Bounded repairability for regular tree languages

Gabriele Puppis
Department of Computer Science
University of Oxford, UK
gabriele.puppis@cs.ox.ac.uk

Cristian Riveros
Department of Computer Science
University of Oxford, UK
cristian.riveros@cs.ox.ac.uk

Sławek Staworko
Mostrare, INRIA & LIFL
(CNRS UMR8022)
University of Lille, France
slawomir.staworko@inria.fr

ABSTRACT

We consider the problem of repairing unranked trees (e.g., XML documents) satisfying a given restriction specification R (e.g., a DTD) into unranked trees satisfying a given target specification T. Specifically, we focus on the question of whether one can get from any tree in a regular language R to some tree in another regular language T with a finite, uniformly bounded, number of edit operations (i.e., deletions and insertions of nodes). We give effective characterizations of the pairs of specifications R and T for which such a uniform bound exists, and we study the complexity of the problem under different representations of the regular tree languages (e.g., non-deterministic stepwise automata, deterministic stepwise automata, DTDs). Finally, we point out some connections with the analogous problem for regular languages of words, which was previously studied in [6].

Categories and Subject Descriptors

H.2.m [**Information Systems**]: Database Management—*Repair*

Keywords

XML, edit distance, repair, curry encoding.

1. INTRODUCTION

When a database does not satisfy integrity constraints, a natural operation to perform is to *repair* it – modify it minimally so that the constraints are satisfied. This approach to data integrity has been investigated in the relational case for a variety of integrity constraints – beginning with classical functional and inclusion dependencies [4], and continuing to wider classes such as tuple generating dependencies [1]. A number of different modification operators have been considered in the relational case, including inserting, changing, and deleting tuples. In addition to finding repairs of documents, much of this line of research deals with querying inconsistent documents via their repairs.

For XML documents, repair has been studied less extensively. In [15], repair is considered for an extension of classical relational constraints such as inclusion dependencies, while in [22] repairs of a document with respect to an XML schema are studied, with an emphasis on consistent querying over such documents. The notion of modification of an XML document is based on standard edit operations on the underlying tree structure, e.g., relabeling a node, deleting a node, and inserting a new node. Quite a different line of work deals with editing of *schemas*, rather than documents. For example, [14] deals with a similarity measure on schemas given by considering *embeddings* that preserve the DTD structure. The notion of similarity thus depends on the syntactic presentation of the schema, not the language of documents that it defines.

In this work we consider the question of the edit distance between XML schemas as well. Like [14], we provide a method to notice similarity of schemas that may differ a bit, and like [22] we will make use of edit distance on trees. Our results give a means of detecting whether schema R (for restriction) can be edited into schema T (for target). Unlike [14] we do this on a purely semantic basis – R is repairable into T exactly when the documents satisfying R can be repaired into documents satisfying T. More formally, we consider a schema R to be "almost included" in schema T if every document t in R can be repaired to a document t' in T using a finite, uniformly bounded number of edits. In this case, we say that R is *bounded repairable* into T. The problem is motivated not only by schema matching, but also from data cleaning: we have some integrity property that we want to assure on documents (the target), and we have some property that we can assume the documents already have (the restriction). Bounded repairability states that we can clean the inputs to ensure conformance to our target constraint with a bounded amount of distortion of the input data.

The following examples show that it is not at all obvious whether one schema is bounded repairable into another.

EXAMPLE 1. *Consider the following DTDs:*

$$\begin{array}{llcll}
R: & r \rightarrow d,\ c^* & \qquad & T: & r \rightarrow a^*,\ e \\
 & d \rightarrow a^*,\ b^* & & & e \rightarrow b^*,\ c^* \\
 & a \rightarrow \mathit{EMPTY} & & & a \rightarrow \mathit{EMPTY} \\
 & b \rightarrow \mathit{EMPTY} & & & b \rightarrow \mathit{EMPTY} \\
 & c \rightarrow \mathit{EMPTY} & & & c \rightarrow \mathit{EMPTY}
\end{array}$$

ICDT 2012, March 26–30, 2012, Berlin, Germany.

The left-hand side schema R defines the language of all trees of the form $r(d(a,\ldots,a,b,\ldots,b),c,\ldots,c)$, while the right-hand side schema T defines the language of all trees of the form $r(a,\ldots,a,e(b,\ldots,b,c,\ldots,c))$. We claim that R is repairable into T with a uniformly bounded number of edit operations. Indeed, given a tree $r(d(a,\ldots,a,b,\ldots,b),c,\ldots,c)$ satisfying R, one can first delete the node labeled by d, obtaining the tree $r(a,\ldots,a,b,\ldots,b,c,\ldots,c)$, and then insert a new e-labeled node under the root, which adopts as children all the nodes labeled by b or c; this results in a tree $r(a,\ldots,a,e(b,\ldots,b,c,\ldots,c))$ that satisfies T.

EXAMPLE 2. *Consider the following DTDs:*

R':	$r \to a$	T':	$r \to a$
	$a \to b^*$		$a \to b^*,\ c$
	$b \to EMPTY$		$b \to EMPTY$
			$c \to EMPTY$

It is easy to see that R' is bounded repairable into T': any tree $r(a(b,\ldots,b))$ in R' can be modified into a tree in T' by inserting a new c-labeled node as a right-sibling of the nodes labeled by b. However, if we replace in both DTDs R' and T' the rule $r \to a$ with the rule $r \to a^*$, we obtain a new pair of languages R'' and T'' such that R'' is not bounded repairable into T''. This example suggests that bounded repairability depends on some interplay between the rules of DTDs and, more generally, between the specifications of the labellings of the nodes at different levels of the trees.

We will deal with the notion of bounded repairability for schemas that are more general than DTDs, e.g., schemas that are given by regular tree languages [19], and which capture the structural part of the W3C's XML schema [13]. We will formalize the edit distance between regular tree languages, and from this define the *bounded repair problem*, that is, the problem of deciding bounded repairability between two given tree languages R and T. Our main result is that it is decidable whether or not R can be repaired into T with a uniformly bounded number of edits.

For regular languages of words, the bounded repair problem was resolved in [6]: there it was shown that the problem is coNP-complete when the languages are represented by deterministic finite state automata, and a characterization of bounded repairability was given using a coverability relation between chains of connected components of the automata. In the case of tree languages, the problem turns out to be much more complex, both in terms of complexity and in terms of proof techniques that are needed to resolve it. We will provide a characterization of bounded repairability that exploits a suitable notion of component of a stepwise tree automaton [11], i.e., a form of automaton that turns out to be particularly convenient for analyzing repairs. An additional complication for the tree case is that we need to consider structures of connected components of stepwise tree automata that take the form of trees, rather than chains. Our characterization of the bounded repairability of R into T requires that every component structure of R can be "covered" by a component structure of T. The notion of covering is subtle, and the proof that it captures bounded repairability requires lifting the notion of edit from the level of the individual trees to the level of the component trees associated with the automata for R and T.

Once we have our characterization, we can apply it to get decidability of the bounded repair problem, and with some additional optimization we can give complexity bounds. We can also apply the characterization theorems to show that bounded repairability is much simpler to check for special classes of schemas. For example, we show that it is much less complex for deterministic DTDs when the alphabet is fixed, and much less complex when the restriction language R is trivial (i.e., the class of all trees).

Organization. In Section 2 we give some background on tree languages and edit operations. In Section 3 we give the formal statement of our main result, that is, a characterization of exactly which pairs of schemas are bounded repairable. The proof takes up Section 4. Section 5 uses the characterization theorem to get complexity bounds, while Section 6 gives a different characterization of bounded repairability for the case where the restriction language is trivial, and derives the corresponding complexity bounds. Section 7 gives conclusions and related work.

2. BACKGROUND

2.1 Curried trees and contexts

Throughout this paper we often use the *curry encoding* to represent unranked trees. The curry encoding [11, 17], also known as the extension encoding, represents an unranked tree labeled with elements of Σ as a binary tree whose inner nodes are labeled with a distinguished symbol @ and leaves with elements of Σ. The encoding, denoted ext, is defined as follows (with $a \in \Sigma$):

$$\mathsf{ext}(a) = a,$$
$$\mathsf{ext}(a(t_1,\ldots,t_n)) = @(\mathsf{ext}(a(t_1,\ldots,t_{n-1}),\mathsf{ext}(t_n))).$$

To simplify the notations, we write the symbol @ as an infix, left-associative operator. Figure 1 illustrates the encoding of an unranked tree. The inverse ext^{-1} of the encoding is defined by providing the symbol @ with the semantics of the extension operator on unranked tree and by evaluating the expression in a bottom-up fashion, i.e., $\mathsf{ext}^{-1}(a) = a$ and $\mathsf{ext}^{-1}(a@t_1@\ldots@t_n) = a(\mathsf{ext}^{-1}(t_1),\ldots,\mathsf{ext}^{-1}(t_n))$. For a finite set of labels Σ, we denote by $\mathcal{T}_\Sigma$ the set of all *curried trees* over Σ.

We make an important observation that the root node of an unranked tree corresponds to the leftmost leaf in the corresponding curried tree. Other remarks follow [11, 17]. We point out that ext is a one-to-one mapping between the set of unranked trees and the set of curried trees. We also note that there is a one-to-one correspondence between the nodes of an unranked tree and the leaves of the curried encoding. Moreover, the *yield* of a curried tree, i.e., the sequence of leaves taken from left to right, corresponds to the standard left-to-right pre-order traversal of the unranked tree. Another observation follows from the semantics of the extension operator: the inner nodes of a curried tree, labeled with @, correspond to the edges of the unranked tree.

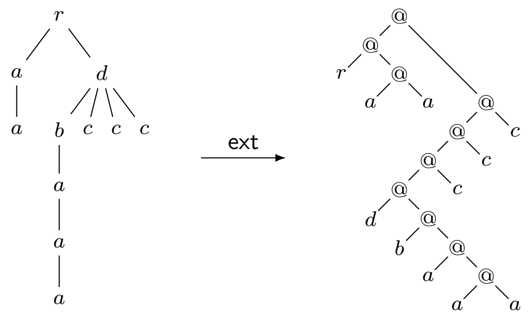

Figure 1: Curry encoding of an unranked tree.

As usual, we use sequences of natural numbers to identify unambiguously the nodes in a tree: ε is the root node and $x{\cdot}i$ is the i-th child of node x. We denote by $\mathsf{nodes}(t)$ the set of all nodes in the tree t. For a tree t and a node $x \in \mathsf{nodes}(t)$, we denote by $t(x)$ the label of the node x in t. We also introduce the partial orders $\preccurlyeq_t^{\mathsf{post}}$ and $\preccurlyeq_t^{\mathsf{anc}}$ on the nodes of a tree t; these two partial orders are induced by the standard *post-order* and *ancestor relation* of t, respectively.

We fix a special label $\bullet$ not in Σ to be used as a placeholder in contexts. Formally, a (*curried*) *context* over Σ is an element of $\mathcal{T}_{\Sigma\cup\{\bullet\}}$ with $\bullet$ occurring exactly once *in a leaf*. By $\mathcal{C}_\Sigma$ we denote the set of all contexts over Σ. The *empty* context is the context $\bullet$ having exactly one node. For a context C and a tree t we denote by $C \circ t$ the tree obtained from the substitution of $\bullet$ by t in C. Similarly, the composition $C_1 \circ C_2$ of two contexts C_1 and C_2 is obtained from the substitution of the placeholder in C_1 by C_2. (this results again in a context in $\mathcal{C}_\Sigma$).

2.2 Stepwise tree automata

We use stepwise tree automata [11, 17] to specify regular tree languages. These are essentially bottom-up tree automata running over the curry encodings of trees. Formally, a *stepwise automaton* is a tuple $\mathcal{A} = (\Sigma, Q, \delta_0, \delta, F)$, where:

1. Σ is a finite set of labels,
2. Q is a finite set of states,
3. $\delta_0 : \Sigma \to 2^Q$ is an assignment of initial states to labels,
4. $\delta : Q \times Q \to 2^Q$ is a transition function,
5. $F \subseteq Q$ is a set of final states.

We say that $\mathcal{A}$ is *deterministic* if δ_0 (resp., δ) can be described as a partial function from Σ (resp., $Q \times Q$) to Q. It is often convenient to represent δ_0 and δ as a set of rules. For instance, we write $a \to q$ to indicate that $q \in \delta_0(a)$ and $q_1@q_2 \to q$ to indicate that $q \in \delta(q_1, q_2)$.

A *run* of a stepwise automaton $\mathcal{A} = (\Sigma, Q, \delta_0, \delta, F)$ on a tree $t \in \mathcal{T}_\Sigma$ is a function $\rho : \mathsf{nodes}(t) \to Q$ such that

- for every leaf node x, $\rho(x) \in \delta_0(t(x))$,
- for every inner node x, $\rho(x) \in \delta(\rho(x \cdot 1), \rho(n \cdot 2))$.

A run ρ is *accepting* if $\rho(\varepsilon) \in F$. The language recognized by $\mathcal{A}$, denoted $\mathscr{L}(\mathcal{A})$, is the set of all trees $t \in \mathcal{T}_\Sigma$ on which $\mathcal{A}$ has an accepting run.

Example 3. *As a running example, consider the two DTDs:*

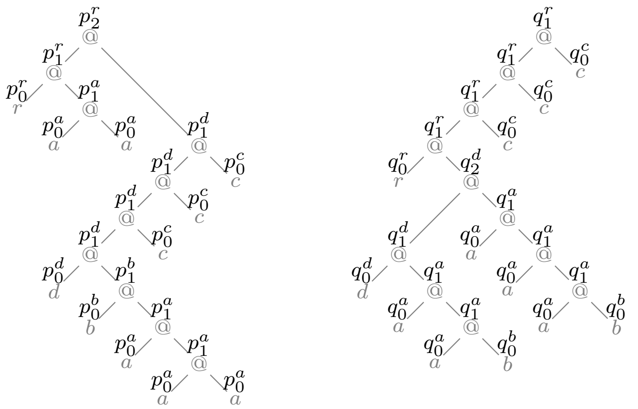

Figure 2: Runs of two automata $\mathcal{R}$ and $\mathcal{T}$.

$$\begin{array}{llll}
D: & r \to a,\ d & D': & r \to d,\ c* \\
 & a \to a \ \mid\ EMPTY & & d \to a,\ a \\
 & d \to b,\ c* & & a \to a \ \mid\ b \\
 & b \to a & & b \to EMPTY \\
 & c \to EMPTY & & c \to EMPTY
\end{array}$$

The following two stepwise automata capture (modulo the curry encoding) the languages defined by the previous DTDs (the underlined states are final and any rule using $q_?^a$ translates to two rules using q_0^a and q_1^a):

$$\begin{array}{lllllll}
\mathcal{R}: & r \to p_0^r & p_0^r@p_?^a \to p_1^r & \quad & \mathcal{T}: & r \to q_0^r & q_0^r@q_2^d \to \underline{q_1^r} \\
 & a \to p_0^a & p_1^r@p_1^d \to \underline{p_2^r} & & & d \to q_0^d & \underline{q_1^r}@q_0^c \to \underline{q_1^r} \\
 & d \to p_0^d & p_0^a@p_?^a \to p_1^a & & & a \to q_0^a & q_0^d@q_1^a \to q_1^d \\
 & b \to p_0^b & p_0^d@p_1^b \to p_1^d & & & b \to q_0^b & q_1^d@q_1^a \to q_2^d \\
 & c \to q_0^c & p_1^d@p_0^c \to p_1^d & & & c \to q_0^c & q_0^a@q_1^a \to q_1^a \\
 & & p_0^b@p_?^a \to p_1^b & & & & q_0^a@q_0^b \to q_1^a
\end{array}$$

Figure 2 presents the (accepting) runs of the automata $\mathcal{R}$ and $\mathcal{T}$ on the curry encodings of some trees t and t', respectively.

Stepwise automata capture exactly the class of regular (unranked) tree languages [11] and they are more succinct than many other classes of tree automata [17]. Other models that capture the same class of languages, such as *unranked tree automata* [19] are more frequently used. Since unranked tree automata can be converted into stepwise tree automata in polynomial time, algorithms for analyzing stepwise automata provide the same complexity bounds for unranked tree automata – all of our theorems will apply to unranked tree automata as well as stepwise tree automata. The main advantage of using stepwise automata in our proofs is due to their ability of capturing the cyclic behavior of a regular tree language, defined via a suitable notion of strongly connected component (see Section 3 for more details).

In the sequel, we will work with *trimmed* automata only, namely, we assume that every state of an automaton appears in some accepting run (this implies that all states are both accessible and co-accessible). Every automaton can be made trimmed in linear time.

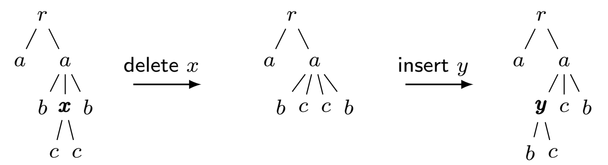

Figure 3: Edit operations on unranked trees.

As is usual for word automata, we extend the transition function δ of a stepwise automaton to trees in $\mathcal{T}_\Sigma$ and to contexts in $\mathcal{C}_\Sigma$. Precisely, we define the function $\delta^* : \mathcal{T}_\Sigma \to 2^Q$ such that $q \in \delta^*(t)$ iff there exists a run ρ of $\mathcal{A}$ on t and $\rho(\varepsilon) = q$. Similarly, we define the function $\delta^*_\bullet : Q \times \mathcal{C}_\Sigma \to 2^Q$ such that $q' \in \delta^*_\bullet(q, C)$ iff there exists a run ρ of $\mathcal{A}_q = (\Sigma \cup \{\bullet\}, Q, \delta_0 \cup \{(\bullet, q)\}, \delta, Q)$ on C and $\rho(\varepsilon) = q'$ (intuitively, we simulate some computation of $\mathcal{A}$ on C under the assumption that the placeholder is assigned state q). By an abuse of notation, we will denote δ^* and $\delta^*_\bullet$ simply by δ.

2.3 Edit distance and bounded repairability

We briefly recall the definitions of some standard edit operations on unranked trees. The first operation, called *deletion*, consists of removing a distinguished (non-root) node x from a tree t and promoting its subtrees as children of its parent. The second operation, called *insertion*, consists of adding a new node x in an unranked tree t, with a possible adoption of a list of subsequent children from the parent of x. Figure 3 gives an example of these two operations. These are the standard edit-operations that are used to define the edit-distance between trees [7]. Note that the operation of *relabeling* a node in an unranked tree, which is sometimes used as a standard edit operation, is subsumed by the previous two operations.

We study the *bounded repair problem* for regular tree languages, which consists of deciding, given two regular tree languages R and T, whether one can transform any tree $t \in R$ into a tree $t' \in T$ using a finite, uniformly bounded number of edits. Formally, given two unranked trees t and t', we denote by $\mathsf{dist}(t, t')$ the minimum number of edits that are needed for transforming t into t'. Given two regular tree languages R and T, we then define

$$\mathsf{dist}(R, T) \;=\; \max_{t \in R} \; \min_{t' \in T} \; \mathsf{dist}(t, t')$$

to be the *worst-case cost of repairing R into T*. If the cost $\mathsf{dist}(R, T)$ is finite, then we say that R *is bounded repairable into* T and we write $R \hookrightarrow_{\mathrm{BR}} T$ for short.

Note that the bounded repairability relation $\hookrightarrow_{\mathrm{BR}}$ satisfies some key properties, which will be used later on:

- Subset-subsumption, i.e., $R \subseteq T$ implies $R \hookrightarrow_{\mathrm{BR}} T$.
- Transitivity, i.e., $R \hookrightarrow_{\mathrm{BR}} T \hookrightarrow_{\mathrm{BR}} S$ implies $R \hookrightarrow_{\mathrm{BR}} S$.
- Union-compatibility, i.e., $R \hookrightarrow_{\mathrm{BR}} T$ and $R' \hookrightarrow_{\mathrm{BR}} T'$ imply $R \cup R' \hookrightarrow_{\mathrm{BR}} T \cup T'$.

3. MAIN CHARACTERIZATION

In this section we give an effective characterization of the bounded repairability relation between regular tree languages. Similarly to the string case [6], this characterization is based on the notion of strongly connected component of the transition graph of a stepwise automaton. In the string case, a suitable coverability relation between chains of components is used to characterize bounded repairability. Because here we work with trees, we need to generalize the notion of coverability to a relation over the so-called synopsis trees, i.e., full binary trees with nodes labeled by strongly connected components.

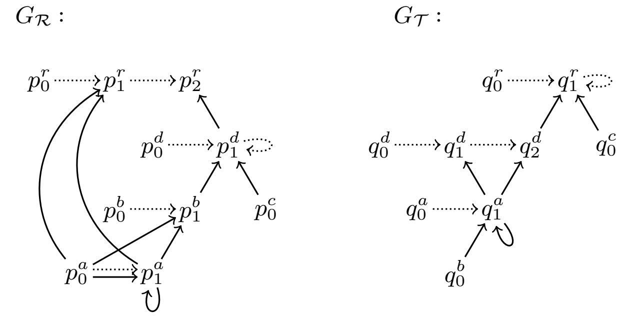

Figure 4: Transitions graphs of automata $\mathcal{R}$ and $\mathcal{T}$.

3.1 Components of stepwise automata

Given a stepwise automaton $\mathcal{A} = (\Sigma, Q, \delta_0, \delta, F)$, the *transition graph* of $\mathcal{A}$ is the graph $G_\mathcal{A} = (Q, E_h \cup E_v)$, where

$$\begin{aligned} E_h &= \big\{(q_1, q) \in Q \times Q \;\mid\; \exists q_2.\; q \in \delta(q_1, q_2)\big\}, \\ E_v &= \big\{(q_2, q) \in Q \times Q \;\mid\; \exists q_1.\; q \in \delta(q_1, q_2)\big\}. \end{aligned}$$

We call the edges in E_v (resp., E_h) *vertical* (resp., *horizontal*). Note that an edge may be both vertical and horizontal. As an example, Figure 4 depicts the transition graphs of the automata $\mathcal{R}$ and $\mathcal{T}$ of Example 3 (dashed arrows represent horizontal edges, solid arrows represent vertical edges).

Recall that a set X of nodes of a graph $G = (V, E)$ is a *strongly connected component* (or simply a *component*) iff X is maximal such that for every two $x, y \in X$, there is a path from x to y visiting nodes in X only. By $\mathsf{SCC}(\mathcal{A})$ we denote the set of all strongly connected components in the transition graph of $\mathcal{A}$. We associate with each component $X \in \mathsf{SCC}(\mathcal{A})$ the language $\mathscr{L}(\mathcal{A} \mid X)$ of contexts that are realizable within X:

$$\mathscr{L}(\mathcal{A} \mid X) \;=\; \big\{C \in \mathcal{C}_\Sigma \;\mid\; \exists p, q \in X.\; q \in \delta(p, C)\big\}.$$

Below, we identify particular types of components in the transition graph of an automaton. We say that a curried context C is *horizontal* if its placeholder $\bullet$ appears at the leftmost leaf (possibly at the root, if the context is a singleton). Essentially, a horizontal context C represents a *hedge*, i.e., a sequence of unranked trees that are encoded by the sub-trees below the leftmost branch of C. Concatenations of hedges thus correspond to compositions of horizontal contexts. For instance, given two horizontal contexts C and C', $C \circ C'$ is a horizontal context that represents the concatenation of the two hedges represented by C and C', respectively.

A component $X \in \mathsf{SCC}(\mathcal{A})$ is *horizontal* iff it realizes horizontal contexts only, namely, for every $C \in \mathscr{L}(\mathcal{A} \mid X)$, C is horizontal. Notice that an horizontal component of $\mathcal{A}$ can

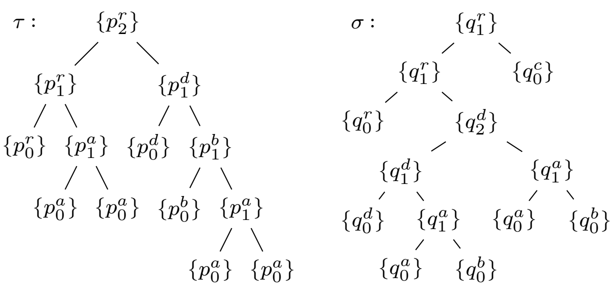

Figure 5: Synopsis trees τ and σ of automata $\mathcal{R}$ and $\mathcal{T}$.

also be seen as a sub-automaton of $\mathcal{A}$ that contains horizontal transitions only. Similarly, we say that X is *trivial* iff it realizes the empty context only, i.e., $\mathscr{L}(\mathcal{A} \mid X) = \{\bullet\}$. Note that trivial components are horizontal.

As an example, consider again the transition graphs of Figure 4. All components except $\{p_1^d\}$, $\{p_1^a\}$, $\{q_1^r\}$, and $\{q_1^a\}$ are trivial. The components $\{p_1^d\}$ and $\{q_1^r\}$ are non-trivial horizontal, since they both realize the contexts $\bullet$, $\bullet@c$, $(\bullet@c)@c$, ... The components $\{p_1^a\}$ and $\{q_1^a\}$ are non-horizontal, since they both realize the contexts $\bullet$, $a@\bullet$, $a@(a@\bullet)$, ...

3.2 Synopsis trees

We now introduce a suitable structure that eases the characterization of bounded repairability, namely, the synopsis tree. Formally, a *synopsis tree* of an automaton $\mathcal{A}$ is a full binary tree whose nodes are labeled with elements of $\mathsf{SCC}(\mathcal{A})$. The *semantics* $[\![\tau]\!]_{\mathcal{A}}$ of a synopsis tree τ of $\mathcal{A}$ is the language of curried trees recursively defined as follows:

$$[\![X]\!]_{\mathcal{A}} \;=\; \big\{ C \circ a \;\mid\; C \in \mathscr{L}(\mathcal{A} \mid X),\ a \in \Sigma \big\}$$

$$[\![X(\tau_1,\tau_2)]\!]_{\mathcal{A}} \;=\; \left\{ C \circ (t_1 @ t_2) \;\middle|\; \begin{array}{l} C \in \mathscr{L}(\mathcal{A} \mid X), \\ t_1 \in [\![\tau_1]\!]_{\mathcal{A}},\ t_2 \in [\![\tau_2]\!]_{\mathcal{A}} \end{array} \right\}$$

with $X \in \mathsf{SCC}(\mathcal{A})$. Figure 5 contains two synopsis trees τ and σ, respectively for the automata $\mathcal{R}$ and $\mathcal{T}$ of Example 3.

Next, we identify a family of synopsis trees that captures "closely enough" the language recognized by an automaton.

Definition 1. *A* primitive synopsis tree *of an automaton $\mathcal{R} = (\Sigma, Q, \delta_0, \delta, F)$ is a synopsis tree τ of $\mathcal{R}$ such that:*

1. *τ respects the transition function of $\mathcal{R}$, i.e., for all nodes x, $x \cdot 1$, and $x \cdot 2$ in τ, there exist some states $q \in \tau(x)$, $q_1 \in \tau(x \cdot 1)$, and $q_2 \in \tau(x \cdot 2)$ such that $q \in \delta(q_1, q_2)$;*
2. *every internal node of τ has label different from the labels of its children, i.e., for all nodes x, $x \cdot 1$, and $x \cdot 2$ in τ, $\tau(x \cdot 1) \neq \tau(x) \neq \tau(x \cdot 2)$.*

$\mathsf{PST}(\mathcal{R})$ *denotes the set of all primitive synopsis trees of $\mathcal{A}$.*

The tree τ depicted in Figure 5 is a primitive synopsis tree. In particular, this tree respects the transitions of the run of the automaton $\mathcal{R}$ on the left-hand side of Figure 2.

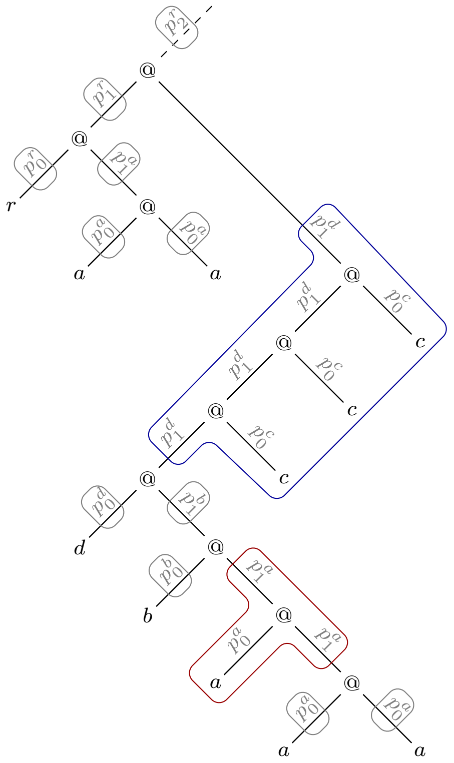

Figure 6: Decomposition of a tree.

The idea underlying the notion of primitive synopsis tree is to capture the "cyclic behavior" of the components of the restriction automaton. This cyclic behavior has to be taken into account in the characterization of bounded repairability because it could generate arbitrary large fragments of trees that cannot be edited with uniformly bounded cost. Moreover, the use of primitive synopsis trees as a representation of the restriction language $\mathscr{L}(\mathcal{R})$ is *sound* because the set of primitively synopsis trees contains (via the previously defined semantics of synopsis tree) the language $\mathscr{L}(\mathcal{R})$:

Lemma 1. *$\mathscr{L}(\mathcal{R}) \subseteq \bigcup_{\tau \in \mathsf{PST}(\mathcal{R})} [\![\tau]\!]_{\mathcal{R}}$ for any stepwise automaton $\mathcal{R}$.*

We illustrate the proof of this lemma on the tree t from Figure 1 and the automaton $\mathcal{R}$ from Example 3. For this we consider the accepting run of $\mathcal{R}$ of Figure 2 and we use it to decompose t into a binary tree of contexts, where each context is realized by some SCC of $\mathcal{R}$. We present this decomposition in Figure 6, where for better visualization we place the states of the run not on the nodes but on the edges above them and we add a virtual edge for the root.

The decomposition procedure works in a recursive manner and begins at the root node. When executed at a node x with a state q (which belongs to some component $X \in \mathsf{SCC}(\mathcal{R})$), the procedure creates an empty context $C = \bullet$ which spans the edge above the node. If a child of x has a state that belongs to the same component X, then the procedure moves to this node and expands the context C to span the edge above the node child and the whole subtree rooted at the opposite child. This step is repeated iteratively until the procedure reaches a leaf node or a node whose both children have states not in X. In the latter case, the procedure is called recursively on both children nodes, creating two separate children context nodes.

Clearly, the contexts created during the decomposition are realized by some components of $\mathcal{R}$ and the structure of these components takes the form of a synopsis tree. For instance, for the decomposition of Figure 6 the resulting synopsis tree is τ in Figure 5. Naturally, this synopsis tree respects the transitions of the run of $\mathcal{R}$ on the input tree. Furthermore, since every context has been maximally expanded, every node of the synopsis tree does not share the same label with any of its children. This shows that the constructed synopsis tree is primitive.

We also remark that the height of a primitive synopsis tree of a stepwise automaton $\mathcal{R}$ is bounded by the number of different components in $G_{\mathcal{R}}$ and hence by the number of states of $\mathcal{R}$. Consequently, $\mathsf{PST}(\mathcal{R})$ is a finite language and, moreover, it can be represented with a simple deterministic binary bottom-up tree automaton whose size is polynomial in the size of $\mathcal{R}$.

In order to represent the target language and the possible edited trees, one needs a relaxed version of primitive synopsis trees:

Definition 2. *A* basic synopsis tree *of an automaton $\mathcal{T}$ is a synopsis tree σ of $\mathcal{T}$ that respects the transition function of $\mathcal{T}$ (cf. Definition 1). We denote by $\mathsf{BST}(\mathcal{T})$ the set of all basic synopsis trees of $\mathcal{T}$.*

For example, the tree σ in Figure 5 is a basic synopsis tree that respects the transitions of the run of the automaton $\mathcal{T}$ depicted on the right-hand side of Figure 2.

Notice that the set of basic synopsis trees of an automaton $\mathcal{T}$ can be represented by a deterministic binary bottom-up tree automaton of size polynomial in the size of $\mathcal{T}$, even though this set is not necessarily finite.

The following lemma shows that the language given by the semantics of a basic synopsis tree of $\mathcal{T}$ is bounded repairable into the language $\mathscr{L}(\mathcal{T})$.

Lemma 2. $[\![\sigma]\!]_{\mathcal{T}} \hookrightarrow_{\mathrm{BR}} \mathscr{L}(\mathcal{T})$ *for any stepwise automaton $\mathcal{T}$ and any $\sigma \in \mathsf{BST}(\mathcal{T})$.*

The proof of this lemma is technical and is based on the following observations. Any tree $t \in [\![\sigma]\!]_{\mathcal{T}}$ can be seen as a composition of contexts, one for every node of σ and each belonging to the language of the corresponding component. Every such context can be decorated with states from a run of $\mathcal{T}$ that justifies that the context belongs to the language of its corresponding component. To make this decoration a proper run of $\mathcal{T}$, one needs to insert additional contexts that allow the transition from one component to the other – this is possible because the basic synopsis tree σ respects the transition function of $\mathcal{T}$ and because σ is labeled with strongly connected components. Additionally, the symbols used for substitution in the semantics of the leaf nodes of σ may need to be replaced by appropriate tree fragments. Finally, we observe that the number of inserted contexts and tree fragments depends only on the size of σ and their sizes depend only on the size of $\mathcal{T}$, which means that the number of edits that are required to repair t into $\mathscr{L}(\mathcal{T})$ is independent of the size of t.

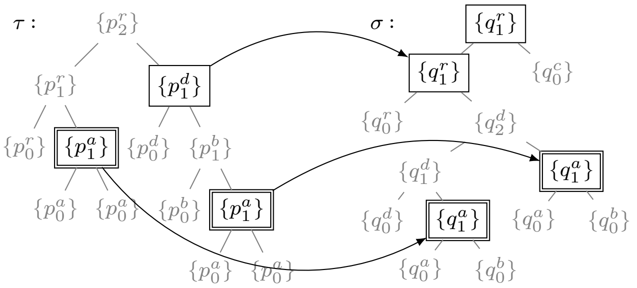

Figure 7: Covering of PST τ by BST σ.

3.3 Coverings

The remaining part of the puzzle is to identify how to connect the primitive synopsis trees of the restriction automaton $\mathcal{R}$ to a finite subset of basic synopsis trees of the target automaton $\mathcal{T}$. This is accomplished by the notion of coverability between synopsis trees.

Definition 3. *Given two automata $\mathcal{R}$ and $\mathcal{T}$ and two synopsis trees τ of $\mathcal{R}$ and σ of $\mathcal{T}$, we say that σ* covers *τ iff there exists an injective mapping λ of non-trivial nodes of τ to non-trivial nodes of σ such that:*

1. *λ maps components in a compatible manner, i.e., $\mathscr{L}(\mathcal{R} \mid \tau(x)) \subseteq \mathscr{L}(\mathcal{T} \mid \sigma(\lambda(x)))$ for every non-trivial node x of τ;*
2. *λ preserves the post-order of non-trivial nodes, i.e., $x \preccurlyeq_{\tau}^{\mathsf{post}} y$ iff $\lambda(x) \preccurlyeq_{\sigma}^{\mathsf{post}} \lambda(y)$ for any two non-trivial nodes x and y of τ;*
3. *λ preserves the ancestorship of non-horizontal nodes, i.e., $x \preccurlyeq_{\tau}^{\mathsf{anc}} y$ iff $\lambda(x) \preccurlyeq_{\sigma}^{\mathsf{anc}} \lambda(y)$ for every non-horizontal node x of τ and every non-trivial node y of τ.*

Figure 7 presents a covering of a primitive synopsis tree τ of $\mathcal{R}$ by a basic synopsis tree σ of $\mathcal{T}$ (square boxes represent non-trivial nodes, and they have double borders when the component is non-horizontal).

We are now able to state the main theorem of the paper:

Theorem 1 (Characterization). *Given two regular tree languages specified by stepwise automata $\mathcal{R}$ and $\mathcal{T}$, $\mathscr{L}(\mathcal{R})$ is bounded repairable into $\mathscr{L}(\mathcal{T})$ iff every primitive synopsis tree of $\mathcal{R}$ is covered by some basic synopsis tree of $\mathcal{T}$.*

Before turning to the proof of the above result, we explain the main ideas underlying the notion of covering. As a

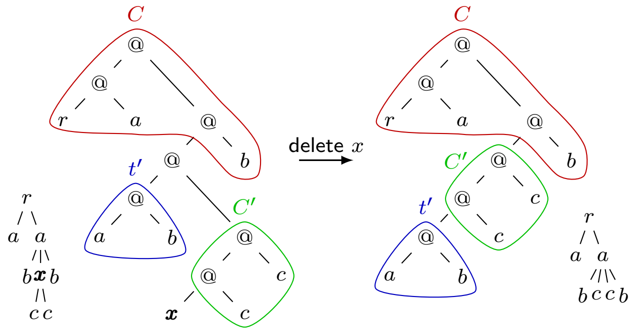

Figure 8: Deleting a node in the curry encoding.

first remark, we observe that, given $\mathcal{R}$ and $\mathcal{T}$ such that $\mathscr{L}(\mathcal{R}) \hookrightarrow_{\mathrm{BR}} \mathscr{L}(\mathcal{T})$, any reasonable strategy for repairing $\mathcal{R}$ into $\mathcal{T}$ with a uniformly bounded number cost will apply the edit operations only at the "junctions" of contexts realized by different components. Intuitively, this property holds because the non-trivial components of $\mathcal{R}$ can realize arbitrary large repetitions of the same context – these repetitions either do not need any editing at all, or they need an arbitrary large amount of editing. This gives an intuitive account for enforcing containments between languages recognized by components in the first condition of Definition 3 (not surprisingly, a similar condition was introduced in [6] for characterizing bounded repairability between regular languages of words).

As for the other two conditions, it is worth looking at the effect of an edit operation on the curry encoding of an unranked tree. Let us consider an unranked tree t with a distinguished node x that has to be deleted. There is a unique way to represent the curry encoding of t together with the node x as an expression of the form $C \circ (t' @ (C' \circ a))$, where C' is a *horizontal* context that represents the hedge of subtrees under x and a is the label of x. The result of the deletion of x from t is encoded by the curried tree $C \circ (C' \circ t')$ (see Figure 8 for an example). Note that this operation does not allow the deletion of the leftmost leaf node in the curried tree (this would correspond to deleting the root node in an unranked tree, an operation that is typically prohibited). The operation of inserting a new node y in an unranked tree t can be described in a similar way using curry encodings and horizontal contexts. Precisely, given an unranked tree t with curry encoding $C \circ (C' \circ t')$, where C' is a horizontal context, the curried tree $C \circ (t' @ (C' \circ a))$ represents the unranked tree that results from the insertion of a new a-labeled node y in t having as children the hedge represented by C'.

We now observe that the transformations on curried trees described above satisfy two crucial properties: (i) they preserve the post-order of the nodes and (ii) they preserve the ancestorship of non-horizontal contexts (e.g., C in Figure 8) with their descendants. These properties are precisely captured by the last two conditions of Definition 3.

4. PROOF OF THE MAIN RESULT

The following subsections are devoted to prove the two directions of the characterization given in Theorem 1.

4.1 From covering to repair

We begin with the proof of the "if" direction of Theorem 1, namely, we show how we can derive a strategy for repairing $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with a uniformly bounded number of edits, under the assumption that every primitive synopsis tree of $\mathcal{R}$ is covered by some basic synopsis tree of $\mathcal{T}$.

In addition to Lemmas 1 and 2, we need the following crucial property:

LEMMA 3. *For any synopsis tree τ of $\mathcal{R}$ and any synopsis tree σ of $\mathcal{T}$, if σ covers τ, then $[\![\tau]\!]_{\mathcal{R}} \hookrightarrow_{\mathrm{BR}} [\![\sigma]\!]_{\mathcal{T}}$.*

Before sketching the proof of Lemma 3, we explain how the "if" direction of Theorem 1 follows from it. By Lemma 1, the restriction language is contained in the union of the languages induced by the semantics of the primitive synopsis trees of $\mathcal{R}$. By hypothesis, each of these trees is covered by a basic synopsis tree of the target automaton $\mathcal{T}$. Thus by Lemma 3 their languages can be transformed into the languages induced by the semantics of some basic synopsis trees, using a bounded number of edits. By Lemma 2 the languages of the basic synopsis trees can be in turn repaired into the target language. The result now follows from the fact that there are only finitely many primitive synopsis trees and the fact that bounded repairability is a transitive relation and is preserved under finite unions.

We now turn to the proof of Lemma 3. For a technical reason, we need to slightly extend the definition of synopsis tree by allowing the use of a special node labeled ϵ that represents a dummy trivial component. The semantics is extended in the natural way by letting $\mathscr{L}(\mathcal{A} \mid \epsilon) = \{\bullet\}$ (for any stepwise automaton $\mathcal{A}$). Because all trivial components have the same semantics (i.e., they all recognize the language $\{\bullet\}$), we shall often identify a trivial component of an automaton with the dummy component ϵ.

The first step consists of "interpolating" the two synopsis trees τ and σ by a synopsis tree θ of $\mathcal{R}$ such that:

- θ has the same labels (i.e., components) of τ on the non-trivial nodes and it covers τ via a *bijection* (between non-trivial nodes) that maps any non-trivial node of τ with label $X \in \mathsf{SCC}(\mathcal{R})$ to a non-trivial node of θ with the same label X (we say that τ *strongly covered* θ),
- θ has the same domain (i.e., set of nodes) of σ and it is covered by σ via the *identity* function between non-trivial nodes (we say that θ is *embedded* in σ).

It is not difficult to show that such an interpolating synopsis tree θ exists and that the language $[\![\theta]\!]_{\mathcal{R}}$ is contained in the language $[\![\sigma]\!]_{\mathcal{R}}$:

LEMMA 4. *If τ is covered by σ, then there is an synopsis tree θ of $\mathcal{R}$ such that τ is strongly covered by θ and θ is embedded in σ.*

LEMMA 5. *For a synopsis tree θ of $\mathcal{R}$ and a synopsis tree σ of $\mathcal{T}$, if θ is embedded in σ, then $[\![\theta]\!]_{\mathcal{R}} \subseteq [\![\sigma]\!]_{\mathcal{T}}$.*

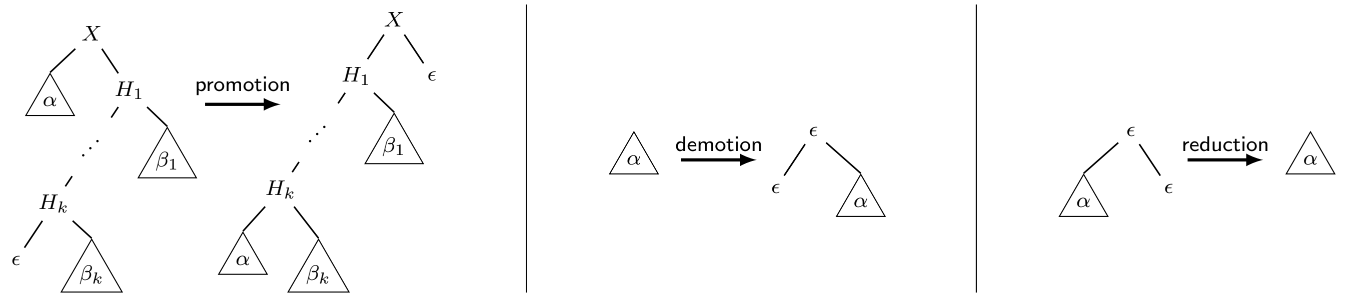

Figure 9: Synopsis tree operations.

Recall that $[\![\theta]\!]_{\mathcal{R}} \subseteq [\![\sigma]\!]_{\mathcal{T}}$ implies $[\![\theta]\!]_{\mathcal{R}} \hookrightarrow_{\mathrm{BR}} [\![\sigma]\!]_{\mathcal{T}}$ and that the bounded repairability relation $\hookrightarrow_{\mathrm{BR}}$ is transitive. We thus reduced the problem to showing that $[\![\tau]\!]_{\mathcal{R}} \hookrightarrow_{\mathrm{BR}} [\![\theta]\!]_{\mathcal{R}}$. Towards this goal, we can take advantage of the notion of strong coverability, and in particular of the fact that this is an *equivalence relation*, namely, it is reflexive, symmetric, and transitive (note that symmetry and transitivity follow from the fact that the mapping that witness strong coverability is a bijection between non-trivial nodes and preserves the labeling). In the sequel we only work with synopsis trees of the automaton $\mathcal{R}$. This allows us to remove the automaton $\mathcal{R}$ from the notation that follows.

The next step is to associate with any synopsis tree ζ (of $\mathcal{R}$) a suitable *normal form* ζ^* that can be used as a canonical representative of the equivalence class of ζ induced by the strong coverability relation. In order to derive the normal form of a given synopsis tree, we will introduce generic editing operations on synopsis trees that preserve the strong coverability relation and entail bounded repairability. We will then prove that $[\![\tau]\!]$ can be repaired into $[\![\theta]\!]$ with a uniformly bounded number of edits by first repairing $[\![\tau]\!]$ into $[\![\tau^*]\!]$ and then repairing $[\![\theta^*]\!]$ (= $[\![\tau^*]\!]$) into $[\![\theta]\!]$ (recall that τ and θ strongly cover each other and hence $\tau^* = \theta^*$ by canonicity of the normal form). The repair strategy that witnesses bounded repairability between $[\![\tau]\!]$ and $[\![\tau^*]\!]$ (resp., $[\![\theta^*]\!]$ and $[\![\theta]\!]$) can be read off the sequence of generic editing operations that takes τ to its normal form τ^* (resp., θ to its normal form θ^*).

We describe below the structure of a synopsis tree *in normal form*.

Definition 4. *A synopsis tree ζ is* in normal form *iff one of the following cases holds:*

1. *$\zeta = \epsilon$, namely, ζ consists of a single node labeled with a trivial component,*
2. *$\zeta = X(\alpha, \epsilon)$, where X is a non-trivial horizontal component and α is a synopsis tree in normal form,*
3. *$\zeta = \epsilon(\alpha, X(\beta, \epsilon))$, where X is a non-horizontal component and α and β are synopsis trees in normal form.*

We observe that the root of a synopsis tree in normal form is a horizontal (possibly trivial) node and that its left sub-tree is also in normal form. In particular, this means that all components along the leftmost branch of a synopsis tree in normal form are horizontal.

The following lemma shows that synopsis trees in normal form can be used as canonical representatives of the equivalence classes induced by the strong coverability relation. The proof of this lemma is by simple structural induction and case analysis.

Lemma 6. *If τ and ζ are two synopsis trees in normal form that strongly cover each other, then τ and ζ are isomorphic.*

Thanks to Lemma 6, we can define *the normal form ζ^** of a synopsis tree ζ as the unique synopsis tree that is *in normal form* and that *strongly covers* ζ, provided that this tree exists.

Our next goal is to prove that the normal form ζ^* of ζ indeed exists, and that can be attained by a finite sequence of generic editing operations on synopsis trees. These operations are called *promotion*, *demotion*, and *reduction*, and are presented in Figure 9. There, ϵ represents a trivial component, X represents an arbitrary component, $H_1, \ldots, H_k$ represent horizontal (possibly trivial) components, and $\alpha, \beta_1, \ldots, \beta_k$ represent arbitrary synopsis trees. Note that the figure describes the case where the promotion, demotion, and reduction operations are applied at the root of a synopsis tree – in general, these operations can be applied to any sub-tree of a synopsis tree. We write $\zeta \rightarrow^*_{\mathsf{op}} \zeta'$ whenever ζ' can be obtained from ζ by applying a finite sequence of promotion, demotion, and reduction operations. In order to give further intuition about these operations, we remark an analogy between the operations of promotion, depicted in Figure 9, and deletion, depicted in Figure 8 (a similar correspondence holds between the operations of demotion and insertion of a new root). In this case, the root X of the synopsis tree is acting as the the context C of the curried tree, the sub-tree α is acting as the curried sub-tree t', and the sub-tree rooted at H_1 is acting as the horizontal context C'.

It is important to point out that the editing operations on synopsis trees described above preserve the post-order of non-trivial nodes and the ancestorship of non-horizontal nodes. From this it follows that they also preserve the strong coverability relation. Furthermore, the following lemma shows that the normal form of a synopsis tree exists and can be obtained via a sequence of promotion, demotion, and reduction operations:

Lemma 7. *For any synopsis tree ζ, we have $\zeta \to_{\mathsf{op}}^{*} \zeta^{*}$.*

The proof goes by a structural induction on the synopsis tree ζ that has to be normalized. Specifically, one first normalizes the left and right sub-trees of ζ separately using induction. One then completes the normalization procedure by applying suitable operations on the basis of the component at the root of ζ: if this component is non-horizontal, then one applies a promotion operation followed by a demotion operation; if it is horizontal and non-trivial, then one only applies a promotion operation; if it is trivial, then one applies a promotion followed by a reduction operation.

The next lemma shows that if $\zeta \to_{\mathsf{op}}^{*} \zeta'$, then the two languages $[\![\zeta]\!]$ and $[\![\zeta']\!]$ are repairable one into each other with a uniformly bounded number of edits. In other words, the application of a promotion, demotion, or reduction operation to a synopsis tree ζ corresponds to a small amount of edits over $[\![\zeta]\!]$. The proof of this result is via a simple analysis of the transformations on languages of unranked trees that are induced by the operations of promotion, demotion, and reduction.

Lemma 8. *For any pair of synopsis trees ζ and ζ', if $\zeta \to_{\mathsf{op}}^{*} \zeta'$, then $[\![\zeta]\!] \hookrightarrow_{\mathrm{BR}} [\![\zeta']\!]$ and $[\![\zeta']\!] \hookrightarrow_{\mathrm{BR}} [\![\zeta]\!]$.*

We have now all the ingredients to prove Lemma 3. Remember that by Lemmas 4 and 5 it only remains to show that $[\![\tau]\!]_{\mathcal{R}} \hookrightarrow_{\mathrm{BR}} [\![\theta]\!]_{\mathcal{R}}$. The latter claim can be proved by combining Lemmas 6, 7, and 8. Indeed, we know from Lemma 7 that τ and θ can be converted into the normal forms τ^{*} and θ^{*}, respectively, by applying sequences of promotions, demotions, and reductions. Lemma 8 implies that $[\![\tau]\!]_{\mathcal{R}} \hookrightarrow_{\mathrm{BR}} [\![\tau^{*}]\!]_{\mathcal{R}}$ and, symmetrically, $[\![\theta^{*}]\!]_{\mathcal{R}} \hookrightarrow_{\mathrm{BR}} [\![\theta]\!]_{\mathcal{R}}$. Since τ strongly covers θ, Lemma 6 implies $\tau^{*} = \theta^{*}$. We thus conclude that $[\![\tau]\!]_{\mathcal{R}} \hookrightarrow_{\mathrm{BR}} [\![\theta]\!]_{\mathcal{R}}$ from the transitivity of the bounded repairability relation.

4.2 From repair to covering

We now sketch the proof of the "only if" direction of Theorem 1.

We fix for the rest of the section two stepwise hedge automata $\mathcal{R} = (\Sigma, Q, \delta_0, \delta, F)$ and $\mathcal{T} = (\Delta, Q', \delta_0', \delta', F')$ recognizing the restriction and the target languages, respectively. We assume that $\mathscr{L}(\mathcal{R})$ is bounded repairable into $\mathscr{L}(\mathcal{T})$ and we prove that every primitive synopsis tree of $\mathcal{R}$ is covered by some basic synopsis tree of $\mathcal{T}$.

The general idea of the proof is to associate with any primitive synopsis tree τ of $\mathcal{R}$ a suitable tree $t_\tau \in \mathscr{L}(\mathcal{R})$, called *witness tree* of τ, such that from any optimal repair of t_τ into $\mathscr{L}(\mathcal{T})$ one can extract a basic synopsis tree σ of $\mathcal{T}$ that covers τ. Intuitively, the witness tree t_τ is obtained from the primitive synopsis tree τ by replacing every non-trivial node x with a sufficiently large number of repetitions of a special context in $\mathscr{L}(\mathcal{R} \mid \tau(x))$, called *fingerprint context*. The number of repetitions of each fingerprint context will depend on the worst-case repair cost $N = \mathsf{dist}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T}))$. Using the definition of the witness tree t_τ and the assumption that t_τ can be repaired into some tree $s_\tau \in \mathscr{L}(\mathcal{T})$ with at most N edits, one can then argue that s_τ contains at least one copy of the fingerprint context associated with each non-trivial node x of τ and, furthermore, the arrangements of the occurrences of these fingerprints inside t_τ and inside s_τ coincide both with respect to the post-order relation and with respect to the ancestorship of the non-horizontal components. One finally looks at some run of $\mathcal{T}$ that accepts the tree s_τ: this run, together with the structure of the fingerprints inside s_τ, induces a basic synopsis tree σ of $\mathcal{T}$ and a coverability relation from τ to σ.

Below, we illustrate in more details the above arguments. We divide up the proof into defining the witness tree t_τ and building the cover from its repair.

Constructing the witness tree. Before constructing the witness tree, we give the following lemma, which defines what we call a fingerprint context of a component of $\mathcal{R}$. Basically, the lemma shows that given a component $X \in \mathsf{SCC}(\mathcal{R})$, one can find a context C_X that can be "pumped" inside the language $\mathscr{L}(\mathcal{R} \mid X)$ (i.e., $C_X \circ \ldots \circ C_X \in \mathscr{L}(\mathcal{R} \mid X)$) and such that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$ iff $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$, for any component $Y \in \mathsf{SCC}(\mathcal{T})$. We say that a context C is *cyclic* for a component X if there is a state $q \in X$ such that $q \in \delta(q, C)$.

Lemma 9. *For every $X \in \mathsf{SCC}(\mathcal{R})$, there exists a cyclic context $C_X \in \mathscr{L}(\mathcal{R} \mid X)$ such that, for every $Y \in \mathsf{SCC}(\mathcal{T})$,*

$$\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y) \qquad \textit{iff} \qquad C_X \in \mathscr{L}(\mathcal{T} \mid Y).$$

We associate with each component X a context C_X that satisfies Lemma 9, and call it a *fingerprint context* of X.

We now fix a primitive synopsis tree τ of $\mathcal{R}$ and we define the corresponding witness tree t_τ by using an induction on τ. In doing so, we will guarantee that $\delta(t_\tau) \cap \tau(\varepsilon) \neq \varnothing$, namely, that there is a run of $\mathcal{R}$ on the witness tree t_τ that reaches a state of the component at the root of τ. We omit the construction for the base case, where τ is a singleton, since it can be easily derived from what follows. Thus, we assume that X is the component at the root of τ and that τ_1 and τ_2 are the non-empty left and right sub-trees of τ. By induction hypothesis, we can denote by t_{τ_1} and t_{τ_2} the witness trees of τ_1 and τ_2, respectively. Moreover, we can fix a state q_1 (resp., q_2) in the non-empty set $\delta(t_{\tau_1}) \cap \tau_1(\varepsilon)$ (resp., $\delta(t_{\tau_2}) \cap \tau_2(\varepsilon)$). The construction of the witness tree t_τ is done in a bottom-up way using the three steps described below (the reader can also refer to Figure 10).

The first step consists of merging the two trees t_{τ_1} and t_{τ_2} into a single tree that can be parsed by the automaton $\mathcal{R}$ ending up in the component X. We know from the definition of primitive synopsis tree that τ respects the transition function of $\mathcal{R}$. In particular, this means that there exist some states $q' \in X$, $q_1' \in \tau_1(\varepsilon)$, and $q_2' \in \tau_2(\varepsilon)$ such that $q' \in \delta(q_1', q_2')$. Moreover, since q_1 and q_1' (resp., q_2 and q_2') belong to the same component at the root of τ_1 (resp., τ_2), there exist some contexts $C_1 \in \mathscr{L}(\mathcal{R} \mid \tau_1(\varepsilon))$ and $C_2 \in \mathscr{L}(\mathcal{R} \mid \tau_2(\varepsilon))$ such that $q_1' \in \delta(q_1, C_1)$ and $q_2' \in \delta(q_2, C_2)$.

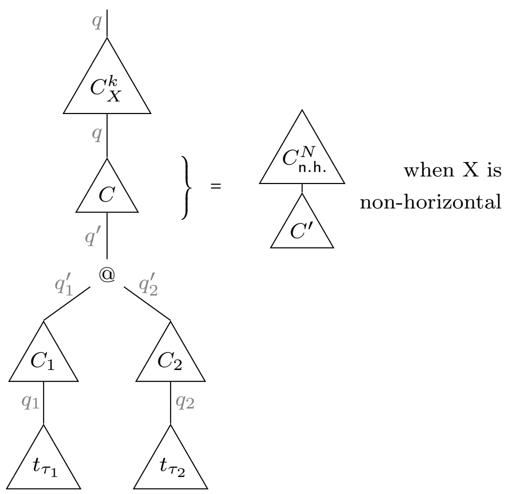

Figure 10: Construction of the witness tree t_τ.

This allows us to construct the tree

$$(C_1 \circ t_{\tau_1}) \ @ \ (C_2 \circ t_{\tau_2})$$

and claim that $\mathcal{R}$ can parse it and end up in state q' of component X.

The next step consists of prolonging the above tree in such a way that one can later attach repetitions of the fingerprint context C_X. This is done by identifying a "recurrent" state q such that $q \in \delta(q, C_X)$ (this state exists since C_X is cyclic) and then connecting it to the state q' using a suitable context $C \in \mathscr{L}(\mathcal{R} \mid X)$ such that $q \in \delta(q', C)$ (note that q and q' belong to the same component X). The resulting tree is of the form

$$C \ \circ \ \big((C_1 \circ t_{\tau_1}) \ @ \ (C_2 \circ t_{\tau_2})\big).$$

In order to avoid that an editing of the witness tree t_τ could modify the ancestorship of C_X with the nodes of the two sub-trees t_{τ_1} and t_{τ_2}, we further assume that, if X is a *non-horizontal* component, then the context C that is used for connecting q to q' is of the form $C^N_{\mathsf{n.h.}} \circ C'$, where $N = \mathsf{dist}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T}))$, $C', C_v \in \mathscr{L}(\mathcal{R} \mid X)$, and $C_{\mathsf{n.h.}}$ is a cyclic non-horizontal context. Recall that the ancestorship of non-horizontal contexts is preserved by the editing operations.

The last step consists of plugging in a sufficiently long repetition of the fingerprint context C_X. For this, we define $k = m \cdot (2N+1)$, where $N = \mathsf{dist}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T}))$ and m is the number of SCCs of $\mathcal{T}$. We then attach the k-fold repetition C_X^k of the fingerprint context C_X to the tree so far constructed, thus obtaining the witness tree

$$t_\tau \ = \ C_X^k \ \circ \ C \ \circ \ \big((C_1 \circ t_{\tau_1}) \ @ \ (C_2 \circ t_{\tau_2})\big).$$

Note that $\mathcal{R}$ can parse t_τ and end up in state $q \in X$. This shows that the invariant $\delta(t_\tau) \cap X \neq \varnothing$ is satisfied.

We remark that it may happen that $\delta(t_\tau) \cap F = \varnothing$ and hence $t_\tau \notin \mathscr{L}(\mathcal{R})$. Strictly speaking, this could violate the claim that one can repair t_τ into $\mathscr{L}(\mathcal{T})$ with at most N edits. However, from the assumption that $\mathcal{R}$ is trimmed it follows that there is a context C_F such that $\delta(q, C_F) \cap F \neq \varnothing$. This means that one can always prolong t_τ to obtain a tree inside the language $\mathscr{L}(\mathcal{R})$. From now on, we assume for the sake of simplicity that $t_\tau \in \mathscr{L}(\mathcal{R})$.

Building the covering from a repaired witness tree. We now turn to deriving a covering of τ by looking into the repair of t_τ. We fix, once and for all, some tree s_τ in the target language $\mathscr{L}(\mathcal{T})$ that is obtained by repairing t_τ with at most N edits.

Remember that the witness tree t_τ contains $k = m \cdot (2N+1)$ copies of each fingerprint context C_X, where $X = \tau(x)$ for any non-trivial node x of τ. As a consequence, the repaired tree s_τ must contain an m-fold repetition C_X^m of each finger context C_X. In the following, we will look at the occurrences of these repeated fingerprint contexts inside s_τ and compare their post-order and ancestor relationships with those induced by t_τ. For this we need some definitions.

Given a context C and a node x of a tree t, we say that C *occurs at node* x if there exist a context C' and a tree t' such that (i) t can be written as $C' \circ C \circ t'$ and (ii) x is the node where the sub-tree $C \circ t'$ of t hangs from. We denote an occurrence of a context C at a node x of a tree t by the pair (C, x). Furthermore, we say that two occurrences (C, x) and (C', x') of two contexts inside the same tree t are *non-overlapping* (resp., in *post-order relation*, *ancestor relation*) if $\{x\} \cdot \mathsf{nodes}(C) \ \cap \ \{y\} \cdot \mathsf{nodes}(C') \ = \ \varnothing$ (resp., $x \preccurlyeq^{\mathsf{post}}_t y$, $x \preccurlyeq^{\mathsf{anc}}_t y$).

The following lemma shows that the occurrences of the contexts C_X^m inside t_τ are in the same post-order relation as the corresponding occurrences inside s_τ, and similarly for the ancestor relation when X is a non-horizontal component.

LEMMA 10. *One can find a mapping f from the non-trivial nodes x of the primitive synopsis tree τ to the nodes $f(x)$ of the repaired witness tree s_τ such that:*

- *the context C_X^m, where $X = \tau(x)$, occurs at node $f(x)$ in s_τ, for all non-trivial nodes x of τ,*
- *all occurrences $(C_X^m, f(x))$, with $X = \tau(x)$ and x non-trivial node of τ, are pairwise non-overlapping,*
- *$x \preccurlyeq^{\mathsf{post}}_\tau y$ iff $f(x) \preccurlyeq^{\mathsf{post}}_{s_\tau} f(y)$, for all pairs of non-trivial nodes x, y of τ,*
- *$x \preccurlyeq^{\mathsf{anc}}_\tau y$ iff $f(x) \preccurlyeq^{\mathsf{anc}}_{s_\tau} f(y)$, for all pairs of non-trivial nodes x, y of τ, with $\tau(x)$ non-horizontal component.*

It only remains to show how to extract a basic synopsis tree σ that covers τ from an accepting run of $\mathcal{T}$ on s_τ.

Let ρ be a run of $\mathcal{T}$ that accepts s_τ, let x be a non-trivial node of τ with label X, and let $f(x)$ be the corresponding node in s_τ induced by the mapping f, as defined in Lemma 10. By the previous arguments, the context C_X^m occurs at node $f(x)$ in s_τ. Let y be the position of the placeholder $\bullet$ in the fingerprint context C_X and observe that C_X occurs m times at nodes ε, y, $y \cdot y$, ..., y^{m-1} of C_X^m. Let $y_{x,0} = f(x)$, $y_{x,1} = f(x) \cdot y$, ..., $y_{x,m-1} = f(x) \cdot y^{m-1}$, and $y_{x,m} = f(x) \cdot y^m$. Clearly, $(C_X, y_{x,0})$, ..., $(C_X, y_{x,m-1})$ are non-overlapping occurrences of the context C_X inside s_τ.

Consider now the states that occur at the $m+1$ nodes $y_{x,0}, y_{x,1}, \ldots, y_{x,m}$ of the run ρ. From the Pigeonhole Principle, we know that there exist two nodes $y_{x,i}$ and $y_{x,j}$, with $0 \le i < j \le m$, and a component Y of $\mathcal{T}$ such that both states $\rho(y_{x,i})$ and $\rho(y_{x,j})$ belong to Y. In fact, by the definition of strongly connected component, we can assume $j = i+1$ and hence $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$. Notice that, until this point, we have not used the property of fingerprint contexts (Lemma 9). In particular, the fact that C_X is a fingerprint context of X and $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$ implies that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$.

What we have just showed is that it is possible to find a mapping from any non-trivial node x of τ to a node $y_x = y_{x,i}$ of s_τ such that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$, where $X = \tau(x)$ and Y is the component of the state $\rho(y_x)$. Thanks to Lemma 10, we can also claim that, for all non-trivial nodes x, x' in τ,

- $x \neq x'$ implies $y_x \neq y_{x'}$,
- $x \preccurlyeq_\tau^{\mathsf{post}} x'$ iff $y_x \preccurlyeq_{s_\tau}^{\mathsf{post}} y_{x'}$,
- $x \preccurlyeq_\tau^{\mathsf{anc}} x'$ iff $y_x \preccurlyeq_{s_\tau}^{\mathsf{anc}} y_{x'}$, provided that the component $\tau(x)$ is non-horizontal.

We are now ready to define the basic synopsis tree σ that covers τ. The domain of σ coincides with the domain of s_τ, i.e., $\mathsf{nodes}(\sigma) = \mathsf{nodes}(s_\tau)$. The labeling $\sigma(x)$ of a node x of σ is given by the component X that contains the corresponding state $\rho(x)$, where ρ is a run of $\mathcal{T}$ that accepts s_τ. Notice that σ trivially satisfies the properties of a basic synopsis tree, as its labellings respects the transitions of $\mathcal{T}$ given by the run ρ. In a similar way, we can define the mapping λ that witnesses the coverability of τ by σ: for this we simply let λ map any non-trivial node x of τ to the node y_x of σ. The fact that λ satisfies Definition 3 follows easily from the properties described by the three items above (for instance, the fact that λ is injective follows from the first item). This completes the proof that τ is covered by a basic synopsis tree of $\mathcal{T}$.

5. COMPLEXITY RESULTS

In this section we will exploit the characterization given in Theorem 1 to obtain some complexity bounds for the bounded repair problem. We will first consider the case where the regular tree languages are represented by non-deterministic stepwise automata. Then, we will concentrate on less succinct and less expressive representations, such as deterministic stepwise automata and DTDs.

5.1 Languages defined by automata

Below, we show that the bounded repair problem between restriction and target languages represented by two non-deterministic stepwise automata $\mathcal{R}$ and $\mathcal{T}$ is in Π_2^{EXP} (i.e., coNEXPTIME$^{\mathrm{NEXPTIME}}$). To achieve this bound we analyze the size of a primitive synopsis tree τ of $\mathcal{R}$ and the minimal size of some basic synopsis tree of $\mathcal{T}$ that covers τ.

It follows easily from Definition 1 that any primitive synopsis tree of $\mathcal{R}$ has height less than the number of components of $\mathcal{R}$, and hence size at most $2^{|\mathsf{SCC}(\mathcal{R})|}$. As for the minimal size of the basic synopsis trees of $\mathcal{T}$ that cover a given primitive synopsis tree of $\mathcal{R}$, we can prove the following:

LEMMA 11. *Given two stepwise automata $\mathcal{R}$ and $\mathcal{T}$ and two synopsis trees $\tau \in \mathsf{PST}(\mathcal{R})$ and $\sigma \in \mathsf{BST}(\mathcal{T})$, if σ covers τ, then there is $\sigma' \in \mathsf{BST}(\mathcal{T})$ that covers τ and has size at most $2 \cdot |\tau| \cdot |\mathsf{SCC}(\mathcal{T})|$, where $|\tau|$ is the number of nodes of τ and $|\mathsf{SCC}(\mathcal{T})|$ is the number of components of $\mathcal{T}$.*

The above lemma, together with Theorem 1, gives a Π_2^{EXP} algorithm that receives two non-deterministic stepwise automata $\mathcal{R}$ and $\mathcal{T}$ and decides whether $\mathscr{L}(\mathcal{R})$ is bounded repairable into $\mathscr{L}(\mathcal{T})$. The algorithm universally guesses a primitive synopsis tree τ of $\mathcal{R}$ of size at most $2^{|\mathsf{SCC}(\mathcal{R})|}$, then it existentially guesses a basic synopsis tree σ of $\mathcal{T}$ of size at most $2^{|\mathsf{SCC}(\mathcal{R})|+1} \cdot |\mathsf{SCC}(\mathcal{T})|$ and a function λ from non-trivial nodes of τ to non-trivial nodes of σ, and finally it checks that λ is a covering of τ by σ (this amounts at deciding language inclusions, which can be done in exponential time [21]).

PROPOSITION 1. *The bounded repair problem between languages represented by non-deterministic stepwise automata is in Π_2^{EXP}.*

As for the complexity lower bound, we are only able to prove that the considered problem is EXPTIME-hard by using a straightforward reduction from the containment problem for non-deterministic stepwise automata [21]. Precisely, given two non-deterministic stepwise automata $\mathcal{R}$ and $\mathcal{T}$, one can decide whether $\mathscr{L}(\mathcal{R}) \subseteq \mathscr{L}(\mathcal{T})$ by first constructing two automata $\mathcal{R}^*$ and $\mathcal{T}^*$ that recognize the languages $\mathscr{L}(\mathcal{R}^*) = \{\#(t_1, ..., t_n) \mid n \in \mathbb{N},\ \forall 1 \le i \le n.\ t_i \in \mathscr{L}(\mathcal{R})\}$ and $\mathscr{L}(\mathcal{T}^*) = \{\#(t_1, ..., t_n) \mid n \in \mathbb{N},\ \forall 1 \le i \le n.\ t_i \in \mathscr{L}(\mathcal{T})\}$, respectively, where $\#$ is a fresh symbol not belonging to the alphabets of $\mathcal{R}$ and $\mathcal{T}$, and then checking whether $\mathscr{L}(\mathcal{R}^*)$ is bounded repairable into $\mathscr{L}(\mathcal{T}^*)$. Note that the latter condition holds iff $\mathscr{L}(\mathcal{R}) \subseteq \mathscr{L}(\mathcal{T})$.

PROPOSITION 2. *The bounded repair problem between languages represented by non-deterministic stepwise automata is EXPTIME-hard.*

5.2 Languages defined by DTDs

We now analyze the complexity of the bounded repair problem when the languages are represented by DTDs.

Recall that a DTD is a function mapping any letter a in the underlying alphabet to a regular expression $D(a)$ over the same alphabet. The language defined by a DTD D is the set of all unranked trees t such that (i) the root of t is labeled with some letter from a distinguished set of initial symbols and (ii) for every node $x \in \mathsf{nodes}(t)$, the sequence of labels of the children of x is a word inside the language $D(t(x))$. A DTD D is *non-recursive* if its dependency graph (i.e., the graph that connects a letter a to a letter b whenever b occurs in the language $D(a)$) is acyclic. A DTD D is *deterministic* if all its regular expressions $D(a)$ are 1-unambiguous [10]. This is also equivalent to saying that the *Glushkov automata* [10] that correspond to the regular expressions of the DTD are deterministic. From results in [12] (in particular, from Proposition 4 and Theorem 5) it follows that any deterministic DTD can be turned in polynomial time into an equivalent deterministic stepwise automaton.

As a first remark, we recall that the containment problem for non-deterministic DTDs is PSPACE-complete. The lower bound follows easily from the PSPACE-hardness of containment of regular expressions [23]. The upper bound is a folklore result: given two DTDs D and D', one can decide whether the language defined by D is contained in the language defined by D' by first removing useless rules and then checking that, for every letter a in the alphabet of D, the language $D(a)$ is contained in the language $D(a')$.

Following the same idea in the proof of Proposition 2, one can reduce the containment problem for regular expressions to the bounded repairability problem for languages defined by non-recursive non-deterministic DTDs, thus showing that the latter problem is at least PSPACE-hard. Proposition 3 below gives a stronger result by showing that PSPACE-hardness holds also in the presence of non-recursive *deterministic* DTDs. The proof of this result is technical and uses a reduction from the corridor tiling problem [8].

PROPOSITION 3. *The bounded repair problem between languages represented by deterministic DTDs is PSPACE-hard, even for non-recursive DTDs.*

For DTDs, even deterministic ones, we do not have a better upper bound than in the general case. Recall that containment of deterministic stepwise automata (and hence deterministic DTDs as well) is in PTIME. It is thus conceivable that a PSPACE upper bound for the bounded repair problem exists for deterministic DTDs. As we are not able to provide such a tight complexity bound, we can only reuse Proposition 1 to claim that the complexity of the bounded repair problem for deterministic DTDs is between PSPACE and Π_2^{EXP}.

For non-recursive DTDs we can do better, since they define languages of trees of bounded height, and can be coded by regular word languages (see, for instance, [20]). The edit operations on trees translate to edit operations on the word encodings, and hence we can apply the fact that bounded repairability for word languages recognized by deterministic finite automata is in coNP [6]. Because there is an exponential blow-up in the encoding, this gives us a coNEXPTIME upper bound for non-recursive DTDs. Again, the best lower bound we have for non-recursive DTDs is PSPACE.

We now consider a specialization of the problem where the alphabet Σ of the restriction language is fixed. We show that, in this case, the problem is between PSPACE and EXPTIME for languages represented by non-deterministic DTDs, and between coNP and Π_2^{P} (i.e., $\mathrm{coNP}^{\mathrm{NP}}$) for languages represented by deterministic DTDs.

Let us first discuss the complexity upper bounds. Suppose that D is a DTD defining a restriction language over the fixed alphabet Σ. A close inspection to the translation from DTDs to stepwise automata [12] discloses the following crucial property:

LEMMA 12. *Given a non-deterministic (resp., deterministic) DTD D that defines a restriction language R over an alphabet Σ, one can compute in polynomial time a non-deterministic (resp., a deterministic) stepwise automaton $\mathcal{R} = (\Sigma, Q, \delta_0, \delta, F)$ that recognizes R and whose state space can be partitioned into $k \leq 2|\Sigma|$ subsets $Q_1, \ldots, Q_k$ such that*

- *every component of $\mathcal{R}$ is contained in some set Q_i,*
- *for all states $q_1, q_2, q \in Q$, if $q \in \delta(q_1, q_2)$ and q_2 and q are in different components, then $q_2 \in Q_i$ and $q \in Q_j$ for some $1 \leq i < j \leq k$.*

As an example, the automaton $\mathcal{R}$ described in Example 3 is a deterministic stepwise automaton whose state space can be partitioned into 9 sets that satisfy the first part of the claim: $Q_1 = \{p_0^a\}$, $Q_2 = \{p_1^a\}$, $Q_3 = \{p_0^b\}$, $Q_4 = \{p_1^b\}$, $Q_5 = \{p_0^c\}$, $Q_6 = \{p_0^d\}$, $Q_7 = \{p_1^d\}$, $Q_8 = \{p_0^r, p_1^r\}$, $Q_9 = \{p_2^r\}$.

Lemma 12 above implies that any path in the transition graph of $\mathcal{R}$ (see, for instance, the left-hand side graph of Figure 4) traverses at most $2|\Sigma| - 1$ *vertical* edges that connect pairs of states in different components. As a consequence, any primitive synopsis tree of $\mathcal{R}$ has size at most $|Q|^{2^{|\Sigma|}}$, i.e., polynomial in the size of $\mathcal{R}$ when Σ is fixed.

Putting together Lemma 12, Lemma 11, and Theorem 1 one obtains an EXPTIME (resp., a Π_2^{P}) algorithm that decides whether $R \hookrightarrow_{\mathrm{BR}} T$, where R and T are languages defined by non-deterministic (resp., deterministic) DTDs and R is over a fixed alphabet Σ. The algorithm has the same structure of the one presented in the previous sub-section. Namely, it translates the input DTDs into equivalent stepwise automata $\mathcal{R}$ and $\mathcal{T}$, then it guesses universally a primitive synopsis tree τ of $\mathcal{R}$ of polynomial size and existentially a basic synopsis tree σ of $\mathcal{T}$ of polynomial size, together with a function λ from non-trivial nodes in τ to non-trivial nodes in σ, and finally it checks that λ is a covering of τ by σ. As previously mentioned, the last step of the algorithm can be performed in either exponential time or polynomial time, depending on whether the input DTDs are non-deterministic or deterministic. Note that the complexity upper bounds do not increase if the target languages are given directly by non-deterministic/deterministic stepwise automata.

PROPOSITION 4. *The bounded repair problem between a restriction language represented by a non-deterministic (resp., deterministic) DTD over a fixed alphabet and a target language represented by a non-deterministic (resp., deterministic) stepwise automaton over an arbitrary alphabet is in EXPTIME (resp., in Π_2^P).*

Finally, we show that even strong restrictions, including fixing both alphabets, cannot get us below PSPACE in the non-deterministic case. We have already discussed how the containment problem for regular expressions can be reduced to the bounded repair problem for languages defined by non-recursive *non-deterministic* DTDs. The same reduction holds when we fix the alphabets, thus showing that the bounded repair problem between non-recursive non-deterministic DTDs over fixed alphabets is still PSPACE-hard. We also provide a coNP lower bound for the analogous problem when the languages are represented by non-recursive *deterministic* DTDs over fixed alphabets. This lower bound follows quite easily from a reduction from the validity problem for propositional formulas in disjunctive form (i.e., the dual of the SAT problem). The idea is to encode in the restriction language all the possible valuations for the propositional variables and then restrict the target language to consist only of encodings of valuations that satisfy at least one clause of the propositional formula. A similar reduction was given in [6] for languages of words recognized by deterministic finite automata. The additional complication here is that we have to fix the restriction and the target alphabets; however, the reduction is still possible by encoding the valuation of each variable with a block of nodes labeled over a binary alphabet.

Proposition 5. *The bounded repair problem between languages represented by non-recursive non-deterministic (resp., deterministic) DTDs with both restriction and target alphabets fixed is PSPACE-hard (resp., coNP-hard).*

6. THE UNRESTRICTED CASE

In this section we consider the so-called unrestricted case of the bounded repairability problem, namely, a variant of the problem where the restriction language is assumed universal (i.e., equal to $\mathcal{T}_\Sigma$) and the target language is represented by a stepwise automaton $\mathcal{T}$.

We recall the assumption that any stepwise automaton $\mathcal{T}$ is trimmed (i.e., every state of $\mathcal{T}$ appears in some accepting run of $\mathcal{A}$ on some input tree). Under this assumption, we say that an automaton $\mathcal{T}$ is *complete over* Σ if for every tree $t \in \mathcal{T}_\Sigma$ there is a (possibly non-accepting) run of $\mathcal{T}$ on t.

In this section we also make use of *deterministic visibly pushdown transducers* [18, 2] as suitable devices that transform unranked trees in a streaming fashion, namely, that receive the serialized version of an unranked tree (i.e., a well-formed sequence of opening and closing tags corresponding to the pre-order traversal of a tree) and output the serialized version of another unranked tree. By a slight abuse of notation we identify unranked trees with their serializations. The following result gives equivalent conditions for bounded repairability in the unrestricted case.

Proposition 6. *Given an alphabet Σ and an automaton $\mathcal{T} = (\Delta, Q, \delta_0, \delta, F)$, the following conditions are equivalent:*

- *$\mathcal{T}_\Sigma$ is bounded repairable into $\mathscr{L}(\mathcal{T})$,*
- *$\mathcal{T}$ is complete over Σ,*
- *there exist $k \in \mathbb{N}$ and a deterministic visibly pushdown transducer that receives any unranked tree t over Σ and outputs an unranked tree t' such that $\mathsf{dist}(t, t') \leq k$ and $\mathsf{ext}(t') \in \mathscr{L}(\mathcal{T})$.*

From the above characterization one can derive a polynomial-time algorithm that decides whether $\mathcal{T}_\Sigma$ is bounded repairable into $\mathscr{L}(\mathcal{T})$, when $\mathcal{T}$ is given by a *deterministic* stepwise automaton. For this it is sufficient to turn $\mathcal{T}$ into a trimmed deterministic automaton $\mathcal{T}' = (\Sigma, Q', \delta'_0, \delta', F')$ over Σ and then check that (i) for every symbol $a \in \Sigma$, $\delta'_0(a) \neq \varnothing$ and (ii) for every pair of states $q_1, q_2 \in Q'$, $\delta'(q_1, q_2) \neq \varnothing$. When the target language is represented by a *non-deterministic* stepwise automaton $\mathcal{T}$, the complexity increases to EXPTIME: one can simply determinize $\mathcal{T}$ and then use the decision procedure for the deterministic case.

As one could expect, the above complexity bounds (i.e., PTIME for deterministic stepwise automata and EXPTIME for non-deterministic stepwise automata) are tight – hardness proofs can be derived from reductions of the emptiness and universality problems, respectively, on the corresponding classes of automata.

Proposition 7. *The bounded repair problem in the unrestricted case when the target language is represented by a non-deterministic (resp., deterministic) stepwise automaton is EXPTIME-complete (resp., PTIME-complete).*

It is worth remarking some similarities and differences between the considered problem and the analogous unrestricted bounded repair problem for regular languages of words [6]. First of all, the characterization given in Proposition 6 implies that, in the unrestricted case only, if the universal tree language $\mathcal{T}_\Sigma$ is bounded repairable into a regular tree language T, then the repair strategy can be implemented by a deterministic visibly pushdown transducer that works directly on serializations of unranked trees. This is reminiscent of the equivalence between the unrestricted repair problems for regular word languages in the non-streaming and in the streaming settings. We also remark that bounded repairability for tree languages is a much stronger property than that for word languages. In [6] (Corollary 20), it is shown that for every alphabet Σ and every regular word language T, Σ^* is bounded repairable into T or into its complement T^{C}. When considering languages of trees the situation is quite different: for every alphabet Σ, there are regular tree languages T such that $\mathcal{T}_\Sigma$ is bounded repairable neither into T nor into its complement T^{C}.

7. CONCLUSIONS AND RELATED WORK

In this paper we present the first algorithm for determining whether one regular tree language can be edited to another regular tree language with a finite number of edits.

Edit distance between trees has been extensively studied [7], and several polynomial-time algorithms exist based on dynamic-programming. The problem of computing the distance between a tree and a schema has also been studied [9]. Regarding similarity between schemas, syntactic similarity between schemas were studied in [14] and a similarity set-based measure was proposed in [3], but both works did not consider edit operations between trees. Approximation of regular tree languages by single-type regular tree languages has been studied in [16] were the setting is restricted to a particular class of tree languages (single-type) and the approximation is modeled by language containment. To the best of our knowledge, the problem of editing all trees in a schema to trees in another schema has not been studied explicitly. In the word case, the analogous notion has been studied only recently [6].

We leave several problems open; most importantly, we have a gap in the complexity for the repair problem. We hope to close this by lowering the complexity to EXPTIME.

Bounded repairability is a strong notion of inclusion between schemas: there may be two schemas where the number of edits is unbounded, but is small as a percentage of the tree size. In [5] we showed a way to compute this "percentage of edits" in the word case. We would like to generalize this to the setting of trees. It would also be interesting to consider restricted processors that work in an online fashion – in the word case, the requirement for a repair to be "streaming" has been studied [6], where the streaming notion is captured by property that the repairs can be generated by a finite state transducer without lookahead. For trees, it is not clear what model should be used as a "streaming repair processor". Theorem 6 shows that in the unrestricted case it always suffices to repair with a visibly pushdown transducer, but in the general case VPTs maybe too restrictive.

Acknowledgments. We would like to thank Michael Benedikt for the many helpful remarks on the paper. The first two authors were supported by the EPSRC (UK) grant EP/G004021/1 and by the EU project FOX (FP7-ICT-233599). The third author has been partially supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the Contrat de Projets Etat Region (CPER) 2007-2013, and Codex project ANR-08-DEFIS-004.

8. REFERENCES

[1] Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41, 2009.

[2] Rajeev Alur and Parthasarathy Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009.

[3] Timos Antonopoulos, Floris Geerts, Wim Martens, and Frank Neven. Generating, sampling and counting subclasses of regular tree languages. In *ICDT*, pages 30–41, 2011.

[4] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[5] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. The cost of traveling between languages. In *ICALP*, pages 234–245, 2011.

[6] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Regular repair of specifications. In *LICS*, pages 335–344, 2011.

[7] Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, 2005.

[8] Peter Van Emde Boas. The convenience of tilings. In *Complexity, Logic and Recursion Theory*, volume 187, pages 331–363, 1997.

[9] Utsav Boobna and Michel de Rougemont. Correctors for XML data. In *XSym*, pages 97–111, 2004.

[10] Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Inf. Comput.*, 140(2):229–253, 1998.

[11] Julien Carme, Joachim Niehren, and Marc Tommasi. Querying unranked trees with Stepwise Tree Automata. In *RTA*, pages 105–118, 2004.

[12] Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Efficient inclusion checking for deterministic tree automata and XML schemas. *Inf. Comput.*, 207(11):1181–1208, 2009.

[13] David Fallside and Priscilla Walmsley. *XML Schema Part 0: Primer Second Edition.* W3C Recommendation, October 2004.

[14] Wenfei Fan and Philip Bohannon. Information preserving XML schema embedding. *ACM Trans. Database Syst.*, 33(1), 2008.

[15] Sergio Flesca, Filippo Furfaro, Sergio Greco, and Ester Zumpano. Querying and repairing inconsistent XML data. In *WISE*, pages 175–188, 2005.

[16] Wouter Gelade, Tomasz Idziaszek, Wim Martens, and Frank Neven. Simplifying XML schema: single-type approximations of regular tree languages. In *PODS*, pages 251–260, 2010.

[17] Wim Martens and Joachim Niehren. On the minimization of XML schemas and tree automata for unranked trees. *J. Comput. Syst. Sci.*, 73(4):550–583, 2007.

[18] Jean-François Raskin and Frédéric Servais. Visibly pushdown transducers. In *ICALP*, pages 386–397, 2008.

[19] Thomas Schwentick. Automata for XML - a survey. *J. Comput. Syst. Sci.*, 73(3):289–315, 2007.

[20] Luc Segoufin and Victor Vianu. Validating streaming XML documents. In *PODS*, pages 53–64, 2002.

[21] Helmut Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.*, 19(3):424–437, 1990.

[22] Slawomir Staworko and Jan Chomicki. Validity-sensitive querying of XML databases. In *EDBT Workshops*, pages 164–177, 2006.

[23] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time. In *STOC*, pages 1–9, 1973.