HAL
open science

Which DTDs are Streaming Bounded Repairable?

Gabriele Puppis, Cristian Riveros, Pierre Bourhis

To cite this version:

Gabriele Puppis, Cristian Riveros, Pierre Bourhis. Which DTDs are Streaming Bounded Repairable?. International Conference on Database Theory (ICDT), Mar 2013, Genoa, Italy. pp.57-68, 10.1145/2448496.2448505 . hal-00829423

HAL Id: hal-00829423

https://hal.science/hal-00829423

Submitted on 3 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Which DTDs are streaming bounded repairable?

Pierre Bourhis
University of Oxford
pierre.bourhis@cs.ox.ac.uk

Gabriele Puppis
LaBRI / CNRS
gabriele.puppis@labri.fr

Cristian Riveros
University of Oxford
cristian.riveros@cs.ox.ac.uk

ABSTRACT

Integrity constraint management concerns both checking whether data is valid and taking action to restore correctness when invalid data is discovered. In XML the notion of valid data can be captured by schema languages such as Document Type Definitions (DTDs) and more generally XML schemas. DTDs have the property that constraint checking can be done in streaming fashion. In this paper we consider when the corresponding action to restore validity – repair – can be done in streaming fashion. We formalize this as the problem of determining, given a DTD, whether or not a streaming procedure exists that transforms an input document so as to satisfy the DTD, using a number of edits independent of the document. We show that this problem is decidable. In fact, we show the decidability of a more general problem, allowing a more general class of schemas than DTDs, and requiring a repair procedure that works only for documents that are already known to satisfy another class of constraints. The decision procedure relies on a new analysis of the structure of DTDs, reducing to a novel notion of game played on pushdown systems associated with the schemas.

1. INTRODUCTION

A basic problem in data management is to ensure that data is valid – that is, satisfies all integrity constraints associated with a schema. A particularly attractive feature of XML documents is that the notion of valid data can be captured in an expressive yet highly intuitive language – that of Document Type Definitions (DTDs) and more generally XML schemas [10]. DTDs and XML schemas are heavily used in practice, and the basic validation task can be performed in a one-pass process using limited memory, that is, they admit *streaming validation* [6, 8, 13].

For many XML-based applications, the desired behaviour when data integrity constraints fail is not simply to raise an error, but to fix it. The most obvious example of this is for HTML. Mal-formed or non-conformant HTML is more the rule than the exception, and browsers react to non-conformant documents by simply changing them to conformant ones. That is, *repair* is a well-accepted procedure for XML-based data.

In this paper, we tackle the question of which schemas admit 'streaming repair', an analogue of streaming validation. Intuitively, a streaming repair is a procedure that inserts, deletes, or modifies document content while reading the document in pre-order fashion, producing an output that satisfies the constraint. Clearly, there is a vacuous streaming repair that simply deletes the entire document and inserts a new document that satisfies the constraints. The unacceptability of such a repair strategy stems from the fact that the number of changes it makes to a document is proportional to its size. Clearly, we would like a stream repair processor to make a 'small' number of changes to the input. We formalize this requirement via the notion of a *bounded repair strategy* [12], i.e. a repair strategy that makes a maximum number of repairs that is finite and independent of the input document. Although less stringent notions of 'small repair' can be demanded (e.g. by requiring that a small percentage of the document be repaired, analogous to the notion explored for words in [1]) we feel this is a natural starting point for the exploration of streaming repair.

We follow our previous work in the non-streaming setting [12] by looking at the general scenario where there is a restriction schema which the document is assumed to satisfy and a target schema that we wish to enforce. We study the problem of determining whether there is a stream processor that will (i) ensure that any document satisfying the restriction is repaired to a document satisfying the target and (ii) performs a number of edits that is uniformly bounded and independent of the document. We consider only the tag structure of the documents, ignoring string content. Moreover, the edits we consider are the standard tree edits [3]. Our prior work [12] gave a characterization and decision procedure for determining whether a bounded repair strategy exists in the non-streaming setting. In this work we give a characterization and decision procedure for determining whether a streaming bounded repair strategy exists, in the important case of DTD schemas, and more generally of 'deterministic top-down schemas' (the formal definition is given in Section 2, but for now let us only remark that this is a class that subsumes not only DTDs, but also XSDs).

The solution to the streaming bounded repair problem is

© 2013 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the national government of France. As such, the government of France retains a non-exclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
EDBT/ICDT 2013, March 18–22, 2013, Genoa, Italy
Copyright 2013 ACM 978-1-4503-1598-2/13/03. . . $15.00

challenging both from the point of view of giving a characterization, and showing that it is both effective and correct. The first part of the solution is adapted from our prior work [12]: we associate a graph with each schema, and then look at the corresponding notion of connected component; such components represent 'repeatable behaviours', namely, families of trees that can be created by a certain kind of pumping operation. Our characterization will involve a novel game played on stacks of components in the two graphs, with one player, called Generator, managing the stack for the restriction and corresponding to generation of families of trees satisfying the restriction schema, and the other player, called Repairer, managing the stack for the target. Repairer needs to play in such a way that a certain relation holds between the components on the top of the stacks, corresponding to containment of a set of trees. The characterization theorem says that a streaming repair with uniformly bounded cost is possible exactly when Repairer has a winning strategy in the game. The possible moves of Generator will be restricted in a way that ensures finiteness of the game, and thus decidability of a winner.

Both directions of the proof of our characterization are highly non-trivial. In one direction, we manufacture an effective document repair transducer from a winning strategy for Repairer. In the other, we use a repair transducer of uniformly bounded cost to get a winning strategy for Repairer.

With our characterization in hand, we are able to give an EXPTIME upper bound on the complexity of determining the existence of a streaming repair strategy of uniformly bounded cost. We complement this with a matching lower bound, and go on to isolate subcases where the complexity decreases (to PSPACE).

Organization. Section 2 gives basic definitions, including the notions of schema and repair considered in the paper. Section 3 states the streaming repair problem formally, and gives examples that explain the difficulties of the problem and motivate its solution. Section 4 states our main characterization theorem, which relies on defining a new kind of game played on stacks of components of the restriction and target automata. An overview of the main ingredients of the proof of correctness is also given. Section 5 considers the consequences of the characterization theorem for complexity, while Section 6 gives conclusions and discusses future work.

2. PRELIMINARIES

We adopt the usual notations for strings, denoting, for instance, a finite alphabet by Σ, the empty string by ε, and the concatenation of two strings by $u \cdot v$. We will often deal with sequences of natural numbers, usually denoted by $\vec{i}$, and stacks, usually denoted by $\vec{x}$.

2.1 Trees, contexts, and their serializations

In this paper we work with finite unranked ordered trees and forests whose nodes are labelled over fixed finite alphabets. Formally, an *unranked tree/forest* can be seen a function t mapping non-empty sequences of positive natural numbers to symbols from a finite alphabet (e.g. Σ), where the domain of t, denoted $\mathsf{nodes}(t)$, satisfies the following closure under lexicographic order: if $\vec{i} \cdot j \in \mathsf{nodes}(t)$, then $\vec{i} \cdot k \in \mathsf{nodes}(t)$ for all $1 \leq k \leq j$, and either $\vec{i} = \varepsilon$ or $\vec{i} \in \mathsf{nodes}(t)$. Notice that the roots of a forest are represented by singleton sequences (in particular, the empty sequence does not belong to the domain). Given an unranked tree/forests t, we write $\vec{i} \in \mathsf{nodes}(t)$ to denote an arbitrary node of t and $t(\vec{i})$ to denote the label of $\vec{i}$ in t. The set of all finite unranked trees labelled over Σ is denoted by $\mathcal{T}_\Sigma$. We often describe unranked trees by means of pictures or unranked terms such as $a(b, b, b)$.

It is known that trees, and more generally forests, can be represented by their serializations (XML Documents). Formally, given an alphabet Σ, we introduce a disjoint copy of Σ of the form $\bar{\Sigma} = \{\bar{a} \ : \ a \in \Sigma\}$. The elements in Σ represent the *opening tags* of the serializations, while the elements in $\bar{\Sigma}$ represent the *closing tags*. The *serialization* $\hat{t}$ of a tree t is defined recursively by $\hat{t} = \varepsilon$ if t is empty, and by $\hat{t} = a \cdot \hat{t}_1 \cdot \ldots \cdot \hat{t}_n \cdot \bar{a}$ if $t = a(t_1, \ldots, t_n)$. The serialization of a forest is the concatenation of the serializations of its trees. Clearly, every serialization of a tree/forest produces a *well-matched* string over $\Sigma \uplus \bar{\Sigma}$ and vice-versa.

Next we define contexts, also known as 'pointed trees', which are trees/forests with a distinguished hole. We use a special symbol $\bullet$, not in the alphabet Σ, to label the hole of a context – this acts as a placeholder for substituting trees/forests in the context. Formally, a *context* over Σ is a tree or a forest labelled over $\Sigma \uplus \{\bullet\}$, where $\bullet$ occurs exactly once in a *leaf that has no right sibling* (this restriction is motivated by the tree automaton model that we will introduce later). Examples of contexts are $a(b, b, \bullet)$ and $a(b, b)\ \bullet$. On the other hand, $a(b, \bullet, b)$ is not a valid context in our setting. We denote by $\mathcal{C}_\Sigma$ the set of all contexts over the alphabet Σ.

Note that the serialization $\hat{C}$ of a context C is a word that contains a single occurrence of the substring $\bullet\bar{\bullet}$. We will denote by $\hat{C}^{\mathsf{prefix}}$ (resp. $\hat{C}^{\mathsf{suffix}}$) the prefix (resp. suffix) of $\hat{C}$ that ends immediately before the occurrence of $\bullet$ (resp. that starts immediately after the occurrence of $\bar{\bullet}$).

Given a context C and a tree t, we denote by $C \circ t$ the tree obtained from the substitution of $\bullet$ in C by t. The composition $C \circ C'$ of two contexts C and C' is defined similarly and results again in a context. We observe that the composition of contexts with trees/contexts has analogous operations on serializations, that is, $\widehat{C \circ t} = \hat{C}^{\mathsf{prefix}} \cdot \hat{t} \cdot \hat{C}^{\mathsf{suffix}}$ and $\widehat{C \circ C'} = \hat{C}^{\mathsf{prefix}} \cdot \hat{C}' \cdot \hat{C}^{\mathsf{suffix}}$.

2.2 Top-down tree automata

In this paper we make use of languages of unranked trees, such as those defined by the structural components of DTDs and XSD schemas [11, 10]. We work with 'deterministic top-down schemas' [9, 5], which generalize DTDs and can be seen as typing systems in which the type associated with each internal node of a tree depends uniquely on the type of the parent and the type of the left sibling (if this exists). Equivalently, one can think of these schemas as deterministic top-down binary tree automata running on the standard first-child-next-sibling encoding of an unranked tree.

To avoid switching every time between unranked trees and their encodings, we define the runs of our automata directly on unranked trees and unranked forests. Given an unranked tree/forest t, we denote by $\mathsf{nodes}^+(t)$ the *extended domain*

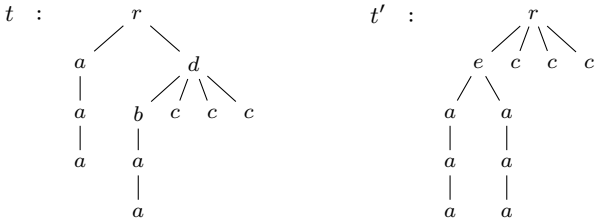


Figure 1: Two unranked trees t and t'.

of t, which is defined as the set that contains all nodes of t and all sequences $\vec{i}\cdot j\cdot 1 \in \mathsf{nodes}^+(t)$ and $\vec{i}\cdot(j+1) \in \mathsf{nodes}^+(t)$, with $\vec{i}\cdot j \in \mathsf{nodes}(t)$. Intuitively, $\mathsf{nodes}^+(t)$ is the extension of the domain of t that results from adding a new child to each leaf and a new sibling to each node with no right sibling.

Definition 1. *A* determinitic top-down tree automata *is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, where*

- *Σ is a finite alphabet,*
- *Q is a finite set of states,*
- *$\delta : Q \times \Sigma \rightharpoonup Q \times Q$ is a partial transition function,*
- *$q_0 \in Q$ is an initial state,*
- *$F \subseteq Q$ is a set of final states.*

A run *of $\mathcal{A}$ on t is any Q-labelled tree/forest ρ such that $\mathsf{nodes}(\rho) = \mathsf{nodes}^+(t)$ and $\delta\big(\rho(\vec{i}\cdot j), t(\vec{i}\cdot j)\big) = \big(\rho(\vec{i}\cdot j\cdot 1), \rho(\vec{i}\cdot(j+1))\big)$ for all $\vec{i}\cdot j \in \mathsf{nodes}(t)$. A run ρ is said to be* successful *if $\rho(1) = q_0$ and $\rho(\vec{i}) \in F$ for all nodes $\vec{i} \in \mathsf{nodes}(\rho) \setminus \mathsf{nodes}(t)$. The language recognized by $\mathcal{A}$ is the set $\mathscr{L}(\mathcal{A})$ of all trees $t \in \mathcal{T}_\Sigma$ that induce a successful run of $\mathcal{A}$.*

In the sequel, we will work with *trimmed* automata only, namely, we assume that all states of our automata appear in some successful run. Because useless states of automata can be detected and removed in linear time, this assumption will have no impact on our complexity results.

Example 1. *As a running example, consider the DTDs:*

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

Figure 1 gives examples of two unranked trees satisfying the DTDs D and D'.

The following are the transition rules for two deterministic top-down tree automata $\mathcal{R}$ and $\mathcal{T}$ that recognize the languages defined by D and D', respectively (p_0^r and q_0^r are the initial states and the final states are underlined):

$$
\begin{array}{llllll}
\mathcal{R}: & p_0^r \xrightarrow{r} p_0^a\ \underline{f} & & \mathcal{T}: & q_0^r \xrightarrow{r} q_0^e\ \underline{f} \\
 & p_0^a \xrightarrow{a} \underline{p_1^a}\ p_0^d & & & q_0^e \xrightarrow{e} q_0^a\ \underline{q_0^c} \\
 & \underline{p_1^a} \xrightarrow{a} \underline{p_1^a}\ \underline{f} & & & q_0^a \xrightarrow{a} \underline{q_1^a}\ \underline{q_1^a} \\
 & p_0^d \xrightarrow{d} p_0^b\ \underline{f} & & & \underline{q_1^a} \xrightarrow{a} \underline{q_1^a}\ \underline{f} \\
 & p_0^b \xrightarrow{b} \underline{p_1^a}\ \underline{p_0^c} & & & \underline{q_0^c} \xrightarrow{c} \underline{f}\ \underline{q_0^c} \\
 & \underline{p_0^c} \xrightarrow{c} \underline{f}\ \underline{p_0^c} & & &
\end{array}
$$

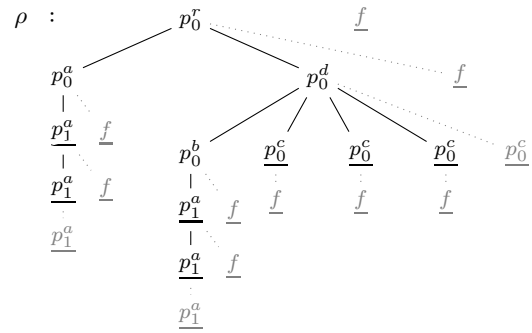


Figure 2: Run of top-down tree automaton $\mathcal{R}$ on t.

Figure 2 presents the successful run of the automaton $\mathcal{R}$ on the tree t of Figure 1 (black nodes correspond to elements of t, while gray nodes have no corresponding element in t).

As usual for finite automata on words, we derive from the transition function δ of a top-down tree automaton $\mathcal{A}$ a transition function $\hat{\delta}$ on contexts. Formally, we define $\hat{\delta}$ as the partial function from $Q \times \mathcal{C}_\Sigma$ to Q by letting $\hat{\delta}(q, C) = q'$ iff the context C is accepted by the automaton $\mathcal{A}$ where the initial state is replaced by q and the transition function δ is extended in such a way that $\delta(q', \bullet) = (f, f)$, with f being a new final state.

2.3 Tree edits over serializations

We briefly recall the definitions of some standard edit operations on unranked trees [3]. 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 operations that are used to define the edit-distance between unranked ordered trees [3, 14]. Note that the operation of *relabelling* a node in an unranked tree, which is sometimes used as a standard edit operation, is subsumed by the previous two operations.

For the setting considered in this paper, we want to edit trees by editing their serializations – by deleting and inserting tags – in such a way that the resulting operations implement tree edit operations. We can code a sequence of string edits via another string, called alignment. Formally, given two strings $u \in \Sigma^*$ and $v \in \Delta^*$, an *alignment* of u and v is any string e over the alphabet $(\Sigma \uplus \{\varepsilon\}) \times (\Delta \uplus \{\varepsilon\})$ whose projection over the first (resp. second) component gives u (resp. v). The *cost* of an alignment e, denoted

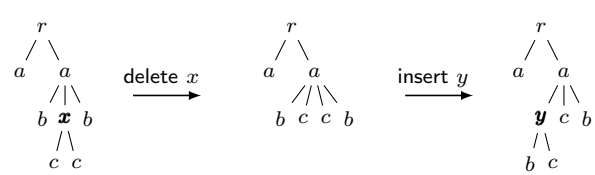


Figure 3: Edit operations on unranked trees.

$\mathsf{cost}(e)$, is the number of occurrences in e that are not of the form (a,a), with $a \in \Sigma$, nor of the form $(\varepsilon,\varepsilon)$. The *edit-distance* (or *Levenshtein distance*) between two strings $u \in \Sigma^*$ and $v \in \Delta^*$, denoted $\mathsf{dist}(u,v)$, is defined as the minimum cost of all possible alignments of u and v [16]. As an example, the string $\binom{a}{a}\binom{b}{\varepsilon}\binom{c}{c}\binom{d}{d}\binom{\varepsilon}{e}$ is an alignment between the strings *abcd* and *acde* that achieves optimal cost 2 (hence $\mathsf{dist}(abcd, acde) = 2$).

As we mentioned earlier, we are interested in repairing serializations of unranked trees and, more specifically, in alignments that can be directly translated into editing operations on the corresponding trees with similar costs. This is captured by the notion of *tree edit alignment* between well-matched words. Let us fix some well-matched words $u \in (\Sigma \uplus \bar{\Sigma})^*$ and $v \in (\Delta \uplus \bar{\Delta})^*$ and an alignment e between them. We first define two matching relations $\sim_u$ and $\sim_v$ between positions of e as follows: given $1 \le i < j \le |e|$, we write $i \sim_u j$ (resp. $i \sim_v j$) iff the infix $e[i,j]$ projected onto the first (resp. second) components is a well-matched word. We then say that e is a *tree edit alignment* if the following implications hold:

- if $e(i) = (a,a)$, then there is $1 \le j \le |e|$ such that $e(j) = (\bar{a},\bar{a})$, $i \sim_u j$, and $i \sim_v j$,
- if $e(j) = (\bar{a},\bar{a})$, then there is $1 \le i \le |e|$ such that $e(i) = (a,a)$, $i \sim_u j$, and $i \sim_v j$.

Example 2. *Given two unranked trees $t = a(a(b),c)$ and $t' = a(a(c),b)$ and their serializations $\hat{t} = aab\bar{b}\bar{a}c\bar{c}\bar{a}$ and $\hat{t}' = aac\bar{c}\bar{a}b\bar{b}\bar{a}$, the following are two possible alignments between $\hat{t}$ and $\hat{t}'$:*

$$e = \binom{a}{a}\binom{a}{a}\binom{b}{c}\binom{\bar{b}}{\bar{c}}\binom{\bar{a}}{\bar{a}}\binom{c}{b}\binom{\bar{c}}{\bar{b}}\binom{\bar{a}}{\bar{a}}$$

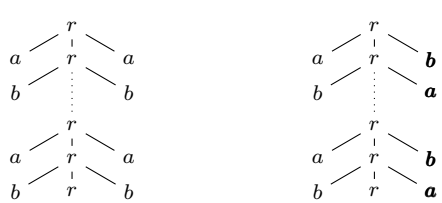
$$e' = \binom{a}{a}\binom{a}{a}\binom{\varepsilon}{c}\binom{\varepsilon}{\bar{c}}\binom{\varepsilon}{\bar{a}}\binom{b}{b}\binom{\bar{b}}{\bar{b}}\binom{\bar{a}}{\bar{a}}\binom{c}{\varepsilon}\binom{\bar{c}}{\varepsilon}\binom{\bar{a}}{\varepsilon}.$$

However, only the first alignment e is a tree edit alignment. For the second one, if we choose $i = 2$, we observe that $e'(i) = (a,a)$ and $j = 8$ is the unique position such that $e(j) = (\bar{a},\bar{a})$, but $i \not\sim_v j$.

It is not difficult to see that, given two trees t and t', there is a sequence of tree edit operations turning t into t' and having cost N iff there is a tree edit alignment between the serializations $\hat{t}$ and $\hat{t}'$ having cost $2N$.

Interestingly, the following example shows that the generic notion of alignment between serializations is too powerful to capture the costs of edit operations on trees, even up to multiplicative constants. There exist indeed families of trees on which the costs of alignments are uniformly bounded, while the costs of tree edit operations get arbitrary high.

Example 3. *Consider pairs of trees of the same height and of the following forms (some labels are in bold to mark the differences between the left-hand and right-hand side):*



It is easy to see that, no matter how one chooses to transform the left-hand side tree into the right-hand side one using tree edit operations, the cost grows at least linearly with the height of the trees. On the other hand, there exist alignments between the serializations of these pairs of trees that have uniformly bounded cost, e.g.

$$\binom{r}{r}\binom{a}{a}\binom{\bar{a}}{\bar{a}}\binom{r}{r}\binom{b}{b}\binom{\bar{b}}{\bar{b}}\cdots\binom{r}{r}\binom{a}{a}\binom{\bar{a}}{\bar{a}}\binom{r}{r}\binom{b}{b}\binom{\bar{b}}{\bar{b}}\binom{r}{r}\cdot$$

$$\binom{\bar{r}}{\boldsymbol{\varepsilon}}\binom{\boldsymbol{b}}{\boldsymbol{\varepsilon}}\binom{\bar{\boldsymbol{b}}}{\boldsymbol{\varepsilon}}\binom{\bar{r}}{\bar{r}}\binom{a}{a}\binom{\bar{a}}{\bar{a}}\binom{\bar{r}}{\bar{r}}\cdots\binom{b}{b}\binom{\bar{b}}{\bar{b}}\binom{\bar{r}}{\bar{r}}\binom{a}{a}\binom{\bar{a}}{\bar{a}}\binom{\bar{r}}{\bar{r}}\binom{\boldsymbol{\varepsilon}}{\boldsymbol{b}}\binom{\boldsymbol{\varepsilon}}{\bar{\boldsymbol{b}}}\binom{\boldsymbol{\varepsilon}}{\bar{\boldsymbol{r}}}.$$

It is important to notice that the previous alignment is not a tree edit alignment.

2.4 Transducers

A streaming repair process is a machine that consumes parts of the input and produces edits. We capture this with the notion of sub-sequential *transducer*. This device can be described by a tuple of the form $\mathcal{Z} = (\Sigma, \Delta, Z, \kappa, z_0, \Omega)$, where Σ is a finite alphabet for the input, Δ is a finite alphabet for the output, Z is a (possibly infinite) set of states, κ is a partial transition function from $Z \times (\Sigma \uplus \{\varepsilon\})$ to $\Delta^* \times Z$, $z_0 \in Z$ is an initial state, and Ω is a final output function from Z to Δ^*. A run of $\mathcal{Z}$ consists of a sequence of transitions of the form

$$z_0 \xrightarrow{u_1/v_1} z_1 \xrightarrow{u_2/v_2} \ldots \xrightarrow{u_n/v_n} z_n \xrightarrow{\varepsilon/v_{n+1}}$$

where $u_i \in \Sigma \uplus \{\varepsilon\}$, $v_i \in \Delta^*$, $v_{n+1} = \Omega(z_n)$, and $\delta(z_{i-1}, u_i) = (v_i, z_i)$ for all $1 \le i \le n$. Given the above run, we say that $v = v_1 \cdot v_2 \cdot \ldots \cdot v_n \cdot v_{n+1}$ is the *output* of $\mathcal{Z}$ on *input* $u = u_1 \cdot u_2 \cdot \ldots \cdot u_n$. In order to guarantee that $\mathcal{Z}$ produces at most one output on each input, we forbid the possibility that both $\delta(z,a)$, with $a \in \Sigma$, and $\delta(z,\varepsilon)$ are defined on the same state z.

The above definition of run of a transducer implicitly defines an alignment between the input and the output. Recall that the definition of an alignment refers not only to the input and output words, but to a particular way of synchronizing them with ε. Thus we will first 'disambiguate' the edits induced by the run of the transducer, determining whether a given transition u/v is to be considered as a deletion, an insertion, or a deletion followed by an insertion. Formally, given a run ρ of $\mathcal{Z}$ like the one described above, we define the *canonical alignment of* ρ as

$$\mathsf{align}(\rho) = \mathsf{align}\binom{u_1}{v_1} \cdot \mathsf{align}\binom{u_2}{v_2} \cdot \ldots \cdot \mathsf{align}\binom{u_n}{v_n} \cdot \mathsf{align}\binom{\varepsilon}{v_{n+1}}$$

where

$$\mathsf{align}\binom{u}{v} = \begin{cases} \binom{a}{\varepsilon}\binom{\varepsilon}{b_1}\ldots\binom{\varepsilon}{b_k} & \text{if } u = a,\ v = b_1 \ldots b_k, \\ & \text{and } a \neq b_i \text{ for all } 1 \le i \le k \\ \binom{\varepsilon}{b_1}\ldots\binom{a}{b_i}\ldots\binom{\varepsilon}{b_k} & \text{if } u = a,\ v = b_1 \ldots b_k, \\ & \text{and } i = \min_{j \le k}\{j \ : \ a = b_j\} \\ \binom{\varepsilon}{b_1}\ldots\binom{\varepsilon}{b_k} & \text{if } u = \varepsilon \text{ and } v = b_1 \ldots b_k. \end{cases}$$

We define the *cost* of a run ρ of $\mathcal{Z}$ as the cost of its canonical alignment $\mathsf{align}(\rho)$.

In general, canonical alignments of runs of transducers can be of any form. In the following, however, we restrict ourselves to transducers that only work on serializations of trees and whose canonical alignments are tree edit alignments. We call these transducers *tree edit transducers*.

3. PROBLEM SETTING

This paper focuses on the *streaming bounded repairability problem* for languages of unranked trees. The setting is given by two languages R and T of unranked trees, called *restriction* and *target* languages. Trees in R (resp. T) are labelled over a finite alphabet Σ (resp. Δ), and they are encoded by serializations. The languages R and T are represented by means of DTDs or deterministic top-down tree automata.

The goal is to decide whether it is possible to 'repair' any tree $t \in R$ into a tree $t' \in T$, using a number of edits on serializations that is uniformly bounded by a constant (the constant may depend on the restriction and target languages, but not on the tree t). Here, we are specially interested in repair strategies that are *streaming*, that is, we only consider repairs of serializations of trees that are produced in an online way, by means of tree edit transducers.

Formally, the streaming bounded repairability problem consists of deciding, given two languages R and T (presented as DTDs or top-down tree automata), whether there exists a tree edit transducer $\mathcal{Z}$ such that (i) on any input $\hat{t}$, with $t \in R$, $\mathcal{Z}$ outputs the serialization $\hat{t}'$ of some tree $t' \in T$, and (ii) the cost of the runs of $\mathcal{Z}$ are uniformly bounded by a constant (this implies that the edit-distance between the input $\hat{t}$ and the corresponding output $\hat{t}'$ is also bounded by a constant). Some examples of positive and negative instances of the streaming bounded repairability problem follow.

Example 1 (continued). *Consider again the languages $R = \mathscr{L}(\mathcal{R})$ and $T = \mathscr{L}(\mathcal{T})$ described in our running example. It is possible to transform every tree $t \in R$ (e.g. the left-hand side tree of Figure 1) into a tree $t' \in T$ (e.g. the right-hand side tree of Figure 1) using just 3 edit operations: one first deletes the d-labelled child of the root, then relabels the b-labelled node into a, finally one inserts a new e-labelled node as a first child of the root, adopting the two chains of a-labelled nodes as sub-trees. This strategy can be implemented at the level of serializations by a transducer that first copies the opening tag r from the input, then it produces an opening tag e and copies the portion $a \dots a\,\bar{a} \dots \bar{a}$ of the input; subsequently, it replaces the input string $d\,b$ with a, it copies another portion $a \dots a\,\bar{a} \dots \bar{a}$ of the input, it prepends $\bar{a}\,\bar{e}$ to the next incoming string $c\,\bar{c} \dots c\,\bar{c}$, and finally it erases the closing tag $\bar{d}$ and copies the last symbol $\bar{r}$.*

Example 4. *The following is a variant of an example from [2], which shows the difference between non-streaming edit strategies and streaming ones. Consider the language R of all trees of the form $r(x, c, \dots, c, y, \dots, y)$, with $x, y \in \{a, b\}$, and the language T of all trees of the form $r(x, c, \dots, c, x, \dots, x)$, with $x \in \{a, b\}$. A simple way to edit a tree of R into a tree of T is to replace the label x of the first child with the label y occurring the rightmost sibling. This strategy has uniformly bounded cost, but it cannot be implemented by a tree edit transducer of similar cost. Indeed, every transducer of bounded cost that parses a serialization of a tree of R has to commit to either preserving or modifying the label x of the first child before seeing the right siblings labelled by y. We have that the language R is not streaming repairable into T with uniformly bounded cost.*

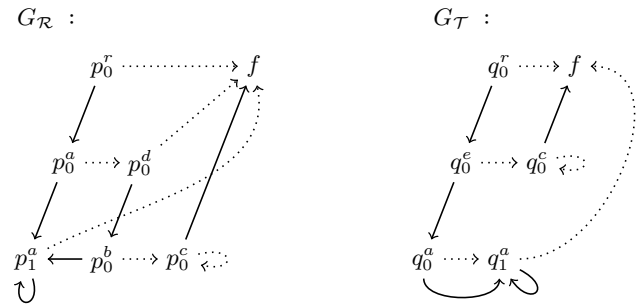


Figure 4: Transitions graphs of automata $\mathcal{R}$ and $\mathcal{T}$.

4. MAIN CHARACTERIZATION

In this section, we present an effective characterization of streaming bounded repairability for the languages recognized by top-down tree automata (thus including those languages definable by DTDs). This characterization combines ideas both from the solution of the streaming repair problem for regular word languages [2] and from the solution of the non-streaming repair problem for regular languages of unranked trees [12]. For instance, in [2] streaming bounded repairability for regular word languages was characterized in terms of a simulation game over the directed acyclic graphs of the strongly connected components of the automata – similar concepts are used also in the present paper. In [12] special conditions related to the behaviour of tree automata along the vertical (i.e. first-child) axis were taken into account – here we do something similar in the presence of 'vertical' contexts. To describe formally our characterization, we need to first introduce some definitions and notations.

4.1 Components of automata

It is easy to see that any top-down tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ can be equivalently represented by its transition graph $G_{\mathcal{A}} = \big(Q, (E_a^1)_{a\in\Sigma}, (E_a^2)_{a\in\Sigma}\big)$, where the nodes are the control states of $\mathcal{A}$ and the edge relations E_a^1 and E_a^2 are defined as follows:

$$(q, q') \in E_a^1 \quad \text{iff} \quad \exists\, q'' \in Q.\ \ \delta(q, a) = (q', q''),$$
$$(q, q') \in E_a^2 \quad \text{iff} \quad \exists\, q'' \in Q.\ \ \delta(q, a) = (q'', q').$$

Intuitively, the edges in E_a^1, called *vertical edges*, represent transitions of $\mathcal{A}$ from a given node of a tree to its first child, while the edges in E_a^2, called *horizontal edges*, represent transitions of $\mathcal{A}$ from a given node to its next sibling.

Figure 4 depicts the transition graphs of the automata $\mathcal{R}$ and $\mathcal{T}$ of Example 1 (dotted arrows represent horizontal edges, solid arrows represent vertical edges).

Using the graph representation of an automaton $\mathcal{A}$, we can derive the notion of *strongly connected component* (or simply a *component*) of $\mathcal{A}$: this is a maximal set X of nodes of $G_{\mathcal{A}}$ such that for all $q, q' \in X$, there is a directed path from q to q' visiting only nodes in X and traversing edges in $\bigcup_{a\in\Sigma} E_a^1 \cup E_a^2$. We observe that for every $q, q' \in X$, there is a context C such that $\hat{\delta}(q, C) = q'$.

We denote by $\mathsf{SCC}(\mathcal{A})$ the set of all components of $\mathcal{A}$ and we distinguish between two types of components $X \in \mathsf{SCC}(\mathcal{A})$:

- X is *horizontal* if all its edges are horizontal, namely, if $(q, q') \notin E_a^1$ for all $q, q' \in X$ and $a \in \Sigma$,
- X is *non-horizontal* if it contains at least one vertical edge, namely, if $(q, q') \in E_a^1$ for some $q, q' \in X$ and $a \in \Sigma$.

The left-hand side graph of Figure 4 contains six horizontal components and only one non-horizontal component (i.e. the one consisting of the single state p_1^a); similarly, the right-hand side graph of Figure 4 contains five horizontal components and one non-horizontal component.

Non-trivial components, namely, components that contain at least one edge, represent 'repeatable behaviours' of the automaton. These components have to be taken into account in the characterization of streaming bounded repairability because they could generate arbitrary large fragments of trees (namely, contexts) that cannot be edited with uniformly bounded cost. To make this statement more precise, we associate with each component X of an automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ the *language of contexts realizable in* X:

$$\mathscr{L}(\mathcal{A} \,|\, X) \;=\; \big\{C \;:\; \exists\, q, q' \in X.\;\; \hat{\delta}(q, C) = q'\big\}.$$

Recall that contexts are trees or forests with a single placeholder ($\bullet$-labelled node) occurring at a leaf with no right sibling. It is easy to see that a component X of $\mathcal{A}$ is horizontal iff the placeholders of all contexts in the language $\mathscr{L}(\mathcal{A} \,|\, X)$ occur at the top level (i.e. as rightmost roots). Such contexts are called *horizontal* and intuitively represent hedges of trees.

As an example, the automaton $\mathcal{R}$ of our running Example 1 contains one non-trivial horizontal component $\{p_0^c\}$, which realizes contexts of the form $c\,c \ldots c\bullet$. The other non-trivial component of $\mathcal{R}$ is $\{p_1^a\}$, which is non-horizontal and realizes contexts of the form $a(a(\ldots a(\bullet)\ldots))$.

4.2 Prefix-rewriting systems

To understand our characterization result, it is useful to think of a deterministic top-down tree automaton as a device that processes serializations of trees in a single-threaded left-to-right fashion, rather than in parallel. This could be formalized in terms of special forms of Visibly Pushdown Automata [8] that run on serializations of trees and simulate exactly the computations of deterministic top-down tree automata. Here, we prefer to avoid such a formalization and only introduce the minimum amount of terminology that is necessary for understanding our results.

Given a deterministic top-down tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, a state q of it, and a prefix u of the serialization of a tree, we say that q is the *current state at the end of* u iff $\hat{\delta}(q_0, C) = q$ and $\hat{C}^{\mathsf{prefix}} = u$ for some context C. We remark that this current state only depends on u, and not the context C; indeed, due to top-down determinism, $\hat{C}_1^{\mathsf{prefix}} = \hat{C}_2^{\mathsf{prefix}}$ implies $\hat{\delta}(q_0, C_1) = \hat{\delta}(q_0, C_2)$, for any two contexts C_1, C_2.

We now turn back to our streaming bounded repairability problem. Informally, being able to perform a bounded repair from a restriction automaton $\mathcal{R}$ to a target automaton $\mathcal{T}$, one needs to respond to prefixes u of serializations of trees in $\mathscr{L}(\mathcal{R})$ by prefixes v of serializations of trees in $\mathscr{L}(\mathcal{T})$ in such a way that, at any point, if we take the component of the current state of $\mathcal{R}$ at the end of u, the language of contexts realized in this component is covered by the language of contexts realized in the component of the current state of $\mathcal{T}$ at the end of v. In this way, if the prefix u is repeatedly extended in a cyclic way – without exiting the component of the current state – the repair processor can respond by just copying the input symbols, incurring no cost. Of course, it is not feasible to look at all possible prefixes u of serializations of trees in $\mathcal{R}$. Thus our characterization of streaming bounded repairability is based on a sort of simulation game in which *abstractions* of runs of $\mathcal{R}$ are produced by one player, and are countered by abstractions of runs of $\mathcal{T}$, produced by the other player.

The abstractions are stacks of components, representing the states at the frontier of the portion of the tree that is represented by the prefix of the serialization. For example, extending a prefix u of a serialization with a new opening tag a induces a transition of $\mathcal{R}$ from the current state p to two states p_1 and p_2 (one associated with the new a-labelled child, the other associated with a forthcoming right sibling). This transition is abstracted at the level of components by a corresponding push-and-swap move that replaces the component of p at the top of the restriction stack with the components of p_1 and p_2.

A key observation is that it is not necessary to mimic all transitions of the restriction automaton, but only those that exit the current component and reach new components with both successor states. This will keep the length of the plays in the simulation game bounded, allowing us to determine the winner effectively.

Formalizing this, we capture the dynamics of stacks of components via prefix-rewriting systems associated with the restriction and target automata $\mathcal{R}$ and $\mathcal{T}$. These systems act on stacks of components of $\mathcal{R}$ and $\mathcal{T}$ and they are naturally obtained from the 'lifting' of the transition rules to the strongly connected components. Stacks of components are presented as strings under the usual convention that the top element of a stack is listed first. Given a stack $\vec{z}$, we denote by $\mathsf{top}(\vec{z})$ its top element and by $\mathsf{tail}(\vec{z})$ the sub-stack below this element. We will use $\vec{x}, \vec{x}', \vec{x}''$ (resp. $\vec{y}, \vec{y}', \vec{y}''$) to denote stacks of components of $\mathcal{R}$ (resp. $\mathcal{T}$).

We start with the definition of the prefix-rewriting system associated with the restriction automaton $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$. This is the relation $\overset{\mathcal{R}}{\mapsto} \;\subseteq\; \mathsf{SCC}(\mathcal{R})^* \times \mathsf{SCC}(\mathcal{R})^*$ between stacks of components of $\mathcal{R}$ defined by:

$$\begin{array}{lll} X \cdot \vec{x} \;\overset{\mathcal{R}}{\mapsto}\; X_1\, X_2 \cdot \vec{x} & \text{iff} & X_1 \neq X \;\wedge\; X_2 \neq X \;\wedge \\ & & \exists\, p \in X, p_1 \in X_1, p_2 \in X_2, a \in \Sigma \\ & & \delta(p, a) = (p_1, p_2) \\ X \cdot \vec{x} \;\overset{\mathcal{R}}{\mapsto}\; \vec{x} & & \text{always} \end{array}$$

where X, X_1, X_2 denote single components of $\mathcal{R}$. Note that $\vec{x} \overset{\mathcal{R}}{\mapsto} \vec{x}'$ implies either $|\vec{x}| = |\vec{x}'| + 1$ or $|\vec{x}| + 1 = |\vec{x}'|$. Moreover, according to the above definition, the component X at the top of the stack cannot be rewritten into a copy of it (this is due to the condition $X_1 \neq X \;\wedge\; X_2 \neq X$).

The prefix-rewriting system associated with the target automaton $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ is defined in a similar way, with only two differences. First, we allow components of $\mathcal{T}$ to be rewritten into themselves (for instance, we allow rules of the form $Y \overset{\mathcal{T}}{\mapsto} Y\,Y$ whenever $\gamma(q,a) = (q_1, q_2)$ for some states $q, q_1, q_2 \in Y$). This difference is required essentially because several components of $\mathcal{R}$ could be covered by the same component of $\mathcal{T}$. Second, we allow rewriting rules that simulate the execution of several transitions of $\mathcal{T}$ at once: this is done by taking the reflexive and transitive closure of a basic rewriting relation $\overset{\mathcal{T}}{\mapsto}$, which is defined just below. This corresponds to the fact that in the target we can make multiple repairs (e.g. insert multiple symbols) in response to a single input symbol of the restriction.

We associate with the target automaton $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ the relation $\overset{\mathcal{T}}{\mapsto} \subseteq \mathsf{SCC}(\mathcal{T})^* \times \mathsf{SCC}(\mathcal{T})^*$ defined by:

$$\begin{array}{lll} Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y_1\,Y_2 \cdot \vec{y} & \text{iff} & \exists\, q \in Y, q_1 \in Y_1, q_2 \in Y_2, a \in \Delta \\ & & \gamma(q,a) = (q_1, q_2) \\ Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} \vec{y} & \text{iff} & \vec{y} \neq \varepsilon. \end{array}$$

We denote by $\overset{\mathcal{T}}{\mapsto}{}^*$ the reflexive and transitive closure of the relation $\overset{\mathcal{T}}{\mapsto}$.

Example 1 (continued). *Consider the automata $\mathcal{R}$ and $\mathcal{T}$ of our running example (see also Figure 4 for a quick reference of the transitions). The following are two valid derivations of the prefix-rewriting systems $\overset{\mathcal{R}}{\mapsto}$ and $\overset{\mathcal{T}}{\mapsto}{}^*$:*

$$\{p_0^r\} \overset{\mathcal{R}}{\mapsto} \{p_0^a\}\{f\} \overset{\mathcal{R}}{\mapsto} \{p_1^a\}\{p_0^d\}\{f\} \overset{\mathcal{R}}{\mapsto} \{p_0^d\}\{f\}$$

$$\{q_0^r\} \overset{\mathcal{T}}{\mapsto}{}^* \{q_0^a\}\{q_0^c\}\{f\} \overset{\mathcal{T}}{\mapsto}{}^* \{q_1^a\}\{q_1^a\}\{q_0^c\}\{f\} \overset{\mathcal{T}}{\mapsto}{}^* \{f\}.$$

4.3 The simulation game

Now, we have all the ingredients to characterize streaming bounded repairability for two languages $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$ in terms of a suitable simulation game between the prefix-rewriting systems $\overset{\mathcal{R}}{\mapsto}$ and $\overset{\mathcal{T}}{\mapsto}{}^*$ associated with $\mathcal{R}$ and $\mathcal{T}$.

To explain the general idea we first consider the simpler case where all components of the restriction automaton $\mathcal{R}$ are horizontal. In this case, the simulation game takes place between two players, called *Generator* and *Repairer*, who control two stacks $\vec{x} \in \mathsf{SCC}(\mathcal{R})^*$ and $\vec{y} \in \mathsf{SCC}(\mathcal{T})^*$ using the prefix-rewriting relations $\overset{\mathcal{R}}{\mapsto}$ and $\overset{\mathcal{T}}{\mapsto}{}^*$, respectively. The game starts with the initial singleton stacks X_0 and Y_0, where X_0 is the component of the initial state of $\mathcal{R}$ and Y_0 is the component of the initial state of $\mathcal{T}$. Repairer moves first by applying to his stack Y_0 a sequence of prefix-rewriting rules satisfying $\overset{\mathcal{T}}{\mapsto}{}^*$ (this corresponds to the fact that the repair processor is allowed to insert some initial prefix of the output, prior to any input being received). Generator responds by applying to his stack X_0 a single prefix-rewriting rule satisfying $\overset{\mathcal{R}}{\mapsto}$. Then the game continues in a similar way from the new pair of stacks. Some invariants have to be enforced. Every time Repairer moves, he has to guarantee that the language $\mathscr{L}(\mathcal{T}|\mathsf{top}(\vec{y}))$ of contexts realizable in the top component of his stack $\vec{y}$ contains the language $\mathscr{L}(\mathcal{R}|\mathsf{top}(\vec{x}))$ of contexts realizable in the top component of the stack $\vec{x}$ of Generator. We will see later in Section 4.4 how this covering property between languages of components eases the repair process. Eventually, one of the two players will not be able to move, in which case the other player wins.

In order to correctly characterize streaming bounded repairability in the presence of non-horizontal components of $\mathcal{R}$, we need to consider a variant of the simulation game where a special separator symbol $\triangleleft$ is prepended to the non-horizontal components of the stacks. For the sake of presentation, it is convenient to describe the variant of the simulation game by introducing a third player, called *Referee*, who handles the occurrences of the separator symbol $\triangleleft$ in the two stacks. The game goes as before by alternating between moves of Repairer and moves of Generator. However, if after a move of Repairer the element at the top of the stack of Generator happens to be a non-horizontal component, then Referee comes into play: he inserts the separator symbol $\triangleleft$ just below the top components of the stacks of Generator and Repairer and he passes the turn to Generator. From there after, neither Generator nor Repairer are allowed to modify the parts of their stacks that are hidden under a separator. If after a move of Generator the top element of his stack becomes $\triangleleft$, then Referee comes again into play: he removes $\triangleleft$ from the top of the stack of Generator, he pops from the stack of Repairer the top-most separator and all elements above it, and he finally passes the turn to Repairer. We remark that in the above formulation of the game, Referee cannot choose his moves, as these are always determined by the current configuration of the game. This makes the game equivalent to a classical turn-based two-player reachability game, whose winner is known to be determined.

A formal definition of the arena of the game follows. For the sake of readability, we use a different notation (i.e. $[\![\vec{x}, \vec{y}]\!]$ and $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle$) for the positions of the arena that belong to Generator and Repairer; for the positions owned by Referee the notation is that of the player who moves next.

Definition 2. *Let $\mathcal{R}$ and $\mathcal{T}$ be two top-down tree automata and let $\vec{x}, \vec{x}', \vec{x}''$ (resp. $\vec{y}, \vec{y}', \vec{y}''$) denote generic sequences over $\mathsf{SCC}(\mathcal{R}) \uplus \{\triangleleft\}$ (resp. $\mathsf{SCC}(\mathcal{T}) \uplus \{\triangleleft\}$). The arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ for the simulation game is defined as follows:*

- *the positions owned by Generator are the pairs $[\![\vec{x}, \vec{y}]\!]$, where $\mathsf{top}(\vec{x})$ and $\mathsf{top}(\vec{y})$ are components such that $\mathscr{L}(\mathcal{R}|\mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T}|\mathsf{top}(\vec{y}))$, and where $\mathsf{top}(\mathsf{tail}(\vec{x})) = \triangleleft$ whenever $\mathsf{top}(\vec{x})$ is non-horizontal;*
- *the positions owned by Repairer are the pairs $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle$, where $\mathsf{top}(\vec{x}) \neq \triangleleft$;*
- *the positions owned by Referee are the pairs $[\![\vec{x}, \vec{y}]\!]$, where $\mathsf{top}(\vec{x})$ and $\mathsf{top}(\vec{y})$ are non-horizontal components, $\mathscr{L}(\mathcal{R}|\mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T}|\mathsf{top}(\vec{y}))$, and $\mathsf{top}(\mathsf{tail}(\vec{x})) \neq \triangleleft$, as well as the pairs $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle$, where $\mathsf{top}(\vec{x}) = \triangleleft$;*
- *the initial position is the pair $\langle\!\langle \vec{x}_0, \vec{y}_0 \rangle\!\rangle$, which is owned by Repairer, where $\vec{x}_0$ (resp. $\vec{y}_0$) is the singleton stack that consists of the component of the initial state of $\mathcal{R}$ (resp. $\mathcal{T}$);*

- *the possible moves for Generator are of the form* $[\![\vec{x}\cdot\vec{x}'', \vec{y}]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle \vec{x}'\cdot\vec{x}'', \vec{y}\rangle\!\rangle$*, where* $\vec{x} \overset{\mathcal{R}}{\mapsto} \vec{x}'$ *is a single prefix-rewriting rule associated with* $\mathcal{R}$ *(in particular,* $\triangleleft$ *occurs neither in* $\vec{x}$ *nor in* $\vec{x}'$*);*
- *the possible moves for Repairer are of the form* $\langle\!\langle \vec{x}, \vec{y}\cdot\vec{y}''\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}, \vec{y}'\cdot\vec{y}'']\!]$*, where* $\vec{y} \overset{\mathcal{T}}{\mapsto}{}^{*} \vec{y}'$ *is a sequence of prefix-rewriting rules associated with* $\mathcal{T}$ *(in particular,* $\triangleleft$ *occurs neither in* $\vec{y}$ *nor in* $\vec{y}'$*);*
- *the possible moves for Referee are of the form* $[\![X\cdot\vec{x}'', Y\cdot\vec{y}'']\!] \overset{\mathsf{Ref}}{\mapsto} [\![X\cdot\triangleleft\cdot\vec{x}'', Y\cdot\triangleleft\cdot\vec{y}'']\!]$*, where* X *is non-horizontal, and those of the form* $\langle\!\langle \triangleleft\cdot\vec{x}, \vec{y}\cdot\triangleleft\cdot\vec{y}''\rangle\!\rangle \overset{\mathsf{Ref}}{\mapsto} \langle\!\langle \vec{x}, \vec{y}''\rangle\!\rangle$*, where* $\triangleleft$ *does not occur in* $\vec{y}$*.*

We observe that all plays that could possibly arise from the simulation game over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ are finite: this is because each position of $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ is visited at most once during a play and the set of all reachable positions is finite, due to the restriction on the moves of Generator. Indeed the stacks that could be derived from the prefix-rewriting system $\overset{\mathcal{R}}{\mapsto}$ have length at most $|\mathsf{SCC}(\mathcal{R})|$. This allows us to define the winner of a play as the last player who moved (this must be either Generator or Repairer).

Example 1 (continued). *We continue our running example by describing a prefix of possible play over the arena* $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ *(to save space and improve readability, we write the pairs for the positions of the arena vertically):*

$$\left\langle\!\!\left\langle \begin{smallmatrix}\{p_0^r\}\\ \{q_0^r\}\end{smallmatrix}\right\rangle\!\!\right\rangle \overset{\mathsf{Rep}}{\mapsto} \left[\!\!\left[\begin{smallmatrix}\{p_0^r\}\\ \{q_0^r\}\end{smallmatrix}\right]\!\!\right] \overset{\mathsf{Gen}}{\mapsto} \left\langle\!\!\left\langle \begin{smallmatrix}\{p_0^a\}\{f\}\\ \{q_0^r\}\end{smallmatrix}\right\rangle\!\!\right\rangle \overset{\mathsf{Rep}}{\mapsto} \left[\!\!\left[\begin{smallmatrix}\{p_0^a\}\{f\}\\ \{q_0^a\}\{q_0^c\}\{f\}\end{smallmatrix}\right]\!\!\right] \overset{\mathsf{Gen}}{\mapsto}$$

$$\left\langle\!\!\left\langle \begin{smallmatrix}\{p_1^a\}\{p_0^d\}\{f\}\\ \{q_0^a\}\{q_0^c\}\{f\}\end{smallmatrix}\right\rangle\!\!\right\rangle \overset{\mathsf{Rep}}{\mapsto} \left[\!\!\left[\begin{smallmatrix}\{p_1^a\}\{p_0^d\}\{f\}\\ \{q_1^a\}\{q_1^a\}\{q_0^c\}\{f\}\end{smallmatrix}\right]\!\!\right] \overset{\mathsf{Ref}}{\mapsto} \left[\!\!\left[\begin{smallmatrix}\{p_1^a\}\triangleleft\{p_0^d\}\{f\}\\ \{q_1^a\}\triangleleft\{q_1^a\}\{q_0^c\}\{f\}\end{smallmatrix}\right]\!\!\right]$$

$$\overset{\mathsf{Gen}}{\mapsto} \left\langle\!\!\left\langle \begin{smallmatrix}\triangleleft\{p_0^d\}\{f\}\\ \{q_1^a\}\triangleleft\{q_1^a\}\{q_0^c\}\{f\}\end{smallmatrix}\right\rangle\!\!\right\rangle \overset{\mathsf{Ref}}{\mapsto} \left\langle\!\!\left\langle \begin{smallmatrix}\{p_0^d\}\{f\}\\ \{q_1^a\}\{q_0^c\}\{f\}\end{smallmatrix}\right\rangle\!\!\right\rangle \quad \ldots$$

It is easy to see that Repairer has a strategy to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$*.*

As we explained earlier, it is more difficult for Repairer to win the simulation game when the stack he controls contains some separator symbols – in this case he cannot apply the prefix-rewriting rules arbitrarily deep into his stack. The purpose of the following example is to demonstrate that, without this limitation, Repairer can win the simulation game even if the restriction language is not streaming bounded repairable into the target language.

Example 5. *Let* $\mathcal{R}'$ *and* $\mathcal{T}'$ *be the deterministic top-down tree automata with the following transitions (*p_0 *and* q_0 *are the initial states, all other states are final):*

$$\begin{array}{llll} \mathcal{R}' : & p_0 \xrightarrow{r} \underline{p_1} & \underline{f} \\ & \underline{p_1} \xrightarrow{a} \underline{p_1} & \underline{f} \\ & \underline{p_1} \xrightarrow{b} \underline{f} & \underline{p_2} \\ & \underline{p_2} \xrightarrow{b} \underline{f} & \underline{p_2} \end{array} \qquad \begin{array}{llll} \mathcal{T}' : & q_0 \xrightarrow{r} \underline{q_1} & \underline{f} \\ & \underline{q_1} \xrightarrow{a} \underline{q_2} & \underline{q_3} \\ & \underline{q_2} \xrightarrow{a} \underline{q_2} & \underline{f} \\ & \underline{q_3} \xrightarrow{b} \underline{f} & \underline{q_3} \end{array}$$

The following are examples of trees in $\mathscr{L}(\mathcal{R}')$ *and in* $\mathscr{L}(\mathcal{T}')$*:*

Clearly, $\mathscr{L}(\mathcal{R}')$ *is not bounded repairable into* $\mathscr{L}(\mathcal{T}')$ *(not even with an offline repair strategy). Accordingly, Repairer loses the simulation game over* $\mathcal{G}_{\mathcal{R}',\mathcal{T}'}$ *in the presence of separator symbols: Generator has a winning strategy that consists of first reaching the restriction stack* $\{p_1\}\triangleleft\{f\}$*, forcing Repairer to respond with a target stack of the form* $\{q_2\}\triangleleft\ldots\{q_3\}\{f\}$*, and later rewriting his stack to* $\{p_2\}\triangleleft\{f\}$*, thus leading to a losing position for Repairer (the component* $\{p_2\}$ *of* $\mathcal{R}'$ *is not covered by any component of* $\mathcal{T}'$ *that is reachable from* $\{q_2\}$*).*

On the other hand, Repairer can easily win the simulation game if the separators are omitted: from any position of the arena of the form $\langle\!\langle \{p_2\}\{f\}, \{q_2\}\ldots\{q_3\}\{f\}\rangle\!\rangle$*, Repairer could simply pop the top component from his stack and cover in this way the top component of the restriction stack.*

We are now ready to state our main characterization result:

Theorem 1. *Given a pair of deterministic top-down tree automata* $\mathcal{R}$ *and* $\mathcal{T}$*, there exists a streaming repair strategy from* $\mathscr{L}(\mathcal{R})$ *to* $\mathscr{L}(\mathcal{T})$ *with uniformly bounded cost iff Repairer has a strategy to win the simulation game over* $\mathcal{G}_{\mathcal{R},\mathcal{T}}$*.*

The effectiveness of the above characterization is discussed in Section 5, together with tight complexity bounds for the streaming bounded repairability problem. In the following we give an intuitive account of the proof of Theorem 1. Finally, it is important to point out that from this proof one can effectively construct a tree edit transducer that repairs $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with uniformly bounded cost whenever Repairer wins the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

4.4 Outline of the proof of the main theorem

We explain first the idea underlying the proof of the only-if-direction of Theorem 1. In this direction, we assume the existence of a tree edit transducer $\mathcal{Z}$ that implements a streaming repair strategy of $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$, with uniformly bounded cost, and we derive from that the existence of a strategy for Repairer to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Once again, it is convenient to think of the restriction and target automata as devices that process serializations of trees. We thus reuse the notion of *current state at the end of a prefix of a serialization* (cf. Section 4.2).

A key ingredient for constructing a winning strategy for Repairer lies in the fact that, without loss of generality, one can assume that the transducer $\mathcal{Z}$ satisfies the following invariant: for every prefix u of an input serialization, if X is the component of the current state of $\mathcal{R}$ at the end of u and Y is the component of the current state of $\mathcal{T}$ at the end of the corresponding output v, then the language of contexts

realized in X is covered by the language of contexts realized in Y, namely,

$$\mathscr{L}(\mathcal{R}\,|\,X) \;\subseteq\; \mathscr{L}(\mathcal{T}\,|\,Y).$$

Indeed, if this were not the case, then the prefix u could be expanded by an iteration of a context that stays within the same component X and, unless the corresponding output induces a change of component in the target automaton, each context would have to be repaired into Y, thus resulting in unbounded repair cost.

Thanks to the above invariant, one can abstract the runs of the transducer $\mathcal{Z}$ into valid plays over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, which turn out to be winning for Repairer (for this it is crucial that the positions $[\![\vec{x}, \vec{y}]\!]$ that are reached after each move of Repairer satisfy the containment $\mathscr{L}(\mathcal{R}\,|\,\mathsf{top}(\vec{x})) \;\subseteq\; \mathscr{L}(\mathcal{T}\,|\,\mathsf{top}(\vec{y})))$.

We outline now the main ideas underlying the proof of the if-direction. Given a strategy for Repairer to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, we have to construct a tree edit transducer $\mathcal{Z}$ that transforms serializations of trees in $R = \mathscr{L}(\mathcal{R})$ into serializations of trees in $T = \mathscr{L}(\mathcal{T})$, using a uniformly bounded number of editing operations. For the sake of simplicity, we will overlook the details related to the presence of non-horizontal components in $\mathcal{R}$ and the role of Referee in the simulation game.

It is convenient to construct the transducer $\mathcal{Z}$ incrementally, that is, as a cascade composition of fairly simple transducers $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$. Intuitively, the first transducer $\mathcal{Z}_1$ decomposes the input tree t into a uniformly bounded number of contexts, each one realizable within a single component of $\mathcal{R}$ (this may require deleting a small number of nodes in t). Furthermore, the output of $\mathcal{Z}_1$ is formed in such a way that one can easily extract a sequence of prefix-rewriting steps of the form $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} X_1\, X_2 \cdot \vec{x}$ or $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} \vec{x}$. The second transducer $\mathcal{Z}_2$ receives the output of $\mathcal{Z}_1$ and computes the responses of Repairer to the moves of Generator induced by the rewriting steps provided by $\mathcal{Z}_1$ (for this purpose, we exploit the existence of a winning strategy for Repairer). Furthermore, $\mathcal{Z}_2$ annotates the contexts of the decomposition of t with partial runs of the target automaton $\mathcal{T}$. Finally, the third transducer $\mathcal{Z}_3$ receives the output of $\mathcal{Z}_2$ and glues the pieces of runs of $\mathcal{T}$ in order to form a complete run on a tree $t' \in \mathscr{L}(\mathcal{T})$ (this requires inserting additional contexts of uniformly bounded size, which can be extracted from the moves of Repairer provided by $\mathcal{Z}_2$). In the following, we describe the two intermediate languages U and V that are implicitly defined by these transducers, and we argue that there exist streaming repair strategies of uniformly bounded cost from R to U, from U to V, and from V to T, which are implemented respectively by the transducers $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$.

To define the first intermediate language U, we need to introduce the concept of $\mathcal{R}$-*decomposition tree*. The idea is to describe a decomposition of a tree $t \in R$ into a uniformly bounded number of contexts, each one realized within a component of $\mathcal{R}$, and, at the same time, to provide a corresponding sequence of prefix-rewriting steps on the stack controlled by Generator in the simulation game. Because contexts realized within components may become large and because

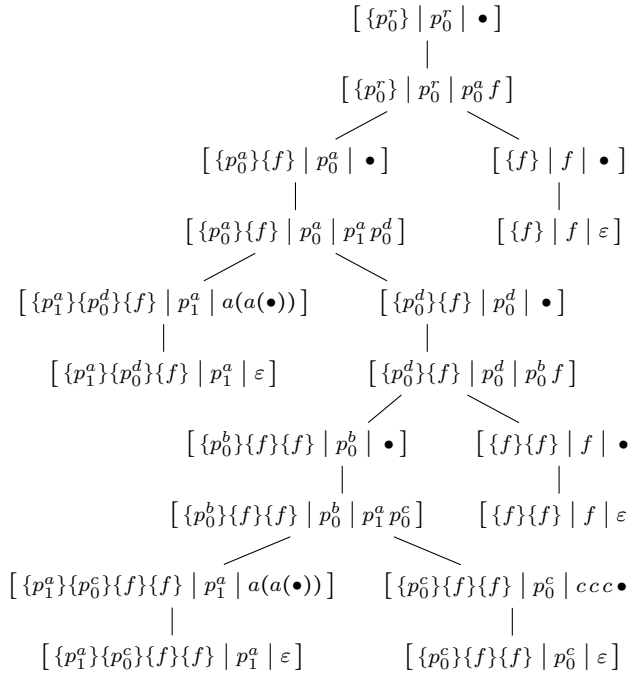


Figure 5: An $\mathcal{R}$-decomposition tree.

we need to treat them as atomic objects, it is convenient to think of decomposition trees as finite trees labelled over an *infinite ranked alphabet*, which we denote by $[\Sigma]$. The elements of this alphabet are *nullary symbols* of the form $[\vec{x} \mid p \mid \varepsilon]$, *unary symbols* of the form $[\vec{x} \mid p \mid C]$, and *binary symbols* of the form $[\vec{x} \mid p \mid p_1\, p_2]$, where $\vec{x}$ denotes a stack of components of $\mathcal{R}$, p, p_1, p_2 denote states of $\mathcal{R}$, and C denotes a context. We enforce the following constraints on any $\mathcal{R}$-decomposition tree:

- the root is labelled with a symbol $[X_0 \mid p_0 \mid C]$, where p_0 is the initial state of $\mathcal{R}$ and X_0 is its component,
- every unary node with label $[\vec{x} \mid p \mid C]$ satisfies $p \in \mathsf{top}(\vec{x})$ and $\hat{\delta}(p, C) \in \mathsf{top}(\vec{x})$ and has for child a leaf or a binary node whose label $[\vec{x}' \mid p' \mid \gamma]$ satisfies $\vec{x}' = \vec{x}$ and $\hat{\delta}(p, C) = p'$,
- every binary node with label $[\vec{x} \mid p \mid p_1\, p_2]$ satisfies $p \in \mathsf{top}(\vec{x})$, $p_1, p_2 \notin \mathsf{top}(\vec{x})$, and $\delta(p, a) = (p_1, p_2)$ for some $a \in \Sigma$, and has for children two unary nodes with labels $[\vec{x}_1 \mid p_1' \mid C_1]$ and $[\vec{x}_2 \mid p_2' \mid C_2]$ satisfying $\vec{x}_1 = X_1 X_2 \cdot \mathsf{tail}(\vec{x})$, $\vec{x}_2 = X_2 \cdot \mathsf{tail}(\vec{x})$, with $p_1 \in X_1$, $p_2 \in X_2$.

Example 1 (continued). *In Figure 5 we show a decomposition tree for the restriction automaton $\mathcal{R}$ of our running example. Intuitively, this decomposition tree can be obtained from tree t that is depicted in Figure 1 by simulating a run of $\mathcal{R}$ on it and by extracting maximal contexts realized within single components of $\mathcal{R}$; the binary nodes of this decomposition tree correspond to the transitions of $\mathcal{R}$ that induce a change of component along both successor states.*

We define the *serialization* $\hat{t}$ of an $\mathcal{R}$-decomposition tree t in the usual way by introducing an opening tag $\langle \vec{x} \mid p \mid \gamma \rangle$ and a closing tag $\langle / \vec{x} \mid p \mid \gamma \rangle$ for each of the infinitely many symbols

$[\vec{x} \mid p \mid \gamma] \in [\Sigma]$. The only detail here is that, for a technical reason that will be clear soon, we need to define the opening and closing tags of the unary symbols $[\vec{x} \mid p \mid C]$ respectively as $\langle \vec{x} \mid p \mid \hat{C}^{\mathsf{prefix}} \rangle$ and $\langle / \vec{x} \mid p \mid \hat{C}^{\mathsf{suffix}} \rangle$ (recall that $\hat{C}^{\mathsf{prefix}}$ is the prefix of the serialization of C ending immediately before $\bullet$, while $\hat{C}^{\mathsf{suffix}}$ is the suffix starting immediately after $\bar{\bullet}$).

We observe that from the serialization $\hat{t}$ of an $\mathcal{R}$-decomposition tree one can derive a sequence of derivation steps that satisfy the prefix-rewriting relation $\overset{\mathcal{R}}{\mapsto}$: for this it is sufficient to replace each occurrence of an opening tag $\langle X \cdot \vec{x} \mid p \mid p_1\, p_2 \rangle$ with the push-and-swap move $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} X_1 X_2 \cdot \vec{x}$, where X, X_1, X_2 are the components of the states p, p_1, p_2, respectively, replace each occurrence of a closing tag $\langle / X \cdot \vec{x} \mid p \mid \hat{C}^{\mathsf{suffix}} \rangle$ with the pop move $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} \vec{x}$, and discard all other tags.

We define the first intermediate language U as the set of all $\mathcal{R}$-decomposition trees. The fact that this language is not, strictly speaking, recognizable by a (finite) deterministic top-down tree automaton is not an issue, since here we are mainly interested in proving that serializations of trees of R can be transformed into serializations of trees of U using special forms of transducers of uniformly bounded cost. We should however explain how transducers can turn sequences of tags over Σ into sequences of tags over $[\Sigma]$ and what is the induced cost. For this we adopt a variant of the notion of tree edit transducer which can consume *in a single transition* a long portion u of the input and provide as output a single tag of the form $\langle \vec{x} \mid p \mid \gamma_u \rangle$ or $\langle / \vec{x} \mid p \mid \gamma_u \rangle$. To enforce functionality of the transducer, it is sufficient to assume that the substrings u that can be consumed by such a transition range over a prefix-code, namely, a language in which no pair of words are one prefix of the other. Moreover, by viewing the content γ_u of the output tag as a string, we can define the cost of such a transition as $1 + \mathsf{dist}(u, \gamma_u)$.

We briefly explain how the restriction language R can be repaired into U with uniformly bounded cost. The idea is to simulate the run of the restriction automaton $\mathcal{R}$ on the disclosed portion of the input tree t and, at the same time, decompose t into a contexts realizable within single components of $\mathcal{R}$. This requires the use of special transitions for replacing large portions of the input of the form $\hat{C}^{\mathsf{prefix}} \cdot a$, with two opening tags of the form $\langle \vec{x} \mid p \mid \hat{C}^{\mathsf{prefix}} \rangle\, \langle \vec{x} \mid p' \mid p_1\, p_2 \rangle$, and, similarly, for replacing portions of the input of the form $\bar{a} \cdot \hat{C}^{\mathsf{suffix}}$ with two closing tags of the form $\langle / \vec{x} \mid p' \mid p_1\, p_2 \rangle\ \langle / \vec{x} \mid p \mid \hat{C}^{\mathsf{prefix}} \rangle)$, where $\hat{\delta}(p, C) = p'$, $p, p' \in \mathsf{top}(\vec{x})$, $\delta(p, a) = (p_1, p_2)$, and $p_1, p_2 \notin \mathsf{top}(\vec{x})$. In this way, the content of the input serialization is reproduced almost unchanged inside the output tags – only few input symbols are deleted, which correspond to the transitions of $\mathcal{R}$ that induce a change of component along both successors. Note that, thanks to top-down determinism, a change of component can be detected as soon as the corresponding open symbol is processed. This explains how R is repaired into U by a streaming transducer $\mathcal{Z}_1$ of uniformly bounded cost.

We turn to the second intermediate language V. Exactly as we did for U, we define V as a set of decomposition trees for the target automaton $\mathcal{T}$. These trees are labelled over the infinite ranked alphabet $[\Delta]$ that contains nullary symbols $[\vec{y} \mid q \mid \varepsilon]$, unary symbols $[\vec{y} \mid q \mid C]$, and binary symbols $[\vec{y} \mid q \mid q_1 q_2]$, with $\vec{y} \in \mathsf{SCC}(\mathcal{T})^+$, $q \in \mathsf{top}(\vec{y})$, C context such that $\hat{\gamma}(q, C) \in \mathsf{top}(\vec{y})$, and $\gamma(q, a) = (q_1, q_2)$ for some $a \in \Delta$. The only interesting difference with respect to the previous definition of decomposition tree is that a node of a $\mathcal{T}$-decomposition tree can be labelled with a binary symbol $[\vec{y} \mid q \mid q_1 q_2]$ even if q_1 or q_2 belong to the same component of q (this reflects the different definitions of the prefix-rewriting systems $\overset{\mathcal{R}}{\mapsto}$ and $\overset{\mathcal{T}}{\mapsto}$, cf. Section 4.2).

In order to transform $\mathcal{R}$-decomposition trees into $\mathcal{T}$-decomposition trees, we allow new editing operations of bounded cost, that is: relabellings and insertions of binary nodes, insertions of unary nodes, and, finally, relabellings of unary nodes from $[\vec{x} \mid p \mid C] \in [\Sigma]$ to $[\vec{y} \mid q \mid C] \in [\Delta]$. We observe that when relabelling a unary node, we can only change the stack of components and the state, but not the context, which must then belong to both languages $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x}))$ and $\mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}))$. Streaming strategies that transform $\mathcal{R}$-decomposition trees into $\mathcal{T}$-decomposition trees are defined in the usual way as transducers working on serializations.

Given a strategy for Repairer to win the game $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, one can construct a transducer $\mathcal{Z}_2$ of bounded cost that transforms the serialization of any tree in U into the serialization of a tree in V. This is achieved by constructing a corresponding play inside $\mathcal{G}_{\mathcal{R},\mathcal{T}}$: the moves of Generator are derived from the series of input tags of the form $\langle \vec{x} \mid p \mid p_1\, p_2 \rangle$ and $\langle / \vec{x} \mid p \mid \hat{C}^{\mathsf{suffix}} \rangle$, while the moves of Repairer are obtained from his winning strategy. For each move $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}, \vec{y}']\!]$ that is generated during this process, a certain number of opening and closing tags will be produced in the output; these tags represent the basic steps of the prefix-rewriting relation $\overset{\mathcal{T}}{\mapsto}{}^*$. Similarly, for each move $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}, \vec{y}']\!]$, the label $[\vec{x} \mid p \mid C]$ of a descendant node may be changed to a label of the form $[\vec{y}' \mid q \mid C]$. We observe that in doing so one needs to guarantee that the states q and $\hat{\gamma}(q, C)$ belong to the same component $\mathsf{top}(\vec{y}')$; this is possible thanks to the fact that the game position $[\![\vec{x}, \vec{y}']\!]$ owned by Generator satisfies the containment $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}'))$ and because the following property holds:

Lemma 1. *If $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$, then for every $p \in X$, there exist $q \in Y$ such that for all contexts C, $\hat{\delta}(p, C) \in X$ implies $\hat{\gamma}(q, C) \in Y$.*

We finally turn to the last stage of the processing line, namely, the transducer that repairs V into T. Here the idea is that every $\mathcal{T}$-decomposition tree can be turned into a concrete tree satisfying the target specification T by glueing together the contexts that appear in the unary nodes. To achieve this it might be necessary to produce additional contexts of small size that connect states from different components. For instance, if $[\vec{y} \mid q \mid q_1\, q_2]$ is the label of a binary node and $[\vec{y}_1 \mid q_1' \mid C_1]$ and $[\vec{y}_2 \mid q_2' \mid C_2]$ are the labels of its children, then suitable contexts C_1' and C_2' will be inserted in such a way that $\hat{\gamma}(q_1, C_1') = q_1'$ and $\hat{\gamma}(q_2, C_2') = q_2'$. As the size and number of these contexts is bounded, we have that V is streaming bounded repairable into T via a suitable transducer $\mathcal{Z}_3$.

By chaining all the transducers together, one obtains a tree edit transducer $\mathcal{Z} = \mathcal{Z}_1 \circ \mathcal{Z}_2 \circ \mathcal{Z}_3$ that repairs R into T with a uniformly bounded number of editing operations.

5. COMPLEXITY RESULTS

In the previous section we gave a game-theoretic characterization of streaming bounded repairability. The effectiveness of such a characterization, and hence the decidability of the streaming bounded repairability problem, follows from the fact that the considered simulation game can be seen as a specific reachability game [7], whose plays are uniformly bounded in length. More precisely, given a restriction $\mathcal{R}$ and a target $\mathcal{T}$, the plays that could possibly arise over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ have length at most exponential in the number of components of $\mathcal{R}$. This gives a straightforward alternating exponential-time procedure that exhaustively searches all plays to determine the winner of the simulation game, and possibly synthesize a winning strategy.

Below, we improve the complexity result that we just derived to a tight EXPTIME bound.

THEOREM 2. *The problem of streaming bounded repairability for languages recognized by top-down tree automata is in EXPTIME.*

The proof of the EXPTIME upper bound is based on constructing a variant of the simulation game that still characterizes streaming bounded repairability, but whose configurations can be succinctly represented in polynomial space. More precisely, given two automata $\mathcal{R}$ and $\mathcal{T}$, we recall that the stacks controlled by Generator in the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ never exceed in length the number of components of $\mathcal{R}$. Unfortunately, an analogous bound to the lengths of the stacks controlled by Repairer does not hold – this is essentially due to the existence of prefix-rewriting rules of the form $Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y_1 Y_2 \cdot \vec{y}$, with $Y_1 = Y$, which can be iterated to produce arbitrarily long stacks. To overcome this problem and be able to perform an exhaustive search on the arena in alternating polynomial space, one considers an equivalent version of the simulation game, which is obtained by introducing a dummy copy $\widetilde{Y}$ of each component Y of $\mathcal{T}$, by replacing every prefix-rewriting rule $Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y Y_2 \cdot \vec{y}$ with the rule $\widetilde{Y} \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y_2 \widetilde{Y} \cdot \vec{y}$, and by replacing every occurrence of Y with $Y \widetilde{Y}$ in the right-hand side of a rule. The modified game is shown to be equivalent to the original game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, but the reachable configurations can now be represented within polynomial size with respect to $\mathcal{R}$ and $\mathcal{T}$. This gives an alternating polynomial-space procedure that determines the winner of the simulation game, thus proving the EXPTIME upper bound for the problem of streaming bounded repairability.

In the next theorem, we show that the problem of streaming bounded repairability for top-down tree automata is EXPTIME-hard. In fact, we show that EXPTIME-hardness holds for languages specified by *deterministic DTDs* (also known as *one-unambiguous DTDs*) [4]. Formally, a DTD is said to be *deterministic* if the regular expression in the right-hand side of every rule can be translated efficiently (in PTIME) into an equivalent deterministic finite state automaton. Given that any deterministic DTD can be efficiently translated into an equivalent deterministic top-down tree automaton [10], the EXPTIME-hardness result can be transferred to languages recognized by deterministic top-down tree automata.

THEOREM 3. *The problem of streaming bounded repairability for languages defined by deterministic DTDs is EXPTIME-hard.*

The proof of the above result is based on a reduction from the problem of deciding the winner of a tiling game over a corridor of polynomial width and exponential height. The tiling game is run by two players, Adam and Eve. At each turn, one of the two players extends the current tiling by inserting a new row on top of the previous one. In doing so, the two players have to satisfy some constraints for the pairs of adjacent tiles. The last player who cannot move loses. We know from [15] that deciding the winner of a tiling game is APSPACE-hard (hence EXPTIME-hard). Reducing this problem to the streaming bounded repairability problem amounts at constructing, in polynomial time, two deterministic DTDs $\mathcal{R}$ and $\mathcal{T}$ such that the language defined by $\mathcal{R}$ is streaming bounded repairable into the language defined by $\mathcal{T}$ iff Eve wins the tiling game. The main idea of the reduction is that the restriction DTD $\mathcal{R}$ will generate encodings of rows of tiles representing the possible moves of Adam, while the target DTD $\mathcal{T}$ will require interleaving these encodings by other ones, which represent Eve responses to Adam. The first technical ingredient lies in the encodings of the rows produced by Adam: we need to allow some redundancy, that is, repeat each tile in a row several times, in order to forbid any repair processor from modifying the rows with boundedly many edits. Another difficulty lies in enforcing the tiling constraints: since we cannot guarantee that the rows generated in the restriction satisfy the vertical constraints, we allow Adam to 'cheat' by producing rows that do not match with the previous ones. This freedom is countered by the possibility of Eve of producing an ad-hoc repair that exposes a violation of the constraints, making it checkable by a DTD of small size.

We now exhibit a sub-class of restriction automata on which the streaming bounded repairability problem becomes easier to solve, namely, PSPACE-complete. This sub-class is obtained by restricting the accessibility graph of the components of $\mathcal{R}$ to have the shape of a tree. More precisely, given two components X and X' of $\mathcal{R}$, we write $X \longrightarrow^*_{\mathcal{R}} X'$ whenever there exist some states $q \in X$ and $q' \in X'$ that are connected in the transition graph $\mathcal{G}_{\mathcal{R}}$ by a directed path of (horizontal or vertical) edges. The graph that consists of the components of $\mathcal{R}$ and the edges $X \longrightarrow^*_{\mathcal{R}} X'$ is a directed acyclic graph, and it is denoted by $\mathsf{DAG}(\mathcal{R})$. We say that $\mathcal{R}$ is *tree-shaped* if $\mathsf{DAG}(\mathcal{R})$ is diamond-free, namely, if $X_1 \longrightarrow^*_{\mathcal{R}} X'$ and $X_2 \longrightarrow^*_{\mathcal{R}} X'$ imply either $X_1 \longrightarrow^*_{\mathcal{R}} X_2$ or $X_2 \longrightarrow^*_{\mathcal{R}} X_1$. Similarly, we say that a restriction DTD is tree-shaped if its language is recognized by a tree-shaped top-down tree automaton.

Below, we show that the problem of streaming bounded repairability is PSPACE-complete for restriction languages

recognized by tree-shaped automata, and it is hard already for languages specified by tree-shaped deterministic DTDs.

THEOREM 4. *The problem of streaming bounded repairability for restriction languages recognized by tree-shaped top-down tree automata is in PSPACE.*

A sketch of a proof of the PSPACE upper bound is as follows. From the fact that the restriction automaton is tree-shaped, one derives a polynomial bound on the length of the possible plays over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. To compute the winner of the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ we run an alternating polynomial-time procedure that exhaustively searches all plays.

The PSPACE-hardness result below follows from ideas similar to the proof of Theorem 3, that is, by reducing the satisfiability problem for quantified boolean formulas to the problem of streaming bounded repairability of a tree-shaped restriction DTD into a target DTD.

THEOREM 5. *The problem of streaming bounded repairability for restriction languages defined by tree-shaped deterministic DTDs is PSPACE-hard.*

We conclude the section by pointing out a result from [12] that concerns a specific case of the streaming bounded repairability problem. From Propositions 6 and 7 of [12] it follows that the complexity of the streaming bounded repairability problem drops to PTIME when the restriction language contains all trees over a given alphabet Σ.

6. DISCUSSION

We gave a characterization of which DTDs and XML schemas are streaming bounded repairable, and analysed the complexity of the resulting decision problem. Our techniques do depend heavily on the top-down determinism of the schemas – for the case of schemas given by arbitrary tree automata, decidability is still open. We also do not know the exact complexity of determining the optimal repair transducer, where optimality is expressed in terms of maximal number of repairs.

Our work highlights the issue of the proper notion of edit processor for trees that have a canonical serialization as a string, as is the case with XML. Example 3 shows that the ability to edit tree serializations is more powerful than emitting tree edits. The example can be used to show that there are XML schemas that can be repaired in streaming fashion with a bounded number of edits on the serialization, but where there is no bounded repair processor of any sort (even non-streaming) that repairs using only tree edits. We do not know if this last phenomena can occur for more limited schemas, such as DTDs.

Acknowledgments. We would like to thank Michael Benedikt for the many helpful remarks on the paper. The first author was supported by the EPRCS project "Query-Driven Data Acquisition from Web-based Datasources" (EPSRC EP/H017690/1). The last two authors were supported by the EPSRC project "Enforcement of Constraints on XML streams" (EPSRC EP/G004021/1).

7. REFERENCES

[1] M. Benedikt, G. Puppis, and C. Riveros. The cost of traveling between languages. In *Proc. of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 6756 of *LNCS*, pages 234–245. Springer, 2011.

[2] M. Benedikt, G. Puppis, and C. Riveros. Regular repair of specifications. In *Proc. of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 335–344. IEEE Computer Society, 2011.

[3] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005.

[4] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 140(2):229–253, 1998.

[5] J. Cristau, C. Löding, and W. Thomas. Deterministic automata on unranked trees. In *Foundations of Semistructured Data*, volume 05061 of *Dagstuhl Seminar Proc.*, pages 68–79, 2005.

[6] O. Gauwin, J. Niehren, and Y. Roos. Streaming tree automata. *Information Processing Letters*, 109(1):13–17, 2008.

[7] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: a guide to current research*. Springer, 2002.

[8] V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly pushdown automata for streaming XML. In *Proc. of the 16th International Conference on World Wide Web (WWW)*, pages 1053–1062. ACM, 2007.

[9] W. Martens, F. Neven, and T. Schwentick. Deterministic top-down tree automata: past, present, and future. In *Logic and Automata: history and perspectives*, volume 2 of *Texts in Logic and Games*, pages 505–530. Amsterdam University Press, 2008.

[10] W. Martens, F. Neven, T. Schwentick, and G. Jan Bex. Expressiveness and complexity of XML schema. *ACM Transactions on Database Systems*, 31(3):770–813, 2006.

[11] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology*, 5(4):660–704, 2005.

[12] G. Puppis, C. Riveros, and S. Staworko. Bounded reparability for regular tree languages. In *Proc. of the 15th International Conference on Database Theory (ICDT)*, pages 155–168. ACM, 2012.

[13] L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proc. of the 21st ACM SIGMOD symposium on Principles of Database Systems (PODS)*, pages 53–64. ACM, 2002.

[14] K.C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.

[15] P. Van Emde Boas. The convenience of tilings. In *Complexity, Logic, and Recursion Theory*, pages 331–363. Marcel Dekker, Inc., 1997.

[16] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.