, @n = y)] \ :- \ K[L(@p = x, @n = y)].$$

This rule says that every node of type L in a source tree T must appear in every solution for T. The second rule of $\Sigma_{\mathbf{ST}}$ is defined as:

$$H_1(@\ell = x)[H_2(@\ell = y)[H_3(@\ell = z)]] \ :- \ K[C(@f = x, @s = y, @t = z)].$$

Notice that this rule is not fully-specified since it does not say whether the parent of H_1 is a node of type either G_1 or G_2 or G_3. Also notice that in this rule we use neither descendant // nor wildcard _.

The previous rule says that for every C-node v of a source tree T, the values i, j, k of attributes $@f$, $@s$, $@t$ of v must appear in every solution for T in a subtree of the form shown in Figure 4 (a). For example, every solution for tree T_θ shown in Figure 3 must have a subtree of the form shown in Figure 4 (b) since T_θ has a C-node with values 1, 3, 6 in attributes $@f$, $@s$, $@t$.

When constructing a solution for T_θ, we are actually constructing a truth assignment for θ. For example, let v be the C-node of T_θ with values 1, 3, 6 in attributes $@f$, $@s$, $@t$. To construct a solution T' for T we have to instantiate the second dependency of $\Sigma_{\mathbf{ST}}$ on values 1, 3 and 6, and then we have to construct a subtree of the form shown in Figure 4 (b) and place it in T', that is, we have to choose the type of the parent of the node of type H_1. The three alternatives for the type of this parent are shown in Figures 4 (c), (d) and (e). These alternatives represent three different ways of satisfying the clause stored in the children of v. We say that a literal i has been assigned value 1 if i is the value of attribute $@\ell$ of a great-grandchild of a node of type G_1. Thus, in Figure 4 (c), literal 6 (corresponding to $\neg x_3$) has been assigned value 1 since 6 is the value of attribute $@\ell$ a great-grandchild (of type H_3) of a node of type G_1. On the other hand, in Figure 4 (d), literal 3 (corresponding to x_2) has been assigned value 1 since 3 is the value of attribute $@\ell$ of a great-grandchild (of type H_2) of a node of type G_1, and in Figure 4 (e), literal 1 (corresponding to x_1) has been assigned value 1 since 1 is the value of attribute $@\ell$ of a great-grandchild (of type H_1) of a node of type G_1. Notice that when constructing a truth assignment for θ, it is possible to choose value 1 for two complementary literals. For example, we can choose value 1 for x_2 when considering the first clause in T_θ, and we can choose value 1 for $\neg x_2$ when considering the second clause in this tree. To take care of this problem we use the following Boolean $\mathcal{CTQ}$-query Q:

$$\exists x \exists y \, (L(@p = x, @n = y) \ \wedge \ G_1[\_[\_[\_(@\ell = x)]]] \ \wedge \ G_1[\_[\_[\_(@\ell = y)]]]).$$

Intuitively, query Q says that there exists two complementary literals x and y such that both x and y have been assigned value 1, that is, x and y are both values of attribute $@\ell$ of great-grandchildren of nodes of type G_1. Notice that if Q does not hold, then we have constructed a well defined truth assignment.

Now we prove that for every 3-CNF propositional formula θ, we have that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = \mathit{false}$, where T_θ is constructed from θ in PTIME as shown above.

($\Rightarrow$) Assume that θ is satisfiable and let σ be a truth assignment satisfying θ. Define a solution T' for T_θ as follows. The structure of the L-nodes of T' is copied from T_θ. The structure of the G_1-nodes of T' is defined as follows. For every C-node v in T_θ having i, j, k in attributes $@f$, $@s$, $@t$, let T_v be the tree shown in Figure 4 (a), and let v' be the node identifier of the root of this tree (v' is of type

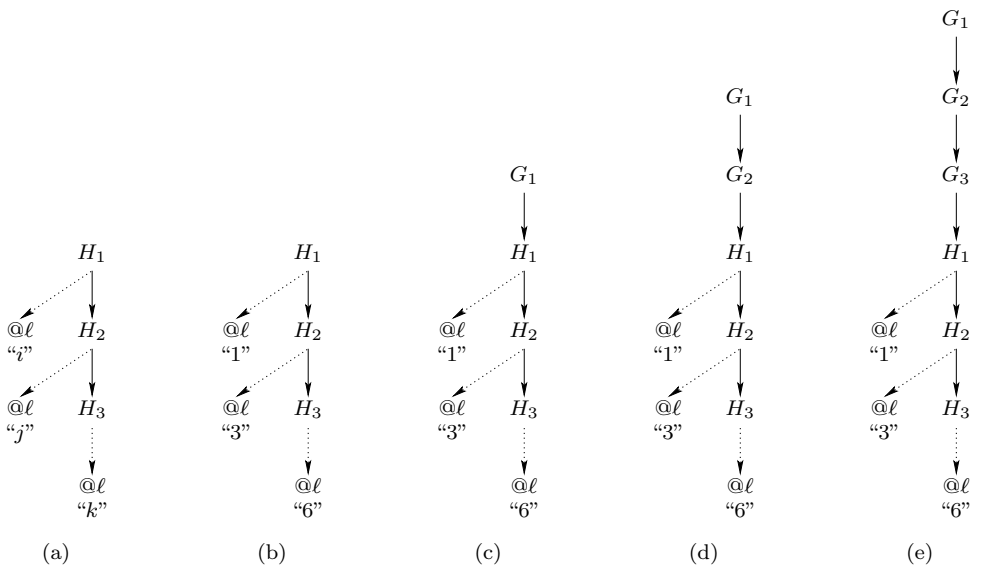


Fig. 4. Different alternatives for satisfying the second rule of $\Sigma_{\mathbf{ST}}$ in case STD($\_$, //) of the proof of Theorem 5.11.

H_1). To place T_v into T', we have to decide what is the type of the parent of v'. If σ makes true the third literal of the clause stored in v, then T' has a node of type G_1 having v' as its only child. If σ makes true the second literal of the clause stored in v, then T' has a node of type G_1 having only one child (of type G_2) and v' as its only grandchild. If σ makes true the first literal of the clause stored in v, then T' has a node of type G_1 having only one child (of type G_2), only one grandchild (of type G_3) and v' as its only great-grandchild. It is straightforward to prove that T' conforms to $D_{\mathbf{T}}$ and satisfies $\Sigma_{\mathbf{ST}}$. Furthermore, $T' \not\models Q$ since σ is well defined and, therefore, for every propositional variables x, either x or $\neg x$ is not assigned value 1. We conclude that $\underline{certain}(Q, T_\theta) = \mathit{false}$ since T' is a solution for T_θ.

($\Leftarrow$) Assume that $\underline{certain}(Q, T_\theta) = \mathit{false}$ and let T' be a solution for T_θ such that $T' \not\models Q$. We define a truth assignment for the propositional variables of θ as follows. For every clause in θ, find the values i, j, k assigned in T_θ (as values of attributes $@f$, $@s$ and $@t$) to the literals of that clause. Then find in T' a subtree of the form shown in Figure 4 (a). Let v' be the root of this tree. If the parent of v' is of type G_1, then σ assigns value 1 to the third literal of the clause. If the parent of v' is of type G_2, then σ assigns value 1 to the second literal of the clause. If the parent of v' is of type G_3, then σ assigns value 1 to the first literal of the clause. Since $T' \not\models Q$, we have that σ is well defined. Thus θ is satisfiable since σ satisfies this formula by definition. This concludes the proof of the theorem. □

Thus, from now one we concentrate on fully-specified STDs. We note that STDs handled by Clio [Miller et al. 2001; Popa et al. 2002] are fully-specified. 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 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* 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* 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*.

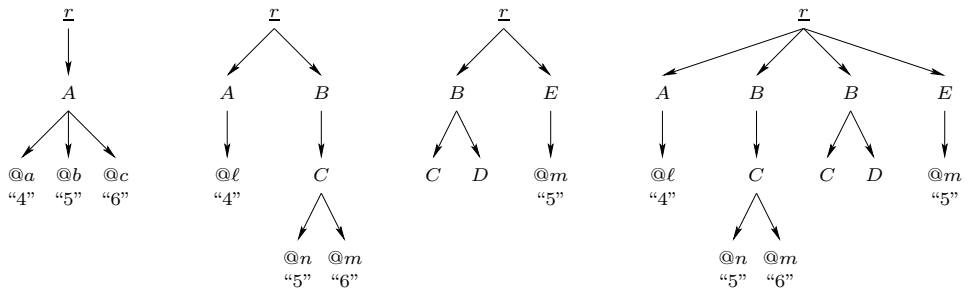
THEOREM 6.2. **(Dichotomy)** *Let $\mathcal{C}$ be an admissible class of regular expressions and $\mathcal{Q}$ be one of $\mathcal{CTQ}$, $\mathcal{CTQ}^{//}$, $\mathcal{CTQ}^{\cup}$ and $\mathcal{CTQ}^{//,\cup}$. Then $\mathcal{C}$ is either tractable, or strongly* coNP*-complete for $\mathcal{Q}$-queries.*

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 checks whether there exists a solution for T, and if this holds then 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 prove this result and present 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 a *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.



(a) T (b) $T_{\psi_1(4,5,6)}$ (c) $T_{\psi_2(5)}$ (d) $cps(T)$

Fig. 5. Construction of the canonical pre-solution for T.

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 $i \leq k$.

Recall that an STD $\psi_{\mathbf{T}}(\bar{x}, \bar{z}) \coloneq \varphi_{\mathbf{S}}(\bar{x}, \bar{y})$ 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 _. Now we introduce the notion of canonical pre-solution. Given a source tree T conforming to $D_{\mathbf{S}}$, the *canonical pre-solution for* T, denoted by $cps(T)$, is an XML tree defined as follows. Let X be a set of XML trees such that for every fully-specified STD $\psi_{\mathbf{T}}(\bar{x}, \bar{z}) \coloneq \varphi_{\mathbf{S}}(\bar{x}, \bar{y})$ in $\Sigma_{\mathbf{ST}}$ and for every pair of tuples $\bar{s}$, $\bar{s}'$ of strings such that $|\bar{s}| = |\bar{x}|$, $|\bar{s}'| = |\bar{y}|$ and $T \models \varphi_{\mathbf{S}}(\bar{s}, \bar{s}')$, we have that X includes $T_{\psi_{\mathbf{T}}(\bar{s}, \bar{s}'')}$, where $\bar{s}''$ is an arbitrary tuple of length $|\bar{z}|$ of fresh distinct null values. Assume that $X = \{T_1, \ldots, T_m\}$ and let v_1, ..., v_m be the root nodes of T_1, ..., T_m. Then $cps(T)$ is generated from T_1, ..., T_m by replacing v_1, ..., v_m by a single root node v_0.

Example 6.3. Assume $\Sigma_{\mathbf{ST}}$ contains rules $\psi_1(x, y, z) \coloneq \varphi(x, y, z)$ and $\psi_2(y) \coloneq \varphi(x, y, z)$, where:

$$\begin{aligned} \psi_1(x, y, z) &= \underline{r}[A(@\ell = x),\ B[C(@n = y, @m = z)]], \\ \psi_2(y) &= \underline{r}[B[C, D], E(@m = y)], \\ \varphi(x, y, z) &= \underline{r}[A(@a = x, @b = y, @c = z)], \end{aligned}$$

and assume that T is the source tree shown in Figure 5 (a). To construct $cps(T)$, we instantiate the variables x, y, z in the right hand sides of the STDs on values 4, 5 and 6, respectively, and we generate the XML trees associated to $\psi_1(4, 5, 6)$ and $\psi_2(5)$, shown in Figures 5 (b) and (c). Then we merge the roots of $T_{\psi_1(4,5,6)}$ and $T_{\psi_2(5)}$ into a single node, generating the canonical pre-solution for T, shown in Figure 5 (d). □

Canonical pre-solutions can be computed in PTIME; the problem is that they may not conform to the target DTD. For example, if $D_{\mathbf{T}}$ contains a rule $P_{\mathbf{T}}(\underline{r}) =$

$(ABE)^*$, then the canonical pre-solution shown in Figure 5 (d) does not conform to $D_{\mathbf{T}}$. 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 $\{@a \mid \rho_{@a}(v) \text{ is defined in } T'\} \neq R_{\mathbf{T}}(\lambda_{T'}(v))$ or $\lambda_{T'}(\mathit{children}(v)) \notin \pi(P_{\mathbf{T}}(\lambda_{T'}(v)))$, where $\lambda_{T'}(\mathit{children}(v))$ 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 **ChangeAtt** receives as parameters a target tree T' and a node v such that $\{@a \mid \rho_{@a}(v) \text{ is defined in } T'\} \neq R_{\mathbf{T}}(\lambda_{T'}(v))$. 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, **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 **ChangeReg** (defined later) tries to repair these violations: It receives as parameters a target tree T' and a node v such that $\lambda_{T'}(\mathit{children}(v)) \notin \pi(P_{\mathbf{T}}(\lambda_{T'}(v)))$, and it either fails or returns a tree T'' such that $\lambda_{T''}(\mathit{children}(v)) \in \pi(P_{\mathbf{T}}(\lambda_{T''}(v)))$.

Functions **ChangeAtt** and **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.

EXAMPLE 6.4. Let $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$ be the DTDs shown in Figures 6 (a) and (b). Notice that in the latter DTD, $P_{\mathbf{T}}(\underline{r}) = (BC)^*$, $P_{\mathbf{T}}(B) = \varepsilon$, $P_{\mathbf{T}}(C) = D$, $R_{\mathbf{T}}(B) = \{@m\}$ and $R_{\mathbf{T}}(C) = \emptyset$. Assume that T is the source tree, conforming to $D_{\mathbf{S}}$, shown in Figure 6 (c), and assume that $\Sigma_{\mathbf{ST}}$ is given by the fully-specified STD $\underline{r}[B(@m = x)] \text{ :- } A(@\ell = x)$. Then the canonical pre-solution for T is the tree shown in Figure 6 (d), and a canonical solution for T is shown in Figure 6 (e). □

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

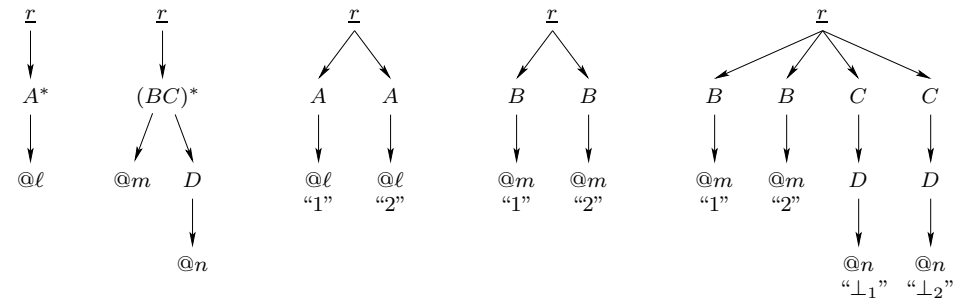
LEMMA 6.5. *If $D_{\mathbf{T}}$ is a $\mathcal{C}_U$-DTD, then for every source tree T:*

(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 **ChangeAtt** and **ChangeReg**.

LEMMA 6.6. *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.*



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

Fig. 6. 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.

By putting together these two lemmas we obtain:

PROPOSITION 6.7. *$\mathcal{C}_U$ is tractable for $\mathcal{CTQ}^{//,\cup}$-queries.*

Now we define the class $\mathcal{C}_U$, explain how **ChangeReg** works and then prove Lemmas 6.5 and 6.6. 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 $a \in alph(w)$, recall that $\#_a(w)$ is 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 $\pi(r)$:

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

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:

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

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 $b \in alph(w)$.

Once **ChangeReg** has constructed $rep(w, r)$, it replaces w by a string w' from $rep(w, r)$. In general, $rep(w, r)$ will have more than one element, so we need a criterion to decide which string is the "best" candidate. For example, $ccdd$ and cd belong to $rep(cc, (cd)^*(cde)^*)$. Which one is better? If we decide to merge two

nodes into a single one, we have to put their attributes together. But this is not possible if these nodes have different constants values on the same attribute. Thus, we prefer candidates that merge as less elements as possible to avoid attribute clashes. For example, we prefer *ccdd* to *cd*, since *ccdd* does not merge any element type. Furthermore, we do not want to add element types if we are not force to do it and, hence, we prefer *ccdd* to *ccdde*. We formalize our preference relation as a preorder $\preceq_w$. Formally, $w_1 \preceq_w w_2$ iff (1) $\#_b(w_2) \geq \min\{\#_b(w_1), \#_b(w)\}$ for all $b \in \mathit{alph}(w)$, and (2) $\mathit{alph}(w_2) \setminus \mathit{alph}(w) \subseteq \mathit{alph}(w_1) \setminus \mathit{alph}(w)$. Thus, **ChangeReg** replaces w by $w' \in \max_{\preceq_w} \mathit{rep}(w, r)$.

The canonical solution for a source tree must be unique, no matter which string **ChangeReg** picks from $\max_{\preceq_w} \mathit{rep}(w, r)$ 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) $\max_{\preceq_w} \mathit{rep}(w, r)$ 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 $a \in \mathit{alph}(r)$, 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 $r = a \,|\, aab^*$, 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,

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

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

LEMMA 6.8. *$c(r)$ is finite for every r.*

PROOF. Let r be a regular expression and $a \in \mathit{alph}(r)$. It is enough to prove that there exists a natural number k such that $c_a(r) \leq k$. By Lemma 5.4, there exist regular expressions $s_1, \ldots, s_n$ such that $\pi(r) = \pi(s_1|\cdots|s_n)$ and each s_i $(i \in [1, n])$ is of the form $w_0(w_1)^* \cdots (w_m)^*$, where each w_j is a string $(j \in [0, m])$. Assume that $s_i = w_i(w_{i,1})^* \cdots (w_{i,m_i})^*$ $(i \in [1, n])$, where w_i, $w_{i,1}$, …, w_{i,m_i} are strings over $\mathit{alph}(r)$. We will show that $c_a(r) \leq \max_{i \in [1,n]} \#_a(w_i)$.

By contradiction, assume that either $\{\ell \mid$ there exists $w \in \mathit{fixed}_a(r)$ such that $\#_a(w) = \ell\}$ is unbounded or $c_a(r) > \#_a(w_i)$, for every $i \in [1, n]$. In either case, there exists a string $w \in \mathit{fixed}_a(r)$ such that $\#_a(w) > \#_a(w_i)$, for every $i \in [1, n]$. Since $w \in \pi(r)$ and $\pi(r) = \pi(s_1|\cdots|s_n)$, there exists $j \in [1, n]$ such that $w \in \pi(s_j)$. Thus, w is a permutation of a string $w_j w'$, where w' is in the regular language defined by $(w_{j,1})^* \cdots (w_{j,m_j})^*$. Given that $\#_a(w) > \#_a(w_j)$, we have that $a \in \mathit{alph}(w')$ and, therefore, $a \in \mathit{alph}((w_{j,1})^* \cdots (w_{j,m_j})^*)$. Hence, string $ww_{j,1}w_{j,2}\cdots w_{j,m_j}$ is in $\pi(s_j) \subseteq \pi(r)$ and $\#_a(w) < \#_a(ww_{j,1}w_{j,2}\cdots w_{j,m_j})$, which contradicts the fact that $w \in \mathit{fixed}_a(r)$. This concludes the proof of the lemma. □

DEFINITION 6.9. *We say that a regular expression r is* univocal *if $c(r) \leq 1$ and for every string w such that $\mathit{rep}(w, r) \neq \emptyset$, the set $\mathit{rep}(w, r)$ has a maximum element with respect to $\preceq_w$: that is, an element $w' \in \mathit{rep}(w, r)$ such that $w'' \preceq_w w'$ for all $w'' \in \mathit{rep}(w, r)$. We write $\mathcal{C}_U$ for the class of univocal regular expressions.*

For example, all of the following are univocal regular expressions: $bc^+d^*e?$, $(b^*|c^*)$ and $(bc)^*(de)^*$. It is easy to see that all simple regular expressions are univocal, and hence $\mathcal{C}_U$ is an admissible class.

PROPOSITION 6.10. *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.*

PROOF. Assume that $alph(r) = \{\sigma_1, \ldots, \sigma_n\}$. By Lemma 5.4 we know that $\pi(r) = \pi(s_1|\ldots|s_m)$, where each s_i ($i \in [1,m]$) is of the form $w_0(w_1)^* \cdots (w_\ell)^*$, where each w_j is a string ($j \in [0,\ell]$). Let $i \in [1,m]$ and assume that $s_i = w_0 w_1^* \cdots w_\ell^*$. In the proof of Proposition 5.3, we showed that for each s_i, there exists an $n \times \ell$ matrix $\mathbf{A}_i$ of non-negative integers such that for every string w, we have that $w \in \pi(s_i)$ iff there is an ℓ-vector $\vec{x}$ of nonnegative integers such that $\mathbf{A}_i\vec{x} + \vec{b} = \vec{c}$, where $\vec{b}$ and $\vec{c}$ are n-vectors with $b_j = \#_{\sigma_j}(w_0)$ and $c_j = \#_{\sigma_j}(w)$ ($j \in [1,n]$). Since $\mathbf{A}_i$ depends only on s_i, there exists a formula $\varphi_i(x_1, \ldots, x_n)$ of Presburger Arithmetic such that for every string w, we have

$$w \in \pi(s_i) \text{ iff } \varphi_i(\#_{\sigma_1}(w), \ldots, \#_{\sigma_n}(w)) \text{ holds.}$$

We conclude that formula $\varphi_r(x_1, \ldots, x_n) = \bigvee_{i=1}^m \varphi_i(x_1, \ldots, x_n)$ is such that a string w is in $\pi(r)$ iff $\varphi_r(\#_{\sigma_1}(w), \ldots, \#_{\sigma_n}(w))$ holds.

Now simply by examining the definition of univocality we notice that we only refer to number of occurrences of symbols in strings, and the definition itself can be stated in first-order logic using φ_r; hence the result follows. □

Summing up, if $D_{\mathbf{T}}$ is a $\mathcal{C}_U$-DTD, then **ChangeReg**(T', v) is defined as shown in Figure 7 (for the sake of completeness, we also include function **ChangeAtt** in Figure 7). Recall that $\ell = \lambda_{T'}(v)$, $w = \lambda_{T'}(children(v))$ and $r = P_{\mathbf{T}}(\ell)$. Initially, **ChangeReg** checks whether $rep(w,r)$ is empty. If this is the case, then it fails. Otherwise, **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 **ChangeReg** adds $(q-p)$ new children to v of type b, each of them having no attributes and no children[5]. If $q < p$, then $q = 1$ (since r is univocal) and, thus, **ChangeReg** replaces the sequence v_1, …, 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 **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 $\rho_{@a}(v_i) \neq \rho_{@a}(v_j)$.

In the rest of this section, we prove Lemmas 6.5 and 6.6 that are used to show that the algorithm presented in this section is correct and it can be implemented in polynomial time, for every fixed target DTD $D_{\mathbf{T}}$ containing only univocal regular expressions. But before doing this, we note that all regular expressions used in nested-relational DTDs are univocal. Hence, the following extension of relational data exchange handled by Clio [Popa et al. 2002] falls in the following large tractable case:

[5]Violations generated by adding b-nodes without attributes or children are repaired later by repeatedly applying **ChangeAtt** and **ChangeReg**.

ChangeAtt(T : tree, v : node identifier)
  if there exists $@a \in A_{\mathbf{T}} \setminus R_{\mathbf{T}}(\lambda_T(v))$ such that $\rho_{@a}(v)$ is defined **then** fail
  else for every $@a \in R_{\mathbf{T}}(\lambda_T(v))$ **do**
    if $\rho_{@a}(v)$ is not defined in T **then** assign to $\rho_{@a}(v)$ a fresh null value
  return(T)

ChangeReg(T : tree, v : node identifier)
  $\ell := \lambda_T(v)$
  $w := \lambda_T(\mathit{children}(v))$ /* if v has no children, then $w = \varepsilon$ */
  if $\mathit{rep}(w, P_{\mathbf{T}}(\ell)) = \emptyset$ **then** fail
  else
    Choose $w' \in \max_{\preceq_w} \mathit{rep}(w, P_{\mathbf{T}}(\ell))$
    for every $b \in \mathit{alph}(P_{\mathbf{T}}(\ell))$ **do**
      if $\#_b(w) < \#_b(w')$ **then** add $(\#_b(w') - \#_b(w))$ new children of type b to v
      else if $\#_b(w) > \#_b(w')$ **then**
        $v_1, \ldots, v_k :=$ sequence of node identifiers of the children of v of type b
        if there exists $@a \in R(b)$ and $i, j \in [1, k]$ such that
          $\rho_{@a}(v_i) \in \mathsf{Const}$, $\rho_{@a}(v_j) \in \mathsf{Const}$ and $\rho_{@a}(v_i) \neq \rho_{@a}(v_j)$ **then** fail
        else
          Replace $v_1, \ldots, v_k$ by a single fresh node identifier v'
          for every $@a \in R(b)$ **do**
            if there exists $i \in [1, k]$ such that $\rho_{@a}(v_i) \in \mathsf{Const}$ **then** $\rho_{@a}(v') := \rho_{@a}(v_i)$
  return(T)

Fig. 7. Rules for constructing a chase sequence.

COROLLARY 6.11. *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 also note 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.

To prove Lemmas 6.5 and 6.6, we need to introduce some terminology and prove some intermediate results. In particular, we need to introduce the notion of chase sequence.

Given an XML tree T over $(E_{\mathbf{T}}, A_{\mathbf{T}})$, we say that function **ChangeAtt**, shown in Figure 7, can be applied to T if there exists a node v in T such that $\{@a \in A_{\mathbf{T}} \mid \rho_{@a}(v) \text{ is defined in } T\} \neq R_{\mathbf{T}}(\lambda_T(v))$ and **ChangeAtt**(T, v) does not fail. Moreover, we say that function **ChangeReg**, shown in Figure 7, can be applied to T if there exists a node v in T such that $\lambda_T(\mathit{children}(v)) \notin \pi(P_{\mathbf{T}}(\lambda_T(v)))$ and **ChangeReg**(T, v) does not fail. Notice that if $T \models D_{\mathbf{T}}$, then neither **ChangeAtt** nor **ChangeReg** can be applied to T since for every node v in T we have that $\rho_{@a}(v)$ is defined for every $@a \in R_{\mathbf{T}}(\lambda_T(v))$ and $\lambda_T(\mathit{children}(v)) \in \pi(P_{\mathbf{T}}(\lambda_T(v)))$. A (possible infinite) sequence of trees $T_0, \ldots, T_i, \ldots$ over $(E_{\mathbf{T}}, A_{\mathbf{T}})$ is a *chase sequence* if for every T_j, T_{j+1} in the sequence, either **ChangeAtt** or **ChangeReg** can be applied to T_j to generate T_{j+1}, that is, $T_j \neq T_{j+1}$ and there exists a node v in T_j such that either $T_{j+1} =$ **ChangeAtt**(T_j, v) or $T_{j+1} =$ **ChangeReg**(T_j, v).

Moreover, a chase sequence $T_0, \ldots, T_i, \ldots$, is said to be *terminal* if it is finite and its last tree T_n is such that neither **ChangeAtt** nor **ChangeReg** can be applied to it.

LEMMA 6.12. *Every chase sequence $T_0, \ldots, T_i, \ldots$ is finite.*

PROOF. Let $T_0, \ldots, T_i, \ldots$ be a chase sequence. We prove this lemma by induction on the depth of T_0. If the depth of T_0 is 1, then T_0 has only one node. Given that $D_{\mathbf{T}}$ is a consistent DTD, we have that $T_0, \ldots, T_i, \ldots$ is contained in a terminal chase sequence having as last element a tree T_n conforming to $D_{\mathbf{T}}$.

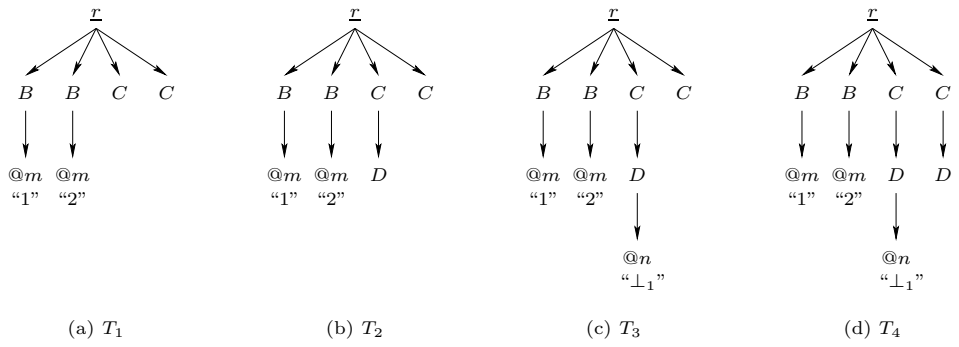
Assume that the depth of T_0 is $m > 1$ and that the property holds for every chase sequence having as first element a tree with depth at most $m-1$. Furthermore, for every pair T_j, T_{j+1} of trees in $T_0, \ldots, T_i, \ldots$, assume that either $T_{j+1} = \mathbf{ChangeAtt}(T_j, v_j)$ or $T_{j+1} = \mathbf{ChangeReg}(T_j, v_j)$. Let u_0 be the root node of T_0. Since u_0 can be used at most twice to construct the chase sequence $T_0, \ldots, T_i, \ldots$, once as input of **ChangeAtt** and the other one as input of **ChangeReg**, we can assume that there exists $j \geq 0$ such that $u_0 \neq v_k$, for every $k \geq j$. Let $u_1, \ldots, u_p$ be the sequence of children of u_0 in T_j, and for every $k \in [1,p]$, let $T_0^k, \ldots, T_i^k, \ldots$ be a chase sequence defined as follows. T_0^k is the subtree of T_j rooted at u_k. Let T', T'' be a consecutive pair of trees in $T_0, \ldots, T_i, \ldots$ such that T', T'' is the ℓ-th ($\ell \geq 1$) pair of consecutive trees in the chase sequence $T_j, \ldots, T_i, \ldots$ such that either $T'' = \mathbf{ChangeAtt}(T', v')$ or $T'' = \mathbf{ChangeReg}(T', v')$ and v' is a descendant of u_k. Then $T_\ell^k = \mathbf{ChangeAtt}(T_{\ell-1}^k, v')$, if $T'' = \mathbf{ChangeAtt}(T', v')$, and $T_\ell^k = \mathbf{ChangeReg}(T_{\ell-1}^k, v')$ otherwise. Given that u_0 is not used to construct the chase sequence $T_j, \ldots, T_i, \ldots$, the children of the root u_0 are the same in every tree of this sequence and, hence, every node used to construct $T_j, \ldots, T_i, \ldots$ is a descendant of some node u_k ($k \in [1,p]$). Thus, if every chase sequence $T_0^k, \ldots, T_i^k, \ldots$ ($k \in [1,p]$) is finite, then $T_j, \ldots, T_i, \ldots$ is finite and, hence, $T_0, \ldots, T_i, \ldots$ is finite. But the depth of T_0^k ($k \in [1,p]$) is at most $m-1$ and, hence, by induction hypothesis we have that $T_0^k, \ldots, T_i^k, \ldots$ is finite. This proves that our original chase sequence $T_0, \ldots, T_i, \ldots$ is also finite. □

As a corollary of the previous lemma we obtain that every chase sequence $T_0, \ldots, T_i, \ldots$ is contained in a terminal sequence.

We say that a chase sequence $T_0, \ldots, T_n$ is *successful* if $T_n \models D_{\mathbf{T}}$, and we say that $T_0, \ldots, T_n$ is a *failing* chase sequence if $T_0, \ldots, T_n$ is terminal and $T_n \not\models D_{\mathbf{T}}$. Moreover, given a source tree T conforming to $D_{\mathbf{S}}$, we say that $T_0, \ldots, T_n$ is a *chase sequence for* T if $T_0 = cps(T)$, and we say that a target tree T' is a *canonical solution for* T if there exists a successful chase sequence $T_0, \ldots, T_n$ for T such that $T' = T_n$.

EXAMPLE 6.13. Let $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$ be the DTDs shown in Figures 6 (a) and (b), and $\Sigma_{\mathbf{ST}}$ be the set $\{\underline{r}[B(@m = x)] \coloneq A(@\ell = x)\}$ of fully-specified STDs. Notice that $D_{\mathbf{T}}$ is a $\mathcal{C}_U$-DTD since $(BC)^*$ is a univocal regular expression.

Assume that T is the source tree, conforming to $D_{\mathbf{S}}$, shown in Figure 6 (c). Then the canonical pre-solution for T is the tree shown in Figure 6 (d), and the canonical solution for T shown in Figure 6 (e) is constructed as follows. Let $T_0 = cps(T)$, v_0 be the root node of T_0, v_1, v_2 the sequence of children of v_0 and $w = \lambda_{T_0}(v_1)\lambda_{T_0}(v_2) = BB$. Since $BB \notin \pi(P_{\mathbf{T}}(\underline{r})) = \pi((BC)^*)$, we invoke $\mathbf{ChangeReg}(T_0, v_0)$.



(a) T_1 (b) T_2 (c) T_3 (d) T_4

Fig. 8. Chase sequence.

Since $rep(BB, (BC)^*) = min\_ext(B, (BC)^*) \cup min\_ext(BB, (BC)^*) = \{BC, CB\} \cup \{BBCC, BCBC, BCCB, CBBC, CBCB, CCBB\}$, **ChangeReg** does not fail and it chooses a string from $\max_{\preceq_{BB}} rep(BB, (BC)^*)$. Given that $BC \equiv_{BB} CB$ ($BC \preceq_{BB} CB$ and $CB \preceq_{BB} BC$), $BC \prec_{BB} BBCC$ ($BC \preceq_{BB} BBCC$ and $BBCC \not\preceq_{BB} BC$) and $BBCC$ is equivalent to all the strings in $min\_ext(BB, (BC)^*)$, the algorithm chooses $w' = BBCC$. Since $\#_B(BB) = \#_B(BBCC)$ and $\#_C(BB) < \#_C(BBCC)$, the algorithm adds to v_0 two new children of type C, generating tree T_1 shown in Figure 8 (a). Let v_3 and v_4 be the node identifiers of these new nodes.

Given that $\lambda_{T_1}(v_3) = C$, $P_{\mathbf{T}}(C) = D$ and v_3 has no children, we invoke **ChangeReg**(T_1, v_3). Since $rep(\varepsilon, D)$ is not empty ($rep(\varepsilon, D) = \{D\}$), **ChangeReg** does not fail and it chooses a string w' from $\max_{\preceq_\varepsilon} rep(\varepsilon, D)$, in this case $w' = D$. Then it adds to v_3 a new child of type D since $\#_D(\varepsilon) < \#_D(D)$, generating tree T_2 shown in Figure 8 (b). Let v_5 be the node identifier of this new node. Since $\lambda_{T_2}(v_5) = D$, $R_{\mathbf{T}}(D) = \{@n\}$ and v_5 has no attributes, we invoke **ChangeAtt**(T_2, v_5). This function adds to v_5 an attribute $@n$ with value $\perp_1$, where $\perp_1$ is a fresh null value. The resulting tree T_3 is shown in Figure 8 (c).

Recall that v_4 is the last children of root node v_0. Given that $\lambda_{T_3}(v_4) = C$, $P_{\mathbf{T}}(C) = D$ and v_4 has no children, we invoke **ChangeReg**(T_3, v_4). Since $rep(\varepsilon, D)$ is not empty, **ChangeReg** does not fail and it chooses a string w' from $\max_{\preceq_\varepsilon} rep(\varepsilon, D)$, in this case $w' = D$. Then it adds to v_4 a new child of type D since $\#_D(\varepsilon) < \#_D(D)$, generating tree T_4 shown in Figure 8 (d). Finally, let v_6 be the node identifier of this new node. Since $\lambda_{T_4}(v_6) = D$, $R_{\mathbf{T}}(D) = \{@n\}$ and v_6 has no attributes, we invoke **ChangeAtt**(T_4, v_6). This function adds to v_6 an attribute $@n$ with value $\perp_2$, where $\perp_2$ is a fresh null value. The resulting tree is the canonical solution for T, shown in Figure 6 (e). □

Next we show that canonical solutions can be used to compute certain answers. To do this, first we introduce the notion of homomorphism for trees. Let $E \subset El$ be a finite set of element types, $A \subset Att$ be a finite set of attributes and T, T' be XML trees over (E, A). Assume that $T = (N, <_{\text{child}}, root)$ and $\mathsf{Str}(T) = \{s \in \mathsf{Str} \mid$ there exists $v \in N$ and $@a \in A$ such that $\rho_{@a}(v) = s\}$. Furthermore, assume that $T' = (N', <'_{\text{child}}, root')$ and $\mathsf{Str}(T')$ is defined as above. Then a function

$h : N \cup \mathsf{Str}(T) \to N' \cup \mathsf{Str}(T')$ is a homomorphism from T to T', denoted by $h : T \to T'$, if:

—for every $v \in N$, $h(v) \in N'$;
—for every $s \in \mathsf{Const} \cap \mathsf{Str}(T)$, $h(s) = s$, and for every $s \in \mathsf{Var} \cap \mathsf{Str}(T)$, $h(s) \in \mathsf{Str}(T')$;
—$h(root) = root'$;
—for every $v_1, v_2 \in N$, if $v_1 <_{\text{child}} v_2$, then $h(v_1) <'_{\text{child}} h(v_2)$;
—for every $v \in N$, $\lambda_T(v) = \lambda_{T'}(h(v))$;
—for every $v \in N$ and $@a \in A$ such that $\rho_{@a}(v)$ is defined, $h(\rho_{@a}(v)) = \rho_{@a}(h(v))$.

It is easy to see that the following lemma holds.

LEMMA 6.14. *Let E be a finite set of element types, A a finite set of attributes, T, T' XML trees over (E, A), $Q(\bar{x})$ a $\mathcal{CTQ}^{//,\cup}$-query over (E, A) and $\bar{s}$ a tuple of constants such that $T \models Q(\bar{s})$. If there exists a homomorphism $h : T \to T'$, then $T' \models Q(\bar{s})$.*

LEMMA 6.15. *Let $T_0, \ldots, T_n$ be a terminal chase sequence for a tree T conforming to $D_{\mathbf{S}}$.*

(a) For every solution T' for T and every $i \in [0, n]$, there exists a homomorphism $h_i : T_i \to T'$.
(b) If $T_0, \ldots, T_n$ is a failing sequence, then there is no solution for T.

PROOF. (a) By induction on $i \in [0, n]$. Since $T_0 = cps(T)$ and T' is a solution for T, it is easy to see that there exists a homomorphism $h_0 : T_0 \to T'$. Assume that the property holds for $i < n$, that is, there exists a homomorphism $h_i : T_i \to T'$. To show that there exists a homomorphism $h_{i+1} : T_{i+1} \to T'$, we consider two cases. In both cases we assume that N', N_i, N_{i+1} are the sets of node identifiers of T', T_i and T_{i+1}, respectively.

First, assume that $T_{i+1} = \mathbf{ChangeAtt}(T_i, v)$, and define function $h_{i+1} : N_{i+1} \cup \mathsf{Str}(T_{i+1}) \to N' \cup \mathsf{Str}(T')$ as follows. For every $u \in N_{i+1}$, define $h_{i+1}(u)$ as $h_i(u)$. We note that this is well defined since $T_{i+1} = \mathbf{ChangeAtt}(T_i, v)$ and, hence, $N_i = N_{i+1}$. For every string $s \in \mathsf{Str}(T_{i+1}) \cap \mathsf{Const}$, define $h_{i+1}(s)$ as s. We note that this is well defined since $\mathsf{Str}(T_i) \cap \mathsf{Const} = \mathsf{Str}(T_{i+1}) \cap \mathsf{Const}$. Finally, for every $s \in \mathsf{Str}(T_{i+1}) \cap \mathsf{Var}$, we consider two cases. If $s \in \mathsf{Str}(T_i)$, then define $h_{i+1}(s)$ as $h_i(s)$. Otherwise, s is a null value that is added to T_i by $\mathbf{ChangeAtt}(T_i, v)$ and, therefore, there exists $@a \in R_{\mathbf{T}}(\lambda_{T_{i+1}}(v))$ such that $s = \rho_{@a}(v)$. Define $h_{i+1}(s)$ as $\rho_{@a}(h_i(v))$. We observe that this is well defined since $T' \models D_{\mathbf{T}}$ and, hence, $\rho_{@a}(h_i(v))$ is defined in T' since $@a \in R_{\mathbf{T}}(\lambda_{T_i}(v)) = R_{\mathbf{T}}(\lambda_{T'}(h_i(v)))$. Since h_i is a homomorphism, it is easy to see that h_{i+1} is a homomorphism from T_{i+1} to T'.

Second, assume that $T_{i+1} = \mathbf{ChangeReg}(T_i, v)$. To define homomorphism $h_{i+1} : T_{i+1} \to T'$ we need to prove an intermediate result. Let $w_i = \lambda_{T_i}(children(v))$, $w_{i+1} = \lambda_{T_{i+1}}(children(v))$ and $w' = \lambda_{T'}(children(h_i(v)))$.

CLAIM 6.16.

(a) $alph(w_{i+1}) \subseteq alph(w')$.
(b) If $a \in alph(w_i)$ is such that $\#_a(w_{i+1}) = 1$ and $\#_a(w_i) > 1$, then $\#_a(w') = 1$.

The proof of Claim 6.16 can be found in the appendix. We use Claim 6.16 to define function $h_{i+1} : N_{i+1} \cup \mathsf{Str}(T_{i+1}) \to N' \cup \mathsf{Str}(T')$. For every $u \in N_{i+1}$ that is not a child of v, define $h_{i+1}(u)$ as $h_i(u)$. We observe that this is well defined since **ChangeReg**(T_i, v) modifies only the sequence of children of v. For every $s \in \mathsf{Str}(T_{i+1}) \cap \mathsf{Const}$, define $h_{i+1}(s)$ as s. We note that this is well defined since $\mathsf{Str}(T_i) \cap \mathsf{Const} = \mathsf{Str}(T_{i+1}) \cap \mathsf{Const}$. Finally, for every children u of v and every $@a \in R_{\mathbf{T}}(\lambda_{T_{i+1}}(u))$, we define $h_{i+1}(u)$ and $h_{i+1}(\rho_{@a}(u))$ as follows. Assume that $\lambda_{T_{i+1}}(u) = \ell$. If $\#_\ell(w_{i+1}) \geq \#_\ell(w_i)$ and $u \in N_i$, then $h_{i+1}(u) = h_i(u)$ and $h_{i+1}(\rho_{@a}(u)) = h_i(\rho_{@a}(u))$. If $\#_\ell(w_{i+1}) \geq \#_\ell(w_i)$ and $u \notin N_i$, then define $h_{i+1}(u)$ as u', where u' is an arbitrary child of $h_i(v)$ in T' of type ℓ. We note that such a node exists since, by Claim 6.16, $alph(w_{i+1}) \subseteq alph(w')$. If $\#_\ell(w_{i+1}) < \#_\ell(w_i)$, then $\#_\ell(w_{i+1}) = 1$ by definition of **ChangeReg** and , by Claim 6.16, $\#_\ell(w') = 1$. Let u' be the only child of $h_i(v)$ of type ℓ. Then define $h_{i+1}(u)$ as u' and $h_{i+1}(\rho_{@a}(u))$ as $\rho_{@a}(u')$. We observe that this is well defined since if $\rho_{@a}(u) = s$ is a constant, then by definition of **ChangeReg** there exists a child u'' of v in T_i of type ℓ such that $\rho_{@a}(u'') = s$ and $h_i(u'') = u'$ (u' is the only child of $h_i(v)$ of type ℓ), and, hence, $h_{i+1}(s) = \rho_{@a}(u') = h_i(\rho_{@a}(u'')) = h_i(s) = s$. Since h_i is a homomorphism, it is easy to see that h_{i+1} is a homomorphism from T_{i+1} to T'. This concludes the proof of the first part of the lemma.

(b) By contradiction, assume that $T_0, \ldots, T_n$ is a failing sequence and that there exists a solution T' for T. Then there exists a node v in T_n such that either $\{@a \in A_{\mathbf{T}} \mid \rho_{@a}(v) \text{ is defined in } T_n\} \neq R_{\mathbf{T}}(\lambda_{T_n}(v))$ and **ChangeAtt**(T_n, v) fails or $\lambda_{T_n}(children(v)) \notin \pi(P_{\mathbf{T}}(\lambda_{T_n}(v)))$ and **ChangeReg**(T_n, v) fails. Let $w = children(v)$, h a homomorphism from T_n to T' (such a homomorphism exists by (a)) and $w' = children(h(v))$. We consider three cases.

First, assume that there exists a node v in T_n such that $\{@a \in A_{\mathbf{T}} \mid \rho_{@a}(v)$ is defined in $T_n\} \neq R_{\mathbf{T}}(\lambda_{T_n}(v))$ and **ChangeAtt**(T_n, v) fails. Then there exists $@a \in A_{\mathbf{T}} \setminus R_{\mathbf{T}}(\lambda_{T_n}(v))$ such that $\rho_{@a}(v)$ is defined in T_n. Thus, given that h is a homomorphism from T_n to T', we have that $\rho_{@a}(h(v)) = h(\rho_{@a}(v))$. We conclude that $T' \not\models D_{\mathbf{T}}$ since $@a \notin R_{\mathbf{T}}(\lambda_{T'}(h(v))) = R_{\mathbf{T}}(\lambda_{T_n}(v))$, which contradicts the fact that T' is a solution for T.

Second, assume that there exists a node v in T_n such that $\lambda_{T_n}(children(v)) \notin \pi(P_{\mathbf{T}}(\lambda_{T_n}(v)))$ and **ChangeReg**(T_n, v) fails because $rep(w, P_{\mathbf{T}}(\lambda_{T_n}(v))) = \emptyset$. Since $h : T_n \to T'$, we have that $alph(w) \subseteq alph(w')$. Define string w_1 as follows: $alph(w_1) = alph(w)$ and $\#_a(w_1) = 1$, for every $a \in alph(w_1)$. Then $w_1 \preceq w'$ since $alph(w) \subseteq alph(w')$. Thus, there exists $w_2 \in min\_ext(w_1, P_{\mathbf{T}}(\lambda_{T_n}(v)))$ such that $w_1 \preceq w_2 \preceq w'$ since $w' \in \pi(P_{\mathbf{T}}(\lambda_{T'}(h(v)))) = \pi(P_{\mathbf{T}}(\lambda_{T_n}(v)))$. By definition of $rep(w, P_{\mathbf{T}}(\lambda_{T_n}(v)))$ we have that $w_2 \in rep(w, P_{\mathbf{T}}(\lambda_{T_n}(v)))$ and, therefore, $rep(w, P_{\mathbf{T}}(\lambda_{T_n}(v)))$ is not empty, which contradicts our original assumption.

Third, assume that there exists a node v in T_n such that $\lambda_{T_n}(children(v)) \notin \pi(P_{\mathbf{T}}(\lambda_{T_n}(v)))$ and **ChangeReg**(T_n, v) fails because the algorithm chooses $w_1 \in \max_{\preceq_w} rep(w, P_{\mathbf{T}}(\lambda_{T_n}(v)))$ such that there exist $b \in alph(P_{\mathbf{T}}(\lambda_{T_n}(v)))$, $@a \in R_{\mathbf{T}}(b)$ and distinct children u_1, u_2 of v of type b such that $\#_b(w_1) < \#_b(w)$, $\rho_{@a}(u_1) \in \mathsf{Const}$, $\rho_{@a}(u_2) \in \mathsf{Const}$ and $\rho_{@a}(u_1) \neq \rho_{@a}(u_2)$. To establish a contradiction we need the following claim.

Claim 6.17. *Let r be a univocal regular expression. For every string w_1 such*

that $rep(w_1, r) \neq \emptyset$*, every* $w_2 \in \max_{\preceq_{w_1}} rep(w_1, r)$ *and every* $a \in alph(w_1)$*, if* $\#_a(w_2) < \#_a(w_1)$*, then* $\#_a(w_2) = 1$*.*

The proof of Claim 6.17 can be found in the appendix. We use this claim to show that $\#_b(w') = 1$. On the contrary, assume that $\#_b(w') > 1$. Since h is a homomorphism from T_n to T', we have that $alph(w) \subseteq alph(w')$. Define string w_2 as follows: $alph(w_2) = alph(w)$, $\#_b(w_2) = 2$ and $\#_c(w_2) = 1$, for every $c \in alph(w_2) \setminus \{b\}$. Then $w_2 \preceq w'$ since $alph(w) \subseteq alph(w')$ and $\#_b(w') > 1$. Thus, there exists $w_3 \in min\_ext(w_2, P_{\mathbf{T}}(\lambda_{T_n}(v)))$ such that $w_2 \preceq w_3 \preceq w'$ since $w' \in \pi(P_{\mathbf{T}}(\lambda_{T'}(h(v)))) = \pi(P_{\mathbf{T}}(\lambda_{T_n}(v)))$. By definition of $rep(w, P_{\mathbf{T}}(\lambda_{T_n}(v)))$ we have that $w_3 \in rep(w, P_{\mathbf{T}}(\lambda_{T_n}(v)))$. Since $\#_b(w_2) = 2$, we have that $\#_b(w_3) > 1$. Furthermore, given that $\#_b(w_1) < \#_b(w)$ and $w_1 \in \max_{\preceq_w} rep(w, P_{\mathbf{T}}(\lambda_{T_n}(v)))$, by Claim 6.17 we have that $\#_b(w_1) = 1$. Hence, given that $\#_b(w_3) > 1$, $\#_b(w_1) = 1$ and $\#_b(w) > \#_b(w_1)$, we conclude that $w_3 \not\preceq_w w_1$, which contradicts the fact that $(rep(w, P_{\mathbf{T}}(\lambda_{T_n}(v))), \preceq_w)$ has a maximum element (recall that $P_{\mathbf{T}}(\lambda_{T_n}(v))$ is a univocal regular expression) and $w_1 \in \max_{\preceq_w} rep(w, P_{\mathbf{T}}(\lambda_{T_n}(v)))$.

Since $\#_b(w') = 1$, there is only one child of $h(v)$ of type b, say u'. Since $h : T_n \to T'$ is a homomorphism and both u_1 and u_2 are child of v of type b, we have that $h(u_1) = h(u_2) = u'$. Thus, $h(\rho_{@a}(u_1)) = h(\rho_{@a}(u_2)) = \rho_{@a}(u')$. Thus, given that both $\rho_{@a}(u_1) \in \mathsf{Const}$ and $\rho_{@a}(u_2) \in \mathsf{Const}$, we conclude that $\rho_{@a}(u_1) = h(\rho_{@a}(u_1)) = h(\rho_{@a}(u_2)) = \rho_{@a}(u_2)$, which contradicts our original assumption. This concludes the proof of the second part of the lemma. □

We are finally ready to prove Lemma 6.5.

Proof of Lemma 6.5. (a) We only need to prove one direction. Assume that there is no canonical solution for T. By Lemma 6.12, there exists a failing chase sequence for T and, therefore, there is no solution for T by Lemma 6.15.

(b) First, assume that $T^\star \not\models Q(\bar{s})$. Then $\bar{s} \notin \underline{certain}(Q, T)$ since $T^\star$ is a solution for T. Second, assume that $T^\star \models Q(\bar{s})$ and let T' be a solution for T. Then, by Lemma 6.15 we know that there exists a homomorphism h from $T^\star$ to T'. Thus, by Lemma 6.14 we have that $T' \models Q(\bar{s})$ since Q is a $\mathcal{CTQ}^{//,\cup}$-query. We conclude that for every solution T' for T, it is the case that $T' \models Q(\bar{s})$ and, hence, $\bar{s} \in \underline{certain}(Q, T)$. □

To show that the canonical solution can be computed in polynomial time, we just need to prove one additional lemma.

Lemma 6.18. *Let* r *be a fixed regular expression. Then the following problems are solvable in* PTIME*:*

(1) Given an input string w*, determine whether* $rep(w, r) \neq \emptyset$*.*

(2) Given an input string w *such that* $rep(w, r) \neq \emptyset$*, compute* $w' \in \max_{\preceq_w} rep(w, r)$*.*

Proof. Since r is a fixed regular expression, by Proposition 5.3, the problem of checking whether a string $w \in \pi(r)$ can be solved in time $O(p(|w|))$, where p is a fixed polynomial.

(1) In what follows, we assume that $alph(r) = \{a_1, \ldots, a_k\}$ and $f : \{a_1, \ldots, a_k\}^* \to \mathbb{N}^k$ is a mapping defined as $f(w) = (\#_{a_1}(w), \ldots, \#_{a_k}(w))$. We

note that if $f(w_1) = f(w_2)$, then $w_1 \in \pi(r)$ if and only if $w_2 \in \pi(r)$. To determine whether $rep(w, r) \neq \emptyset$, we have to check whether there exists string w' such that $alph(w') = alph(w)$, $w' \preceq w$ and $min\_ext(w', r) \neq \emptyset$. By Lemma 5.8, to verify whether $min\_ext(w', r) \neq \emptyset$, we only need to check whether there exists string w'' such that $w'' \in \pi(r)$, $w' \preceq w''$ and $|w''| \leq (|w'| + 1) \cdot \|r\|$. Thus, given that

$$|\{f(w'') \mid \text{ for every } a \in alph(r),\ \#_a(w'') \leq (|w'| + 1) \cdot \|r\|\}| \ \leq \ ((|w'| + 1) \cdot \|r\| + 1)^{|alph(r)|}$$

we have that the problem of verifying whether $min\_ext(w', r) \neq \emptyset$ can be solved in time $O(p((|w'| + 1) \cdot \|r\|) \cdot ((|w'| + 1) \cdot \|r\| + 1)^{|alph(r)|})$. Therefore, given that

$$|\{f(w') \mid alph(w') = alph(w) \text{ and } w' \preceq w\}| \ = \ \prod_{a \in alph(w)} \#_a(w) \ \leq \ \prod_{a \in alph(w)} |w| \ \leq \ \prod_{a \in alph(r)} |w| \ = \ |w|^{|alph(r)|},$$

we conclude that the problem of verifying whether $rep(w, r) \neq \emptyset$ can be solved in time $O(|w|^{|alph(r)|} \cdot p((|w|+1) \cdot \|r\|) \cdot ((|w|+1) \cdot \|r\| + 1)^{|alph(r)|})$, which is polynomial on $|w|$ since r is fixed.

(2) Given that r is a fixed regular expression, we know by (1) that for every string w' such that $alph(w') = alph(w)$ and $w' \preceq w$, set $\{f(w'') \mid w'' \in min\_ext(w', r)\}$ can be computed in polynomial time on $|w'| \leq |w|$. Furthermore, we also know by (1) that the number of $f(w')$ for strings w' satisfying the previous condition is polynomial on $|w|$ and, therefore, it is possible to compute $\{f(w'') \mid w'' \in rep(w, r)\}$ in polynomial time on $|w|$. By Lemma 5.8 the length of a string in $rep(w, r)$ is at most $(|w|+1) \cdot \|r\|$ and, thus, it is possible to compute $\max_{\preceq_w} rep(w, r)$ in polynomial time on $|w|$. This concludes the proof of the lemma. □

Proof of Lemma 6.6. Given a fixed data exchange setting, function **ChangeReg** can be implemented in polynomial time by Lemma 6.18. Thus, by applying **ChangeAtt** and **ChangeReg** to $cps(T)$ in a depth-first search manner (as shown in Example 6.4), we can compute a terminal chase sequence $T_0, \ldots, T_n$ for T in polynomial time. If $T_n \not\models D_{\mathbf{T}}$, then we know by Lemma 6.15 that there is no solution for T and, in particular, there is no canonical solution for this tree. If $T_n \models D_{\mathbf{T}}$, then T_n is a canonical solution for T. □

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 6.19. *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, which are proved in the rest of this section.

Lemma 6.20. *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 6.21. *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.*

PROOF OF LEMMA 6.20. Let r be a regular expression such that $c(r) = k \geq 2$ and $\mathcal{C}$ be an admissible class of regular expressions containing r. Since $c(r) = k$, there exists $a \in \mathit{alph}(r)$ and $w \in \mathit{fixed}_a(r)$ such that $w = a^k a_1 \cdots a_\ell$, where $a_i \in \mathit{alph}(r)$ and $a \neq a_i$ $(i \in [1, \ell])$.

To show that $\mathcal{C}$ is strongly coNP-complete, we define a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ and a Boolean $\mathcal{CTQ}$-query Q such that $D_{\mathbf{S}}$ is a simple DTD, $D_{\mathbf{T}}$ is a $\mathcal{C}$-DTD, $\Sigma_{\mathbf{ST}}$ is a set of fully-specified STDs and 3SAT can be reduced to the complement of CERTAIN-ANSWERS(Q), that is, for every propositional formula θ in 3-CNF, there exists a PTIME constructible XML tree T_θ conforming to $D_{\mathbf{S}}$ such that θ is satisfiable if and only if $\underline{\mathit{certain}}(Q, T_\theta) = \mathit{false}$.

Simple DTD $D_{\mathbf{S}}$ is defined as follows. Let $E_{\mathbf{S}} = \{B,\ C,\ H,\ L,\ I_1,\ \ldots,\ I_k,\ J_1,\ \ldots,\ J_\ell\}$ be a set of element types and $A_{\mathbf{S}} = \{@f,\ @s,\ @t,\ @p,\ @n,\ @id\}$ a set of attributes. Then $D_{\mathbf{S}} = (P_{\mathbf{S}}, R_{\mathbf{S}}, B)$ is a DTD over $(E_{\mathbf{S}}, A_{\mathbf{S}})$, where $P_{\mathbf{S}}$ is defined as:

$$\begin{aligned} P_{\mathbf{S}}(B) &= C^* H^* L^* I_1^* \cdots I_k^* J_1^* \cdots J_\ell^*, \\ P_{\mathbf{S}}(\ell) &= \varepsilon, \quad \text{for every } \ell \in E_{\mathbf{S}} \setminus \{B\}, \end{aligned}$$

and $R_{\mathbf{S}}$ is defined as:

$$\begin{array}{lllllll} R_{\mathbf{S}}(B) & = & \emptyset, & & R_{\mathbf{S}}(H) & = & \{@t, @f\}, \\ R_{\mathbf{S}}(C) & = & \{@f, @s, @t\}, & & R_{\mathbf{S}}(L) & = & \{@p, @n\}, \\ R_{\mathbf{S}}(I_i) & = & \{@id\}, \text{ for every } i \in [1,k], & & R_{\mathbf{S}}(J_i) & = & \{@id\}, \text{ for every } i \in [1,\ell]. \end{array}$$

In the following example we explain how XML trees conforming to $D_{\mathbf{S}}$ are used to represent propositional formulae. Let θ be 3-CNF formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$. To construct tree T_θ, first we assign a distinct natural number to each literal, say

$$\begin{array}{llll} x_1 \mapsto 1, & x_2 \mapsto 3, & x_3 \mapsto 5, & x_4 \mapsto 7, \\ \neg x_1 \mapsto 2, & \neg x_2 \mapsto 4, & \neg x_3 \mapsto 6, & \neg x_4 \mapsto 8. \end{array}$$

Then we represent each clause of θ as a node of type C, being the values of attributes $@f$, $@s$, $@t$ the first, second and third literal of that clause, respectively. For each propositional variable x in θ, we use the attributes $@p$, $@n$ of a node of type L to store the values assigned to x and $\neg x$, respectively. Furthermore, we create a node of type H and store in its attribute $@t$ value 1 (representing *true* value) and in its attribute $@f$ value 0 (representing *false* value). Finally, for every $i \in [1,k]$ we create a node of type I_i with value i in its attribute $@id$, and for every $j \in [1,\ell]$ we create a node of type J_j with value j in its attribute $@id$ (it will become clear why we need these nodes when we define $D_{\mathbf{T}}$ and $\Sigma_{\mathbf{ST}}$). Tree T_θ for $\theta = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$ is shown in Figure 9.

$\mathcal{C}$-DTD $D_{\mathbf{T}}$ is defined as follows. Let $E_{\mathbf{T}} = \{B,\ C,\ H,\ G\} \cup \mathit{alph}(r)$ be a set of element types such that $\{B,\ C,\ H,\ G\} \cap \mathit{alph}(r) = \emptyset$, and let $A_{\mathbf{T}} = \{@f,\ @s,\ @t,\ @id,\ @e,\ @\ell\}$ be a set of attributes. Then $D_{\mathbf{T}} = (P_{\mathbf{T}}, R_{\mathbf{T}}, B)$ is a DTD over $(E_{\mathbf{T}}, A_{\mathbf{T}})$, where $P_{\mathbf{T}}$ is defined as:

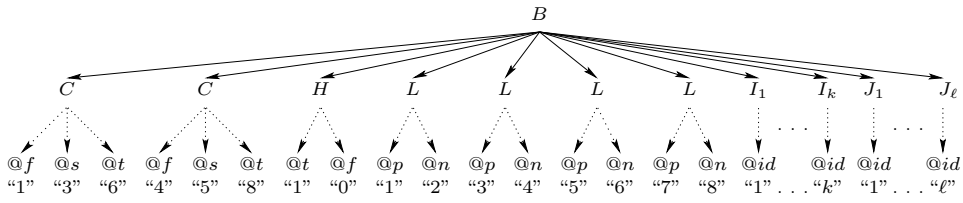


Fig. 9. XML tree T_θ, defined in the proof of Lemma 6.20, representing propositional formula $\theta = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$.

$$\begin{array}{llllllll} P_{\mathbf{T}}(B) & = & C^*H^*G^*, & P_{\mathbf{T}}(G) & = & r, & P_{\mathbf{T}}(\ell) = \varepsilon, \text{ for every } \ell \in alph(r), \\ P_{\mathbf{T}}(C) & = & \varepsilon, & P_{\mathbf{T}}(H) & = & \varepsilon, & \end{array}$$

and $R_{\mathbf{T}}$ is defined as:

$$\begin{array}{llllll} R_{\mathbf{T}}(B) & = & \emptyset, & R_{\mathbf{T}}(C) & = & \{@f, @s, @t\}, \\ R_{\mathbf{T}}(G) & = & \emptyset, & R_{\mathbf{T}}(a) & = & \{@id, @e, @\ell\}, \\ R_{\mathbf{T}}(H) & = & \{@f\}, & R_{\mathbf{T}}(\ell) & = & \{@id\}, \text{ for every } \ell \in alph(r) \setminus \{a\}. \end{array}$$

Finally, set $\Sigma_{\mathbf{ST}}$ of fully-specified STDs is defined as follows. The first rule of $\Sigma_{\mathbf{ST}}$ says that every node of type C in a source tree T must appear in every solution for T:

$$B[C(@f = x, @s = y, @t = z)] \;\; :- \;\; B[C(@f = x, @s = y, @t = z)].$$

The second rule of $\Sigma_{\mathbf{ST}}$ says that the value of attribute $@f$ of a node of type H in a source tree T must appear in every solution for T as an attribute $@f$ of a node of type H:

$$B[H(@f = x)] \;\; :- \;\; B[H(@f = x)].$$

Finally, the third rule of $\Sigma_{\mathbf{ST}}$ is defined as follows:

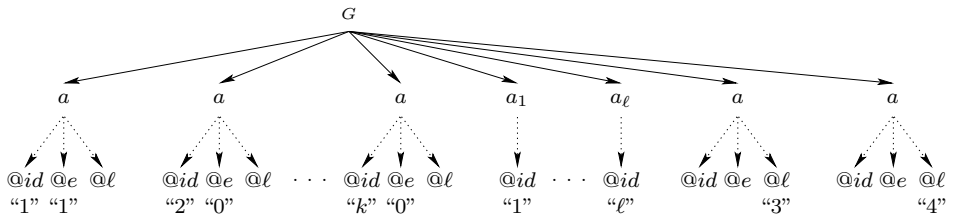
$$\begin{array}{l} B[G[a(@id = u_1, @e = x), \bigwedge_{i=2}^{k} a(@id = u_i, @e = x'), \\ \quad \bigwedge_{j=1}^{\ell} a_j(@id = u'_j),\, a(@\ell = y),\, a(@\ell = y')]] \;\; :- \\ \quad B[H(@t = x, @f = x'),\, L(@p = y, @n = y'),\, \bigwedge_{i=1}^{k} I_i(@id = u_i),\, \bigwedge_{j=1}^{\ell} J_j(@id = u'_j)]. \end{array}$$

In this rule we use symbol $\bigwedge$ to denote a sequence of formulae separated by commas. Indeed, conjunction is not allowed in tree-pattern formulae. Thus, for example, $\bigwedge_{i=2}^{k} a(@id = u_i, @e = x')$ is a shorthand for:

$$a(@id = u_2, @e = x'), \ldots, a(@id = u_k, @e = x').$$

To explain the meaning of the previous rule, we considered again tree T_θ shown in Figure 9. Let i, j be a pair of complementary literals in T_θ, say $i = 3$ and $j = 4$. The previous rule says that for each of such pair and for each solution T' for T_θ,

there should exist a node v of type G in T' having as children the following sequence of nodes:



Thus, v has $k+2$ children of type a. The first of these a-nodes has 1 as value of attribute $@id$ and 1 as value of attribute $@e$ since the third rule of $\Sigma_{\mathbf{ST}}$ says that for this node $@id = u_1$ and $@e = x$, being u_1 and x the values of the attribute $@id$ of the node of type I_1 in T_θ and the attribute $@t$ of the node of type H in T_θ, respectively. For the i-th ($i \in [2,k]$) a-node in the tree shown above, values of attributes $@id$ and $@e$ are i and 0, respectively. For the last two a-nodes in this tree, attribute $@\ell$ takes values 3 and 4, respectively, since STD mentioned above says that there exist children of v having $y = 3$ and $y' = 4$ as values of their attributes $@\ell$ (formula $a(@\ell = y)$, $a(@\ell = y')$ above). Furthermore, v has also as children a sequence of nodes of types a_1, ..., a_ℓ, being i the value of attribute $@id$ of the i-th node of this sequence.

Let w' be the string of types of the children of node v in the tree shown above ($w' = a^k a_1 \cdots a_\ell a^2$). Since $w \in \mathit{fixed}_a(r)$, $w \preceq w'$ and $\#_a(w) < \#_a(w')$, we have that $w' \notin \pi(r)$. Thus, to construct a solution for T_θ we need to replace w' by a string $w'' \in \pi(r)$ such that the generated tree continue satisfying $\Sigma_{\mathbf{ST}}$ and either $w \preceq w''$ and $\#_a(w) = \#_a(w'')$ or $w \not\preceq w''$. The latter case is not possible since the first k children of v of type a have pairwise distinct values in attribute $@id$ and the sequence of children of v of types a_1, ..., a_ℓ have also pairwise distinct values in attribute $@id$ (recall that $a_i \neq a$, for every $i \in [1,\ell]$). Thus, we have to construct a string w'' from w' such that $w \preceq w''$ and $\#_a(w) = \#_a(w'')$. Given that the first k children of v of type a have pairwise distinct values in attribute $@id$, the only way to do this is to remove the last two children of v of type a and choose among the first k a-nodes where to place 3 and 4 as values of attribute $@\ell$. For example, Figure 10 shows a possible way of constructing string w'', where 3 and 4 are the values of attribute $@e$ of the last and the first node of type a, respectively. In this figure, dots next to the node of type a_ℓ represent the fact that v can have some extra children not of type a.

We observe that when choosing where to place values 3 and 4, we are actually choosing the truth values for x_2 and $\neg x_2$. For example, in the solution for T_θ shown in Figure 10, we have chosen value 1 for $\neg x_2$ since 4 is the value of attribute $@\ell$ of a node with value distinct from 0 in attribute $@e$, and we have chosen value 0 for x_2 since 3 is the value of attribute $@\ell$ of a node with value 0 in attribute $@e$. We will use this property to prove that θ is satisfiable if and only if $\underline{\mathit{certain}}(Q, T_\theta) = \mathit{false}$. It is worth mentioning that if $k > 2$, then it is possible to choose value 0 for both x_2 and $\neg x_2$. After defining query Q, we will show that this alternative does not cause any problems.

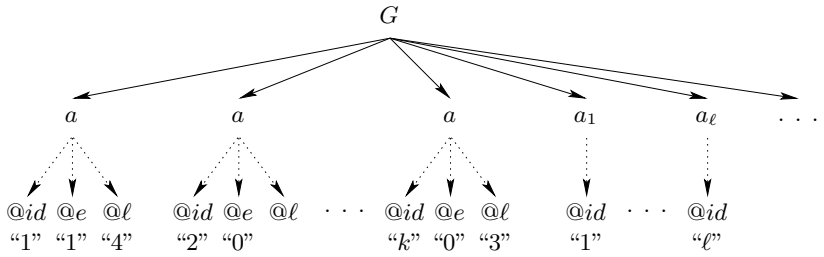


Fig. 10. A solution T' for source tree T_θ.

Boolean $\mathcal{CTQ}$-query Q is defined as:

$$\exists x \exists y \exists z \exists u\ B[C(@f = x, @s = y, @t = z),\ H(@f = u),\\ G[a(@e = u, @\ell = x)],\ G[a(@e = u, @\ell = y)],\ G[a(@e = u, @\ell = z)]].$$

Intuitively, query Q says that there exists a node v of type C such that each "literal" of v is assigned value 0, that is, the values i_1, i_2, i_3 of the attributes $@f$, $@s$, $@t$ of v, respectively, are such that for every $i \in \{i_1, i_2, i_3\}$, there exists a node v' of type a such that 0, i are the values of attributes $@e$, $@\ell$ of v', respectively.

Now we prove that for every 3-CNF propositional formula θ, we have that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = false$, where T_θ is constructed from θ in PTIME as shown above.

($\Rightarrow$) Assume that θ is satisfiable and let σ be a truth assignment satisfying θ. Define a solution T' for T_θ as follows. The structure of the C- and H-nodes of T' is copied from T_θ. For every propositional variables x in θ, we define a G-node v of T' as follows. The string of types of the children of v is w. The values of attributes $@id$ and $@e$ of the children of v are assigned as shown above. If $\sigma(x) = 1$, then the value assigned to x in T_θ is the value of the attribute $@\ell$ of the first child of v of type a, which has value 1 in attribute $@e$, and the value assigned to $\neg x$ in T_θ is the value of the attribute $@\ell$ of the second child of v of type a, which has value 0 in attribute $@e$. Otherwise, $\sigma(x) = 0$ and the value assigned to $\neg x$ in T_θ is the value of the attribute $@\ell$ of the first child of v of type a, and the value assigned to x in T_θ is the value of the attribute $@\ell$ of the second child of v of type a.

It is straightforward to prove that T' conforms to $D_{\mathbf{T}}$ and satisfies $\Sigma_{\mathbf{ST}}$. Furthermore, $T' \not\models Q$ since σ satisfies θ and, therefore, at least one literal per clause C is not assigned value 0. We conclude that $\underline{certain}(Q, T_\theta) = false$ since T' is a solution for T_θ.

($\Leftarrow$) Assume that $\underline{certain}(Q, T_\theta) = false$ and let T' be a solution for T_θ such that $T' \not\models Q$. We define a truth assignment for the propositional variables of θ as follows. For every clause in θ, find a C-node v of T' such that the values i_1, i_2, i_3 of attributes $@f$, $@s$, $@t$ of v are the values assigned in T_θ to the literals of that clause. Since $T' \not\models Q$, there exists $i \in \{i_1, i_2, i_3\}$ such that for every node v' of type a with value 0 in its attribute $@e$, we have that i is not the value of attribute $@\ell$ of v'. If i corresponds to a positive literal x, then define $\sigma(x)$ as 1. If i corresponds to a negative literal $\neg x$, then define $\sigma(x)$ as 0 ($\sigma(\neg x) = 1$)

We show that σ is well defined. On the contrary, assume that there exists a propositional variable x such that 1 was assigned to both $\sigma(x)$ and $\sigma(\neg x)$. Let i, j be the values assigned in T_θ to x and $\neg x$. Then for every node v' of type a with value 0 in its attribute $@e$, we have that i is not the value of attribute $@\ell$ of v'. Let v'' be a G-node of T' that satisfies the left hand side of the third rule of $\Sigma_{\mathbf{ST}}$ instantiated on the following values from T_θ:

$$B[H(@t = 1, @f = 0),\ L(@p = i, @n = j),\ \bigwedge_{i=1}^{k} I_i(@id = i),\ \bigwedge_{j=1}^{\ell} J_j(@id = j)].$$

This node has as children two distinct a-nodes v_1, v_2 with values i, j in attribute $@\ell$, respectively. Since i must be the value of attribute $@\ell$ of a child of v'' of type a with value distinct from 0 in attribute $@e$, and there is exactly one child of v'' satisfying this condition (with value 1 in attribute $@e$), we have that the value of attribute $@e$ of v_2 is 0. Thus, there exists a node v' of type a such that 0, j are the values of attributes $@e$, $@\ell$ of v', respectively, and, therefore, 1 is not assigned to $\neg x$, which contradicts our original assumption.

Since σ is well defined and σ satisfies θ (by definition of σ), we conclude that θ is satisfiable. This concludes the proof Lemma 6.20. □

Proof of Lemma 6.21. Assume that $\mathcal{C}$ is an admissible class of regular expressions containing a non-univocal regular expression r such that $c(r) \leq 1$, and assume that $alph(r) = \Sigma$.

Since r is not a univocal regular expression, there exists $w \in \Sigma^*$ such that $w \notin \pi(r)$, $rep(w, r) \neq \emptyset$ and $rep(w, r)$ does not have a maximum element with respect to $\preceq_w$, that is, there exist w_1, ..., w_n in $rep(w, r)$ ($n \geq 2$) such that (1) for every $w' \in rep(w, r)$, there exists $i \in [1, n]$ such that $w' \preceq_w w_i$, and (2) for every $i, j \in [1, n]$, $i \neq j$, we have that $w_i \not\preceq_w w_j$. Furthermore,

Claim 6.22. *If $w' \in \pi(r)$ is such that $alph(w) \subseteq alph(w')$, then there exists $i \in [1, n]$ such that $w' \preceq_w w_i$.*

The proof of Claim 6.22 can be found in the appendix. To prove the lemma, we consider three cases.

(I) First, assume that there exists distinct $i, j \in [1, n]$ such that $w \preceq w_i$ and $w \preceq w_j$. Without loss of generality, assume that $i = 1$ and $j = 2$. Define $I = \{i \in [2, n] \mid w \preceq w_i\}$ and for every $i \in I \cup \{1\}$, define $X(w_i)$ as $alph(w_i) \setminus alph(w)$. Furthermore, define Y as

$$Y = \left(\bigcup_{i \in I} X(w_i)\right) \setminus X(w_1).$$

Since $w_2 \not\preceq_w w_1$, there exists $a_0 \in X(w_1)$ such that $a_0 \notin X(w_2)$. Given that $w_1 \not\preceq_w w_i$ ($i \in I$), there exists $b \in X(w_i)$ such that $b \notin X(w_1)$. Thus, by definition of Y, we conclude that $Y \cap X(w_1) = \emptyset$ and $X(w_i) \cap Y \neq \emptyset$, for every $i \in I$.

To show that $\mathcal{C}$ is strongly coNP-complete, we define a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ and a Boolean $\mathcal{CTQ}$-query Q such that $D_{\mathbf{S}}$ is a simple DTD, $D_{\mathbf{T}}$ is a $\mathcal{C}$-DTD, $\Sigma_{\mathbf{ST}}$ is a set of fully-specified STDs and 3SAT can be reduced to the

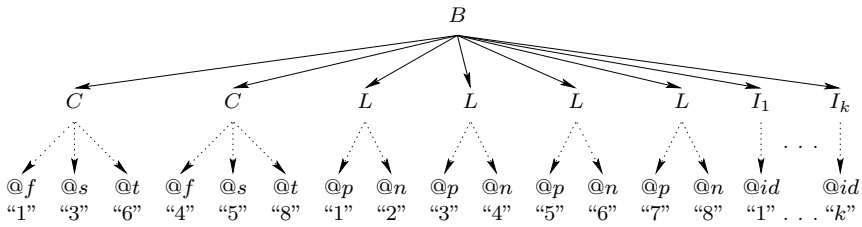


Fig. 11. XML tree T_θ, defined in case (I) of the proof of Lemma 6.21, representing propositional formula $\theta = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$.

complement of CERTAIN-ANSWERS(Q), that is, for every propositional formula θ in 3-CNF, there exists a PTIME constructible XML tree T_θ conforming to $D_{\mathbf{S}}$ such that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = false$. Simple DTD $D_{\mathbf{S}}$ is defined as follows. Let $k = \max\{\#_b(w) \mid b \in alph(w)\}$, $E_{\mathbf{S}} = \{B, C, L, I_1, \ldots, I_k\}$ be a set of element types and $A_{\mathbf{S}} = \{@f, @s, @t, @p, @n, @id\}$ a set of attributes. Then $D_{\mathbf{S}} = (P_{\mathbf{S}}, R_{\mathbf{S}}, B)$ is a DTD over $(E_{\mathbf{S}}, A_{\mathbf{S}})$, where $P_{\mathbf{S}}$ is defined as:

$$\begin{aligned} P_{\mathbf{S}}(B) &= C^* L^* I_1^* \cdots I_k^*, \\ P_{\mathbf{S}}(\ell) &= \varepsilon, \quad \text{for every } \ell \in E_{\mathbf{S}} \setminus \{B\}. \end{aligned}$$

and $R_{\mathbf{S}}$ is defined as:

$$\begin{array}{llll} R_{\mathbf{S}}(B) = \emptyset, & & R_{\mathbf{S}}(C) = \{@f, @s, @t\}, & \\ R_{\mathbf{S}}(L) = \{@p, @n\}, & & R_{\mathbf{S}}(I_i) = \{@id\}, & \text{for every } i \in [1,k]. \end{array}$$

XML trees conforming to $D_{\mathbf{S}}$ are used to represent propositional formulae. Let θ be 3-CNF formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$. To construct tree T_θ, first we assign a distinct natural number to each literal, say

$$\begin{array}{llll} x_1 \mapsto 1, & x_2 \mapsto 3, & x_3 \mapsto 5, & x_4 \mapsto 7, \\ \neg x_1 \mapsto 2, & \neg x_2 \mapsto 4, & \neg x_3 \mapsto 6, & \neg x_4 \mapsto 8. \end{array} \tag{4}$$

Then we represent each clause of θ as a node of type C, being the values of attributes $@f$, $@s$, $@t$ the first, second and third literal of that clause, respectively. For each propositional variable x in θ, we use the attributes $@p$, $@n$ of a node of type L to store the values assigned to x and $\neg x$, respectively. Finally, for every $i \in [1,k]$ we create a node of type I_i with value i in its attribute $@id$ (it will become clear why we need these nodes when we define $D_{\mathbf{T}}$ and $\Sigma_{\mathbf{ST}}$). Tree T_θ for the formula defined above is shown in Figure 11.

$\mathcal{C}$-DTD $D_{\mathbf{T}}$ is defined as follows. Let $E_{\mathbf{T}} = \{B, C\} \cup \{G_i \mid i \in [1,8]\} \cup alph(r)$ be a set of element types such that $(\{B, C\} \cup \{G_i \mid i \in [1,8]\}) \cap alph(r) = \emptyset$, and let $A_{\mathbf{T}} = \{@f, @s, @t, @id, @e, @c, @d\}$ be a set of attributes. Then $D_{\mathbf{T}} = (P_{\mathbf{T}}, R_{\mathbf{T}}, B)$ is a DTD over $(E_{\mathbf{T}}, A_{\mathbf{T}})$, where $P_{\mathbf{T}}$ is defined as:

$$\begin{array}{llllll} P_{\mathbf{T}}(B) = G_1^* C^*, & P_{\mathbf{T}}(G_1) = G_2^* G_3^*, & & P_{\mathbf{T}}(G_2) = G_4^* G_5^*, \\ P_{\mathbf{T}}(G_4) = G_7^*, & P_{\mathbf{T}}(G_7) = G_8^*, & & P_{\mathbf{T}}(G_8) = \varepsilon, \\ P_{\mathbf{T}}(G_5) = r, & P_{\mathbf{T}}(\ell) = \varepsilon, & \text{for every } \ell \in alph(r), & P_{\mathbf{T}}(G_3) = G_6^*, \\ P_{\mathbf{T}}(G_6) = \varepsilon, & P_{\mathbf{T}}(C) = \varepsilon. & & \end{array}$$

and $R_{\mathbf{T}}$ is defined as:

$$\begin{array}{llll} R_{\mathbf{T}}(B) = \emptyset, & R_{\mathbf{T}}(G_1) = \{@e\}, \\ R_{\mathbf{T}}(G_2) = \{@e\}, & R_{\mathbf{T}}(G_4) = \emptyset, \\ R_{\mathbf{T}}(G_7) = \emptyset, & R_{\mathbf{T}}(G_8) = \{@c\}, \\ R_{\mathbf{T}}(G_5) = \emptyset, & R_{\mathbf{T}}(\ell) = \emptyset, \text{ for every } \ell \in alph(r) \setminus (\{a_0\} \cup Y \cup alph(w)), \\ R_{\mathbf{T}}(G_3) = \emptyset, & R_{\mathbf{T}}(\ell) = \{@id\}, \text{ for every } \ell \in alph(w), \\ R_{\mathbf{T}}(a_0) = \{@c\}, & R_{\mathbf{T}}(\ell) = \{@d\}, \text{ for every } \ell \in Y, \\ R_{\mathbf{T}}(G_6) = \{@d\}, & R_{\mathbf{T}}(C) = \{@f, @s, @t\}. \end{array}$$

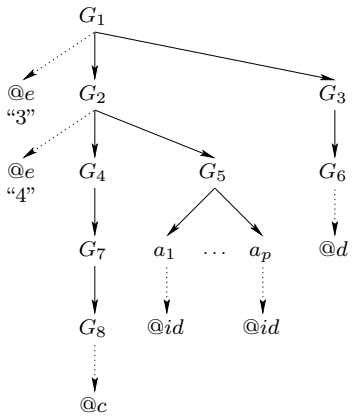
Finally, the set $\Sigma_{\mathbf{ST}}$ of fully-specified STDs is defined as follows. The first rule of $\Sigma_{\mathbf{ST}}$ is defined as:

$$B[C(@f = x, @s = y, @t = z)] \ :- \ B[C(@f = x, @s = y, @t = z)].$$

This rule says that every node of type C in a source tree T must appear in every solution for T. The second rule of $\Sigma_{\mathbf{ST}}$ is defined as follows:

$$B[G_1(@e = x)[G_2(@e = y)[G_4[G_7[G_8]], G_5[\bigwedge_{b \in alph(w)} \bigwedge_{i=1}^{\#_b(w)} b(@id = u_i)]],$$
$$G_3[G_6]] \ :- \ B[L(@p = x, @n = y), \bigwedge_{i=1}^{k} I_i(@id = u_i)].$$

As in the proof of Lemma 6.20, we use symbol $\bigwedge$ to denote a sequence of formulae separated by commas. Indeed, conjunction is not allowed in tree-pattern formulae. Thus, for example, $\bigwedge_{i=1}^{\#_b(w)} b(@id = u_i)$ denotes formula $b(@id = u_1), \ldots, b(@id = u_{\#_b(w)})$. To explain the meaning of this rule, we considered again tree T_θ shown in Figure 11. Assuming that $w = a_1 \cdots a_p$ ($p \geq 0$), the previous rule says that for each pair of complementary literals, say $(3, 4)$, and for each solution T' for T_θ, there should exist a node v_1 of type G_1 in T' having the following descendants:



In this figure, assume that v_i ($i \in [1, 8]$) is the identifier of the node of type G_i. Node v_1 has an attribute $@e$ with value 3 and it has two children, node v_2 of type G_2 and node v_3 of type G_3. Node v_2 has an attribute $@e$ with value 4 and it has

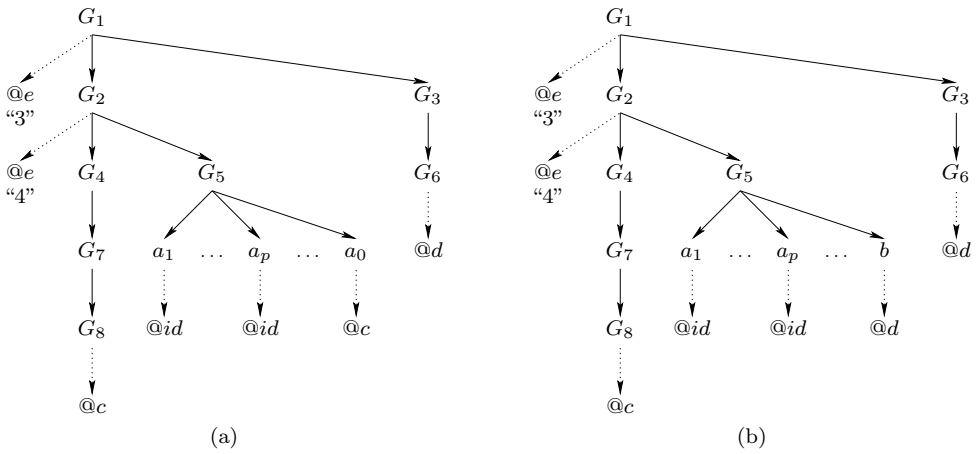
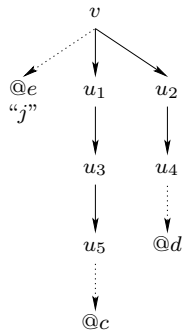


Fig. 12. Possible ways of repairing string w in case (I) of the proof of Lemma 6.21.

two children, node v_4 of type G_4 and node v_5 of type G_5. Let $b \in alph(w)$. Node v_5 has $\#_b(w)$ children of this type. Furthermore, these b-nodes are assigned pairwise distinct values in attribute @id, taken from attribute @id of the nodes of type I_1, ..., I_k in T_θ. We observe that if $w = \varepsilon$, then v_5 has no children.

Since $w \notin \pi(r)$, to construct a solution T' for T_θ we need to replace w by a string $w' \in \pi(r)$ such that the generated tree continue satisfying $\Sigma_{\mathbf{ST}}$. By Claim 6.22, we know that there exists $i \in [1, n]$ such that $w' \preceq_w w_i$. First assume that $i = 1$. Then, by definition of $\preceq_w$, we have that $a_0 \in alph(w')$ and, therefore, the generated solution is of the form shown in Figure 12 (a). Second, assume that $i \neq 1$. Since the children of v of type b ($b \in alph(w)$) have pairwise distinct values in attribute @id, we cannot remove any of these nodes in order to construct a solution for T'. Thus, $w \preceq w'$ and, therefore, $w \preceq w_i$. We conclude that $i \in I$. Given that $w' \preceq_w w_i$, there exists $b \in Y$ such that $b \in alph(w')$ and, hence, the generated solution is of the form shown in Figure 12 (b).

We note that when choosing whether to replace w by either a string contained in w_1 or a string contained in w_i ($i \in I$), we are actually choosing the truth values for x_2 and $\neg x_2$. We say that a literal j has been assigned value 0 if j is the value of attribute @e of a node v having the following descendants:



That is, v has a "great-grandchild" u_5 with an attribute $@c$ and it has a "grandchild" u_4 with an attribute $@d$. For example, in the solution for T_θ shown in Figure 13 (a), we have chosen value 0 for x_2 since 3 is the value of attribute $@e$ of a node (of type G_1) having a "great-grandchild" (of type a_0) with an attribute $@c$ and having a "grandchild" (of type G_6) with an attribute $@d$. On the other hand, in the solution for T_θ shown in Figure 13 (b), we have chosen value 0 for $\neg x_2$ since 4 is the value of attribute $@e$ of a node (of type G_2) having a "great-grandchild" (of type G_8) with an attribute $@c$ and having a "grandchild" (of type b) with an attribute $@d$. It is worth mentioning that it is possible to choose value 0 for both x_2 and $\neg x_2$. After defining query Q, we will show that this alternative does not cause any problems. From now on, we say that a node v is a $(@c, @d)$-node if v has a "great-grandchild" with an attribute $@c$ and it has a "grandchild" with an attribute $@d$. Thus, we say that we have chosen value 0 for a propositional variable x if there exists a $(@c, @d)$-node v having in attribute $@e$ the value assigned to x in T_θ.

Boolean $\mathcal{CTQ}$-query Q is defined as:

$$\begin{aligned}\exists x \exists y \exists z \exists u_1 \exists u_2 \exists u_3 \exists v_1 \exists v_2 \exists v_3\ (&C(@f = x, @s = y, @t = z) \wedge \\ &\_(@e = x)[\_[\_[\_(@c = u_1)]],\ \_[\_(@d = v_1)]] \wedge \\ &\_(@e = y)[\_[\_[\_(@c = u_2)]],\ \_[\_(@d = v_2)]] \wedge \\ &\_(@e = z)[\_[\_[\_(@c = u_3)]],\ \_[\_(@d = v_3)]]).\end{aligned}$$

Intuitively, query Q says that there exists a node v of type C such that each "literal" of v is assigned value 0, that is, the values i_1, i_2, i_3 of the attributes $@f$, $@s$, $@t$ of v, respectively, are such that for every $i \in \{i_1, i_2, i_3\}$, there exists a $(@c, @d)$-node v' having i as value of attribute $@e$.

Now we prove that for every 3-CNF propositional formula θ, we have that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = \mathit{false}$, where T_θ is constructed from θ in PTIME as shown above.

($\Rightarrow$) Assume that θ is satisfiable and let σ be a truth assignment satisfying θ. Define a solution T' for T_θ as follows. The structure of the C-nodes of T' is copied from T_θ. For every propositional variables x in θ, we define a G_1-node v_1 of T' as follows. Node v_1 has the value assigned to x in T_θ in attribute $@e$ and it has two children, node v_2 of type G_2 and node v_3 of type G_3. Node v_3 has only one child (of type G_6). Node v_2 has the value assigned to $\neg x$ in T_θ in attribute $@e$ and it has two children, node v_4 of type G_4 and node v_5 of type G_5. Node v_4 has only one child (of type G_7), which in turn has only one child (of type G_8). The sequence of children of v_5 is defined as follows. If $\sigma(x) = 0$, then the string of types of the children of v_5 is w_1. For every $b \in \mathit{alph}(w)$, exactly $\#_b(w)$ children of v_5 of type b are assigned pairwise distinct values in attribute $@id$, taken from attribute $@id$ of the nodes of type I_1, ..., I_k in T_θ. Furthermore, attributes with fresh null values are added to the descendants of v_1 according to $R_{\mathbf{T}}$. We note that in this case we have assigned value 0 to $\sigma(x)$ since the value assigned to x in T_θ is the value of attribute $@e$ of a $(@c, @d)$-node (of type G_1). We also note that in this case we have not assigned value 0 to $\sigma(\neg x)$. If $\sigma(x) = 1$, then the string of types of the children of v_5 is w_2. For every $b \in \mathit{alph}(w)$, exactly $\#_b(w)$ children of v_5 of type b are assigned pairwise distinct values in attribute $@id$, taken from attribute $@id$ of

the nodes of type I_1, ..., I_k in T_θ. Furthermore, attributes with fresh null values are added to the descendants of v according to $R_{\mathbf{T}}$. We note that in this case we have assigned value 0 to $\sigma(\neg x)$ since the value assigned to $\neg x$ in T_θ is the value of attribute $@e$ of a $(@c, @d)$-node (of type G_2). We also note that in this case we have not assigned value 0 to $\sigma(x)$.

It is straightforward to prove that T' conforms to $D_{\mathbf{T}}$ and satisfies $\Sigma_{\mathbf{ST}}$. Furthermore, $T' \not\models Q$ since σ satisfies θ and, therefore, at least one literal per clause C is not assigned value 0. We conclude that $\underline{certain}(Q, T_\theta) = \mathit{false}$ since T' is a solution for T_θ.

($\Leftarrow$) Assume that $\underline{certain}(Q, T_\theta) = \mathit{false}$ and let T' be a solution for T_θ such that $T' \not\models Q$. We define a truth assignment for the propositional variables of θ as follows. For every clause in θ, find a C-node v of T' such that the values i_1, i_2, i_3 of attributes $@f$, $@s$, $@t$ of v are the values assigned in T_θ to the literals of that clause. Since $T' \not\models Q$, there exists $i \in \{i_1, i_2, i_3\}$ such that no $(@c, @d)$-node v' has i as value of attribute $@e$. If i corresponds to a positive literal x, then define $\sigma(x)$ as 1. If i corresponds to a negative literal $\neg x$, then define $\sigma(x)$ as 0. We will show that σ is well defined. On the contrary, assume that there exists a propositional variable x such that 1 was assigned to $\sigma(x)$ and $\sigma(\neg x)$. Let i, j be the values assigned in T_θ to x and $\neg x$. Then no $(@c, @d)$-node v' has j as value of attribute $@e$. Let v_1 be a G-node of T' that satisfies the left hand side of the second rule of $\Sigma_{\mathbf{ST}}$ instantiated on the following values from T_θ:

$$B[L(@p = i, @n = j), \bigwedge_{i=1}^{k} I_i(@id = i)].$$

Then v_1 has a child of type G_2, which in turn has a child of type G_5. Let v_5 be the identifier of this node and $w' = \lambda_{T'}(\mathit{children}(v_5))$. Since no $(@c, @d)$-node v' has j as value of attribute $@e$, we have that $\mathit{alph}(w') \cap Y = \emptyset$. Thus, by Claim 6.22, we conclude that $w' \preceq_w w_1$ and, therefore, $a_0 \in \mathit{alph}(w')$. But $@c \in R_{\mathbf{T}}(a_0)$ and, hence, v_1 is a $(@c, @d)$-node having i as value of attribute $@e$. We conclude that there exists a $(@c, @d)$-node having i as value of attribute $@e$, which contradicts the fact that 1 was assigned to $\sigma(x)$.

Since σ is well defined and σ satisfies θ (by definition of σ), we conclude that θ is satisfiable. This concludes the proof of case (I).

(II) Second, assume that there exists distinct $i, j \in [1, n]$ and $a \in \mathit{alph}(w)$ such that (1) $\#_a(w_i) < \#_a(w)$ and (2) for every $b \in \mathit{alph}(w)$, we have that $\#_b(w_i) \leq \#_b(w_j)$ or $\#_b(w_j) \geq \#_b(w)$. Without loss of generality, assume that $i = 1$ and $j = 2$. Here, we consider two sub-cases.

(II.1) Assume that for every $b \in \mathit{alph}(w)$ such that $\#_b(w_1) < \#_b(w)$, we have that $\#_b(w_1) = \#_b(w_2)$. Given that $w_1 \not\preceq_w w_2$, there exist $b \in \mathit{alph}(w_2) \setminus \mathit{alph}(w)$ such that $b \notin \mathit{alph}(w_1)$, and given that $w_2 \not\preceq_w w_1$, there exist $c \in \mathit{alph}(w_1) \setminus \mathit{alph}(w)$ such that $c \notin \mathit{alph}(w_2)$.

Let w' be a string such that: $\mathit{alph}(w') = \mathit{alph}(w)$ and $\#_b(w') = \min\{\#_b(w_1), \#_b(w)\}$, for every $b \in \mathit{alph}(w')$. We note that $w' \notin \pi(r)$ since $w_1 \preceq_w w'$ and $w' \not\preceq_w w_1$ ($c \in \mathit{alph}(w_1)$ and $c \notin \mathit{alph}(w)$). Since $w_1 \in \mathit{rep}(w, r)$,

there exists string $s \preceq w$ such that $w_1 \in \mathit{min\_ext}(s, r)$. Then $s \preceq w'$. On the contrary, assume that $s \not\preceq w'$. Hence, given that $s \preceq w$, there exists $d \in \mathit{alph}(w)$ such that $\#_d(w') < \#_d(s) \leq \#_d(w)$. Thus, by definition of w' we have that $\#_d(w_1) = \#_d(w') < \#_d(s)$, which contradicts the fact that $s \preceq w_1$. Given that $s \preceq w'$, we have that $\{w'' \mid w' \preceq w'' \text{ y } w'' \in \pi(r)\} \subseteq \{w'' \mid s \preceq w'' \text{ y } w'' \in \pi(r)\}$ and, therefore, $w_1 \in \mathit{min\_ext}(w', r)$. We conclude that $w_1 \in \mathit{rep}(w', r)$. Similarly, we conclude that $w_2 \in \mathit{rep}(w', r)$.

Now we show that $w_1 \in \max_{\preceq_{w'}} \mathit{rep}(w', r)$. On the contrary, assume that there exists $s' \in \mathit{rep}(w', r)$ such that $w_1 \prec_{w'} s'$. Given that $w_1 \preceq_{w'} s'$, for every $a \in \mathit{alph}(w) = \mathit{alph}(w')$ such that $\#_a(w_1) < \#_a(w)$, we have that $\#_a(w_1) = \#_a(w') \leq \#_a(s')$. Furthermore, given that $w_1 \preceq_{w'} s'$, we have that $\mathit{alph}(s') \setminus \mathit{alph}(w') \subseteq \mathit{alph}(w_1) \setminus \mathit{alph}(w')$. Thus, given that $\mathit{alph}(w') = \mathit{alph}(w)$, we conclude that $w_1 \preceq_w s'$. Since $\mathit{alph}(w) \subseteq \mathit{alph}(s')$, by Claim 6.22 we have that $s' \preceq_w w_i$ ($i \in [1, n]$). If we assume that $i = 1$, then $\mathit{alph}(w_1) \setminus \mathit{alph}(w) \subseteq \mathit{alph}(s') \setminus \mathit{alph}(w)$ and, therefore, $\mathit{alph}(w_1) \setminus \mathit{alph}(w') \subseteq \mathit{alph}(s') \setminus \mathit{alph}(w')$. Thus, given that $w' \preceq w$, we conclude that $s' \preceq_{w'} w_1$, which contradicts our assumption that $w_1 \prec_{w'} s'$. Thus, $s' \preceq_w w_i$, where $i \neq 1$. Given that $w_1 \preceq_w s'$, we conclude that $w_1 \preceq_w w_i$, which contradicts our original assumption. Thus, $w_1 \in \max_{\preceq_{w'}} \mathit{rep}(w', r)$ and, similarly, $w_2 \in \max_{\preceq_{w'}} \mathit{rep}(w', r)$.

Given that there exist $b \in \mathit{alph}(w_2) \setminus \mathit{alph}(w)$ such that $b \notin \mathit{alph}(w_1)$ and there exist $c \in \mathit{alph}(w_1) \setminus \mathit{alph}(w)$ such that $c \notin \mathit{alph}(w_2)$, we have that $w_1 \not\preceq_{w'} w_2$ and $w_2 \not\preceq_{w'} w_1$. Thus, given that $w' \notin \pi(r)$, $w_1 \in \max_{\preceq_{w'}} \mathit{rep}(w', r)$, $w_2 \in \max_{\preceq_{w'}} \mathit{rep}(w', r)$, $w' \preceq w_1$ and $w' \preceq w_2$, we have that w' is a string satisfying case (I) of this proof. Hence, we use case (I) to prove that $\mathcal{C}$ is strongly coNP-complete.

(II.2) Assume that there exists $a_0 \in \mathit{alph}(w)$ such that $\#_{a_0}(w_1) < \#_{a_0}(w)$ and $\#_{a_0}(w_1) < \#_{a_0}(w_2)$. Since $w_1 \not\preceq_w w_2$, we have that there exists $b \in \mathit{alph}(w_2) \setminus \mathit{alph}(w)$ such that $b \notin \mathit{alph}(w_1)$. Let X be set of all such elements, that is, $X = \{b \in \mathit{alph}(r) \mid \text{there exists } i \in [2, n] \text{ such that } b \in \mathit{alph}(w_i) \setminus \mathit{alph}(w) \text{ and } b \notin \mathit{alph}(w_1)\}$. Then $X \neq \emptyset$.

To show that $\mathcal{C}$ is strongly coNP-complete, we define a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ and a Boolean $\mathcal{CTQ}$-query Q such that $D_{\mathbf{S}}$ is a simple DTD, $D_{\mathbf{T}}$ is a $\mathcal{C}$-DTD, $\Sigma_{\mathbf{ST}}$ is a set of fully-specified STDs and 3SAT can be reduced to the complement of Certain-Answers(Q), that is, for every propositional formula θ in 3-CNF, there exists a PTIME constructible XML tree T_θ conforming to $D_{\mathbf{S}}$ such that θ is satisfiable if and only if $\underline{\mathit{certain}}(Q, T_\theta) = \mathit{false}$. Simple DTD $D_{\mathbf{S}}$ is defined as follows. Let $k = \max\{\#_b(w_1) \mid b \in \mathit{alph}(w)\}$, $E_{\mathbf{S}} = \{B, C, L, I_1, \ldots, I_k\}$ be a set of element types and $A_{\mathbf{S}} = \{@f, @s, @t, @p, @n, @id\}$ be a set of attributes. Then $D_{\mathbf{S}} = (P_{\mathbf{S}}, R_{\mathbf{S}}, B)$ is a DTD over $(E_{\mathbf{S}}, A_{\mathbf{S}})$, where $P_{\mathbf{S}}$ is defined as:

$$\begin{aligned} P_{\mathbf{S}}(B) &= C^* L^* I_1^* \cdots I_k^*, \\ P_{\mathbf{S}}(\ell) &= \varepsilon, \quad \text{for every } \ell \in E_{\mathbf{S}} \setminus \{B\}. \end{aligned}$$

and $R_{\mathbf{S}}$ is defined as:

$$\begin{array}{llll} R_{\mathbf{S}}(B) = \emptyset, & & R_{\mathbf{S}}(C) = \{@f, @s, @t\}, & \\ R_{\mathbf{S}}(L) = \{@p, @n\}, & & R_{\mathbf{S}}(I_i) = \{@id\}, & \text{for every } i \in [1, k]. \end{array}$$

XML trees conforming to $D_{\mathbf{S}}$ are used to represent propositional formulae exactly as in case (I). Assume that θ and T_θ are as shown in Figure 11, where variables x_1, x_2, x_3, x_4 and their negations have been assigned the same values as in case (I) (see equation (4)).

$\mathcal{C}$-DTD $D_{\mathbf{T}}$ is defined as follows. Let $E_{\mathbf{T}} = \{B, G, C, N, F\} \cup alph(r)$ be a set of element types such that $\{B, G, C, N, F\} \cap alph(r) = \emptyset$, and let $A_{\mathbf{T}} = \{@f, @s, @t, @id, @e\}$ be a set of attributes. Then $D_{\mathbf{T}} = (P_{\mathbf{T}}, R_{\mathbf{T}}, B)$ is a DTD over $(E_{\mathbf{T}}, A_{\mathbf{T}})$, where $P_{\mathbf{T}}$ is defined as:

$$\begin{array}{lllllllll} P_{\mathbf{T}}(B) & = & G^*C^*, & P_{\mathbf{T}}(G) & = & r, & P_{\mathbf{T}}(\ell) & = & \varepsilon, \text{ for every } \ell \in alph(r) \setminus \{a_0\}, \\ P_{\mathbf{T}}(a_0) & = & N^*F^*, & P_{\mathbf{T}}(N) & = & \varepsilon, & P_{\mathbf{T}}(F) & = & \varepsilon, \\ P_{\mathbf{T}}(C) & = & \varepsilon. & & & & & & \end{array}$$

and $R_{\mathbf{T}}$ is defined as:

$$\begin{array}{lllllll} R_{\mathbf{T}}(B) & = & \emptyset, & R_{\mathbf{T}}(C) & = & \{@f, @s, @t\}, \\ R_{\mathbf{T}}(G) & = & \emptyset, & R_{\mathbf{T}}(\ell) & = & \{@id\}, \text{ for every } \ell \in alph(w) \setminus \{a_0\} \\ R_{\mathbf{T}}(a_0) & = & \{@id, @e\}, & R_{\mathbf{T}}(\ell) & = & \{@f\}, \text{ for every } \ell \in X, \\ R_{\mathbf{T}}(N) & = & \{@e\}, & R_{\mathbf{T}}(\ell) & = & \emptyset, \text{ for every } \ell \in alph(r) \setminus (X \cup alph(w)), \\ R_{\mathbf{T}}(F) & = & \{@f\}. & & & \end{array}$$

Finally, set $\Sigma_{\mathbf{ST}}$ of fully-specified STDs is defined as follows. The first rule of $\Sigma_{\mathbf{ST}}$ is defined as:

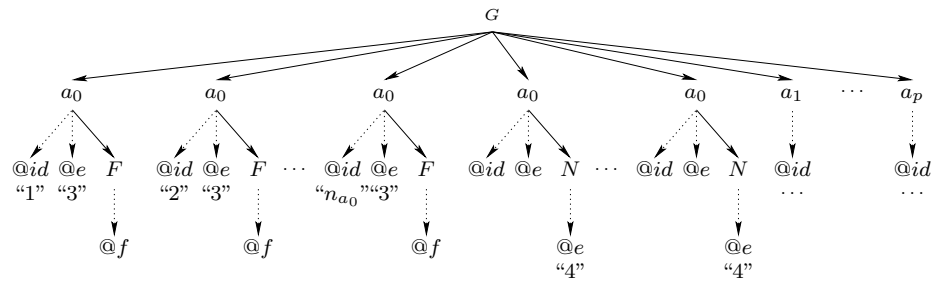
$$B[C(@f = x, @s = y, @t = z)] \ :\!- \ B[C(@f = x, @s = y, @t = z)].$$

This rule says that every node of type C in a source tree T must appear in every solution for T. The second rule of $\Sigma_{\mathbf{ST}}$ is defined as (we use $\bigwedge$ to denote a sequence of formulae separated by commas, as in case (I)):

$$\begin{aligned} B[G[\bigwedge_{b \in alp(w) \setminus \{a_0\}} \Big(\bigwedge_{i=1}^{n_b} b(@id = u_i), \bigwedge_{i=n_b+1}^{\#_b(w)} b \Big), \\ \bigwedge_{i=1}^{n_{a_0}} a_0(@id = u_i, @e = x)[F], \bigwedge_{i=n_{a_0}+1}^{\#_{a_0}(w)} a_0[N(@e = y)]]] \ :\!- \\ B[L(@p = x, @n = y), \bigwedge_{i=1}^{k} I_i(@id = u_i)], \end{aligned}$$

where $n_b = \min\{\#_b(w), \#_b(w_1)\}$, for every $b \in alph(w)$. In particular, $n_{a_0} = \min\{\#_{a_0}(w), \#_{a_0}(w_1)\} = \#_{a_0}(w_1)$.

To explain the meaning of the previous rule, we considered again tree T_θ shown in Figure 11. Assuming that $w = a_0^{\#_{a_0}(w)} a_1 \cdots a_p$, where $a_i \neq a_0$ ($i \in [1, p]$), the previous rule says that for each pair of complementary literals, say $(3, 4)$, and for each solution T' for T_θ, there should exist a node v of type G in T' having the following descendants:



In this tree, v has $\#_{a_0}(w)$ children of type a_0. The first n_{a_0} of these children are defined as follows. Let $i \in [1, n_{a_0}]$. Then the i-th child of v, from left to right, of type a_0 has a child of type F and it has i as value of attribute $@id$ and 3 as value of attribute $@e$ since the second rule of $\Sigma_{\mathbf{ST}}$ says that for this node $@id = u_i$ and $@e = x$, being u_i and x the values of attribute $@id$ of the node of type I_i in T_θ and attribute $@p$ of a node of type L in T_θ, respectively. The remaining $(\#_{a_0}(w) - n_{a_0})$ children of v of type a_0 are defined as follows. Let $i \in [n_{a_0} + 1, \#_{a_0}(w)]$. Then the i-th child of v of type a_0 has as child a node of type N having 4 as value of attribute $@e$. Moreover, for every $b \in \mathit{alph}(w) \setminus \{a_0\}$, the first n_b children of v of type b have pairwise distinct values in attribute $@id$.

Since $w \notin \pi(r)$, to construct a solution T' for T_θ we need to replace w by a string $w' \in \pi(r)$ such that the generated tree continue satisfying $\Sigma_{\mathbf{ST}}$. By Claim 6.22, we know that there exists $i \in [1, n]$ such that $w' \preceq_w w_i$. First assume that $i = 1$. Then, by definition of $\preceq_w$, we have that $\#_{a_0}(w') \leq \#_{a_0}(w_i) = n_{a_0}$. Thus, given that the first n_{a_0} children of v of type a_0 have pairwise distinct values in attribute $@id$, to construct w' from w we have to remove the last $(\#_{a_0}(w) - n_{a_0})$ children of v of type a_0 and choose among the first n_{a_0} nodes of this type where to place the descendants of the removed a_0-nodes. For example, we can make them children of the first a_0-node of w, generating the subtree shown in Figure 13 (a). Second, assume that $i \neq 1$ and let s be a string such that $\mathit{alph}(s) = \mathit{alph}(w)$ and $\#_b(s) = n_b$, for every $b \in \mathit{alph}(s)$. Since the first n_b children of v of type b have pairwise distinct values in attribute $@id$, for every $b \in \mathit{alph}(w)$, we conclude that $s \preceq w'$. Given that $s \preceq w'$, $w' \preceq_w w_i$, and $\#_b(s) = \min\{\#_b(w), \#_b(w_1)\}$ for every $b \in \mathit{alph}(w)$, we have that for every $c \in \mathit{alph}(w)$ such that $\#_c(w_i) < \#_c(w)$, it is the case that $\#_c(w_i) \geq \#_c(w') \geq \#_c(s) = \#_c(w_1)$. Thus, given that $w_1 \not\preceq_w w_i$, there exists $b \in \mathit{alph}(w_i) \setminus \mathit{alph}(w)$ such that $b \notin \mathit{alph}(w_1)$. But $\mathit{alph}(w_i) \setminus \mathit{alph}(w) \subseteq \mathit{alph}(w') \setminus \mathit{alph}(w)$ and, thus, $b \in \mathit{alph}(w')$. Therefore, there exists a node in w' of type $b \in X$ and, hence, the generated subtree is of the form shown in Figure 13 (b). We observe that w' must contain at least n_{a_0} nodes of type a_0, each of them having value 3 in attribute $@e$, since the first n_{a_0} children of v of type a_0 have pairwise distinct values in attribute $@id$.

We note that when choosing whether to replace w by either a string contained in w_1 or a string contained in w_i ($i \neq 1$, $i \in [1, n]$), we are actually choosing the truth values for x_2 and $\neg x_2$. We say that a literal j has been assigned value 0 if j is the value of attribute $@e$ of a node having a sibling with an attribute $@f$. For example, in the solution for T_θ shown in Figure 13 (a), we have chosen value 0 for $\neg x_2$ since 4 is the value of attribute $@e$ of a node (of type N) having a sibling (of type F) with an attribute $@f$. On the other hand, in the solution for T_θ shown in

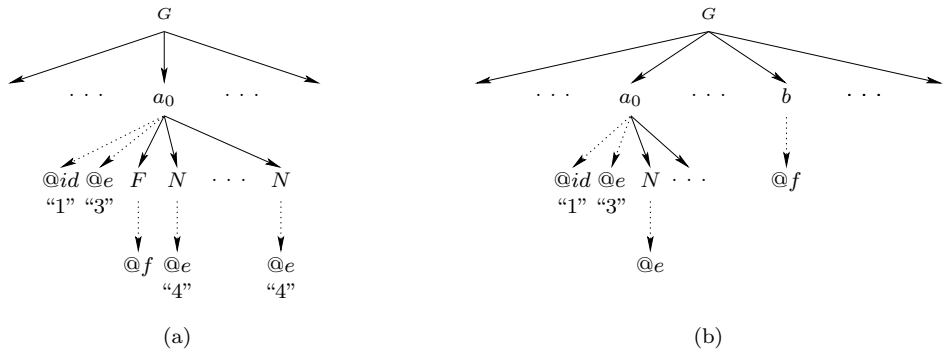


Fig. 13. Possible ways of repairing string w in case (II.2) of the proof of Lemma 6.21.

Figure 13 (b), we have chosen value 0 for x_2 since 3 is the value of attribute $@e$ of a node (of type a_0) having a sibling (of type b) with an attribute $@f$. We will use this property to prove that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = false$. It is worth mentioning that it is possible to choose value 0 for both x_2 and $\neg x_2$. After defining query Q, we will show that this alternative does not cause any problems.

Boolean $\mathcal{CTQ}$-query Q is defined as:

$$\exists x \exists y \exists z \exists u_1 \exists u_2 \exists u_3 \ (C(@f = x, @s = y, @t = z) \wedge \\ \_[\_(@f = u_1), \_(@e = x)] \ \wedge \ \_[\_(@f = u_2), \_(@e = y)] \wedge \\ \_[\_(@f = u_3), \_(@e = z)]).$$

Intuitively, query Q says that there exists a node v of type C such that each "literal" of v is assigned value 0, that is, the values i_1, i_2, i_3 of the attributes $@f$, $@s$, $@t$ of v, respectively, are such that for every $i \in \{i_1, i_2, i_3\}$, there exists a node v' having i as value of attribute $@e$ and having a sibling with an attribute $@f$.

Now we prove that for every 3-CNF propositional formula θ, we have that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = false$, where T_θ is constructed from θ in PTIME as shown above.

($\Rightarrow$) Assume that θ is satisfiable and let σ be a truth assignment satisfying θ. Define a solution T' for T_θ as follows. The structure of the C-nodes of T' is copied from T_θ. For every propositional variables x in θ, we define a G-node v of T' as follows. If $\sigma(x) = 1$, then the string of types of the children of v is w_1. For every $b \in alph(w)$, exactly n_b children of v of type b are assigned pairwise distinct values in attribute $@id$, taken from attribute $@id$ of the nodes of type I_1, ..., I_k in T_θ. The value assigned to x in T_θ is the value of attribute $@e$ of the n_{a_0} children of v of type a_0. Furthermore, the first child of v of type a_0 has a child of type F and $(\#_{a_0}(w) - n_{a_0})$ children of type N, each of them having the value assigned to $\neg x$ in T_θ as value of attribute $@e$. The remaining $n_{a_0} - 1$ children of v of type a_0 have exactly one child (of type F). Finally, each node of type F has an attribute $@f$ with value a fresh null. We note that in this case we have assigned value 0 to $\sigma(\neg x)$, since the value assigned to $\neg x$ in T_θ is the value of attribute $@e$ of a node (of type N) having a sibling (of type F) with an attribute $@f$. We also note that in this case we have not assigned value 0 to $\sigma(x)$. Now, let us consider $\sigma(x) = 0$. In this

case, the string of types of the children of v is w_2. For every $b \in alph(w)$, exactly n_b children of v of type b are assigned pairwise distinct values in attribute $@id$, taken from attribute $@id$ of the nodes of type $I_1, \ldots, I_k$ in T_θ. The value assigned to x in T_θ is the value of attribute $@e$ of the first n_{a_0} children of v of type a_0. Each of these nodes also has a child of type F, and these are the only descendant of v having children of type F. Since $\#_{a_0}(w_1) < \#_{a_0}(w_2)$, there exists at least one child of v of type a_0 not having a child of type F. Let v' one of these nodes. Then v' has as children $\#_{a_0}(w_1) - n_{a_0}$ nodes of type N, each of them having the value assigned to $\neg x$ in T_θ as value of attribute $@e$. Finally, every node of type either F or $b \in alph(w_2) \setminus alph(w_1)$ has an attribute $@f$ with value a fresh null. We note that in this case we have assigned value 0 to $\sigma(x)$, since the value assigned to x in T_θ is the value of attribute $@e$ of a node (of type a_0) having a sibling (of type $b \in alph(w_2) \setminus alph(w_1)$) with an attribute $@f$. We also note that in this case we have not assigned value 0 to $\sigma(\neg x)$ (no node of type a_0 has both a child of type F and a child of type N).

It is straightforward to prove that T' conforms to $D_{\mathbf{T}}$ and satisfies $\Sigma_{\mathbf{ST}}$. Furthermore, $T' \not\models Q$ since σ satisfies θ and, therefore, at least one literal per clause C is not assigned value 0. We conclude that $\underline{certain}(Q, T_\theta) = false$ since T' is a solution for T_θ.

($\Leftarrow$) Assume that $\underline{certain}(Q, T_\theta) = false$ and let T' be a solution for T_θ such that $T' \not\models Q$. We define a truth assignment for the propositional variables of θ as follows. For every clause in θ, find a C-node v of T' such that the values i_1, i_2, i_3 of attributes $@f$, $@s$, $@t$ of v are the values assigned in T_θ to the literals of that clause. Since $T' \not\models Q$, there exists $i \in \{i_1, i_2, i_3\}$ such that every node v' having i as value of attribute $@e$ does not have a sibling with an attribute $@f$. If i corresponds to a positive literal x, then define $\sigma(x)$ as 1. If i corresponds to a negative literal $\neg x$, then define $\sigma(x)$ as 0. We will show that σ is well defined. On the contrary, assume that there exists a propositional variable x such that 1 was assigned to $\sigma(x)$ and $\sigma(\neg x)$. Let i, j be the values assigned in T_θ to x and $\neg x$. Then every node v' having j as value of attribute $@e$ does not have a sibling with an attribute $@f$. Let v'' be a G-node of T' that satisfies the left hand side of the second rule of $\Sigma_{\mathbf{ST}}$ instantiated on the following values from T_θ:

$$B[L(@p = i, @n = j), \bigwedge_{i=1}^{k} I_i(@id = i)],$$

and let $w'' = \lambda_{T'}(children(v''))$. This node has at least n_{a_0} children of type a_0 having i as value of attribute $@e$ and has at least $\#_{a_0}(w_1) - n_{a_0} > 0$ descendants of type N having j as value of attribute $@e$. Thus, given that every node v' having j as value of attribute $@e$ does not have a sibling with an attribute $@f$, we have that v'' has at least $n_{a_0} + 1$ nodes of type a_0. Hence, by Claim 6.22, we conclude that $w'' \preceq_w w_i$, where $i \neq 1$. Since the first n_b children of v'' of type b have pairwise distinct values in attribute $@id$, for every $b \in alph(w)$, we conclude that $\#_b(w'') \geq n_b = \min\{\#_b(w), \#_b(w_1)\}$ for every $b \in alph(w)$. Thus, given that $w'' \preceq_w w_i$, we have that for every $c \in alph(w)$ such that $\#_c(w_i) < \#_c(w)$, it is the case that $\#_c(w_i) \geq \#_c(w'') \geq n_c = \#_c(w_1)$. Hence, given that $w_1 \not\preceq_w w_i$, there exists $b \in alph(w_i) \setminus alph(w)$ such that $b \notin alph(w_1)$. But $alph(w_i) \setminus alph(w) \subseteq$

$alph(w'') \setminus alph(w)$ and, thus, $b \in alph(w'')$. Therefore, there exists a children of v'' of type $b \in X$ having an attribute $@f$ and, hence, there exists a node (of type a_0) having i as value of attribute $@e$ and having a sibling (of type b) with an attribute $@f$, which contradicts the fact that 1 was assigned to $\sigma(x)$.

Since σ is well defined and σ satisfies θ (by definition of σ), we conclude that θ is satisfiable. This concludes the proof of case (II).

(III) Finally, assume that (I) and (II) do not hold. Then

Claim 6.23.

(a) For every $i \in [1,n]$, there exists $a \in alph(w)$ such that $\#_a(w_i) < \#_a(w)$.

(b) For every $i \in [1,n]$ and every $a \in alph(w)$ such that $\#_a(w_i) < \#_a(w)$, we have that $\#_a(w_i) = 1$.

(c) For every $i, j \in [1,n]$, $i \neq j$, there exists $a \in alph(w)$ such that $\#_a(w_i) = 1 < \#_a(w)$ and $\#_a(w_i) < \#_a(w_j)$.

Proof. (a) On the contrary, assume that there exists $i \in [1,n]$ such that $w \preceq w_i$. Then, given that (I) does not hold, for every $j \in [1,n]$, $j \neq i$, there exists $a \in alph(w)$ such that $\#_a(w_j) < \#_a(w)$. But $n \geq 2$ and, thus, there exist distinct $i, j \in [1,n]$ and $a \in alph(w)$ such that (1) $\#_a(w_j) < \#_a(w)$ and (2) for every $b \in alph(w)$, we have that $\#_b(w_j) \leq \#_b(w_i)$ or $\#_b(w_i) \geq \#_b(w)$, which contradicts the fact that (II) does not hold.

(b) On the contrary, assume that there exists $i \in [1,n]$ and $a \in alph(w)$ such that $1 < \#_a(w_i) < \#_a(w)$. Since $c(r) \leq 1$, we have that $w_i \notin fixed_a(r)$ and, therefore, there exists $w' \in \pi(r)$ such that $w_i \preceq w'$ and $\#_a(w_i) < \#_a(w')$. Let s be a string such that $alph(s) = alph(w)$, $\#_a(s) = \#_a(w_i)+1$ and $\#_b(s) = \min\{\#_b(w), \#_b(w_i)\}$, for every $b \in alph(w) \setminus \{a\}$. Then $s \preceq w$ and $s \preceq w'$ and, therefore, there exists $s' \in min\_ext(s,r)$ such that $s \preceq s' \preceq w'$ and $s' \in rep(w,r)$. By Claim 6.22, there exists $j \in [1,n]$ such that $s \preceq_w w_j$. Notice that $j \neq i$ since $\#_a(w_i) < \#_a(w)$ and $\#_a(w_i) + 1 = \#_a(s)$. It is easy to see that for every $b \in alph(w)$, we have that $\#_b(w_i) \leq \#_b(w_j)$ or $\#_b(w_j) \geq \#_b(w)$. Thus, there exists distinct $i, j \in [1,n]$ and $a \in alph(w)$ such that (1) $\#_a(w_i) < \#_a(w)$ and (2) for every $b \in alph(w)$, we have that $\#_b(w_i) \leq \#_b(w_j)$ or $\#_b(w_j) \geq \#_b(w)$, which contradicts the fact that (II) does not hold.

(c) On the contrary, assume that there exist $i, j \in [1,n]$, $i \neq j$, such that for every $b \in alph(w)$ such that $\#_b(w_i) = 1 < \#_b(w)$, we have that $\#_b(w_i) \geq \#_b(w_j)$. Then, by (b), we conclude that for every $b \in alph(w)$ such that $\#_b(w_i) < \#_b(w)$, we have that $\#_b(w_i) \geq \#_b(w_j)$. But by (a) we know that there exists $a \in alph(w)$ such that $\#_a(w_j) < \#_a(w)$ and, therefore, there exist distinct $i, j \in [1,n]$ and $a \in alph(w)$ such that (1) $\#_a(w_j) < \#_a(w)$ and (2) for every $b \in alph(w)$, we have that $\#_b(w_j) \leq \#_b(w_i)$ or $\#_b(w_i) \geq \#_b(w)$, which contradicts the fact that (II) does not hold. □

For every $i \in [1,n]$, let $X(w_i)$ be the set of element types $\{b \in alph(w) \mid 1 = \#_b(w_i) < \#_b(w)\}$. By Claim 6.23 we know that $X(w_i) \neq \emptyset$, for every $i \in [1,n]$, and that $X(w_i) \not\subseteq X(w_j)$, for every $i, j \in [1,n]$, $i \neq j$. Let $a_0 \in X(w_1)$ such that

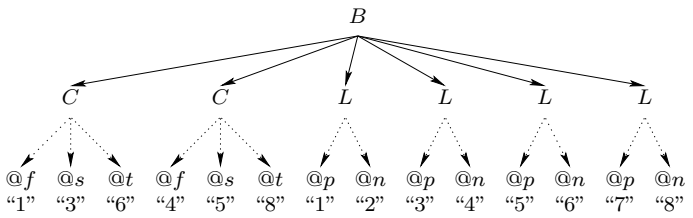


Fig. 14. XML tree T_θ, defined in case (III) of the proof of Lemma 6.21, representing propositional formula $\theta = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$.

$a_0 \notin X(w_2)$ and let

$$Y \;=\; \left(\bigcup_{i=2}^{n} X(w_i)\right) \setminus X(w_1).$$

Then $a_0 \notin Y$, $X(w_1) \cap Y = \emptyset$ and $X(w_i) \cap Y \neq \emptyset$, for every $i \in [2, n]$.

As in all the previous cases, to show that $\mathcal{C}$ is strongly coNP-complete, we define a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ and a Boolean $\mathcal{CTQ}$-query Q such that $D_{\mathbf{S}}$ is a simple DTD, $D_{\mathbf{T}}$ is a $\mathcal{C}$-DTD, $\Sigma_{\mathbf{ST}}$ is a set of fully-specified STDs and 3SAT can be reduced to the complement of Certain Answers(Q), that is, for every propositional formula θ in 3-CNF, there exists a PTIME constructible XML tree T_θ conforming to $D_{\mathbf{S}}$ such that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = \mathit{false}$. Simple DTD $D_{\mathbf{S}}$ is defined as follows. Let $E_{\mathbf{S}} = \{B, C, L\}$ be a set of element types and $A_{\mathbf{S}} = \{@f, @s, @t, @p, @n\}$ be a set of attributes. Then $D_{\mathbf{S}} = (P_{\mathbf{S}}, R_{\mathbf{S}}, B)$ is a DTD over $(E_{\mathbf{S}}, A_{\mathbf{S}})$, where $P_{\mathbf{S}}$ is defined as:

$$\begin{aligned} P_{\mathbf{S}}(B) \;&=\; C^* L^*, \\ P_{\mathbf{S}}(\ell) \;&=\; \varepsilon, \quad \text{for every } \ell \in E_{\mathbf{S}} \setminus \{B\}. \end{aligned}$$

and $R_{\mathbf{S}}$ is defined as:

$$R_{\mathbf{S}}(B) \;=\; \emptyset, \qquad R_{\mathbf{S}}(C) \;=\; \{@f, @s, @t\}, \qquad R_{\mathbf{S}}(L) \;=\; \{@p, @n\}.$$

XML trees conforming to $D_{\mathbf{S}}$ are used to represent propositional formulae. Let θ be 3-CNF formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$. To construct tree T_θ, first we assign a distinct natural number to each literal, say

$$\begin{aligned} x_1 &\mapsto 1, & x_2 &\mapsto 3, & x_3 &\mapsto 5, & x_4 &\mapsto 7, \\ \neg x_1 &\mapsto 2, & \neg x_2 &\mapsto 4, & \neg x_3 &\mapsto 6, & \neg x_4 &\mapsto 8. \end{aligned}$$

Then we represent each clause of θ as a node of type C, being the values of attributes $@f$, $@s$, $@t$ the first, second and third literal of that clause, respectively. For each propositional variable x in θ, we use the attributes $@p$, $@n$ of a node of type L to store the values assigned to x and $\neg x$, respectively. Tree T_θ for $\theta = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$ is shown in Figure 14.

$\mathcal{C}$-DTD $D_{\mathbf{T}}$ is defined as follows. Let $E_{\mathbf{T}} = \{B, G, C, N, F\} \cup \mathit{alph}(r)$ be a set of element types such that $\{B, G, C, N, F\} \cap \mathit{alph}(r) = \emptyset$, and let $A_{\mathbf{T}} = \{@f, @s, @t, @e\}$ be a set of attributes. Then $D_{\mathbf{T}} = (P_{\mathbf{T}}, R_{\mathbf{T}}, B)$ is a DTD over $(E_{\mathbf{T}}, A_{\mathbf{T}})$, where $P_{\mathbf{T}}$ is defined as:

$$\begin{array}{lll} P_{\mathbf{T}}(B) = G^*C^*, & P_{\mathbf{T}}(G) = r, & P_{\mathbf{T}}(\ell) = \varepsilon, \text{ for every } \ell \in \mathit{alph}(r) \setminus (Y \cup \{a_0\}), \\ P_{\mathbf{T}}(C) = \varepsilon, & P_{\mathbf{T}}(F) = \varepsilon, & P_{\mathbf{T}}(\ell) = N^*F^*, \text{ for every } \ell \in Y \cup \{a_0\}, \\ P_{\mathbf{T}}(N) = \varepsilon. & & \end{array}$$

and $R_{\mathbf{T}}$ is defined as:

$$\begin{array}{lll} R_{\mathbf{T}}(B) = \emptyset, & R_{\mathbf{T}}(G) = \emptyset, & R_{\mathbf{T}}(\ell) = \emptyset, \text{ for every } \ell \in \mathit{alph}(r) \\ R_{\mathbf{T}}(C) = \{@f, @s, @t\}, & R_{\mathbf{T}}(N) = \{@e\}, & R_{\mathbf{T}}(F) = \{@f\}. \end{array}$$

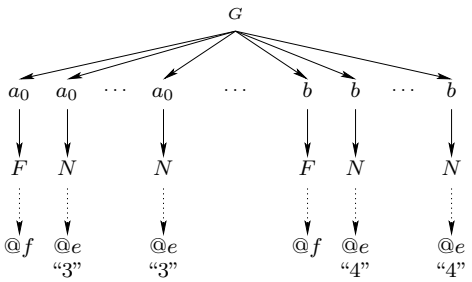
Finally, set $\Sigma_{\mathbf{ST}}$ of fully-specified STDs is defined as follows. The first rule of $\Sigma_{\mathbf{ST}}$ is defined as:

$$B[C(@f = x, @s = y, @t = z)] \;\; :\!- \;\; B[C(@f = x, @s = y, @t = z)].$$

This rule says that every node of type C in a source tree T must appear in every solution for T. The second rule of $\Sigma_{\mathbf{ST}}$ is defined as (we use $\bigwedge$ to denote a sequence of formulae separated by commas, as in cases (I) and (II)):

$$B[G[a_0[F], \bigwedge_{i=2}^{\#_{a_0}(w)} a_0[N(@e = x)], \bigwedge_{b \in Y} \Big(b[F], \bigwedge_{i=2}^{\#_b(w)} b[N(@e = y)] \Big), \bigwedge_{c \in \mathit{alph}(w) \setminus (Y \cup \{a_0\})} \bigwedge_{i=1}^{\#_c(w)} c]] \;\; :\!- \;\; B[L(@p = x, @n = y)].$$

To explain the meaning of this rule, we considered again tree T_θ shown in Figure 14. Assuming that $b \in Y$, the previous rule says that for each pair of complementary literals, say $(3, 4)$, and for each solution T' for T_θ, there should exist a node v of type G in T' having the following descendants:



In this figure, v has $\#_{a_0}(w)$ children of type a_0. The first of these children has only one child (of type F). Let $i \in [2, \#_{a_0}(w)]$. Then the i-th child of v, from left to right, of type a_0 has only one child (of type N) having 3 as value of attribute $@e$. Moreover, v has $\#_b(w)$ children of type b, for every $b \in Y$. The first of these children has only one child (of type F). Let $i \in [2, \#_b(w)]$. Then the i-th child of v, from left to right, of type b has only one child (of type N) having 4 as value of attribute $@e$. Finally, for every $c \in \mathit{alph}(w) \setminus (Y \cup \{a_0\})$, we have that v has exactly $\#_c(w)$ children of type c.

Since $w \notin \pi(r)$, to construct a solution T' for T_θ we need to replace w by a string $w' \in \pi(r)$ such that $\mathit{alph}(w) \subseteq \mathit{alph}(w')$. By Claim 6.22, we know that there

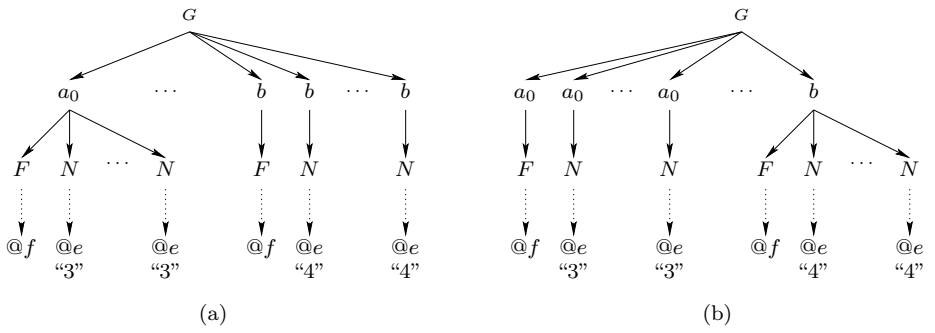


Fig. 15. Possible ways of repairing string w in case (III) of the proof of Lemma 6.21.

exists $i \in [1, n]$ such that $w' \preceq_w w_i$. First assume that $i = 1$. Then, given that $\#_{a_0}(w_1) = 1 < \#_{a_0}(w)$, we have that $\#_{a_0}(w') = 1$ and, hence, to construct w' from w we have to group all a_0-nodes together into a single node, generating the subtree shown in Figure 15 (a). Second, assume that $i \neq 1$. Then, there exists $b \in Y$ such that $\#_b(w_i) = 1 < \#_b(w)$ and, thus, $\#_b(w') = 1$. In this case, to construct w' from w we have to group all b-nodes together into a single node, generating the subtree shown in Figure 15 (b).

We note that when choosing whether to replace w by either a string contained in w_1 or a string contained in w_i ($i \neq 1$, $i \in [1, n]$), we are actually choosing the truth values for x_2 and $\neg x_2$. As in case (II.2), we say that a literal j has been assigned value 0 if j is the value of attribute $@e$ of a node having a sibling with an attribute $@f$. For example, in the solution for T_θ shown in Figure 15 (a), we have chosen value 0 for x_2 since 3 is the value of attribute $@e$ of a node (of type N) having a sibling (of type F) with an attribute $@f$. On the other hand, in the solution for T_θ shown in Figure 15 (b), we have chosen value 0 for $\neg x_2$ since 4 is the value of attribute $@e$ of a node (of type N) having a sibling (of type F) with an attribute $@f$. We will use this property to prove that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = \mathit{false}$. It is worth mentioning that it is possible to choose value 0 for both x_2 and $\neg x_2$. After defining query Q, we will show that this alternative does not cause any problems.

Boolean $\mathcal{CTQ}$-query Q is defined exactly as in case (II.2):

$$\exists x \exists y \exists z \exists u_1 \exists u_2 \exists u_3 \; (C(@f = x, @s = y, @t = z) \wedge \\ \_[\_(@f = u_1), \_(@e = x)] \; \wedge \; \_[\_(@f = u_2), \_(@e = y)] \wedge \\ \_[\_(@f = u_3), \_(@e = z)]).$$

Intuitively, query Q says that there exists a node v of type C such that each "literal" of v is assigned value 0, that is, the values i_1, i_2, i_3 of the attributes $@f$, $@s$, $@t$ of v, respectively, are such that for every $i \in \{i_1, i_2, i_3\}$, there exists a node v' having i as value of attribute $@e$ and having a sibling with an attribute $@f$.

Now we prove that for every 3-CNF propositional formula θ, we have that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = \mathit{false}$, where T_θ is constructed from θ in PTIME as shown above.

($\Rightarrow$) Assume that θ is satisfiable and let σ be a truth assignment satisfying θ. Define a solution T' for T_θ as follows. The structure of the C-nodes of T' is copied from T_θ. For every propositional variables x in θ, we define a G-node v of T' as follows. If $\sigma(x) = 0$, then the string of types of the children of v is w_1. In particular, v has only one child of type a_0, which has as children one node of type F and $(\#_{a_0}(w)-1)$ nodes of type N, each of them having in attribute $@e$ the value assigned to x in T_θ. For every $b \in Y$, we have that v has $\#_b(w_1)$ children of type b ($\#_b(w_1) > 1$ by definition of Y). The first child of v of type b has as children a node of type F. Each of the remaining $(\#_b(w_1) - 1)$ b-nodes has as children only one node (of type N) having in attribute $@e$ the value assigned to $\neg x$ in T_θ. Finally, each node of type F has an attribute $@f$ with value a fresh null. We note that in this case we have assigned value 0 to $\sigma(x)$, since the value assigned to x in T_θ is the value of attribute $@e$ of a node (of type N) having a sibling (of type F) with an attribute $@f$. Observe that these nodes are descendants of a node of type a_0. We also note that in this case we have not assigned value 0 to $\sigma(\neg x)$. Now, let us consider $\sigma(x) = 1$. In this case, the string of types of the children of v is w_2. In particular, v has only one child of type b, for some $b \in Y$, which has as children one node of type F and $(\#_b(w) - 1)$ nodes of type N, each of them having in attribute $@e$ the value assigned to $\neg x$ in T_θ. Furthermore, v has $\#_{a_0}(w_2)$ children of type a_0 ($\#_{a_0}(w_2) > 1$ since $a_0 \notin X(w_2)$). The first child of v of type a_0 has as children a node of type F. Each of the remaining $(\#_{a_0}(w_2) - 1)$ a_0-nodes has as children only one node (of type N) having in attribute $@e$ the value assigned to x in T_θ. Finally, each node of type F has an attribute $@f$ with value a fresh null. We note that in this case we have assigned value 0 to $\sigma(\neg x)$, since the value assigned to $\neg x$ in T_θ is the value of attribute $@e$ of a node (of type N) having a sibling (of type F) with an attribute $@f$. Observe that these nodes are descendants of a node of type $b \in Y$. We also note that in this case we have not assigned value 0 to $\sigma(x)$.

It is straightforward to prove that T' conforms to $D_{\mathbf{T}}$ and satisfies $\Sigma_{\mathbf{ST}}$. Furthermore, $T' \not\models Q$ since σ satisfies θ and, therefore, at least one literal per clause C is not assigned value 0. We conclude that $\underline{certain}(Q, T_\theta) = \mathit{false}$ since T' is a solution for T_θ.

($\Leftarrow$) Assume that $\underline{certain}(Q, T_\theta) = \mathit{false}$ and let T' be a solution for T_θ such that $T' \not\models Q$. We define a truth assignment for the propositional variables of θ as follows. For every clause in θ, find a C-node v of T' such that the values i_1, i_2, i_3 of attributes $@f$, $@s$, $@t$ of v are the values assigned in T_θ to the literals of that clause. Since $T' \not\models Q$, there exists $i \in \{i_1, i_2, i_3\}$ such that every node v' having i as value of attribute $@e$ does not have a sibling with an attribute $@f$. If i corresponds to a positive literal x, then define $\sigma(x)$ as 1. If i corresponds to a negative literal $\neg x$, then define $\sigma(x)$ as 0. We will show that σ is well defined. On the contrary, assume that there exists a propositional variable x such that 1 was assigned to $\sigma(x)$ and $\sigma(\neg x)$. Let i, j be the values assigned in T_θ to x and $\neg x$. Then every node v' having j as value of attribute $@e$ does not have a sibling with an attribute $@f$. Let v'' be a G-node of T' that satisfies the left hand side of the second rule of $\Sigma_{\mathbf{ST}}$ instantiated on the following values from T_θ:

$$B[L(@p = i, @n = j)],$$

and let $w'' = \lambda_{T'}(\mathit{children}(v''))$. Since every node v' having j as value of attribute $@e$ does not have a sibling with an attribute $@f$, for every $b \in Y$ we have that either $\#_b(w'') = 0$ or $\#_b(w'') \geq 2$. Thus, by Claims 6.22 and definition of Y, we have that $w'' \preceq_w w_1$ and, hence, $\#_{a_0}(w'') = 1$. We conclude that the only child of v'' of type a_0 must have a child of type F and a child of type N having i as value of attribute $@e$ and, hence, there exists a node having i as value of attribute $@e$ and having a sibling with an attribute $@f$, which contradicts the fact that 1 was assigned to $\sigma(x)$.

Since σ is well defined and σ satisfies θ (by definition of σ), we conclude that θ is satisfiable. This concludes the proof of case (III) and the proof of Lemma 6.21. □

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 [Popa et al. 2002]. 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. We would like to remove the distinct-variable restriction on source patterns used in our investigation of the consistency problem, and analyze the resulting settings for decidability/complexity issues. The standard notions of local-as-view and global-as-view from data integration [Lenzerini 2002] have been adapted in relational data exchange [Fagin et al. 2005; Fagin et al. 2005] 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. One very specific issue one needs to address is the complexity of checking univocality of regular expressions (which guarantees tractable query answering). We showed that the notion is decidable by reduction to Presburger arithmetic, but this does not give us good complexity bounds.

Acknowledgments

We are very grateful to Ronald Fagin, Phokion Kolaitis, and Lucian Popa for many helpful discussions during the early stages of this project, to Pablo Barceló and Wenfei Fan for their comments on the first draft, and to the referees for their

comments. Part of this work was done while both authors were at the University of Toronto, supported by a grant from NSERC, and while M. Arenas was visiting IBM Almaden, supported by an IBM graduate fellowship. In addition, M. Arenas is supported by FONDECYT grants 1050701 and 1070732 and grant P04-067-F from the Millennium Nucleus Center for Web Research, and L. Libkin is supported by the EC Marie Curie Excellence grant MEXC-CT-2005-024502 and EPSRC grant E005039.

REFERENCES

Abiteboul, S. and Duschka, O. M. 1998. Complexity of answering queries using materialized views. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, NY, 254–263.

Abiteboul, S., Kanellakis, P. C., and Grahne, G. 1991. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci. 78,* 1, 158–187.

Abiteboul, S., Segoufin, L., and Vianu, V. 2001. Representing and querying XML with incomplete information. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, NY, 150–161.

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

Amer-Yahia, S. and Kotidis, Y. 2004. Web-services architecture for efficient XML data exchange. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. IEEE Computer Society Press, Los Alamitos, CA, 523–534.

Arenas, M., Barceló, P., Fagin, R., and Libkin, L. 2004. Locally consistent transformations and query answering in data exchange. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, NY, 229–240.

Arenas, M. and Libkin, L. 2005. XML data exchange: consistency and query answering. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, NY, 13–24.

Benedikt, M., Fan, W., and Kuper, G. M. 2003. Structural properties of XPath fragments. In *Proceedings of the International Conference on Database Theory (ICDT)*. Springer, Heidelberg, 79–95.

Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. 2007. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`. release October, 12th 2007.

Deutsch, A. and Tannen, V. 2001. Containment and integrity constraints for XPath. In *Proceedings of the International Workshop on Knowledge Representation meets Databases (KRDB)*. CEUR-WS.org.

Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. 2003. Data exchange: Semantics and query answering. In *Proceedings of the International Conference on Database Theory (ICDT)*. Springer, Heidelberg, 207–224.

Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. 2005. Data exchange: semantics and query answering. *Theor. Comput. Sci. 336,* 1, 89–124.

Fagin, R., Kolaitis, P. G., and Popa, L. 2003. Data exchange: getting to the core. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, NY, 90–101.

Fagin, R., Kolaitis, P. G., and Popa, L. 2005. Data exchange: getting to the core. *ACM Trans. Database Syst. 30,* 1, 174–210.

Fagin, R., Kolaitis, P. G., Popa, L., and Tan, W. C. 2005. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst. 30,* 4, 994–1055.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

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

Imielinski, T. and Lipski, W. 1984. Incomplete information in relational databases. *J. ACM 31,* 4, 761–791.

Kozen, D. 2002. On two letters versus three. In *Proceedings of the Fixed Points in Computer Science (FICS)*. University of Aarhus, 44–50.

Krishnamurthy, R., Kaushik, R., and Naughton, J. F. 2003. XML-SQL query translation literature: The state of the art and open problems. In *Proceedings of the International XML Database Symposium (XSym)*. Springer, Heidelberg, 1–18.

Lakshmanan, L. V. S., Ramesh, G., Wang, H., and Zhao, Z. 2004. On testing satisfiability of tree pattern queries. In *Proceedings of the International Conference Very Large Data Bases (VLDB)*. Morgan Kaufmann, San Mateo, CA, 120–131.

Lenstra, H. W. 1983. Integer programming in a fixed number of variables. *Math. Oper. Res. 8,* 4, 538–548.

Lenzerini, M. 2002. Data integration: A theoretical perspective. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, NY, 233–246.

Miller, R. J., Hernández, M. A., Haas, L. M., Yan, L.-L., Ho, C. T. H., Fagin, R., and Popa, L. 2001. The clio project: Managing heterogeneity. *SIGMOD Record 30,* 1, 78–83.

Neven, F. 2002. Automata, logic, and XML. In *Proceedings of the Conference for Computer Science Logic (CSL)*. Springer, Heidelberg, 2–26.

Neven, F. and Schwentick, T. 2003. XPath containment in the presence of disjunction, DTDs, and variables. In *Proceedings of the International Conference on Database Theory (ICDT)*. Springer, Heidelberg, 312–326.

Popa, L., Velegrakis, Y., Miller, R. J., Hernández, M. A., and Fagin, R. 2002. Translating web data. In *Proceedings of the International Conference Very Large Data Bases (VLDB)*. Morgan Kaufmann, San Mateo, CA, 598–609.

Seidl, H. 1990. Deciding equivalence of finite tree automata. *SIAM J. Comput. 19,* 3, 424–437.

Shu, N. C., Housel, B. C., Taylor, R. W., Ghosh, S. P., and Lum, V. Y. 1977. Express: A data extraction, processing, amd restructuring system. *ACM Trans. Database Syst. 2,* 2, 134–174.

Vianu, V. 2001. A web odyssey: From Codd to XML. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, NY, 1–15.

Wood, P. T. 2003. Containment for XPath fragments under DTD constraints. In *Proceedings of the International Conference on Database Theory (ICDT)*. Springer, Heidelberg, 297–311.

Yu, C. and Popa, L. 2004. Constraint-based xml query rewriting for data integration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM Press, New York, NY, 371–382.

A. STRING AND TREE AUTOMATA

In this section, we recall some basic facts about ranked and unranked tree automata. We write NFA and DFA for non-deterministic and deterministic finite automata. If $\mathcal{A}$ is an automaton, then $L(\mathcal{A})$ is the language accepted by it. Recall that given a regular expression r, constructing an NFA $\mathcal{A}_r$ with $L(\mathcal{A}_r) = L(r)$ can be done in polynomial time.

A nondeterministic (ranked) finite tree automaton (NFTA) on binary node-labeled trees over alphabet Γ is defined as $\mathcal{A} = \langle Q, q_0, \delta, F\rangle$ where Q is the set of states, $q_0 \in Q, F \subseteq Q$ and $\delta : \Gamma \times Q \times Q \to 2^Q$ is the transition function. Given a binary tree T, a run of $\mathcal{A}$ on T is a function $run_{\mathcal{A}} : T \to Q$ that assigns states to nodes. For a leaf s labeled a, we require that $run_{\mathcal{A}}(s) \in \delta(a, q_0, q_0)$, and for a node s labeled a with two children s_1 and s_2 we require that $run_{\mathcal{A}}(s) \in \delta(a, run_{\mathcal{A}}(s_1), run_{\mathcal{A}}(s_2))$. A tree T is accepted if there is a run $run_{\mathcal{A}}$ such that $run_{\mathcal{A}}(root) \in F$. Given an NFTA $\mathcal{A}$, testing whether $L(\mathcal{A}) = \emptyset$ can be done in time linear in the size of

$\mathcal{A}$ [Comon et al. 2007], and testing whether there is a tree not accepted by $\mathcal{A}$ is EXPTIME-complete [Seidl 1990].

An unranked nondeterministic finite tree automaton (UNFTA) on ordered unranked node-labeled trees over alphabet Γ is defined as $\mathcal{A} = \langle Q, \delta, F\rangle$ where Q is the set of states, $F \subseteq Q$ and $\delta : Q \times \Gamma \to 2^{Q^*}$ is the transition function such that $\delta(q, a)$ is a regular language for every $s \in Q$ and $a \in \Gamma$. Given an ordered unranked tree T, a run of $\mathcal{A}$ on T is again a function $run_{\mathcal{A}} : T \to Q$ that assigns states to nodes. If s is a node whose children are $s_1, \ldots, s_n$ ordered by the sibling relation as $s_1 <_{\text{sib}} \ldots <_{\text{sib}} s_n$, then the word $run_{\mathcal{A}}(s_1) \ldots run_{\mathcal{A}}(s_n)$ over Q must be in $\delta(run_{\mathcal{A}}(s), a)$. In particular, if s is a leaf, then a run can assign a state q to it iff $\varepsilon \in L(\delta(q, a))$. A tree T is accepted if there is a run $run_{\mathcal{A}}$ such that $run_{\mathcal{A}}(root) \in F$. An automaton is deterministic if every tree admits only one run.

The standard representation of UNFTAs uses NFAs for transitions, that is, δ maps pairs state-letter to NFAs over Q. It is known that testing nonemptiness is again polynomial-time [Neven 2002]. If an automaton is deterministic and furthermore all transitions are represented by DFAs, then we refer to UFTA(DFA).

DTDs can naturally be represented by tree automata. We shall consider DTDs without attributes. Such a DTD D over a set E of element types is represented by an automaton $\mathcal{A}_D$ in which the set of states is E, and $\delta(\ell, \ell)$ is defined to be an automaton for $P(\ell)$, and $\delta(\ell, \ell') = \emptyset$ otherwise, and $F = \{\underline{r}\}$.

B. PROOFS

B.1 Proof of Proposition 4.4

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 start with a). By Claim 4.2, it suffices to consider STDs in which all attribute formulae are of the form $\ell \in El$. Furthermore, every nonrecursive DTD D that does not use the Kleene star can be in polynomial time transformed into a DTD D' in which all productions are of the form $\ell \to \ell'\ell''$, or $\ell \to \ell'|\ell''$, or $\ell \to \varepsilon$. This is done by repeatedly changing each rule $\ell \to r_1 r_2$ into $\ell \to \ell_1\ell_2, \ell_1 \to r_1, \ell_2 \to r_2$, and each rule $\ell \to r_1|r_2$ into $\ell \to \ell_1|\ell_2, \ell_1 \to r_1, \ell_2 \to r_2$, where ℓ_1 and ℓ_2 are fresh symbols from El. This also establishes an embedding $h : \text{SAT}(D) \to \text{SAT}(D')$ such that for every tree-pattern formula given by the grammar

$$\varphi \;:=\; \ell,\ \ell \in E \;\mid\; \ell[\varphi] \;\mid\; //\varphi,$$

one can find a formula φ' from the same class such that $T \models \varphi$ iff $h(T) \models \varphi'$, for every $T \in \text{SAT}(D)$. (This formula simply leaves // in place, and replaces $\ell[\varphi]$ by an explicit chain of new element types introduced in the translation). In view of this, we can assume, without loss of generality, that in all DTDs all the productions are of the form $\ell \to \ell'\ell''$, or $\ell \to \ell'|\ell''$, or $\ell \to \varepsilon$.

Now we show membership in PSPACE. Since by the assumption the target DTD $D_{\mathbf{T}}$ is fixed and is not recursive and does not use the Kleene star, there are only finitely many trees that conform to it (if we disregard attribute values), that can

be explicitly listed. Let $T_1, \ldots, T_m$ be the listing of those trees. Then for each STD $\psi :- \varphi$ and each T_i we can verify if $T_i \models \psi$. Furthermore, since each tree-pattern formula can be translated into an FO formula in the vocabulary that contains both child $<_{\text{child}}$ and descendant $<^*_{\text{child}}$ relation, checking whether $T_i \models \psi$ can be done in PSPACE (since the combined complexity of FO is PSPACE).

Let Φ_i be the set of all φ's such that $T_i \not\models \psi$ where $\psi :- \varphi$ is an STD in $\Sigma_{\mathbf{ST}}$. Then the setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ is consistent iff there is a tree $T \models D_{\mathbf{S}}$ and $i \leq n$ such that $T \not\models \varphi$ for every $\varphi \in \Phi_i$ (in this case T_i is a solution for T). Thus, to show PSPACE-membership, it suffices to show that for a given set Φ of right-hand sides of STDs from $\Sigma_{\mathbf{ST}}$, one can check, in PSPACE, whether there exists a tree T that conforms to $D_{\mathbf{S}}$ and falsifies every formula in Φ. This is proved by providing an alternating PTIME algorithm; since alternating PTIME equals PSPACE, the result follows. Given the restriction on STDs, each φ simply checks the existence of a path with a fixed set of labels $\ell_1, \ldots, \ell_k$, indicating for each consecutive ones whether they are in the child or descendant relation. Hence the alternating algorithm makes an existential step for each $\ell \to \ell'|\ell''$ rule in the DTD, and a universal step for each $\ell \to \ell'\ell''$ rule. It takes polynomial time to keep each path in memory and verify whether it falsifies every $\varphi \in \Phi$. Since $D_{\mathbf{S}}$ is non-recursive, this alternating algorithm runs in polynomial time, thus proving PSPACE-membership.

For PSPACE-hardness, we show reduction from QBF. Suppose we are given a quantified Boolean formula

$$\alpha \;=\; Q_1 x_1 \ldots Q_m x_m \;\big(C_1 \wedge \ldots \wedge C_n\big), \tag{5}$$

where each Q_i is either $\forall$ or $\exists$, and each clause C_i is a disjunction $(x_{j_i}^{+,-} \vee x_{k_i}^{+,-} \vee x_{\ell_i}^{+,-})$. Here $x^{+,-}$ indicates that variables could be negated: x^+ refers to x and x^- to $\neg x$. Since the restriction of QBF to 3-CNF quantifier-free formulae is known to be PSPACE-complete, we can assume this restriction on the shape of clauses.

To represent formula (5) we use the following data exchange setting. The source DTD $D_{\mathbf{S}}$ has element types $\{\underline{r},\ x_1^+, x_1^-, \ldots, x_m^+, x_m^-\}$, assuming that formula (5) mentions propositional variables x_1, …, x_m. The rules of $D_{\mathbf{S}}$ are as follows:

$$\begin{array}{lcll}
\underline{r} & \to & x_1^+ x_1^- & \text{if } Q_1 = \forall \\
\underline{r} & \to & x_1^+ | x_1^- & \text{if } Q_1 = \exists \\
x_i & \to & x_{i+1}^+ x_{i+1}^- & \text{if } Q_{i+1} = \forall, \text{ for } 1 \leq i < m \\
x_i & \to & x_{i+1}^+ | x_{i+1}^- & \text{if } Q_{i+1} = \exists, \text{ for } 1 \leq i < m
\end{array}
\qquad
\begin{array}{lcl}
x_m^+ & \to & \varepsilon \\
x_m^- & \to & \varepsilon
\end{array}$$

Let f be an element type that does not occur in the target DTD, whose root is denoted by $\underline{r}'$. For each clause C, consider the assignment that invalidates it. For example, let $C = (x_i \vee \neg x_j \vee x_k)$ with $i < j < k$. Then the invalidating assignment is $x_i = 0, x_j = 1, x_k = 0$. In this case we put the following STD into $\Sigma_{\mathbf{ST}}$:

$$\underline{r}'[f] \;\; :- \;\; \underline{r}[//x_i^-[//x_j^+[//x_k^-]]],$$

assuming $i - 1, j - i, k - j > 1$. In other words, if C is invalidated, we attempt to put an node in the target tree that is inconsistent with the target DTD. When some of the indices are consecutive, we have to omit the descendant //: for example, if $i = 1$, $j > 2$ and $k = j + 1$, we use

$$\underline{r}'[f] \;\; :- \;\; \underline{r}[x_i^-[//x_j^+[x_k^-]]],$$

since in this case x_i^- has to be witnessed at a child of the root, and not a descendant of a child of a root.

We now let $\Sigma_{\mathbf{ST}}$ contain all such STDs for all the clauses used in the formula. Then a straightforward argument shows that α is true iff there is a tree $T \models D_{\mathbf{S}}$ such that no right-hand side of an STD is satisfied, which in turn is equivalent to the consistency of the setting.

We continue with b). This proof is similar to the previous case. Again we assume that formulae in STDs do not have free variables. We shall need the following observation. If we have a path-pattern formula φ, a tree T, and a node v of T, one can verify if v witnesses φ in T in time $O(\|T\| \cdot \|\varphi\|)$. Indeed, we inductively find sets of nodes in which subformulae of φ are true. First, each formula ℓ is true in nodes labeled ℓ. To find nodes in which $\ell[\varphi']$ holds we find ℓ-nodes which have a child for which by induction we already know that φ' holds. To find nodes in which $//\varphi'$ holds, we depth-first traverse the tree and for each node in which φ' holds, mark all its ancestors with $//\varphi'$ as we go backwards. Thus, for each subformula of φ we need time linear in the size of T, and hence the algorithm runs in time $O(\|T\| \cdot \|\varphi\|)$.

We now show membership in NP. Since $D_{\mathbf{T}}$ is fixed and does not use the Kleene star, there may be only a fixed number of trees that conform to it. Let $T_1, \ldots, T_n$ enumerate them. As before, let $\Phi_i = \{\varphi \mid \psi :\!- \varphi \in \Sigma_{\mathbf{ST}} \text{ and } T_i \not\models \psi\}$. Then it suffices to check in NP if for some $i \leq n$, there exists a tree T such that $T \models D_{\mathbf{S}}$ and $T \not\models \varphi$ for all $\varphi \in \Phi_i$. If that is the case, the pair $\langle T, T_i \rangle$ witnesses consistency; if there is no such tree for all i, the setting is inconsistent. For the latter, we guess a tree T simply by making a choice for each of the rules $\ell \to \ell_1 | \ldots | \ell_m$. Then, by the earlier observation, we can verify whether $T \not\models \varphi$ in polynomial time in the sizes of T (which is linear in the size of $D_{\mathbf{S}}$) and φ. This proves membership in NP.

For hardness, we simply apply the proof for QBF in the case when all quantifiers are existential. Then every rule is of the form $\ell \to \ell' | \ell''$ or $\ell \to \varepsilon$, and the QBF reduction becomes a 3SAT reduction. This proves NP-hardness, and the proposition.

B.2 Proof of Theorem 5.11

We consider here classes STD($\underline{r}, //$) and STD($\underline{r}, \_$). The proof for the class STD($\_, //$) was already given in Section 5.3.

Case of STD($\underline{r}, //$): We define a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ and a Boolean $\mathcal{CTQ}$-query Q such that both $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$ are simple DTDs, $\Sigma_{\mathbf{ST}}$ is a set of source-to-target dependencies in STD($\underline{r}, //$) and 3SAT can be reduced to the complement of Certain-Answers(Q), that is, for every propositional formula θ in 3-CNF, there exists a PTIME constructible XML tree T_θ conforming to $D_{\mathbf{S}}$ such that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = \mathit{false}$. Simple DTD $D_{\mathbf{S}}$ is defined exactly as in the proof for STD($\_, //$) (see Section 5.3). Simple DTD $D_{\mathbf{T}}$ is defined as follows. Let $E_{\mathbf{T}} = \{K, L, B_1, B_2, G_1, G_2, J, H_1, H_2, C_1, C_2, C_3\}$ be a set of element types and $A_{\mathbf{T}} = \{@\ell, @p, @n, @m\}$ a set of attributes. Then $D_{\mathbf{T}} = (P_{\mathbf{T}}, R_{\mathbf{T}}, K)$ is a DTD over $(E_{\mathbf{T}}, A_{\mathbf{T}})$, where $P_{\mathbf{T}}$ is defined as:

$$
\begin{array}{llllllllll}
P_{\mathbf{T}}(K) & = & B_1^*L^*, & P_{\mathbf{T}}(B_1) & = & B_2^*, & P_{\mathbf{T}}(B_2) & = & G_1^*J^*C_1^*, \\
P_{\mathbf{T}}(G_1) & = & G_2^*J^*, & P_{\mathbf{T}}(G_2) & = & J^*, & P_{\mathbf{T}}(J) & = & H_1^*, \\
P_{\mathbf{T}}(H_1) & = & H_2^*, & P_{\mathbf{T}}(H_2) & = & \varepsilon, & P_{\mathbf{T}}(C_1) & = & C_2^*, \\
P_{\mathbf{T}}(C_2) & = & C_3^*, & P_{\mathbf{T}}(C_3) & = & \varepsilon, & P_{\mathbf{T}}(L) & = & \varepsilon.
\end{array}
$$

and $R_{\mathbf{T}}$ is defined as:

$$
\begin{array}{llllllllll}
R_{\mathbf{T}}(K) & = & \emptyset, & R_{\mathbf{T}}(B_1) & = & \emptyset, & R_{\mathbf{T}}(B_2) & = & \emptyset, \\
R_{\mathbf{T}}(G_1) & = & \{@m\}, & R_{\mathbf{T}}(G_2) & = & \emptyset, & R_{\mathbf{T}}(J) & = & \{@m\}, \\
R_{\mathbf{T}}(H_1) & = & \emptyset. & R_{\mathbf{T}}(H_2) & = & \emptyset. & R_{\mathbf{T}}(C_1) & = & \{@\ell\}, \\
R_{\mathbf{T}}(C_2) & = & \{@\ell\}, & R_{\mathbf{T}}(C_3) & = & \{@\ell\}, & R_{\mathbf{T}}(L) & = & \{@p, @n\}.
\end{array}
$$

Finally, $\Sigma_{\mathbf{ST}}$ is defined as follows. The first rule of $\Sigma_{\mathbf{ST}}$ is defined as:

$$K[L(@p = x, @n = y)] \ :- \ K[L(@p = x, @n = y)].$$

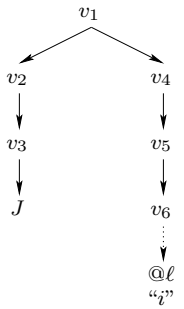
This rule says that every node of type L in a source tree T must appear in every solution for T. The second rule of $\Sigma_{\mathbf{ST}}$ is defined as:

$$K[B_1[B_2[\_(@m = u)[\_[\_]],\ C_1(@\ell = x)[C_2(@\ell = y)[C_3(@\ell = z)]]]]] \ :- \ K[C(@f = x, @s = y, @t = z)].$$

Notice that this rule is not fully-specified since it does not say whether the node having an attribute $@m$ is of type either G_1 or J. Also notice that in the left hand side of this rule we have a formula of the form $K[B_1[B_2[\varphi_1, \varphi_2]]]$ not using descendant //, where K is the type of the root.

The previous rule says that for every C-node v of a source tree T having i, j, k as values of attributes $@f$, $@s$, $@t$, and for every solution T' for T, T' must have as subtree at least one of the trees shown in Figure 16.

When constructing a solution for T_θ, we are actually constructing a truth assignment for θ. Let v be a C-node of T_θ with values i, j, k in attributes $@f$, $@s$, $@t$. To construct a solution T' for T we have to instantiate the second dependency of $\Sigma_{\mathbf{ST}}$ on values i, j and k, and then we have to choose among the trees shown in Figure 16 which one is going to be a subtree of T'. These alternatives represent three different ways of satisfying the clause stored in the children of v. We say that a literal i has been assigned value 1 if i appears in a subtree of the form:



That is, there exists a node v_1 that has at least two great-grandchildren, one of type J and another one having i in attribute $@\ell$. Thus, in Figure 16 (a), literal i has been assigned value 1 since there is a node (of type K) having as great-grandchildren

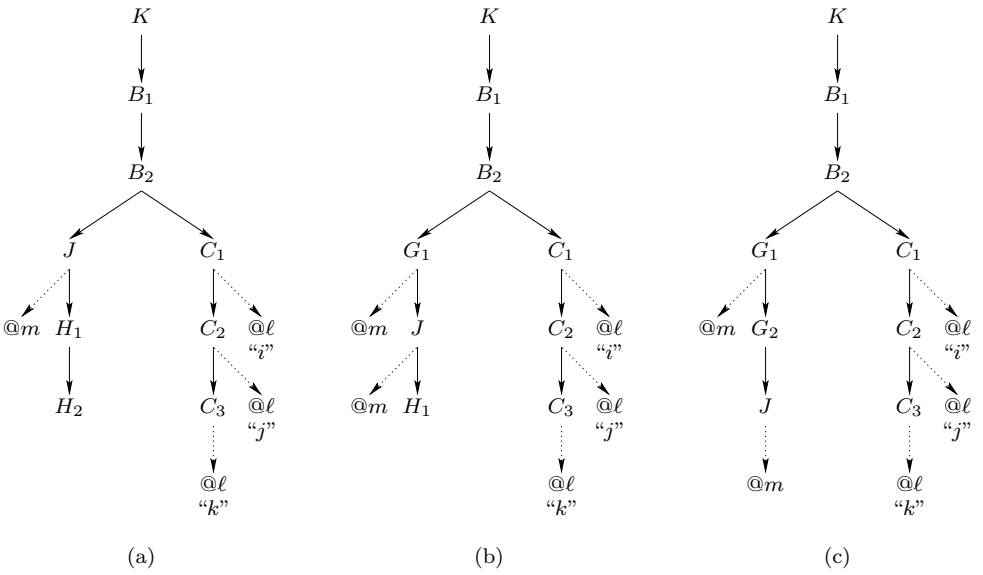


Fig. 16. Different alternatives for satisfying the second rule of $\Sigma_{\mathbf{ST}}$ in case STD($\underline{r}$, //) of the proof of Theorem 5.11.

a node of type J and a node (of type C_1) having i in attribute $@\ell$. On the other hand, in Figure 16 (b), literal j has been assigned value 1 since there is a node (of type B_1) having as great-grandchildren a node of type J and a node (of type C_2) having j in attribute $@\ell$, and in Figure 16 (c), literal k has been assigned value 1 since there is a node (of type B_2) having as great-grandchildren a node of type J and a node (of type C_3) having k in attribute $@\ell$. Notice that when constructing a truth assignment for θ, it is possible to choose value 1 for two complementary literals (when considering complementary literals in two distinct clauses). To take care of this problem we use the following Boolean $\mathcal{CTQ}$-query Q:

$$\exists x \exists y\,(L(@p = x, @n = y) \ \wedge\ \_[\_[\_[J]],\ \_[\_[\_(@\ell = x)]]] \ \wedge\ \_[\_[\_[J]],\ \_[\_[\_(@\ell = y)]]]).$$

Intuitively, query Q says that there exists two complementary literals x and y such that both x and y have been assigned value 1, that is, there exists a node having at least two great-grandchildren, one of type J and the other one having x in attribute $@\ell$, and the same holds for y.

Now we prove that for every 3-CNF propositional formula θ, we have that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = \mathit{false}$, where T_θ is constructed from θ in PTIME as in the proof for STD($\_$, //) (see Section 5.3).

($\Rightarrow$) Assume that θ is satisfiable and let σ be a truth assignment satisfying θ. Define a solution T' for T_θ as follows. The structure of the L-nodes of T' is copied from T_θ. For every C-node v in T_θ having i, j, k in attributes $@f$, $@s$, $@t$, if σ makes true the first literal of this clause, then T' has as subtree the tree shown in Figure 16 (a). If σ makes true the second literal of this clause, then T' has as subtree the tree shown in Figure 16 (b). Finally, if σ makes true the third literal of this clause, then T' has as subtree the tree shown in Figure 16 (c). It is straightforward

to prove that T' conforms to $D_{\mathbf{T}}$ and satisfies $\Sigma_{\mathbf{ST}}$. Furthermore, $T' \not\models Q$ since σ is well defined and, therefore, for every propositional variables x, either x or $\neg x$ is not assigned value 1. We conclude that $\underline{certain}(Q, T_\theta) = false$ since T' is a solution for T_θ.

($\Leftarrow$) Assume that $\underline{certain}(Q, T_\theta) = false$ and let T' be a solution for T_θ such that $T' \not\models Q$. We define a truth assignment for the propositional variables of θ as follows. For every clause in θ, find the values i, j, k assigned in T_θ (as values of attributes $@f$, $@s$ and $@t$) to the literals of that clause. Then T' has as subtree at least one of the trees shown in Figure 16. If T' has as subtree the tree shown in Figure 16 (a), then σ assigns value 1 to the first literal of the clause. If T' has as subtree the tree shown in Figure 16 (b), then σ assigns value 1 to the second literal of the clause. Finally, if T' has as subtree the tree shown in Figure 16 (c), then σ assigns value 1 to the third literal of the clause . Since $T' \not\models Q$, we have that σ is well defined. Thus θ is satisfiable since σ satisfies this formula by definition. This concludes the proof of case of STD($\underline{r}$, //).

Case of STD($\underline{r}$, _): We define a data exchange setting $(D_{\mathbf{S}}, D_{\mathbf{T}}, \Sigma_{\mathbf{ST}})$ and a Boolean $\mathcal{CTQ}$-query Q such that both $D_{\mathbf{S}}$ and $D_{\mathbf{T}}$ are simple DTDs, $\Sigma_{\mathbf{ST}}$ is a set of source-to-target dependencies in STD($\underline{r}$, _) and 3SAT can be reduced to the complement of Certain-Answers(Q), that is, for every propositional formula θ in 3-CNF, there exists a PTIME constructible XML tree T_θ conforming to $D_{\mathbf{S}}$ such that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = false$.

Simple DTD $D_{\mathbf{S}}$ is defined exactly as in the previous case. Simple DTD $D_{\mathbf{T}}$ is defined as follows. Let $E_{\mathbf{T}} = \{K,\ L,\ B_1,\ B_2,\ G_1,\ G_2,\ J,\ C_1,\ C_2,\ C_3\}$ be a set of element types and $A_{\mathbf{T}} = \{@\ell, @p, @n\}$ a set of attributes. Then $D_{\mathbf{T}} = (P_{\mathbf{T}}, R_{\mathbf{T}}, K)$ is a DTD over $(E_{\mathbf{T}}, A_{\mathbf{T}})$, where $P_{\mathbf{T}}$ is defined as:

$$\begin{array}{llllllllll}
P_{\mathbf{T}}(K) & = & B_1^* L^*, & P_{\mathbf{T}}(B_1) & = & B_2^*, & P_{\mathbf{T}}(B_2) & = & G_1^* J^* C_1^*, \\
P_{\mathbf{T}}(G_1) & = & G_2^* J^*, & P_{\mathbf{T}}(G_2) & = & J^*, & P_{\mathbf{T}}(J) & = & \varepsilon, \\
P_{\mathbf{T}}(C_1) & = & C_2^*, & P_{\mathbf{T}}(C_2) & = & C_3^*, & P_{\mathbf{T}}(C_3) & = & \varepsilon, \\
P_{\mathbf{T}}(L) & = & \varepsilon. & & & & & &
\end{array}$$

and $R_{\mathbf{T}}$ is defined as:

$$\begin{array}{llllllllll}
R_{\mathbf{T}}(K) & = & \emptyset, & R_{\mathbf{T}}(B_1) & = & \emptyset, & R_{\mathbf{T}}(B_2) & = & \emptyset, \\
R_{\mathbf{T}}(G_1) & = & \emptyset, & R_{\mathbf{T}}(G_2) & = & \emptyset, & R_{\mathbf{T}}(J) & = & \emptyset, \\
R_{\mathbf{T}}(C_1) & = & \{@\ell\}, & R_{\mathbf{T}}(C_2) & = & \{@\ell\}, & R_{\mathbf{T}}(C_3) & = & \{@\ell\}, \\
R_{\mathbf{T}}(L) & = & \{@p, @n\}. & & & & & &
\end{array}$$

Finally, $\Sigma_{\mathbf{ST}}$ is defined as follows. The first rule of $\Sigma_{\mathbf{ST}}$ is defined as:

$$K[L(@p = x, @n = y)] \ \text{:-} \ K[L(@p = x, @n = y)].$$

This rule says that every node of type L in a source tree T must appear in every solution for T. The second rule of $\Sigma_{\mathbf{ST}}$ is defined as:

$$K[B_1[B_2[//J,\ C_1(@\ell = x)[C_2(@\ell = y)[C_3(@\ell = z)]]]]] \ \text{:-} \ K[C(@f = x, @s = y, @t = z)].$$

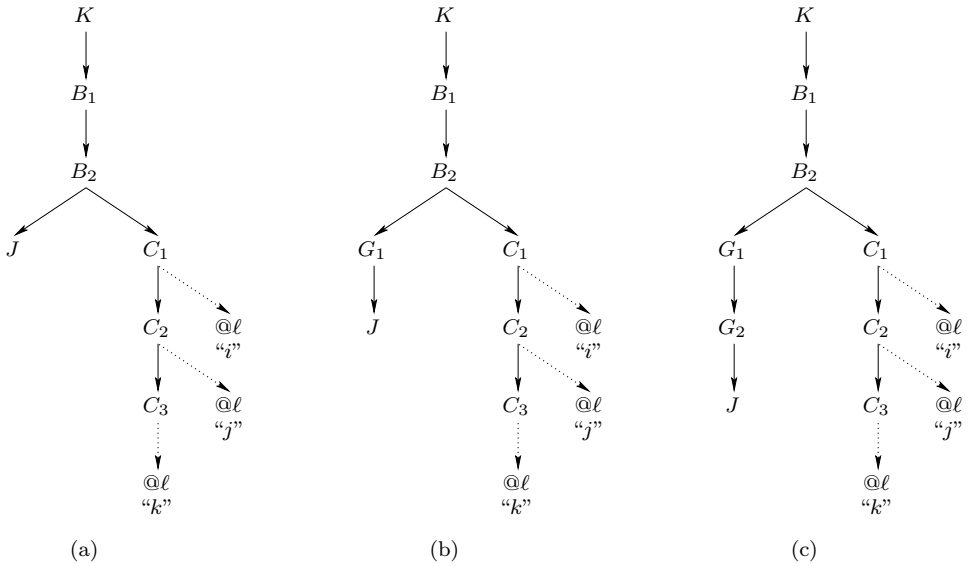


Fig. 17. Different alternatives for satisfying the second rule of $\Sigma_{\mathbf{ST}}$ in case STD($\underline{r}$, _) of the proof of Theorem 5.11.

Notice that this rule is not fully-specified since it does not say whether the father of J is a node of type either B_2 or G_1 or G_2. Also notice that in the left hand side of this rule we have a formula of the form $K[B_1[B_2[\varphi_1, \varphi_2]]]$ not using wildcard _, where K is the type of the root. The previous rule says that for every C-node v of a source tree T having i, j, k as values of attributes @f, @s, @t, and for every solution T' for T, T' must have as subtree at least one of the trees shown in Figure 17.

When constructing a solution for T_θ, we are actually constructing a truth assignment for θ. Let v be a C-node of T_θ with values i, j, k in attributes @f, @s, @t. To construct a solution T' for T we have to instantiate the second dependency of $\Sigma_{\mathbf{ST}}$ on values i, j and k, and then we have to choose among the trees shown in Figure 17 which one is going to be a subtree of T'. These alternatives represent three different ways of satisfying the clause stored in the children of v. As in the previous case, we say that a literal i has been assigned value 1 if there exists a node having at least two great-grandchildren, one of type J and another one having i in attribute @ℓ. Thus, in Figure 17 (a), literal i has been assigned value 1 since there is a node (of type K) having as great-grandchildren a node of type J and a node (of type C_1) having i in attribute @ℓ. On the other hand, in Figure 17 (b), literal j has been assigned value 1 since there is a node (of type B_1) having as great-grandchildren a node of type J and a node (of type C_2) having j in attribute @ℓ, and in Figure 17 (c), literal k has been assigned value 1 since there is a node (of type B_2) having as great-grandchildren a node of type J and a node (of type C_3) having k in attribute @ℓ. Notice that when constructing a truth assignment for θ, it is possible to choose value 1 for two complementary literals (when considering complementary literals in two distinct clauses). To take care of this problem we

use the same Boolean $\mathcal{CTQ}$-query Q as in the previous case:

$$\exists x \exists y \, (L(@p = x, @n = y) \ \wedge \ \_[\_[\_[J]], \ \_[\_[\_(@\ell = x)]]] \ \wedge \ \_[\_[\_[J]], \ \_[\_[\_(@\ell = y)]]]).$$

Exactly as in the previous case, we prove that for every 3-CNF propositional formula θ, we have that θ is satisfiable if and only if $\underline{certain}(Q, T_\theta) = \mathit{false}$, where T_θ is constructed from θ in PTIME as in the proof for STD($\_$, //) (see Section 5.3). This concludes the proof of case STD($\underline{r}$, $\_$) and the proof of Theorem 5.11.

B.3 Proof of Claim 6.16

(a) On the contrary, assume that $\mathit{alph}(w_{i+1}) \not\subseteq \mathit{alph}(w')$. Since h_i is a homomorphism from T_i to T', we have that $\mathit{alph}(w_i) \subseteq \mathit{alph}(w')$ and, therefore, $\mathit{alph}(w_{i+1}) \setminus \mathit{alph}(w_i) \not\subseteq \mathit{alph}(w') \setminus \mathit{alph}(w_i)$. Define string w'_i as follows: $\mathit{alph}(w'_i) = \mathit{alph}(w_i)$ and $\#_a(w'_i) = 1$, for every $a \in \mathit{alph}(w'_i)$. Then $w'_i \preceq w'$ since $\mathit{alph}(w_i) \subseteq \mathit{alph}(w')$. Thus, there exists $w''_i \in \mathit{min\_ext}(w'_i, P_{\mathbf{T}}(\lambda_{T_i}(v)))$ such that $w'_i \preceq w''_i \preceq w'$ since $w' \in \pi(P_{\mathbf{T}}(\lambda_{T'}(h_i(v)))) = \pi(P_{\mathbf{T}}(\lambda_{T_i}(v)))$. By definition of $\mathit{rep}(w_i, P_{\mathbf{T}}(\lambda_{T_i}(v)))$ we have that $w''_i \in \mathit{rep}(w_i, P_{\mathbf{T}}(\lambda_{T_i}(v)))$. Since $\mathit{alph}(w''_i) \subseteq \mathit{alph}(w')$ and $\mathit{alph}(w_{i+1}) \setminus \mathit{alph}(w_i) \not\subseteq \mathit{alph}(w') \setminus \mathit{alph}(w_i)$, we have that $\mathit{alph}(w_{i+1}) \setminus \mathit{alph}(w_i) \not\subseteq \mathit{alph}(w''_i) \setminus \mathit{alph}(w_i)$. We conclude that $w''_i \not\preceq_{w_i} w_{i+1}$, which contradicts the fact that $(\mathit{rep}(w_i, P_{\mathbf{T}}(\lambda_{T_i}(v))), \preceq_{w_i})$ has a maximum element (recall that $P_{\mathbf{T}}(\lambda_{T_i}(v))$ is a univocal regular expression) and $w_{i+1} \in \max_{\preceq_{w_i}} \mathit{rep}(w_i, P_{\mathbf{T}}(\lambda_{T_i}(v)))$.

(b) On the contrary, assume that $\#_a(w') > 1$. Since h_i is a homomorphism from T_i to T', we have that $\mathit{alph}(w_i) \subseteq \mathit{alph}(w')$. Define string w'_i as follows: $\mathit{alph}(w'_i) = \mathit{alph}(w_i)$, $\#_a(w'_i) = 2$ and $\#_b(w'_i) = 1$, for every $b \in \mathit{alph}(w'_i) \setminus \{a\}$. Then $w'_i \preceq w'$ since $\mathit{alph}(w_i) \subseteq \mathit{alph}(w')$ and $\#_a(w') > 1$. Thus, there exists $w''_i \in \mathit{min\_ext}(w'_i, P_{\mathbf{T}}(\lambda_{T_i}(v)))$ such that $w'_i \preceq w''_i \preceq w'$ since $w' \in \pi(P_{\mathbf{T}}(\lambda_{T'}(h_i(v)))) = \pi(P_{\mathbf{T}}(\lambda_{T_i}(v)))$. By definition of $\mathit{rep}(w_i, P_{\mathbf{T}}(\lambda_{T_i}(v)))$ we have that $w''_i \in \mathit{rep}(w_i, P_{\mathbf{T}}(\lambda_{T_i}(v)))$. Given that $\#_a(w''_i) \geq \#_a(w'_i) > 1$, $\#_a(w_{i+1}) = 1$ and $\#_a(w_i) > 1$, we conclude that $w''_i \not\preceq_{w_i} w_{i+1}$, which contradicts the fact that $(\mathit{rep}(w_i, P_{\mathbf{T}}(\lambda_{T_i}(v))), \preceq_{w_i})$ has a maximum element (recall that $P_{\mathbf{T}}(\lambda_{T_i}(v))$ is a univocal regular expression) and $w_{i+1} \in \max_{\preceq_{w_i}} \mathit{rep}(w_i, P_{\mathbf{T}}(\lambda_{T_i}(v)))$. This concludes the proof of the claim.

B.4 Proof of Claim 6.17

By contradiction, assume that there exist strings w_1, w_2 and $a \in \mathit{alph}(r)$ such that $\mathit{rep}(w_1, r) \neq \emptyset$, $w_2 \in \max_{\preceq_{w_1}} \mathit{rep}(w_1, r)$ and $2 \leq \#_a(w_2) < \#_a(w_1)$. Since $c(r) \leq 1$, we have that $w_2 \notin \mathit{fixed}_a(r)$ and, hence, there exists a string $w_3 \in \pi(r)$ such that $w_2 \preceq w_3$ and $\#_a(w_2) < \#_a(w_3)$. Let w_4 be a string such that $\mathit{alph}(w_4) = \mathit{alph}(w_1)$, $\#_a(w_4) = \#_a(w_2) + 1$ and for every $b \in \mathit{alph}(w_1) \setminus \{a\}$, $\#_b(w_4) = 1$. Since $w_4 \preceq w_3$ and $w_3 \in \pi(r)$, we conclude that $\mathit{min\_ext}(w_4, r) \neq \emptyset$. Let $w_5 \in \mathit{min\_ext}(w_4, r)$. By definition of $\mathit{min\_ext}(w_4, r)$, we have that $\#_a(w_5) \geq \#_a(w_4) > \#_a(w_2)$, and by definition of w_4 and $\mathit{rep}(w_1, r)$, we have that $w_5 \in \mathit{rep}(w_1, r)$. We conclude that there exists a string $w_5 \in \mathit{rep}(w_1, r)$ such that $\#_a(w_2) < \#_a(w_5)$. Thus, given that $\#_a(w_2) < \#_a(w_1)$, we have that $w_5 \not\preceq_{w_1} w_2$, which contradicts the fact that $w_2 \in \max_{\preceq_{w_1}} \mathit{rep}(w_1, r)$ and $(\mathit{rep}(w_1, r), \preceq_{w_1})$ has a maximum element.

B.5 Proof of Claim 6.22

Let s be a string such that $alph(s) = alph(w)$ and $\#_a(s) = \min\{\#_a(w'), \#_a(w)\}$, for every $a \in alph(s)$. We note that $s \preceq w$ and $s \preceq w'$. Since $w' \in \pi(r)$, there exists $s' \in min\_ext(s, r)$ such that $s \preceq s' \preceq w'$. By definition of $rep(w, r)$, we have that $s' \in rep(w, r)$ and, hence, there exists $i \in [1, n]$ such that $s' \preceq_w w_i$. We conclude that $w' \preceq_w w_i$ since (1) for every $a \in alph(w)$ such that $\#_a(w_i) < \#_a(w)$, we have that $\#_a(w_i) \geq \#_a(s') \geq \#_a(s) = \#_a(w')$ and (2) $alph(w_i) \setminus alph(w) \subseteq alph(s') \setminus alph(w) \subseteq alph(w') \setminus alph(w)$.

Received Month Year; revised Month Year; accepted Month Year