Querying Regular Sets of XML Documents[⋆]

Slawomir Staworko[1,2], Emmanuel Filiot[1], and Jan Chomicki[2]

[1] INRIA Lille Nord-Europe
[2] Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY

Abstract. We investigate the problem of querying (regular) sets of XML documents represented with tree automata and we consider n-ary tree automata queries whose expressive power captures MSO on trees. Because finite automata can represent infinite sets of documents, we propose the notions of *universal* and *existential* query answers, answers that are present resp. in all and some documents. We study complexity of query answering and show that computing existential query answers is in PTIME if we assume the arity of the query to be a fixed parameter. On the other hand, computing universal query answers is EXPTIME-complete, but we show that it is in PTIME if we assume that the query is fixed (data complexity). Finally, we argue that the framework captures problems central to many novel XML applications like querying inconsistent XML documents. In particular, we demonstrate how to use our framework to compute consistent query answers in XML documents that do not satisfy the schema. This solution significantly extends our previous results in this area.

1 Introduction

In this paper we investigate the problem of querying potentially large sets of XML documents having a small compact representation by a finite tree automaton. Our work is inspired by the problem of evaluating a query in a document that possibly does not satisfy the schema. Since the satisfaction of the schema is usually assumed during formulation of a query, evaluation of the query against a document that does not satisfy the schema may yield incorrect and misleading answers. This problem has been previously recognized in the setting of relational databases [3] and the proposed framework of repairs and consistent query answers (CQA) has been adapted to semi-structured databases [18, 10, 11]. A *repair* is a document satisfying the schema and obtained from the original document by a minimal number of standard edit operations [1, 5]: inserting, deleting, and modifying an element of the document. An answer is *consistent* (also called *valid*) if it is an answer to the query in every repair.

Our research shows that the set of repairs of a document is a regular language that has a compact representation by a finite (weighted) tree automata.

[⋆] Research supported by NSF Grants IIS-0119186 and IIS-0307434.

This reduces the problem of querying the set of repairs to a more general problem of querying a regular set of documents represented by a tree automaton. Because the set of documents represented by an automaton can be very large, an approach where we return the collection of the sets of answers in every tree may be simply inappropriate for many applications. Consequently, we propose two ways of computing answers to queries in sets of documents: (i) *universal* answers that are present in every document and (ii) *existential* answers that are present in some document. Obviously, universal answers capture consistent answers. We also note that the existential answers capture the notion of *possible* answers, i.e. the answers that are present in some repair [10], often considered next to consistent answers in the framework of CQA. Another motivation to study the problem of querying regular sets of documents is that there are other frameworks where the queries are evaluated not on the instance itself but rather on the set of instances obtained by some (often nondeterministic) process from the original one. An example is XML data exchange [4]: the queries are formulated against *target* schema whose instances (called *solutions*) are obtained from a *source* instance with a set of *source-to-target dependencies* specifying how the parts of the source instance translate to an instance in target schema. In the data exchange scenario, the notion of universal and existential answers coincide resp. with *certain* and *maybe* answers [15]. It is, however, yet to be seen if the set of solutions can be represented by a tree automaton.

Now, we briefly summarize our contributions:

– We define the problem of universal and existential querying of sets of documents represented by a finite tree automaton with attribute values and we consider n-ary queries expressed with tree automata.
– We thoroughly study the complexity of computing existential and universal answer. For computing existential answers, we show that its combined complexity is NP-complete, but its complexity parametrized by the arity of the query is FPT (*Fixed Parameter Tractable*) [9]. This result is not surprising as the number of answers to an n-ary query can be exponential in n. Computing universal answers is, however, EXPTIME-complete in terms of both the combined and parametric complexity. On the other hand we show that its data-complexity is PTIME.
– We show how to compute consistent query answers by constructing a *repair automaton* that defines the set of all repairs of a document w.r.t. a schema (expressed with a tree automaton). This extends our previous results [18, 17] in several directions: (i) we consider n-ary automata queries whose expressive power captures MSO as compared to a restricted class of unary Core XPath queries, (ii) schema can be expressed using tree automata which are strictly more expressive than DTDs, (iii) we show that data complexity of computing consistent answer to any n-ary automata query is PTIME, (iv) we consider a more general set of editing tree operations that allow to operate on inner nodes as compared to operations on leaves only.

Related work [10] investigates querying XML documents that are valid but violate functional dependencies. Two repairing actions are considered: updating element values with a *null* value and marking nodes as unreliable. Such nodes are simply omitted in the query answers. Only simple descending path queries $a_1/a_2/\ldots/a_n$ are considered. A polynomial algorithm for computing consistent and possible answers is presented. [11] considers editing operations operating on leaves only to define the set of repairs for consistent querying of documents that violate functional dependencies. A different, set-theoretic, notion of minimality is used when defining repairs. Both consistent and possible answers are considered. For restricted classes of functional dependencies and DTDs a polynomial algorithm is proposed to compute consistent answers to n-ary conjunctions of path expressions.

[12] considers evaluation of monadic Datalog queries on compressed trees (represented by a tree automaton) and shows that combined complexity is PSPACE-complete and data complexity is PTIME. We note that this framework is close to existential querying: a (finite) set of trees can be gathered in one tree with a new root symbol. The difference between the complexity results (for unary queries we have polynomial algorithm in terms of combined and data complexity) comes from different representations of queries. [16] extends this approach to trees represented by straight-line context-free grammars which is strictly stronger than regular languages.

[21] study the problem of checking if a document is within a specified *alignment distance* to the given schema. We note that the edit distance is more general than the alignment distance which imposes certain conditions on the sequence of editing operations [6] and hence our approach is more general. A compact representation of all repairs (obtained with restricted sequences of editing operations) as a regular language is also presented.

The paper is organized as follows. Section 2 contains basic XML notions and streaming tree automata. In Section 3 we define existential and universal answers to n-ary queries and present algorithm for computing them. In Section 4 we study computational implications of our framework. Section 5 shows how to use our framework to compute consistent query answers. Because of space limitations the proofs are omitted; they can be found in the appendix available at [19].

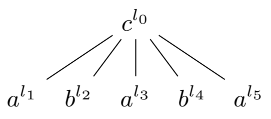
2 Basic notions

2.1 Trees and streams

We model XML documents using ordered unranked trees whose nodes are labeled with elements of a finite set of symbols Σ. Every node is additionally labeled with an attribute whose value is drawn from an infinite set Λ. We denote the set of all trees by $\mathcal{T}$. The size of t, denoted by $|t|$, is the number of nodes of t.

In our framework only the attribute values are used to define query answers. The attributes of a tree can store unique node identifiers and we call such trees *standard*. In general, however, the attribute can store (possibly repeated) data values.

In this paper we work mainly with the serialized version of trees, i.e. well-formed sequences of opening and closing tags (corresponding to a preorder traversal of the tree) with attribute values associated to the opening tags. The set of all tags is $\Sigma_{\Diamond} = \{\langle a\rangle | a \in \Sigma\} \cup \{\langle /a\rangle | a \in \Sigma\}$. When working with a serialized version of a tree $e_1, \ldots, e_n$ we write $tag(e_i)$ for the tag of e_i and $att(e_i)$ for the attribute value of e_i. Given a tree t its serialized version is denoted by $\bar{t}$. We



	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
tag	$\langle c\rangle$	$\langle a\rangle$	$\langle /a\rangle$	$\langle b\rangle$	$\langle /b\rangle$	$\langle a\rangle$	$\langle /a\rangle$	$\langle b\rangle$	$\langle /b\rangle$	$\langle a\rangle$	$\langle /a\rangle$	$\langle /c\rangle$
att	l_0	l_1	–	l_2	–	l_3	–	l_4	–	l_5	–	–

Fig. 1. An example of a tree t_0 and the corresponding tag sequence

use also unranked terms over the signature $\Sigma \times \Lambda$ to represent trees. Figure 1 contains an example of a tree $t_0 = c^{l_0}(a^{l_1}, b^{l_2}, a^{l_3}, b^{l_4}, a^{l_5})$ and its serialization.

2.2 Streaming tree automata

To capture regular tree languages we use streaming tree automata [13] which are Visibly Pushdown Automata [2] working on serializations of unranked trees. They are allow to capture Extended DTDs [14] which makes them equivalent to standard (ranked) tree automata working on encodings of unranked trees [8]. We choose this model because it is better fitted to capture repairs of XML documents: repairs are obtained by edit operations on nodes and those easily translate to string edit operations on pairs of matching tags. We also extend this model by allowing it to specify attribute values: this way the repairs can be defined in terms of the attribute values of the original document. For simplicity of presentation we fix the set of attribute values Λ.

Definition 1. *An* attributed streaming tree automaton *(attributed STA) is a tuple $M = (\Sigma, \Gamma_M, Q_M, I_M, \Delta_M, F_M)$, where Σ is a finite set of node* labels*, Γ_M is a finite set of* stack symbols*, Q_M is a finite set of* states*, and $I_M \subseteq Q_M$ is the set of* initial *states, $F_M \subseteq Q_M$ is the set of* final *states. Δ_M is a finite set of* transitions *of one of the following types:* opening *transition $p \xrightarrow[l]{\langle a\rangle:\gamma} q$, and* closing *transition $p \xrightarrow{\langle /a\rangle:\gamma} q$, where $p, q \in Q_M$, $a \in \Sigma$, $\gamma \in \Gamma_M$, and $l \in \Lambda \cup \{*\}$ ($*$ is a* wildcard*).*

The *size* of M is $|M| = |Q_M| + |\Delta_M|$ and by $Dom(M) \subseteq \Lambda \cup \{*\}$ we denote the set of different attribute labels used in Δ_M.

Essentially, an attributed STA is a push-down automaton working on sequences of tags with the stack manipulation restricted to: placing a symbol on the stack when reading an opening tag, and removing top symbol from the stack when reading a closing tag. A *configuration* is an tuple $(q, \bar{\alpha}, \bar{s})$, where q is the current state, $\bar{\alpha}$ is the current stack, and $\bar{s}$ is the remaining (possibly unbalanced) tag sequence. The *move* relation $\rightarrow_M$ is a binary relation on configurations defined as follows:

(i) $(q, \bar{\alpha}, e \cdot \bar{s}) \rightarrow_M (p, \gamma \cdot \bar{\alpha}, \bar{s})$ if $q \xrightarrow[l]{\langle a \rangle : \gamma} p \in \Delta_M$, $tag(e) = \langle a \rangle$, and if $l \neq *$, then $att(e) = l$.

(ii) $(q, \gamma \cdot \bar{\alpha}, e \cdot \bar{s}) \rightarrow_M (p, \bar{\alpha}, \bar{s})$ if $q \xrightarrow{\langle /a \rangle : \gamma} p \in \Delta_M$ and $tag(e) = \langle /a \rangle$.

$\rightarrow_M^*$ is the reflexive and transitive closure of $\rightarrow_M$. The *tree language* of M is defined as $L(M) = \{t \mid (q, \varepsilon, \bar{t}) \rightarrow_M^* (p, \varepsilon, \varepsilon), q \in I_M, p \in F_M\}$.

An *STA* is an attributed STA that imposes no restrictions on the attribute values, i.e. it uses only $*$. Then, we also omit the attribute labels altogether. Figure 2 contains an example of an STA M_0 that recognizes trees with root label c satisfying the DTD D_0 given by the rules: $c \rightarrow (a \cdot b)^* \cdot a$, $a \rightarrow \epsilon$, $b \rightarrow \epsilon$. We observe that the tree t_0 (Fig. 1) satisfies M_0.

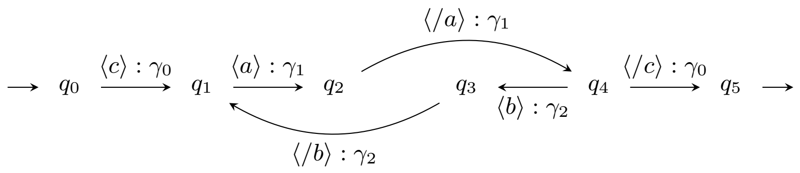

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, "\langle c \rangle : \gamma_0"] & q_1 \arrow[r, "\langle a \rangle : \gamma_1"] & q_2 \arrow[rr, bend left, "\langle /a \rangle : \gamma_1"] & q_3 \arrow[ll, bend left, "\langle /b \rangle : \gamma_2"] & q_4 \arrow[l, "\langle b \rangle : \gamma_2"] \arrow[r, "\langle /c \rangle : \gamma_0"] & q_5 \arrow[r] & {}
\end{tikzcd}$$

Fig. 2. The STA M_0 for the DTD D_0

Weighted STAs To represent the sets of minimal repairs we further extend attributed STAs by assigning to every transition its *weight*, a non-negative real value. The weights are used to restrict the set of recognized trees to those with a run of minimal summary weight.

More formally, a weight of a run is the sum the weights of the transition used at each step. For a tree (with an accepting run) we associate the minimal weight of its accepting run. $L(M)$ contains only the trees that whose weight is equal to the minimum of weights of all trees (with an accepting run).

2.3 STA queries

We define an *n-ary query* as a function that takes a tree and returns a set of n-ary tuples of values from Λ (that are used in the tree).

To define an n-ary query we use an extension of (standard) STAs where with every opening transition we associate a set of variables $X \subseteq \{x_1, \dots, x_n\}$ that indicates the positions of the resulting tuple $(x_1, \ldots, x_n)$ that are to be filled when the transition is used in a run (the positions are filled with the attribute value). Each position of the resulting tuple has to be filled *exactly* once during a run, otherwise the run is non-accepting. The set of query answers consists of all tuples obtained from *all* accepting runs. We use name *STA queries* to refer to such automata and to distinguish them from STAs we use Greek capital letters $\Phi, \Psi, \ldots$. We also put the sets X in the superscript of the opening transitions. Figure 3 contains an STA query Φ selecting pairs (x_1, x_2) of any node a and its immediate right sibling b (satisfaction of DTD D_0 is assumed). On the tree t_0

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[loop above, "\Sigma_\Diamond : \gamma_0"] \arrow[r, "\langle a \rangle : \gamma_1", "\{x_1\}"'] & q_1 \arrow[r, "\langle /a \rangle : \gamma_1"] & q_2 \arrow[r, "\langle b \rangle : \gamma_2", "\{x_2\}"'] & q_3 \arrow[r, "\langle /b \rangle : \gamma_2"] & q_4 \arrow[loop above, "\Sigma_\Diamond : \gamma_0"] \arrow[r] & {}
\end{tikzcd}$$

Fig. 3. An example of a binary STA query

(Fig. 1) this query has two answers: (l_1, l_2) and (l_3, l_4).

We define query answers formally as follows. A configuration of an n-ary STA query Φ is an element $(q, \bar{\alpha}, \bar{s}, \tau)$, where q, $\bar{\alpha}$, and $\bar{t}$ are as before, and $\tau \in (\Lambda \cup \{\bot\})^n$ is the tuple of values assigned so far, with $\bot$ (the *null* value) indicating that the value has not been yet assigned. We define the *move* relation analogously:

(i) $(q, \bar{\alpha}, e \cdot \bar{s}, \tau) \to_\Phi (p, \gamma \cdot \bar{\alpha}, \bar{s}, \tau')$ if $q \xrightarrow{\langle a \rangle : \gamma} p \in \Delta_\Phi$, $tag(e) = \langle a \rangle$, $\tau_i = \bot$ for every $x_i \in X$, and $\tau' = \tau[X/att(e)]$,

(ii) $(q, \gamma \cdot \bar{\alpha}, e \cdot \bar{s}, \tau) \to_\Phi (p, \bar{\alpha}, \bar{s}, \tau)$ if $q \xrightarrow{\langle /a \rangle : \gamma} p \in \Delta_\Phi$ and $tag(e) = \langle /a \rangle$.

Again, $\to_\Phi^*$ is the reflexive and transitive closure of $\to_\Phi$. The set of *answers* to an n-ary STA query Φ in t is $QA(\Phi, t) = \{\tau \in \Lambda^n | (q, \varepsilon, \bar{t}, (\bot, \ldots, \bot)) \to_\Phi^* (p, \varepsilon, \varepsilon, \tau), p \in I_\Phi, q \in F_\Phi\}$.

It is known [13] that over standard trees n-ary STA queries have the same expressive power as MSO formulas with n free variables over the first-child, next-sibling signature of unranked trees In particular, STA queries subsume unary CoreXPath queries. It should be noted, however, that similarly translating an MSO formulas may yield a automata of non-elementary size [20, 13]. This generally applies also to CoreXPath queries, but it can be easily seen that simple descending XPath queries with no test expressions translate to STA queries of linear size.

3 Existential and universal querying of attributed STAs

Now, we consider querying sets of trees defined by attributed STAs. First, we note that the set of trees defined by an attributed STA may be infinite and even if the STA is weighted, the number of trees may be exponential in the size of the automaton (modulo different attribute values). Therefore, an approach where we return the collection of the sets of answers obtained in every tree may be inappropriate for many applications. Consequently, we propose two ways of querying sets of trees.

Definition 2 (Universal and existential answers). *Given a (possibly weighted) attributed STA M such that $L(M) \neq \varnothing$ and n-ary STA query Φ*

- *the* universal *answers to Φ in M are $QA^{\forall}(\Phi, M) = \bigcap_{t \in L(M)} QA(\Phi, t)$,*
- *the* existential *answers to Φ in M are $QA^{\exists}(\Phi, M) = \bigcup_{t \in L(M)} QA(\Phi, t)$.*

We note that the language defined by a weighted automaton is empty if and only if the corresponding automaton without weights defines an empty language. Emptiness of an attributed STA can be tested in cubic time using the classical algorithm for PDAs. Because our algorithms have complexity of a higher degree, from now on we will assume that we always deal with automata defining nonempty language. Also, the set $QA^{\forall}(\Phi, M)$ is always finite, but $QA^{\exists}(\Phi, M)$ may be infinite if M uses wildcards. Hence, we also allow to use wildcards in answers to finitely represent $QA^{\exists}$.

Now, we present Algorithm 1 computing existential answers to an n-ary query Φ in an attributed STA M. This algorithm is is based on a product technique

Algorithm 1 Computing existential answers to n-ary Φ in M

```
function QA$^{\exists}(\Phi, M)$
    macros: $Q :\equiv Q_\Phi \times Q_M,\ I :\equiv I_\Phi \times I_M,\ F :\equiv F_\Phi \times F_M$
        $(p_1, q_1) \xrightarrow[X:=l]{x:(\gamma_1,\gamma_2)} (p_2, q_2) :\equiv p_1 \xrightarrow[X]{x:\gamma_1} p_2 \in \Delta_\Phi \wedge q_1 \xrightarrow[l]{x:\gamma_2} q_2 \in \Delta_M$,
 1:  for $(u, v) \in Q^2$ do
 2:      $T_0[u, v] = \begin{cases} \{(\bot, \ldots, \bot)\}, & \text{if } u = v, \\ \varnothing, & \text{otherwise.} \end{cases}$
 3:  for $i \leftarrow 1, \ldots, n|Q|^2$ do
 4:      for $(u, v) \in Q^2$ do
 5:          $H_i[u, v] = T_{i-1}[u, v]$
 6:      for $j \leftarrow 1, \ldots, \lceil \log(n|Q|^2) \rceil$ do
 7:          for $(u, v) \in Q^2$ do
 8:              $H_i[u, v] \leftarrow H_i[u, v] \cup \bigcup\{merge(H_i[u, w], H_i[w, v]) | w \in Q\}$
 9:      for $(u, v) \in Q^2$ do
10:          $T_i[u, v] \leftarrow T_{i-1}[u, v] \cup \bigcup\{assign_X(H_i[u', v'], l) | u \xrightarrow[X:=l]{\langle a \rangle:\gamma} u' \wedge v' \xrightarrow{\langle /a \rangle:\gamma} v\}$
11:    return $\{\tau \in (\Lambda \cup \{*\})^n | \tau \in T_{n|Q|^2}[u, v] \wedge u \in I \wedge v \in F\}$
end function
```

of the two input automata. Essentially, it evaluates the query on every tree of height and width $\leq n|Q|^2$, where $Q = Q_\Phi \times Q_M$. This procedure yields correct results thanks to pumping properties of STAs. In particular, if there is a tree

$t \in L(M)$ and tuple $\tau \in QA^{\exists}(\Phi, t)$, there is also a tree t^* whose depth and width is bounded by $n|Q|^2$, such that $t^* \in L(M)$ and $\tau \in QA^{\exists}(\Phi, t^*)$. Consequently, we need to consider query runs of depth and width bound by $n|Q|^2$. This space can be explored with a simple dynamic programming technique because runs of an STA on a tree share the structure of the tree. In particular, T_i and H_i store all tuples "*collected*" from runs on resp. trees and hedges (sequences of trees) of depth $\leq i$ and width $\leq n|Q|^2$. We use $assign_X(A, l)$ to assign the value l on positions X to every tuple from A (tuples having a value different from $\perp$ on those positions are discarded). $merge(A, B)$ returns the set of merged tuples from sets A and B (two tuples having assigned value on the same position cannot be merged). An easy complexity analysis shows that:

Theorem 1. *For an attributed STA M and n-ary query Φ Algorithm 1 computes $QA^{\exists}(\Phi, M)$ in time $O((|\Phi||M|)^6|Dom(M)|^{2n})$.*

Extending the algorithm to weighted attributed STAs is not difficult because minimal runs enjoy optimal substructure properties. Also, if we first perform the run of the algorithm on the attributed STA where every attribute value has been replaced by one unique constant, we can find which tuples in the intermediate steps are removed by the *apply* and *merge* operations. This allow us to replace the $|Dom(M)|^{2n}$ factor by $2^{2n}|QA^{\exists}(\Phi, M)|$.

Corollary 1. *For any weighted attributed STA M and n-ary query Φ the set $QA^{\exists}(\Phi, M)$ can be computed in time $O(2^{2n}(|\Phi||M|)^6|QA^{\exists}(\Phi, M)|)$.*

We observe that the high complexity in terms of $|M|$ comes from the particular pumping properties of STAs. We note, however, that the algorithm could be easily adapted to standard tree automata working on binary representation of unranked trees. Those automata enjoy nicer pumping properties and in particular the complexity would be cubic in terms of the size of the input automata.

To compute universal answers we observe that $QA^{\forall}(\Phi, M) \subseteq QA^{\exists}(\Phi, M)$. Instead of computing universal answers directly, we compute $QA^{\exists}(\Phi, M)$ and on every tuple we perform *tuple check*, i.e. we find if the tuple is an answer in every tree. Because we use it as a tool for analyzing the complexity of universal and existential query answers, we formally define it:

> ***Existential (universal) tuple check*** is a decision problem where given an attributed STA M, an n-ary STA query Φ, and a tuple $\tau \in (\Lambda \cup \{*\})^n$ find if τ is an existential (universal resp.) answer to Φ in M.

Theorem 2. *Universal tuple check can be decided in time $O(f(|\Phi|)|M|^3))$, where $f(|\Phi|) = 2^{|\Phi|^2 2^{2n}}$ (and $O(f(|\Phi|)log(|M|)|M|^4))$ if M is weighted).*

Consequently, we obtain a characterization of the simple procedure of computing universal answers.

Corollary 2. *For any weighted attributed STA M and n-ary query Φ the set $QA^{\forall}(\Phi, M)$ can be computed in time $O(2^{|\Phi|^2 2^{2n}}|\Phi|^6|M|^6|QA^{\exists}(\Phi, M)|)$.*

4 Complexity analysis

Now, we analyze tractability of computing existential and universal query answers by investigating the complexity of universal and existential tuple check (defined in the previous section). We start with combined complexity where all the elements are considered to be the part of the input.

Theorem 3. *Combined complexity of existential and universal tuple check are NP-complete and EXPTIME-complete respectively.*

We remark that the EXPTIME-hardness is proved with a reduction of the containment problem of two tree automata to a universal tuple check where one of the automata is treated as a 0-ary query.

Next, we observe that if the arity of the query is fixed, then the existential tuple check can be done in polynomial time. Moreover, the degree of the polynomial does not depend on the arity of the query. Hence, we can characterize the (multiplicative) *fixed parametric complexity* [9].

Corollary 3. *When the arity of the STA query is a parameter, the existential tuple check is FPT (Fixed Parameter Tractable).*

We note that universal tuple check remains intractable when fixing the arity of the query as the EXPTIME-hardness proof uses an STA query of arity 0.

Finally, Theorem 2 give us a characterization of data complexity [22] of universal query answers (the query is assumed to be fixed).

Corollary 4. *Data complexity of universal tuple check is PTIME.*

5 Consistent querying of XML documents

We recall the basic notions of the framework of consistent query answers for semi-structured databases. The process of repairing an XML document is modeled with the standard edit operations on trees: (i) *renaming* the node, (ii) *deleting* a node (different than the root) which involves promoting its children to the parent of the node (placed in the same order from the position of the node), and (iii) *inserting* a node (different than the root) with a possible adoption of a list of subsequent children from the parent of the node (dual to the delete operation). With every editing operation we associate a cost: c_R, c_D, and c_I the costs for renaming, deleting, and inserting a node respectively. We note, however, that our approach can be easily extended to weights that depend on properties of the node, for example its label.

The *edit distance* $d(t_1, t_2)$ between two trees t_1 and t_2 is the minimal cost of transforming t_1 to t_2 with a series of edit operations. Given a tree t and an STA M (expressing the schema), we define the distance between t and M, denoted $d(M, t)$, as the minimum edit distance between t and any t' valid w.r.t. M.

A *repair* of t w.r.t. M is a tree $t' \in L(M)$ such that $d(t, t') = d(t, M)$. By $Rep(t, M)$ we denote the set of all repairs of t w.r.t. M. Given an n-ary STA query Φ we say that a tuple τ is a *consistent* (or *valid*) answer to Φ in t w.r.t. M if and only if τ is an answer to Φ in every repair of t w.r.t. M. By $CQA(\Phi, t, M)$ we denote the set of all consistent answers to Φ in t w.r.t. M.

5.1 Repair automaton

In this part we define a weighted attributed STA that defines the set of all repairs of a document. For ease of construction we allow the use of ϵ-transitions. As they have no attribute value and perform no operations on the stack, we can easily remove them by standard closure (remembering to aggregate the weight). Also, we make a natural assumption that the schema does not allow an empty tree.

Definition 3. *Let M be an STA, t be a tree with n nodes, and $\bar{t} = (e_1, \ldots, e_{2n})$ be the serialization of t. Assume that the nodes of t are numbered with consecutive natural numbers $0, 1, \ldots, n-1$ in the standard document order. For an opening or closing tag e_i let m_i be the number assigned to that node.*

The repair automaton *of t w.r.t. M is a weighted attributed STA $R(t, M) = (\Sigma, \Gamma_R, Q_R, I_R, \Delta_R, F_R)$ where: $\Gamma_R = \Gamma_M \cup \Gamma_M \times \{1, \ldots, n\}$, $Q_R = I_M \times \{0\} \cup Q_M \times \{1, \ldots, 2n-1\} \cup F_M \times \{2n\}$, $I_R = I_M \times \{0\}$, and $F_R = F_M \times \{2n\}$. The state (q, i) will be denoted as q^i. The transitions of Δ_R (with their attributes and weights) capture edit operations performed on the tag stream as follows:*

– $q^{i-1} \xrightarrow[att(e_i)]{x:(\gamma, m_i)} p^i$ renaming *e_i to x if $tag(e_i) \neq x$ and* doing nothing *if $tag(e_i) = x$, for every $i \in \{1, \ldots, 2n\}$ and every $q \xrightarrow{x:\gamma} p \in \Delta_M$; its weight is $c_R/2$ if $tag(e_i) \neq x$ and 0 if $tag(e_i) = x$.*
– $q^i \xrightarrow{x:\gamma} p^i$ inserting *x (before e_i), for every $i \in \{2, \ldots, 2n-1\}$ and every $q \xrightarrow{x:\gamma} p \in \Delta_M$; its weight is $c_I/2$ and it is attributed with $*$ if x is an opening tag.*
– $q^{i-1} \xrightarrow{\varepsilon} q^i$ deleting *e_i, for every $i \in \{2, \ldots, 2n-1\}$ and every $q \in Q_M$; its weight is $c_D/2$.*

Example 1. Figure 4 contains the repair automaton of the tree $t_1 = c^{l_0}(a^{l_1}, b^{l_2})$ (Fig. 1) and the STA M_0 (Fig. 2). The weights are assumed to be $w_R = w_I = w_D = 1$. Because this graph is very intricate, for clarity we present in all details only the transitions that produce the minimal trees. Also, γ_i^j is short for (γ_i, j). $R(t_1, M_0)$ defines the set of trees $\{c^{l_0}(a^{l_1})\} \cup \{c^{l_0}(a^{l_1}, b^{l_2}, a^l) | l \in \Lambda\}$, i.e. the set of repairs of t_1 w.r.t. M_0.

Theorem 4. *For any tree t, any STA M, and any STA query Φ we have that $Rep(t, M) = L(R(t, M))$ and $CQA(\Phi, t, M) = QA^\forall(\Phi, R(t, M))$.*

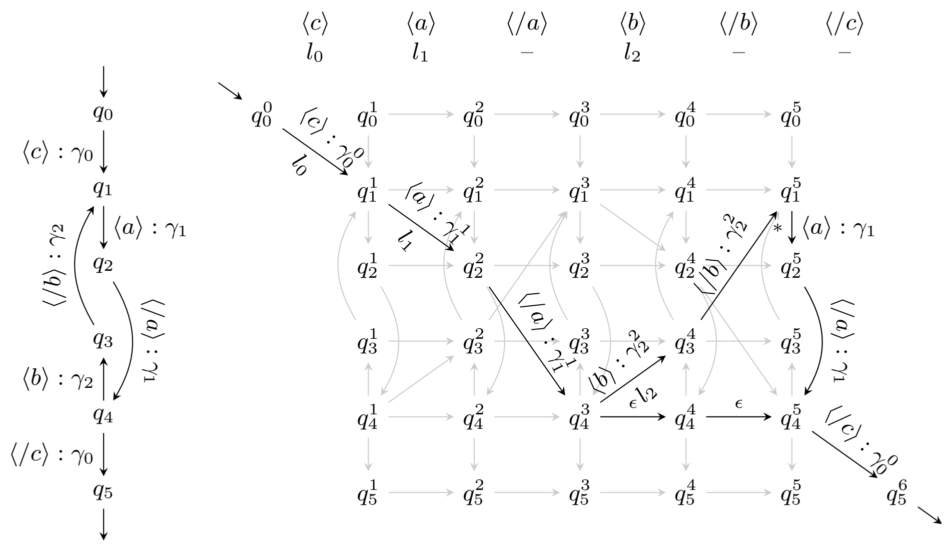
5.2 Complexity analysis

From Corollary 2 we get directly:

Corollary 5. *The data complexity of computing consistent answers to an n-ary query w.r.t. an STA is PTIME.*

To further analyse the tractability of consistent query answers we investigate the complexity of the problem of tuple check for consistent query answers [17]. Similarly to universal answers the problem is intractable.

Theorem 5. *The combined complexity of consistent query answers is Π_p^2-complete if $w_I > 0$ and EXPTIME-complete if $w_I = 0$.*



$\langle c\rangle$
l_0
$\langle a\rangle$
l_1
$\langle/a\rangle$
$-$
$\langle b\rangle$
l_2
$\langle/b\rangle$
$-$
$\langle/c\rangle$
$-$
q_0
$\langle c\rangle : \gamma_0$
q_1
$\langle a\rangle : \gamma_1$
q_2
$\langle/b\rangle : \gamma_2$
$\langle/a\rangle : \gamma_1$
q_3
$\langle b\rangle : \gamma_2$
q_4
$\langle/c\rangle : \gamma_0$
q_5
q_0^0
$\langle c\rangle : \gamma_0$
l_0
$\langle a\rangle : \gamma_1$
l_1
$\langle/a\rangle : \gamma_1$
$\langle b\rangle : \gamma_2$
$\epsilon\, l_2$
ϵ
$\langle/b\rangle : \gamma_2$
$*$
$\langle a\rangle : \gamma_1$
$\langle/a\rangle : \gamma_1$
$\langle/c\rangle : \gamma_0$
q_5^6

Fig. 4. A repair graph $R(t_1, M_0)$

6 Conclusions and future work

In this paper we presented the framework of querying regular sets of XML documents. We considered the class of n-ary automata queries and introduced notions of *universal* and *existential* answers, i.e. answers present in every and some documents represented by a tree automata. We investigated computational properties of our framework and presented algorithms for computing universal and existential answers. Finally, we used our framework to compute consistent query answers in XML documents that do not satisfy the schema. This solution significantly extends our previous results in this area [18].

We envision several possible directions of further study. Firstly, we would like to investigate the problem of querying regular sets of trees with a query defined in a logical formalism which allow comparison of data-values. Such formalisms have been studied in [7] where FO logic over data trees is considered. It is shown that FO is decidable for the two-variable fragment with a successor relation and a predicate to compare data-values. Another important direction of future study is to investigate if the setting of data exchange could be effectively captured by our framework. Finally, we would like to investigate using more expressive formalisms, for instance context-free tree grammars, to represent the sets of trees.

References

1. S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J. L. Wiener. Incremental Maintenance for Materialized Views over Semistructured Data. In *International Conference on Very Large Data Bases (VLDB)*, pages 38–49, 1998.
2. R. Alur and P. Madhusudan. Visibly Pushdown Languages. In *ACM Symposium on Theory of Computing (STOC)*, pages 202–211, 2004.
3. M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.
4. M. Arenas and L. Libkin. XML Data Exchange: Consistency and Query Answering. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 13–24, 2005.
5. A Balmin, Y. Papakonstantinou, and V. Vianu. Incremental Validation of XML Documents. *ACM Transactions on Database Systems (TODS)*, 29(4):710–751, December 2004.
6. P. Bille. Tree Edit Distance, Aligment and Inclusion. Technical Report TR-2003-23, The IT University of Copenhagen, 2003.
7. M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and xml reasoning. In *PODS*, pages 10–19, 2006.
8. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997. release 2007.
9. M. R. Fellows. Parameterized Complexity: The Main Ideas and Connections to Practical Computing. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 61, 2002.
10. S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Repairs and Consistent Answers for XML Data with Functional Dependencies. In *International XML Database Symposium (Xsym)*, volume 2824 of *Lecture Notes in Computer Science*, pages 238–253. Springer, 2003.
11. S. Flesca, F Furfaro, S. Greco, and E. Zumpano. Querying and Repairing Inconsistent XML Data. In *Web Information Systems Engineering (WISE)*, pages 175–188, 2005.
12. M. Frick, M. Grohe, and C. Koch. Query Evaluation on Compressed Trees. In *Logic in Computer Science (LICS)*, pages 188–200, 2003.
13. O. Gauwin, A.-C. Caron, J. Niehren, and S. Tison. Complexity of Earliest Query Answering with Streaming Tree Automata. In *Programming Language Technologies for XML (PLAN-X)*, 2008.
14. V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly Pushdown Automata for Streaming XML. In *International Conference on World Wide Web (WWW)*, pages 1053–1062, 2007.
15. L. Libkin. Data Exchange and Incomplete Information. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 60–69, 2006.
16. M. Lohrey and S. Maneth. The Complexity of Tree Automata and XPath on Grammar-compressed Trees. *Theoretical Computer Science (TCS)*, 363(2):196–210, 2006.
17. S. Staworko. *Declarative Inconsistencies Handling in Relational and Semi-structured Databases*. PhD thesis, State University of New York at Buffalo, 2007.
18. S. Staworko and J. Chomicki. Validity-Sensitive Querying of XML Databases. In *EDBT Workshops (dataX)*, pages 164–177. Springer, 2006.

19. S. Staworko, E. Filiot, and J. Chomicki. Querying Regular Sets of XML Documents. In *International Workshop on Logic in Databases (LiD)*, 2008. Full version available at `http://www.grappa.univ-lille3.fr/~staworko/papers/lid08.pdf`.
20. J. W. Thatcher and Wrightm J. B. Generalized Finite Automata with an Application to a Decision Problem of Second-order Logic. *Mathematical System Theory*, 2:57–82, 1968.
21. A. Thomo, S. Venkatesh, and Y. Y. Ye. Visibly Pushdown Transducers for Approximate Validation of Streaming XML. In *Foundations of Information and Knowledge Systems (FOIKS)*, pages 219–238, 2008.
22. M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.

A Universal and existential query answering: omitted proofs

A.1 Proof of Theorem 1

We first prove two pumping lemmata.

For any tree t, we denote by $d(t)$ its depth (maximal length of a path from the root to a leaf) and by $w(t)$ its width (maximal number of children for any node).

Lemma 1. *Let $M = (\Sigma, \Gamma, I, F, Q, \Delta)$ be an attributed STA and t a tree such that $t \in L(M)$. There is a tree t^* such that:*

- *width and depth of t^* is bounded by $|Q|^2$;*
- *$t^* \in L(M)$*

Proof. Vertical pumping Suppose that $d(t) > |Q|^2$. Since $t \in L(M)$, there is a successful run r of M on the serialization $\overline{t}$ of t. Since there is a root-to-leaves path p of length strictly greater than $|Q|^2$, there is states q, q' such that q and q' correspond to well-matched opening and closing tags respectively in the subrun of r corresponding to p, and q, q' occurs twice in this nested structure.

More formally, there are words $u_1, \ldots, u_3, u'_1, \ldots, u'_2$ over $\Sigma_\Diamond$, e_i, e'_i opening and closing tags respectively, $i = 1, 2$, states $q, q' \in Q$, words of states $\alpha_1, \ldots, \alpha_3, \alpha'_1, \ldots, \alpha'_2$ such that:

- $\overline{t}$ is equal to $u_1 e_1 u_2 e_2 u_3 e'_2 u'_2 e'_1 u'_1$;
- e_i, e'_i are well-matched, $i = 1, 2$;
- $r = \alpha_1 q \alpha_2 q \alpha_3 q' \alpha'_2 q' \alpha'_1$;
- $q \alpha_2 q \alpha_3 q' \alpha'_2 q'$ is a run of M on $e_1 u_2 e_2 u_3 e'_2 u'_2 e'_1$;
- $q \alpha_3 q'$ is a run of M on $e_2 u_3 e'_2$;

Hence, $\alpha_1 q \alpha_3 q' \alpha'_1$ is a successful run of M on $u_1 e_2 u_3 e'_2 u'_1$. Therefore, as soon as there is a root-to-leaves path of length stricly greater than $|Q|^2$, we can compact it into a path of strictly shorter length and keep membership to $L(M)$. By iterating this process we get a tree t^* such that $d(t^*) \leq |Q|^2$ and $t^* \in L(M)$;

Horizontal pumping Suppose that $w(t) > |Q|^2$. The argument is very similar to vertical pumping, so that we do not formalize it. Intuively, if there is a sequence of children whose length is strictly greater than $|Q|^2$, in any successful runs r of M on $\bar{t}$, a pair of states q, q' corresponding in r to the opening and closing tags of a child will be repeated at least twice in the sequence of children. Hence we can collapse the sequence of children delimited by the two children where the repetition occurs, while keeping membership to $L(M)$.

Lemma 2. *Let $M = (\Sigma, \Gamma_M, I_M, F_M, Q_M, \Delta_M)$ be an STA and $\Phi = (\Sigma, \Gamma_\Phi, I_\Phi, F_\Phi, Q_\Phi, \Delta_\Phi$ be an STA query over variables $\{x_1, \ldots, x_n\}$. If there is some tree $t \in L(M)$ and some tuple $\tau \in QA(\Phi, t)$, then there is a tree t^* such that:*

- *depth and width of t^* are bounded by $n|Q_M|^2|Q_\Phi|^2$*
- *$t^* \in L(M)$*
- *$\tau \in QA(\Phi, t^*)$*

Proof. The arguments are very similar to the proof of Lemma **??**, except that when we collapse some contexts of the tree, we must care about keeping the membership of the tree both in $L(M)$ and $L(\Phi)$ (this gives us the bound $|Q_M|^2|Q_\Phi|^2$, as we consider the product automaton of M and Φ). And we also must care about keeping the tuple in the result, hence we cannot collapse contexts of the tree where an attribute has been selected (this gives us the factor n in the bound).

Proof of Theorem 1 We first prove correctness by induction. The length of a hedge is the number of its trees. After loop 4, $H_i[u, v]$ contains all tuples collected on hedges of length 1, depth $\leq i-1$, and width $\leq n|Q|^2$, by runs from u to v. The loop at line 6 performs a transitive closure, so that after loop at line 6, $H_i[u, v]$ contains all tuples collected on hedges of depth $\leq i - 1$ and width $\leq n|Q|^2$ by runs from u to v. Using H_i, $T_i[u, v]$ can be udpated as the union of all tuples collected on trees of depth $\leq i - 1$ and width $\leq n|Q|^2$ (represented by $T_{i-1}[u, v]$ by induction hypothesis) by runs from u to v, and the set of trees obtained by rooting hedges from H_i, with compatible transitions.

Hence for every tuple τ returned by the algorithm, there is a tree t such that $\tau \in QA^\exists(\Phi, t)$.

Soundness is proved by using **??**, and by definition of H_i and T_i.

A.2 Proof of Theorem 2

Since we reduce the universal tuple check problem to inclusion of STAs, we first prove the following:

Proposition 1. *Given a non-weighted attributed STA N and a weighted attributed STA M, testing whether $L(M) \subseteq L(N)$ can be done in time $O(|att(M)||M|2^{|att(M)||N|})$, where $att(M)$ is the set of attribute values from $\Lambda \cup \{*\}$ occuring in the transitions of M.*

Proof. First assume that M is non-weighted.

From M and N we construct to (non-attributed) STAs M' and N' such that $L(M) \subseteq L(N)$ iff $L(M') \subseteq L(N')$, $|M| = |M'|$, and $|N'| = |att(M)||N|$. Since (non-attributed) STAs are *Visibly Pushdown Automata* running on serialization of trees, we already know by [2] that inclusion is decidable in EXPTIME.

We let $M = (\Sigma, \Gamma_M, Q_M, I_M, F_M, \Delta_M)$ and $N = (\Sigma, \Gamma_N, Q_N, I_N, F_N, \Delta_N)$. We define M' by $(\Sigma', \Gamma'_M, Q'_M, I'_M, F'_M, \Delta'_M)$ where

– $\Sigma' = \Sigma \times att(M)$, $\Gamma'_M = \Gamma_M$, $Q'_M = Q_M$, $I'_M = I_M$, $F'_M = F_M$;
– $\Delta'_M = \{p \xrightarrow{\langle a,l\rangle:\gamma} q \mid p \xrightarrow[l]{\langle a\rangle:\gamma} q \in \Delta_M\} \cup \{p \xrightarrow{\langle /a,l\rangle:\gamma} q \mid \exists p' \xrightarrow[l]{\langle a\rangle:\gamma} q' \in \Delta_M \text{ and } p \xrightarrow{\langle /a\rangle:\gamma} q \in \Delta_M\}$

The STA $N' = (\Sigma', \Gamma'_N, Q'_N, I'_N, F'_N, \Delta'_N)$ is defined similarly, but in addition, for all transitions $p \xrightarrow[*]{\langle a\rangle:\gamma} q \in \Delta_N$, and all attribute values $l \in att(M)$, we add to Δ'_N the following transitions rules: $p \xrightarrow{\langle a,l\rangle:\gamma} q$ and the transitions $p' \xrightarrow{\langle /a,l\rangle:\gamma} q'$, for all $p' \xrightarrow{\langle /a\rangle:\gamma} q' \in \Delta_N$.

We now prove correctness of these constructions w.r.t. the inclusion problem. *forth direction* Suppose that $L(M) \subseteq L(N)$, and let $t \in L(M')$. We associate to t the set attributed trees $T(t)$ over Σ obtained by replacing any label (a, l) in t by the label a with attribute l, if $l \neq *$, or with any attribute from Λ if $l = *$. We can easily show that $T(t) \subseteq L(M)$. Note also that $T(t)$ is infinite iff t contains at least one node labeled $(a, *)$, for some $a \in \Sigma$. Moreover, if $T(t)$ is finite, then it is a singleton.

Suppose that $T(t)$ is finite, and equal to some singleton $\{t'\}$, where t' is an attributed tree over Σ. We can easily show that there is a successful sequence of transitions of M on $\overline{t'}$ which uses only transitions without $*$ is the (opening or closing) tag. Since $t' \in L(N)$, there is also a successful sequence s of transitions of N on $\overline{t'}$. If we put the attributes values of the transitions of s in the opening tag and in the corresponding closing tags, we get a sequence s' of transitions of N'. This sequence does not necessarily accepts t, because it might the case that s contain a transition with a wildcard. Therefore s' can contain a transition $\delta = p \xrightarrow{\langle a,*\rangle:\gamma}$ with a wildcard in the tag. We can substitute it with a transition $p \xrightarrow{\langle a,l\rangle:\gamma}$ where l is the attribute value of t at the position where δ is applied (this rule exists, since $l \in att(M)$ and by definition of N'). Hence we get a successful sequence of transitions of N' on $\overline{t}$, and $t \in L(N')$.

Now suppose that $T(t)$ is infinite. It means that t contains a wildcard. We only sketch the proof and emphasize the changes compared to the previous case, since it is technical but not difficult. We have to construct a successful sequence of transitions of N' on t. If we take an arbitrary tree $t' \in T(t)$, and use a successful sequence of transitions of N on it, it might be the case that M uses a transition with wildcard to accept $\overline{t'}$ at some position i of $\overline{t'}$, while N uses a transition without wildcard at the same position. In particular, this situation may occur when the attribute value of t' at position i is in $att(N)$. We can subsitute in t' this

value by a value which is not in $att(N)$ to get a tree $t'' \in L(M) \subseteq L(N)$, to ensure that N will use a transition with wildcard at position i when processing $\overline{t''}$. That way we can prove that there is a tree $t'' \in T(t)$ such that there is two successful sequences s_M and s_N of transitions of M and N on $\overline{t''}$ respectively, such that they use the same attribute values in their respective transitions. Moreover, we can choose t'' such that the sequence of tags and attributes used in s_M corresponds exactly to the labels used by the transitions of a successful sequence of transitions of M' on $\bar{t}$. By putting the attributes value of s_N in the tags (opening and respectively closing), we get a successful sequence of transitions of N' on $\bar{t}$. Hence $t \in L(N')$.
Back direction Suppose that $L(M') \subseteq L(N')$, and let $t \in L(M)$. Hence, there is a tree t' such that $t' \in L(M')$ and $t \in T(t')$. Hence $t' \in L(N')$, and by definition of N', we get $T(t') \subseteq L(N)$. Hence $t \in L(N)$.
Complexity We have $|M'| = O(|att(M)||M|)$, and $|N'| = 0(|att(M)||N|)$, so that inclusion of M' in N' is decidable in time $O(|att(M)|^3|M|^3 2^{3(|att(M)||N|)^2})$ [2].

Proposition 2. *Universal tuple check can be decided in time $O(f(|\Phi|)|M|^3))$ where $f(|\Phi|) = 2^{|\Phi|^2 2^{2n}}$.*

Proof. From M, Φ and τ we construct two STAs $M(\tau)$ and $\Phi(\tau)$ such that $M(\tau) \subseteq \Phi(\tau)$ iff $\tau \in\in QA^{\forall}(M, \Phi)$. Let $att(\tau)$ be the value from $\Lambda \cup \{*\}$ occuring in τ, and we denote by $\tau[X]$ the value of τ occuring at position x, for all $x \in X$ (it is undefined if the values are different). To construct $M(\tau)$ we replace every opening transition of the form $p \xrightarrow[l]{\langle a \rangle : \gamma}$ by $p \xrightarrow[*]{\langle a \rangle : \gamma}$ if $l \notin att(\tau)$. Hence $L(M) \subseteq L(M(\tau))$.

To construct $\Phi(\tau)$, we substitute set of variables X in selecting transitions by the value of the tuple at positions X (we simply remove the transition if there is multiple values in τ at positions X). We simulate selections by putting the sets of variables in the states, and in addition, we ensure that every variable is used exactly once in a successful run. More formally, let $\Phi = (\Sigma, \Gamma, Q, I, F, \Delta)$ over the set of variables $V = \{x_1, \ldots, x_n\}$. We define $\Phi(\tau)$ by $(\Sigma, \Gamma, Q \times 2^V, I \times \{\varnothing\}, F \times \{V\}, \Delta')$ where Δ' is defined as follows: $\forall p, q \in Q, X, Y \subseteq V, a \in \Sigma, \gamma \in \Gamma, l \in \Lambda \cup \{*\}$

- $(p, X) \xrightarrow[l]{\langle a \rangle : \gamma} (q, X \cup Y) \in \Delta'$ if $p \xrightarrow[Y]{\langle a \rangle : \gamma} q \in \Delta$, $X \cap Y = \varnothing$, and for all $y \in Y$, every value of τ at position y is equal to l. If $Y = \varnothing$, then l must be equal to $*$;
- $(p, X) \xrightarrow[l]{\langle /a \rangle : \gamma} (q, X) \in \Delta'$ if $p \xrightarrow{\langle /a \rangle : \gamma} q \in \Delta$.

We now prove that $\tau \in QA^{\forall}(M, \Phi)$ iff $M(\tau) \subseteq \Phi(\tau)$.

forth direction: assume that $\tau \in QA^{\forall}(M, \Phi)$, and let $\bar{t} \in L(M(\tau))$. Hence there is a successful run r of $M(\tau)$ on $\bar{t}$. By definition of $M(\tau)$, we can easily change the attributes of t to obtain a tree t' such that it has the same structure as t, the same labels at the same respective positions, and $\overline{t'} \in L(M)$. Indeed, it might the case that r uses a transition of $M(\tau)$ of the form $p \xrightarrow[*]{\langle a \rangle : \gamma} q$ at a position labeled $\langle a \rangle$ with attribute values l in $\bar{t}$. If this transition corresponds to a transition $p \xrightarrow[l']{\langle a \rangle : \gamma} q$ of M, where $l' \in \Lambda$ and $l \neq l'$. In this case we replace l by l' in t. Note that by definition of $M(\tau)$, $l' \notin att(\tau)$. Hence, we replace some of the attributes of t by attributes different from any attribute of $att(\tau) \cap \Lambda$. By assumption, we get $\tau \in QA(\Phi, t')$, meaning that there is a successful sequence of rules $s = \delta_1 \dots \delta_n$ of Φ applied on $\overline{t'}$ which selects the tuple τ (where n is the length of $\overline{t'}$). We now construct a successful sequence of rule applications $\delta'_1 \dots \delta'_n$ of $\Phi(\tau)$ on $\overline{t'}$, and then prove that it is also successful on $\bar{t}$. It is inductively defined by:

- $\delta'_1 = (p, \varnothing) \xrightarrow[l]{\langle a \rangle : \gamma} (q, X)$ if $\delta_1 = p \xrightarrow[X]{\langle a \rangle : \gamma} X$, where $l = \tau[X]$ if $X \neq \varnothing$, and $l = *$ otherwise;
- for $i > 1$, $\delta'_i = (p, X) \xrightarrow[l]{\langle a \rangle : \gamma} (q, X \cup Y)$ if (i) $\delta_i = p \xrightarrow[Y]{\langle a \rangle : \gamma}$, (ii) the rhs of δ'_{i-1} is p, X, where $l = \tau[Y]$ if $Y \neq \varnothing$, and $l = *$ otherwise;
- for $i > 1$, $\delta'_i = (p, X) \xrightarrow{\langle / a \rangle : \gamma} (q, X)$ if (i) $\delta_i = p \xrightarrow{\langle / a \rangle : \gamma}$, (ii) the rhs of δ'_{i-1} is p, X.

By definition of $\delta_1 \dots \delta_n$, we get a sequence $\delta'_1 \dots \delta'_n$ of rules which starts by an initial state of $\Phi(\tau)$ and ends in a final state of $\Phi(\tau)$. Hence $\overline{t'} \in L(\Phi(\tau))$. Now, remark that $\delta'_1 \dots \delta'_n$ is also a successful application of rules on $\bar{t}$, since $\Phi(\tau)$ only tests for attribute values in τ and the values of t' which are different from their respective values in t at the same position are necessarily different from values in $att(\tau)$. Hence we get $\bar{t} \in L(\Phi(\tau))$.

back direction: assume that $L(M(\tau)) \subseteq L(\Phi(\tau))$, and let $\bar{t} \in L(M)$. We prove that $\tau \in QA(\Phi, t)$. Since $L(M) \subseteq L(M(\tau)) \subseteq L(\Phi(\tau))$, we get $\bar{t} \in L(\Phi(tau))$. Hence there is a successful sequence $\delta_1 \dots \delta_n$ of rule applications of $N(\tau)$ on $\bar{t}$. We build a successful sequence $\delta'_1 \dots \delta'_n$ of rule application of Φ on $\bar{t}$, defined as follows: for all $i \in \{1, \dots, n\}$, $\delta'_i = p \xrightarrow[X]{\langle a \rangle : \gamma} q$ if $\delta_i = (p, Y) \xrightarrow[l]{\langle a \rangle : \gamma} (q, Y \cup X)$, for some $X, Y \subseteq V$ such that $X \cap Y = \varnothing$ (we necessarily have $\tau[X] = l$ by definition of $\Phi(\tau)$). We also define $\delta'_i = p \xrightarrow{\langle / a \rangle : \gamma} q$ if $\delta_i = (p, X) \xrightarrow[l]{\langle / a \rangle : \gamma} (q, X)$, for some $X \subseteq V$. Since $\Phi(\tau)$ controls that every variable is used added exactly once is the second component of its states, we get that $\delta'_1 \dots \delta'_n$ is a successful sequence of rules application of Φ on $\bar{t}$ which selects τ. Hence $\tau \in QA(\Phi, t)$.

Finally, by Proposition **??**, inclusion of $L(M(\tau))$ in $L(\Phi(\tau))$ is decidable in time $(O(|M(\tau)|.2^{|att(M(\tau))||\Phi(\tau)|}$, where $att(M(\tau))$ is the set of attribute values from Λ used in $M(\tau)$. Since $|\Phi(\tau)| = 2^n|\Phi|$, and $att(M(\tau)) \leq |\tau| = n$, inclusion is decidable in time $O(|M(\tau)|^3 2^{3n^2|\Phi|^2 2^{2n}})$, ie $O(|M|^3 2^{|\Phi|^2 2^{2n}})$.

A.3 Proof of Theorem 3

The proof is divided into several propositions.

Proposition 3. *Existential tuple check is NP-hard.*

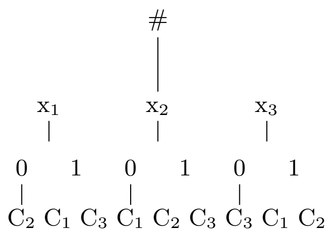


Fig. 5. Tree associated with the formula $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \wedge x_2 \wedge \neg x_3)$

Proof. We reduce the 3-SAT problem. Let $\Phi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_k$ be a formula in DNF with variables $x_1, \ldots, x_n$. We associate Φ with a tree t_Φ over the alphabet $\Sigma = \{\#, x_1, \ldots, x_n, C_1, \ldots, C_k, 0, 1\}$. The tree t_Φ is defined by $\#(x_1(t^0_{x_1}, t^1_{x_1}), \ldots, x_n(t^0_{x_n}, t^1_{x_n}))$, where for all $i \in \{1, \ldots, n\}$ and all $b \in \{0, 1\}$, $t^b_{x_i}$ is defined by: $t^b_{x_i} = b(C_{i_1}, \ldots, C_{i_j})$

where $\{C_{i_1}, \ldots, C_{i_j}\}$ is the set of clauses (given in order) which are true under the assignement $x_i \mapsto b$. Figure **??** shows an example of this association.

We can easily view t_Φ as a stream $\overline{t_\Phi}$ whose labels are both considered as attributes and labels of the stream. We can construct an attributed STA M_ϕ such that its size is in $O(|t_\Phi|)$ and $L(M) = \{\overline{t_\Phi}\}$.

Now, we construct a querying attributed STA N such that $(C_1, \ldots, C_k) \in QA^\exists(M, N)$ iff Φ is satisfiable. We use k variables $y_1, \ldots, y_k$ intended to select the attribute values $C_1, \ldots, C_k$. The following must be satisfied: if N selects some value in the 0-part of some variable x_i, then it cannot select any other value in the 1-part of x_i. This control can be done by the states of N. We use three states q, q_s, q'_s: the first state allows to navigate in the tree, and, if it used in a subtree rooted by a variable, it means that no attribute value has been selected. The state q_s is used in the subtrees rooted by a variable x, and means that something has been selected in the 0 part of them. Finally, q'_s is used in the subtrees rooted by 1 if something has been selected in the previous sibling tree rooted by 0 (hence nothing should be selected below 1). We use the labels as stack alphabet, q is both initial and final, and for all $i \in \{1, \ldots, n\}$, and for all $j \in \{1, \dots, k\}$, the set of rules is defined by:

$$q \xrightarrow{\langle \# \rangle : \#} q \quad q \xrightarrow{\langle / \# \rangle : \#} q \quad q \xrightarrow{\langle x_i \rangle : x_i} q \quad q \xrightarrow{\langle / x_i \rangle : x_i} q$$

$$q \xrightarrow{\langle 0 \rangle : 0} q \quad q \xrightarrow{\langle / 0 \rangle : 0} q \quad q_s \xrightarrow{\langle / 0 \rangle : 0} q_s$$

$$q \xrightarrow{\langle 1 \rangle : 1} q \quad q_s \xrightarrow{\langle 1 \rangle : 1} q'_s \quad q'_s \xrightarrow{\langle / 1 \rangle : 1} q \quad q \xrightarrow{\langle / 1 \rangle : 1} q \quad q_s \xrightarrow{\langle / 1 \rangle : 1} q$$

$$q \xrightarrow{\langle C_j \rangle : C_j} q \quad q \xrightarrow{\langle / C_j \rangle : C_j} q \quad q \xrightarrow[y_j]{\langle C_j \rangle : C_j} q_s \; q_s \xrightarrow[y_j]{\langle C_j \rangle : C_j} q_s \; q_s \xrightarrow{\langle / C_j \rangle : C_j} q_s$$
$$q'_s \xrightarrow{\langle C_j \rangle : C_j} q'_s \; q'_s \xrightarrow{\langle / C_j \rangle : C_j} q'_s$$

Proposition 4. *Existential tuple check is in NP.*

Proof. We can transform Algorithm 1 into a probabilistic algorithm which outputs an answer sets of size at most $|Q|^2$, such that $\tau \in QA^{\exists}(M, N)$ iff there is an execution which outputs a set to which τ belongs. It suffices to choose which sets are merged, and which transitions are used for the $assign_X$ in order to store at most one tuple in $T_i[u, v]$ and $H_i[u, v]$, for all i, u, v.

Proposition 5. *Universal tuple check is EXPTIME-hard.*

Proof. Note that $\mathcal{TC}^{\forall}$ for 0-ary STA queries is equivalent to testing inclusion of two STA languages. This is kwnow to be EXPTIME-complete for (non-deterministic) top-down tree automata (even binary). We give a polynomial-time translation of tree automata into STA, which will be sufficient to conclude. We start from a ranked alphabet Σ consisting of binary and constant symbols. We let $A = (\Sigma, Q, I, \Delta)$ be a top-down tree automaton. We associate with A the STA $\overline{A} = (\overline{\Sigma}, \overline{Q}, \overline{\Gamma}, \overline{I}, \overline{F}, \overline{\Delta})$, where $\overline{\Sigma} = \{\langle f \rangle \mid f \in \Sigma\} \cup \{\langle / f \rangle \mid f \in \Sigma\}$; $\overline{Q} = Q \cup Q \times Q \cup \{q_{end}\}$; $\overline{Gamma} = Q$; $\overline{I} = I$; $\overline{F} = Q$, and $\overline{\Delta}$ is defined by:

$$\begin{array}{ll}
q \xrightarrow{\langle f \rangle : q} (q_1, q_2) & \text{for all rules } q \to f(q_1, q_2) \in \Delta \\
(q_1, q_2) \xrightarrow{\langle f \rangle : q_2} (q'_1, q'_2) & \text{for all rules } q_1 \to f(q'_1, q'_2) \in \Delta \\
q \xrightarrow{\langle a \rangle : q} q_{end} & \text{for all rules } q \to a \in \Delta \\
(q_1, q_2) \xrightarrow{\langle a \rangle : q_2} q_{end} & \text{for all } q_1, q_2 \in Q \\
q_{end} \xrightarrow{\langle / a \rangle : q} q & \text{for all } q \in Q \\
q \xrightarrow{\langle / f \rangle : p} p & \text{for all } q, p \in Q
\end{array}$$

Let $\overline{t}$ denotes the stream encoding over $\overline{\Sigma}$ of any tree over Σ. We can prove the following: for all trees t, $t \in L(A)$ iff $\overline{t} \in L(\overline{A})$. Finally note that the translation is in PTIME.

Proposition 6. *Universal tuple-check is in EXPTIME.*

Proof. We construct a deterministic attributed STA $\Phi[\tau]$ such that $L(\Phi[\tau]) = \{t \mid \tau \in QA(\Phi, t)\}$. Then we test if $L(M) \subseteq L(\Phi[\tau])$, which is equivalent to test if $\tau \in QA^{\forall}(\Phi, M)$.

For all $i \in \{1, \dots, n\}$, we let τ_i be the value of τ at rank i, and we let $A = \{\tau_i \mid i = 1, \dots, n\}$. We first construct an attributed STA Φ' over the set of attribute values Λ extended with fresh values denoted (l, V), for all set of variables V labeling an opening transition of Φ, and such that $\tau(X) = l$ for all $X \in V$. Let us denote Λ' this new alphabet. Φ' is obtained first by:(i) removing from Φ every selecting transition (by a set of variables V) such that $\exists X, Y \in V$, $\tau(X) \neq \tau(Y)$; (ii) replacing every selecting transition of the form $q \xrightarrow[V]{\langle a\rangle:\gamma} p$ such that $\forall X, Y \in V, \tau(X) = \tau(Y) = l$ for some $l \in A$, by the attributed transition $q \xrightarrow[(l,V)]{\langle a\rangle:\gamma} p$. Hence, Φ' accepts attributed trees some of their attribute values denoting positions where the attribute might be selected. Then we determinize Φ'.

In a second step, we obtain $\Phi[\tau]$ (over Σ and Λ) from Φ' (over Σ and Λ') by adding a control that every component of τ is selected. Let $Q_{\Phi'}, F_{\Phi'}$ be the set of states (accepting states resp.) of Φ'. The set of states $Q_{\Phi[\tau]}$ is defined by $Q_{\Phi'} \times 2^{\{X_1,\dots,X_n\}}$, and $F_{\Phi[\tau]}$ by $F_{\Phi'} \times \{X_1, \dots, X_n\}$. The set of rules $\Delta_{\Phi[\tau]}$ is defined by:

$$\begin{array}{ll}(q,V) \xrightarrow{e:\gamma} (p,V) & \forall q \xrightarrow{e:\gamma} p \in \Delta_{\Phi'},\ \forall V \subseteq \{X_1,\dots,X_n\} \\ (q,V) \xrightarrow[l]{\langle a\rangle:\gamma} (p, V \cup V') & \forall q \xrightarrow[(l,V')]{\langle a\rangle:\gamma} p \in \Delta_{\Phi'}, \\ & \forall V \subseteq \{X_1,\dots,X_n\}\ st\ V \cap V' = \varnothing\end{array}$$

Note that $\Phi[\tau]$ is still deterministic, and its size is $|\Phi[\tau]| = O(2^n|\Phi'|) = O(2^n 2^{|\Phi|})$. Hence, testing whether $L(M) \subseteq L(\Phi[\tau])$ can be done in time $O(|M| 2^n 2^{|\Phi|})$.

B Consistent query answering: omitted proofs and definitions

An *tag* is an element of $\Sigma_\Diamond = \Sigma_\langle \cup \Sigma_\rangle$, where $\Sigma_\langle = \{\langle a\rangle | a \in \Sigma\}$ is the set of opening tags and $\Sigma_\rangle = \{\langle /a\rangle | a \in \Sigma\}$ is the set of closing tags. An *attributed* tag is an element of $T = \Sigma_\langle \times \Lambda \cup \Sigma_\rangle$. As before for an attributed tag e by $tag(e)$ we denote its tag and if it is an opening tag, we denote by $att(e)$ its attribute value. When it does not cause confusion we omit the adjective attributed. A pair of tags e_1 and e_2 is *matching* if $tag(e_1) = \langle a\rangle$ and $tag(e_n) = \langle /a\rangle$ for some $a \in \Sigma$.

Definition 4 (Well-formed tag sequence). Ballanced tag sequence *(b.t.s) is defined as follows:*

- *ε is a b.t.s,*
- *$e_1 \cdot \bar{t} e_2$ is a b.t.s if $\bar{t}$ is a b.t.s and e_1 and e_2 are a pair of matching tags.*
- *$\bar{t}_1 \cdot \bar{t}_2$ is a b.t.s if $\bar{t}_1$ is b.t.s. and $\bar{t}_2$ is b.t.s.*

$\bar{t}$ is a well-formed *sequence of tags if $\bar{t}$ is empty or $\bar{t}$ is b.t.s. and starts with an opening tag.*

Definition 5 (General STA). *A* general streaming tree automaton *(GSTA) with is a tuple $M = (\Sigma, \Lambda, \Gamma_M, Q_M, I_M, \Delta_M, F_M, w_M)$, where Σ is a finite set of tree symbols, Λ is an infinite set of attribute values, Γ_M is a finite set of stack symbols, Q_M is a finite set of states, $I_M \subseteq Q_M$ is a set of initial states, $F_M \subseteq Q_M$ is a set of final states, $w_M : \Delta_M \to \mathbb{R}_+ \cup 0$ is a weight function, and Δ_M is a finite set of the following types of transitions:*

– opening *transitions $p \xrightarrow[L]{\langle a \rangle : \gamma} q$, where $p, q \in Q_M$, $a \in \Sigma$, L is a finite or cofinite subset of Λ, $\gamma \in \Gamma_M$;*
– closing *transitions $p \xrightarrow{\langle /a \rangle : \gamma} q$, where $p, q \in Q_M$, $a \in \Sigma$, and $\gamma \in \Gamma_M$;*
– *ε-transitions $p \xrightarrow{\varepsilon} q$, where $p, q \in Q_M$.*

Definition 6 (Configuration and move relation of a GSTA). *A* configuration *of a GSTA M is a element $(q, \bar{\gamma}, \bar{t})$ of the set $C_M = Q_M \times \Gamma_M^* \times T^*$, where q is the current automaton state, $\bar{\gamma}$ is the current state of the stack, and $\bar{t}$ is the remaining tag sequence. The* move *relation $\to_M \subseteq C_m \times C_M$ of a GSTA M is defined as follows:*

– *$(p, \bar{\alpha}, e \cdot \bar{t}) \to_M (q, \gamma \cdot \bar{\alpha}, \bar{t})$, if $tag(e) = \langle a \rangle$ and there exists $p \xrightarrow[L]{\langle a \rangle : \gamma} q \in \Delta_M$ such that $att(e) \in L$,*
– *$(p, \gamma \cdot \bar{\alpha}, e \cdot \bar{t}) \to_M (q, \bar{\alpha})$ if $tag(e) = cta$ and there exists $p \xrightarrow{\langle /a \rangle : \gamma} q \in \Delta_M$,*
– *$(p, \bar{\alpha}, \bar{t}) \to_M (q, \bar{\alpha}, \bar{t})$ if there exists $q \xrightarrow{\varepsilon} p \in \Delta_M$.*

$\to_M^* \subseteq C_M \times C_M$ is the reflexive and transitive closure of $\to_M$.

Definition 7 (Run of a GSTA). *A* run *of a GSTA M on a tag sequence $\bar{t}$ is a seqence $\bar{r} = q_0, d_1, q_1, \ldots, d_n, q_n$ of elements taken alternatingly from Q_N and Δ such that $\bar{t}$ can be decomposed into a sequence $s_1, \ldots, s_n$ of elements of $T \cup \{\varepsilon\}$ and for every $i \in \{1, \ldots, n\}$ one of the following holds:*

– *$d_i = q_{i-1} \xrightarrow[L]{\langle a \rangle : \gamma} q_i \in \Delta_M$, $s_i \in T$, $tag(s_i) = \langle a \rangle$, and $att(s_i) \in L$,*
– *$d_i = q_{i-1} \xrightarrow{\langle /a \rangle : \gamma} q_i \in \Delta_M$, $s_i \in T$, and $tag(e_i) = \langle /a \rangle$,*
– *$q_{i-1} \xrightarrow{\varepsilon} q_i \in \Delta_M$ and $s_i = \varepsilon$.*

Usually, we will write the run $\bar{r} = q_0, d_1, q_1, \ldots, d_n, q_n$ as $q_0 \to \cdots \to q_n$ with proper labels on the arrows. With a simple inductive prove we can show equvalence of a run and $\to_M^*$ over ballanced tag sequences.

Proposition 7. *For any b.s.t $\bar{t}$, any $p, q \in Q$, and any $\alpha \in \Gamma^*$, $(p, \bar{\alpha}, \bar{t}) \to_M^* (q, \bar{\alpha}, \varepsilon)$ if and only if there exits a run $\bar{r} = p \to \ldots \to q$ such that $\pi_{tag}(\bar{r}) = \bar{t}$.*

Also, with a simple induction it can be shown that a run on a ballanced tag sequence shares the structure of the tag sequence.

Proposition 8 (Structure of a run). *A run of M on a ballanced tag sequence is one of the following:*

(i) ε *is a run on* ε *(the empty sequence of tags),*

(ii) $q \xrightarrow{\varepsilon} p$ *is a run on* ε*,*

(iii) if $q_0 \xrightarrow[L]{\langle a\rangle:\gamma} q_1$*,* $q_{n-1} \xrightarrow{\langle /a\rangle:\gamma} q_n$*, and* $q_1 \to \cdots \to q_{n-1}$ *is a run on* $\bar{t}$*, then* $q_o \xrightarrow{\langle a\rangle:\gamma} q_1 \to \cdots \to q_{n-1} \xrightarrow{\langle /a\rangle:\gamma} q_n$ *is a run on any* $e_1 \cdot \bar{t} \cdot e_n$ *such that* $tag(e_1) = \langle a\rangle$*,* $att(e_1) \in L$*, and* $tag(e_n) = \langle /a\rangle$*.*

(iv) if $q_1 \to \cdots \to q_k$ *is a run on* $\bar{t}_1$ *and* $q_k \to \cdots \to q_n$ *is a run on* $\bar{t}_2$*, then* $q_1 \to \cdots \to q_k \to \cdots \to q_n$ *is a run on* $\bar{t}_1 \cdot \bar{t}_2$*.*

Definition 8 (Weight of a run). *Givena a run* $\bar{r} = q_0, d_1, q_1 \ldots, q_{n-1}, d_n, q_n$ *of* M*, its* weight *is defined as* $w_M(\bar{r}) = \sum_{i=1}^{n} w_M(d_i)$*.*

Proposition 9 (Optimal substructure of a minimal run). *The minimal weight of any run from* p *to* q *on a b.s.t.* $\bar{t}$*, denoted by* $w_M^{min}(p, \bar{t}, q)$*, is the minimum of the following values:*

(i) 0 *if* $p = q$ *and* $\bar{t} = \varepsilon$*,*

(ii) $w_M(p \xrightarrow{\varepsilon} q)$ *if* $\bar{t} = \varepsilon$*,*

(iii) $w_M(p \xrightarrow[L]{\langle a\rangle:\gamma} p') + w_M^{min}(p', \bar{t}', q') + w_M(p \xrightarrow{\langle a\rangle:\gamma} p')$ *for any* $p \xrightarrow[L]{\langle a\rangle:\gamma} p', p \xrightarrow{\langle a\rangle:\gamma} p' \in \Delta_M$ *such that* $\bar{t} = e_1 \cdot \bar{t}' \cdot e_n$*,* $tag(e_1) = \langle a\rangle$*,* $att(e_1) \in L$*, and* $tag(e_n) = \langle /a\rangle$*,*

(iv) $w_M^{min}(p, \bar{t}_1, s) + w_M^{min}(s, \bar{t}_2, q)$ *for any* $s \in Q$ *and any* $\bar{t}_1$ *and* $\bar{t}_2$ *such that* $\bar{t} = \bar{t}_1 \cdot \bar{t}_2$*.*

B.1 Repair graph

Two sequences of operations are *equivalent* on a tree t if their application to t yields the same tree. We observe that some sequences may perform redundant operations, for instance an insertion and subsequent removal of a node. Because we focus on finding cheapest sequences of operations, we restrict our considerations to *redundancy-free* sequences (those for which there is no equivalent but cheaper sequence).

Since we are working with the streamed representation of documents, we port the tree editing operation to this medium. We consider sequences of *stream editing operations* which specify what action should be taked with regard to *every* tag. We use: D for *deleting* a tag, M_e for *modifying* a tag to e (together with an attribute if e is an opening tag), and I_e for *inserting* e. Finally, to indicate tags that are not modified we introduce the *reading* (or *doing-nothing*) operation R. The result of applying a sequence of stream editing operations to a tag sequence is defined as follows:

$$\begin{aligned} apply(\varepsilon, \varepsilon) &= \varepsilon, \\ apply(R \cdot \bar{s}, e \cdot \bar{t}) &= e \cdot apply(\bar{s}, \bar{t}), \\ apply(M_{e_1} \cdot \bar{s}, e_2 \cdot \bar{t}) &= e_1 \cdot apply(\bar{s}, \bar{t}), \\ apply(D \cdot \bar{s}, e \cdot \bar{t}) &= apply(\bar{s}, \bar{t}), \\ apply(I_e \cdot \bar{r}, \bar{s}) &= e \cdot apply(\bar{r}, \bar{s}). \end{aligned}$$

Obviously, not for every pair the operation *apply* is defined or returns a b.s.t. Therefore, we explicitly define editing scripts that can be applied to a given tag sequence.

Definition 9 (Editing script). *A* balanced editing sequence *(b.e.s.) of a balanced tag sequence (b.t.s.) is defined as follows:*

1. *ε is a b.e.s. of ε.*
2. *D is a b.e.s. of a sequence consisting on one tag e.*
3. *for any b.e.s. $\bar{s}$ of $\bar{t}$:*
   (a) *$I_{e_1} \cdot \bar{s} \cdot I_{e_2}$ is a b.e.s. of $\bar{t}$ for any pair of matching tags e_1 and e_2;*
   (b) *$R \cdot \bar{s} \cdot R$ is a b.e.s. of $e_1 \cdot \bar{t} \cdot e_2$ for any pair of matching tags e_1 and e_2;*
   (c) *$M_{e_1} \cdot \bar{s} \cdot M_{e_2}$ is a b.e.s. of $e'_1 \cdot \bar{t} \cdot e'_2$ for two pairs (e_1, e_2) and (e'_1, e'_2) of mathcing tags.*
4. *$\bar{s}_1 \cdot \bar{s}_2$ is b.e.s. of $\bar{t}_1 \cdot \bar{t}_2$ for any $\bar{s}_1$ and $\bar{s}_2$ b.e.s. of $\bar{t}_1$ and $\bar{t}_2$ respectively.*

A b.e.s. $\bar{s}$ of $\bar{t}$ is an editing script *of the well-formed tag sequence $\bar{t}$ if the first and last element of $\bar{r}$ are either both reading or both changing (root node cannot be deleted nor inserted).*

We assume that the cost of operations on node are fixed: c_D for deleting a node, c_I for inserting a node, c_M for modifying a node. Because a node operation is represented by two operations on the sequence of tags, we split the value equally: $c_D/2$ for a tag deleting operation, $c_I/2$ for a tag inserting operation, and $c_M/2$ for a tag modifying operation. The cost of an editing script $\bar{s}$ is denoted by $c(\bar{s})$.

Lemma 3. *For any editing script $\bar{r}$ of a tag sequence $\bar{s}$ $apply(\bar{r}, \bar{s})$ is a tag sequence.*

Proof. We first note that with a simple induction we can prove that

$$apply(\varepsilon, \varepsilon) = \varepsilon,$$

for any one elment tag sequence e

$$apply(D, e) = \varepsilon,$$

for any b.t.s. $\bar{t}$, any b.e.s. $\bar{s}$ of $\bar{t}$, and any pair of matching tags (e_1, e_2)

$$apply(I_{e_1} \cdot \bar{s} \cdot I_{e_2}, \bar{t}) = e_1 \cdot apply(\bar{s}, \bar{t}) \cdot e_2,$$

for any b.t.s. $\bar{t}$, any b.e.s. $\bar{s}$ of $\bar{t}$, and any pairs of matching tags (e_1, e_2) and (e'_1, e'_2)

$$apply(M_{e_1} \cdot \bar{s} \cdot M_{e_2}, e'_1 \cdot \bar{t} \cdot e'_2) = e_1 \cdot apply(\bar{s}, \bar{t}) \cdot e_2,$$

and for any b.t.s. $\bar{t}_1$ and $\bar{t}_2$ and any b.e.s. $\bar{s}_1$ and $\bar{s}_2$ of $\bar{t}_1$ and $\bar{t}_2$ resp.

$$apply(\bar{s}_1 \cdot \bar{s}_2, \bar{t}_1 \cdot \bar{t}_2) = apply(\bar{s}_1, \bar{t}_1) \cdot apply(\bar{s}_2, \bar{t}_2).$$

What follows is that for any b.t.s. $\bar{t}$ and any b.e.s. $\bar{s}$ of $\bar{t}$ the result of $apply(\bar{s}, \bar{t})$ is a b.t.s. Finally, we note that when the first and last element of an $\bar{s}$ of a well-formed tag sequence $\bar{t}$ are reading of modifying, then $apply(\bar{s}, \bar{t})$ is also a well-formed tag sequence.

A seuquence of tree editing operations on a tree t is *equivalent* to an editing script on $\bar{t}$ if and only if the result of applying the editing script to $\bar{t}$ is a streamed representation of the result of applying the sequence of tree editing operations on a tree t.

Lemma 4. *For any tree and every sequence of editing operations on the tree there exists an equivalent editing script of the same cost, and vice versa.*

Proof. $\boxed{\Leftarrow}$ For any tree t and any editing script $\bar{r}$ on $\bar{t}$ we construct the equivalent sequence of editing operations on t by recursion over the strucutre of $\bar{s}$ and the tag sequence $\bar{t} = (e_1, \ldots, e_n)$ (along the lines of Definition **??**):

- $op(\varepsilon, \varepsilon) = \varepsilon$.
- $op(D, e_i) = \varepsilon$ if e_i is a closing tag,
- $op(D, e_i)$ when e_i is an opening tag: Let n be the node of T corresponding to s_i. The result is an operation of deleting n.
- $op(R \cdot \bar{s}' \cdot R, e_i \cdot \bar{t}' \cdot e_j) = op(\bar{s}', \bar{t}')$.
- $op(M_{e_1} \cdot \bar{s}' \cdot M_{e_2}, e_i \cdot \bar{t}' \cdot e_j)$: Let n be the node of T corresponding to $e_i \cdot \bar{t}' \cdot e_j$. The result is an operation of changing the label of n to $tag(e_1)$ and the attribute value to $att(e_1)$ followed by $op(\bar{s}', \bar{t}')$.
- $op(I_{e_1} \cdot \bar{s}' \cdot I_{e_2}, \bar{t}')$: Let n be the identifier of the inserted node, $n_1, \ldots, n_l$ be the sequence of consecutive nodes of t corresponding to the $apply(\bar{s}', \bar{t}')$ and p be the parent of $n_1, \ldots, n_l$. The result is $op(\bar{r}', \bar{s})$ followed by an operation of inserting the node n labeled with $tag(e_1)$ and attribute $att(e_1)$ as a child of p adopting children $n_1, \ldots, n_l$.
- $op(\bar{s}_1 \cdot \bar{s}_2, \bar{t}_1 \cdot \bar{t}_2) = op(\bar{s}_1, \bar{t}_1) \cdot op(\bar{s}_1, \bar{t}_1)$ if $\bar{t}_1$ and $\bar{t}_2$ are b.t.s. and $\bar{s}_1$ and $\bar{s}_2$ are b.e.s. of $\bar{t}_1$ and $\bar{t}_2$ respecitvely.

It is easy to see that $op(\bar{s}, \bar{t})$ returns a sequence of tree editing operations equivalent to $\bar{s}$ and $c(op(\bar{s}, \bar{t})) = c(\bar{s})$.

$\boxed{\Rightarrow}$ For a given tree and a sequence of editing operations we construct the equivalent editing script by translating every editing operation into an equivalent editing script and consequently composing all editing scripts into one.

Given a b.t.s $\bar{t}$, a *void* script $\bar{\emptyset}_{\bar{t}}$ on $\bar{t}$ is an editing script on $\bar{t}$ whose all elements are R. For a given editing operation α on T we construct the equivalent *atomic* editing script $\bar{s}_\alpha$ on $\bar{t}$ as follows:

1. if α is an operation *Modifying* the node n by changing the symbol to a and reassigning the attribute to l, then all elements of $\bar{r}_\alpha$ are R except for $r_i = C_{e_1}$ and $r_j = C_{e_2}$, where i and j are the positions of respectively the opening and closing tag corresponding to the node n, $tag(e_1) = \langle a \rangle$, $att(e_1) = l$, and $tag(e_2) = \langle /a \rangle$;

2. if α is an operation *Deleting* a node n, then all elements of $\bar{r}_\alpha$ are R except for $r_i = r_j = D$, where i and j are the positions of respectively the opening and closing tag corresponding to the node n;
3. if α is an operation *Inserting* a node n with label a and attribute value l as a child of p and "adopting" a subsequence $n_1, \ldots, n_l$ of children of p, then all elements of $\bar{r}_\alpha$ are R except for $r_{i-1} = I_{e_1}$ and $r_{j+1} = I_{e_2}$, where i is position of the opening tag corresponding to n_1, j is the position of the closing tag corresponding to n_l, $tag(e_1) = \langle a \rangle$, $att(e_1) = l$, and $tag(e_2) = \langle /a \rangle$.

Next, we define the composition of two b.e.s. Given any b.t.s. $\bar{t}$, any b.e.s. $\bar{s}_1$ on $\bar{t}$, and any b.e.s. $\bar{s}_2$ on $apply(\bar{s}_1, \bar{t})$, the *composition* of $\bar{s}_1$ and $\bar{s}_2$, denoted $\bar{r}_1 \parallel \bar{r}_2$ is defined as follows:

$$
\begin{aligned}
\varepsilon \parallel \varepsilon &= \varepsilon, \\
(D \cdot \bar{x}) \parallel \bar{y} &= D \cdot (\bar{x} \parallel \bar{y}), \\
\bar{x} \parallel (I_e \cdot \bar{y}) &= I_e \cdot (\bar{x} \parallel \bar{y}), \\
(R \cdot \bar{x}) \parallel (R \cdot \bar{y}) &= R \cdot (\bar{x} \parallel \bar{y}), \\
(R \cdot \bar{x}) \parallel (D \cdot \bar{y}) &= D \cdot (\bar{x} \parallel \bar{y}), \\
(R \cdot \bar{x}) \parallel (C_e \cdot \bar{y}) &= C_e \cdot (\bar{x} \parallel \bar{y}), \\
(C_e \cdot \bar{x}) \parallel (R \cdot \bar{y}) &= C_e \cdot (\bar{x} \parallel \bar{y}), \\
(C_e \cdot \bar{x}) \parallel (D \cdot \bar{y}) &= D \cdot (\bar{x} \parallel \bar{y}), \\
(C_{e_1} \cdot \bar{x}) \parallel (C_{e_2} \cdot \bar{y}) &= (C_{e_2} \cdot \bar{x}) \parallel \bar{y}, \\
(I_e \cdot \bar{x}) \parallel (R \cdot \bar{y}) &= I_e \cdot (\bar{x} \parallel \bar{y}), \\
(I_e \cdot \bar{x}) \parallel (D \cdot \bar{y}) &= \bar{x} \parallel \bar{y}, \\
(I_{e_1} \cdot \bar{x}) \parallel (C_{e_2} \cdot \bar{y}) &= (I_{e_2} \cdot \bar{x}) \parallel \bar{y}.
\end{aligned}
$$

It is clear from the definition of $\parallel$ that

$$apply(\bar{s}_1, apply(\bar{s}_2, \bar{t})) = apply(\bar{s}_1 \parallel \bar{s}_2, \bar{t}).$$

Also, it should be clear that $c(bars_1 \parallel \bar{s}_2) \leq c(\bar{s}_1) + c(\bar{s}_2)$.

Now, if we take any tree t and any sequence of editing operations $\bar{\alpha} = (\alpha_1, \ldots, \alpha_m)$ on t, the equivalent editing script on $\bar{t}$ is $\bar{s}_{\bar{\alpha}} = (\cdots((\bar{\emptyset}_{\bar{t}} \parallel \bar{s}_{\alpha_1}) \parallel \bar{s}_{\alpha_2}) \cdots) \parallel \bar{s}_{\alpha_m}$. Naturally, $c(\bar{s}_{\bar{\alpha}}) \leq c(\bar{\alpha})$. Also, $c(\bar{s}_{\bar{\alpha}}) = c(\bar{\alpha})$ or $alpha$ is not redundancy-free, i.e. $op(\bar{s}_{\bar{\alpha}}, \bar{t})$ is an equivalent sequence of editing oeprations with lesser cost.

Definition 10. *An editing script $\bar{s}$ is* repairing *t w.r.t. M if $apply(\bar{s}, \bar{t} = \bar{t}'$ and $t' \in L(M)$.*

Proposition 10. *For any repairng script $\bar{s}$ of t w.r.t M resulting in t' there exists an accepting run $\bar{r}$ of $R(t, M)$ on t' such that $c(\bar{s}) = w_{R(t,M)}(\bar{r})$, and vice versa.*

Proof. Since $\bar{s}$ is a repairing script of t w.r.t. M, there exists an accepting run $\bar{r}'$ of M on t', where $\bar{t}' = apply(\bar{s}, \bar{t})$. We construct the accepting run $\bar{r}$ of $R(t, M)$

on t' with a simple recursion on the structure of $\bar{s}$ and $\bar{t}'$. The opposite direction is handled similarily, we take the run $\bar{r}$ of $R(t, M)$ on $\bar{t}'$ and construct the script $\bar{s}$ by recursion on the structure of $\bar{r}$ on $\bar{t}'$. The details of those construction are very technical and we omit them.

Theorem 6. *The combined complexity of consistent query answers is EXPTIME-complete. If $w_I > 0$, then the problem is coNP-complete.*

Proof. We reduce the containment of tree automata $M \subseteq N$ to checking if an empty tuple () is a consistent answer to a 0-ary query N in an empty tree ε w.r.t. M with $w_I = 0$. Naturally, $L(R(\varepsilon, M)) = L(M)$. From the proof of EXPTIME-hardness of universal tuple check we have that $() \in QA^{\forall}(N, R(\varepsilon, M))$ if and only if $L(M) \subseteq L(N)$. Membership to EXPTIME follows from Theorem 3.