Noname manuscript No.
(will be inserted by the editor)

Which XML schemas are streaming bounded repairable?

Pierre Bourhis · Gabriele Puppis · Cristian Riveros

Received: date / Accepted: date

Abstract In this paper we consider the problem of repairing, that is, restoring validity of, documents with respect to XML schemas. We formalize this as the problem of determining, given an XML schema, whether or not a streaming procedure exists that transforms an input document so as to satisfy the XML schema, 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, which allows the repair procedure to work on documents that are already known to satisfy another XML schema. The decision procedure relies on the analysis of the structure of an automaton model specifying the restriction and target XML schemas and reduces te problem to a novel notion of game played on pushdown systems associated with the schemas.

Keywords bounded repair · streaming repair · DTD · XML Schema · top-down deterministic tree automaton

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 [16]. DTDs and XML schemas are

Pierre Bourhis
CNRS, LIFL Université Lille 1 and INRIA Lille Nord Europe
E-mail: pierre.bourhis@univ-lille1.fr

Gabriele Puppis
CNRS, LaBRI Université Bordeaux
E-mail: gabriele.puppis@labri.fr

Cristian Riveros
Pontificia Universidad Católica de Chile
E-mail: criveros@ing.puc.cl

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* [12,14,19].

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* [18], 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 [4]) we feel this is a natural starting point for the exploration of streaming repair.

We follow our previous work in the non-streaming setting [18] 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 [7]. Our prior work [18] 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 'top-down deterministic 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 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 [18]: we associate a graph with each schema, and then look at the corresponding notion of connected component; such a component represents a 'repeatable behaviour', namely, a family of trees that can be generated 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).

In summary our contributions are:

1. We formalize the property of bounded repairability for languages of unranked trees in the streaming setting and we introduce a suitable notion of transducer that captures streaming edit strategies on serialized trees.
2. We give a game theoretic characterization of streaming bounded repairability for languages of unranked trees defined by 'top-down deterministic schemas' (e.g. XSDs).
3. Using the characterization above, we derive decidability and tight complexity results for the streaming bounded repairability problem and a number of variants of it.

Organization. Section 2 gives basic definitions, including the notions of schema and repair considered in the paper. Section 3 formalizes the streaming bounded repairability problem and states the main characterization theorem. Section 4 gives a detailed proof of the "if direction" of the characterization theorem, which derives a steaming repair processor form a given winning strategy of Repairer. Section 5 gives a detailed proof of the converse direction of the characterization theorem, namely, it shows that from a streaming repair processor of uniformly bounded cost one can extract a winning strategy for Repairer. Section 6 considers the consequences of the characterization theorem for complexity, while Section 7 gives conclusions and discusses future work.

2 Preliminaries

In this paper we work with *finite unranked ordered trees and forests* (hereafter simply called trees and forests) whose nodes are labeled over a fixed finite alphabet. Formally, a *forest* is a function t mapping non-empty sequences of positive natural numbers to symbols from a finite alphabet, e.g. Σ. The domain of this function t is denoted $\mathsf{nodes}(t)$ and satisfies the following closure under lexicographic order: for all $\vec{\imath} \in \mathbb{N}^*$ and all $j, k \in \mathbb{N}$, with $k \leq j$, if $\vec{\imath} \cdot j \in \mathsf{nodes}(t)$, then $\vec{\imath} \cdot k \in \mathsf{nodes}(t)$ and either $\vec{\imath} = \varepsilon$ or $\vec{\imath} \in \mathsf{nodes}(t)$. The roots of the forest t are represented by singleton sequences; in particular, the empty sequence ε does not belong to the domain of a forest. *Trees* are forests with a single root. Given a tree or forest t and a node

$\vec{\imath} \in \mathsf{nodes}(t)$, we say that $t(\vec{\imath})$ is the label of node $\vec{\imath}$ in t. We often describe trees and forests by means of pictures or unranked terms such as $a(b, b, b) \cdot c$.

2.1 Languages of trees and forests

We will consider a sub-class of regular tree languages that contains the languages defined by the structural components of XML Document Type Definitions (DTDs) and XSD schemas [17,16]. This family of languages will be formally defined using a suitable model of tree automaton. However, because many example languages can be conveniently described in terms of DTDs, we begin with a short overview of DTDs.

An *XML Document Type Definition* (*DTD*) is a tuple $\mathcal{D} = (\Sigma, S, L)$, where Σ is a finite alphabet, $S \subseteq \Sigma$ is the set of initial symbols, and L is a function that maps symbols in Σ to regular expressions over Σ [10]. A tree t satisfies the DTD $\mathcal{D}$ if the root of t is labeled with an initial symbol of $\mathcal{D}$, i.e. $t(1) \in S$, and every node $\vec{\imath} \in \mathsf{nodes}(t)$ satisfies $t(\vec{\imath} \cdot 1) \dots t(\vec{\imath} \cdot n) \in L(t(\vec{\imath}))$, where n is the number of children of $\vec{\imath}$. We denote by $\mathscr{L}(\mathcal{D})$ the language of trees satisfying the DTD $\mathcal{D}$. In accordance with the XML standards, DTDs are often assumed to be *deterministic*, that is, to use only rules with one-unambiguous regular expressions [9] and a singleton set of initial symbols. We will often omit the alphabet and the set of initial symbols from the definition of a DTD, since these can be easily understood from the context (e.g. the initial symbol is usually the first to be listed in the expansion rules).

Example 1 Consider the DTD $\mathcal{D}$ defined by the following rules:

$$\mathcal{D}: \quad \begin{array}{lcl} r & \to & a\,d \\ a & \to & a + \varepsilon \\ d & \to & b\,c^* \\ b & \to & a \\ c & \to & \varepsilon \end{array} \qquad t: \quad \begin{array}{ccccccccc} & & & & r & & & & \\ & & a & & & & d & & \\ & & | & & & & & & \\ & & a & & b & & c & c & c \\ & & | & & | & & & & \\ & & a & & a & & & & \\ & & & & | & & & & \\ & & & & a & & & & \end{array}$$

The right-hand side describes an example tree t that satisfies the DTD $\mathcal{D}$.

In Section 6 we will establish some lower bounds to the complexity of the streaming bounded repair problem in the presence of languages defined by DTDs. In doing so we will consider some special forms of DTDs that have been extensively studied in the literature [19,16]. For example, we will consider DTDs that are *non-recursive*, namely, DTDs $\mathcal{D} = (\Sigma, S, L)$ whose dependency graph – i.e. the graph that connects a letter a to a letter b whenever b occurs in the language $L(a)$ – is acyclic.

We now describe an automaton model that generalizes DTDs [15,11]. This model can be seen as a typing system 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), or, equivalently, as a top-down deterministic binary tree automaton that runs on the standard first-child-next-sibling encoding of the input tree. Formally, a *top-down deterministic tree automaton* is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, where:

– Σ is a finite alphabet,

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

To avoid switching every time between unranked trees and their first-child-next-sibling encodings, we define the runs of top-down deterministic automata directly on unranked trees and unranked forests. Given a tree or forest t, we denote by $\mathsf{nodes}^+(t)$ the *extended domain* of t, which contains all nodes of t and all sequences $\vec{\imath} \cdot j \cdot 1$ and $\vec{\imath} \cdot (j+1) \in \mathsf{nodes}^+(t)$, with $\vec{\imath} \cdot j \in \mathsf{nodes}(t)$. 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. A *run* of $\mathcal{A}$ on t is a function $\rho : \mathsf{nodes}^+(t) \to Q$ such that, for all $\vec{\imath} \cdot j \in \mathsf{nodes}(t)$,

$$\delta\big(\rho(\vec{\imath} \cdot j), t(\vec{\imath} \cdot j)\big) \;=\; \big(\rho(\vec{\imath} \cdot j \cdot 1), \rho(\vec{\imath} \cdot (j+1))\big).$$

A run ρ is *accepting* if $\rho(1) = q_0$ and $\rho(\vec{\imath}) \in F$ for all nodes $\vec{\imath} \in \mathsf{nodes}^+(t) \setminus \mathsf{nodes}(t)$. The language recognized by $\mathcal{A}$ is the set $\mathscr{L}(\mathcal{A})$ of all trees/forests $t \in \mathcal{T}_\Sigma$ that induce an accepting run of $\mathcal{A}$.

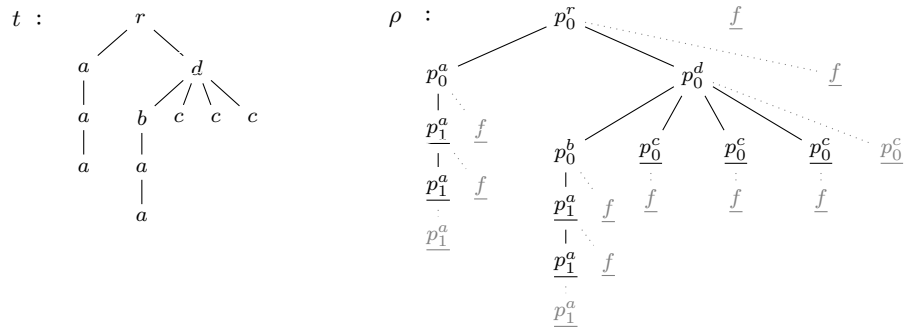
Example 2 We introduce here our running example. Consider the two DTDs below:

$$\begin{array}{lll} \mathcal{D}: & r & \to\ a\,d \\ & a & \to\ a + \varepsilon \\ & d & \to\ b\,c^* \\ & b & \to\ a + \varepsilon \\ & c & \to\ \varepsilon \end{array} \qquad\qquad \begin{array}{lll} \mathcal{D}': & r & \to\ e\,c^* \\ & e & \to\ a + a\,a \\ & a & \to\ a + \varepsilon \\ & c & \to\ \varepsilon \end{array}$$

We can equivalently describe the languages defined by DTDs using top-down deterministic tree automata. In the specific case that we consider, the automata equivalent to the above DTDs have a particular form that allow at most one letter to be parsed from each control state (we remark that this is not the case in general). For instance, the following are the transitions rules of two top-down deterministic tree automata $\mathcal{R}$ and $\mathcal{T}$ that recognize the languages $\mathscr{L}(\mathcal{R}) = \mathscr{L}(\mathcal{D})$ and $\mathscr{L}(\mathcal{T}) = \mathscr{L}(\mathcal{D}')$, respectively (for convenience, we annotate each state with the unique letter that can be parsed from it, we underline the final states, and we tacitly assume that the initial states are p_0^r and q_0^r):

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

Below are examples of a tree t satisfying the DTD $\mathcal{D}$ and the induced accepting run ρ of $\mathcal{R}$ (states of the run are colored in black or gray depending on whether they label nodes of t or nodes in $\mathsf{nodes}^+(t) \setminus \mathsf{nodes}(t)$):



In the sequel, we will tacitly assume that automata are *trimmed*, namely, that all states of automata appear in some accepting run. Because useless states of automata can be detected and removed in linear time [10], this assumption will have no impact on our complexity results.

It is known that top-down deterministic tree automata define a proper subset of the class of regular tree languages [16,15]. We consider this sub-class of regular tree languages because our characterization of streaming bounded repairability is correct only for tree languages specified by means of top-down deterministic tree automata (we refer the reader to Section 3 for further explanations and examples). Despite of the lack of expressiveness, top-down deterministic tree automata are at least as expressive as DTDs and XML Schemas [16], which are commonly used in practice.

2.2 Contexts

As we mentioned in the introduction, part of our solution to the bounded repairability problem in the streaming setting is adapted from a previous work [18]. In particular, we will make extensive use of the notion of context [10], i.e. a tree or a forest with a distinguished hole that acts as a placeholder for substitution. The context hole will be identified by means of a special symbol $\bullet$, not in the underlying alphabet Σ. The general idea is to use contexts to represent pieces of an input tree that can be easily composed, in analogy with concatenation of words.

Formally, a *context* over Σ is a tree or a forest labeled over $\Sigma \uplus \{\bullet\}$, with the symbol $\bullet$ occurring exactly once *in a leaf that has no right sibling*. 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. 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 is defined similarly and results again in a context.

The rationale behind allowing the hole symbol $\bullet$ to occur only at leaves with no right sibling lies in the possibility of extending the transition function of an automaton from single letters to contexts. More precisely, we derive from the transition function δ of a top-down deterministic tree automaton $\mathcal{A}$ a partial transition function δ° on contexts, as follows. Given a state q and a context C, we let $\delta^\circ(q, C) = q'$ if and only if 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. Note that this definition makes only sense when the hole of C occurs at a leaf with no right sibling.

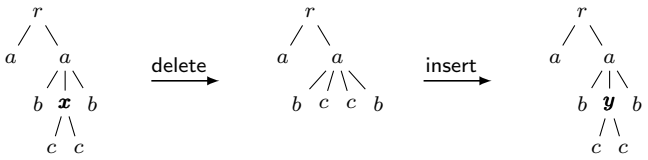


Fig. 1 Edit operations on unranked trees.

2.3 Serializations

It is known that trees, and more generally forests, can be represented by their serializations (XML Documents). Formally, for a given finite alphabet Σ, we introduce a disjoint copy $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$. 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 inductively by letting $\hat{t} = a\bar{a}$ if t is the singleton tree a, and $\hat{t} = a \cdot \hat{t}_1 \cdot \ldots \cdot \hat{t}_n \cdot \bar{a}$ 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 or forest produces a *well-matched* string over $\Sigma \uplus \bar{\Sigma}$ and vice-versa. For example, the serialization of the tree t of Example 2 is:

$$\hat{t} \;=\; r\, a\, a\, a\, \bar{a}\, \bar{a}\, \bar{a}\, d\, b\, a\, a\, \bar{a}\, \bar{a}\, \bar{b}\, c\, \bar{c}\, c\, \bar{c}\, c\, \bar{c}\, \bar{d}\, \bar{r}.$$

According to the above definition, the serialization $\hat{C}$ of a context C is a word that contains a single occurrence of the substring $\bullet\,\bar{\bullet}$. We 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 $\bar{\bullet}$). Observe also that the composition of contexts with trees/contexts has analogous operations on serializations, that is,

$$\widehat{C \circ t} \;=\; \hat{C}^{\mathsf{prefix}} \cdot \hat{t} \cdot \hat{C}^{\mathsf{suffix}} \qquad \text{and} \qquad \widehat{C \circ C'} \;=\; \hat{C}^{\mathsf{prefix}} \cdot \hat{C}' \cdot \hat{C}^{\mathsf{suffix}}.$$

2.4 Edit operations on trees and serializations

The central notion of this paper is that of tree repair, that is a sequence of edit operations on unranked trees. We first recall the definition of the edit operations on unranked trees that are used to derive the standard notion of edit-distance [20, 7, 18]. Next, we discuss what should be the analogous operations on tree serializations, which will be used to implement our repair strategies.

The first edit operation is that of *deletion*, which consists of removing a distinguished (non-root) node $\vec{\imath}$ from an unranked tree t and promoting its subtrees as children of its parent. The second operation is that of *insertion*, which consists of adding a new element below a node $\vec{\imath}$ of an unranked tree t, with a possible adoption of a list of subsequent children of $\vec{\imath}$. Figure 1 gives examples of deletion and insertion operations (to ease readability, we highlighted in bold the deleted and inserted nodes). Sometimes a third edit operation is used, which consists of modifying the label of a distinguished node in a tree; this operation is subsumed by insertion and deletion of nodes and for this reason it will be not considered here as basic edit operation.

In order to characterize bounded repairability of tree languages in the streaming setting, we need to reason on analogous editing operations on serializations. A natural idea is to consider the notion of alignment. 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 $\|e\|$, is the number of occurrences in e that are not of the form (a, a) with $a \in \Sigma$, nor of the form $(\varepsilon, \varepsilon)$. It is known that the minimum cost of all possible alignments between pairs of strings captures the notion of string edit distance [21]. As an example, the string $\binom{a}{a}\binom{b}{\varepsilon}\binom{c}{c}\binom{d}{d}\binom{\varepsilon}{a}$ is an alignment between $u = abcd$ and $v = acda$ that achieves optimal cost 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 having similar costs. For this we need to enforce suitable restriction to the alignments of the tree serializations. This is captured by the notion of *tree edit alignment*. Let us consider two serializations $u \in (\Sigma \uplus \bar{\Sigma})^*$ and $v \in (\Delta \uplus \bar{\Delta})^*$ and an alignment e between them. First, in a way similar to the framework of nested words described in [2], we capture the nesting structure of the words u, v underlying the alignment e by means of two relations $\sim_u$ and $\sim_v$ defined over the positions of e. Formally, given two positions i,, with $1 \leq i < j \leq |e|$, we write $i \sim_u j$ (resp. $i \sim_v j$) if and only if the infix $e[i, j]$ projected onto the first (resp. second) components is a valid serialization of some tree (i.e. a well-matched word). We then say that e is a *tree edit alignment* if the following implications hold:

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

Example 3 Let $t = a(a(b), c)$ and $t' = a(a(c), b)$ and let $\hat{t} = aab\bar{b}\bar{a}c\bar{c}\bar{a}$ and $\hat{t}' = aac\bar{c}\bar{a}b\bar{b}\bar{a}$ be the corresponding serializations. The following are two possible alignments between $\hat{t}$ and $\hat{t}'$:

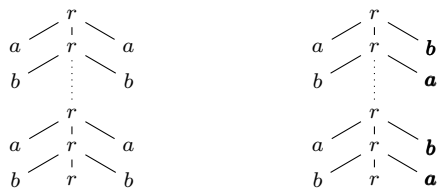
$$e = \binom{a}{a}\binom{a}{a}\binom{b}{c}\binom{\bar{b}}{\bar{c}}\binom{\bar{a}}{\bar{a}}\binom{c}{b}\binom{\bar{c}}{\bar{b}}\binom{\bar{a}}{\bar{a}}$$

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

However, only the first alignment e is a tree edit alignment, which corresponds precisely to the editing strategy that swaps the labels b and c of a tree. The second alignment is not a tree edit alignment, since when we consider the position $i = 2$, 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 of length N that turns t into t' if and only if there is a tree edit alignment of cost $2N$ between the serializations $\hat{t}$ and $\hat{t}'$. Interestingly, the following example inspired from [1] shows that the notion of alignment between serializations needs to be restricted in order to correctly capture the costs of edit operations on trees, even up to multiplicative constants. Indeed, there exist families of trees on which the costs of alignments are uniformly bounded, while the costs of tree edit operations get arbitrary high.

Example 4 Consider pairs of trees of the same height and with the following shapes (bold labels highlight the differences between left-hand and right-hand sides):



No matter how one chooses to transform the left-hand side tree into the right-hand side one using tree edit operations, the repair cost grows at least linearly with the height of the trees. Indeed, it is easy to see that every tree edit operation preserves the order of (non-deleted) nodes induced by the pre-order visit of the tree. In particular, this means that at least one node from each triple of siblings (a, r, a), (b, r, b), ... in the left-hand side tree must be deleted in order to obtain the right-hand side tree. On the other hand, there exist alignments between the serializations of trees of the above forms having uniformly bounded cost, for example:

$$e \;=\; \tbinom{r}{r}\tbinom{a}{a}\tbinom{\bar{a}}{\bar{a}}\tbinom{r}{r}\tbinom{b}{b}\tbinom{\bar{b}}{\bar{b}}\cdots\tbinom{r}{r}\tbinom{a}{a}\tbinom{\bar{a}}{\bar{a}}\tbinom{r}{r}\tbinom{b}{b}\tbinom{\bar{b}}{\bar{b}}\tbinom{r}{r}$$
$$\tbinom{\bar{\boldsymbol{r}}}{\boldsymbol{\varepsilon}}\tbinom{\boldsymbol{b}}{\boldsymbol{\varepsilon}}\tbinom{\bar{\boldsymbol{b}}}{\boldsymbol{\varepsilon}}\tbinom{\bar{r}}{\bar{r}}\tbinom{a}{a}\tbinom{\bar{a}}{\bar{a}}\tbinom{\bar{r}}{\bar{r}}\cdots\tbinom{b}{b}\tbinom{\bar{b}}{\bar{b}}\tbinom{\bar{r}}{\bar{r}}\tbinom{a}{a}\tbinom{\bar{a}}{\bar{a}}\tbinom{\bar{r}}{\bar{r}}\tbinom{\boldsymbol{\varepsilon}}{\boldsymbol{b}}\tbinom{\boldsymbol{\varepsilon}}{\bar{\boldsymbol{b}}}\tbinom{\boldsymbol{\varepsilon}}{\bar{\boldsymbol{r}}}.$$

It is important to note that the above alignment e is not a tree edit alignment. Indeed, if we consider the position $i = 1$ in e, where the symbol $\binom{r}{r}$ occurs, then for every position j of an occurrence of $\binom{\bar{r}}{\bar{r}}$, we have that $i \nsim_v j$, where v is the projection of e onto the second component.

2.5 Transducers

A streaming repair process is a machine that consumes strings from a restriction language and produces strings in a target language. We formalize this by means of the notion of transducer. A (*sequential*) *transducer* is a tuple $\mathcal{Z} = (\Sigma, \Delta, Z, \kappa, z_0, \Omega)$, where

- Σ is a finite alphabet for the input strings,
- Δ is a finite alphabet for the repaired strings,
- 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,
- 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 $\kappa(z_{i-1}, u_i) = (v_i, z_i)$ 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 $\delta(z, \varepsilon)$ and $\delta(z, a)$, for

some $a \in \Sigma$, are defined on the same state z. Observe that the above definition allows for transducers with an infinite number of states: this is needed because our transducers will be used to implement streaming repair strategies between serializations of XML schemas, which, like validation, requires unbounded memory.

The above definition of run of a transducer implicitly defines an alignment between the input and the output, that is, a particular way of synchronizing the characters in the input and output strings. More precisely, we first show how to 'disambiguate' the edits induced by the run of a 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, the *induced alignment* of a run ρ of $\mathcal{Z}$ such as the one described above is the sequence

$$\mathsf{align}(\rho) \ =^{\text{def}} \ \mathsf{align}\left(\begin{smallmatrix} u_1 \\ v_1 \end{smallmatrix}\right) \cdot \mathsf{align}\left(\begin{smallmatrix} u_2 \\ v_2 \end{smallmatrix}\right) \cdot \ldots \cdot \mathsf{align}\left(\begin{smallmatrix} u_n \\ v_n \end{smallmatrix}\right) \cdot \mathsf{align}\left(\begin{smallmatrix} \varepsilon \\ v_{n+1} \end{smallmatrix}\right)$$

where

$$\mathsf{align}\left(\begin{smallmatrix} u \\ v \end{smallmatrix}\right) = \begin{cases} \left(\begin{smallmatrix} a \\ \varepsilon \end{smallmatrix}\right)\left(\begin{smallmatrix} \varepsilon \\ b_1 \end{smallmatrix}\right) \ldots \left(\begin{smallmatrix} \varepsilon \\ b_k \end{smallmatrix}\right) & \text{if } u = a,\ v = b_1 \ldots b_k, \text{ and } a \neq b_i \text{ for all } 1 \leq i \leq k \\ \left(\begin{smallmatrix} \varepsilon \\ b_1 \end{smallmatrix}\right) \ldots \left(\begin{smallmatrix} a \\ b_i \end{smallmatrix}\right) \ldots \left(\begin{smallmatrix} \varepsilon \\ b_k \end{smallmatrix}\right) & \text{if } u = a,\ v = b_1 \ldots b_k, \text{ and } i = \min_{j \leq k}\{j \mid a = b_j\} \\ \left(\begin{smallmatrix} \varepsilon \\ b_1 \end{smallmatrix}\right) \ldots \left(\begin{smallmatrix} \varepsilon \\ b_k \end{smallmatrix}\right) & \text{if } u = \varepsilon \text{ and } v = b_1 \ldots b_k. \end{cases}$$

We define the *aggregate cost* of a transducer $\mathcal{Z}$ on input u, denoted $\mathsf{cost}(u, \mathcal{Z})$, as the cost of the induced alignment of its run on u.

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

3 Bounded repairability in the streaming setting

We describe here the *streaming bounded repair 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 labeled over a finite alphabet Σ (resp. Δ), and they are encoded by their serializations. We assume that the languages R and T are presented by means of top-down deterministic 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 that is uniformly bounded by a constant (i.e. independent of t). We are particularly interested in repair strategies that are *streaming*, that is, that can be applied to serializations of trees and that can be produced in an online way by means of a tree edit transducer.

Formally, let $\mathsf{stream}(R, T)$ be the class of all tree edit transducers $\mathcal{Z}$ such that $\mathcal{Z}(\hat{t}) \in T$ for all $t \in R$. For a tree edit transducer $\mathcal{Z} \in \mathsf{stream}(R, T)$, we define the *worst-case aggregate cost* of $\mathcal{Z}$ as

$$\mathsf{cost}_{\mathcal{Z}}(R, T) \ =^{\text{def}} \ \sup_{t \in R} \mathsf{cost}(\hat{t}, \mathcal{Z}).$$

Finally, we define the *streaming worst-case aggregate cost* for two languages R, T as the minimum of $\mathsf{cost}_{\mathcal{Z}}(R, T)$ taken over all tree edit transducers $\mathcal{Z} \in \mathsf{stream}(R, T)$:

$$\mathsf{cost}_{\mathsf{stream}}(R, T) \ =^{\text{def}} \ \min_{\mathcal{Z} \in \mathsf{stream}(R,T)} \mathsf{cost}_{\mathcal{Z}}(R, T).$$

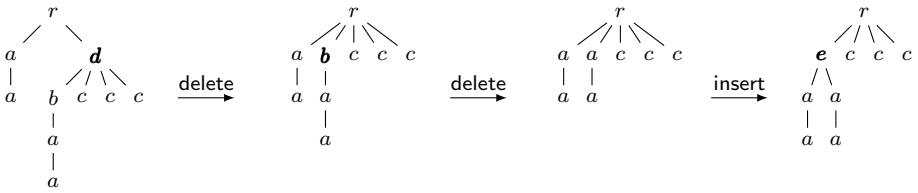


Fig. 2 Example of repair of a tree satisfying DTD $\mathcal{D}$ into a tree satisfying DTD $\mathcal{D}'$.

The *streaming bounded repair problem* consists of deciding, given two regular tree languages R, T whether $\mathsf{cost}_{\mathsf{stream}}(R, T) < \infty$.

The following examples show that it is not at all obvious whether a given schema is bounded repairable into another. The last example shows the difference between non-streaming edit strategies and streaming ones.

Example 2 (continued) Consider the languages R and T defined by the DTDs $\mathcal{D}$ and $\mathcal{D}'$ of our running example. It is possible to transform every tree $t \in R$ into a tree $t' \in T$ using just 3 edit operations: one first deletes the d-labeled child of the root and the b-labeled child of it, and then inserts a new e-labeled node as the first child of the root, adopting the two chains of a-labeled nodes as sub-trees (see Figure 2). This repair strategy can be easily implemented at the level of serializations by a transducer that first copies the opening tag r from the input, then produces an opening tag e and copies the portion $a \dots a\ \bar{a} \dots \bar{a}$ of the input; subsequently, it erases the incoming string $d\,b$, copies the next portion $a \dots a\ \bar{a} \dots \bar{a}$ of the input, replaces the next incoming symbol $\bar{b}$ with $\bar{e}$, copies the string $c\,\bar{c} \dots c\,\bar{c}$, and finally erases the closing tag $\bar{d}$ and copies the last symbol $\bar{r}$ of the stream. Accordingly, we say that R is *streaming bounded repairable* into T.

Example 5 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 repair any tree of R into a tree of T is to replace the label x of the first child with the label y occurring at the rightmost sibling. This strategy has uniformly bounded cost, but 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 labeled with y. Thus, the language R is *not* streaming bounded repairable into T.

The rest of this section is devoted to present an effective characterization of bounded repairability in the streaming setting for pairs of languages recognized by top-down deterministic tree automata (this includes languages definable by DTDs). The characterization combines ideas from previous results related to streaming repairability of regular word languages [5,6] and to non-streaming repairability of regular tree languages [18]. For instance, in [6] streaming bounded repairability was characterized in terms of a simulation game over the directed acyclic graphs of strongly connected components – similar concepts are used in our main characterization. On the other hand, in [18] special conditions related to the behavior of tree automata along the vertical (i.e. first-child) axis were taken into account – here we do something similar in the presence of contexts with a

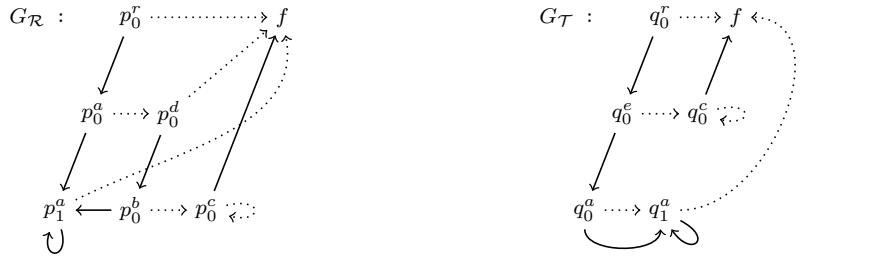


Fig. 3 Transition graphs of two deterministic top-down tree automata, where horizontal and vertical edges are represented by dotted and solid arrows, respectively.

vertical axis. To describe formally our characterization we need to first introduce some definitions and notations.

3.1 Components of automata

The transition structure of a top-down deterministic tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ can be conveniently represented in terms of the graph $G_{\mathcal{A}} = (Q, E_h \uplus E_v)$, where the nodes are the control states of $\mathcal{A}$ and the edge relations E_v, E_h are defined by

$$E_v \ =^{\text{def}} \ \big\{(q, q_1) \in Q \times Q \ \big|\ \exists q_2 \in Q.\ \exists a \in \Sigma.\ \delta(q, a) = (q_1, q_2)\big\}$$

$$E_h \ =^{\text{def}} \ \big\{(q, q_2) \in Q \times Q \ \big|\ \exists q_1 \in Q.\ \exists a \in \Sigma.\ \delta(q, a) = (q_1, q_2)\big\}.$$

Intuitively, the edges in E_v, called *vertical edges*, represent the transitions of $\mathcal{A}$ from a given node of a tree to its first child. The edges in E_h, called *horizontal edges*, represent the transitions of $\mathcal{A}$ from a given node to its next sibling. Figure 3 depicts the transition graphs for the automata $\mathcal{R}$ and $\mathcal{T}$ of Example 2 (dotted arrows represent horizontal edges, solid arrows represent vertical edges).

From the graph representation of an automaton $\mathcal{A}$ we derive the notion of *strongly connected 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 $E_v \cup E_h$. Observe that for all states $q, q' \in X$, there exists a context C such that $\delta^\circ(q, C) = q'$. In fact, we can associate with each component X of the automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ the *language of contexts realizable in* X:

$$\mathscr{L}(\mathcal{A} \mid X) \ =^{\text{def}} \ \big\{C \in \mathcal{C}_\Sigma \ \big|\ \exists\, q, q' \in X.\ \ \delta^\circ(q, C) = q'\big\}.$$

Intuitively, the contexts in the languages of the form $\mathscr{L}(\mathcal{A} \mid X)$ represent 'pieces' of arbitrary large size that form the trees that can be possibly given in input to a streaming repair processor. Because these pieces can be repeated many times, it is better not to repair any of them in order to get into the target language, as doing otherwise would easily incur an unbounded repair cost.

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

– *horizontal* components X, where all edges are horizontal, namely, where $(q, q') \notin E_v$ for all $q, q' \in X$,

– *non-horizontal* components X, which contain at least one vertical edge, namely, where $(q, q') \in E_v$ for some $q, q' \in X$.

It is easy to see that a component X of $\mathcal{A}$ is horizontal if and only if the holes of all contexts of the language $\mathscr{L}(\mathcal{A} \mid X)$ occur at the top level, precisely at the rightmost roots. Such contexts are called *horizontal* and intuitively represent forests, that is, sequences of sub-trees. Symmetrically, contexts with holes occurring at non-root nodes are called *vertical*.

As an example, the left-hand side graph of Figure 3 contains only two components with non-trivial languages of contexts, that is, one horizontal component at state p_0^c realizing contexts of the form $c{\cdot}\ldots{\cdot}c{\cdot}\bullet$, and one non-horizontal component at state p_1^a realizing contexts of the form $a(a(\ldots a(\bullet)\ldots))$. The components of the right-hand side graph are similar, but arranged in a different layout.

3.2 Prefix-rewriting systems

To understand our characterization result, it is useful to think of a top-down deterministic tree automaton as a device that processes serializations of trees in a single-threaded left-to-right fashion, rather than a device that processes the branches of a tree in parallel. For this, we need to reason on states that are reached after parsing prefixes of tree serializations.

Formally, given a top-down deterministic tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ and a prefix u of the serialization of a tree, we define the *state reached by* $\mathcal{A}$ *on* u as the unique state of the form $\delta^\circ(q_0, C)$, for some context C such that $\hat{C}^{\mathsf{prefix}} = u$ (note that this is well-defined because $\hat{C}_1^{\mathsf{prefix}} = \hat{C}_2^{\mathsf{prefix}}$ implies $\delta^\circ(q_0, C_1) = \delta^\circ(q_0, C_2)$ for any two contexts C_1, C_2).

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 state reached by $\mathcal{R}$ on u, the language of contexts realized in this component is *contained* in the language of contexts realized in the component of the state reached by $\mathcal{T}$ on v. In this way, if the prefix u is repeatedly extended in a cyclic way – without leaving the component of the state reached on u – 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 $\mathcal{R}$. For this reason, 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 reached state p to two new states p_1 and p_2 (one associated with the new a-labeled 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.

Formally, 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

$$X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} X_1\, X_2 \cdot \vec{x} \quad \text{if and only if} \quad X_1 \neq X \ \wedge\ X_2 \neq X \ \wedge\ \exists\, p \in X,\ p_1 \in X_1,\ p_2 \in X_2,\ a \in \Sigma.\ \delta(p, a) = (p_1, p_2)$$

$$X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} \vec{x} \quad \text{always}$$

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, due to the condition $X_1 \neq X \wedge X_2 \neq X$ in the above definition, the component X at the top of the stack cannot be rewritten into a copy of it. Together with the fact that the accessibility graph of the components of $\mathcal{R}$ is acyclic, this implies that all sequences of rewriting steps according to $\overset{\mathcal{R}}{\mapsto}$ have finite bounded length.

The prefix-rewriting system for 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 $\mathcal{T}$ (see Section 3.3). 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

$$Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y_1\, Y_2 \cdot \vec{y} \quad \text{if and only if} \quad \exists\, q \in Y,\ q_1 \in Y_1,\ q_2 \in Y_2,\ a \in \Delta.\ \gamma(q, a) = (q_1, q_2)$$

$$Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} \vec{y} \quad \text{if and only if} \quad \vec{y} \neq \varepsilon.$$

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

Example 2 (continued) Consider the automata $\mathcal{R}$ and $\mathcal{T}$ of our running example (see also Figure 3 for a quick reference of their transitions). The following are two valid derivations of the prefix-rewriting systems $\overset{\mathcal{R}}{\mapsto}$ and $\overset{\mathcal{T}}{\mapsto}{}^*$:

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

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

3.3 The simulation game

We have now 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 however. Every time Repairer moves, he has to apply appropriate prefix-rewriting rules to reach a stack $\vec{y}$ such that the language $\mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}))$ of contexts realizable in the top component of $\vec{y}$ contains the language $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x}))$ of contexts realizable in the top component of the stack $\vec{x}$ controlled by Generator, that is,

$$\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y})).$$

We will see later, in Section 4 and 5, how this containment property between languages of contexts 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 an extension 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 extension 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 is 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. $[\![\vec{x}, \vec{y}]\!]$, $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle$, and $(\!|\vec{x}, \vec{y}|\!)$ for the positions of the arena that belong to Generator, Repairer, and Referee (respectively).

Definition 1 Let $\mathcal{R}$ and $\mathcal{T}$ be two top-down deterministic 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 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:

- $[\![\vec{x} \cdot \vec{x}'', \vec{y}]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle \vec{x}' \cdot \vec{x}'', \vec{y} \rangle\!\rangle$ if $\mathsf{top}(\vec{x}' \cdot \vec{x}'') \neq \triangleleft$, and
- $[\![\vec{x} \cdot \vec{x}'', \vec{y}]\!] \overset{\mathsf{Gen}}{\mapsto} (\!|\vec{x}' \cdot \vec{x}'', \vec{y}|\!)$ otherwise,

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:

- $\langle\!\langle \vec{x}, \vec{y} \cdot \vec{y}'' \rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}, \vec{y}' \cdot \vec{y}'']\!]$ if $\mathsf{top}(\vec{x})$ is horizontal, and
- $\langle\!\langle \vec{x}, \vec{y} \cdot \vec{y}'' \rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} (\!|\vec{x}, \vec{y}' \cdot \vec{y}''|\!)$ otherwise,

where $\vec{y} \overset{\mathcal{T}}{\mapsto}{}^{*} \vec{y}'$ is a sequence of prefix-rewriting rules associated with $\mathcal{T}$ and $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}' \cdot \vec{y}''))$; finally the moves for Referee are of the form:

- $(\!|\triangleleft \cdot \vec{x}, \vec{y} \cdot \triangleleft \cdot \vec{y}''|\!) \overset{\mathsf{Ref}}{\mapsto} \langle\!\langle \vec{x}, \vec{y}'' \rangle\!\rangle$ if $\vec{y}$ does not contain any occurrence of $\triangleleft$,
- $(\!|X \cdot \vec{x}'', Y \cdot \vec{y}''|\!) \overset{\mathsf{Ref}}{\mapsto} [\![X \cdot \triangleleft \cdot \vec{x}'', Y \cdot \triangleleft \cdot \vec{y}'']\!]$ if X is non-horizontal and $\mathsf{top}(\vec{x}'') \neq \triangleleft$,
- $(\!|X \cdot \vec{x}'', Y \cdot \vec{y}''|\!) \overset{\mathsf{Ref}}{\mapsto} [\![X \cdot \vec{x}'', Y \cdot \vec{y}'']\!]$ otherwise.

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 (which must be either Generator or Repairer).

Example 2 (continued) We continue our running example by describing a beginning of a possible play over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ (to save space and improve readability, we write vertically the pairs of stacks encoding the positions of the arena):

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

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

The first move is due to Repairer, who leaves the stack unchanged. Then Generator moves by replacing the top component $\{p_0^r\}$ with the pair of components $\{p_0^a\}\ \{f\}$.

Fig. 4 Examples of trees where player Referee is important.

Repairer responds with a sequence of two operations: a push-and-swap operation that replaces $\{q_0^r\}$ with $\{q_0^e\}\ \{f\}$, followed by a push-and-swap operation that replaces $\{q_0^e\}\ \{f\}$ with $\{q_0^a\}\ \{q_0^c\}\ \{f\}$. The play continues in a similar way as a sequence of moves taken alternatively by Repairer and Generator, and possibly interleaved by moves of Referee. It is not difficult to see that Repairer has a strategy to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

As we mentioned 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 6 Let R' and T' be the restriction and target languages that contain the trees depicted in Figure 4. These languages are recognized by two top-down deterministic tree automata $\mathcal{R}'$ and $\mathcal{T}'$, using the following transitions (the states p_0 and q_0 are initial, all other states are final):

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

Clearly, R' is not bounded repairable into T'. 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. Indeed, 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 some components at the top of his stack and cover with $\{q_3\}$ the component $\{p_2\}$ at the top of the restriction stack.

We are now ready to state our main characterization result:

Theorem 1 *Given two top-down deterministic 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 worst-case aggregate cost if and only if 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 6, together with tight complexity bounds for the streaming bounded repairability problem. In Section 4 and 5 we give the detailed proofs for the two directions of Theorem 1. More precisely, in Section 4 we show that if Repairer has a strategy to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, then we can derive from this strategy a tree edit transducer that repairs with bounded cost all serializations of trees from $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$. In Section 5 we prove the converse direction, namely, that if there exists a streaming repair strategy from $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ which incurs a bounded cost then Repairer has a strategy to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

4 From simulation games to repairs

This section is devoted to the proof of the if direction of Theorem 1. We fix two top-down deterministic tree automata $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$ and $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ that recognize the restriction and target languages, respectively, and we assume that Repairer wins the game over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. We will then prove that $\mathscr{L}(\mathcal{R})$ is streaming bounded repairable into $\mathscr{L}(\mathcal{T})$.

Recall that the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ is equivalent to a two-player reachability game: Referee cannot choose his moves, he cannot win nor lose, so one of the other players must win by reaching a position in the arena where the opponent cannot move. We also know that reachability games are positionally determined [13]: this means that Repairer has a winning strategy that is defined only on the basis of the positions of the arena. We describe the positional winning strategy of Repairer by means of a function W that maps any position $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle$ in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ that is owned by Repairer to a move of the form $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}', \vec{y}']\!]$. The function W will be used to derive a transducer $\mathcal{Z}$ that implements a repair processor from $\mathscr{L}(\mathcal{R})$ to $\mathscr{L}(\mathcal{T})$ having uniformly bounded aggregate cost. In fact, it is convenient to construct $\mathcal{Z}$ incrementally, namely, as a cascade composition of fairly simple transducers $\mathcal{Z}_1$, $\mathcal{Z}_2$, $\mathcal{Z}_3$, and $\mathcal{Z}_4$. Below, we give a brief overview of each transducer.

Intuitively, the first transducer $\mathcal{Z}_1$ decomposes the input tree $t \in \mathscr{L}(\mathcal{R})$ into a small (uniformly bounded) number of factors. Some factors will have arbitrary large size: these will not be edited by the repair process and will be represented by contexts that are entirely realizable within single components of the restriction automaton $\mathcal{R}$. The other factors will consist instead of single letters that induce transitions entering different components of $\mathcal{R}$: these factors will be edited (deleted) by the global repair process. The decomposition of t will also induce a corresponding factorization of the unique run of the restriction automaton on t. All factors of t will be accordingly annotated with the initial states of the emerging partial runs. The resulting decomposition annotated with states will be output by the transducer $\mathcal{Z}_1$ in the form of a stream, namely, as a serialized XML tree.

The second transducer $\mathcal{Z}_2$ will use the serialized decomposition produced by $\mathcal{Z}_1$ to construct a corresponding play for the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. The moves of Generator will be derived from the changes of components in $\mathcal{R}$ that are

induced by the single-letter factors of the decomposition (at the same time, the rewriting process erases these single-letter factors). While constructing the moves of Generator, we will guarantee the invariant that the state that annotates each factor in the serialized decomposition belongs to the top component reached by the most recent move of Generator. On the other hand, the moves of Repairer will be obtained from his winning strategy W, as responses to Generator moves. The output of the transducer $\mathcal{Z}_2$ will thus consist of the serialized decomposition, devoid of the single-letter factors, and annotated with moves describing a valid play inside the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

The goal of the third transducer $\mathcal{Z}_3$ is to choose corresponding partial runs for the target automaton $\mathcal{T}$ on the remaining factors of the decomposition. We will see how the existence of these partial runs follows from a technical lemma and from the containment relations $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}))$ that are enforced to all positions $[\![\vec{x}, \vec{y}]\!]$ of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Only the initial states of the chosen partial runs will be annotated inside the input factors. In addition, the transducer $\mathcal{Z}_3$ will replace moves of Repairer with corresponding sequences of transitions in the target automaton. Overall, the output of $\mathcal{Z}_3$ will be the serialization of a partial decomposition tree for $\mathcal{T}$.

Finally, a fourth transduction will insert additional factors of uniformly bounded size to connect the various states in the decomposition produced by the transducer $\mathcal{Z}_3$. The result can be seen as a complete decomposition of a tree t' that is accepted by the target automaton $\mathcal{T}$. The output of the fourth transducer $\mathcal{Z}_4$ will be the projection onto the target alphabet of this complete decomposition, namely, the serialization $\hat{t'}$ of a tree $t' \in \mathscr{L}(\mathcal{T})$.

We will describe each sub-transducer in a different subsection; in the last subsection we will show how they all work together.

4.1 Transducer $\mathcal{Z}_1$: decomposing the tree

As we explained in the overview of the section, the main goal of transducer $\mathcal{Z}_1$ is to receive the serialization $\hat{t}$ of a tree $t \in \mathscr{L}(\mathcal{R})$ and, on the basis of the reached states on each prefix of $\hat{t}$, derive a decomposition of t into a bounded number of contexts, each one realizable within a component of $\mathcal{R}$. Recall that a context is realizable within a component X of $\mathcal{R}$ if it induces a run of $\mathcal{R}$ where every transition has at least one successor state in the same component X. As $\mathcal{Z}_1$ needs to generate a decomposition with a bounded number of contexts, it should never decompose a context unless it is really necessary, that is, unless both successor states fall outside the current component. Moreover, recall that the decomposition of t needs to be constructed in a streaming way while reading the input serialization $\hat{t}$ – we will see later how the construction of such a decomposition relies on the assumption that the restriction automaton $\mathcal{R}$ is top-down deterministic.

We begin by formalizing the concept of decomposition tree, which, as a matter of fact, has some similarities with the notion of synopsis tree introduced in [18]. Here a decomposition is essentially a binary tree whose nodes are labeled over an infinite *ranked* alphabet $[\Sigma]$. Labels in $[\Sigma]$ are of three different types:

- unary labels $[p : C]$, where p is a state of $\mathcal{R}$ and C is a context over Σ,
- binary labels $[p : a]$, where p is a state of $\mathcal{R}$ and a is a letter in Σ,

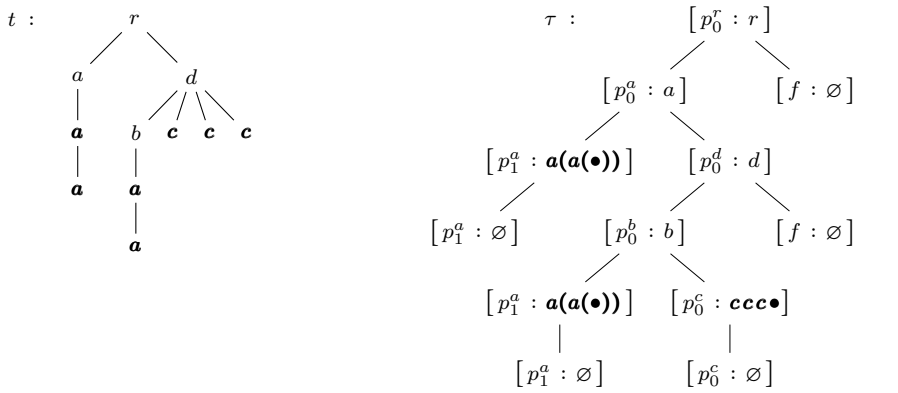


Fig. 5 A tree and an $\mathcal{R}$-decomposition of it.

– nullary labels $[p : \varnothing]$, where p is a state of $\mathcal{R}$.

The idea is that the contexts and the letters over Σ that appear in a decomposition can be arranged together in order to reconstruct the original tree t. The states that appear at each node of a decomposition are used to keep track of the initial states of the partial computations of $\mathcal{R}$ on the factors of t. A formal definition of decomposition follows.

Definition 2 Let $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$ be a restriction automaton. An $\mathcal{R}$-*decomposition* is a ranked tree σ labeled over $[\Sigma]$ such that:

– the root is labeled with a unary, binary, or nullary symbol of the form $[p_0 : \alpha]$, where p_0 is the initial state of $\mathcal{R}$ and α is a context, a single letter in Σ, or $\varnothing$;
– for every node labeled with a unary symbol $[p : C]$, the state $p_1 = \delta^\circ(p, C)$ is well defined and belongs to the same component as p; furthermore, the node has a unique child which is labeled with a unary, binary, or nullary symbol of the form $[p_1 : \alpha]$;
– for every node labeled with a binary symbol $[p : a]$, the pair of states $(p_1, p_2) = \delta(p, a)$ is well defined and neither p_1 nor p_2 belong to the same component as p; furthermore, the node has left and right children labeled respectively with symbols of the form $[p_1 : \alpha_1]$ and $[p_2 : \alpha_2]$;
– every node labeled with a nullary symbol $[p : \varnothing]$ is a leaf and p is a final state.

We say that σ is an $\mathcal{R}$-decomposition *of a tree or forest* t if and only if $[\![\sigma]\!] = t$, where $[\![\sigma]\!]$ is defined inductively as follows:

$$[\![\sigma]\!] \quad =^{\text{def}} \quad \begin{cases} \varnothing & \text{if } \sigma = [p : \varnothing], \\ C \circ [\![\sigma_1]\!] & \text{if } \sigma = [p : C]\,(\sigma_1), \\ a([\![\sigma_1]\!]) \cdot [\![\sigma_2]\!] & \text{if } \sigma = [p : a]\,(\sigma_1, \sigma_2). \end{cases}$$

Example 2 (continued) In Figure 5 we depict a tree t that is accepted by the restriction automaton $\mathcal{R}$ of our running example, together with a possible $\mathcal{R}$-decomposition σ of it. Intuitively, the decomposition σ is obtained from t by looking at the accepting run of $\mathcal{R}$ on t and by extracting maximal contexts realizable

within single components of $\mathcal{R}$ (these contexts are represented by bold terms in the figure). Binary nodes are added to the decomposition when the transitions of the run induce a change of component along both successor states.

The following lemma shows that every tree accepted by the restriction automaton has a decomposition consisting of a small (bounded) number of nodes.

Lemma 1 *For every tree $t \in \mathscr{L}(\mathcal{R})$, there exists an $\mathcal{R}$-decomposition of t with at most $2^{2\cdot|\mathsf{SCC}(\mathcal{R})|}$ nodes.*

Proof. In order to enable a reasoning by structural induction, we need to treat trees and forests in the same manner. For this, we generalize the notion of acceptance by a top-down deterministic tree automaton in the natural way. We will allow a change of the initial state of $\mathcal{R}$ during our induction. More precisely, we will prove that for every tree or forest t accepted by $\mathcal{R}$ from some given initial state, there exists an $\mathcal{R}$-decomposition of t with at most $2^{2\cdot|\mathsf{SCC}(\mathcal{R})|}$ nodes.

The base case of the inductive construction considers the empty forest t, which is clearly accepted by $\mathcal{R}$. In this case, the decomposition is straightforward, that is, $[p_0 : \varnothing]$, where p_0 is the initial state of $\mathcal{R}$.

For the inductive step, consider a non-empty tree or forest t that is accepted by $\mathcal{R}$ and denote by ρ the corresponding accepting run. Note that the state at the first root of ρ is the initial state p_0 of $\mathcal{R}$; we denote by X_0 its component. Consider any maximal path π from the leftmost root of ρ that visits only states in the component X_0 (different decompositions may result from different choices of π, but all of them will satisfy the statement of the lemma). Let $\vec{\imath}\cdot j$ be the last node along this path π. By construction, two cases are possible:

1. either $\vec{\imath}\cdot j \in \mathsf{nodes}^+(t) \setminus \mathsf{nodes}(t)$, namely, the path π reaches a leaf of ρ,
2. or $\vec{\imath}\cdot j \in \mathsf{nodes}(t)$ and neither state $\rho(\vec{\imath}\cdot j\cdot 1)$ nor state $\rho(\vec{\imath}\cdot(j+1))$ belong to X_0.

In the first case, we obtain a context C by extending the domain of t with node $\vec{\imath}\cdot j$ and by marking it with the hole symbol $\bullet$. In particular, the defined context C satisfies $C \circ \varnothing = t$, where $\varnothing$ denotes the empty forest, and $p_1 = \delta^\circ(p_0, C) \in X_0$. Accordingly, we define an $\mathcal{R}$-decomposition of t as follows:

$$\sigma \;=^{\text{def}}\; [p_0 : C]\,\big(\,[p_1 : \varnothing]\,\big).$$

In the second case, we let a be the label of the node $\vec{\imath}\cdot j$ in t and we derive from t a context C by relabeling the node $\vec{\imath}\cdot j$ with $\bullet$ and by removing all proper descendants of $\vec{\imath}\cdot j$ and all subtrees issued from the right siblings of $\vec{\imath}\cdot j$. We also define a forest t_1 by restricting t to the proper descendants of node $\vec{\imath}\cdot j$; similarly, we define a forest t_2 by restricting t to the subtrees issued from the right siblings of $\vec{\imath}\cdot j$. Clearly, we have $C\circ\big(a(t_1)\cdot t_2\big) = t$ and $\delta^\circ(p_0, C) = p_1 \in X_0$, where $p_1 = \rho(\vec{\imath}\cdot j)$. Since both t_1 and t_2 are accepted by $\mathcal{R}$ starting from suitable initial states, i.e., $\rho(\vec{\imath}\cdot j\cdot 1)$ and $\rho(\vec{\imath}\cdot(j+1))$, we know from the inductive hypothesis that there exist some $\mathcal{R}$-decompositions σ_1 and σ_2 for t_1 and t_2, respectively. We can thus obtain an $\mathcal{R}$-decomposition of t by letting

$$\sigma \;=^{\text{def}}\; [p_0 : C]\,\big(\,[p_1 : a]\,(\sigma_1, \sigma_2)\,\big).$$

It remains to prove the upper bound to the number of nodes in σ. We first observe that the construction that we just described gives an $\mathcal{R}$-decomposition in

which unary and binary nodes strictly alternate along any path (with the only exception of the leaves which are labeled by nullary symbols). Moreover, the state associated with any binary node and the state associated with its successor along any path in σ belong to distinct components of $\mathcal{R}$. This implies that the length of any path in σ is at most twice the number of components in $\mathcal{R}$, and hence the total number of nodes in the decomposition σ is at most $2^{2\cdot|\mathsf{SCC}(\mathcal{R})|}$. □

We now turn to explaining how decompositions of trees can be constructed in a streaming way, namely, by means of a transducer. We begin by defining the *serialization* $\hat{\sigma}$ of an $\mathcal{R}$-decomposition σ. In the same spirit of XML-encoding style, we introduce opening and closing *macro-tags* for the symbols $[p : \alpha]$ in $[\Sigma]$. The important detail here is that we embed different strings inside the opening and closing macro-tags, depending on the type of encoded label – as we shall see, this corresponds to handling different pieces of information that become available at different moments during a repair process. Formally:

- for each unary label $[p : C]$, with C context, we introduce the opening macro-tag $\langle p : \hat{C}^{\mathsf{prefix}} \rangle$ and the closing macro-tag $\langle / \hat{C}^{\mathsf{suffix}} \rangle$ (recall that $\hat{C}^{\mathsf{prefix}}$ is the prefix of the serialization of the context C that ends immediately before $\bullet$, while $\hat{C}^{\mathsf{suffix}}$ is the suffix that starts immediately after $\bar{\bullet}$; note that $\hat{C}^{\mathsf{suffix}} = \varepsilon$ if C is a horizontal context);
- for each binary label $[p : a]$, with $a \in \Sigma$, we introduce the opening macro-tag $\langle p : a \rangle$ and the closing macro-tag $\langle / a \rangle$;
- for each nullary label $[p : \varnothing]$, we introduce the macro-tags $\langle p : \varnothing \rangle$ and $\langle / \varnothing \rangle$.

The serialization $\hat{\sigma}$ of σ is defined in the same way as for unranked trees. Note that it is always possible to tell apart the opening macro-tags of unary symbols from the opening macro-tags of binary symbols, as the latter ones are of the form $\langle p : a \rangle$, where $\delta(p, a) = (p_1, p_2)$ and the components of p_1 and p_2 are different from that of p. In particular, this means that the serialization $\hat{\sigma}$ can be used as a representation of the decomposition σ.

Remark 1 To describe a transducer that receives as input a serialized tree $\hat{t}$ and produces the serialization $\hat{\sigma}$ of an $\mathcal{R}$-decomposition of t – and more generally to manipulate similar types of streams – it is convenient to think of an opening or closing macro-tag of the form $\langle p : \alpha \rangle$ or $\langle / \alpha \rangle$ as a string over the extended finite alphabet $\Sigma \uplus \bar{\Sigma} \uplus P \uplus \{ \langle, \rangle, /, :, \varnothing \}$. Through the rest of this section, we will adopt this latter presentation of macro-tags. Accordingly, we will assume that the serialization $\hat{\sigma}$ of an $\mathcal{R}$-decomposition is a string over a finite alphabet.

Example 2 (continued) The serialization of the $\mathcal{R}$-decomposition σ of Figure 5 begins with a string of the form

$$\langle p_0^r : \boldsymbol{r} \rangle \ \langle p_0^a : \boldsymbol{a} \rangle \ \langle p_1^a : \boldsymbol{aa} \rangle \ \langle p_1^a : \varnothing \rangle \ \langle / \varnothing \rangle \ \langle / \bar{\boldsymbol{a}}\bar{\boldsymbol{a}} \rangle \ \langle p_0^d : \boldsymbol{d} \rangle \ \langle p_0^b : \boldsymbol{b} \rangle \ \langle p_1^a : \boldsymbol{aa} \rangle \ \langle p_1^a : \varnothing \rangle \ \langle / \varnothing \rangle \ \ldots$$

Observe that the projection of the above string onto the letters of $\Sigma \uplus \bar{\Sigma}$ (written in bold for the sake of readability) gives exactly the serialization of the tree t.

The idea for producing a serialized decomposition $\hat{\sigma}$ from an input serialized tree $\hat{t}$ follows essentially the same construction that was given in the proof of

Lemma 1. The fact that this construction is compatible with a streaming processing stems from the crucial assumption that the automaton $\mathcal{R}$ is *top-down deterministic*. More precisely, one observes that for every node $\vec{\imath} \cdot j$ in the run ρ, the state $\rho(\vec{\imath} \cdot j)$ depends only on the labeling of the nodes of t that precede $\vec{\imath} \cdot j$ in the lexicographic order – this basically coincides with the notion of *state reached by* $\mathcal{R}$ on a prefix of $\hat{t}$ (cf. beginning of Section 3.2). The latter property allows one to construct the path π of the proof of Lemma 1 in a streaming fashion, namely, as longer and longer prefixes of the input serialization $\hat{t}$ are disclosed. In a similar way, and as long as the reached state stays in the initial component X_0 of $\mathcal{R}$, one can build up longer and longer prefixes of the opening macro-tag $\langle p_0 : \hat{C}^{\mathsf{prefix}} \rangle$ that corresponds to the first node of the $\mathcal{R}$-decomposition. We observe that the prefix $\langle p_0 :$ of this opening macro-tag can be output right at the beginning, and that the subsequent symbols of $\hat{C}^{\mathsf{prefix}}$ can be extracted in a streaming way from the input. When the path π cannot be extended further (e.g. when an input symbol induces a transition leaving X_0 with both successor states), the opening macro-tag $\langle p_0 : \hat{C}^{\mathsf{prefix}} \rangle$ is terminated and the construction may proceed in a recursive manner from the newly visited components. Note that the state reached in a new component is immediately determined thanks to top-down determinism and can play the same role as the initial state p_0 during the recursive processing. Moreover, once the sub-processing terminates, the closing macro-tag $\langle / \hat{C}^{\mathsf{suffix}} \rangle$ can be extracted from the continuation of the input.

We have just described informally a transducer $\mathcal{Z}_1$ that receives an input serialization $\hat{t}$ of a tree $t \in \mathscr{L}(\mathcal{R})$ and produces the serialization $\hat{\sigma}$ of an $\mathcal{R}$-decomposition of t. We omit the tedious details of the definition and correctness of $\mathcal{Z}_1$. Instead, we observe that the content of the input serialization $\hat{t}$ is reproduced unchanged inside the serialized decomposition $\hat{\sigma}$, i.e. $\hat{t}$ can be recovered from $\hat{\sigma}$ by replacing the macro-tags $\langle p : \alpha \rangle$ and $\langle / \alpha \rangle$ with their contents α in the form of strings. We can thus think of $\mathcal{Z}_1$ as a streaming repair process from the restriction language $\mathscr{L}(\mathcal{R})$ to the language of all serialized $\mathcal{R}$-decompositions. The aggregate cost of this repair process is linear in the number of macro-tags produced in the output (or, equally, in the number nodes of the decomposition. In particular, thanks to Lemma 1, the worst-case aggregate cost of $\mathcal{Z}_1$ is $2^{\mathcal{O}(|\mathsf{SCC}(\mathcal{R})|)}$.

4.2 Transducer $\mathcal{Z}_2$: simulating a play

The goal of the second transducer $\mathcal{Z}_2$ is to derive from its input – i.e. the serialized $\mathcal{R}$-decomposition $\hat{\sigma}$ of some tree $t \in \mathscr{L}(\mathcal{R})$ – a corresponding play for the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. The play will be generated by a sequence of moves taken alternatively by Generator, Repairer, and Referee, and interleaved with the opening and closing macro-tags from the input stream $\hat{\sigma}$. Intuitively, the moves of Generator are obtained by lifting the run of $\mathcal{R}$ on t to the level of the strongly connected components; the moves of Repairer instead are obtained from his winning strategy W (recall that this is a function mapping any position in the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ that is owned by Repairer to a sequence of prefix-rewriting steps $\overset{\mathcal{T}}{\mapsto}$).

We recall that the serialized decomposition $\hat{\sigma}$ contains macro-tags of the form $\langle p : \alpha \rangle$ or $\langle / \alpha \rangle$, which are seen as strings over the finite alphabet $\langle \Sigma \rangle = \Sigma \uplus \bar{\Sigma} \uplus P \uplus \{ \langle, \rangle, /, :, \varnothing \}$. In order to be able to annotate the serialized decompo-

sition $\hat{\sigma}$ with a corresponding play over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, we introduce an additional alphabet $\Xi = \mathsf{SCC}(\mathcal{R}) \uplus \mathsf{SCC}(\mathcal{T}) \uplus \{\triangleleft, [\![,]\!], \langle\!\langle, \rangle\!\rangle, (\!|, |\!)\}$. Hereafter, we will identify plays over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ with their string encodings over Ξ.

We begin by describing the crucial properties that must be satisfied by the output of $\mathcal{Z}_2$, which is a string over $\langle \Sigma \rangle \uplus \Xi$ called 'annotated decomposition'.

Definition 3 An *annotated $\mathcal{R}$-decomposition* is a sequence π of symbols in $\langle \Sigma \rangle \uplus \Xi$ such that:

- the projection of π onto $\langle \Sigma \rangle$ gives a serialized $\mathcal{R}$-decomposition $\hat{\sigma}$, devoid of all occurrences of macro-tags $\langle p : a \rangle$ and $\langle /a \rangle$, with $a \in \Sigma$;
- the projection of π onto Ξ gives a valid play over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$;
- let π' be a prefix of π and let $\mathsf{pos}(\pi')$ denote the position in the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ that is reached by the last move encoded in π'; if π' ends with an opening macro-tag $\langle p : \alpha \rangle$, or with a closing macro-tag matching $\langle p : \alpha \rangle$, then $\mathsf{pos}(\pi')$ is a position owned by Generator, say $\mathsf{pos}(\pi') = [\![\vec{x}, \vec{y}]\!]$, and the state p belongs to the component $\mathsf{top}(\vec{x})$ at the top of the stack $\vec{x}$.

Remark 2 From the definition of decomposition tree (cf. second item of Definition 2) and that of annotated decomposition (cf. the last item of Definition 3), we derive the following crucial properties. Suppose that π' is a prefix of an annotated $\mathcal{R}$-decomposition π ending with an opening macro-tag $\langle p : \hat{C}^{\mathsf{prefix}} \rangle$, and suppose that $\langle /\hat{C}^{\mathsf{suffix}} \rangle$ is the matching closing macro-tag in π. If $\mathsf{pos}(\pi') = [\![\vec{x}, \vec{y}]\!]$, then the state $\delta^\circ(p, C)$ is well defined and belongs to the component $\mathsf{top}(\vec{x})$. In particular, the context C belongs to the language $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x}))$. Moreover, we know from the definition of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ (cf. moves for Repairer in Definition 1) that $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}))$, and hence $C \in \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}))$. The above properties will be used later in Section 4.3 to transform an annotated decomposition of $t \in \mathscr{L}(\mathcal{R})$ into a decomposition of a tree t' in the target language $\mathscr{L}(\mathcal{T})$.

We now turn to describing how the transducer $\mathcal{Z}_2$ transforms the serialization of an $\mathcal{R}$-decomposition $\hat{\sigma}$ into an annotated $\mathcal{R}$-decomposition π by inserting appropriate moves of Generator, Repairer, and Referee.

While constructing the annotated decomposition, the transducer will focus on the occurrences in $\hat{\sigma}$ of the opening macro-tags of the form $\langle p : a \rangle$, with $a \in \Sigma$, $\delta(p, a) = (p_1, p_2)$, and with the components of p_1 and p_2 different from that of p. We call these tags *binary macro-tags*, in analogy with the binary symbols $[p : a]$ of the ranked alphabet of the decomposition. Recall that every occurrence of a binary macro-tag in $\hat{\sigma}$ originates from a transition of $\mathcal{R}$ that induces a change of component in both successor states: this is precisely the moment where we need to simulate a push-and-swap move of Generator. Similarly, we need to simulate a pop move of Generator each time we complete a visit of a left or right subtree rooted at a binary node of the decomposition.

It is convenient to describe the transducer $\mathcal{Z}_2$ as a recursive editing process. To correctly exploit recursion, we need to assume that the output π depends, not only on the serialized decomposition $\hat{\sigma}$, but also on a parameter $\langle\!\langle \vec{x}_0 , \vec{y}_0 \rangle\!\rangle$ that represents a generic initial position of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ – this position is owned by Repairer and during the recursive calls will be updated so as to construct a valid play. To stress the dependency of the annotated decomposition π in terms of both parameters $\hat{\sigma}$

and $\langle\!\langle\, \vec{x}_0 \,,\, \vec{y}_0 \,\rangle\!\rangle$, we will use the functional notation $\pi\big(\hat{\sigma}, \langle\!\langle\, \vec{x}_0 \,,\, \vec{y}_0 \,\rangle\!\rangle\big)$. Moreover, we will assume that recursive calls can be made on any *infix* of a serialized decomposition $\hat{\sigma}$, provided that the binary macro-tags in it are well-matched.

The first step of the editing process consists of applying Repairer's winning strategy W to the initial position $\langle\!\langle\, \vec{x}_0 \,,\, \vec{y}_0 \,\rangle\!\rangle$ and derive in this way the first move that will be inserted in the annotated decomposition:

$$W\big(\langle\!\langle\, \vec{x}_0 \,,\, \vec{y}_0 \,\rangle\!\rangle\big) \;=\; \begin{cases} \langle\!\langle\, \vec{x}_0 \,,\, \vec{y}_0 \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_0 \,,\, \vec{y}_0' \,]\!] & \text{if } \mathsf{top}(\vec{x}_0) \text{ is horizontal,} \\ \langle\!\langle\, \vec{x}_0 \,,\, \vec{y}_0 \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} (\!|\, \vec{x}_0 \,,\, \vec{y}_0' \,|\!) & \text{otherwise.} \end{cases}$$

If the component $\mathsf{top}(\vec{x}_0)$ at the top of the stack $\vec{x}_0$ is non-horizontal, then the above move is immediately followed by a move of Referee, which inserts the separator symbol $\triangleleft$ below the top elements of the two stacks $\vec{x}_0, \vec{y}_0'$; this results in a move of the form

$$(\!|\, \vec{x}_0 \,,\, \vec{y}_0' \,|\!) \overset{\mathsf{Ref}}{\mapsto} [\![\, \vec{x}_1 \,,\, \vec{y}_1 \,]\!] \qquad \text{where} \qquad \begin{cases} \vec{x}_1 \;=\; \mathsf{top}(\vec{x}_0) \cdot \triangleleft \cdot \mathsf{tail}(\vec{x}_0) \\ \vec{y}_1 \;=\; \mathsf{top}(\vec{y}_0') \cdot \triangleleft \cdot \mathsf{tail}(\vec{y}_0') \end{cases}$$

Otherwise, if the top component $\mathsf{top}(\vec{x}_0)$ is horizontal, we simply let $\vec{x}_1 = \vec{x}_0$ and $\vec{y}_1 = \vec{y}_0'$.

We now focus on the first occurrence in $\hat{\sigma}$ of a binary opening macro-tag $\langle p : a \rangle$, together with the matching occurrence of the closing macro-tag $\langle /a \rangle$ (if there are no such occurrences, then we simply skip all the following steps and we just append to the previous moves the annotated decomposition $\hat{\sigma}$). We factorize $\hat{\sigma}$ as follows:

$$\hat{\sigma} \;=\; \hat{\sigma}_1 \cdot \langle p : a \rangle \cdot \hat{\sigma}_2 \cdot \hat{\sigma}_3 \cdot \langle /a \rangle \cdot \hat{\sigma}_4$$

where $\hat{\sigma}_1$ is a prefix of $\hat{\sigma}$ (not necessarily a well-matched string) that contains no occurrences of binary macro-tags, $\hat{\sigma}_2$ and $\hat{\sigma}_3$ are well-matched infixes encoding valid $\mathcal{R}$-decomposition (sub-)trees, and $\hat{\sigma}_4$ is a suffix of $\hat{\sigma}$ (not necessarily well-matched, but containing well-matched binary macro-tags).

The prefix $\hat{\sigma}_1$ of the input will be reproduced by the transducer without modifications. On the other hand, the first opening binary macro-tag $\langle p : a \rangle$ will be replaced by the following move of Generator:

$$[\![\, \vec{x}_1 \,,\, \vec{y}_1 \,]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle\, \vec{x}_1' \,,\, \vec{y}_1 \,\rangle\!\rangle \qquad \text{where} \qquad \begin{cases} \vec{x}_1' \;=\; X_1\, X_2 \cdot \mathsf{tail}(\vec{x}_1) \\ X_1 \;=\; \text{component of } p_1 \\ X_2 \;=\; \text{component of } p_2 \\ (p_1, p_2) \;=\; \delta(p, a). \end{cases}$$

Note that the symbol a in the binary macro-tag $\langle p : a \rangle$ gets lost during the transduction. This, however, is not problematic for the repair process, since only boundedly many such symbols can disappear. Also observe that the new position $\langle\!\langle\, \vec{x}_1' \,,\, \vec{y}_1 \,\rangle\!\rangle$ is owned by Repairer (i.e. Referee does not enter the game here).

The transducer will then process the infix $\hat{\sigma}_2$ in a recursive manner, starting from position $\langle\!\langle\, \vec{x}_1' \,,\, \vec{y}_1 \,\rangle\!\rangle$ and outputting an annotated decomposition $\pi\big(\hat{\sigma}_2, \langle\!\langle\, \vec{x}_1' \,,\, \vec{y}_1 \,\rangle\!\rangle\big)$. After the first recursive call, the most recent position reached by the annotated decomposition is owned by Generator; we denote this position

by $[\![\vec{x}_2 , \vec{y}_2]\!]$. Moreover, since the left subtree of a binary node in the decomposition σ has just been visited, the transducer appends a pop move of the form

$$[\![\vec{x}_2 , \vec{y}_2]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle \vec{x}'_2 , \vec{y}_2 \rangle\!\rangle \quad \text{whenever } \mathsf{top}(\vec{x}'_2) \neq \triangleleft, \text{ or}$$
$$[\![\vec{x}_2 , \vec{y}_2]\!] \overset{\mathsf{Gen}}{\mapsto} (\!| \vec{x}'_2 , \vec{y}_2 |\!) \quad \text{otherwise.}$$

where $\vec{x}'_2 = \mathsf{tail}(\vec{x}_2)$. If the top element of the stack $\vec{x}'_2$ happens to be a separator symbol $\triangleleft$, then the appropriate move of Referee is also added:

$$(\!| \vec{x}'_2 , \vec{y}_2 |\!) \overset{\mathsf{Ref}}{\mapsto} \langle\!\langle \vec{x}_3 , \vec{y}_3 \rangle\!\rangle$$

(otherwise, we simply assume $\vec{x}_3 = \vec{x}'_2$ and $\vec{y}_3 = \vec{y}_2$).

Subsequently, the transducer makes a second recursive call starting from position $\langle\!\langle \vec{x}_3 , \vec{y}_3 \rangle\!\rangle$ and transforming the infix $\hat{\sigma}_3$ into the annotated decomposition $\pi\big(\hat{\sigma}_3, \langle\!\langle \vec{x}_3 , \vec{y}_3 \rangle\!\rangle\big)$. As before, a new position $[\![\vec{x}_4 , \vec{y}_4]\!]$ owned by Generator is reached and a second pop move is inserted, possibly followed by an appropriate move of Referee:

$$[\![\vec{x}_4 , \vec{y}_4]\!] \overset{\mathsf{Gen}}{\mapsto} (\!| \vec{x}'_4 , \vec{y}_4 |\!) \overset{\mathsf{Ref}}{\mapsto} \langle\!\langle \vec{x}_5 , \vec{y}_5 \rangle\!\rangle \qquad \text{where} \qquad \vec{x}'_4 \;=\; \mathsf{tail}(\vec{x}_4)$$

(again, we assume that $\vec{x}_5 = \vec{x}'_4$ and $\vec{y}_5 = \vec{y}_4$ if Referee does not enter the game).

Finally, the transducer removes the next incoming closing macro-tag $\langle /a \rangle$ and makes a third recursive call to edit the remaining suffix $\hat{\sigma}_4$. This results in a sequence $\pi\big(\hat{\sigma}_4, \langle\!\langle \vec{x}_5 , \vec{y}_5 \rangle\!\rangle\big)$.

Putting all together, the output produced by $\mathcal{Z}_2$ is a sequence of the form

$$\begin{aligned}
\pi\big(\hat{\sigma}, \langle\!\langle \vec{x}_0 , \vec{y}_0 \rangle\!\rangle\big) \quad =^{\mathrm{def}} \quad & \langle\!\langle \vec{x}_0 , \vec{y}_0 \rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} (\!| \vec{x}_0 , \vec{y}'_0 |\!) \overset{\mathsf{Ref}}{\mapsto} [\![\vec{x}_1 , \vec{y}_1]\!] \;\cdot \\
& \hat{\sigma}_1 \;\cdot\; [\![\vec{x}_1 , \vec{y}_1]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle \vec{x}'_1 , \vec{y}_1 \rangle\!\rangle \;\cdot \\
& \pi\big(\hat{\sigma}_2, \langle\!\langle \vec{x}'_1 , \vec{y}_1 \rangle\!\rangle\big) \;\cdot\; [\![\vec{x}_2 , \vec{y}_2]\!] \overset{\mathsf{Gen}}{\mapsto} (\!| \vec{x}'_2 , \vec{y}_2 |\!) \overset{\mathsf{Ref}}{\mapsto} \langle\!\langle \vec{x}_3 , \vec{y}_3 \rangle\!\rangle \;\cdot \\
& \pi\big(\hat{\sigma}_3, \langle\!\langle \vec{x}_3 , \vec{y}_3 \rangle\!\rangle\big) \;\cdot\; [\![\vec{x}_4 , \vec{y}_4]\!] \overset{\mathsf{Gen}}{\mapsto} (\!| \vec{x}'_4 , \vec{y}_4 |\!) \overset{\mathsf{Ref}}{\mapsto} \langle\!\langle \vec{x}_5 , \vec{y}_5 \rangle\!\rangle \;\cdot \\
& \pi\big(\hat{\sigma}_4, \langle\!\langle \vec{x}_5 , \vec{y}_5 \rangle\!\rangle\big).
\end{aligned}$$

It is easy to verify that if $\hat{\sigma}$ is the serialization of a complete $\mathcal{R}$-decomposition tree and $\langle\!\langle \vec{x}_0 , \vec{y}_0 \rangle\!\rangle$ is the initial position of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, then $\pi\big(\hat{\sigma}, \langle\!\langle \vec{x}_0 , \vec{y}_0 \rangle\!\rangle\big)$ is an annotated $\mathcal{R}$-decomposition for $\hat{\sigma}$. In particular, the projection of this sequence onto the alphabet Ξ is a valid play for the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, while the projection onto $\Sigma \uplus \bar{\Sigma}$ is a sub-sequence of the serialization $\hat{t}$ of a tree in $\mathscr{L}(\mathcal{R})$.

We have just described a transducer $\mathcal{Z}_2$ that transforms an input serialized $\mathcal{R}$-decomposition $\hat{\sigma}$ into an annotated $\mathcal{R}$-decomposition π. We observe that, for a fixed winning strategy W, the amount of editing required by this transformation is linear in the number of occurrences of macro-tags in $\hat{\sigma}$.

4.3 Transducer $\mathcal{Z}_3$: choosing states in the target

This subsection focuses on the third transducer $\mathcal{Z}_3$ and it is the place where many pieces of the puzzle so far outlined come together. We will exploit, for instance, properties derived from the rules of the simulation game over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, as

well as the notion of annotated decomposition. We will also see how exactly the moves of Referee and the use of separator symbols come into play.

Before entering the details, we recall that the transducer $\mathcal{Z}_3$ receives as input an annotated $\mathcal{R}$-decomposition, which consists of a serialized decomposition of a tree annotated with partial runs of $\mathcal{R}$ on the factors and with a corresponding play inside the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Intuitively, the goal of $\mathcal{Z}_3$ is to transform the partial runs of $\mathcal{R}$ into partial runs of $\mathcal{T}$ on the same factors. Thanks to a technical lemma that we shall disclose soon, such a transformation of partial runs can be derived from the moves of Repairer that annotate the input $\mathcal{R}$-decomposition (further details will be given after Definition 4).

We begin by showing how to extract partial runs of $\mathcal{T}$ from an annotated $\mathcal{R}$-decomposition. The key lemma exploits the containment of languages of contexts realizable within single components of the restriction and target automata. To formalize the statement, we need to parametrize languages of realizable contexts with respect to a specific initial state. Formally, for a given top-down deterministic tree automaton $\mathcal{A} = (\Sigma, P, \delta, p_0, F)$, a given component X of $\mathcal{A}$, and a given state $p \in X$, we define the language of all *contexts realizable in X starting from state p* as follows:

$$\mathscr{L}(\mathcal{A}|_p X) \;=^{\text{def}}\; \big\{C : \delta^\circ(p, C) \in X\big\}.$$

Clearly, we have:

$$\mathscr{L}(\mathcal{A} \mid X) \;=\; \bigcup_{p \in X} \mathscr{L}(\mathcal{A}|_p X).$$

Lemma 2 *Let X be a component of the restriction automaton $\mathcal{R}$ and let Y be a component of the target automaton $\mathcal{T}$ such that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$. Then,*

$$\forall\, p \in X.\ \exists\, q \in Y.\ \mathscr{L}(\mathcal{R}|_p X) \subseteq \mathscr{L}(\mathcal{T}|_q Y).$$

Proof. Let $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$, $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$, $X \in \mathsf{SCC}(\mathcal{R})$, and $Y \in \mathsf{SCC}(\mathcal{T})$. We prove the contrapositive of the claim by exploiting an induction on the size of the component Y. More precisely, we assume that $Y = \{q_1, \ldots, q_n\}$ and that there is $p \in X$ such that $\mathscr{L}(\mathcal{R}|_p X) \nsubseteq \mathscr{L}(\mathcal{T}|_q Y)$ for all $q \in Y$. From this we derive the existence of a sequence of contexts $C_1, \ldots, C_n$ such that, for all $1 \le j \le i \le n$,

$$C_i \in \mathscr{L}(\mathcal{R}|_p X) \setminus \mathscr{L}(\mathcal{T}|_{q_j} Y).$$

Note that the non-containment $\mathscr{L}(\mathcal{R} \mid X) \nsubseteq \mathscr{L}(\mathcal{T} \mid Y)$ follows immediately from letting $i = n$.

For the base case $i = 1$, we use the assumption $\mathscr{L}(\mathcal{R}|_p X) \nsubseteq \mathscr{L}(\mathcal{T}|_{q_1} Y)$ to derive the existence of a context $C_1 \in \mathscr{L}(\mathcal{R}|_p X) \setminus \mathscr{L}(\mathcal{T}|_{q_1} Y)$.

The idea for the inductive step consists of combining a context C_i obtained from the inductive hypothesis and a context C' witnessing the non-containment $\mathscr{L}(\mathcal{R}|_p X) \nsubseteq \mathscr{L}(\mathcal{T}|_{q_{i+1}} Y)$; to correctly combine these contexts, a third context C will be inserted, which connect pairs of states in the same component X of $\mathcal{R}$. More precisely, let p be some state in X and C_i some context satisfying the inductive hypothesis $C_i \in \mathscr{L}(\mathcal{R}|_p X) \setminus \mathscr{L}(\mathcal{T}|_{q_j} Y)$ for all $1 \le j \le i$. We consider the behavior of the target automaton $\mathcal{T}$ on the context C_i starting from state q_{i+1}. If $\gamma^\circ(q_{i+1}, C_i)$ is undefined or does not belong to the component Y, then we simply let $C_{i+1} = C_i$ so as to get $C_{i+1} \notin \mathscr{L}(\mathcal{T}|_{q_j} Y)$ for all $j \le i+1$. Otherwise, we let $q'_{i+1} = \gamma^\circ(q_{i+1}, C_i)$ $(\in Y)$. We recall that $C_i \in \mathscr{L}(\mathcal{R}|_p X)$ and we let $p' = \delta^\circ(p, C_i) \in X$. Since the

states p, p' in X are mutually reachable, we know that there exists a context C such that $\delta^\circ(p', C) = p$, whence $\delta^\circ(p, C_i \circ C) = p$. We then use the original assumption $\mathscr{L}(\mathcal{R}|_p X) \not\subseteq \mathscr{L}(\mathcal{T}|_{q'_{i+1}} Y)$ to derive the existence of a third context C' in $\mathscr{L}(\mathcal{R}|_p X) \setminus \mathscr{L}(\mathcal{T}|_{q'_{i+1}} Y)$. Putting everything together, we see that the context $C_{i+1} = C_i \circ C \circ C'$ belongs to $\mathscr{L}(\mathcal{R}|_p X)$, but not to $\mathscr{L}(\mathcal{T}|_{q_j} Y)$, for all $j \leq i+1$. □

We now introduce the concept of 'partial decomposition' for the target automaton $\mathcal{T}$. This concept is aimed towards defining the output of the third transducer $\mathcal{Z}_3$ and can be seen as a weakening of the notion of $\mathcal{R}$-decomposition (Definition 2). In a similar way as we did for $\mathcal{R}$-decompositions, we introduce an infinite alphabet $[\Delta]$ consisting of

- symbols of the form $[q : C]$, where q is a state of $\mathcal{T}$ and C is a context over Δ,
- symbols of the form $[q : a]$, where q is a state of $\mathcal{R}$ and a is a letter in Δ.

The main differences with Definition 2 are that (i) the underlying alphabet is now unranked (in particular, the nodes of the partial decomposition that are labeled by $[q : C]$ can be either internal nodes or leaves), (ii) the contexts C associated with the nodes of the form $[q : C]$ are not anymore assumed to be realized within a single component, (iii) the successor states $(q_1, q_2) = \gamma(q, a)$ associated with a node of the form $[q : a]$ could remain inside the same component, and (iv) for a node $[q : C]$ and its successor $[q' : \alpha]$, the states q' and $\gamma^\circ(q, C)$ may be distinct, but still reachable one from the other.

Definition 4 Let $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ be a target automaton. A *partial* $\mathcal{T}$*-decomposition* is an unranked tree τ labeled over $[\Delta]$ such that:

- for every node labeled with $[q : C]$, the state $q_1 = \gamma^\circ(q, C)$ is well defined; furthermore, if the node is not a leaf, then it has exactly one child labeled with a symbol of the form $[q'_1 : \alpha]$, with q'_1 belonging to the same component as q_1;
- for every node labeled with $[q : a]$, the pair of states $(q_1, q_2) = \gamma(q, a)$ is well defined; furthermore, the node has two children labeled with symbols of the form $[q'_1 : \alpha_1]$ and $[q'_2 : \alpha_2]$, with q'_1 is in the same component as q_1, and q'_2 is in the same component as q_2.

The serialization of a partial $\mathcal{T}$-decomposition τ consists, as usual, of opening macro-tags of the form $\langle q : \hat{C}^{\mathsf{prefix}} \rangle$ or $\langle q : a \rangle$, matched by closing macro-tags of the form $\langle / \hat{C}^{\mathsf{suffix}} \rangle$ or $\langle / \bar{a} \rangle$.

As we already mentioned, the goal of transducer $\mathcal{Z}_3$ is to transform an annotated $\mathcal{R}$-decomposition π into the serialization of some partial $\mathcal{T}$-decomposition τ. Intuitively, this is done by replacing all opening macro-tags $\langle p : \hat{C}^{\mathsf{prefix}} \rangle$ in π with corresponding macro-tags of the form $\langle q : \hat{C}^{\mathsf{prefix}} \rangle$, where q is a state of the target automaton whose dependency from p is derived from Lemma 2. In addition, binary nodes of the form $[q : a]$ will be inserted in the partial decomposition tree in order to have some branching: these nodes will be derived from the atomic push-and-swap moves that were chosen by Repairer and encoded in the annotated decomposition π. Finally, we will implicitly perform a 'repositioning' of some closing macro-tags $\langle / \hat{C}^{\mathsf{suffix}} \rangle$ inside π, but only under the proviso that the encoded context C is horizontal (in this case, we observe that $\hat{C}^{\mathsf{suffix}} = \varepsilon$ and that the repositioning can be done at bounded cost). Roughly speaking, the repositioning of

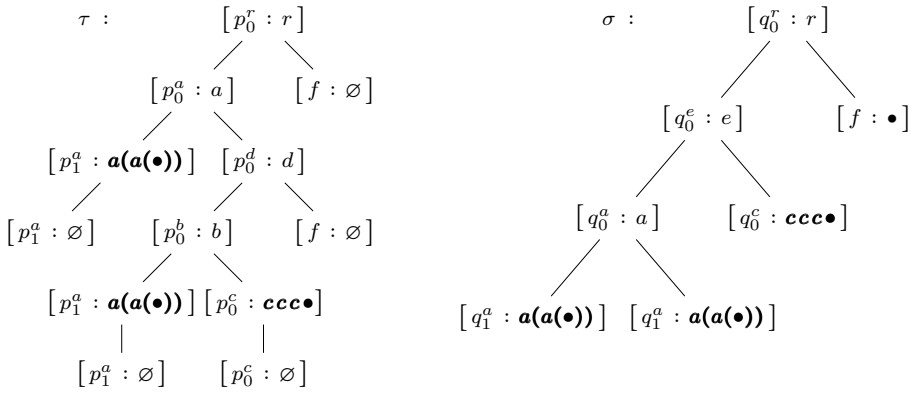


Fig. 6 An $\mathcal{R}$-decomposition τ and a corresponding partial $\mathcal{T}$-decomposition σ.

closing macro-tags with horizontal contexts corresponds to hedge movements that are induced by deletions and insertions of nodes inside concrete unranked trees (see next example). As a matter of fact, a similar phenomenon was taken into account also in [18] for characterizing bounded repairability in the non-streaming setting.

Example 2 (continued) In Figure 6 we reproduce the $\mathcal{R}$-decomposition σ of the tree $t \in \mathscr{L}(\mathcal{R})$ of Figure 5, and we give the corresponding partial $\mathcal{T}$-decomposition τ that is produced by $\mathcal{Z}_3$ (due to space constraints, we omit the intermediate annotated $\mathcal{R}$-decomposition π that forms the input of this transducer). We observe that the single-letter nodes from σ are removed during the process, and new binary nodes are added. Moreover, all unary nodes of σ are copied into τ, with only small changes due to the new associated states and the repositioning of the node with the horizontal context $c\,c\,c\,\bullet$.

Lemma 3 below focuses on transforming an annotated $\mathcal{R}$-decomposition π into a partial $\mathcal{T}$-decomposition τ. During the transformation it is important to preserve most of the string content from the input – this will be useful to prove that the global editing strategy has uniformly bounded cost. Specifically, we will consider the *order of occurrence* of the non-empty strings of the form $\hat{C}^{\mathsf{prefix}}$ that appear inside the opening macro-tags of the annotated $\mathcal{R}$-decomposition π. Formally, by this order of occurrence we mean the maximal sequence of non-empty strings $\hat{C}_0^{\mathsf{prefix}}, \ldots, \hat{C}_n^{\mathsf{prefix}}$ for which there exists a sub-sequence of π of the form $\langle p_0 : \hat{C}_0^{\mathsf{prefix}} \rangle, \ldots, \langle p_n : \hat{C}_n^{\mathsf{prefix}} \rangle$. The lemma below shows precisely how to transform π into a serialized partial $\mathcal{T}$-decomposition $\hat{\tau}$ by preserving the occurrence order of the non-empty strings $\hat{C}^{\mathsf{prefix}}$. Later, we will see how to exploit this property in order to compute the same transformation by means of a transducer with uniformly bounded cost.

Lemma 3 *For every annotated $\mathcal{R}$-decomposition π, there is a partial $\mathcal{T}$-decomposition τ such that the occurrence order in π of the non-empty strings of the form $\hat{C}^{\mathsf{prefix}}$ is preserved in the sequence $\hat{\tau}$.*

Proof. We begin with a series of assumptions related to the given annotated $\mathcal{R}$-decomposition π. Here, we will mostly overlook the moves of Generator and Referee

that appear in π, as these were already exploited for finding the corresponding moves of Repairer and the corresponding positions in the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. It will be also convenient to think of π as a sequence consisting of macro-tags $\langle p : \alpha \rangle$ and $\langle / \alpha \rangle$ interleaved with prefix-rewriting operations on the stack controlled by Repairer. More precisely, we encode each move of Repairer that appears in π by a sequence of basic prefix-rewriting steps of the form $Y \mapsto Y_1 Y_2$ or $Y \mapsto \varepsilon$ – note that the encoding omits the content of the stacks below the top elements, since this can be easily reconstructed from the basic push-and-swap and pop operations (see Section 3.2). For the sake of brevity, we call *atomic operations* the basic prefix-rewriting steps that occur in π and are applied to single components of $\mathcal{T}$. We also assume, without loss of generality, that the sequence π is such that the last visited position $\mathsf{pos}(\pi)$ is owned by Generator and is of the form $[\![\vec{x}, \vec{y}]\!]$, with $\vec{y}$ singleton (if this is not the case, it is sufficient to append to π a sequence of atomic pop operations acting on the stack controlled by Repairer).

The proof of the lemma is based on an induction on the nesting structure of the atomic operations executed by Repairer in the annotated decomposition π. To formalize the induction principle, we associate with each *factor* (i.e. infix) π' of π a number $\mathsf{height}(\pi')$ that is obtained by counting the occurrences of push-and-swap operations and subtracting the number of occurrences of pop operations. Intuitively, the value $\mathsf{height}(\pi')$ describes the difference in the height of the stack controlled by Repairer at the beginning and at the end of the factor π' (separator symbols do not count for this height). We then say that a factor π' is *well-balanced* if we have $\mathsf{height}(\pi') = 0$ and $\mathsf{height}(\pi'') \geq 0$ for all prefixes π'' of π'. Furthermore, we say that an occurrence of a well-balanced factor π' is *delimited* inside π if it is either π itself or is surrounded by atomic operations (i.e. $Y \mapsto Y_1 Y_2$ and $Y \mapsto \varepsilon$). Finally, we denote by $\mathsf{maxheight}(\pi')$ the maximum of $\mathsf{height}(\pi'')$ over all prefixes π'' of π', and by $\mathsf{grounds}(\pi')$ the number prefixes π'' of π' such that $\mathsf{height}(\pi'') = 0$.

We now turn to the core proof of the lemma, which associates with each delimited occurrence π' of a well-balanced factor of π a corresponding partial $\mathcal{T}$-decomposition $\tau(\pi')$. The proof exploits a double induction based on the parameters $\mathsf{maxheight}(\pi')$ and $\mathsf{grounds}(\pi')$, where the former is the dominant one.

The base case amounts to considering a delimited occurrence of a well-balanced factor π' of π such that $\mathsf{maxheight}(\pi') = 0$. By construction, the considered factor π' contains no atomic operations. In particular, π' contains a series of opening macro-tags of the form $\langle p : \hat{C}^{\mathsf{prefix}} \rangle$, which are matched by corresponding closing macro-tags either inside or outside π'. Suppose that these opening macro-tags occur in π' with the following order

$$\langle p_1 : \hat{C}_1^{\mathsf{prefix}} \rangle \dots \langle p_2 : \hat{C}_2^{\mathsf{prefix}} \rangle \dots \langle p_n : \hat{C}_n^{\mathsf{prefix}} \rangle \dots$$

and let $\langle / \hat{C}_1^{\mathsf{suffix}} \rangle$, $\langle / \hat{C}_2^{\mathsf{suffix}} \rangle$, …, $\langle / \hat{C}_n^{\mathsf{suffix}} \rangle$ be the matching occurrences of the corresponding closing macro-tags (not necessarily inside π'). We further let $[\![\vec{x}, \vec{y}]\!]$ be the most recent position that is reached in the annotated decomposition π immediately before the occurrence of the factor π'. Because $\mathsf{maxheight}(\pi') = 0$, in π' there is no atomic operation on the stack controlled by Repairer, and hence all the states $p_1, p_2, \dots, p_n$ belong to the same component $\mathsf{top}(\vec{x})$ of $\mathcal{R}$. Moreover, we know from the definition of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ that

$$\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y})).$$

Thus, we can apply Lemma 2 to the states $p_i \in \mathsf{top}(\vec{x})$ and obtain in this way some corresponding states $q_i \in \mathsf{top}(\vec{y})$ such that

$$C_i \in \mathscr{L}(\mathcal{T}|_{q_i}\, \mathsf{top}(\vec{y})).$$

We can now define the partial $\mathcal{T}$-decomposition $\tau(\pi')$ associated with π'. Intuitively, this has the shape of a vertical chain of unary nodes $[q_1 : C_1]$, $[q_2 : C_2]$, …, $[q_n : C_n]$. Formally, we let

$$\tau(\pi') \;=^{\mathrm{def}}\; [q_1 : C_1]\,\Big(\,[q_2 : C_2]\,\big(\ldots[q_n : C_n]\ldots\big)\,\Big).$$

However, since we need to always construct *non-empty* partial $\mathcal{T}$-decompositions, we need to treat in a different way the degenerate case where $n = 0$. Specifically, if π' contains no macro-tag of the form $\langle p : \hat{C}^{\mathsf{prefix}}\rangle$, then we let $\tau(\pi')$ consist of a single dummy node of the form $[q : \bullet]$, where q is any arbitrary state in the component $\mathsf{top}(\vec{y})$ and $\bullet$ denotes the trivial context. It is straightforward to see that, in any case, the occurrence order of the strings of the form $\hat{C}^{\mathsf{prefix}}$ inside π' is preserved in the transformed sequence $\widehat{\tau(\pi')}$.

We now give a proof of the inductive step. For this we consider a delimited occurrence π' of a well-balanced factor of π, with $\mathsf{maxheight}(\pi') > 0$. The factor π' contains at least one atomic operation on the stack controlled by Repairer. Clearly, the first atomic operation in π' must be a push-and-swap operation of the form $Y \mapsto Y_1\,Y_2$, and this must be eventually followed by a corresponding pop operation of the form $Y' \mapsto \varepsilon$. We accordingly factorize π' as follows:

$$\pi' \;=\; \pi_1 \cdot Y \mapsto Y_1\,Y_2 \cdot \pi_2 \cdot Y' \mapsto \varepsilon \cdot \pi_3$$

where π_1, π_2, π_3 are delimited occurrences of well-balanced factors such that $\mathsf{maxheight}(\pi_1) = 0$, $\mathsf{maxheight}(\pi_2) < \mathsf{maxheight}(\pi')$, $\mathsf{maxheight}(\pi_3) \leq \mathsf{maxheight}(\pi')$, and $\mathsf{grounds}(\pi_3) < \mathsf{grounds}(\pi')$. Using the inductive hypothesis we can transform π_1, π_2, π_3 into corresponding partial $\mathcal{T}$-decompositions $\tau(\pi_1)$, $\tau(\pi_2)$, $\tau(\pi_3)$, whose serializations preserve the occurrence order of the strings of the form $\hat{C}^{\mathsf{prefix}}$ inside π_1, π_2, π_3, respectively. Moreover, we observe that the partial $\mathcal{T}$-decomposition $\tau(\pi_1)$ is a vertical chain of nodes of the form $[q : C]$; we can thus write $\tau(\pi_1)\,(\tau')$ to denote the tree obtained by attaching a sub-tree τ' to the leaf of $\tau(\pi_1)$. Towards a conclusion, we recall that $Y \mapsto Y_1\,Y_2$ is an atomic operation of the prefix-rewriting system $\overset{\mathcal{T}}{\mapsto}$ and we derive from this the existence of a transition of $\mathcal{T}$ of the form $\gamma(q, a) = (q_1, q_2)$, with $q \in Y$, $a \in \Delta$, $q_1 \in Y_1$, $q_2 \in Y_2$. Finally, we transform the factor π' into the following partial $\mathcal{T}$-decomposition:

$$\tau(\pi') \;=^{\mathrm{def}}\; \tau(\pi_1)\,\Big(\,[q : a]\,\big(\,\tau(\pi_2),\,\tau(\pi_3)\,\big)\,\Big).$$

As before, it is straightforward to see the occurrence order of the strings of the form $\hat{C}^{\mathsf{prefix}}$ in π' is preserved in the transformed sequence $\widehat{\tau(\pi')}$. □

We now turn to explaining how the construction given in the above lemma can be implemented by means of a transducer $\mathcal{Z}_3$. For this we will reuse some of the concepts that were introduced in the proof, in particular, the notion of delimited occurrence of a well-balanced factor of π.

A first inspection of the proof of Lemma 3 reveals that the ancestor relation on the nodes of the partial $\mathcal{T}$-decomposition τ respects the nesting structure of the delimited occurrences of well-balanced factors of π. As a consequence, the preorder visit of the nodes $[q : C]$ in τ follows the exact ordering of the corresponding opening macro-tags $\langle p : \hat{C}^{\mathsf{prefix}} \rangle$ inside π: this allows us to correctly gather the strings $\hat{C}^{\mathsf{prefix}}$ that will form the opening macro-tags of the serialization $\hat{\tau}$ of τ. What remains to verify is that the preorder visit of the nodes $[q : C]$ in τ is also compatible with the ordering of the matching closing macro-tags $\langle / \hat{C}^{\mathsf{suffix}} \rangle$, with the only possible exception of some horizontal contexts C, for which we have $\hat{C}^{\mathsf{suffix}} = \varepsilon$ – this means that, in any case, the strings $\hat{C}^{\mathsf{suffix}}$ are determined before closing the macro-tags of the corresponding nodes of the partial decomposition τ. These constraints are formalized in the lemma below as a matching property between the opening and closing macro-tags inside a delimited occurrence of a well-balanced factor of π. The proof exploits in a crucial way the rules of the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ and, in particular, the role of Referee in inserting/removing separator symbols at the appropriate moments.

Lemma 4 *Let π be an annotated $\mathcal{R}$-decomposition. Inside any delimited occurrence of a well-balanced factor of π, all opening and closing macro-tags are well-matched, with the only exception of the macro-tags of the form $\langle p : \hat{C}^{\mathsf{prefix}} \rangle$ and $\langle / \hat{C}^{\mathsf{suffix}} \rangle$ where C is a horizontal context.*

Proof. We consider a delimited occurrence π' of a well-balanced factor of π and we accordingly factorize π as $\pi_1 \cdot \pi' \cdot \pi_2$, where π_1 ends with an atomic operation and π_2 begins with an atomic operation. We then consider an occurrence of an opening macro-tag $\langle p : \hat{C}^{\mathsf{prefix}} \rangle$ inside π', where C is a vertical context. We need to argue that the factor π' contains the matching occurrence of the closing macro-tag $\langle / \hat{C}^{\mathsf{suffix}} \rangle$ (symmetric arguments can be used to prove that all closing macro-tags are matched by opening macro-tags in the same factor π').

The definition of annotated decomposition (cf. third item of Definition 3) implies that the most recent position of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ that is visited before the occurrence of the opening macro-tag $\langle p : \hat{C}^{\mathsf{prefix}} \rangle$ is owned by Generator and it is of the form $[\![\vec{x}, \vec{y}]\!]$, with $p \in \mathsf{top}(\vec{x})$.

We also recall that the projection of π onto the opening and closing macro-tags gives the serialization of some $\mathcal{R}$-decomposition tree, devoid of binary nodes. In its turn, the definition of $\mathcal{R}$-decomposition (cf. second item of Definition 2) implies $\delta^{\circ}(p, C) \in \mathsf{top}(\vec{x})$. Since C is a vertical context, the component $\mathsf{top}(\vec{x})$ at the top of the stack $\vec{x}$ must be non-horizontal.

We then recall the rules of the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ (cf. first item of the moves of Referee, Definition 1) and we observe that Repairer and Referee must have entered the game just before the occurrence of the opening macro-tag $\langle p : \hat{C}^{\mathsf{prefix}} \rangle$. In particular, some separators must be placed below the top components of the two stacks $\vec{x}$ and $\vec{y}$, say $\vec{x} = X \cdot \triangleleft \cdot \vec{x}''$ and $\vec{y} = Y \cdot \triangleleft \cdot \vec{y}''$.

Now, since π' is a delimited occurrence of a well-balanced factor, we know that $\mathsf{height}(\pi_1) = \mathsf{height}(\pi_1 \cdot \pi_2) \leq |\vec{y}|$ and that either π_2 is empty or it begins with a pop operation applied to the stack controlled by Repairer. If π_2 is empty, then π' clearly contains the matching occurrence of the closing macro-tag $\langle / \hat{C}^{\mathsf{suffix}} \rangle$. Otherwise, we observe that after executing the first pop operation in π_2, at least the top component Y is removed from the stack $\vec{y} = Y \cdot \triangleleft \cdot \vec{y}''$. However, because $\vec{y}$

has a separator just below Y, the only way the component Y can be removed from $\vec{y}$ is via a move of Generator followed by a move of Referee, that is, via a sequence of operations of the form:

$$[\![X \cdot \triangleleft \cdot \vec{x}'', Y \cdot \triangleleft \cdot \vec{y}'']\!] \overset{\mathsf{Gen}}{\mapsto} (\!| \triangleleft \cdot \vec{x}'', Y \cdot \triangleleft \cdot \vec{y}'' |\!) \overset{\mathsf{Ref}}{\mapsto} \langle\!\langle \vec{x}'', \vec{y}'' \rangle\!\rangle.$$

Note that the first move above of Generator must appear inside the factor π'. Finally, the definition of annotated $\mathcal{R}$-decomposition (cf. again third item of Definition 3) requires this move to be preceded by the matching occurrence of the closing macro-tag $\langle/\hat{C}^{\mathsf{suffix}}\rangle$. □

Following the proof of Lemma 3 one can easily construct a transducer $\mathcal{Z}_3$ that receives as input an annotated $\mathcal{R}$-decomposition π and produces as output the serialization $\hat{\tau}$ of a partial $\mathcal{T}$-decomposition. We omit the tedious definition of the transducer $\mathcal{Z}_3$. Instead, we stress that, thanks to Lemma 3 and Lemma 4, the occurrence order of the non-empty strings of the form $\hat{C}^{\mathsf{prefix}}$ or $\hat{C}^{\mathsf{suffix}}$ is the same in the input π and in the corresponding output $\hat{\tau}$. Moreover, the transducer $\mathcal{Z}_3$ performs only small edits to transform the macro-tags of π to the macro-tags of $\hat{\tau}$. Finally, $\mathcal{Z}_3$ inserts, for each atomic operation on the stack of Repairer appearing in π, a number of macro-tags that is at most exponential in the size of the target automaton $\mathcal{T}$. Overall, this means that $\mathcal{Z}_3$ can be seen as a repair process that has aggregate cost at most linear in the number of macro-tags and moves of π, and at most exponential in the size of $\mathcal{T}$.

4.4 Transducer $\mathcal{Z}_4$: gluing the contexts

In this subsection we define the last step of the repair process, which completes an input partial $\mathcal{T}$-decomposition τ by adding contexts over Δ that connect pairs of reachable states of the target automaton $\mathcal{T}$. After completing the partial decomposition τ, one obtains a factorization τ' of some tree $t' \in \mathscr{L}(\mathcal{T})$. In particular, we will see that the projection of the serialization of τ' onto the target alphabet $\Delta \uplus \bar{\Delta}$ coincides with the serialization $\hat{t}'$ of t'. The transducer $\mathcal{Z}_4$ will output precisely this serialization $\hat{t}'$.

We start by defining complete decompositions for the target automaton. As before, we let $[\Delta]$ be the ranked alphabet consisting of unary labels $[q : C]$, binary labels $[q : a]$, and nullary labels $[q : \varnothing]$, where q is a state of $\mathcal{T}$, C is a context over Δ, and a is a letter from Δ. The following definition is very similar to that of $\mathcal{R}$-decomposition – only some constraints concerning strongly connected components are removed.

Definition 5 Let $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$. A *complete $\mathcal{T}$-decomposition* is a ranked tree τ' labeled over $[\Delta]$ such that:

- the root is labeled with a unary, binary, or nullary symbol of the form $[q_0 : \alpha]$, with q_0 initial state of $\mathcal{T}$;
- for every node labeled with a unary symbol $[q : C]$, the state $q' = \delta^\circ(q, C)$ is well defined; furthermore, the node has a unique child which is labeled with a unary, binary, or nullary symbol of the form $[q' : \alpha]$;

– for every node labeled with a binary symbol $[q : a]$, the pair of states $(q_1, q_2) = \gamma(q, a)$ is well defined; furthermore, the node has left and right children labeled respectively with symbols $[q_1 : \alpha_1]$ and $[q_2 : \alpha_2]$;
– every node labeled with a nullary symbol $[q : \varnothing]$ is a leaf and q is a final state.

We say that τ' is a $\mathcal{T}$-decomposition *of a tree or forest* t' if and only if $[\![\tau']\!] = t'$, where

$$[\![\tau']\!] \quad =^{\text{def}} \quad \begin{cases} \varnothing & \text{if } \tau' = [q : \varnothing], \\ C \circ [\![\tau'_1]\!] & \text{if } \tau' = [q : C]\,(\tau'_1), \\ a([\![\tau'_1]\!]) \cdot [\![\tau'_2]\!] & \text{if } \tau' = [q : a]\,(\tau'_1, \tau'_2). \end{cases}$$

Remark 3 It is easy to see that the tree $t' = [\![\tau']\!]$ represented by a complete $\mathcal{T}$-decomposition τ' is accepted by the target automaton $\mathcal{T}$. Moreover, the projection of the serialization $\hat{\tau}'$ onto $\Delta \uplus \bar{\Delta}$ coincides with the serialization $\hat{t}'$ of t'.

The following lemma shows how to turn a given partial $\mathcal{T}$-decomposition τ into a complete $\mathcal{T}$-decomposition τ' by inserting a linear number of nodes.

Lemma 5 *For every partial $\mathcal{T}$-decomposition τ, there is a complete $\mathcal{T}$-decomposition τ' such that the projection of $\hat{\tau}$ onto $\Delta \uplus \bar{\Delta}$ is a sub-sequence of the projection of $\hat{\tau}'$ onto $\Delta \uplus \bar{\Delta}$. Moreover, the number of nodes of τ' is linear in the number of nodes of τ.*

Proof. The idea is to insert suitable contexts between adjacent nodes in the partial decomposition τ in order to connect the various states.

More precisely, for each internal node $\vec{\imath}$ of τ labeled with a symbol $[q : C]$, we let $q_1 = \gamma^\circ(q, C)$ (this is well defined) and we let q'_1 be the state associated with the unique successor of $\vec{\imath}$. From the definition of partial $\mathcal{T}$-decomposition (cf. first item of Definition 4) we know that the two states q_1 and q'_1 belong to the same component of $\mathcal{T}$, and hence there is a context C_{q_1,q'_1} over Δ such that $\gamma^\circ(q_1, C_{q_1,q'_1}) = q'_1$. Accordingly, we insert a new node labeled with $[q_1 : C_{q_1,q'_1}]$ between the node $\vec{\imath}$ and its successor.

Similarly, for each internal node $\vec{\imath}$ of τ labeled with a symbol $[q : a]$, we let $(q_1, q_2) = \gamma(q, a)$ and we let q'_1 and q'_2 be the states associated with the left and right successors of $\vec{\imath}$, respectively. Using the fact that q_1, q'_1 (resp. q_2, q'_2) are in the same component, we obtain a context C_{q_1,q'_1} (resp. C_{q_2,q'_2}) such that $\gamma^\circ(q_1, C_{q_1,q'_1}) = q'_1$ (resp. $\gamma^\circ(q_2, C_{q_2,q'_2}) = q'_2$). We then insert a new node labeled with $[q_1 : C_{q_1,q'_1}]$ between the node $\vec{\imath}$ and its left successor, and a new node labeled with $[q_2 : C_{q_2,q'_2}]$ between the node $\vec{\imath}$ and its right successor.

The tree that results from the above insertions satisfies the second and third item of Definition 5. To obtain a complete $\mathcal{T}$-decomposition, it remains to insert a new root with associated initial state and new leaves with associated final states.

Let q'_0 be the state at the root of the partial decomposition τ. Since $\mathcal{T}$ is trimmed, q'_0 is reachable from the initial state q_0, that is, there exists a context C_{q_0,q'_0} such that $\gamma^\circ(q_0, C_{q_0,q'_0}) = q'_0$. We then insert a new root and label it with $[q_0 : C_{q_0,q'_0}]$. Finally, for each leaf $\vec{\imath}$ labeled with $[q : C]$, we let $q_1 = \gamma^\circ(q, C)$. From the fact that $\mathcal{T}$ is trimmed, we derive the existence of a tree or forest t_{q_1} that is accepted by $\mathcal{T}$ starting from state q_1. We can thus form the horizontal context $C_{q_1} = t_{q_1} \bullet$ in such a way that $f = \gamma^\circ(q_1, C_{q_1})$ is a final state of $\mathcal{T}$. Finally, we append under the node $\vec{\imath}$ a chain consisting a unary node labeled with $[q_1 : C_{q_1}]$ and a leaf labeled with $[f : \varnothing]$.

The tree τ' that is obtained from τ by performing all the above insertions is a complete $\mathcal{T}$-decomposition. Moreover, the transformation has clearly preserved all nodes in τ, as well as the ancestor relation on them. This means that the projection of $\hat{\tau}$ onto $\Delta \uplus \bar{\Delta}$ is a sub-sequence of the projection of $\hat{\tau}'$ onto $\Delta \uplus \bar{\Delta}$. □

We conclude the section by observing that the transformation from a partial $\mathcal{T}$-decomposition τ to a complete $\mathcal{T}$-decomposition τ' can be easily implemented at the level of the serializations by means of a transducer. Moreover, the projection of $\hat{\tau}'$ onto $\Delta \uplus \bar{\Delta}$ can be also implemented by a transducer and coincides with the serialization $\hat{t'}$ of a tree $t' \in \mathscr{L}(\mathcal{T})$.

The transducer $\mathcal{Z}_4$ is nothing but the concatenation of the transformations from $\hat{\tau}$ to $\hat{\tau}'$ and from $\hat{\tau}'$ to $\hat{t'}$. In particular, it can be seen as a repair process that transforms the serialization of a partial $\mathcal{T}$-decomposition into the serialization of a tree in the target language. The cost of this transformation is at most linear in the number of macro-tags in $\hat{\tau}$, and exponential in the size of $\mathcal{T}$.

4.5 Correctness and cost of the compound transducer

We can now combine the four transducers $\mathcal{Z}_1$, $\mathcal{Z}_2$, $\mathcal{Z}_3$, $\mathcal{Z}_4$ described in the previous subsections into a cascade composition $\mathcal{Z} = \mathcal{Z}_4 \circ \mathcal{Z}_3 \circ \mathcal{Z}_2 \circ \mathcal{Z}_1$. Formally, we can compute this composition as a pipeline process whose internal memory consists of the configurations of each transducer $\mathcal{Z}_i$ plus the intermediate outputs manipulated between them. Below, we argue that the compound transducer $\mathcal{Z}$ can be regarded as a tree edit transducer implementing a streaming repair strategy from the restriction language $\mathscr{L}(\mathcal{R})$ into the target language $\mathscr{L}(\mathcal{T})$ with bounded aggregate cost.

First of all, it is clear from the previous results that if $\hat{t}$ is the serialization of a tree $t \in \mathscr{L}(\mathcal{R})$, then $\mathcal{Z}(\hat{t})$ is the serialization of some tree $t' \in \mathscr{L}(\mathcal{T})$. Moreover, it is not difficult to see that each of the transducers $\mathcal{Z}_1$, $\mathcal{Z}_2$, $\mathcal{Z}_3$, $\mathcal{Z}_4$ preserves the matching relation on the tags $a, \bar{a}$ that are not erased from the input – in particular, we observe that, thanks to Lemma 4, the repositioning of the closing macro-tags $\langle/\hat{C}^{\mathsf{suffix}}\rangle$ that occurs during the third transduction does not affect the matching relation on the tags that appear in $\hat{C}^{\mathsf{suffix}}$, as this string is necessarily empty. We can thus conclude that $\mathcal{Z}$ is a *tree edit* transducer.

We now analyse the worst-case aggregate cost of the repair strategy implemented by $\mathcal{Z}$. We start by considering the first transducer $\mathcal{Z}_1$, which transforms a serialized tree $\hat{t}$ into a serialized $\mathcal{R}$-decomposition $\hat{\sigma}$. We have seen that its aggregate cost is at most exponential in the number of components of $\mathcal{R}$ – hence uniformly bounded with respect to all possible inputs $\hat{t}$ – and that the serialized $\mathcal{R}$-decomposition $\hat{\sigma}$ contains $2^{\mathcal{O}(|\mathsf{SCC}(\mathcal{R})|)}$ macro-tags. The second transducer $\mathcal{Z}_2$ transforms $\hat{\sigma}$ into an annotated $\mathcal{R}$-decomposition π, incurring an aggregate cost that is at most linear in the number of macro-tags of π – again, uniformly bounded for all possible inputs. Similar bounds hold for the remaining transducers $\mathcal{Z}_3$ and $\mathcal{Z}_4$. Overall, we have that the aggregate cost of the compound transducer $\mathcal{Z}$ is $2^{\mathcal{O}(|\mathsf{SCC}(\mathcal{R})|\cdot|\mathcal{T}|)}$, which is uniformly bounded for all possible inputs $\hat{t}$.

5 From repairs to simulation games

In this section we prove the only-if direction of Theorem 1, namely, we assume the existence of a tree edit transducer $\mathcal{Z}$ that implements a streaming repair strategy from $\mathscr{L}(\mathcal{R})$ to $\mathscr{L}(\mathcal{T})$, with uniformly bounded cost, and we derive from this the existence of a strategy for Repairer to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

The general idea is to derive from the sequence of moves taken by Generator a corresponding sequence of longer and longer prefixes $u_1, u_2, \ldots$ of the serialization of some tree $t \in \mathscr{L}(\mathcal{R})$; then one considers the repairs $v_1, v_2, \ldots$ produced by the transducer $\mathcal{Z}$ on the prefixes $u_1, u_2, \ldots$ and extracts from the corresponding partial runs in $\mathcal{T}$ the responses of Repairer to Generator moves. We will see below how we can guarantee the existence of valid responses of Repairer during the entire play. Moreover, we will see that each time the transducer $\mathcal{Z}$ parses a certain portion of the input without performing a costly repair, the corresponding move of Repairer is trivial, that is, of the form $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}, \vec{y}]\!]$. Finally, the fact that $\mathcal{Z}$ has uniformly bounded aggregate cost will imply that the constructed strategy of Repairer is winning.

We organize the proof into three subsections. In Subsection 5.1 we lay down our assumptions, notations, and a couple of technical lemmas. In Subsection 5.2 we show in detail how to extract from the transducer $\mathcal{Z}$ a strategy for Repairer to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. In doing so we will consider a simplified version of the simulation game, where Referee is not allowed to play. This means that the stacks that define the positions in the arena could be modified only by Generator or Repairer, and they do not contain any occurrence of the separator symbol $\triangleleft$. Finally, in Subsection 5.3 we argue that the constructed strategy for Repairer is safe, in the sense that it does not take advantage of the absence of separators and it correctly mimics the changes to the stacks that are performed by Referee.

5.1 Preliminary assumptions, notations, and technical lemmas

In order to construct a winning strategy for Repairer, we will look at the repairs provided by transducer $\mathcal{Z}$ on arbitrary long streams. In particular, our constructions will not depend on the function Ω of $\mathcal{Z}$ that describes the final output at the end of a stream. Thus, for simplicity, we assume that the final output $\Omega(z)$ is empty for every state z of $\mathcal{Z}$ (if this were not the case, we could simply consider a new transducer $\mathcal{Z}'$ that behaves exactly as $\mathcal{Z}$, but has $\Omega(z) = \varepsilon$ on all states z). This assumption allows us to conveniently denote by $\mathcal{Z}(u)$ the output produced by $\mathcal{Z}$ on a partial input u – in particular, note that $\mathcal{Z}(u)$ is a prefix of $\mathcal{Z}(u')$ whenever u is a prefix of u'.

The general idea of the proof is to consider the states reached[1] by $\mathcal{R}$ on generic prefixes u of serialized trees in $\mathcal{R}$. By looking at how the components of these states change when we prolong the prefixes, we will be able to derive appropriate sequences of moves of Generator in the simulation game. Similarly, by considering

[1] We recall from Section 3.2 that, given a top-down deterministic tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ and a prefix u of the serialization of some tree $t \in \mathscr{L}(\mathcal{A})$, there exists a unique state of the form $\delta^\circ(q_0, C)$ for any context C such that $\hat{C}^{\mathsf{prefix}} = u$; $\delta(q_0, C)$ is precisely called *the state reached by* $\mathcal{A}$ *on a prefix* u.

the components of the states reached by $\mathcal{T}$ on the repaired prefixes $\mathcal{Z}(u)$, we will construct the possible responses of Repairer to Generator's moves. For this, however, we will need appropriate pumping arguments on the possible prolongations of the prefixes u that do not induce a change of component in the automaton $\mathcal{R}$. Intuitively, for long enough prolongations of u that do not make $\mathcal{R}$ quit a certain component X, we will be able to show that the corresponding states reached by $\mathcal{T}$ stabilize within a single component Y such that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$ (see Lemma 7 for a formal statement of this property). Thanks to this property, we will be able to prove that the constructed strategy of Repairer induces plays that visit only valid positions of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

We begin by giving a generalized notion of context, which will be used later to construct a winning strategy for Repairer.

A *multi-context with h holes* is a tree or a forest C with labels over $\Sigma \uplus \{\bullet\}$ that contains exactly h occurrences of the hole symbol $\bullet$, and these occurrences are only at the leaves of C having no right siblings. An example of a multi-context with 3 holes is the forest $a\big(b(c,\bullet),b,\bullet\big)\,\bullet$. Note that forests and contexts over Σ are all special cases of multi-contexts, with 0 and 1 holes respectively.

We define the composition of two multi-contexts C and C' to be the multi-context $C \circ C'$ obtained from substituting the *first* occurrence of $\bullet$ in the preorder visit of C with C'. We will frequently write compositions of multi-contexts without using parentheses, assuming that the order of composition is from left to right. Given two multi-contexts C and C', we say that C' *extends* C if there exists a series of multi-contexts $C_1, \ldots, C_k$ such that $C' = C \circ C_1 \circ \ldots \circ C_k$. Note that the extension relation on multi-contexts is a partial order.

Similar to the serialization of a context, we denote by $\hat{C}^{\mathsf{prefix}}$ the prefix of the serialization of a multi-context C that ends immediately before the first occurrence of the symbol $\bullet$ (clearly, if there is no such an occurrence, then $\hat{C}^{\mathsf{prefix}} = \hat{C}$). It is easy to see that if C' extends C, then $\hat{C}^{\mathsf{prefix}}$ is a prefix of $\hat{C}'^{\mathsf{prefix}}$.

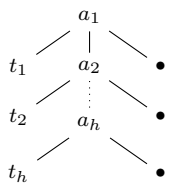
We will also need to reason on computations of automata on portions of trees, precisely, on *multi-contexts* and on *prefixes of serializations*. Given a top-down deterministic tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, we first extend the function δ° representing the behaviour of $\mathcal{A}$ from contexts to multi-contexts. Formally, we let δ° be the partial function that maps a state q and a multi-context C with h holes to the unique tuple of states $\delta^\circ(q, C) = (q_1, \ldots, q_h)$ whenever there exists a 'run' $\rho : \mathsf{nodes}^+(C) \to Q$ satisfying:

- $\rho(\vec{\imath}) = q$, where $\vec{\imath}$ is the leftmost root of ρ,
- $\rho(\vec{\imath}_j) = q_j$ for all $j = 1, \ldots, h$, where $\vec{\imath}_j$ is the node with the j-th occurrence of the hole symbol $\bullet$ following the preorder visit of C,
- $\delta\big(\rho(\vec{\imath}\cdot j), C(\vec{\imath}\cdot j)\big) = \big(\rho(\vec{\imath}\cdot j\cdot 1), \rho(\vec{\imath}\cdot(j+1))\big)$ for all nodes $\vec{\imath}\cdot j \in \mathsf{nodes}(C)$.

By a slight abuse of notation, we further extend the function δ° to prefixes of serialized trees, as follows. Given a prefix u of a serialized tree $\hat{t}$, we factorize u uniquely as $a_1 \cdot \hat{t}_1 \cdot a_2 \cdot \hat{t}_2 \cdot \ldots\ a_h \cdot \hat{t}_h$, where $a_1, a_2, \ldots, a_h$ are all the unmatched opening tags in u and $\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_h$ are the well-matched infixes between them, which represent, respectively, the forests $t_1, t_2, \ldots, t_h$. We then define $\delta^\circ(q, u) =$

$\delta^\circ(q, C_u)$, where C_u is the multi-context with h holes associated with u:

C_u =



Intuitively, $\delta^\circ(q, u)$ gives the stack of states associated with the h holes in a run of $\mathcal{A}$ on C_u that starts from q.

Finally, it will be convenient to use a short notation for the component of a state in an automaton. Precisely, given a state q of $\mathcal{A}$, we denote with $[q]_\mathcal{A}$ (or simply with $[q]$, when the automaton $\mathcal{A}$ is understood from the context) the strongly connected component of $\mathcal{A}$ that contains the state q. We then extend this notation from states to stack of states $q_1 \cdot \ldots \cdot q_n \in Q^+$, namely, we let $[q_1 \cdot \ldots \cdot q_n]_\mathcal{A} = [q_1]_\mathcal{A} \cdot \ldots \cdot [q_n]_\mathcal{A}$.

We now give a couple of technical lemmas that will be extensively used in the rest of the section. For this, we fix two top-down determinsitic tree automata $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$ and $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ recognizing the restriction and target languages, respectively.

Given a state p of $\mathcal{R}$ and a context C, we say that C is *cyclic* on state p if we have $\delta^\circ(p, C) = p$ – in particular, C is realizable within the component of p.

The first lemma shows that one can associate with each component X of $\mathcal{R}$ a suitable context D_X that is cyclic on some state $p \in X$ – hence D_X can be 'pumped' inside the language $\mathscr{L}(\mathcal{R} \mid X)$ – and such that, for every component Y of $\mathcal{T}$, $D_X \in \mathscr{L}(\mathcal{T} \mid Y)$ if and only if $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$. We call any such context D_X a *fingerprint context* for the component X. We remark that this notion of fingerprint context was originally introduced in [18].

Lemma 6 *Every component X of $\mathcal{R}$ has a fingerprint context, namely, for each $X \in \mathsf{SCC}(\mathcal{R})$, there exists a state $p \in X$ and a context $D_X \in \mathscr{L}(\mathcal{R} \mid X)$ such that $\delta^\circ(p, D_X) = p$ and, for every $Y \in \mathsf{SCC}(\mathcal{T})$,*

$$\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y) \qquad \text{if and only if} \qquad D_X \in \mathscr{L}(\mathcal{T} \mid Y).$$

Moreover, if X is non-horizontal, then there exists such a context D_X that is vertical, that is, where the hole is not a root.

Proof. Let X be a component of $\mathcal{R}$ and let $Y_1, \ldots, Y_m$ be all the components of $\mathcal{T}$. We prove by induction on $i = 0, \ldots, m$ that there is a cyclic context $D_i \in \mathscr{L}(\mathcal{R} \mid X)$ such that:

$$\forall 1 \leq j \leq i. \quad \mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_j) \quad \text{if and only if} \quad D_i \in \mathscr{L}(\mathcal{T} \mid Y_j) \qquad (\star)$$

The first claim of the lemma will follow from $(\star)$ when we let $D_X = D_m$ (to satisfy the second claim when X is non-horizontal, it will be sufficient to compose D_m with a cyclic vertical context).

The base case of the induction is $i = 0$, where we simply let D_0 be the trivial context $\bullet$.

For the inductive step, let $0 \leq i < m$ and suppose that there exists a context D_i that satisfies $(\star)$. We construct the next context D_{i+1} that satisfies $(\star)$. To do so, we distinguish two cases, depending on whether $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_{i+1})$ or not. If $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_{i+1})$, then we define $D_{i+1} = D_i$. In this way we easily see that the context D_{i+1} satisfies $(\star)$. Otherwise, if $\mathscr{L}(\mathcal{R} \mid X) \not\subseteq \mathscr{L}(\mathcal{T} \mid Y_{i+1})$, we choose a context $C \in \mathscr{L}(\mathcal{R} \mid X) \setminus \mathscr{L}(\mathcal{T} \mid Y_{i+1})$. Since D_i is cyclic and $C \in \mathscr{L}(\mathcal{R} \mid X)$, there exist some states $p, p', p'' \in X$ such that $\delta(p, D_i) = p$ and $\delta(p', C) = p''$. Moreover, since all states p, p', p'' belong to the same component, there exist some contexts $C', C'' \in \mathscr{L}(\mathcal{R} \mid X)$ such that $\delta(p, C') = p'$ and $\delta(p'', C'') = p$. We then define $D_{i+1} = D_i \circ C' \circ C \circ C''$ and observe that D_{i+1} is a cyclic context:

$$\delta^\circ(p, D_{i+1}) \;=\; \delta^\circ(p, C' \circ C \circ C'') \;=\; \delta^\circ(p', C \circ C'') \;=\; \delta^\circ(p'', C'') \;=\; p.$$

In particular $D_{i+1} \in \mathscr{L}(\mathcal{R} \mid X)$. It is also easy to see that D_{i+1} does not belong to $\mathscr{L}(\mathcal{T} \mid Y_{i+1})$. Indeed, if $D_{i+1} \in \mathscr{L}(\mathcal{T} \mid Y_{i+1})$, then there would exist states $q, q' \in Y_{i+1}$ such that $\delta(q, C) = q'$; this however would contradict the fact that $C \notin \mathscr{L}(\mathcal{T} \mid Y_{i+1})$. Finally, we know from the inductive hypothesis that, for every $1 \leq j \leq i$, if $D_{i+1} \in \mathscr{L}(\mathcal{T} \mid Y_j)$, then $D_i \in \mathscr{L}(\mathcal{T} \mid Y_j)$ and $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_j)$ follow. The converse implication holds in a similar way. We conclude that for every $1 \leq j \leq i+1$, $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_j)$ if and only if $D_{i+1} \in \mathscr{L}(\mathcal{T} \mid Y_j)$. This proves the inductive step for $(\star)$.

It only remains to observe that, if X is a non-horizontal component, then for every $p \in X$, there exists a vertical context C''' such that $\delta^\circ(p, C''') = p$. In this case we can redefine the fingerprint context as $D_X = D_m \circ C'''$, in such a way that the property $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$ if and only if $D_X \in \mathscr{L}(\mathcal{T} \mid Y)$ still holds for all $Y \in \mathsf{SCC}(\mathcal{T})$. □

The lemma below roughly states that, if one repeats a cyclic context D a sufficient number of times, then the states of the target automaton that are reached on the corresponding outputs produced by transducer $\mathcal{Z}$ will eventually stabilize within a single component Y and the language of contexts $\mathscr{L}(\mathcal{T} \mid Y)$ will necessarily contain the cyclic context D. It is worth noticing that a similar lemma appeared in [6] showing that streaming repair strategies of strings 'stabilize' inside components of finite automata. For the following statement, we recall that δ (resp. γ) denotes the transition function of $\mathcal{R}$ (resp. $\mathcal{T}$), p_0 (resp. q_0) denotes the initial state of $\mathcal{R}$ (resp. $\mathcal{T}$), and $\mathcal{Z}$ is a transducer with uniformly bounded aggregate cost and empty final output function (in particular, this implies that $\mathcal{Z}(u)$ is a prefix of $\mathcal{Z}(u')$ whenever u is a prefix of u').

Lemma 7 *Let C be a multi-context with any number $h \geq 1$ of holes such that $\delta^\circ(p_0, C) = (p_1, \ldots, p_h)$ (in particular, this means that $\hat{C}^{\mathsf{prefix}}$ is a prefix of the serialization of some tree in the restriction language). Moreover, let D be a cyclic context such that $\delta^\circ(p_1, D) = p_1$. There exist a number $n_0 \in \mathbb{N}$ and a component Y of $\mathcal{T}$ such that, for all natural numbers m and n, with $0 \leq m \leq n$:*

1. $\mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+n}}^{\mathsf{prefix}}\big) \;=\; \mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0}}^{\mathsf{prefix}}\big) \cdot \widehat{D^{n}}^{\mathsf{prefix}}$,
2. $\mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+n}}^{\mathsf{prefix}} \cdot \hat{t} \cdot \widehat{D^{n_0+m}}^{\mathsf{suffix}}\big) \;=\; \mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+n}}^{\mathsf{prefix}} \cdot \hat{t} \cdot \widehat{D^{n_0}}^{\mathsf{suffix}}\big) \cdot \widehat{D^{m}}^{\mathsf{suffix}}$
   for all trees or forests t accepted by $\mathcal{R}$ starting from state p_1,
3. $\mathsf{top}\big(\gamma^\circ(q_0, v_{n_0+n})\big) \in Y$ *and* $D \in \mathscr{L}(\mathcal{T} \mid Y)$, *where* $v_{n_0+n} = \mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+n}}^{\mathsf{prefix}}\big)$.

Proof. Let $C, p_1, \ldots, p_h, D$ be as in the statement of the lemma. We define $u_n = \hat{C}^{\mathsf{prefix}} \cdot \widehat{D^n}^{\mathsf{prefix}}$, for all $n \in \mathbb{N}$, and we think of the words $u_0, u_1, \ldots$ as a series of prefixes provided as input to the transducer $\mathcal{Z}$. From the fact that $\mathcal{Z}$ is a tree edit transducer with aggregate cost uniformly bounded by a constant c_{max}, we know that the value $\mathsf{cost}(u_n, \mathcal{Z})$ stabilizes for a sufficiently large n, that is, there is a constant n_0 such that, for all $n \in \mathbb{N}$, $\mathsf{cost}(u_{n_0+n}, \mathcal{Z}) = \mathsf{cost}(u_{n_0}, \mathcal{Z}) \leq c_{\mathsf{max}}$. This immediately implies the first property stated in the lemma, that is, for all $n \in \mathbb{N}$:

$$\mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+n}}^{\mathsf{prefix}}\big) \;=\; \mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0}}^{\mathsf{prefix}}\big) \cdot \widehat{D^n}^{\mathsf{prefix}} \tag{1.}$$

Similarly, we recall from the hypothesis of the lemma that $\delta^\circ(p_0, C) = (p_1, \ldots, p_h)$ and $\delta^\circ(p_1, D) = p_1$. Since $u_n = \hat{C}^{\mathsf{prefix}} \cdot \widehat{D^n}^{\mathsf{prefix}}$, this implies that $\delta^\circ(p_0, u_n) = (p_1, \ldots, p_h)$. We fix an arbitrary tree or forest t that is accepted by $\mathcal{R}$ starting from state p_1, and we consider the series of inputs for $\mathcal{Z}$ of the form $u'_{n,m} = \hat{C}^{\mathsf{prefix}} \cdot \widehat{D^n}^{\mathsf{prefix}} \cdot \hat{t} \cdot \widehat{D^m}^{\mathsf{suffix}}$, with $m \leq n$ (note that $\delta^\circ(p_0, u'_{n,m}) = (p_1, \ldots, p_h)$ holds as well). If we let m increase, we derive from the fact that $\mathcal{Z}$ has aggregate cost at most c_{max} the existence of another constant n_1 such that $\mathsf{cost}(u'_{n,n_1+m}, \mathcal{Z}) = \mathsf{cost}(u'_{n,n_1}, \mathcal{Z}) \leq c_{\mathsf{max}}$ for all $m \leq n$. Without loss of generality, we can further assume that n_0 was chosen large enough so as to dominate n_1, that is, $n_0 \geq n_1$. In this way we prove that the second property holds for all $m \leq n \in \mathbb{N}$:

$$\mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+n}}^{\mathsf{prefix}} \cdot \hat{t} \cdot \widehat{D^{n_0+m}}^{\mathsf{suffix}}\big) = \mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+n}}^{\mathsf{prefix}} \cdot \hat{t} \cdot \widehat{D^{n_0}}^{\mathsf{suffix}}\big) \cdot \widehat{D^m}^{\mathsf{suffix}} \tag{2.}$$

Next, we consider the sequence of components of $\mathcal{T}$ of the form $Y_n = [q_n]$, with $q_n = \mathsf{top}\big(\gamma^\circ(q_0, v_n)\big)$ and $v_n = \mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^n}^{\mathsf{prefix}}\big)$, for all $n > 0$. By construction, we have that the sequence $Y_1, Y_2, \ldots$ forms a chain in the directed acyclic graph of the components of $\mathcal{T}$, where the accessibility relation is lifted from states to components in the natural way. Since there are only finitely many components, we know that the sequence $Y_1, Y_2, \ldots$ is ultimately constant, that is, we have $Y_{n_2} = Y_{n_2+1} = \ldots = Y$ for a sufficiently large constant n_2 and a well defined component Y of $\mathcal{T}$. Again, we assume without loss of generality that $n_0 \geq n_2$ and we prove in this way that:

$$\mathsf{top}\big(\gamma^\circ(q_0, v_{n_0+n})\big) \in Y \qquad \text{for all } n \in \mathbb{N}.$$

It remains to prove that the context D is realizable within the component Y. For this, we consider some outputs produced by $\mathcal{Z}$, precisely, the words of the form $v_{n_0+n} = \mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+n}}^{\mathsf{prefix}}\big)$, where $n = 0$ or $n = 1$, and the words of the form $w_{n_0+n} = \mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+1}}^{\mathsf{prefix}} \cdot \hat{t} \cdot \widehat{D^{n_0+n}}^{\mathsf{suffix}}\big)$, where $n = 0$ or $n = 1$ and where t is a generic tree accepted by $\mathcal{R}$ starting from state p_1 (note that $p_1 = \mathsf{top}\big(\delta^\circ(p_0, v_{n_0+n})\big)$). We recall from the properties (*1.*) and (*2.*) above that $v_{n_0+1} = v_{n_0} \cdot \hat{D}^{\mathsf{prefix}}$ and $w_{n_0+1} = w_{n_0} \cdot \hat{D}^{\mathsf{suffix}}$. We then consider the corresponding stacks of states reached by $\mathcal{T}$ on the above words, that is, $\vec{q}_{n_0+n} = \gamma^\circ(q_0, v_{n_0+n})$ and $\vec{r}_{n_0+n} = \gamma^\circ(q_0, w_{n_0+n})$, for both $n = 0$ and $n = 1$. For the sake of brevity, we let $\vec{q} = \mathsf{tail}(\vec{q}_{n_0})$. From the property (*3.*) above, we know that the states $q = \mathsf{top}(\vec{q}_{n_0})$

and $q' = \mathsf{top}(\vec{q}_{n_0+1})$ at the top of the two stacks $\vec{q}_{n_0}$ and $\vec{q}_{n_0+1}$ belong to the component Y. Moreover, it follows from the various definitions that:

$$\begin{aligned}
\vec{q}_{n_0} &= \mathsf{top}(\vec{q}_{n_0}) \cdot \mathsf{tail}(\vec{q}_{n_0}) = q \cdot \vec{q} \\
\vec{q}_{n_0+1} &= \gamma^\circ(q_0, v_{n_0+1}) = \gamma^\circ\big(q_0, \mathcal{Z}(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+1}}^{\mathsf{prefix}})\big) \\
&= \gamma^\circ\big(q_0, \mathcal{Z}(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0}}^{\mathsf{prefix}}) \cdot \hat{D}^{\mathsf{prefix}}\big) = \gamma^\circ(\vec{q}_{n_0}, \hat{D}^{\mathsf{prefix}}) \\
&= q' \cdot \mathsf{tail}\big(\gamma^\circ(\vec{q}_{n_0}, \hat{D}^{\mathsf{prefix}})\big) = q' \cdot \mathsf{tail}\big(\gamma^\circ(q, \hat{D}^{\mathsf{prefix}})\big) \cdot \vec{q} \\
\vec{r}_{n_0+1} &= \gamma^\circ(q_0, w_{n_0+1}) = \gamma^\circ\big(q_0, \mathcal{Z}(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+1}}^{\mathsf{prefix}} \cdot \hat{t} \cdot \widehat{D^{n_0+1}}^{\mathsf{suffix}})\big) \\
&= \mathsf{tail}\big(\gamma^\circ\big(q_0, \mathcal{Z}(\hat{C}^{\mathsf{prefix}})\big)\big) = \mathsf{tail}(\vec{q}_{n_0}) = \vec{q} \\
\vec{r}_{n_0} &= \gamma^\circ(q_0, w_{n_0}) = \gamma^\circ\big(q_0, \mathcal{Z}(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0+1}}^{\mathsf{prefix}} \cdot \hat{t} \cdot \widehat{D^{n_0}}^{\mathsf{suffix}})\big) \\
&= \mathsf{tail}\big(\gamma^\circ\big(q_0, \mathcal{Z}(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D^{n_0}}^{\mathsf{prefix}})\big)\big) = \mathsf{tail}\big(\gamma^\circ(\vec{q}_{n_0}, \hat{D}^{\mathsf{prefix}})\big) \\
&= \mathsf{tail}\big(\gamma^\circ(q, \hat{D}^{\mathsf{prefix}})\big) \cdot \vec{q}\,.
\end{aligned}$$

This proves that $\gamma^\circ(q, D) = q'$ and hence

$$\mathsf{top}\big(\gamma^\circ(q_0, v_{n_0+n})\big) \in Y \quad \text{and} \quad D \in \mathscr{L}(\mathcal{T} \mid Y) \qquad \text{for all } n \in \mathbb{N}. \tag{3.}$$

□

5.2 Construction of a strategy for Repairer

We are now ready to enter the details of the proof. Recall that $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$ and $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ are two top-down deterministic tree automata recognizing the restriction and target languages, and that $\mathcal{Z}$ is a tree edit transducer implementing a streaming repair strategy from $\mathscr{L}(\mathcal{R})$ to $\mathscr{L}(\mathcal{T})$ with aggregate costs $\mathsf{cost}(\hat{t}, \mathcal{Z})$ uniformly bounded by a constant c_{max}, for all trees $t \in \mathscr{L}(\mathcal{R})$.

The strategy of Repairer will be a partial function that maps plays of the form

$$\langle\!\langle\, \vec{x}_0\,, \vec{y}_0 \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_0\,, \vec{y}_1 \,]\!] \overset{\mathsf{Gen}}{\mapsto} \cdots \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_{n-1}\,, \vec{y}_n \,]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle\, \vec{x}_n\,, \vec{y}_n \,\rangle\!\rangle$$

to the next move $\langle\!\langle\, \vec{x}_n\,, \vec{y}_n \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_n\,, \vec{y}_{n+1} \,]\!]$ that has to be chosen by Repairer in order to win the game. The definitions will exploit an induction on the length n of the play, starting from $n = 0$. At the same time, we will associate with the above plays some corresponding sequences of multi-contexts $E_0 = \bullet$, E_1, …, E_{n+1} over Σ that satisfy the following properties, for all $1 \le i \le n+1$:

1. E_i is an extension of E_{i-1},
2. E_i has the same number of holes as the height h_{i-1} of the stack $\vec{x}_{i-1}$,
3. the tuple of states $\delta^\circ(p_0, E_i) = (p_1, \ldots, p_{h_{i-1}})$ is well defined and the lifting to the sequence of components $[p_1 \cdot \ldots \cdot p_{h_{i-1}}]_{\mathcal{R}}$ coincides with the stack $\vec{x}_{i-1}$,
4. if $i \le n$ and $v_i = \mathcal{Z}(\hat{E}_i^{\mathsf{prefix}})$ is the output produced by $\mathcal{Z}$ on input $\hat{E}_i^{\mathsf{prefix}}$, then $\gamma^\circ(q_0, v_i)$ is well defined and $[\gamma^\circ(q_0, v_i)]_{\mathcal{T}}$ coincides with the stack $\vec{y}_i$.

For the base case $n = 0$, we know that $\vec{x}_0$ is the singleton stack that consists of the component $X_0 = [p_0]_{\mathcal{R}}$ of the initial state of $\mathcal{R}$. We apply Lemma 6 to obtain a fingerprint context $D_{X_0} \in \mathscr{L}(\mathcal{R} \mid X_0)$ and a state $p \in X_0$ such that (i) $\delta^\circ(p, D_{X_0}) = p$ and (ii) for every $Y \in \mathsf{SCC}(\mathcal{T})$, $\mathscr{L}(\mathcal{R} \mid X_0) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$ if and only

if $D_{X_0} \in \mathscr{L}(\mathcal{T} \mid Y)$. Since p and p_0 are in the same component X_0, we know that there is a context C that connects the two states, namely, $\delta^\circ(p_0, C) = p$. We can apply Lemma 7 to C, viewed as a multi-context with 1 hole, and D_{X_0}, viewed as cyclic context over the state $p = \delta^\circ(p_0, C)$; we obtain in this way a constant n_0 and a component Y of $\mathcal{T}$ such that, for all $n \in \mathbb{N}$:

- $\mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D_{X_0}^{n_0+n}}^{\mathsf{prefix}}\big) = \mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D_{X_0}^{n_0}}^{\mathsf{prefix}}\big) \cdot \widehat{D_{X_0}^{n}}^{\mathsf{prefix}}$,
- $\mathsf{top}\big(\gamma^\circ(q_0, v_{n_0+n})\big) \in Y$, where $v_{n_0+n} = \mathcal{Z}\big(\hat{C}^{\mathsf{prefix}} \cdot \widehat{D_{X_0}^{n_0+n}}^{\mathsf{prefix}}\big)$,
- $D_{X_0} \in \mathscr{L}(\mathcal{T} \mid Y)$.

We can now define the multi-context E_1 and the first move of Repairer:

$$E_1 \;=^{\mathrm{def}}\; E_0 \circ C \circ D_{X_0}^{n_0+1} \qquad \text{and} \qquad \langle\!\langle\, \vec{x}_0 \,,\, \vec{y}_0 \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_0 \,,\, \vec{y}_1 \,]\!]$$

$$\text{where} \quad \begin{cases} \vec{y}_0 &= [q_0]_{\mathcal{T}} \\ \vec{y}_1 &= [\gamma^\circ(q_0, v_1)]_{\mathcal{T}} \\ v_1 &= \mathcal{Z}(\hat{E}_1^{\mathsf{prefix}}). \end{cases}$$

By construction we have that the (multi-)context E_1 satisfies all the desired properties, that is:

1. E_1 is an extension of $E_0 = \bullet$,
2. E_1 has 1 hole, exactly as the height of the stack $\vec{x}_0 = X_0$,
3. the tuple $(p) = \delta^\circ(p_0, E_1)$ is well defined and, since $C \circ D_{X_0} \in \mathscr{L}(\mathcal{R}|_{p_0} X_0)$, we have $[p]_{\mathcal{R}} = [p_0]_{\mathcal{R}} = \vec{x}_0$,
4. $\vec{y}_1 = [\gamma^\circ(q_0, v_1)]_{\mathcal{T}}$, where $v_1 = \mathcal{Z}(\hat{E}_1^{\mathsf{prefix}})$.

It only remains to show that $\langle\!\langle\, \vec{x}_0 \,,\, \vec{y}_0 \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_0 \,,\, \vec{y}_1 \,]\!]$ is indeed a valid move of Repairer inside the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Clearly, the definition of this move respects the prefix-rewriting relation $\overset{\mathcal{T}}{\mapsto}{}^*$ associated with the target automaton $\mathcal{T}$. Moreover, the containment $\mathscr{L}(\mathcal{R} \mid X_0) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}_1))$ holds thanks to the following facts: (i) $\mathsf{top}(\vec{y}_1) = \big[\mathsf{top}\big(\gamma^\circ(q_0, v_1)\big)\big]_{\mathcal{T}} = Y$, (ii) $D_{X_0} \in \mathscr{L}(\mathcal{T} \mid Y)$, and (iii) D_{X_0} is a fingerprint context for X_0.

We now focus on the inductive argument of the construction. We suppose that $n > 0$ and that Generator has just moved from a position $[\![\, \vec{x}_{n-1} \,,\, \vec{y}_n \,]\!]$ to a position $\langle\!\langle\, \vec{x}_n \,,\, \vec{y}_n \,\rangle\!\rangle$ of the arena. Moreover, we assume that a corresponding sequence of contexts $E_1, \ldots, E_n$ has been already defined so as to satisfy the desired properties. In particular, we have $\vec{x}_{n-1} = \big[\delta^\circ(p_0, E_n)\big]_{\mathcal{R}}$ and $\vec{y}_n = \big[\gamma^\circ(q_0, \mathcal{Z}(\hat{E}_n^{\mathsf{prefix}}))\big]_{\mathcal{T}}$. To define the multi-context E_{n+1} we distinguish two cases depending on whether the last move $[\![\, \vec{x}_{n-1} \,,\, \vec{y}_n \,]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle\, \vec{x}_n \,,\, \vec{y}_n \,\rangle\!\rangle$ of Generator is a push-and-swap operation or a pop operation:

1. Suppose that the last move of Generator is a push-and-swap operation that replaces the stack $\vec{x}_{n-1} = X \cdot \vec{x}$ with the stack $\vec{x}_n = X_1\ X_2 \cdot \vec{x}$, where $X = [p]$, $X_1 = [p_1]$, $X_2 = [p_2]$ and $\delta(p, a) = (p_1, p_2)$. From the inductive hypothesis we know that $\vec{x}_{n-1}$ coincides with the lifting $\big[\delta^\circ(p_0, E_n)\big]_{\mathcal{R}}$, and hence the state $p' = \mathsf{top}\big(\delta^\circ(p_0, E_n)\big)$ belongs to X, exactly as p. We can thus find a context C such that $\delta^\circ(p', C) = p$. Moreover, Lemma 6 gives us a fingerprint context D_{X_1}

for the component X_1 and another state $p_1' \in X_1$ such that $\delta(p_1', D_{X_1}) = p_1'$. Again, since $p_1, p_1' \in X_1$, we can find a third context C' such that $\delta^\circ(p_1, C') = p_1'$. Next, we consider the composition of the multi-context E_n with the multi-context $C \circ a(C', \bullet)$ (note that the latter multi-context has two holes). From the previous definitions we obtain:

$$\begin{aligned}\delta^\circ\big(p_0,\ E_n \circ C \circ a(C', \bullet)\big) &= \delta^\circ\big(p',\ C \circ a(C', \bullet)\big) \cdot \mathsf{tail}\big(\delta^\circ(p_0, E_n)\big)\\ &= \delta^\circ\big(p,\ a(C', \bullet)\big) \cdot \mathsf{tail}\big(\delta^\circ(p_0, E_n)\big)\\ &= \delta^\circ\big(p_1,\ C'\big) \cdot p_2 \cdot \mathsf{tail}\big(\delta^\circ(p_0, E_n)\big)\\ &= p_1' \cdot p_2 \cdot \mathsf{tail}\big(\delta^\circ(p_0, E_n)\big).\end{aligned}$$

We also let n_0 be the number obtained from applying Lemma 7 to the multi-context $E_n \circ C \circ a(C', \bullet)$ and to the cyclic context D_{X_1}.
Finally, we can define the multi-context E_{n+1} for the inductive step:

$$E_{n+1} \quad =^{\text{def}} \quad E_n \circ C \circ a(C', \bullet) \circ D_{X_1}^{n_0+1}.$$

Clearly, the multi-context E_{n+1} is an extension of E_n and, due to the insertion of $a(C', \bullet)$, it has one hole more than the multi-context E_n. In particular, the multi-context E_{n+1} has exactly as many holes as the height of the stack $\vec{x}_n$. Moreover, using the previous equalities, we derive:

$$\begin{aligned}\delta^\circ(p_0, E_{n+1}) &= \delta^\circ\big(p_0,\ E_n \circ C \circ a(C', \bullet) \circ D_{X_1}^{n_0+1}\big)\\ &= \delta^\circ\big(p_1',\ D_{X_1}^{n_0+1}\big) \cdot p_2 \cdot \mathsf{tail}\big(\delta^\circ(p_0, E_n)\big)\\ &= p_1' \cdot p_2 \cdot \mathsf{tail}\big(\delta^\circ(p_0, E_n)\big).\end{aligned}$$

From $\vec{x}_{n-1} = \big[\delta^\circ(p_0, E_n)\big]_{\mathcal{R}}$, $\vec{x}_n = X_1\ X_2 \cdot \mathsf{tail}(\vec{x}_{n-1})$, $p_1' \in X_1$, and $p_2 \in X_2$, it then follows that $\big[\delta^\circ(p_0, E_{n+1})\big]_{\mathcal{R}} = \vec{x}_n$.
It now remains to define the corresponding move of Repairer. As usual, this move is extracted from the repair computed by the transducer $\mathcal{Z}$ on input $\hat{E}_{n+1}^{\mathsf{prefix}}$. More precisely, we define the next move of Repairer to be

$$\langle\!\langle\, \vec{x}_n\,, \vec{y}_n \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_n\,, \vec{y}_{n+1} \,]\!]$$

where $\vec{y}_{n+1} = \big[\gamma^\circ(q_0, v_{n+1})\big]_{\mathcal{T}}$ and $v_{n+1} = \mathcal{Z}(\hat{E}_{n+1}^{\mathsf{prefix}})$. The fact that the above move satisfies the containment $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x}_n)) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}_{n+1}))$ follows from the definition of n_0 (Lemma 7) and, in particular, from the fact that D_{X_1} is a fingerprint context for $X_1 = \mathsf{top}(\vec{x}_n)$ and is realizable within the component $Y = \mathsf{top}(\vec{y}_{n+1})$ of $\mathcal{T}$ (Lemma 6).

2. We now consider the case where the last move of Generator is a pop operation that removes the top component from the stack $\vec{x}_{n-1}$, thus reaching the stack $\vec{x}_n = \mathsf{tail}(\vec{x}_{n-1})$. We know from the inductive hypothesis that $\delta^\circ(p_0, E_n) = (p_1, p_2, \ldots, p_{h_{n-1}})$ is well defined and its lifting $[p_1\ p_2 \cdot \ldots \cdot p_{h_{n-1}}]_{\mathcal{R}}$ coincides with $\vec{x}_{n-1}$. Since $\mathcal{R}$ is trimmed, there exists a tree t that is accepted by $\mathcal{R}$ starting from state p_1. In particular, we have $\delta^\circ(p_0, E_n \circ t) = (p_2, \ldots, p_{h_{n-1}})$ and $[p_2, \ldots, p_{h_{n-1}}]_{\mathcal{R}} = \mathsf{tail}(\vec{x}_{n-1}) = \vec{x}_n$.
Next, we denote by X the component at the top of the stack $\vec{x}_n$ and we note that $p_2 \in X$. Using Lemma 6 we find a fingerprint context D_X for X and we let p_2' be a state in X such that $\delta(p_2', D_X) = p_2'$. Moreover, we let C be any context

that connects p_2 to p_2', namely, such that $\delta(p_2, C) = p_2'$. We then consider the multi-context $E_n \circ t \circ C$ and we verify that

$$\delta^\circ\big(p_0,\ E_n \circ t \circ C\big)\ =\ p_2'\ p_3\ \ldots\ p_{h_{n-1}}.$$

Moreover, we let n_0 be the constant obtained from applying Lemma 7 to the multi-context $E_n \circ t \circ C$ and to the cyclic context D_X.
We can finally define the multi-context and the move of Repairer for the inductive step as follows:

$$E_{n+1}\ =^{\text{def}}\ E_n \circ t \circ C \circ D_X^{n_0} \qquad \text{and} \qquad \langle\!\langle\, \vec{x}_n\,, \vec{y}_n \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_n\,, \vec{y}_{n+1} \,]\!]$$

$$\text{where} \quad \begin{cases} \vec{y}_{n+1} &= [\gamma^\circ(q_0, v_{n+1})]_{\mathcal{T}} \\ v_{n+1} &= \mathcal{Z}(\hat{E}_{n+1}^{\mathsf{prefix}}). \end{cases}$$

Using exactly the same arguments as in the previous case, one can verify that the above definitions satisfy the inductive hypothesis, in particular, the fact that $\big[\delta^\circ(p_0, E_{n+1})\big]_{\mathcal{R}} = \vec{x}_n$ and $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x}_n)) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}_{n+1}))$.

Summing up, from the existence of a transducer $\mathcal{Z}$ that repairs $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with bounded aggregate-cost, we derived the existence of a strategy for Repairer to win the simulation game over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Note that, for the moment, we overlooked the effect of the moves of the third player Referee.

In the following section we will give detailed arguments towards proving that Repairer's strategy is correct also with respect to presence of separator symbols. For this, it is convenient to summarize below the key properties that are achieved by Repairer's strategy (one can easily verify these properties in the above constructions).

Claim 3. For every position $\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle$ that is reached during a play induced by the defined strategy of Repairer, there is a multi-context E, an extension E' of E, and a fingerprint context D_X for X, where $X = \mathsf{top}(\vec{x})$, such that:

- $\big[\delta^\circ(p_0, E)\big]_{\mathcal{R}} = \vec{x}$
  (recall that $\delta^\circ(p_0, E)$ is a sequence of states of the same length as the number of holes in E and $\big[\delta^\circ(p_0, E)\big]_{\mathcal{R}}$ is the lifting of this sequence to the components of $\mathcal{R}$),
- the states $p = \mathsf{top}(\delta^\circ(p_0, E))$ and $p' = \mathsf{top}(\delta^\circ(p_0, E'))$ belong to the same component $X = \mathsf{top}(\vec{x})$,
- D_X is cyclic on the state $p' = \mathsf{top}(\delta^\circ(p_0, E'))$ and it is vertical whenever X is non-horizontal.

In addition, if we let $u_n = \hat{E}'^{\mathsf{prefix}} \cdot \big(\hat{D}_X^{\mathsf{prefix}}\big)^n$ and $\vec{y}_n = \big[\gamma^\circ(q_0, \mathcal{Z}(u_n))\big]_{\mathcal{T}}$, for all $n \in \mathbb{N}$, then the move induced by Repairer's strategy is of the form

$$\langle\!\langle\, \vec{x}\,, \vec{y} \,\rangle\!\rangle \ \overset{\mathsf{Rep}}{\mapsto}\ [\![\, \vec{x}\,, \vec{y}_{n_0+1} \,]\!]$$

where n_0 is a sufficiently large number such that, for some component Y of $\mathcal{T}$ and for all $n \in \mathbb{N}$, $\mathsf{top}(\vec{y}_{n_0+n}) = Y$ and $D_X \in \mathscr{L}(\mathcal{T} \mid Y)$.

5.3 Correctness of Repairer strategy

In the previous part we constructed from a given transducer $\mathcal{T}$ a corresponding strategy for Repairer. For the sake of simplicity, we did so without considering the presence of non-horizontal components in $\mathcal{R}$ and, specifically, the role of the separator symbols introduced by Referee. We dedicate the last part of this section to explain why the strategy of Repairer can be considered correct even in the presence of non-horizontal components. The argument is not straightforward, since it depends on the interaction between the moves of Generator, Repairer, and Referee and on the interleaving of separators and components induced by these moves. In fact, what we will prove below is the correctness of Repairer strategy with respect to a modified, but equivalent, version of the game. Intuitively, this new game is obtained by relaxing the rules for inserting separator symbols in the stack controlled by Repairer. Since we would like to work with reachability games between two players (recall that Referee had no choice in the original game and was introduced only to ease the presentation), we will assume that the additional degree of freedom can be exploited by Repairer. In particular, for Repairer it will be at least as easy to win the modified game as to win the original game. Conversely, we will show that from a strategy of Repairer that is winning in the modified game, one can derive a similar winning strategy of Repairer in the original game, which is of course correct with respect to the more stringent rules.

We begin by presenting the modified game and by proving that it has the same winner as the original game. For this we consider sequences of push-and-swap and pop operations that satisfy the basic prefix-rewriting system $\overset{\mathcal{T}}{\mapsto}$ and that can be associated with a single move of Repairer. Let S be any such sequence, say $\vec{y}_1 \overset{\mathcal{T}}{\mapsto} \vec{y}_2 \overset{\mathcal{T}}{\mapsto} \dots \overset{\mathcal{T}}{\mapsto} \vec{y}_\ell$. We say that a position i is *repeatable in* S if $\mathsf{top}(\vec{y}_i) = \mathsf{top}(\vec{y}_\ell)$ and $\mathsf{tail}(\vec{y}_i)$ is a suffix of $\mathsf{tail}(\vec{y}_j)$, for all positions j with $i \le j \le \ell$. Intuitively, the effect of the series of rewriting steps that follow a repeatable position in the sequence S is to replace the topmost element with a non-empty block of components that has the same element at the top. Note that every sequence S has a repeatable position (possibly the last one), so we can denote by $r(S)$ the *first repeatable position* in S.

Now, suppose that $\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}, \vec{y}' \,]\!]$ is a move of Repairer, where $\vec{x}$ has a non-horizontal component at the top that is not immediately preceded by a separator, and let $S : \vec{y} = \vec{y}_1 \overset{\mathcal{T}}{\mapsto} \vec{y}_2 \overset{\mathcal{T}}{\mapsto} \dots \overset{\mathcal{T}}{\mapsto} \vec{y}_\ell = \vec{y}'$ be the sequence of basic rewriting steps associated with this move. Recall that in the original game Referee would insert a separator symbol just below the top elements of the two stacks $\vec{x}$ and $\vec{y}'$. In the modified game, instead, we allow the separator symbol to be inserted at any position between the top $|\vec{y}'| - |\vec{y}_{r(S)}| + 1$ elements of $\vec{y}'$. Furthermore, this position can be chosen by Repairer. We denote by $\mathcal{G}'_{\mathcal{R},\mathcal{T}}$ the arena of the game modified as we just described.

Below, we prove that the two versions of the game are equivalent, namely, they admit the same winner.

Lemma 8 *Repairers wins the game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ if and only if he wins the modified game over $\mathcal{G}'_{\mathcal{R},\mathcal{T}}$.*

Proof. One direction is straightforward: if Repairer wins the game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, then he can also win the game over $\mathcal{G}'_{\mathcal{R},\mathcal{T}}$ by choosing to insert the separator symbol always below the top element of his stack.

For the converse direction, we consider pairs of moves over the modified arena $\mathcal{G}'_{\mathcal{R},\mathcal{T}}$ where a new separator is inserted, namely, moves of the form

$$\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}, Y_1 \dots Y_h]\!] \overset{\mathsf{Ref}}{\mapsto} [\![\mathsf{top}(\vec{x}) \cdot \triangleleft \cdot \mathsf{tail}(\vec{x}),\ (Y_1 \dots Y_i) \cdot \triangleleft \cdot (Y_{i+1} \dots Y_k \dots Y_h)]\!] \tag{$\dagger$}$$

where $1 \leq i < k \leq h$, $\mathsf{tail}(\vec{y}_{r(S)}) = Y_k \dots Y_h$, and S is a sequence of basic rewriting steps that transform $\vec{y}$ into $Y_1 \dots Y_h$ (note that the separator has been correctly inserted above the suffix $\mathsf{tail}(\vec{y}_{r(S)})$).

The general idea is to show that, starting from moves of the above form, Repairer can simulate any strategy for the modified game with a strategy for the original game. Doing so, however, might be problematic for the following reason. During subsequent rounds in the modified game, Repairer may choose to perform a series of pop operations and reach a stack with a top stack element among $Y_2, \dots, Y_i$. On the other hand, in the original version of the game, the elements $Y_2, \dots, Y_i$ occur under a separator symbol and thus cannot be easily 'accessed' by Repairer. Formally, we aim at proving that, whenever Repairer can induce a play in the modified game that uses the subsequent moves in ($\dagger$) and that reaches later a position of the form

$$\langle\!\langle\, \vec{x}' \cdot \triangleleft \cdot \mathsf{tail}(\vec{x}),\ (Y_j \dots Y_i) \cdot \triangleleft \cdot (Y_{i+1} \dots Y_k \dots Y_h) \,\rangle\!\rangle,$$

with $1 \leq j \leq i$, then, using a similar strategy, he can induce a play in the original game that uses the subsequent moves

$$\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}, Y_1 \dots Y_h]\!] \overset{\mathsf{Ref}}{\mapsto} [\![\mathsf{top}(\vec{x}) \cdot \triangleleft \cdot \mathsf{tail}(\vec{x}),\ Y_1 \cdot \triangleleft \cdot (Y_2 \dots Y_i\, Y_{i+1} \dots Y_k \dots Y_h]\!] \tag{$\ddagger$}$$

and that reaches a position of the form

$$\langle\!\langle\, \vec{x}' \cdot \triangleleft \cdot \mathsf{tail}(\vec{x}),\ (Y_j \dots Y_i) \cdot \triangleleft \cdot (Y_2 \dots Y_i \cdot Y_{i+1} \dots Y_k \dots Y_h) \,\rangle\!\rangle,$$

We will see later how this implies that if Repairer wins the modified game over $\mathcal{G}'_{\mathcal{R},\mathcal{T}}$, then he can also win the original game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

We begin by observing an important property of the portion $Y_1 \dots Y_i$ of the stack that lies above the separator symbol in ($\dagger$). Let S be a sequence $\vec{y} = \vec{y}_1 \overset{\mathcal{T}}{\mapsto} \vec{y}_2 \overset{\mathcal{T}}{\mapsto} \dots \overset{\mathcal{T}}{\mapsto} \vec{y}_\ell = Y_1 \dots Y_h$ of basic rewriting steps that can be associated with the move $\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}, Y_1 \dots Y_h]\!]$. Since $r(S)$ is a repeatable position in S, we know that $\mathsf{top}(\vec{y}_{r(S)}) = Y_1$ and that $\mathsf{tail}(\vec{y}_{r(S)}) = Y_k \dots Y_h$ is a suffix of $Y_2 \dots Y_h$. Note that $\vec{y}_{r(S)} = Y_1\, Y_k \dots Y_h$, and hence the sequence of rewriting steps that transform $\vec{y}_{r(S)}$ into $Y_1 \dots Y_k$ can be pumped an arbitrary number of times, e.g.

$$\underbrace{Y_1 Y_k \dots Y_h}_{= \vec{y}_{r(S)}} \overset{\mathcal{T}}{\mapsto}{}^* (Y_1 \dots Y_{k-1}) \cdot (Y_k \dots Y_h) \overset{\mathcal{T}}{\mapsto}{}^* (Y_1 \dots Y_{k-1}) \cdot (Y_2 \dots Y_{k-1}) \cdot (Y_k \dots Y_h).$$

The above property can be exploited by Repairer when playing in the original version of the game, as follows. Any series of push-and-swap operations performed in the modified game can be immediately executed in the original game, since the top stack element is preserved. When a series of pop operations is performed in

the modified game, Repairer can simulate it in the original game by first pushing a new copy of the block $Y_1 \dots Y_{k-1}$ (using the transformation associated with $\vec{y}_{r(S)}$) and then applying a series of pop operations that reveal the correct component, that is:

$$Y_1 \cdot \triangleleft \cdot (Y_2 \dots Y_h) \quad \overset{\mathcal{T}}{\mapsto}{}^* \quad (Y_1 \dots Y_{k-1}) \cdot \triangleleft \cdot (Y_2 \dots Y_h) \quad \overset{\mathcal{T}}{\mapsto}{}^* \quad (Y_j \dots Y_{k-1}) \cdot \triangleleft \cdot (Y_2 \dots Y_h)$$

for any j, with $2 \le j \le i$. This means that, whenever Repairer can reach, in the modified game, a position of the form $\langle\!\langle\, \vec{x}' \cdot \triangleleft \cdot \mathsf{tail}(\vec{x}),\, (Y_j \dots Y_i) \cdot \triangleleft \cdot (Y_{i+1} \dots Y_k \dots Y_h) \,\rangle\!\rangle$, then in a similar way he can reach, in the original game, a position of the form $\langle\!\langle\, \vec{x}' \cdot \triangleleft \cdot \mathsf{tail}(\vec{x}),\, (Y_j \dots Y_i) \cdot \triangleleft \cdot (Y_2 \dots Y_i \cdot Y_{i+1} \dots Y_k \dots Y_h) \,\rangle\!\rangle$.

To conclude, we observe that the part of the stack under the separator that is reached in the original game (i.e. $(Y_2 \dots Y_i \cdot Y_{i+1} \dots Y_k \dots Y_h)$) contains as a suffix the part of the stack under the separator that is reached in the modified game (i.e. $(Y_{i+1} \dots Y_k \dots Y_h)$). Therefore, if at any moment Referee removes the separator symbols from the stacks, Repairer can immediately reach in the original game the same position that was reached in the modified game. Moreover, all the above arguments carry over in a straightforward way when there are multiple occurrences of separator symbols. This shows that if Repairer wins the modified game, then he also wins the original game. □

We now turn to proving that the strategy of Repairer that we constructed from the transducer $\mathcal{Z}$ is correct for the modified game (this will imply the existence of a correct and winning strategy for Repairer in the original game).

Consider a generic position $\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle$ owned by Repairer. We will make use of the notation and the key properties stated in Claim 3. In particular, we fix the component X (= $\mathsf{top}(\vec{x})$), the multi-contexts E and E' (such that E' extends E, $\big[\delta^\circ(p_0, E)\big]_{\mathcal{R}} = \vec{x}$, and $\mathsf{top}(\delta^\circ(p_0, E')) \in X$), and the fingerprint context D_X for X (which is cyclic on $\mathsf{top}(\delta^\circ(p_0, E'))$). We know that the move induced by Repairer's strategy is of the form

$$\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle \quad \overset{\mathsf{Rep}}{\mapsto} \quad [\![\, \vec{x}, \vec{y}_{n_0+1} \,]\!]$$

where $\vec{y}_n = \big[\gamma^\circ(q_0, \mathcal{Z}(u_n))\big]_{\mathcal{T}}$, $u_n = \hat{E}'^{\,\mathsf{prefix}} \cdot (\hat{D}_X^{\mathsf{prefix}})^n$, and n_0 is a sufficiently large number such that, for some component Y of $\mathcal{T}$ and for all $n \in \mathbb{N}$, $\mathsf{top}(\vec{y}_{n_0+n}) = Y$ and $D_X \in \mathscr{L}(\mathcal{T} \mid Y)$. Now, we apply Lemma 7 to derive

$$\begin{aligned}
\vec{y}_{n_0+1} &= \big[\gamma^\circ\big(q_0, \mathcal{Z}(u_{n_0+1})\big)\big]_{\mathcal{T}} \\
&= \big[\gamma^\circ\big(q_0, \mathcal{Z}(\hat{E}'^{\,\mathsf{prefix}} \cdot (\hat{D}_X^{\mathsf{prefix}})^{n_0+1})\big)\big]_{\mathcal{T}} \\
&= \big[\gamma^\circ\big(q_0, \mathcal{Z}(\hat{E}'^{\,\mathsf{prefix}} \cdot (\hat{D}_X^{\mathsf{prefix}})^{n_0}) \cdot \hat{D}_X^{\mathsf{prefix}}\big)\big]_{\mathcal{T}} \\
&= \vec{y}'' \cdot \mathsf{tail}(\vec{y}_{n_0})
\end{aligned}$$

for some non-empty $\vec{y}''$. This means that the stack $\vec{y}_{n_0}$ occurs at a *repeatable* position in the sequence of basic rewriting steps associated with the move $\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}, \vec{y}_{n_0+1} \,]\!]$:

$$\vec{y} \quad \overset{\mathcal{T}}{\mapsto} \quad \dots \quad \overset{\mathcal{T}}{\mapsto} \quad \boldsymbol{\vec{y}_{n_0}} \quad \overset{\mathcal{T}}{\mapsto} \quad \dots \quad \overset{\mathcal{T}}{\mapsto} \quad \vec{y}_{n_0+1} \;=\; \vec{y}'' \cdot \mathsf{tail}(\vec{y}_{n_0}).$$

Therefore, if $\vec{x}$ has a non-horizontal component at the top that is not immediately preceded by a separator, in the modified game Repairer can choose to place the separator symbol just above the suffix $\mathsf{tail}(\vec{y}_{n_0})$ of $\vec{y}_{n_0+1}$, as follows:

$$\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}, \vec{y}_{n_0+1} \,]\!] \overset{\mathsf{Ref}}{\mapsto} [\![\, \mathsf{top}(\vec{x}) \cdot \triangleleft \cdot \mathsf{tail}(\vec{x}),\ \vec{y}'' \cdot \triangleleft \cdot \mathsf{tail}(\vec{y}_{n_0}) \,]\!].$$

It remains to show that the defined strategy of Repairer never removes components that lie under the separator symbol. Consider again a position $\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle$ owned by Repairer. We recall from Lemma 6 that if the component $X = \mathsf{top}(\vec{x})$ is non-horizontal, then the fingerprint context D_X is vertical. In particular, the serialization of D_X is a string of the form $a \cdot \hat{D} \cdot \bar{a}$, for some letter $a \in \Sigma$ and some context D. We consider a generic prolongation of u_{n_0+1} of the form $u_{n_0+1} \cdot \hat{t}$, where t is a tree or a forest, that is a prefix of a serialized tree in the restriction language $\mathscr{L}(\mathcal{R})$. The corresponding output of the transducer $\mathcal{Z}$ is of the form $\mathcal{Z}(u_{n_0+1} \cdot \hat{t}) = \mathcal{Z}(u_{n_0+1}) \cdot w$, for some word $w \in (\Delta \uplus \bar{\Delta})^*$. In particular, thanks to the previous definitions and the properties claimed in the first two items of Lemma 7, we have:

$$\begin{aligned}
\mathcal{Z}\big(u_{n_0+1} \cdot \hat{t} \cdot \hat{D}_X^{\mathsf{suffix}}\big) &= \mathcal{Z}\big(u_{n_0} \cdot \hat{D}_X^{\mathsf{prefix}} \cdot \hat{t} \cdot \hat{D}_X^{\mathsf{suffix}}\big) \\
&= \mathcal{Z}\big(u_{n_0} \cdot \hat{D}_X^{\mathsf{prefix}} \cdot \hat{t}\big) \cdot \hat{D}_X^{\mathsf{suffix}} \\
&= \mathcal{Z}\big(u_{n_0} \cdot \hat{D}_X^{\mathsf{prefix}}\big) \cdot w \cdot \hat{D}_X^{\mathsf{suffix}} \\
&= \mathcal{Z}\big(u_{n_0}\big) \cdot \hat{D}_X^{\mathsf{prefix}} \cdot w \cdot \hat{D}_X^{\mathsf{suffix}} \\
&= \mathcal{Z}\big(u_{n_0}\big) \cdot a \cdot \hat{D}^{\mathsf{prefix}} \cdot w \cdot \hat{D}^{\mathsf{suffix}} \cdot \bar{a}.
\end{aligned}$$

Moreover, since $\mathcal{Z}$ is a *tree edit* transducer, the infix $\hat{D}^{\mathsf{prefix}} \cdot w \cdot \hat{D}^{\mathsf{suffix}}$ between the opening tag a and the matching closing tag $\bar{a}$ is a well-nested word. This means that $\mathsf{tail}(\vec{y}_{n_0})$ is not only a suffix of $\vec{y}'$, but also a suffix of $\big[\gamma^\circ(q_0, \mathcal{Z}(u_{n_0+1} \cdot \hat{t}))\big]_{\mathcal{T}}$. Finally, since t was chosen arbitrarily, this allows us to conclude that, during any possible continuation of the game from the position $\langle\!\langle\, \vec{x}, \vec{y} \,\rangle\!\rangle$, Repairer will never pop an element from the suffix $\mathsf{tail}(\vec{y}_{n_0})$ before parsing the closing tag $\bar{a}$ that matches the last letter in $u_{n_0} \cdot a$, that is, before Generator pops the top element of $\vec{x}$.

The above arguments show that the strategy for Repairer that we defined in the previous subsection induces valid moves in the modified arena $\mathcal{G}'_{\mathcal{R},\mathcal{T}}$. Moreover, this strategy is winning, that is, Repairer can always move. Finally, by Lemma 8 we conclude that Repairer has a similar strategy for winning the original game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

6 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 [13], 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 (i.e. in ExpSpace)

that simulates the plays in order to determine the winner of the game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, and possibly synthesizes a winning strategy. Below, we improve the complexity result that we just derived to a tight EXP bound.

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

Proof. Let $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$ and $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ be two given top-down deterministic tree automata recognizing the restriction and target languages.

We already mentioned that the stacks controlled by Generator in the simulation game over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ never exceed in length the number of components of $\mathcal{R}$ – this follows essentially from the fact that prefix-rewriting rules of the form $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} X_1\, X_2 \cdot \vec{x}$ are applicable only when $X_1 \neq X \neq X_2$ and both components X_1 and X_2 are accessible from X (i.e. $\delta(p, a) = (p_1, p_2)$ for some $p \in X$, $p_1 \in X_1$, $p_2 \in X_2$, and $a \in \Sigma$). Unfortunately, an analogous bound on the lengths of the stacks controlled by Repairer does not hold. Indeed, the prefix-rewriting system associated with the target automaton can produce arbitrarily long stacks by repeatedly applying rules of the form $Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y\, Y_2 \cdot \vec{y}$, which insert a new component Y_2 below the top component Y. Since the latter type of rules are essentially the root of our problem, we give them the name of *head recursive* rules. Note that rules of the form $Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y_1\, Y \cdot \vec{y}$ are not head recursive, since the component Y can be rewritten only after Y_1 is popped. Moreover, since the rewriting operations follow the accessibility relation between components of $\mathcal{T}$, head recursive rules are the only ones that can be used to generate stacks of arbitrary height. Thus, to bound the height of the stacks and be able to efficiently simulate plays, we consider below an equivalent version of the game that is obtained by modifying the prefix-rewriting rules associated with $\mathcal{T}$ so as to avoid recursion.

We begin by describing the modified prefix-rewriting system obtained from $\overset{\mathcal{T}}{\mapsto}$. For each component Y of $\mathcal{T}$, we introduce a *dummy component* $\widetilde{Y}$, recognizing the empty language of contexts (in particular $\widetilde{Y}$ does not cover any component of $\mathcal{R}$). By a slight abuse of notation, we denote by $\mathsf{SCC}(\widetilde{\mathcal{T}})$ the new set of components, which includes the original components of $\mathcal{T}$ and their dummy copies. We begin by replacing all rules that are not head recursive and of the form

$$Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y_1\, Y_2 \cdot \vec{y} \qquad \text{and} \qquad Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y_1\, Y \cdot \vec{y}$$

where $Y_1 \neq Y$ and $Y_2 \neq Y$, respectively with the rules

$$Y \cdot \vec{y} \overset{\widetilde{\mathcal{T}}}{\mapsto} Y_1\, \widetilde{Y}_1\, Y_2\, \widetilde{Y}_2 \cdot \vec{y} \qquad \text{and} \qquad Y \cdot \vec{y} \overset{\widetilde{\mathcal{T}}}{\mapsto} Y_1\, \widetilde{Y}_1\, Y \cdot \vec{y}$$

In other words, we insert the dummy component $\widetilde{Y}_i$ below each component Y_i produced by a rule in $\overset{\mathcal{T}}{\mapsto}$ that is not head recursive. Subsequently, we replace all head recursive rules in $\overset{\mathcal{T}}{\mapsto}$ of the form

$$Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y\, Y_2 \cdot \vec{y} \qquad \text{and} \qquad Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y\, Y \cdot \vec{y}$$

where $Y_2 \neq Y$, respectively with the rules

$$\widetilde{Y} \cdot \vec{y} \overset{\widetilde{\mathcal{T}}}{\mapsto} Y_2\, \widetilde{Y}_2\, \widetilde{Y} \cdot \vec{y}. \qquad \text{and} \qquad \widetilde{Y} \cdot \vec{y} \overset{\widetilde{\mathcal{T}}}{\mapsto} Y\, \widetilde{Y} \cdot \vec{y}.$$

Finally, we keep the usual pop operations on the components of $\mathcal{T}$ and we add similar operations on their dummy copies:

$$\widetilde{Y} \cdot \vec{y} \;\; \overset{\widetilde{\mathcal{T}}}{\mapsto} \;\; \vec{y}.$$

We denote by $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ the modified prefix-rewriting system, and we denote, as usual, by $\overset{\widetilde{\mathcal{T}}}{\mapsto}^*$ the reflexive and transitive closure of $\overset{\widetilde{\mathcal{T}}}{\mapsto}$. We also define the arena $\mathcal{G}_{\mathcal{R},\widetilde{\mathcal{T}}}$ exactly as in Definition 1, using the rewriting systems $\overset{\mathcal{R}}{\mapsto}$ and $\overset{\widetilde{\mathcal{T}}}{\mapsto}^*$ and by letting the initial position be $\langle\!\langle\, X_0 \,,\, Y_0\, \widetilde{Y}_0 \,\rangle\!\rangle$, where X_0 is the component of the initial state of $\mathcal{R}$, Y_0 is the component of the initial state of $\mathcal{T}$, and $\widetilde{Y}_0$ is the dummy copy of Y_0.

We observe that $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ does not contain head recursive rules, that is, rules of the form $Y \cdot \vec{y} \;\; \overset{\mathcal{T}}{\mapsto} \;\; Y\, Y_2 \cdot \vec{y}$. More generally, one can easily verify that $Y \;\; \overset{\widetilde{\mathcal{T}}}{\mapsto}^* \;\; \vec{y}' \cdot Y \cdot \vec{y}''$ implies $\vec{y}'' = \varepsilon$, namely, whenever a component Y is rewritten into a new stack $\vec{y}$, using possibly several steps satisfying $\overset{\widetilde{\mathcal{T}}}{\mapsto}$, then Y can appear only in the rightmost position of $\vec{y}$. Thanks to the absence of head recursive rules, we can easily see that the stacks derived from $Y_0\, \widetilde{Y}_0$ using $\overset{\widetilde{\mathcal{T}}}{\mapsto}^*$ have length at most linear in the size of $\mathsf{SCC}(\widetilde{\mathcal{T}})$. We will see later how this helps to derive an alternating polynomial space algorithm that solves the streaming bounded repairability problem.

Below, we prove that each of the two prefix-rewriting systems $\overset{\mathcal{T}}{\mapsto}$ and $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ can simulate the other when they start from pairs of stacks of the form Y and $Y\, \widetilde{Y}$, respectively. The notion of simulation is formally captured by the relation $\preceq_n$, that we define below and that is similar to standard definitions in the literature of verification and model checking problems [3].

Definition 6 Let $\mathcal{Y}$ and $\mathcal{Z}$ be two finite sets (e.g. sets of components), let $\approx$ be a relation over $\mathcal{Y} \times \mathcal{Z}$, and let $\overset{\mathcal{Y}}{\mapsto}$ and $\overset{\mathcal{Z}}{\mapsto}$ be two prefix-rewriting systems over stacks in $\mathcal{Y}^*$ and in $\mathcal{Z}^*$, respectively. We say that $\overset{\mathcal{Y}}{\mapsto}$ *is* $\approx$*-simulated by* $\overset{\mathcal{Z}}{\mapsto}$ *up to* n*-steps starting from* $\vec{y} \in \mathcal{Y}^*$ *and* $\vec{z} \in \mathcal{Z}^*$, and we denote it by $(\vec{y},\, \overset{\mathcal{Y}}{\mapsto}) \preceq_n (\vec{z},\, \overset{\mathcal{Z}}{\mapsto})$, if the following conditions hold:

1. $\mathsf{top}(\vec{y}) \approx \mathsf{top}(\vec{z})$, and
2. if $n > 0$ and $\vec{y} \overset{\mathcal{Y}}{\mapsto} \vec{y}'$, then $\vec{z} \overset{\mathcal{Z}}{\mapsto} \vec{z}'$ and $(\vec{y}',\, \overset{\mathcal{Y}}{\mapsto}) \preceq_{n-1} (\vec{z}',\, \overset{\mathcal{Z}}{\mapsto})$ for some $\vec{z}' \in \mathcal{Z}^*$.

Intuitively, the simulation $(\vec{y},\, \overset{\mathcal{Y}}{\mapsto}) \preceq_n (\vec{z},\, \overset{\mathcal{Z}}{\mapsto})$ requires that every sequence of n prefix-rewriting steps in $\overset{\mathcal{Y}}{\mapsto}$ that starts from $\vec{y}$ can be simulated by a sequence of n prefix-rewriting steps in $\overset{\mathcal{Z}}{\mapsto}$ that starts from $\vec{z}$, while preserving the relation $\approx$ on the top elements,

Below, we prove that there exist suitable simulations from $\overset{\mathcal{T}}{\mapsto}$ to $\overset{\widetilde{\mathcal{T}}}{\mapsto}^*$ and, symmetrically, from $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ to $\overset{\mathcal{T}}{\mapsto}^*$ – from this it will follow that the games defined over the arenas $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ and $\mathcal{G}_{\mathcal{R},\widetilde{\mathcal{T}}}$ are equivalent. More precisely, one direction amounts at choosing $\approx$ to be the identity relation between components of $\mathcal{T}$ and at proving that $\overset{\mathcal{T}}{\mapsto}$ is $\approx$-simulated by the reflexive and transitive closure of $\overset{\widetilde{\mathcal{T}}}{\mapsto}$, up to any number n of steps and starting from the initial stacks Y_0 and $Y_0\, \widetilde{Y}_0$:

Claim 1 Let $\approx$ be the identity relation over $\mathsf{SCC}(\mathcal{T}) \times \mathsf{SCC}(\widetilde{\mathcal{T}})$ such that $Y \approx Z$ if and only if $Y = Z$. For all numbers $n \in \mathbb{N}$, we have

$$(Y_0,\, \overset{\mathcal{T}}{\mapsto}) \;\preceq_n\; (Y_0\, \widetilde{Y}_0,\, \overset{\widetilde{\mathcal{T}}}{\mapsto}^*).$$

Proof of Claim 1. We remark that the following proof is uniform in n, in the sense that $\overset{\mathcal{T}}{\mapsto}$ is simulated by $\overset{\widetilde{\mathcal{T}}}{\mapsto}^*$ in a way that does not depend on the particular value of n – in fact one could derive from this proof a stronger simulation relation based on sequences of prefix-rewriting steps of arbitrary length. The general idea is to exploit the presence of dummy components $\widetilde{Y}$ to generate, so to say 'at runtime', those components that were inserted by head recursive rules of $\overset{\mathcal{T}}{\mapsto}$, which have no analogous rules in $\overset{\widetilde{\mathcal{T}}}{\mapsto}^*$. In order to identify pairs of stacks that admit simulations, we introduce a new relation $\searrow$ that holds between any two stacks $\vec{y} = Y_1 Y_2 \dots Y_k \in \mathsf{SCC}(\mathcal{T})^*$ and $\vec{z} \in \mathsf{SCC}(\widetilde{\mathcal{T}})^*$ whenever there exist $\vec{z}_1, \dots, \vec{z}_k$ such that

$$\vec{z} \;=\; Y_1 \widetilde{Y}_1 \cdot \vec{z}_1 \;\overset{\widetilde{\mathcal{T}}}{\mapsto}\; \widetilde{Y}_1 \cdot \vec{z}_1 \;\overset{\widetilde{\mathcal{T}}}{\mapsto}^*\; Y_2 \widetilde{Y}_2 \cdot \vec{z}_2 \;\overset{\widetilde{\mathcal{T}}}{\mapsto}\; \widetilde{Y}_2 \cdot \vec{z}_2 \;\overset{\widetilde{\mathcal{T}}}{\mapsto}^*\; \dots \;\overset{\widetilde{\mathcal{T}}}{\mapsto}^*\; Y_k \widetilde{Y}_k \cdot \vec{z}_k. \quad (1)$$

Note that the relation $\searrow$ holds trivially between the initial stacks $\vec{y} = Y_0$ and $\vec{z} = Y_0 \widetilde{Y}_0$. Moreover, $\vec{y} \searrow \vec{z}$ clearly implies the first condition $\mathsf{top}(\vec{y}) \approx \mathsf{top}(\vec{z})$ of Definition 6. Below, we exploit a case distinction to verify that if $\vec{y} \searrow \vec{z}$ and $\vec{y} \overset{\mathcal{T}}{\mapsto} \vec{y}'$ hold, then there exists $\vec{z}' \in \mathsf{SCC}(\widetilde{\mathcal{T}})^*$ such that $\vec{y}' \searrow \vec{z}'$ and $\vec{z} \overset{\widetilde{\mathcal{T}}}{\mapsto}^* \vec{z}'$. From this it will immediately follow that $(Y_0, \overset{\mathcal{T}}{\mapsto}) \preceq_n (Y_0 \widetilde{Y}_0, \overset{\widetilde{\mathcal{T}}}{\mapsto})$ holds for all $n \in \mathbb{N}$.

Suppose that $\vec{y} \searrow \vec{z}$ and $\vec{y} \overset{\mathcal{T}}{\mapsto} \vec{y}'$. We distinguish between the following cases:

1. If $\vec{y}$ is rewritten into $\vec{y}'$ by a push-and-swap operation that is not head recursive, say
$$\vec{y} \;=\; Y \cdot \mathsf{tail}(\vec{y}) \;\overset{\mathcal{T}}{\mapsto}\; Y_1 Y_2 \cdot \mathsf{tail}(\vec{y}) \;=\; \vec{y}'$$
with $Y_1 \neq Y$ and $Y_2 \neq Y$, then we can simply apply to $\vec{z}$ the rule $Y \cdot \mathsf{tail}(\vec{z}) \overset{\widetilde{\mathcal{T}}}{\mapsto} Y_1 \widetilde{Y}_1 Y_2 \widetilde{Y}_2 \cdot \mathsf{tail}(\vec{z})$ – note that this rule, with $\mathsf{tail}(\vec{y})$ in place of $\mathsf{tail}(\vec{z})$, was used to replace the considered push-and-swap operation of $\overset{\mathcal{T}}{\mapsto}$. In this way we transform the stack $\vec{z} = Y \cdot \mathsf{tail}(\vec{z})$ into the stack $\vec{z}' = Y_1 \widetilde{Y}_1 Y_2 \widetilde{Y}_2 \cdot \mathsf{tail}(\vec{z})$. Moreover, it is immediate to see that $\vec{y}' \searrow \vec{z}'$.
The case of a push-and-swap operation of the form $\vec{y} = Y \cdot \mathsf{tail}(\vec{y}) \overset{\mathcal{T}}{\mapsto} Y_1 Y \cdot \mathsf{tail}(\vec{y}) = \vec{y}'$ is similar, as one can apply a corresponding rule in $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ to transform the stack $\vec{z} = Y \widetilde{Y} \cdot \vec{z}''$ into the stack $\vec{z}' = Y_1 \widetilde{Y}_1 Y \widetilde{Y} \cdot \vec{z}''$ such that $\vec{y}' \searrow \vec{z}'$.
2. A more interesting case is when a head recursive push-and-swap operation of $\overset{\mathcal{T}}{\mapsto}$ is applied, in which case we just let $\vec{z}' = \vec{z}$ (namely, we take no action in the system $\overset{\widetilde{\mathcal{T}}}{\mapsto}^*$). To show that this choice is correct it suffices to verify that $\vec{y}' \searrow \vec{z}$. Suppose that the head recursive push-and-swap operation on $\vec{y}$ is of the form
$$\vec{y} \;=\; Y \cdot \mathsf{tail}(\vec{y}) \;\overset{\mathcal{T}}{\mapsto}\; Y Y_2 \cdot \mathsf{tail}(\vec{y}) \;=\; \vec{y}',$$
where $Y_2 \neq Y$ (the case where $Y_2 = Y$ is similar). We recall that the above operation, devoid of the occurrences of $\mathsf{tail}(\vec{y})$, is replaced by a prefix-rewriting rule of $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ of the form
$$\widetilde{Y} \;\overset{\widetilde{\mathcal{T}}}{\mapsto}\; Y_2 \widetilde{Y}_2 \widetilde{Y}.$$
We also derive from the assumption $\vec{y} \searrow \vec{z}$ the existence of some stacks $\vec{z}_1$ and $\vec{z}''$ such that $\vec{z} = Y \widetilde{Y} \cdot \vec{z}_1$, $\widetilde{Y} \cdot \vec{z}_1 \overset{\widetilde{\mathcal{T}}}{\mapsto}^* \vec{z}''$, and $\mathsf{tail}(\vec{y}) \searrow \vec{z}''$. To prove that

$\vec{y}' = Y\,Y_2 \cdot \mathsf{tail}(\vec{y}) \searrow \vec{z}$, it suffices to define $\vec{z}_2 = \widetilde{Y} \cdot \vec{z}_1$ as the intermediate stack that witnesses a derivation of $\overset{\widetilde{\mathcal{T}}}{\mapsto}{}^*$ satisfying Equation (1), that is:

$$\vec{z} \;=\; Y\,\widetilde{Y} \cdot \vec{z}_1 \;\overset{\widetilde{\mathcal{T}}}{\mapsto}\; \widetilde{Y} \cdot \vec{z}_1 \;\overset{\widetilde{\mathcal{T}}}{\mapsto}{}^*\; Y_2\,\widetilde{Y}_2 \underbrace{\widetilde{Y} \cdot \vec{z}_1}_{\vec{z}_2} \;\overset{\widetilde{\mathcal{T}}}{\mapsto}\; \widetilde{Y}_2\,\widetilde{Y} \cdot \vec{z}_1 \;\overset{\widetilde{\mathcal{T}}}{\mapsto}{}^*\; \vec{z}''$$

3. The last interesting case is when a pop operation is executed that transforms $\vec{y} = Y_1\,Y_2 \ldots Y_k$ into $\vec{y}' = Y_2 \ldots Y_k$. In this case, we derive form the assumption $\vec{y} \searrow \vec{z}$ the existence of a sequence of stacks $\vec{z}_1, \ldots, \vec{z}_k$ satisfying Equation (1). We respond to the pop operation by transforming $\vec{z}$ into $\vec{z}' = Y_2\,\widetilde{Y}_2 \cdot \vec{z}_2$, that is, by first popping the top element of $\vec{z}$ $(= Y_1\,\widetilde{Y}_1 \cdot \vec{z}_1)$, so as to reach the stack $\widetilde{Y}_1 \cdot \vec{z}_1$, and then applying the derivation $\widetilde{Y}_1 \cdot \vec{z}_1 \overset{\widetilde{\mathcal{T}}}{\mapsto}{}^*\; Y_2\,\widetilde{Y}_2 \cdot \vec{z}_2 = \vec{z}'$, as witnessed by Equation (1). Finally, it is easy to verify that $\vec{y}' \searrow \vec{z}'$.

This completes the proof of Claim 1. □

The other direction of the simulation, namely, the one that goes from $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ to $\overset{\mathcal{T}}{\mapsto}{}^*$, is slightly more complicated. First of all, because the basic prefix-rewriting steps of $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ may insert dummy components at the top of the stacks, we need to compare these dummy components with real components of $\mathcal{T}$. For this we define the relation $\approx$ over $\mathsf{SCC}(\mathcal{T}) \times \mathsf{SCC}(\widetilde{\mathcal{T}})$ so as to pair any two components Y and Z, when Z is dummy. We anticipate that such a definition is introduced here only to ease the inductive construction of a simulation – in particular, the specific relationships with the dummy components are immaterial with respect to the moves of the game over $\mathcal{G}_{\mathcal{R},\widetilde{\mathcal{T}}}$. We then aim at proving the following claim:

Claim 2. Let $\approx$ be the relation over $\mathsf{SCC}(\mathcal{T}) \times \mathsf{SCC}(\widetilde{\mathcal{T}})$ such that $Y \approx Z$ if and only if $Y = Z$ or Z is a dummy component. For all $n \in \mathbb{N}$, we have

$$(Y_0\,\widetilde{Y}_0,\ \overset{\widetilde{\mathcal{T}}}{\mapsto}) \;\preceq_n\; (Y_0,\ \overset{\mathcal{T}}{\mapsto}{}^*).$$

Proof of Claim 2.. The difficulties here are somehow symmetric to those of the previous proof. In particular, in the previous proof we had to simulate properly the head recursive rules of $\overset{\mathcal{T}}{\mapsto}$, while here the system $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ to be simulated has no head recursion. On the other hand, the rules of $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ can generate new stacks from dummy components, which is clearly not possible in the system $\overset{\mathcal{T}}{\mapsto}{}^*$. In order to be able to simulate the latter rules on a dummy component $\widetilde{Y}$, we need to act beforehand, namely, when the corresponding component Y appears at the top of the stack. Intuitively, when Y is at the top of a stack, we can execute some head recursive push-and-swap operations of $\overset{\mathcal{T}}{\mapsto}$ so as to generate long enough stacks that can be used later to 'cover' the possible derivations from the dummy copy $\widetilde{Y}$. To do so, we associate with each component Y of $\mathcal{T}$ the (possibly empty) string

$$\mathsf{pump}(Y) \;=\; Y_1 \ldots Y_k$$

that contains all and only the components Y_i (in some arbitrary order) that admit head recursive rules of the form $Y \overset{\mathcal{T}}{\mapsto} Y\,Y_i$. Note that every time a component

Y of $\mathcal{T}$ appears at the top of a stack, one can execute a series of push-and-swap operations in $\overset{\mathcal{T}}{\mapsto}$ that transform Y into $Y \cdot \big(\mathsf{pump}(Y)\big)^n$, for any given $n \in \mathbb{N}$.

We now turn towards describing the pairs of stacks that admit simulations up to n steps. For this we introduce a relation $\nwarrow_n$, also parametrized by n, that holds between any two stacks $\vec{y} \in \mathsf{SCC}(\mathcal{T})^*$ and $\vec{z} = Z_1 Z_2 \dots Z_h \in \mathsf{SCC}(\widetilde{\mathcal{T}})^*$ whenever there exist $\vec{y}_1, \dots, \vec{y}_h$ such that

$$\vec{y} \;=\; f_n(Z_1) \cdot \vec{y}_1 \;\overset{\mathcal{T}}{\mapsto}{}^*\; \vec{y}_1 \;\overset{\mathcal{T}}{\mapsto}{}^*\; f_n(Z_2) \cdot \vec{y}_2 \;\overset{\mathcal{T}}{\mapsto}{}^*\; \vec{y}_2 \;\overset{\mathcal{T}}{\mapsto}{}^*\; \dots \;\overset{\mathcal{T}}{\mapsto}{}^*\; f_n(Z_h) \cdot \vec{y}_h$$

where $f_n(Z)$ is either Y or $\big(\mathsf{pump}(Y)\big)^n$, depending on whether $Z = Y \in \mathsf{SCC}(\mathcal{T})$ or $Z = \widetilde{Y}$. Note that $\vec{y} \nwarrow_n \vec{z}$ implies $\mathsf{top}(\vec{y}) \approx \mathsf{top}(\vec{z})$. Moreover, from the initial stack Y_0 we can always execute a series of head recursive push-and-swap operations of $\overset{\mathcal{T}}{\mapsto}$ so as to obtain a stack $\vec{y}_0 = Y_0 \cdot f_n(\widetilde{Y}_0)$ such that $\vec{y}_0 \nwarrow_n Y_0 \widetilde{Y}_0$. Below, we prove that if $n > 0$, $\vec{y} \nwarrow_n \vec{z}$, and $\vec{z} \overset{\widetilde{\mathcal{T}}}{\mapsto} \vec{z}'$, then there exists a derivation $\vec{y} \overset{\mathcal{T}}{\mapsto}{}^* \vec{y}'$ such that $\vec{y}' \nwarrow_{n-1} \vec{z}'$. Based on all these results, we will be able to conclude that Claim 2 holds. As usual, the proof is by case distinction:

1. We consider first the case of a prefix-rewriting rule

$$\vec{z} \;=\; Y \cdot \mathsf{tail}(\vec{z}) \;\overset{\widetilde{\mathcal{T}}}{\mapsto}\; Y_1 \widetilde{Y}_1 Y_2 \widetilde{Y}_2 \cdot \mathsf{tail}(\vec{z}) \;=\; \vec{z}'$$

   where Y, Y_1, Y_2 are non-dummy components, $Y_1 \neq Y$, and $Y_2 \neq Y$. From the assumption $\vec{y} \nwarrow_n \vec{z}$ we derive that $\vec{y} = f_n(Y) \cdot \mathsf{tail}(\vec{y}) = Y \cdot \mathsf{tail}(\vec{y})$. Accordingly, we can respond to the above move with the push-and-swap operation $\vec{y} = Y \cdot \mathsf{tail}(\vec{y}) \overset{\mathcal{T}}{\mapsto} Y_1 Y_2 \cdot \mathsf{tail}(\vec{y})$, which is not head recursive, followed by a (possibly empty) series of head recursive push-and-swap operations that transform $Y_1 Y_2 \cdot \mathsf{tail}(\vec{y})$ into

$$\vec{y}' \;=\; Y_1 \cdot \big(\mathsf{pump}(Y_1)\big)^n \cdot Y_2 \cdot \mathsf{tail}(\vec{y}).$$

   It is easy to verify that $\vec{y}' \nwarrow_n \vec{z}'$, and hence $\vec{y}' \nwarrow_{n-1} \vec{z}'$.

   The case of a rule of the form $\vec{z} = Y \cdot \mathsf{tail}(\vec{z}) \overset{\widetilde{\mathcal{T}}}{\mapsto} Y_1 \widetilde{Y}_1 Y \cdot \mathsf{tail}(\vec{z}) = \vec{z}'$ is similar.

2. Next, we consider the case of a rule

$$\vec{z} \;=\; \widetilde{Y} \cdot \mathsf{tail}(\vec{z}) \;\overset{\widetilde{\mathcal{T}}}{\mapsto}\; Y_2 \widetilde{Y}_2 \widetilde{Y} \cdot \mathsf{tail}(\vec{z}) \;=\; \vec{z}'$$

   where $Y \neq Y_2$. Thanks to the assumption $\vec{y} \nwarrow_n \vec{z}$, we can write $\vec{y}$ as $\vec{y} = f_n(\widetilde{Y}) \cdot \vec{y}_1 = \big(\mathsf{pump}(Y)\big)^n \cdot \vec{y}_1$ and we can find a derivation $\vec{y}_1 \overset{\mathcal{T}}{\mapsto}{}^* \vec{y}_1'$ such that $\vec{y}_1' \nwarrow_n \mathsf{tail}(\vec{z})$. Consider now the head recursive push-and-swap operation $Y \overset{\mathcal{T}}{\mapsto} Y Y_2$, which corresponds to the above move of $\overset{\widetilde{\mathcal{T}}}{\mapsto}$, devoid of $\mathsf{tail}(\vec{z})$. Since $n > 0$, we know that $\big(\mathsf{pump}(Y)\big)^n = \mathsf{pump}(Y) \cdot \big(\mathsf{pump}(Y)\big)^{n-1}$ and that $\mathsf{pump}(Y)$ contains an occurrence of the component Y_2. This means that by applying a series of pop operations in $\overset{\mathcal{T}}{\mapsto}{}^*$, we can transform the stack $\vec{y} = \big(\mathsf{pump}(Y)\big)^n \cdot \vec{y}_1$ into a stack of the form

$$\vec{y}' \;=\; Y_2 \cdot \vec{y}'' \cdot \big(\mathsf{pump}(Y)\big)^{n-1} \cdot \vec{y}_1,$$

   for some proper suffix $\vec{y}''$ of $\mathsf{pump}(Y)$. Moreover, to verify that $\vec{y}' \nwarrow_{n-1} \vec{z}'$ it suffices to recall that $\vec{z}' = Y_2 \widetilde{Y}_2 \widetilde{Y} \cdot \mathsf{tail}(\vec{z})$ and $\vec{y}_1 \overset{\mathcal{T}}{\mapsto}{}^* \vec{y}_1' \nwarrow_n \mathsf{tail}(\vec{z})$, and to

construct the following derivation:

$$\underbrace{Y_2}_{f_{n-1}(Y_2)} \cdot \vec{y}'' \cdot \big(\mathsf{pump}(Y)\big)^{n-1} \cdot \vec{y}_1 \;\overset{\mathcal{T}}{\mapsto}^{*}\; \underbrace{\big(\mathsf{pump}(Y_2)\big)^{n-1}}_{f_{n-1}(\widetilde{Y}_2)} \cdot \vec{y}'' \cdot \big(\mathsf{pump}(Y)\big)^{n-1} \cdot \vec{y}_1$$
$$\overset{\mathcal{T}}{\mapsto}^{*}\; \underbrace{\big(\mathsf{pump}(Y)\big)^{n-1}}_{f_{n-1}(\widetilde{Y})} \cdot \vec{y}_1 \;\overset{\mathcal{T}}{\mapsto}^{*}\; \vec{y}_1 \;\overset{\mathcal{T}}{\mapsto}^{*}\; \vec{y}_1'.$$

3. It remains to consider the case of a pop operation $\vec{z} = Z \cdot \mathsf{tail}(\vec{z}) \overset{\widetilde{\mathcal{T}}}{\mapsto} \mathsf{tail}(\vec{z}) = \vec{z}'$. The fact that it can be simulated by $\overset{\mathcal{T}}{\mapsto}^{*}$ follows easily from the definition $\vec{y} \nwarrow_n \vec{z}$. Indeed, if $\vec{z} = Z_1 Z_2 \dots Z_h$ and $\vec{y} \nwarrow_n \vec{z}$, then by popping the prefix $f_n(Z_1)$ from $\vec{y}$ and by transforming the remaining part $\vec{y}_1$ into $f_n(Z_2) \cdot \vec{y}_2$, we reach a stack $\vec{y}' = f_n(Z_2) \cdot \vec{y}_2$ such that $\vec{y}' \nwarrow_{n-1} \vec{z}'$.

This completes the proof of Claim 2. □

We proved that the prefix-rewriting system $\overset{\mathcal{T}}{\mapsto}$ is simulated by $\overset{\widetilde{\mathcal{T}}}{\mapsto}^{*}$ up to any number of steps, and, vice versa, the prefix-rewriting system $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ is simulated by $\overset{\mathcal{T}}{\mapsto}^{*}$ up to any number of runs. This allows us to claim that the reachability games over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ and over the arena $\mathcal{G}_{\mathcal{R},\widetilde{\mathcal{T}}}$ are equivalent, namely, they admit the same winner. Indeed, suppose that Repairer has a strategy to win the variant of the game over $\mathcal{G}_{\mathcal{R},\widetilde{\mathcal{T}}}$. Each move suggested by his strategy consists of a finite sequence of basic rewriting steps satisfying $\overset{\widetilde{\mathcal{T}}}{\mapsto}$. Since there are only finitely many plays over $\mathcal{G}_{\mathcal{R},\widetilde{\mathcal{T}}}$, we can define n to be the maximum number of basic rewriting steps executed by Repairer during every possible play over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ induced by his strategy. We then recall from Claim 2 that every sequence of basic prefix-rewriting steps of $\overset{\widetilde{\mathcal{T}}}{\mapsto}$ of length at most n can be simulated by a sequence of prefix-rewriting steps of $\overset{\widetilde{\mathcal{T}}}{\mapsto}^{*}$. Moreover, for every instance $(\vec{z}, \overset{\widetilde{\mathcal{T}}}{\mapsto}) \prec_{n'} (\vec{y}, \overset{\mathcal{T}}{\mapsto}^{*})$ of the simulation relation, as derived in Claim 2, either $\mathsf{top}(\vec{z}) = \mathsf{top}(\vec{y})$ or $\mathsf{top}(\vec{z})$ is a dummy component; in particular, we have $\mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{z})) \subseteq \mathscr{L}(\widetilde{\mathcal{T}} \mid \mathsf{top}(\vec{y}))$ (recall that a dummy component recognizes the empty language of contexts). One can finally use the simulation to construct a strategy for Repairer to win the game over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. The converse direction follows symmetric arguments and exploit Claim 1 – in fact, in this direction the proof is slightly simpler, since there is no need to fix a bound n to the length of the plays (recall that Claim 1 defines a simulation from $\overset{\mathcal{T}}{\mapsto}$ to $\overset{\widetilde{\mathcal{T}}}{\mapsto}^{*}$ in a uniform way, that is, independently of the number of moves to be simulated).

Thanks to the above arguments, we can decide whether there exists a streaming repair strategy from $\mathscr{L}(\mathcal{R})$ to $\mathscr{L}(\mathcal{T})$ with uniformly bounded cost by checking whether Repairer wins the game over $\mathcal{G}_{\mathcal{R},\widetilde{\mathcal{T}}}$. Furthermore, we recall that the stacks that can be reached over the arena $\mathcal{G}_{\mathcal{R},\widetilde{\mathcal{T}}}$ have length at most linear in the size of $\mathcal{R}$ and $\mathcal{T}$. This allows us to use an alternating polynomial-space procedure that simulates the possible plays in $\mathcal{G}_{\mathcal{R},\widetilde{\mathcal{T}}}$, eventually determining the winner of the game. □

The next theorem shows that the streaming bounded repairability problem for top-down deterministic tree automata is Exp-hard. In fact, it shows that Exp-hardness holds even for languages specified by *non-recursive deterministic DTDs*

(see Section 2). Given that any deterministic DTD can be efficiently translated into an equivalent top-down deterministic tree automaton [16], the EXP-hardness result can be transferred to languages recognized by top-down deterministic tree automata.

Theorem 3 *The problem of streaming bounded repairability for languages defined by non-recursive deterministic DTDs is* EXP*-hard.*

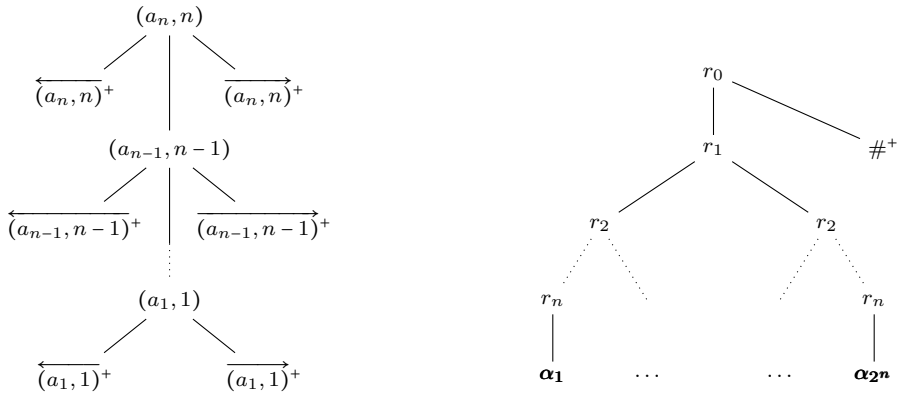
Proof. The proof is by reduction from the problem of deciding the winner of a tiling game over a corridor of polynomial width and exponential height. Formally, an instance of the latter problem is a tuple $I = (n, C, H, V, a_\perp)$, where n is the width of the corridor to be tiled (this number is presented in unary notation), C is a set of available tiles, $H, V \subseteq C \times C$ are the vertical and horizontal constraints, and $a_\perp \in C$ is a special tile that must appear at the bottom row. For any natural number k, we define a *tiling of height* k (for the instance I) to be any function g with domain $[1, k] \times [1, n]$ and codomain C. Furthermore, we say that the tiling g is *correct* if it satisfies the following constraints:

1. $g(1, j) = a_\perp$ for all $1 \leq j \leq n$,
2. $\big(g(i, j-1), g(i, j)\big) \in H$ for all $1 \leq i \leq k$ and all $1 < j \leq n$,
3. $\big(g(i-1, j), g(i, j)\big) \in V$ for all $1 < i \leq k$ and all $1 \leq j \leq n$.

The tiling game is run by two players, *Adam* and *Eve*, as follows. A configuration of the tiling game at round k is a correct tiling g_k of height k (accordingly, the empty tiling of height 0 is the initial configuration of the game). Adam moves at odd rounds (so he moves first by inserting a row of the form $a_\perp \dots a_\perp$), while Eve moves at even rounds. Given a correct tiling g_k at round k, the move of the corresponding player consists of extending g_k to a correct tiling g_{k+1} of height $k+1$. The player who cannot move, due to the enforced constraints, loses. Moreover, without loss of generality, we can assume that Adam wins as soon as the height of the tiling reaches 2^{n+1}. We know from [8] that the problem of deciding the winner of a tiling game is hard for alternating polynomial-space Turing machines, and hence EXP-hard.

The goal of the proof is to construct two DTDs $\mathcal{R}$ and $\mathcal{T}$ of size polynomial in $|I|$ that define languages R and T such that R is streaming bounded repairable into T if and only if Eve wins the instance I of the tiling game. As the reduction is quite technical, the reader may want to first look at a simplified version of it, which is given in the proof of Theorem 5.

The general intuition is that the restriction DTD $\mathcal{R}$ will generate encodings of rows of tiles, which represent the possible moves of Adam at the odd rounds. We will allow some redundancy in the encodings of Adam rows in order to forbid any repair processor to modify them with boundedly many edits. Symmetrically, the target DTD $\mathcal{T}$ will require 'interleaving' the rows produced by Adam with new rows, representing Eve responses to Adam. Since we cannot directly enforce that the rows generated by the restriction DTD satisfy the vertical constraints, we need to 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. In order to handle all the constraints of the reduction, we will adopt suitable encodings of rows of tiles based on trees. In particular, these



(a) Encoding of a single row. (b) Encoding of 2^n rows and dummy tiles.

Fig. 7 Encoding of rows produced by Adam. The nodes annotated with left and right arrows in the left figure give redundant encodings of the tiles at their parents (the superscript + denotes an arbitrary non-empty repetition of the underlying symbol).

encodings will ease detecting the possible violations of the vertical constraints, as well as checking that the height of a tiling never exceeds 2^{n+1}. We now describe in detail how the restriction and target DTDs encode the rows produced by Adam and Eve and how the target can expose a possible violation of the constraints.

We generically denote a row produced by Adam by $\alpha = a_1 \dots a_n$ (recall that n is the width of the corridor), and we encode such a row by means of a tree, like the one shown in Figure 7(a). We observe that, according to this encoding, the tiles $a_1, \dots, a_n$ of α are paired with the indices $1, \dots, n$ of the columns where they appear, and they are listed from bottom to top along the middle spine of the tree (e.g., the rightmost tile appears at the root of the spine). Moreover, to make the encoding robust to editings of bounded cost, tiles can be repeated an arbitrary number of times both to the left and to the right of the spine (these repetitions of tiles are encoded by new copies of the symbols (a_i, i) annotated with left and right arrows; the superscript + in Figure 7(a) denotes an arbitrary non-empty repetition of these symbols). We will see later how the above encoding eases the exposure of a possible violation of a vertical constraint between two contiguous rows (one produced by Adam and the other produced by Eve).

The list of the possible rows that could be produced by Adam is represented inside a larger tree. As Adam produces exactly 2^n rows, say $\alpha_1, \dots, \alpha_{2^n}$, it is sufficient to construct a full binary tree of height n and append to its leaves the encodings of $\alpha_1, \dots, \alpha_{2^n}$. Moreover, for a technical reason that will be clear later, Adam will produce a repetition of a dummy tile # at the end of the list of his rows. The rough intuition is that, if Adam cheats, then a repair process can easily get to the target language by deleting a dummy tile. A tree-shaped encoding of the rows provided by Adam is shown in Figure 7(b). For the sake of simplicity, we named the sub-trees containing the encodings of the rows of Adam with the bold letters $\alpha_1, \dots, \alpha_{2^n}$.

Trees of the above form are easily defined by a DTD of size polynomial in the instance I of the tiling game. Specifically, they belong to the following restriction

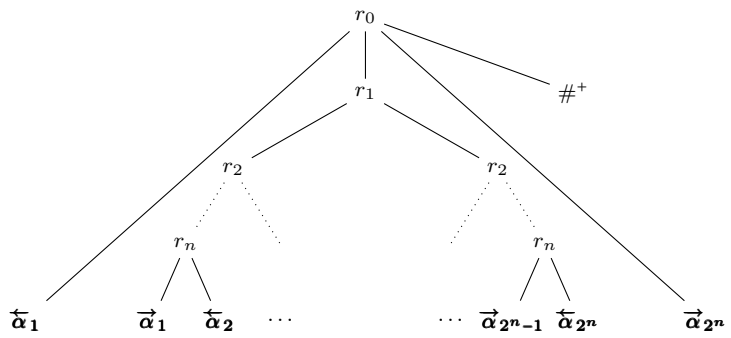


Fig. 8 Intermediate stage of a repair.

DTD (the symbols with no derivation rules are meant to be terminal):

$$
\begin{array}{lllll}
\mathcal{R}: & r_0 \to r_1\ \#^+ & & r_n \to \bigcup_{a\in C} (a,n) \\
& r_1 \to r_2\ r_2 & & (a,n) \to \overleftarrow{(a,n)}^+ \left(\bigcup_{(a',a)\in H} (a',n-1) \right) \overrightarrow{(a,n)}^+ \\
& \vdots & & \vdots \\
& r_{n-1} \to r_n\ r_n & & (a,1) \to \overleftarrow{(a,1)}^+\ \overrightarrow{(a,1)}^+
\end{array}
$$

We observe that the above DTD already enforces the horizontal constraints within the rows; the vertical constraints will be validated during the repair process.

Before describing the target language, we formalize a canonical repair strategy that is applicable to the serialization of a generic tree in the restriction language, under the assumption that Eve wins the tiling game. The first step of the canonical repair strategy consists of flattening the input tree from the restriction language by removing all nodes labeled with r_0, r_i, or (a,i), for all $1 \le i \le n$ and $a \in C$. This results in a series of forests $\overleftarrow{\alpha}_1$, $\overrightarrow{\alpha}_1$, …, $\overleftarrow{\alpha}_{2^n}$, $\overrightarrow{\alpha}_{2^n}$, where each $\overleftarrow{\alpha}_i$ (resp. $\overrightarrow{\alpha}_i$) can be seen as a redundant encoding of the i-th row α_i produced by Adam, read from right to left (resp. from left to right). The next step of the repair process consists of reconstructing a new full binary tree that spans the previously mentioned forests (possibly modified) and groups them into pairs, leaving out just the first and the last forest. This operation is required to induce a shift in the grouping of the forests, so as to ease the verification of the vertical constraints between consecutive rows. Intuitively, at this stage of the repair process an intermediate tree, like the one shown in Figure 8, is obtained (the nodes in bold can still be modified).

We remark that if Adam started the tiling game by inserting a row different from $\alpha_\perp = a_\perp \dots a_\perp$, already at this stage of the repair process one could easily detect the violation of the constraints and accordingly get into a suitable target language. In the following, we will describe the remaining editing operations, which are performed when Adam did not cheat right at the beginning.

We recall that Eve plays the tiling game according to a winning strategy. Because of that, and due to the shape of the intermediate tree, we know that Adam must produce at some point, say at some turn i^* of the game, a row that violates the vertical constraints. Until that moment, the canonical repair strategy can simply mimic the moves of Eve and modify the input tree as follows. Suppose

that at some turn $i < i^*$ Adam plays the row $\alpha_i = a_{i,1} \dots a_{i,n}$ and Eve can respond with the row $\beta_i = b_{i,1} \dots b_{i,n}$. At this moment, a prefix of the serialized input tree is disclosed and its flattening (obtained by removing the opening and closing tags for the nodes labeled over $\{r_i \mid 1 \le i \le n\} \cup \{(a,i) \mid a \in C, 1 \le i \le n\}$) ends with a string of the form

$$\underbrace{\Big(\overleftarrow{(a_{i,n},n)}\,\overline{\overleftarrow{(a_{i,n},n)}}\Big)^+ \dots \Big(\overleftarrow{(a_{i,1},1)}\,\overline{\overleftarrow{(a_{i,1},1)}}\Big)^+}_{\overleftarrow{\alpha}_i} \underbrace{\Big(\overrightarrow{(a_{i,1},1)}\,\overline{\overrightarrow{(a_{i,1},1)}}\Big)^+ \dots \Big(\overrightarrow{(a_{i,n},n)}\,\overline{\overrightarrow{(a_{i,n},n)}}\Big)^+}_{\overrightarrow{\alpha}_i}$$

(the overlined symbols denote the matching closing tags). Accordingly, the canonical repair strategy modifies the string $\overrightarrow{\alpha}_i$ by prepending the opening tag $(b_{i,j}, j)$ to each factor $\Big(\overrightarrow{(a_{i,j},j)}\,\overline{\overrightarrow{(a_{i,j},j)}}\Big)^+$, thus forming the output

$$\Big(\overleftarrow{(a_{i,n},n)}\,\overline{\overleftarrow{(a_{i,n},n)}}\Big)^+ \dots \Big(\overleftarrow{(a_{i,1},1)}\,\overline{\overleftarrow{(a_{i,1},1)}}\Big)^+ \mathbf{(b_{i,1},1)}\,\Big(\overrightarrow{(a_{i,1},1)}\,\overline{\overrightarrow{(a_{i,1},1)}}\Big)^+ \dots \mathbf{(b_{i,n},n)}\,\Big(\overrightarrow{(a_{i,n},n)}\,\overline{\overrightarrow{(a_{i,n},n)}}\Big)^+$$

(the inserted opening tags are listed in bold and will be closed at the next repair step). Suitable constraints in the target language will enforce the fact that the row $\beta_i = b_{i,1} \dots b_{i,n}$ inserted by Eve satisfies the horizontal constraints and also the vertical constraints with respect to the row $\alpha_i = a_{i,1} \dots a_{i,n}$ inserted by Adam. After this edit, the input is resumed and the first part of the encoding of the next row $\alpha_{i+1} = a_{i+1,1} \dots a_{i+1,n}$ is consumed, that is:

$$\underbrace{\Big(\overleftarrow{(a_{i+1,n},n)}\,\overline{\overleftarrow{(a_{i+1,n},n)}}\Big)^+ \dots \Big(\overleftarrow{(a_{i+1,1},1)}\,\overline{\overleftarrow{(a_{i+1,1},1)}}\Big)^+}_{\overrightarrow{\alpha}_{i+1}} .$$

Now, if the row α_{i+1} is correct, namely, if $i+1 < i^*$, then the canonical repair strategy appends to each block $\Big(\overleftarrow{(a_{i+1,j},j)}\,\overline{\overleftarrow{(a_{i+1,j},j)}}\Big)^+$ the closing tag $\overline{(b_{i,j},j)}$, thus forming the output

$$\Big(\overleftarrow{(a_{i+1,n},n)}\,\overline{\overleftarrow{(a_{i+1,n},n)}}\Big)^+ \overline{\mathbf{(b_{i,n},n)}} \dots \Big(\overleftarrow{(a_{i+1,1},1)}\,\overline{\overleftarrow{(a_{i+1,1},1)}}\Big)^+ \overline{\mathbf{(b_{i,1},1)}} .$$

Otherwise, if the row is not correct, namely, if $i+1 = i^*$, then we know that there is a column j that witnesses the violation, namely, such that $(b_{i,j}, a_{i+1,j}) \notin V$. In this case, the canonical repair strategy performs an editing similar to the previous one, with the only difference that the forests encoded in the j-th and the $(j-1)$-th blocks of $\overleftarrow{\alpha}_{i+1}$ are gathered together under the same node labeled with $b_{i,j}$, and then highlighted by a node with a special label ERR. This trick is used, not because the tile $a_{i+1,j-1}$ is relevant for the violation itself, but because doing so will induce a shift in the tiling produced thereafter, which will propagate up to the root of the output tree, allowing in this way the validation by a target DTD. Formally,

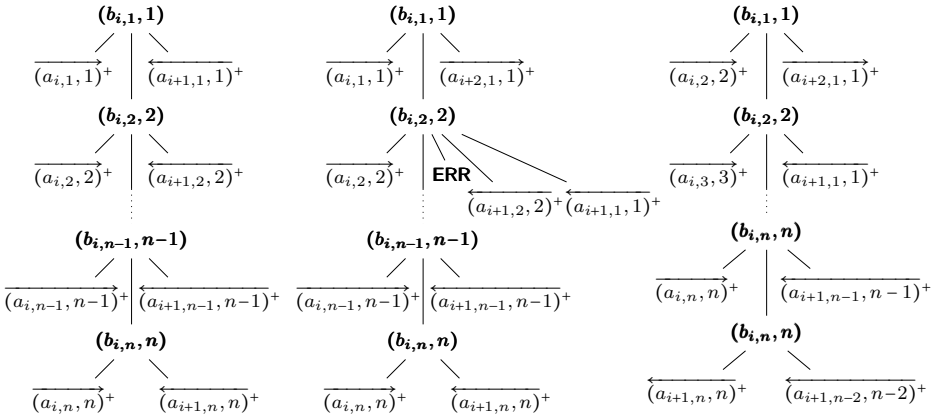


Fig. 9 Examples of subtrees that can arise from the canonical repair process.

the output produced in this case is:

$$\Big(\overleftarrow{(a_{i+1,n},n)}\,\overline{\overleftarrow{(a_{i+1,n},n)}}\Big)^+\,\overline{\boldsymbol{(b_{i,n},n)}}\ldots\Big(\overleftarrow{(a_{i+1,j+1},j+1)}\,\overline{\overleftarrow{(a_{i+1,j+1},j+1)}}\Big)^+\,\overline{\boldsymbol{(b_{i,j+1},j+1)}}$$

$$\textbf{ERR}\ \overline{\textbf{ERR}}\ \Big(\overleftarrow{(a_{i+1,j},j)}\,\overline{\overleftarrow{(a_{i+1,j},j)}}\Big)^+\Big(\overleftarrow{(a_{i+1,j-1},j-1)}\,\overline{\overleftarrow{(a_{i+1,j-1},j-1)}}\Big)^+\,\overline{\boldsymbol{(b_{i,j},j)}}\ \ldots$$

$$\Big(\overleftarrow{(a_{i+1,j-2},j-2)}\,\overline{\overleftarrow{(a_{i+1,j-2},j-2)}}\Big)^+\,\overline{\boldsymbol{(b_{i,j-1},j-1)}}\ \ldots\Big(\overleftarrow{(a_{i+1,1},1)}\,\overline{\overleftarrow{(a_{i+1,1},1)}}\Big)^+\,\overline{\boldsymbol{(b_{i,2},2)}}.$$

We observe that, differently from the previous case, the repaired string ends with the closing tag $\overline{(b_{i,2},2)}$. Accordingly, the remaining tag $\overline{(b_{i,1},1)}$ will be appended to the next incoming block, that is, $\Big(\overrightarrow{(a_{i+2,1},1)}\,\overline{\overrightarrow{(a_{i+2,1},1)}}\Big)^+$, if $i+1<2^n$, or to the last part of the input that consists of a repetition of #. It should be now clear that, when Adam produces an incorrect row, the canonical repair strategy has the possibility of hiding the repetition of # under a node and accordingly get into the target language.

For the sake of clarity, we describe in Figure 9 some subtrees that could possibly arise from the repair process. The left-hand side tree can be used before a violation of the constraints occurred, the tree in the middle can be used when the first violation of the constraints occurs – in this specific case, a violation on column 2 –, and the right-hand side tree is used after a violation has been exposed. Note that at this stage it is possible to check, at least locally, whether a violation has occurred or not; for instance, the middle tree violates the constraints because $(b_{i,2},a_{i+1,2})\notin V$ – this can be easily checked by a deterministic DTD.

On the grounds of the above explanations, it is natural to define the following DTD for the target language:

$$
\begin{array}{lll}
\mathcal{T}: & r_0 \to \overleftarrow{A}_{\perp} \cdot r_1 \cdot \overrightarrow{A}_{\#} & r_n \to \bigcup\limits_{b \in C} (b,1) \\
& r_0 \to \overleftarrow{A}_{\not\perp} \cdot r_1 \cdot \overrightarrow{A}_{\#} & (b,1) \to \mathsf{Left}(b,1) \cdot \Big(\bigcup\limits_{(b,b') \in H} (b',2) \Big) \cdot \mathsf{Right}(b,1) \\
& r_1 \to r_2\ r_2 & \vdots \\
& \quad \vdots & (b,n-1) \to \mathsf{Left}(b,n-1) \cdot \Big(\bigcup\limits_{(b,b') \in H} (b',n) \Big) \cdot \mathsf{Right}(b,n-1) \\
& r_{n-1} \to r_n\ r_n & (b,n) \to \mathsf{Left}(b,n) \cdot \mathsf{Right}(b,n)
\end{array}
$$

where

- $\overleftarrow{A}_{\perp}$ is the language: $\Big(\overleftarrow{(a_{\perp},n)}\Big)^{+} \cdots \Big(\overleftarrow{(a_{\perp},1)}\Big)^{+}$,
- $\overrightarrow{A}_{\#}$ is the language: $\Big(\bigcup\limits_{a \in C} \overrightarrow{(a,2)}^{+}\Big) \cdots \Big(\bigcup\limits_{a \in C} \overrightarrow{(a,n)}^{+}\Big) \cdot \#^{+}$,
- $\overleftarrow{A}_{\not\perp}$ is the language of the form:

$$\Bigg(\bigcup_{1 \le i \le n} \Big(\bigcup_{a \in C} \overleftarrow{(a,n)}^{+}\Big) \cdots \Big(\bigcup_{a \in C \setminus \{a_{\perp}\}} \overleftarrow{(a,i)}^{+}\Big) \cdots \Big(\bigcup_{a \in C} \overleftarrow{(a,1)}^{+}\Big)\Bigg) \cdot \Big(\bigcup_{a \in C} \overrightarrow{(a,1)}^{+}\Big),$$

- $\mathsf{Left}(b,j)$, for $1 \le j < n$, is the language: $\Big(\bigcup\limits_{(a,b) \in V} \overrightarrow{(a,j)}^{+}\Big) \ \cup \ \Big(\bigcup\limits_{a \in C} \overrightarrow{(a,j+1)}^{+}\Big)$,
- $\mathsf{Left}(b,j)$, for $j = n$, is the language : $\Big(\bigcup\limits_{(a,b) \in V} \overrightarrow{(a,n)}^{+}\Big) \ \cup \ \Big(\bigcup\limits_{a \in C} \overleftarrow{(a,n)}^{+}\Big)$,
- $\mathsf{Right}(b,j)$, for $1 < j \le n$, is the language:

$$\Big(\bigcup_{(b,a) \in V} \overleftarrow{(a,j)}^{+}\Big) \ \cup \ \Big(\mathsf{ERR} \cdot \bigcup_{(b,a) \notin V} \overleftarrow{(a,j)}^{+} \cdot \bigcup_{a \in C} \overleftarrow{(a,j-1)}^{+}\Big) \ \cup \ \Big(\bigcup_{a \in C} \overleftarrow{(a,j-1)}^{+}\Big),$$

- $\mathsf{Right}(b,j)$, for $j = 1$, is the language:

$$\Big(\bigcup_{(b,a) \in V} \overleftarrow{(a,1)}^{+}\Big) \ \cup \ \Big(\mathsf{ERR} \cdot \bigcup_{(b,a) \notin V} \overleftarrow{(a,1)}^{+} \cdot \bigcup_{a \in C} \overrightarrow{(a,1)}^{+}\Big) \ \cup \ \Big(\bigcup_{a \in C} \overrightarrow{(a,1)}^{+}\Big).$$

Note that the above languages can be defined by DFAs of polynomial size, and the target DTD can be produced in polynomial time.

We already described a canonical repair strategy that, under the assumption that Eve wins the tiling game, transforms any tree from the restriction language into a tree of the target language with a uniformly bounded cost. It remains to prove the converse, that is, if there is a repair processor from $\mathcal{R}$ to $\mathcal{T}$ with uniformly bounded cost, then Eve wins the tiling game. To prove this, we first recall that the restriction language $\mathcal{R}$ contains all trees that encode lists of 2^n rows that satisfy the horizontal constraints and that can be chosen by Adam, independently of the possible responses by Eve (of course, the rows chosen by Eve are not encoded inside the restriction language). Thus, as far as we are concerned with the sequence of rows that Adam can choose during a play of the tiling game, it suffices to look at the trees in $\mathcal{R}$. In particular, from any prefix of a serialized tree in $\mathcal{R}$, we can easily read off a sequence of rows, which we assume can be played by Adam.

Now, suppose that there is a tree-edit transducer that receives the serializations of the trees in $\mathcal{R}$ and transforms them, with uniformly bounded cost, to some trees in the target language $\mathcal{T}$. Recall that, within the trees of $\mathcal{R}$, each tile can be encoded with an arbitrary amount of redundancy. It is thus impossible for the bounded cost transducer to erase or replace even a single tile in a row produced by Adam. Moreover, because the repairs are implemented by a *tree edit* transducer, the occurrence order of the tiles produced by Adam is preserved during the repair. Similarly, it is also impossible for the transducer to forge new rows and pretend they were played by Adam, as this anomaly is easily detected in the target language by an increase of the number of rows. This means that the choices of Adam are still correctly represented in the edited serializations of the input trees. Moreover, in the edited serializations, the rows produced of Adam that occur before the ERR tag, are interleaved with other rows. We can use the latter rows as responses of Eve to Adam moves. Formally, by looking at the edited serializations and by exploiting a simple induction based on the number of rounds in the tiling game, we can construct a strategy for Eve that associates with each partial play ending with a row produced by Adam a corresponding row that should be chosen by Eve. We claim that the defined strategy for Eve is correct, in the sense that, no matter how Adam plays, Eve will respond with an appropriate row that satisfies the constraints of the tiling game. Indeed, we know that the edited serializations belong to the target language $\mathcal{T}$, and hence every row that is inserted by the transducer (which can be then chosen by Eve) must satisfy the horizontal constraints and the vertical constraints with respect to the underlying row produced by Adam. Moreover, we recall that all edited serializations contain an occurrence of the ERR tag. This means that, along any play induced by Eve's strategy, Adam eventually violates the vertical constraints. By contraposition, this shows that Eve wins the tiling game if Adam never violates the constraints. □

We now exhibit a sub-class of tree automata recognizing restriction languages for which the streaming bounded repairability problem becomes easier to solve, namely, PSPACE-complete. The sub-class is obtained by restricting the accessibility graph of the strongly connected components of $\mathcal{R}$ so as to take the shape of a tree. 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 $\mathrm{dag}(\mathcal{R})$. We say that $\mathcal{R}$ is *tree-shaped* if $\mathrm{dag}(\mathcal{R})$ 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 $\mathcal{D}$ is tree-shaped if its language is recognized by a tree-shaped automaton of size linear in $\mathcal{D}$.

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 deterministic tree automata is in* PSPACE.

Proof. A proof of the PSPACE upper bound is as follows. From the fact that the restriction automaton is tree-shaped, we first derive a polynomial bound on the length of the possible plays over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. This bound follows easily from the fact

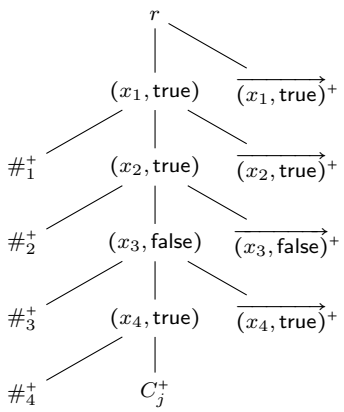


Fig. 10 Tree encoding of a valuation.

that every move of Generator induces a change of component in $\mathcal{R}$, precisely, it is of the form

$$[\![X \cdot \vec{x}, \vec{y}]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle X_1\, X_2 \cdot \vec{x}, \vec{y} \rangle\!\rangle$$

where the two components X_1 and X_2 are different from X and belong to disjoint sub-trees of $\mathrm{dag}(\mathcal{R})$ strictly below X. In particular, this implies that the number of moves that can be subsequently chosen by Generator never exceeds the number of components of $\mathcal{R}$. Based on this bound, we can simulate the possible plays over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ by means of an alternating polynomial-time procedure. This shows that the considered repairability problem is in PSPACE. □

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

Proof. We give a reduction from the problem of deciding validity of a quantified boolean sentence of the form

$$\phi \;=\; \forall x_1\, \exists y_1\, \ldots\, \forall x_n\, \exists y_n \quad \psi(x_1, y_1, \ldots, x_n, y_n)$$

where $\psi(x_1, y_1, \ldots, x_n, y_n)$ is in conjunctive normal form. More precisely, we will construct two tree-shaped DTDs $\mathcal{R}$ and $\mathcal{T}$ such that ϕ is satisfiable if and only if $\mathcal{R}$ is streaming bounded repairable into $\mathcal{T}$. Some of the ingredients of this proof will be similar to the reduction from the tiling game given in Theorem 3. Let ϕ be a quantified boolean sentence such as the above one and let $C_1, \cdots, C_k$ be the clauses (i.e. disjunctions of literals) in ψ. Without loss of generality, we can assume that every clause contains a fixed number of literals, say 3 (the validity problem in this case is still PSPACE-hard).

The restriction DTD $\mathcal{R}$ will produce all possible redundant encodings of valuations for the variables $x_1, \ldots, x_n$, followed by a clause of ψ that needs to be satisfied. Figure 10 shows an example of the encoding of a valuation for four variables $x_1, \ldots, x_4$. The variables with their valuations are listed from top to bottom along the middle spine and repeated several times to the right. The $\#_i$-labeled nodes to the left of the spine will be used to force the repair processor to guess a valuation for the variable y_i, before having seen the valuation of the next variable

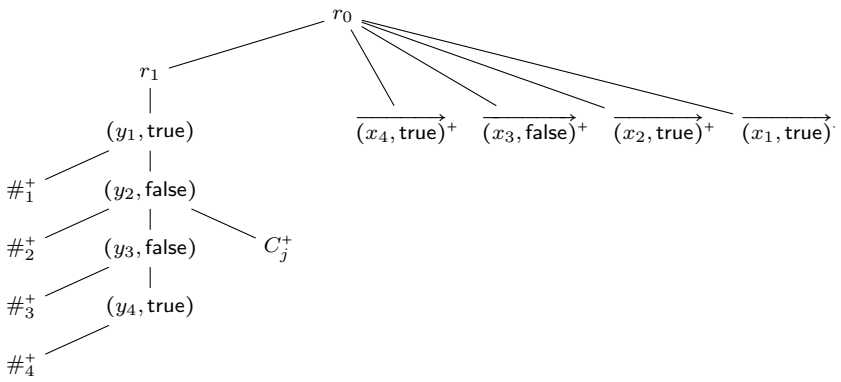


Fig. 11 Possible outcome of a repair strategy.

x_{i+1}. The repeated C_j-labeled nodes at the bottom of the spine designate a clause C_j to be satisfied.

Formally, the DTD $\mathcal{R}$ defining the restriction language is given as follow (ν denotes a boolean value among $\mathsf{true}, \mathsf{false}$):

$$
\begin{aligned}
\mathcal{R}\ :\qquad r &\to \bigcup_{\nu \in \{\mathsf{true},\mathsf{false}\}} (x_1, \nu) \cdot \overrightarrow{(x_1, \nu)}^+ \\
(x_1, \nu) &\to \bigcup_{\nu' \in \{\mathsf{true},\mathsf{false}\}} \#_1^+ \cdot (x_2, \nu') \cdot \overrightarrow{(x_2, \nu')}^+ \\
&\vdots \\
(x_{n-1}, \nu) &\to \bigcup_{\nu' \in \{\mathsf{true},\mathsf{false}\}} \#_{n-1}^+ \cdot (x_n, \nu') \cdot \overrightarrow{(x_n, \nu')}^+ \\
(x_n, \nu) &\to \bigcup_{1 \le j \le k} \#_n^+ \cdot C_j^+
\end{aligned}
$$

Note that the above DTD is in a tree-shaped form.

Symmetrically, the target language will require some valuations $\mu_1, \ldots, \mu_n$ for the variables $y_1, \ldots, y_n$ to be emitted by the repair processor, in such a way that all clauses of ϕ are satisfied. The target will also require the repair processor to produce a "proof" of satisfiability for the clause that is designated by the restriction. More precisely, a canonical repair strategy will modify the input tree by first removing the spine of nodes labeled by (x_i, ν_i), flattening in this way the tree. While doing that, the repair process will also provide some valuations $\mu_1, \ldots, \mu_n$ for the remaining variables $y_1, \ldots, y_n$. The order in which the valuations ν_i are consumed and the valuations μ_i are produced must reflect the alternation of the universal and existential quantifiers on the variables $x_1, y_1, \ldots, x_n, y_n$. In particular, the encoding of each valuation μ_i needs to be produced when the corresponding valuation ν_i is consumed, and before seeing the next valuation ν_{i+1}, if $i < n$, or the clause C_j. Formally, this policy is enforced by requiring the repair processor to emit an opening tag of the form (y_i, μ_i) before any occurrence of the opening tag $\#_i$.

Once all valuations are fixed, the goal of the repair processor is to move the forest C_j^+ that comes from the input stream to some ancestor node. We explain

how this goal can be accomplished, assuming that all clauses are satisfied by the compound valuation. If the designated clause C_j is satisfied by a valuation $y_i = \mu_i$, for some $1 \leq i \leq n$, then the canonical repair will position the forest C_j^+ just under the node labeled with (y_i, μ_i). This will ease the validation of the clause by the target DTD. For instance, if C_j is some clause of ϕ satisfied by the valuation $y_2 = \mathsf{false}$, then a possible outcome of the repair strategy is shown in Figure 11. Otherwise, the clause C_j is satisfied by some evaluation $x_i = \nu_i$, and in this case the canonical repair will promote the forest C_j^+, below the root and just before the occurrences of the nodes $\overrightarrow{(x_n, \nu_n)}$, ..., $\overrightarrow{(x_1, \nu_1)}$. Like before, this transformation will ease the validation of the clause by the target DTD. Formally, the DTD $\mathcal{T}$ for the target language is define as follows:

$$
\begin{aligned}
\mathcal{T}: \qquad r_0 &\to r_1 \cdot \big(\mathsf{SomeX} \cup \mathsf{OK}\big) \\
r_1 &\to \bigcup_{\nu \in \{\mathsf{true},\mathsf{false}\}} (y_1, \nu) \cdot \mathsf{SomeY}(y_1, \nu) \\
(y_1, \nu) &\to \bigcup_{\nu' \in \{\mathsf{true},\mathsf{false}\}} \#_1^+ \cdot (y_2, \nu') \cdot \mathsf{SomeY}(y_2, \nu') \\
&\vdots \\
(y_{n-1}, \nu) &\to \bigcup_{\nu' \in \{\mathsf{true},\mathsf{false}\}} \#_{n-1}^+ \cdot (y_n, \nu') \cdot \mathsf{SomeY}(y_n, \nu') \\
(y_n, \nu) &\to \#_n^+
\end{aligned}
$$

where

- SomeX is the language that consists of all strings of the form $C_j^+ \cdot \overrightarrow{(x_n, \nu_n)}^+ \ldots \overrightarrow{(x_1, \nu_1)}^+$, where $\nu_1, \ldots, \nu_n \in \{\mathsf{true}, \mathsf{false}\}$ and C_j is a clause satisfied under the valuation $x_1 = \nu_1$, ..., $x_n = \nu_n$, (recall that the number of literals in each clause is fixed, so the language SomeX is recognized by a regular expression of size polynomial in the size of ϕ)
- OK is the language that consists of all strings of the form $\overrightarrow{(x_n, \nu_n)}^+ \ldots \overrightarrow{(x_1, \nu_1)}^+$, with $\nu_1, \ldots, \nu_n \in \{\mathsf{true}, \mathsf{false}\}$,
- $\mathsf{SomeY}(y_i, \nu)$ is the union of the languages C_j^+, for all clauses C_j that are satisfied under the valuation $y_i = \nu$.

We observe that the above DTD is in a tree-shaped form. We have already proved that, when the quantified boolean sentence ϕ is valid, there exists a canonical repair that transforms any tree from the restriction language to a tree in the target language using a bounded number of edit operations. Conversely, if there exists a tree-edit transducer that repairs every tree in $\mathcal{R}$ to a tree in $\mathcal{T}$ with uniformly bounded cost, then one can easily construct from the outputs of this transducer some functions $f_i(\nu_1, \ldots, \nu_i)$ that associate with each partial sequence of valuations $\nu_1, \ldots, \nu_i$ a corresponding valuation μ_i for the variable y_i in such a way that every clause C_j is satisfied by the compound valuation $\nu_1, f(\nu_1), \ldots, \nu_n, f(\nu_1, \ldots, \nu_n)$. This proves that ϕ is valid whenever $\mathcal{R}$ is streaming bounded repairable into $\mathcal{T}$.

□

We conclude the section by recalling an interesting result from [18] that concerns a specific case of the streaming bounded repairability problem. Specifically,

it follows from Proposition 6 and Proposition 7 in [18] that the complexity of the streaming bounded repairability problem drops to *polynomial time* when the restriction language contains all trees over a given alphabet Σ.

7 Conclusions

In this paper, we studied the streaming bounded repair problem over trees. Our main result is a characterization of this problem in terms of a two-stack game between Generator and Repairer. This game characterization includes and generalizes most of the concepts introduced in previous papers [5,6,18]. For example, trees in the restriction and target languages are abstracted by using strongly connected components [18] and suitable games are used to characterize the existence of streaming bounded repair strategies [6]. The game that characterizes streaming bounded repairability is basically a reachability game over a system of prefix-rewriting rules applicable to two distinct stacks: one stack is associated with the restriction language and controlled by Generator, the other stack is associated with the target language and controlled by Repairer. By exploiting the fact that the stack controlled by Generator has bounded height, one could decide which of the two players wins the reachability game. We use this fact to derive an alternating polynomial-space (i.e. Exp) algorithm that solves the streaming bounded repair problem and we show that the considered problem is indeed Exp-hard.

A weakness of our technique lies in the fact that we heavily depend on the top-down determinism of the two schemas: in the presence of languages represented by arbitrary tree automata decidability of the streaming bounded repair problem is still open. A better understanding of the cyclic behaviour of tree automata is thus needed in order to effectively characterize streaming bounded repairability in the general case. Along this direction, we would like to highlight Proposition 6 from [18] which shows that the streaming bounded repair problem is decidable when the restriction language is universal (i.e. unrestricted).

Our characterization shows that when Repairer has a strategy to win the reachability game, one can effectively extract from it a streaming repair strategy as a cascade composition of transducers $\mathcal{Z}_1$, $\mathcal{Z}_2$, $\mathcal{Z}_3$, and $\mathcal{Z}_4$ (see Section 4). One can notice that, in the cascade composition of these transducers, only two stacks are needed in order to produce the appropriate repairs: the two stacks are used essentially to simulate the computation of the restriction and target automata on the input tree and on the edited tree, respectively. An interesting open question is to determine whether two stacks are really needed or whether a repair strategy can be always implemented by a single-stack process. Furthermore, we do not know the exact complexity of determining an optimal streaming repair strategy, where optimality is measured in terms of maximal number of edits. In this respect, we expect that further simplifications in our approach could lead to a better understanding of the problem and, eventually, to a complexity analysis of the problem of determining the cost of an optimal repair strategy.

We also leave open some complexity gaps when the restriction and target languages are given by non-deterministic schemas. For example, we did not analyzed in detail the case where the restriction and target languages are given by general DTDs. We recall here that a DTD schema can be seen as a deterministic top-down tree automaton only after determinizing each regular expression on the head of

each rule of the DTD. From this observation and by applying our decision procedure to a "determinized" DTD, we easily derive a double exponential algorithm that decides streaming bounded repairability for general DTDs. Unfortunately, Exp-hardness is the best lower bound that one can get from our results, which is still far from the 2Exp upper bound that we just derived. We also recall that similar gaps were left open in the string case [6]. We think that new insights are required to close these complexity gaps.

Finally, our work highlights the issue of a proper notion of edit processor for trees that have a canonical serialization as a string, as is the case of XML documents. Example 4 (see also [1]) shows that the ability of editing 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 input serializations, but for which there exist no bounded repair processor of any sort (even non-streaming) that repairs using only tree edits. We do not know if this last phenomenon can occur in the presence of more limited schemas, such as DTDs.

Acknowledgements 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). The last author was also supported by CONICYT + PAI / Concurso Nacional Apoyo al Retorno de Investigadores/as desde el extranjero – Convocatoria 2013 + 821320001.

References

1. Akutsu, T.: A relation between edit distance for ordered trees and edit distance for euler strings. Information Processing Letters **100**(3), 105–109 (2006)
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journal of the ACM **56**(3), 16 (2009)
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT press Cambridge (2008)
4. Benedikt, M., Puppis, G., Riveros, C.: The cost of traveling between languages. In: Automata, Languages and Programming (ICALP), pp. 234–245 (2011)
5. Benedikt, M., Puppis, G., Riveros, C.: Regular repair of specifications. In: Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 335–344 (2011)
6. Benedikt, M., Puppis, G., Riveros, C.: Bounded repairability of word languages. Journal of Computer and System Sciences (JCSS) **79**(8), 1302–1321 (2013)
7. Bille, P.: A survey on tree edit distance and related problems. Theoretical Computer Science (TCS) **337**(1), 217–239 (2005)
8. Boas, P.V.E.: The convenience of tilings. In: Complexity, Logic, and Recursion Theory, pp. 331–363 (1997)
9. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. Information and Computation **142**(2), 182–206 (1998)
10. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007)
11. Cristau, J., Löding, C., Thomas, W.: Deterministic automata on unranked trees. In: Fundamentals of Computation Theory, pp. 68–79 (2005)
12. Gauwin, O., Niehren, J., Roos, Y.: Streaming tree automata. Information Processing Letters **109**(1), 13–17 (2008)
13. Grädel, E., Thomas, W., Wilke, T.: Automata, logics, and infinite games: a guide to current research, vol. 2500. Springer (2003)
14. Kumar, V., Madhusudan, P., Viswanathan, M.: Visibly pushdown automata for streaming XML. In: Proceedings of the 16th International Conference on World Wide Web (WWW), pp. 1053–1062 (2007)

15. Martens, W., Neven, F., Schwentick, T.: Deterministic top-down tree automata: past, present, and future. In: Logic and Automata: History and Perspectives, *Texts in Logic and Games*, vol. 2, pp. 505–530 (2008)
16. Martens, W., Neven, F., Schwentick, T., Bex, G.J.: Expressiveness and complexity of XML schema. ACM Transactions on Database Systems (TODS) **31**(3), 770–813 (2006)
17. Murata, M., Lee, D., Mani, M., Kawaguchi, K.: Taxonomy of XML schema languages using formal language theory. ACM Transactions on Internet Technology (TOIT) **5**(4), 660–704 (2005)
18. Puppis, G., Riveros, C., Staworko, S.: Bounded repairability for regular tree languages. In: Proceedings of the 15th International Conference on Database Theory (ICDT), pp. 155–168 (2012)
19. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: Proceedings of the 21th ACM SIGMOD Symposium on Principles of Database Systems (PODS), pp. 53–64 (2002)
20. Tai, K.C.: The tree-to-tree correction problem. Journal of the ACM (JACM) **26**(3), 422–433 (1979)
21. Wagner, R., Fischer, M.: The string-to-string correction problem. Journal of the ACM (JACM) **21**(1), 168–173 (1974)