Repairing strings and trees

Cristian Riveros
Wolfson College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity Term 2013

Dedicated to Flavia
and the new members of our family

Abstract

What do you do if a computational object fails a specification? An obvious approach is to *repair* it, namely, to modify the object minimally to get something that satisfies the constraints. In this thesis we study foundational problems of repairing regular specifications over strings and trees. Given two regular specifications R and T we aim to understand how difficult it is to transform an object satisfying R into an object satisfying T. The setting is motivated by considering R to be a *restriction* – a constraint that the input object is guaranteed to satisfy – while T is a *target* – a constraint that we want to enforce.

We first study which pairs of restriction and target specifications can be repaired with a "small" numbers of changes. We formalize this as the *bounded repair problem* – to determine whether one can repair each object satisfying R into T with a uniform number of edits. We provide effective characterizations of the bounded repair problem for regular specifications over strings and trees. These characterizations are based on a good understanding of the cyclic behaviour of finite automata. By exploiting these characterizations, we give optimal algorithms to decide whether two specifications are bounded repairable or not.

We also consider the impact of limitations on the editing process – what happens when we require the repair to be done sequentially over serialized objects. We study the bounded repair problem over strings and trees restricted to this streaming setting and show that this variant can be characterized in terms of finite games. Furthermore, we use this characterization to decide whether one can repair a pair of specifications in a streaming fashion with bounded cost and how to obtain a streaming repair strategy in this case.

The previous notion asks for a uniform bound on the number of edits, but having this property is a strong requirement. To overcome this limitation, we study how to calculate the maximum number of edits per character needed to repair any object in R into T. We formalize this as the *asymptotic cost* – the limit of the number of edits divided by the length of the input in the worst case. Our contribution is an algorithm to compute the asymptotic cost for any pair of regular specifications over strings. We also consider the streaming variant of this cost and we show how to compute it by reducing this problem to mean-payoff games.

Acknowledgements

I would like to thank my supervisor Michael Benedikt for his guidance and support during my studies in Oxford, and for being always willing to listen all my crazy ideas. I would also like to thank Gabriele Puppis for the exciting years we spent discussing our research.

This thesis would not have been possible without the financial support from Becas Chile and from the EPSRC project "Enforcement of Constraints on XML streams" (EPSRC EP/G004021/1). Additional funding was provided by the School of Engineering at Pontificia Universidad Católica de Chile.

As always, I am also thankful for my family and friends who have supported me over many years. I would like to specially thank the "Oxford Chilean Gang". They have been one of the highlights during my stay at Oxford.

Finally, I will always be in debt with Flavia, my wife and girlfriend. Without her nothing of this would be possible.

Contents

List of Figures

Chapter 1

Introduction

A basic problem in data management is to ensure that data is valid – namely, satisfies all integrity constraints associated with a schema. The validity of data with respect to a schema is crucial in any database system; if data does not satisfy all integrity constraints, then a system cannot guarantee that the expected output is correct. Nevertheless, there are many practical scenarios where data management systems have to deal with invalid data. When data does not satisfy constraints an obvious approach is to *repair* it. That is, to modify it minimally so that it becomes valid. We may want to perform this transformation on the data, or we may be merely interested in knowing how difficult it would be to perform – that is, determining how far a given collection of data is from satisfying the specification. For example, in some applications one may accept some data as valid even if it only contains a few errors where "few" could be defined by a user-threshold or by a percentage of the number of errors with respect to the input size [GT04].

On relational data, this problem has been extensively studied under the notion of *constraint repair* (see e.g. [ABC99, AK09]): in this case the specifications considered are relational integrity constraints, such as keys and foreign keys, and the problems considered include determining how much a database needs to be modified in order to satisfy a given constraint. This approach to data integrity has been investigated for a variety of integrity constraints – beginning with classical functional and inclusion dependencies [ABC99], and continuing to wider classes such as tuple generating dependencies [AK09]. Also, a number of different modification operators have been considered in the relational case, including inserting, changing, and deleting tuples. In addition to finding repairs of relations, much of this line of research deals with querying inconsistent documents via their repairs.

For XML-based applications, the repair approach is even more natural and common than the relational case. The most obvious example of this is HTML. Malformed or non-conformant HTML is more the rule than the exception, and browsers react to invalid documents by simply changing them to conformant ones. That is, repair is a well-accepted procedure for dealing with invalid XML-based data. One could also argue that the tree-structure of an XML document makes them more natural

for repairing. It is more natural to insert or remove nodes in a tree rather than tuple or columns in a relational structure.

Despite the fact that XML data seems more suitable for repairing, this problem has been studied less extensively. In [FFGZ05], repairs are considered for an extension of classical relational constraints to XML such as inclusion dependencies, while in [SC06] repairs of a document with respect to an XML schema are studied, with an emphasis on consistent querying over such documents. Quite a different line of work deals with editing of *schemas*, rather than documents. For example, [FB08] deals with a similarity measure on schemas given by considering *embeddings* that preserve the DTD structure. The notion of similarity thus depends on the syntactic presentation of the schema, not the language of documents that it defines.

The notion of repairing strings or trees (an abstraction for XML documents) is indeed more obvious than in the case of a relational databases: for example, we can simply consider the *edit distance* between strings, a standard measure of how many basic edit operations it takes to get from one string to the next (note that the edit distance between strings can be naturally generalized to trees). The edit operations here consist of inserting, deleting, or modifying letters or nodes. Edit distance can even be lifted in a natural way to give a measure $\mathsf{dist}(w, L)$ of the distance of a string w to a language (collection of strings) L: the minimal distance of w to any string in L. Furthermore, $\mathsf{dist}(w, L)$ can be computed efficiently when L is a regular language given by any standard regular specification (e.g. NFA) [Wag74]. In this setting, w can be seen as the data and L as the specification. Thus, $\mathsf{dist}(w, L)$ is measuring how difficult is to repair the data w in order to satisfy L.

In this thesis we take the next step in repairing – given two regular specifications R and T over strings or trees we aim to calculate how difficult it is to transform an object satisfying R into an object satisfying T. The notation is motivated by considering R to be a *restriction* — a constraint that the input is guaranteed to satisfy – while T is a *target* – a constraint that we want to enforce. We consider the worst-case over all object w satisfying R (denoted by $w \in R$) of the number of edit operations needed to move w into T: $\sup_{w \in R} \mathsf{dist}(w, T)$. That is, we look at the worst-case number of operations needed to get from R to T. Of course, this number may be infinite. The main purpose of this thesis is to study different problems related to the finiteness or asymptotic growth of this number. Specifically, these problems are the *bounded repair problem* and the *asymptotic cost*.

Bounded repair problem. The first question that we study is whether the number of edits needed for repairing every object satisfying R into T is bounded, that is, whether the number $\sup_{w \in R} \mathsf{dist}(w, T)$ is finite. Intuitively, a repair process is any procedure that, giving any $w \in R$, inserts, deletes, or modifies w producing an output that satisfies the constraints of T. Clearly, there is a vacuous repair process that simply deletes the data and inserts new data that satisfies the target constraints. The unacceptability of such a repair strategy stems from the fact that the number of changes it makes to the input is proportional to its size. Clearly, we would like a repair processor

that makes a 'small' number of changes to the input. We formalize this requirement via the notion of a *bounded repair process*, that is, a repair process that makes a maximum number of repairs that is finite and independent of the input data.

Example 1. *Consider the regular expressions:*

$$\begin{aligned} R &= \mathrm{P}^+ \cdot \mathrm{S}^+ \\ T &= \mathrm{P}^+ \cdot \mathrm{D} \cdot \mathrm{S}^+. \end{aligned}$$

Informally, R defines the set of documents that starts with a sequence of products P *followed by a sequence of sales* S*. Instead, documents satisfying T contain the same two sequences with an extra tag* D *(for details) between both products and sales. Clearly, any document satisfying R can be repaired to a document that satisfies T with at most* 1 *edit – by adding a* D *one can convert each document from R into T.*

In the previous example, we say that the restriction specification R is *bounded repairable* into the target specification T. That is, there exists a fix number of edit operations needed to transform every document satisfying R into T. Furthermore, no matter how big is the input document $w \in R$, one can make that w satisfies T with just one edit, that is, $\sup_{w \in R} \mathsf{dist}(w, T)$ is finite and equal to 1.

We argue that the notion of bounded repairability is subtle and fundamental in any study of data repair. Intuitively, a specification R is "almost included" in a specification T whenever R is bounded repairable into T. In this sense, bounded repairability is a natural generalization of containment between specifications where containment measures whether one specification is "more general" than another. It is well-known that containment is one of the basic notions in computer science used in order to compare specifications, queries, or even systems.

Asymptotic cost. The previous notion ask for the finiteness of the number of edits $\sup_{w \in R} \mathsf{dist}(w, T)$. However, having a uniform bound on the number of edits is a strong requirement. Instead, we look not at the absolute number of edits required to get from R to T, but rather at the *percentage* of repair operations needed. That is, we would like to measure how grows the value $\mathsf{dist}(w, T)$ whenever the size of $w \in R$ tends to infinite.

Example 2. *Consider the regular expressions:*

$$\begin{aligned} R' &= \big(\mathrm{P} \cdot \mathrm{S}\big)^+ \\ T' &= \big(\mathrm{P} \cdot \mathrm{S} \cdot \mathrm{D}\big)^+. \end{aligned}$$

Here, R' is specifying that every product P *has to be paired with a sale* S*. Instead, T requires that each sequence of product and sale must also be followed by details* D*. Roughly, for any pair of consecutive occurrences of* P *and* S *in a document satisfying R, we will have to perform one edit in order to ensure that a* D *follows after each subsequence* $\mathrm{P} \cdot \mathrm{S}$*. In particular, the number of edits required to get from a string in R to a string in T is unbounded. On the other hand, it is clear that*

we need to edit approximately half of the document in the worst case in order to produce a document in T (i.e. we need to add $\frac{n}{2}$ many D *for any document $w \in R$ of size n).*

We measure the gap from R to T via the worst case, over all documents w satisfying R, of the number of edits needed to bring w into T divided by the length of w. Since we want the definition to be robust to a finite number of outliers, we take the limit of this quantity as the strings are of larger and larger length – this is the *asymptotic (normalized) cost* in getting from R to T. Intuitively, this gives us a measure of the distortion needed to get from R to T, lying always between 0 and 1. In the previous example, we argued that the asymptotic cost from R into T is 1/2.

Streaming repair. Above we considered the use of repair processes that can read the whole document in memory. In this work, we also consider the impact of limitations on the editing process – what happens when we require the editing to be done by a transducer, reading the input letter-by-letter and producing the corrected output, based only on a finite amount of control state and a fixed amount of lookahead in the string or (serialized) tree. Intuitively, a streaming repair process is a procedure that inserts, deletes, or modifies documents while reading each document in pre-order fashion, producing an output that satisfies the target constraints. We study the bounded repair problem and asymptotic cost of repairing strings or trees when we are only allowed to repair our data by using streaming repair process. One can easily see that the answer to our main questions changes when we restrict to this streaming setting.

Example 3. *Consider the regular expressions:*

$$\begin{aligned} R'' &= \left(\mathrm{P}_1 + \mathrm{P}_2\right) \cdot \mathrm{D}^+ \cdot \left(\mathrm{S}_1^+ + \mathrm{S}_2^+\right) \\ T'' &= \left(\mathrm{P}_1 \cdot \mathrm{D}^+ \cdot \mathrm{S}_1^+\right) + \left(\mathrm{P}_2 \cdot \mathrm{D}^+ \cdot \mathrm{S}_2^+\right). \end{aligned}$$

Documents in R'' starts with a product of type P_1 *or* P_2*, followed by a sequence of details* D*, and ending with a sequence of sales of type* S_1 *or* S_2*. Instead, the target specification matches sequences starting with a product of type* P_1 *(*P_2*) with sales of type* S_1 *(*S_2 *resp.). In this example, one can easily get from R'' to T'' by only editing the initial letter in order to match the type of the initial product with the type of the ending sales. That is, the cost of repairing any document from R'' into T'' is equal to* 1*. However, a streaming repair process must commit to changing the initial letter or leaving it be. If it makes the "incorrect" choice, it will have to edit an unbounded final segment. Intuitively, any streaming repair process will incur in an unbounded cost for repairing documents from R'' into T''.*

As the previous example shows, restricting the access of the repair process to the input could vary the final repair cost. In this chapter, we study the bounded repair problem and asymptotic cost also in this streaming setting. Moreover, we study this problem for streaming XML documents.

Strings and trees. The previous notions of bounded repairability and asymptotic cost were presented with examples using strings. We would like to emphasize that in this work we consider

these problems also over trees. Here, R and T are specifications over trees and we want to study whether (1) R is repairable with an uniformly bounded number of edits into T or, if not, (2) computing the asymptotic cost for repairing from R into T. Unfortunately, we study (2) only over strings. In this thesis, we left the problem of computing the asymptotic cost between trees specifications for future work (see the description of the chapters below).

The purpose of studying repair problems over strings and trees is to better understand the repair of XML-based data as it was motivated before. Here, we only concentrate on studying the repair of the structural part of XML documents or, in simple words, the structure of nodes given by open and close tags. In this work we do not care of the data (e.g. PCDATA) of an XML document and we only restrict to the study of its structure. For the sake of this goal, we concentrate on strings and trees specifications contained in the set of *regular languages* (i.e. the ones define in terms of finite state automata).

Although we have motivated our work with a database perspective, it is important to note that the results in this thesis have also implications in other areas like verification of formal systems. For example, a repair approach can also be applied to traces of a system where the restriction language is defined by the potential behaviour of a program and the target language is a linear temporal formula which specifies the constraints that the program must satisfy. A streaming process here can be thought as an external "rectification" process that dynamically modifies the behaviour of the program whenever is necessary.

In the following, we give a complete overview of the contributions of this thesis.

Chapter 2: Bounded repairing of strings. We begin by studying the bounded repair problem over strings. The core of our results is an effective characterization for the *bounded repair problem*, namely, to determine whether there exists an uniform bound in the number of edits needed to repair every string in R into T for any pair of regular languages R and T given by deterministic and non-deterministic finite state automata.

Our characterization is based on a good understanding of the cyclic behaviour of finite state automata. Roughly, we decompose the structure of finite automata into strongly connected components and we characterize bounded repairability by a matching relation between components of the restriction and target language. We exploit the above characterization to give a precise map of the complexity of the bounded repair problem given by different regular specifications (e.g. deterministic and non-deterministic finite automata). Unfortunately, we show that even for languages given by deterministic finite automata the bounded repair problem is intractable, namely, coNP-complete.

In Chapter 2 we also study the bounded repair problem in the streaming setting. We provide a game-based characterization to determine which pairs of regular languages admit streaming bounded repairing. This result is a natural extension of the non-streaming characterization in terms of

games. We use this characterization to isolate the complexity of the streaming repair problem for any lookahead and for any representation of the languages considered in the non-streaming case. Our main algorithmic result shows that is decidable in polynomial time whether one can streaming repair each string in R into T. Moreover, one can derive from this result a streaming repair process effectively. Interestingly, this contrasts with the intractability of the non-streaming setting pointed above.

Towards the end of this chapter we show that streaming and non-streaming repair problems have very different flavors: the former are closely related to games played on the components of two automata, while the latter require a more global analysis, and exhibit a close relation to *distance automata*. Nevertheless, there are connections between these different problems: we show that in the case where there is no restriction, the bounded repair problems are the same for both streaming and non-streaming setting. We also show that the bounded repair problem in the streaming setting is independent of the lookahead, and is robust under plausible alternative definitions.

Most of the results presented in this chapter are straightforward extensions of the results previously presented in [BPR11b, BPRa].

Chapter 3: Asymptotic repairing of strings. Next, we move from the bounded repair problem to study how to compute the asymptotic cost of regular languages. The main result of Chapter 3 is an algorithm that computes the asymptotic cost of repairing R into T for any pair of regular languages R and T. This result shows that the asymptotic cost is always rational and, furthermore, it can be computed in double exponential time in the size of R and T. The techniques used for the asymptotic cost analysis relies on the connection of this problem with the theory of *distance automata* [Sim88], and in particular on an application of determinization of distance automata, closely related to Mohri's determinization procedure [Moh97]. Similar to the bounded repair problem, we decompose the restriction and target languages into strongly connected component and we derive a determinizable distance automaton for each component. By applying a complex construction over this set of distance automata, we reduce our problem to an optimization problem which is solvable in double exponential time.

Similarly to the bounded repair problem, we also study the asymptotic repair cost in the streaming setting. We measure a streaming repair processor by the number of edits per character it requires to get from any string in R to a string in T, again looking at the limit as the string length gets large. We accordingly define the *streaming asymptotic cost* to be the optimal cost of a streaming processor. Interestingly, we show that this quantity can also be calculated effectively, using techniques from mean-payoff games. By using these techniques, we show that the streaming asymptotic cost is always rational and can be compute in polynomial time. This radically contrasts with the non-streaming scenario where our best algorithm runs in double exponential time.

All the results shown in this chapter were previously presented in [BPR11a, BPRb].

Chapter 4: Bounded repairing of trees. We turn our attention from repairing strings to repairing trees. Namely, we consider the bounded repair problem between tree languages which can be specified by XML schemas or, more general, by regular tree languages [Sch07]. Our main result is that it is decidable whether or not R can be repaired into T with a uniformly bounded number of edits where R and T are regular tree languages.

In the case of tree languages, the problem turns out to be much more complex, both in terms of complexity and in terms of proof techniques that are needed to resolve it. We provide a characterization of bounded repairability that exploits a suitable notion of component of a stepwise tree automaton [CNT04], i.e., a form of automaton that turns out to be particularly convenient for analysing repairs. An additional complication for the tree case is that we need to consider structures of connected components of stepwise tree automata that take the form of trees, rather than chains. Our characterization of the bounded repairability of R into T requires that every component structure of R can be "covered" by a component structure of T. The notion of covering is subtle, and the proof that it captures bounded repairability requires lifting the notion of edit from the level of the individual trees to the level of the component trees associated with the automata for R and T.

Once we have our characterization, we can apply it to get decidability of the bounded repair problem, and with some additional optimization we can give tight complexity bounds. It turns out that, in contrast with the string setting, deciding the bounded repair problem for regular tree languages is equally complex no matter whether the finite tree automata is deterministic or non-deterministic, or whether it is given by a DTD or even by a non-recursive DTD. We show that for all these cases the bounded repair problem is coNEXPTIME-complete. For the sake of positive results, we study further restrictions over tree specifications and show that bounded repairability is much simpler to check for special classes of schemas. For example, we show that it is much less complex for deterministic DTDs when the alphabet is fixed, and much less complex when the restriction language R is trivial (i.e., the class of all trees).

The main characterization and most of the results presented in this chapter were included in [PRS12]. Nevertheless, in this chapter we include unpublished results. In particular, the coNEXPTIME-completeness result is new and was not considered in [PRS12].

Chapter 5: Streaming repairing of trees. In the last chapter we tackle the question of which pairs of tree specifications are streaming repairable with uniformly bounded cost. Our main result is a characterization and decision procedure for determining whether a streaming bounded repair strategy exists, in the important case of DTD schemas, and more generally of "deterministic top-down schemas".

Our 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 the previous chapters: we associate a graph with each schema, and

then look at the corresponding notion of connected components. 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 the other player, called Repairer, managing the stack for the target. Repairer needs to play in such a way that the "repeatable behaviour" of the components on the top of the stacks matches. 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 this game, and thus decidability of a winner. Both directions of the proof of our characterization are highly non-trivial.

With our characterization in hand, we are able to give an EXPTIME algorithm to determine the existence of a streaming repair strategy of uniformly bounded cost. This algorithm is completed with a non-trivial EXPTIME-hardness lower bound. Interestingly, this result shows that the complexity for streaming repairing trees is lower than in the non-streaming scenario (which is shown to be coNEXPTIME-complete in the previous chapter). Finally, we go on to isolate subcases where the complexity decreases to PSPACE.

Most of the results shown in this chapter are an extended version of the results presented in [BPR13].

Chapter 2

Bounded repairing of strings

In this chapter, we begin our study of repairing problems over strings. Our aim is to understand how difficult it is to transform a string satisfying R into a string satisfying T given two languages R and T. The notion is motivated by considering R to be a *restriction* – a constraint that the input is guaranteed to satisfied – while T is a *target* – a constraint that we want to enforce. For any string $u \in R$ we consider the worst-case of the number of edit operations needed to move u into T. That is, we look at the worst-case number of edits needed to get from R to T. Of course, this number may be infinite; the core of our results is a characterization for understanding the *bounded repair problem*, namely, to determine whether the supremum above is finite. In order to solve this effectively, we restrict our study to languages R and T specified by both deterministic and non-deterministic finite state automata. In both cases, we use the above characterization to determine the complexity of the bounded repair problem.

Above we considered the use of an edit/correction function that can read the whole string in memory. In this chapter we also consider the impact of limitations on the editing process – what happens when we require the editing to be done by a transducer, reading in the input letter-by-letter and producing the corrected output, based only on a finite amount of control state and a fixed amount of lookahead in the word. We refer to this as the (bounded) *streaming* repair problem. Interestingly, we show that this variant of the bounded repair problem can be characterized in terms of finite games. Furthermore, we use this characterization to isolate the complexity of the streaming repair problem for any lookahead and for any representation of the languages considered in the non-streaming case.

The streaming and non-streaming repair problems have very different flavors: the former are closely related to games played on the components of two automata, while the latter require a more global analysis, and exhibit a close relation to *distance automata*. However, there are connections between the different problems: we show that in the case where there is no restriction, the bounded repair problems are the same for both the streaming and non-streaming setting. We also show that

the bounded repair problem in the streaming setting is independent of the lookahead, and is robust under plausible alternative definitions.

The organization of this chapter is as follows: in Section 2.1 we review the main definitions for finite automata over words. Then in Section 2.2 we introduce the bounded repair problem for the non-streaming and streaming setting. In Section 2.3 we give our main characterizations for the bounded repair problem. We use these characterizations in Section 2.4 to analyzes the complexity of the bounded repair problem in both settings. Then we show in Section 2.5 some connections of the bounded repair problem with distance automata and games. Finally, we give some concluding remarks and related work in Section 2.6.

2.1 Regular specifications

In this chapter we base our setting on finite words (or strings) over a finite alphabet Σ. A word is basically a finite sequence of elements labeled in Σ. We denote by Σ^* the set of all finite words over Σ and by ϵ the empty word. Usually, we call a set $L \subseteq \Sigma^*$ a *language*.

Given $w \in \Sigma^*$, we denote by $|w|$ its length and, given two positions $1 \leq i \leq j \leq |w|$, we denote by $w[i]$ (resp., $w[i \dots j]$) the i-th symbol of w (resp., the infix of w starting at position i and ending at position j). The concatenation of two words w and w' is denoted by $w \cdot w'$ and the i-repetition of w is denoted by w^i for any $i \in \mathbb{N}$ (i.e. $w^i = w \cdot \overset{\text{i-times}}{\dots} \cdot w$ and $w^0 = \epsilon$). As usual, concatenation between words can be extended to languages $L, L' \subseteq \Sigma^*$ easily as follows: $L \cdot L' = \{w \cdot w' \mid w \in L \wedge w' \in L'\}$.

Word automata. A *non-deterministic finite-state automata* (shortly, *NFA*) is tuple of the form $\mathcal{A} = (\Sigma, Q, E, I, F)$, where:

- Σ is a finite alphabet,
- Q is a finite set of states,
- $E \subseteq Q \times \Sigma \times Q$ is a transition relation, and
- $I, F \subseteq Q$ are sets of initial and finite states.

Given a word $w = w_1 \dots w_n$ over Σ, we define a *run* ρ of $\mathcal{A}$ over w to be a function $\rho : \{0, \dots, n\} \to Q$ such that $(\rho(i-1), w_i, \rho(i)) \in E$ for each $i \in \{1, \dots, n\}$. We usually consider a run $\rho : \{0, \dots, n\} \to Q$ of $\mathcal{A}$ over w as a path $\rho(0) \xrightarrow{w_1} \rho(1) \xrightarrow{w_2} \dots \xrightarrow{w_n} \rho(n)$ over $\mathcal{A}$ viewed as a directed labeled graph. We say that a run ρ of $\mathcal{A}$ over w is accepting if $\rho(0) \in I$ and $\rho(n) \in F$. Thus, we define $\mathscr{L}(\mathcal{A}) \subseteq \Sigma^*$ to be the language recognized by $\mathcal{A}$:

$$\mathscr{L}(\mathcal{A}) \;=\; \{w \in \Sigma^* \mid \text{there exists an accepting run } \rho \text{ of } \mathcal{A} \text{ over } w\}$$

We say that a language $L \in \Sigma^*$ is *regular* iff there exists an NFA $\mathcal{A}$ such that $L = \mathscr{L}(\mathcal{A})$

We say that $\mathcal{A}$ is a *deterministic finite-state automaton* (DFA) whenever $|I| = 1$ and for each $q \in Q$ and $a \in \Sigma$ there exist at most one $q' \in Q$ such that $(q, a, q') \in E$. For any DFA $\mathcal{A}$ we usually denote the unique initial state by q_0 and turn its transition relation E into a partial function δ from $Q \times \Sigma^*$ to Q defined by $\delta(q, \epsilon) = q$ and $\delta(q, a\, u) = \delta(q', u)$ iff $(q, a, q') \in E$. Throughout this thesis, we implicitly use that DFA are equally expressive than NFA without any further reference (see [HU79] for more details).

For technical reasons, it is convenient to assume that all finite state automaton in this work are *trimmed*. Specifically, for every $q \in Q$ there exist $w \in \Sigma^*$ and an accepting run ρ of $\mathcal{A}$ over w such that $\rho(i) = q$ for some $i \in \mathbb{N}$. That is, all states are reachable from some initial states (i.e., they are accessible) and they can reach some final states (i.e., they are co-accessible). It is worth noticing that, since the decision problems we are going to deal with are at least NLOGSPACE-hard, then states that are not accessible or not co-accessible can be pruned in NLOGSPACE (see [HU79, CLRS09] for a survey). Then this assumption will have no impact on our complexity results.

Since automata can be viewed as directed labeled graphs, we inherit standard definitions and constructions in graph theory. In particular, given an automaton $\mathcal{A} = (\Sigma, Q, E, I, F)$ and a state $q \in Q$, we denote by $[q]$ the *strongly connected component* (SCC, for short) of $\mathcal{A}$ that contains all states mutually reachable from q (see [CLRS09] for a survey). Further, we denote by $\mathsf{SCC}(\mathcal{A})$ the set of all strongly connected components of $\mathcal{A}$, that is, $\mathsf{SCC}(\mathcal{A}) = \{[q] \subseteq Q \mid q \in Q\}$. We say that a component $X \in \mathsf{SCC}(\mathcal{A})$ is *final* if a final state is reachable from it (possibly outside X). Given a set X of states of $\mathcal{A}$ (e.g., a SCC), we denote by $\mathcal{A}|X$ the NFA obtained by restricting $\mathcal{A}$ to the set X and by letting the new initial and final states be all and only the states in X (note that if X consists of a single state without transitions to itself, then the language $\mathscr{L}(\mathcal{A}|X)$ recognized by the subautomaton $\mathcal{A}|X$ is equal to $\{\epsilon\}$). Finally, we denote by $\mathrm{dag}(\mathcal{A})$ the directed acyclic graph of the SCCs of $\mathcal{A}$ and by $\mathrm{dag}^*(\mathcal{A})$ the graph obtained from the symmetric and transitive closure of the edges of $\mathrm{dag}(\mathcal{A})$.

Regular expressions. During this work, we usually use regular expressions to give examples of regular languages. A *regular expression* E over a finite alphabet Σ is given by the following grammar:

$$E \;:=\; \epsilon \mid a \in \Sigma \mid E \cdot E \mid E + E \mid E^*$$

The language $\mathscr{L}(E) \subseteq \Sigma^*$ defined by a regular expression E over Σ is inductively defined over it syntactic tree. Specifically, for any $a \in \Sigma$ and regular expressions E and E', we define:

$$\begin{aligned}
\mathscr{L}(a) &= \{a\} \\
\mathscr{L}(\epsilon) &= \{\epsilon\} \\
\mathscr{L}(E \cdot E') &= \mathscr{L}(E) \cdot \mathscr{L}(E') \\
\mathscr{L}(E + E') &= \mathscr{L}(E) \cup \mathscr{L}(E') \\
\mathscr{L}(E^*) &= \textstyle\bigcup_{i=0}^{\infty} \mathscr{L}(E)^i.
\end{aligned}$$

It is well known that regular expressions are equally expressive than NFA and one can convert a regular expression E into an NFA $\mathcal{A}_E$ such that $\mathscr{L}(E) = \mathscr{L}(\mathcal{A}_E)$ in polynomial time (see [HU79] for a survey). We use this result tacitly in all our results and in examples where regular expressions are considered.

Transducers. A (sub-sequential) *transducer* is a tuple $\mathcal{Z} = (\Sigma, \Delta, Z, \kappa, z_0, \Omega)$, where Σ is a finite input alphabet, Δ is a finite output alphabet, Q is a finite set of states, κ is a partial transition function from $Q \times \Sigma$ to $\Delta^* \times Q$, z_0 is an initial state, and Ω is a partial function from Q to Δ^*. For every input word $u = a_1 \dots a_n \in \Sigma^*$, there is at most one run of $\mathcal{Z}$ on u of the form

$$z_0 \xrightarrow{a_1/v_1} z_1 \xrightarrow{a_2/v_2} \dots \xrightarrow{a_n/v_n} z_n \xrightarrow{\epsilon/v_{n+1}}$$

where $\kappa(z_i, a_i) = (v_i, z_{i+1})$ for all $0 \le i < n$ and $\Omega(z_n) = v_{n+1}$ [Sch77]. In such a case, we define the *output* of $\mathcal{Z}$ on u to be the word $\mathcal{Z}(u) = v_1 v_2 \dots v_n v_{n+1}$. Observe that the transducer outputs an additional, possibly empty, word to be added on at the end of the computation.

Transducers as above produce an output word (possibly empty) immediately on reading an input character. We will also consider transducers with a bounded amount of "delay". A *k-lookahead* transducer, with $k \in \mathbb{N}$, is as above, but where the transition function κ now has input in $Z \times \Sigma \times (\Sigma_\perp)^k$, with $\Sigma_\perp = \Sigma \cup \{\perp\}$ and $\perp \notin \Sigma$. Given an input word u and a position $1 \le i \le |u|$ in it, we denote by $\vec{u}_i$ the $(k+1)$-character subword of $u \perp^k$ that starts at position i and ends at position $i+k$. The output of a k-lookahead transducer $\mathcal{Z}$ on an input u of length n is the unique word $v = v_1 v_2 \dots v_n v_{n+1}$ for which there exists a sequence of states $z_0, ..., z_n$ satisfying $\kappa(z_i, \vec{u}_i) = (v_i, z_{i+1})$, for all $1 \le i \le n$, and $\Omega(z_n) = v_{n+1}$. Clearly, a 0-lookahead transducer is simply a standard (sub-sequential) transducer.

2.2 The bounded repair problem

In this chapter we repair strings by using the standard set of edit operations over letters [WF74]. Let $w = a_1 \dots a_n$ be a generic word in Σ^*. The first operation, called *deletion*, consists of removing a letter at position $i \in \{1, \dots, |w|\}$ from w, that is, $\mathsf{del}(w, i) = a_1 \dots a_{i-1}\, a_{i+1} \dots a_n$. The second operation, called *insertion*, consists of adding a letter $b \in \Sigma$ at position i in w, that is, $\mathsf{ins}(w, b, i) = a_1 \dots a_i\, b\, a_{i+1} \dots a_n$. The third and final operation, called *relabelling*, consists of modifying a letter at position i with a new label b, that is, $\mathsf{relab}(w, b, i) = a_1 \dots a_{i-1}\, b\, a_{i+1} \dots a_n$. Given two words $u \in \Sigma^*$ and $v \in \Delta^*$, we denote by $\mathsf{dist}(u, v)$ the *edit distance* between u and v, which is defined as the length of a shortest sequence s of edit operations (e.g., deleting a single character, modifying a single character, and inserting a single character) that transforms u into v [WF74, Wag74].

We are interested in quantifying how difficult it is to edit a word in one language to obtain a word in another language. That is, we have finite alphabets Σ and Δ and regular languages $R \subseteq \Sigma^*$ and $T \subseteq \Delta^*$, called the *restriction* and *target* languages, respectively. We would like to edit any string that is known to belong to the restriction language R into a string in the target language T.

A *repair strategy* for two languages R and T is any function from R to T. For a repair strategy f and a word $u \in R$, we define the (absolute) *cost of* f *on* u, denoted $\mathsf{cost}(u, f)$, as the edit distance between u and $f(u)$.

How do we measure the cost of edits needed to get from R to T? One method is to look at the largest number of edit operations that are needed to get into T from strings in R, that is, the supremum over all $u \in R$ of the minimum over all $v \in T$ of $\mathsf{dist}(u, v)$:

$$\mathsf{cost}(R, T) \ =^{\text{def}} \ \sup_{u \in R} \ \min_{v \in T} \ \mathsf{dist}(u, v).$$

Intuitively, having a finite uniform bound on the edit distance of words in R from T, means that the language R is "quite close to being a subset" of T – the gap between strings in R and strings in T is small. Note that the value $\mathsf{cost}(R, T)$ can be equally described as the minimum over all repair strategies f for R and T of the worst-case cost of f: indeed, the best repair strategy for R and T is just to output on any $u \in R$ the word in T that is closest to u with respect to the edit distance.

Example 4. *Consider the regular languages* $R = a^* \ b^*$ *and* $T = a^* \ c \ b^*$. *Clearly, any string in* R *can be converted to a string in* T *with at most* 1 *edit operation, namely,* $\mathsf{cost}(R, T) = 1$*; a repair strategy that achieves this cost maps any word* $a^n \ b^m \in R$ *to the word* $a^n \ c \ b^m \in T$.

The *bounded repair problem* is to decide, given two regular languages R and T, whether $\mathsf{cost}(R, T)$ is finite or not, namely, whether all words in R can be edited into T with at most some cost that does not depend on the input word. A variant of the bounded repair problem, called *threshold repair problem*, consists of deciding whether $\mathsf{cost}(R, T) \leq \nu$ for two given regular languages R and T and for a given number ν. This problem is the decision version of the problem of computing the exact value of $\mathsf{cost}(R, T)$. Clearly, the regular languages R and T must be finitely specified, for instance, in terms of finite state automata. During this chapter, we will study the complexity of the bounded repair problem for input languages specified by means of (i) deterministic finite state automata (DFA) and (ii) non-deterministic finite state automata (NFA).

Streaming vs non-streaming. The notion of "how much does it cost to edit a word in R to a word in T" assumes that a repair strategy could be any mapping from R to T (in principle, such a mapping could even fail to be computable). However, we know from [Wag74] that there is a dynamic programming algorithm that, given a word u and a regular target language T specified by means of a finite state automaton $\mathcal{T}$, computes in time $O(|u| \cdot |\mathcal{T}|)$ an optimal edit sequence that transforms u into some word in T. In particular, this shows that optimal repair strategies can be described by functions of fairly low complexity.

Sometimes it is desirable to have repair strategies that are in even more limited classes. Perhaps the ideal case is when we can repair R into T with a one-pass algorithm, that is, using a real-time sub-sequential transducer. Recall that a real-time sub-sequential transducer defines a word-to-word

partial function; if this function happens to produce a word in T for every input $u \in R$, then we say that it is a *streaming repair strategy* for R and T. Similarly, we can consider k-lookahead transducers, with $k \in \mathbb{N}$: this type of transducer outputs words on the basis of its current state and an input $(k+1)$-character window that represents a substring of u of the form $u[i] \dots u[i+k]$, where $u[i]$ is either the i-th symbol of w, if $i \leq |u|$, or a dummy symbol $\perp$, if $i > |u|$. Accordingly, we talk about a *k-lookahead streaming repair strategy* for R and T.

Given a k-lookahead streaming edit strategy $\mathcal{Z}$ for R and T and given a word $u \in R$, we can define the *cost of $\mathcal{Z}$ on u* in two ways:

1. letting $z_0 \xrightarrow{a_1/v_1} z_1 \xrightarrow{a_2/v_2} \dots \xrightarrow{a_n/v_n} z_n \xrightarrow{v_{n+1}}$ be the run of $\mathcal{Z}$ on u, we define the *aggregate cost of $\mathcal{Z}$ on u*, denoted $\mathsf{cost}^{\mathsf{aggr}}_{\mathcal{Z}}(u)$, to be the length of the final output v_{n+1} plus the sum, over all indices $1 \leq i \leq n$, of $\mathsf{dist}(a_i, v_i)$, where $\mathsf{dist}(a_i, v_i)$ is 1 if v_i is empty, $|v_i| - 1$ if a_i occurs in v_i, and $|v_i|$ otherwise;

2. considering the transducer $\mathcal{Z}$ as a repair strategy, we define the *edit cost of $\mathcal{Z}$ on u*, denoted $\mathsf{cost}^{\mathsf{edit}}_{\mathcal{Z}}(u)$, to be simply the edit distance between u and the output $\mathcal{Z}(u)$.

The first notion of cost considers the distortions performed in producing the input from the output – it is equivalent to considering the transducer as producing edit sequences rather than strings and counting the number of edits produced. The second notion of cost is global and it considers only the output and not its production (clearly, the edit cost never exceeds the aggregate cost). These two models of cost can be very different in general. As an example, consider a transducer $\mathcal{Z}$ on the input alphabet $\Sigma = \{a, b\}$ that converts a's to b's, and b's to a's. On the string $u_n = (ab)^n$, the aggregate cost is $2n$ since $\mathcal{Z}$ changes each letter, but the edit distance between u and $\mathcal{Z}(u)$ (i.e., the edit cost of $\mathcal{Z}$ on u in our sense) is only 2.

Similarly, we define the *worst-case aggregate/edit cost* of the k-lookahead streaming repair strategy $\mathcal{Z}$ as follows:

$$\mathsf{cost}^{\lambda}_{\mathcal{Z}}(R) \ =^{\text{def}} \ \sup_{u \in R} \ \mathsf{cost}^{\lambda}_{\mathcal{Z}}(u) \quad \text{for } \lambda \in \{\mathsf{aggr}, \mathsf{edit}\}.$$

The *k-lookahead streaming aggregate/edit cost* $\mathsf{cost}^{\lambda}_{k-\mathsf{lookahead}}(R, T)$ for two languages R and T and for $\lambda \in \{\mathsf{aggr}, \mathsf{edit}\}$ is then defined as the *minimum* of $\mathsf{cost}^{\lambda}_{\mathcal{Z}}(R)$ taken over all k-lookahead streaming strategies $\mathcal{Z}$ for R and T.

The following example illustrates the difference between bounded repairability in the streaming and non-streaming settings (a similar example was shown in the introduction, Chapter 1).

Example 5. *Consider the languages $R = (a+b)\ c^*\ (a^* + b^*)$ and $T = a\ c^*\ a^* + b\ c^*\ b^*$. In the non-streaming case, one can get from R to T by only editing the initial letter and thus $\mathsf{cost}(R, T) = 1$. In contrast, a k-lookahead streaming edit strategy must decide whether to leave or change the initial letter, and then it could be forced to repair an unbounded sequence of a's or b's after the sequence of c's. In this case we have $\mathsf{cost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}(R, T) = \mathsf{cost}^{\mathsf{edit}}_{k-\mathsf{lookahead}}(R, T) = \infty$.*

The *bounded repair problem in the k-lookahead streaming setting* consists of deciding whether $\mathsf{cost}^{\lambda}_{k-\mathsf{lookahead}}(R,T) < \infty$ for a given pair of regular languages R and T, a given $\lambda \in \{\mathsf{aggr}, \mathsf{edit}\}$, and a given $k \in \mathbb{N}$. To stress the difference between the streaming and the non-streaming settings, we explicitly refer to the original problem as the bounded repair problem in the *non-streaming case*. Even though the two models of aggregate and edit cost for streaming repair strategies can be very different, it will turn out that for the bounded repair problem it does not matter which cost model we choose (see Corollary 2.3.6).

Special cases. We are also interested in a variant of the bounded repair problem where the restriction language is assumed to be universal, i.e., a language of the form Σ^*. In this case, the input to the problem consists of a restriction alphabet Σ and a regular target language T. We refer to this variant as the *unrestricted case* of the bounded repair problem.

2.3 Characterizations of bounded repairability

We fix a restriction language R and a target language T and we assume, for the moment, that these languages are recognized by two NFA $\mathcal{R}$ and $\mathcal{T}$, respectively. Recall that $\mathrm{dag}(\mathcal{R})$ is the directed acyclic graph of the SCCs of $\mathcal{R}$ and $\mathrm{dag}^*(\mathcal{T})$ is the symmetric and transitive closure of $\mathrm{dag}(\mathcal{T})$. Moreover, recall that we assume that both $\mathcal{R}$ and $\mathcal{T}$ are trimmed, i.e. all unreachable and sink states are removed from $\mathcal{R}$ and $\mathcal{T}$.

2.3.1 Non-streaming setting

In order to characterize the positive instances of the bounded repair problem, we need to consider the relationships between the paths in $\mathrm{dag}(\mathcal{R})$ and the paths in $\mathrm{dag}^*(\mathcal{T})$. We say that a path $\pi = X_1 \dots X_k$ of SCCs in $\mathrm{dag}(\mathcal{R})$ is *covered* by a path $\tau = Y_1 \dots Y_h$ of SCCs in $\mathrm{dag}^*(\mathcal{T})$ if we have $k = h$ and $\mathscr{L}(\mathcal{R}|X_i) \subseteq \mathscr{L}(\mathcal{T}|Y_i)$ for all indices $1 \leq i \leq k$. In other words, if the language recognized by the i-th component along π is contained in the language recognized by the i-th component along τ.

The following characterization reduces the bounded repair problem in the non-streaming case to a path coverability problem between finite directed acyclic graphs.

Theorem 2.3.1. *Given two NFA $\mathcal{R}$ and $\mathcal{T}$, the following conditions are equivalent*

1. $\mathsf{cost}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) < \infty$ *(i.e., there is a strategy that repairs $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with a uniformly bounded number of edits),*
2. *every path in* $\mathrm{dag}(\mathcal{R})$ *is covered by some path in* $\mathrm{dag}^*(\mathcal{T})$*,*
3. $\mathsf{cost}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) \leq (1 + |\mathrm{dag}(\mathcal{R})|) \cdot |\mathcal{T}|$.

Proof of Theorem 2.3.1. Let $\mathcal{R} = (\Sigma, Q, E, I, F)$ and $\mathcal{T} = (\Delta, Q', E', I', F')$ be two NFA. We first prove the implication from *2.* to *3.*; later we will prove the implication from *1.* to *2.* (the implication from *3.* to *1.* is trivial and thus omitted).

We briefly outline the main ideas underlying the proof of the implication from *2.* to *3.*. In this direction, we assume that the coverability condition is satisfied, and we repair a generic word $u \in \mathscr{L}(\mathcal{R})$ as follows. We first consider the path π of SCCs of $\mathcal{R}$ that is induced by a successful run on u, and we observe that the input word u can be factorized into a bounded number of pieces, each one realizable within a component of the path. For instance, if $\pi = X_1 \dots X_k$ is the considered path inside $\mathrm{dag}(\mathcal{R})$, then we can factorize u as $u_1\ a_1\ u_2\ \dots\ a_{k-1}\ u_k$, with $u_i \in \mathscr{L}(\mathcal{R}|X_i)$ and $a_j \in \Sigma$ for all $1 \leq i \leq k$ and all $1 \leq j < k$. We then consider a path inside $\mathrm{dag}^*(\mathcal{T})$ that covers π, say $\tau = Y_1 \dots Y_k$, and we observe that each factor u_i of u is also realizable within the component Y_i of $\mathcal{T}$. In particular, no repair is needed on the factors $u_1, \dots, u_k$. On the other hand, the characters $a_1, \dots, a_{k-1}$ that are interleaved with the factors of u and that induce jumps along the components $X_1, \dots, X_k$ of π can be replaced by small strings inducing analogous jumps along the components $Y_1, \dots, Y_k$ of the covering path τ. This is possible because there exist partial runs connecting arbitrary states in each SCC Y_i to arbitrary states in the next SCC Y_{i+1}. In the end, the repair of u is formed by interleaving the factors $u_1, \dots, u_k$ with small strings replacing $a_1, \dots, a_{k-1}$, and by appending a final string to reach an accepting state of $\mathcal{T}$.

We now turn to the technical details of this proof. Suppose that every path in $\mathrm{dag}(\mathcal{R})$ is covered by some path in $\mathrm{dag}^*(\mathcal{T})$. We have to prove the existence of a repair strategy of $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with edit cost uniformly bounded by $(1+|\mathrm{dag}(\mathcal{R})|)\cdot|\mathcal{T}|$. Let us fix a generic word u in the restriction language $\mathscr{L}(\mathcal{R})$, and let ρ be an accepting run of $\mathcal{R}$ on u. The run ρ of $\mathcal{R}$ induces a path π in $\mathrm{dag}(\mathcal{R})$, which is defined as the sequence of SCCs of $\mathcal{R}$ visited by ρ. Let $\pi = X_1 X_2 \dots X_k$. Accordingly, we factorize the word u into the sequence of subwords $u_1, a_1, u_2, a_2, \dots, u_k$, where $u_i \in \mathscr{L}(\mathcal{R}|X_i)$ for all $1 \leq i \leq k$ and $a_i \in \Sigma$ for all $1 \leq i \leq k-1$. From the assumption of coverability of the paths in $\mathrm{dag}(\mathcal{R})$, we know that there is a path $\tau = Y_1 Y_2 \dots Y_k$ in $\mathrm{dag}^*(\mathcal{T})$ that covers π, namely, $\mathscr{L}(\mathcal{R}|X_i) \subseteq \mathscr{L}(\mathcal{T}|Y_i)$ for all $1 \leq i \leq k$. This shows that each subword u_i, with $1 \leq i \leq k$, belongs to the language $\mathscr{L}(\mathcal{T}|Y_i)$.

We can now construct inductively a corresponding word $f(u)$ that has the form $v_0\ u_1\ v_1\ \dots\ u_k\ v_k$ and that is accepted by $\mathcal{T}$. Recall that $\tau = Y_1 \dots Y_k$ is a path in $\mathrm{dag}^*(\mathcal{T})$ and that the automaton $\mathcal{T}$ is trimmed (i.e. it does not contain useless states). This means that all states in Y_1 can be reached from some initial state of $\mathcal{T}$ and, similarly, all states in Y_k can reach some final state. In particular, since $u_1 \in \mathscr{L}(\mathcal{T}|Y_1)$, we know that there exist an initial state q_0 of $\mathcal{T}$, two states p_1 and q_2 in Y_1, and a word $v_0 \in \Delta^*$ such that $\mathcal{T}$ admits a run of the form $q_0 \xrightarrow{v_0} p_1 \xrightarrow{u_1} q_1$. Moreover, without loss of generality, we can assume that the length of v_0 is bounded by the number of states of $\mathcal{T}$. As for the inductive step, we assume that the words $v_0, \dots, v_{i-1}$, with $0 \leq i \leq k-1$, are defined and that $\mathcal{T}$ admits a run on $v_0\ u_1\ \dots\ v_{i-1}$ from the initial state q_0 to a state $q_i \in Y_i$. Since every state in Y_{i+1} is

reachable from every state in Y_i, we know that there is a word v_i, with $|v_i| \leq |\mathcal{T}|$, and two states p_{i+1} and q_{i+1} in Y_{i+1} such that $\mathcal{T}$ admits a run of the form $q_i \xrightarrow{v_i} p_{i+1} \xrightarrow{u_{i+1}} q_{i+1}$. As for the final step, we assume that $v_0, \ldots, v_{k-1}$ are defined and that $\mathcal{T}$ admits a run on $v_0\ u_1\ \ldots\ v_{k-1}\ u_k$ from the initial state q_0 to a state $q_k \in Y_k$. Using arguments similar to the previous ones and the fact that Y_k can reach a final state, we derive the existence of a word v_k, with $|v_k| \leq |\mathcal{T}|$, and a final state p_{k+1} of $\mathcal{T}$ such that $\mathcal{T}$ admits a run of the form $q_k \xrightarrow{v_k} p_{k+1}$. Putting this all together, we obtain the existence of a successful run of $\mathcal{T}$ of the form

$$q_0 \xrightarrow{v_0} p_1 \xrightarrow{u_1} q_1 \xrightarrow{v_1} p_2 \xrightarrow{u_2} \ \ldots\ p_k \xrightarrow{u_k} q_k \xrightarrow{v_k} p_{k+1}$$

and hence the word $f(u) = v_0\ u_1\ v_1\ u_2\ \ldots\ u_k\ v_k$ is accepted by $\mathcal{T}$. Moreover, since $k \leq |\operatorname{dag}(\mathcal{R})|$ and $|v_i| \leq |\mathcal{T}|$ for all $0 \leq i \leq k$, we have that the edit distance between u and $f(u)$ is at most $(1+|\operatorname{dag}(\mathcal{R})|)\cdot|\mathcal{T}|$. This proves that there is a repair strategy f for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$ with edit cost uniformly bounded by $(1+|\operatorname{dag}(\mathcal{R})|)\cdot|\mathcal{T}|$, and hence

$$\mathsf{cost}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) \ \leq\ (1+|\operatorname{dag}(\mathcal{R})|)\cdot|\mathcal{T}|.$$

We now prove the implication from *1.* to *2.*. We assume that there exists a repair strategy f for $\mathcal{R}$ and $\mathcal{T}$ with bounded cost and then we show that every path in $\operatorname{dag}(\mathcal{R})$ is covered by a path in $\operatorname{dag}^*(\mathcal{T})$. The general idea is to associate with any path $\pi = X_1, \ldots, X_k$ inside $\operatorname{dag}(\mathcal{R})$ a suitable word $u_\pi \in \mathscr{L}(\mathcal{R})$, called the *witnessing word* of π, whose repair according to the strategy f induces a path τ inside $\operatorname{dag}^*(\mathcal{T})$ that covers π. Intuitively, the witnessing word u_π is obtained from the path π by replacing every non-trivial component X_i with a sufficiently large number of repetitions of a special string in $\mathscr{L}(\mathcal{R}|X_i)$, which we called *the fingerprint of* X_i. The number of repetitions of each fingerprint will depend on the worst-case repair cost $N = \max_{u\in\mathscr{L}(\mathcal{R})}\{\mathsf{cost}(u, f)\}$, and will be large enough to imply that any repair of the witnessing word u_π achieved by at most N edits contains at least one copy of the fingerprint of each non-trivial component X_i of π. Moreover, the order of occurrence of the fingerprints inside the repaired string will correspond to the order of the components in the path π. One finally looks at some run of $\mathcal{T}$ that accepts $f(u_\pi)$: thanks to the occurrence order of the fingerprints of $f(u_\pi)$, this run must induce a path τ inside $\operatorname{dag}^*(\mathcal{T})$ that covers π.

Before constructing the witnessing word, we prove a technical lemma, which defines precisely the concept of fingerprint of a component of $\mathcal{R}$. Intuitively, the lemma shows that, for any given component X of $\mathcal{R}$, one can find a word u_X that can be arbitrarily "pumped" inside the language $\mathscr{L}(\mathcal{R}|X)$ (i.e., $u_X \cdot \ldots \cdot u_X \in \mathscr{L}(\mathcal{R}|X)$) and such that, for all components Y of $\mathcal{T}$, $u_X \in \mathscr{L}(\mathcal{T}|Y)$ iff $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y)$. Below, we say that a word u is *cyclic* (for a given component X) if there is a state $q \in X$ such that $\mathcal{R}$ admits a run of the form $q \xrightarrow{u} q$.

Lemma 2.3.2. *For every component X of $\mathcal{R}$, there is a* cyclic *word $u_X \in \mathscr{L}(\mathcal{R}|X)$ such that, for every component Y of $\mathcal{T}$,*

$$u_X \in \mathscr{L}(\mathcal{T}|Y) \quad \textit{iff} \quad \mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y).$$

Proof. Let X be a component of $\mathcal{R}$ and let $Y_1, \ldots, Y_m$ be all the components of $\mathcal{T}$. We construct the cyclic word u_X by exploiting an induction over the number m of components in $\mathcal{T}$. That is, we prove that for every $1 \leq i \leq m$, there is a cyclic word $u_i \in \mathscr{L}(\mathcal{R}|X)$ such that:

$$\forall 1 \leq j \leq i. \qquad \mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y_j) \quad \text{iff} \quad u_i \in \mathscr{L}(\mathcal{T}|Y_j) \tag{$\star$}$$

Clearly, the lemma follows from $(\star)$ when we let $u_X = u_m$.

For the base step $i = 1$, we distinguish two cases, depending on whether $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y_1)$ or not. If $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y_1)$, then we define $u_1 = \epsilon$ and $(\star)$ trivially holds. Otherwise, if $\mathscr{L}(\mathcal{R}|X) \nsubseteq \mathscr{L}(\mathcal{T}|Y_1)$, then we let u be any word in $\mathscr{L}(\mathcal{R}|X) \setminus \mathscr{L}(\mathcal{T}|Y_1)$ and $p, q \in X$ be two states such that $\mathcal{R}$ admits a run $p \xrightarrow{u} q$. Since X is connected, there exists u' such that $\mathcal{R}$ admits a run $q \xrightarrow{u'} p$. We thus define $u_1 = u \cdot u'$. By construction, we have that u_1 is cyclic. Furthermore, it is easy to see that $u_1 \notin \mathscr{L}(\mathcal{T}|Y_1)$ and hence $(\star)$ holds.

For the inductive step, let $i < m$ and suppose that there exists a word u_i that satisfies $(\star)$. We prove that $(\star)$ holds for $i+1$ as well. Again, we distinguish between two cases, depending on whether $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y_{i+1})$ or not. If $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y_{i+1})$, then we define $u_{i+1} = u_i$, in such a way that $(\star)$ holds trivially. Otherwise, let u be a word in $\mathscr{L}(\mathcal{R}|X) \setminus \mathscr{L}(\mathcal{T}|Y_{i+1})$. Since u_i is cyclic and $u \in \mathscr{L}(\mathcal{R}|X)$, there exist some states $r, p, q \in X$ such that $\mathcal{R}$ admits runs of the form $r \xrightarrow{u_i} r$ and $p \xrightarrow{u} q$. Let u' and u'' be some words in $\mathscr{L}(\mathcal{R}|X)$ such that $\mathcal{R}$ admits runs of the form $r \xrightarrow{u'} p$ and $q \xrightarrow{u''} r$ (clearly, u' and u'' exist since X is a component of $\mathcal{R}$ and $r, p, q \in X$). Now, define $u_{i+1} = u' \cdot u \cdot u'' \cdot u_i$. One can easily show that u_{i+1} is a cyclic word in $\mathscr{L}(\mathcal{R}|X)$ by following the run:

$$r \xrightarrow{u'} p \xrightarrow{u} q \xrightarrow{u''} r \xrightarrow{u_i} r.$$

It is also easy to show that u_{i+1} is not in $\mathscr{L}(\mathcal{T}|Y_{i+1})$. Indeed, if $u_{i+1} \in \mathscr{L}(\mathcal{T}|Y_{i+1})$, then there would exist states $p', q' \in Y_{i+1}$ such that $p' \xrightarrow{u} q'$. This is a contradiction since $u \notin \mathscr{L}(\mathcal{T}|Y_{i+1})$. Finally, we know from the inductive hypothesis that for every $1 \leq j \leq i$, if $u_{i+1} \in \mathscr{L}(\mathcal{T}|Y_j)$, then $u_i \in \mathscr{L}(\mathcal{T}|Y_j)$ and $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y_j)$. The converse implication follows in a similar way. We conclude that, for every $1 \leq j \leq i+1$, $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y_j)$ iff $u_{i+1} \in \mathscr{L}(\mathcal{T}|Y_j)$. This proves the inductive step for $(\star)$. □

We associate with each component X a word u_X that satisfies Lemma 2.3.2, and we call it the *fingerprint of X*.

Consider now any path $\pi = X_1 \ldots X_k \in \mathrm{dag}(\mathcal{R})$ and the fingerprint u_{X_i} for each component X_i. Since each word u_{X_i} is cyclic for X_i and since all states in the sequence $X_1 \ldots X_k$ are mutually

reachable, we know that there exist some states $q_1 \in X_1, \ldots, q_k \in X_k$ and some words $u_1, \ldots, u_{k-1}$ such that $\mathcal{R}$ admits a run of the form $q_1 \xrightarrow{u_{X_1}} q_1 \xrightarrow{u_1} q_2 \xrightarrow{u_{X_2}} q_2 \xrightarrow{u_2} \cdots \xrightarrow{u_{k-1}} q_k \xrightarrow{u_{X_k}} q_k$. Namely, the words u_i connects the fingerprints of $X_1, \ldots, X_k$ by going from one cyclic state to the next one. Furthermore, we know that $\mathcal{R}$ is trimmed and then there exists words u_0, u_k such that $\mathcal{R}$ admits runs $q_I \xrightarrow{u_0} q_1$ and $q_k \xrightarrow{u_k} q_F$ for some states $q_I \in I$ and $q_F \in F$. Putting everything together, we have that, for every positive natural number n, the word

$$u^{(n)} =^{\text{def}} u_0\ u_{X_1}^n\ u_1\ u_{X_2}^n\ \ldots\ u_{X_k}^n\ u_k$$

is accepted by $\mathcal{R}$. Thus, we associate with π the witness word $u_\pi = u^{(M)}$ where $M = (N+1) \cdot (|\operatorname{dag}(\mathcal{R})| + 1)$.

Towards the end of the proof, we use the witness word u_π to extract from $f(u_\pi)$ a path π' in $\operatorname{dag}^*(\mathcal{T})$ such that π' covers π. First of all, recall that $f(u_\pi)$ is accepted by $\mathcal{T}$ and f modifies at most N parts of u_π. From this last fact and the construction of u_π, one can easily show that $f(u_\pi)$ is of the form $v_0\ (u_{X_1})^l\ v_1\ (u_{X_2})^l\ \ldots\ (u_{X_k})^l\ v_k$ where $l = |\operatorname{dag}(\mathcal{R})| + 1$. Notice here that for each X_i we have that $f(u_\pi) = v\ (u_{X_1})^l\ v'$ for some words v, v'. By the pigeon-hole principle, this means that every run of $\mathcal{T}$ over $f(u_\pi)$ must have a subrun $p_i \xrightarrow{u_{X_i}} p_i'$ where p_i and p_i' are contained in the same component (i.e. $[p_i] = [p_i']$) for every $1 \leq i \leq k$. This implies that $f(u_\pi)$ can be written as $f(u_\pi) = v_0'\ u_{X_1}\ v_1'\ u_{X_2}\ \ldots\ u_{X_k}\ v_k'$ and there exists an accepting run of $\mathcal{T}$ over $f(u_\pi)$ of the form:

$$p_I \xrightarrow{v_0'} p_1 \xrightarrow{u_{X_1}} p_1' \xrightarrow{v_1'} p_2 \xrightarrow{u_{X_2}} p_2' \ldots\ p_k \xrightarrow{u_{X_k}} p_k' \xrightarrow{v_k'} p_F$$

where $[p_i] = [p_i']$ for every $1 \leq i \leq k$. In other words, this means that $u_{X_i} \in \mathscr{L}(\mathcal{T}|[p_i])$ for every $1 \leq i \leq k$. From this last fact we can easily extract a path π' in $\operatorname{dag}^*(\mathcal{T})$ that covers π. Indeed, define the path $\pi' = [p_1] \ldots [p_k]$ in $\operatorname{dag}^*(\mathcal{T})$. Given that each $u_{X_i} \in \mathscr{L}(\mathcal{T}|[p_i])$, then we have by Lemma 2.3.2 that $\mathscr{L}(\mathcal{R}|X_i) \subseteq \mathscr{L}(\mathcal{T}|[p_i])$ and, thus, π' covers π . Finally, we conclude that every path π in $\operatorname{dag}(\mathcal{R})$ is covered by path π' in $\operatorname{dag}^*(\mathcal{T})$. □

2.3.2 Streaming setting

We now modify Theorem 2.3.1 to give a characterization of the positive instances of the bounded repair problem in the streaming case, adding in a game setting. For this it is convenient to assume that $\mathcal{R}$ and $\mathcal{T}$ are two DFA recognizing the restriction and target languages. We associate with $\mathcal{R}$ and $\mathcal{T}$ a *reachability game* between two players, Adam and Eve, on a suitable arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, defined in terms of the SCCs of $\mathcal{R}$ and $\mathcal{T}$. The idea underlying the game is as follows: during Adam's construction of a path π in $\operatorname{dag}(\mathcal{R})$, Eve has to provide a construction of a corresponding path $f(\pi)$ in $\operatorname{dag}^*(\mathcal{T})$ that covers π; moreover, the resulting function f must satisfy the following condition: if $\pi \cdot X$ is an extension of the path π in $\operatorname{dag}(\mathcal{R})$ by a single SCC, then either $f(\pi \cdot X)$ coincides with $f(\pi)$ or it is an extension of $f(\pi)$ by a single SCC, i.e. $f(\pi \cdot X)$ is of the form $f(\pi) \cdot Y$.

Formally, the nodes of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ for Adam (resp., Eve) are the pairs of the form (X, Y) (resp., (Y, X)), where X is a SCC of $\mathcal{R}$ and Y is a SCC of $\mathcal{T}$. The edges of the arena connect Adam's nodes (X, Y) to Eve's nodes (Y, X') where (X, X') is an edge of $\mathrm{dag}(\mathcal{R})$ and, similarly, Eve's nodes (Y, X) to Adam's nodes (X, Y') where (Y, Y') is an edge of $\mathrm{dag}^*(\mathcal{T})$ and, in addition, $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y')$. The initial node is an Eve node (Y_0, X_0), where X_0 is the SCC of the initial state of $\mathcal{R}$ and Y_0 is the SCC of the initial state of $\mathcal{T}$. The last player who moves wins. Intuitively, Adam's objective is to reach a node (X, Y) where Eve cannot respond with any move. Conversely, Eve's objective is to reach a node (Y, X) where Adam cannot respond with any move. As usual, we say that a player has a winning strategy on the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ if he/she can win the reachability game on $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ independently of the choices of the other player.

The following characterization reduces the bounded repair problem in the streaming setting to the problem of determining the winner of a reachability game.

Theorem 2.3.3. *Given two DFA $\mathcal{R}$ and $\mathcal{T}$, the following two conditions are equivalent:*

1. *there exists a streaming strategy with some lookahead that repairs $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with uniformly bounded edit cost,*

2. *Eve has a winning strategy in the reachability game on $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.*

The idea underlying the proof of the direction from *2.* to *1.* is as follows. If we have a winning strategy for Eve, then we can get a streaming repair strategy for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$ by tracking the current SCC X of the input string and maintaining the invariant that the component Y of the current repaired string is such that (X, Y) is a position consistent with Eve's winning strategy. When a new letter comes in and changes the SCC in the restriction from X to X', we respond with a repair that moves from Y to the response SCC Y' that preserves the invariant.

For the direction from *1.* to *2.*, we assume that there is a streaming k-lookahead edit strategy that repairs $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with uniformly bounded cost and derive a strategy for Eve. The strategy will maintain the invariant that the position (X, Y) corresponds to some input string u and response v consistent with the repair strategy. If, by way of contradiction, we reach a pair (X, Y) corresponding to some string u, there is a successor SCC X' of X corresponding to some extension uu', and (Y, X') is a losing position for Eve, then we can construct a single counterexample word for every candidate SCC. Given that for every successor SCC Y' of Y there is $v' \in \mathscr{L}(\mathcal{R}|X') \setminus \mathscr{L}(\mathcal{T}|Y')$, we can concatenate multiple copies of v' together. If we make the number of copies large enough, such a string cannot be repaired by our transducer with a bounded number of edit operations, which yields a contradiction.

Before turning to the proof of the above theorem, it is convenient to establish two preliminary lemmas, which will be also reused in other proofs (e.g., in the proof of Proposition 2.4.4). For the sake of brevity, given an NFA (resp., DFA) $\mathcal{A}$, a SCC X of it, and a state $q \in X$, we denote by $\mathcal{A}|_q X$

the NFA (resp., the DFA) obtained from the subautomaton $\mathcal{A}|X$ by letting q be the unique initial state (recall that the final states of $\mathcal{A}|X$ are all the states in X).

Lemma 2.3.4. *Let $\mathcal{R}$ be an NFA, let $\mathcal{T}$ be a DFA, and let X and Y be some SCCs of $\mathcal{R}$ and $\mathcal{T}$, respectively. We have that*

$$\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y) \text{ iff } \exists\, q \in X,\ r \in Y.\ \mathscr{L}(\mathcal{R}|_q X) \subseteq \mathscr{L}(\mathcal{T}|_r Y).$$

Proof. Let $\mathcal{R} = (\Sigma, Q, E, I, q_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', q_0', F')$. We first prove the right-to-left implication. Suppose that $\mathscr{L}(\mathcal{R}|_q X) \subseteq \mathscr{L}(\mathcal{T}|_r Y)$ for some states $q \in X$ and $r \in Y$. Let u be a word in $\mathscr{L}(\mathcal{R}|X)$. Since $q \in X$ and X is a SCC, we know that there is a word $u_0 \in \Sigma^*$ such that $u_0\ u \in \mathscr{L}(\mathcal{R}|_q X)$. Since $\mathscr{L}(\mathcal{R}|_q X) \subseteq \mathscr{L}(\mathcal{T}|_r Y)$, the same word $u_0\ u$ belongs also to the language $\mathscr{L}(\mathcal{T}|_r Y)$. In particular, this shows that $u \in \mathscr{L}(\mathcal{T}|Y)$.

We now prove the contrapositive of the left-to-right implication by assuming that $\mathscr{L}(\mathcal{R}|_q X) \nsubseteq \mathscr{L}(\mathcal{T}|_r Y)$ for all $q \in X$ and all $r \in Y$ and deriving $\mathscr{L}(\mathcal{R}|X) \nsubseteq \mathscr{L}(\mathcal{T}|Y)$. Let $Y = \{r_1, \ldots, r_n\}$. We first prove, by exploiting an induction on $i \leq n$, that there is a word u_i that belongs to $\mathscr{L}(\mathcal{R}|X)$ but not to $\mathscr{L}(\mathcal{T}|_{r_j} Y)$ for all indices $1 \leq j \leq i$. The base case $i = 1$ is trivial. As for the inductive case, we assume that the claim holds for $i < n$ and we prove it for $i + 1$. Let u_i be the word obtained from the inductive hypothesis such that $u_i \in \mathscr{L}(\mathcal{R}|X) \setminus \mathscr{L}(\mathcal{T}|_{r_j} Y)$ for all $1 \leq j \leq i$. Moreover, let r'_{i+1} be the state reached by $\mathcal{T}$ from r_{i+1} after parsing the word u_i, that is, $r'_{i+1} = \delta'(r_{i+1}, u_i)$. If $r'_{i+1} \notin Y$, then the claim follows trivially. It remains to consider the case $r'_{i+1} \in Y$. Let p_i and q_i be two states in X such that $p_i \xrightarrow{u_i} q_i$. We know from the original assumption that $\mathscr{L}(\mathcal{R}|_{q_i} X) \nsubseteq \mathscr{L}(\mathcal{T}|_{r'_{i+1}} Y)$. In particular, we know that there is a word $v_{i+1} \in \mathscr{L}(\mathcal{R}|_{q_i} X) \setminus \mathscr{L}(\mathcal{T}|_{r'_{i+1}} Y)$. This shows that the word $u_{i+1} = u_i\ v_{i+1}$ belongs to the language $\mathscr{L}(\mathcal{R}|X)$, but not to the language $\mathscr{L}(\mathcal{T}|_{r_{i+1}} Y)$. Finally, since Y is a SCC, it follows that u_{i+1} does not belong to any of the languages $\mathscr{L}(\mathcal{T}|_{r_j} Y)$ either, for all $1 \leq j \leq i$, and this concludes the proof. □

Lemma 2.3.5. *Let $\mathcal{R}$ and $\mathcal{T}$ be two DFA and let X and Y be some SCCs of $\mathcal{R}$ and $\mathcal{T}$, respectively. We have that*

$$\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y) \qquad \text{implies} \qquad \forall\, q \in X.\ \exists\, r \in Y.\ \mathscr{L}(\mathcal{R}|_q X) \subseteq \mathscr{L}(\mathcal{T}|_r Y).$$

Proof. Let $\mathcal{R} = (\Sigma, Q, \delta, q_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', q_0', F')$ be two DFA, let X and Y be two SCC of $\mathcal{R}$ and $\mathcal{T}$, respectively, such that $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y)$, and q be a state in X. We know from Lemma 2.3.4 that there exist two states $p \in X$ and $p' \in Y$ such that $\mathscr{L}(\mathcal{R}|_p X) \subseteq \mathscr{L}(\mathcal{T}|_{p'} Y)$. Moreover, since all states in X are reachable from each other, there is a word u such that $\delta(p, u) = q$. Since $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y)$, we know that the state $r = \delta(p', u)$ belongs to Y. Finally, for every word $v \in \mathscr{L}(\mathcal{R}|_q X)$, we have $u\ v \in \mathscr{L}(\mathcal{R}|_p X)$ and hence, since $\mathscr{L}(\mathcal{R}|_p X) \subseteq \mathscr{L}(\mathcal{T}|_{p'} Y)$, $u\ v \in \mathscr{L}(\mathcal{T}|_{p'} Y)$, whence $v \in \mathscr{L}(\mathcal{T}|_r Y)$. This shows that $\mathscr{L}(\mathcal{R}|_q X) \subseteq \mathscr{L}(\mathcal{T}|_r Y)$. □

Proof of Theorem 2.3.3. Let $\mathcal{R} = (\Sigma, Q, \delta, q_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', r_0, F')$ be two DFA and let $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ be the arena obtained from $\mathcal{R}$ and $\mathcal{T}$ as described above. Below, we prove first the implication from *2.* to *1.* and then the implication from *1.* to *2.*

Suppose that Eve has a winning strategy in the reachability game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Since reachability games are positionally determined, Eve's strategy can be described by a partial function[1] g that maps a node of the form (Y, X), with $X \in \mathrm{dag}(\mathcal{R})$ and $Y \in \mathrm{dag}^*(\mathcal{T})$, to a successor $g(Y, X) = (X, Y')$ (if there is any) in the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. We can use Eve's winning strategy g to construct a real-time sub-sequential 0-lookahead transducer $\mathcal{Z}$ that implements a streaming repair strategy for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$ having uniformly bounded aggregate cost. Intuitively, the transducer $\mathcal{Z}$ works as follows. While parsing the input word u from the restriction language $\mathscr{L}(\mathcal{R})$ and emitting a corresponding word v, the transducer $\mathcal{Z}$ mimics, at the same time, the transitions of both $\mathcal{R}$ and $\mathcal{T}$. Each time the restriction automaton $\mathcal{R}$ exits the current SCC X and enters a new SCC X', a corresponding move $(X, Y) \xrightarrow{\mathsf{Adam}} (Y, X')$ for Adam is identified; accordingly, on the basis of Eve's response $g(Y, X') = (X', Y')$ (recall that Eve's strategy was assumed to be winning), the transducer $\mathcal{Z}$ outputs a suitable word that makes the target automaton $\mathcal{T}$ move from the SCC Y to the SCC Y' (the new state in Y' can be determined using Lemma 2.3.5 since we have $\mathscr{L}(\mathcal{R}|X') \subseteq \mathscr{L}(\mathcal{R}|Y')$).

The formal definition of the transducer $\mathcal{Z}$ is a bit more technical due to the treatment of some special cases. First of all, we can assume, without loss of generality, that the initial state q_0 of $\mathcal{R}$ has no entering transitions (we can always enforce this condition by introducing duplicate states). Then, we let X_0 be the SCC of q_0 in $\mathcal{R}$, Y_0 be the SCC of the initial state r_0 of $\mathcal{T}$, $g(Y_0, X_0) = (X_0, Y_1)$ be target node in the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ for the first move of the player Eve (recall that g is the winning strategy of Eve), and r_1 be some arbitrary state in Y_1 such that $\mathscr{L}(\mathcal{R}|_{q_0} X_0) \subseteq \mathscr{L}(\mathcal{T}|_{r_1} Y_1)$ (note that $\mathscr{L}(\mathcal{R}|X_0) \subseteq \mathscr{L}(\mathcal{R}|Y_1)$ and hence, by Lemma 2.3.5, such a state r_1 must exist). Accordingly, we define v_0 to be the shortest word such that $\delta'(r_0, v_0) = r_1$ (note that since (Y_0, Y_1) is an edge in $\mathrm{dag}^*(\mathcal{T})$, the state r_1 is reachable from the state r_0). Intuitively, the word v_0 is the first prefix emitted by the transducer $\mathcal{Z}$ at the beginning of the computation (it should be now clear why we assumed that the initial state q_0 of $\mathcal{R}$ has no entering transitions). We are finally ready to define the transducer $\mathcal{Z} = (\Sigma, \Delta, Z, \kappa, z_0, \Omega)$:

- $Z = Q \times Q'$;
- for every state $z = (q, r) \in Z$ and every symbol $a \in \Sigma$, $\kappa(s, a) = (v, z')$, where $z' = (\delta(q, a), \delta'(r, v))$ and v is the word over Δ defined as follows
  1. if both states q and $\delta(q, a)$ belong to the same SCC of $\mathcal{R}$, then we let v be either the input symbol a or the word $v_0 \cdot a$, depending on whether $q \neq q_0$ or $q = q_0$;

[1] The reason for the strategy function g to be partial is that some positions in the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ may have no successors.

2. otherwise, if the states q and $\delta(q,a)$ belong to different SCCs, we denote by X' the SCC of the state $\delta(q,a)$ in $\mathcal{R}$, by Y the SCC of the state r in $\mathcal{T}$, by (X',Y') the response $g(Y,X')$ given by Eve's winning strategy, by r' some arbitrary state in Y' such that $\mathscr{L}(\mathcal{R}|_{\delta(q,a)}X') \subseteq \mathscr{L}(\mathcal{T}|_{r'}Y')$ (we know from the containment $\mathscr{L}(\mathcal{R}|X') \subseteq \mathscr{L}(\mathcal{R}|Y')$ and from Lemma 2.3.5 that such a state r' exists), and by v' the shortest word such that $\delta'(r,v') = r'$ (note that since (Y,Y') is an edge in $\mathrm{dag}^*(\mathcal{T})$, the state r' is reachable from the state r); accordingly, we let v be either the word v' or the word $v_0\ v'$, depending on whether $q \neq q_0$ or $q = q_0$;

- $z_0 = (q_0, r_0)$;
- for every state $z = (q,r) \in Z$, $\Omega(z)$ is the shortest word $v \in \Delta^*$ such that $\delta(r,v) \in F'$ (note that the existence of such a word v is guaranteed by the assumption that every state of $\mathcal{T}$ can reach some final state).

Using arguments similar to those used in the proof of Theorem 2.3.1 (e.g., by associating with each successful run of $\mathcal{R}$ a corresponding path π in $\mathrm{dag}(\mathcal{R})$ and a covering path τ in $\mathrm{dag}^*(\mathcal{T})$ induced by Eve's winning strategy), one can show that the transducer $\mathcal{Z}$ maps any word $u \in \mathscr{L}(\mathcal{R})$ to a word $\mathcal{Z}(u) \in \mathscr{L}(\mathcal{T})$. Moreover, by construction, the output produced at each transition of $\mathcal{Z}$ can be different from the input symbol only if $\mathcal{Z}$ is at the beginning of the computation, at the end of the computation, or if the current SCC of the automaton $\mathcal{R}$ has just changed; in each of these cases, the length of the produced output does not exceed the number of states in $\mathcal{T}$. This shows that the the aggregate cost of $\mathcal{Z}$ on input $u \in \mathscr{L}(\mathcal{R})$ is at most $(1 + |\mathrm{dag}(\mathcal{R})|) \cdot |\mathcal{T}|$ and hence

$$\mathsf{cost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) \ \leq\ (1 + |\mathrm{dag}(\mathcal{R})|) \cdot |\mathcal{T}|.$$

We now prove the implication from *1.* to *2.*. More specific, we assume that $\mathcal{Z} = (\Sigma, \Delta, Z, \kappa, z_0, \Omega)$ is a transducer with k-lookahead that implements a repair strategy for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$ with cost uniformly bounded by a natural number N and we construct from this a winning strategy for Eve. More precisely, we shall specify Eve's moves $g(Y,X)$ on those nodes (Y,X) in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, with $X \in \mathrm{dag}(\mathcal{R})$ and $Y \in \mathrm{dag}^*(\mathcal{T})$, for which there exist some words $u_X, u'_X \in \Sigma^*$ and $v_Y \in \Delta^*$ such that

- $\delta(q_0, u_X\ u'_X) \in X$,
- $\delta'(r_0, v_Y) \in Y$,
- $\kappa(z_0, u_X) = (v_Y, z')$ for some $z' \in Z$.

We call these nodes *reachable*. On the remaining (unreachable) nodes, we let the function g be unspecified. As a preliminary remark, we observe that from the definition of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ and from the above properties, the domain of the function g is closed under the ancestor relation. In

other words, if $g(Y, X)$ is defined and (Y', X') is an ancestor of (Y, X) in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, then $g(Y', X')$ is defined as well. Below, we define the moves $g(Y, X)$ by exploiting an induction on the distance of the SCC X from the root of $\mathrm{dag}(\mathcal{R})$, which is the SCC of the initial state q_0 of $\mathcal{R}$.

Let us consider a reachable node (Y, X) in the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Since (Y, X) is reachable, we know that there exist some words $u_X, u'_X \in \Sigma^*$ and $v_Y \in \Delta^*$ such that (i) $\delta(q_0, u_X\ u'_X) \in X$, (ii) $\delta'(r_0, v_Y) \in Y$, and (iii) $\kappa(z_0, u_X) = (v_Y, z')$ for some $z' \in Z$. For the sake of brevity, we let $q = \delta(q_0, u_X\ u'_X)$. We claim that there is a SCC Y' that is a successor of Y in $\mathrm{dag}^*(\mathcal{T})$ (possibly $Y' = Y$) such that $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y')$. Indeed, suppose, by way of contradiction that this is not the case. Let $Y_1, \ldots, Y_h$ be all the the descendants of Y, topologically ordered according to their accessibility relation (hence $Y_1 = Y$). Using arguments similar to those used in the proof of Theorem 2.3.1, we can recursively construct a sequence of words $u_0, u_1, u'_1, \ldots, u_h, u'_h$ over Σ such that

- $p \xrightarrow{u_i\ u'_i} p$ is a run of $\mathcal{R}$ on $u_i\ u'_i$ for all $1 \leq i \leq h$,
- $u_i \notin \mathscr{L}(\mathcal{T}|Y_i)$ for all $1 \leq i \leq h$.

In particular, the first property implies that the word

$$u_X\ u'_X\ (u_1\ u'_1)^{N+1}\ \ldots\ (u_h\ u'_h)^{N+1}$$

is a prefix of some word u in the language $\mathscr{L}(\mathcal{R})$ (recall the assumption that all states in $\mathcal{R}$ can reach some final states). Moreover, since u contains $N+1$ occurrences of the subwords $u_1, \ldots, u_h$ and since $\mathcal{Z}$ implements a repair strategy with cost uniformly bounded by N, we have that the words $u_1, \ldots, u_h$ must occur at least once, and in this particular order, as subwords of $\mathcal{Z}(u)$ (observe that having a transducer with k-lookahead does not help here). However, since v_Y is a prefix of $\mathcal{Z}(u)$, $\delta(r_0, v_Y) \in Y$, and $u_i \notin \mathscr{L}(\mathcal{T}|Y_i)$ for all $1 \leq i \leq h$, we have that $\mathcal{Z}(u)$ can not belong to the target language $\mathscr{L}(\mathcal{T})$, which is against the assumption that $\mathcal{Z}$ implements a repair strategy for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$. We must conclude that there is a SCC Y' that is a successor of Y in $\mathrm{dag}^*(\mathcal{T})$ and that satisfies $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y')$. Accordingly, we define Eve's move on node (Y, X) to be $g(Y, X) = (X, Y')$ (note that this is a valid move in the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$).

Turning to the proof of the main theorem, we argue that the above defined strategy g for Eve is winning. This is equivalent to proving that for every partial play of the form

$$(Y_0, X_0) \xrightarrow{\mathsf{Eve}} (X_0, Y_1) \xrightarrow{\mathsf{Adam}} \ldots \xrightarrow{\mathsf{Adam}} (Y_n, X_n)$$

that follows Eve's strategy g, Eve is able to respond with an appropriate move, namely, g is defined on the node (Y_n, X_n). In fact, to prove this property it is sufficient to show that the node (Y_n, X_n) is reachable and this can be easily verified by exploiting an induction on n and the definition of the strategy g given above. We thus conclude that Eve has a winning strategy for the reachability game on $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. □

Quite surprisingly, the above proof also shows that the bounded repair problem in the streaming setting is not sensitive to the notions of transducer with/without lookahead and to the models of aggregate/edit cost. Specifically, from the proof of Theorem 2.3.3 we derive the following corollary.

Corollary 2.3.6. *Given two DFA $\mathcal{R}$ and $\mathcal{T}$, there exists a streaming strategy with some lookahead that repairs $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with uniformly bounded edit cost iff there exists a streaming strategy without lookahead (i.e. 0-lookahead) that repairs $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with edit cost bounded by* $(1 + |\operatorname{dag}(\mathcal{R})|) \cdot |\mathcal{T}|$.

2.4 Complexity analysis

In this section we analyse the complexity of the bounded repair problem and the threshold problem. We distinguish, in both cases, between the non-streaming and streaming settings and between NFA- and DFA-based specifications of regular languages. We will also consider special cases where the restriction or the target languages are fixed, and when the restriction language is universal (i.e. no restriction). Throughout this section we also assume some familiarity with complexity theory (see [Pap94] for a survey in computational complexity theory).

2.4.1 The bounded repair problem in the non-streaming case

NFA vs NFA. Theorem 2.3.1 leads straightforwardly to a PSPACE algorithm that solves the bounded repair problem between two NFA $\mathcal{R}$ and $\mathcal{T}$ in the non-streaming setting: the algorithm first guesses universally [CKS81, Pap94] a path $\pi = X_1 ... X_k$ in $\operatorname{dag}(\mathcal{R})$, then it guesses existentially [CKS81, Pap94] a path $\tau = Y_1 ... Y_k$ of the same length in $\operatorname{dag}^*(\mathcal{T})$, and finally it checks the containment of the sub-automaton $\mathcal{R}|X_i$ in the sub-automaton $\mathcal{T}|Y_i$ for all indices $1 \leq i \leq n$. Together with the PSPACE lower bound proven later in Corollary 2.4.10 (which deals with the case of a universal restriction language), we obtain:

Corollary 2.4.1. *The bounded repair problem in the non-streaming case, where both the restriction and target languages are specified by NFA, is PSPACE-complete.*

NFA vs DFA. The same characterization result can be used to solve the problem when the restriction language is specified by an NFA and the target language is specified by a DFA. In this case, we can take advantage of the determinism to show that the problem turns out to be coNP-complete:

Theorem 2.4.2. *The bounded repair problem in the non-streaming case, where the restriction language is specified by an NFA and the target language is specified by a DFA, is in coNP and it is coNP-hard already for languages specified by DFA.*

Before turning to the proof of the theorem, we establish the following complexity result for the coverability problem:

Lemma 2.4.3. *Given two NFA $\mathcal{R}$ and $\mathcal{T}$ and a path $\pi = X_1 \dots X_k$ in $\mathrm{dag}(\mathcal{R})$, the problem of deciding whether π is covered by some path in $\mathrm{dag}^*(\mathcal{T})$ is in PTIME with an oracle for deciding containment between the languages of the form $\mathscr{L}(\mathcal{R}|X_i)$ and $\mathscr{L}(\mathcal{T}|Y)$, where Y is a SCC of $\mathcal{T}$.*

Proof. The basic idea for checking whether the path π is covered by some path in $\mathrm{dag}^*(\mathcal{T})$ is to incrementally process larger and larger prefixes of π while keeping the frontier of the paths in $\mathrm{dag}^*(\mathcal{T})$ that cover these prefixes. Algorithm 2.4.1 below show the pseudo-code for an algorithm that implements this idea.

Algorithm 2.4.1: PATHCOVERABILITY($\mathcal{R}$, $\mathcal{T}$, π)

```
let π = X_1 ... X_k
let dag*(T) = (V', E')
F ← ∅
for all Y ∈ V'
  do { if CHECKCONTAINMENT(R, T, X_1, Y)
       then F ← F ∪ {Y}
for i ← 2 to k
  do { F' ← F
       F ← ∅
       for all Y' ∈ F' and (Y', D) ∈ E'
         do { if CHECKCONTAINMENT(R, T, X_i, Y)
              then F ← F ∪ {D}
return (F ≠ ∅)
```

Note that the pseudo-code uses CHECKCONTAINMENT$(\mathcal{R}, \mathcal{T}, X_i, D)$ as an oracle for deciding containment between the languages $\mathscr{L}(\mathcal{R}|X_i)$ and $\mathscr{L}(\mathcal{T}|Y)$. The proof of the correctness of the algorithm is straightforward, based on the following invariant: at each iteration of the loop on i, the set F contains a SCC Y of $\mathcal{T}$ iff there is a path $\tau = Y_1 \dots Y_i$ in $\mathrm{dag}^*(\mathcal{T})$, with $Y_i = Y$, that covers $\pi_i = X_1 \dots X_i$. Clearly the described procedure runs in polynomial time with respect to the size of the input NFA $\mathcal{R}$ and $\mathcal{T}$. □

Proof of Theorem 2.4.2. We first prove the complexity upper bound. Let $\mathcal{R}$ be an NFA and $\mathcal{T}$ be a DFA. Thanks to Theorem 2.3.1, deciding whether $\mathsf{cost}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) < \infty$ amounts at first guessing universally a path π in $\mathrm{dag}(\mathcal{R})$ (this can be done in coNP) and then checking whether π is covered by some path in $\mathrm{dag}^*(\mathcal{T})$. By Lemma 2.4.3, this can be done in PTIME using an oracle for deciding the containment of languages recognized by SCCs of $\mathcal{R}$ and $\mathcal{T}$.

What remains to be done is to show that containment of a language $\mathscr{L}(\mathcal{R}|X)$ inside a language $\mathscr{L}(\mathcal{T}|Y)$, where X (resp., Y) is a SCC of $\mathcal{R}$ (resp., $\mathcal{T}$), is decidable in PTIME, even if the successful runs can start from arbitrary states in $\mathcal{T}|Y$. For this we use Lemma 2.3.4, which reduces the containment problem $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\mathcal{T}|Y)$ to a containment problem between a non-deterministic

sub-automaton of $\mathcal{R}$ (i.e., $\mathcal{R}|_q X$ for some $q \in X$) and a *deterministic* sub-automaton of $\mathcal{T}$ (i.e., $\mathcal{T}|_r Y$ for some $r \in Y$). Since deterministic automata can be easily complemented, the latter containment problem can be decided in polynomial time by reducing to an emptiness problem involving the intersection of two automata (i.e., $\mathcal{R}|_q X$ and the complement of $\mathcal{T}|_r Y$).

Putting everything together, we have that the bounded repair problem in the non-streaming case, where the restriction language is given by an NFA and the target language is given by a DFA, is in coNP.

We now prove the lower bound, which follows from a reduction from the validity problem for propositional formulas in disjunctive normal form (i.e., the dual of the SAT problem [Pap94]). The general idea is to encode in the restriction language all the possible valuations for the propositional variables and then restrict the target language to consist only of encodings of valuations that satisfy at least one clause of the formula. We further allow some redundancy in the encodings of the valuations in order to forbid the repair strategy from modifying the encoded valuations.

We consider a set $X = \{x_1, \ldots, x_k\}$ of propositional variables and we denote by $L = \{x_1, \ldots, x_k\} \cup \{\neg x_1, \ldots, \neg x_k\}$ the corresponding set of literals. For the sake of brevity, we identify $\neg\neg x_i$ with x_i. A valuation for the variables in X can be viewed as a subset V of L such that, for every literal $l \in L$, we have $l \in V$ iff $\neg l \notin V$. Let us consider a formula in disjunctive normal form

$$\varphi \;=\; \bigvee_{1 \leq i \leq m} \bigwedge_{1 \leq j \leq h_i} l_{i,j}$$

with $l_{i,j} \in L$ for every pair of indices $1 \leq i \leq m$ and $1 \leq j \leq h_i$. Below, we describe suitable restriction and target languages R and T such that R can be repaired into T with uniformly bounded cost iff φ is valid, namely, if for every valuation V, there is an index $1 \leq i \leq m$ such that $l_{i,1}, \ldots, l_{i,h_i} \in V$.

We define the restriction language over the alphabet $\Sigma = L$ to be:

$$R \;=^{\text{def}}\; (\{x_1\}^* \cup \{\neg x_1\}^*)\ (\{x_2\}^* \cup \{\neg x_2\}^*)\ \ldots\ (\{x_k\}^* \cup \{\neg x_k\}^*).$$

Note that it is easy to construct a DFA $\mathcal{R}$ that recognizes R and that has size polynomial in the number of variables used by φ. Similarly, we define the target language over the alphabet $\Delta = \{a_1, \ldots, a_m\} \cup L$ to be:

$$T \;=^{\text{def}}\; \bigcup_{1 \leq i \leq m} \{a_i\}\ (L \setminus \{\neg l_{i,1},\ \ldots,\ \neg l_{i,h_i}\})^*.$$

Again, it is easy to construct a DFA $\mathcal{T}$ that recognizes T and that has size linear in the number of clauses of φ and in the number of its variables.

We verify that R can be repaired into T with uniformly bounded cost iff φ is valid. For the right-to-left implication, suppose that φ is valid and let u be a word in R. Clearly, u is of the form $(l_1 \ldots l_1)\ (l_2 \ldots l_2)\ \ldots\ (l_k \ldots l_k)$, with $l_i \in \{x_i, \neg x_i\}$ for all $1 \leq i \leq k$. Such a word encodes the valuation $V_u = \{l_1, l_2, \ldots, l_k\}$. Moreover, since φ is valid, there is an index $1 \leq i \leq m$ such

that $l_{i,1}, \ldots, l_{i,h_i} \in V$. The repair strategy f for R and T could then map the word u to the word $f(u) = a_i\ u$, which clearly belongs to T. As for the converse implication, we assume that φ is not valid. This means that there is a valuation $V = \{l_1, l_2, \ldots, l_k\}$, with $l_i \in \{x_i, \neg x_i\}$ for all $1 \leq i \leq k$, such that for every index $1 \leq i \leq m$, there is an index $1 \leq j \leq h_i$ satisfying $l_{i,j} \notin V$. We then consider the family of words $u_n = (l_1)^n\ (l_2)^n\ \ldots\ (l_k)^n$. We know that for every $1 \leq i \leq m$, there is $1 \leq j \leq h_i$ such that the edit distance between the sub-word $(\neg l_{i,j})^n$ of u_n and any word in the sub-language $T_i = \{a_i\}\ (L \setminus \{\neg l_{i,1},\ \ldots,\ \neg l_{i,h_i}\})^*$ of T is at least n. This shows that any repair strategy of R into T has unbounded cost. □

Fixed restriction or target. We briefly outline some parameterized complexity results. Quite surprisingly, if we fix either the restriction language or the target language, then the bounded repair problem in the non-streaming case becomes tractable.

We first consider the case where the restriction automaton is fixed and the target automaton is a DFA provided as input to the problem. Using arguments similar to the previous coNP upper bound (Theorem 2.4.2), one can show that the bounded repair problem between a fixed restriction language and the target language recognized by a given DFA is in PTIME.

Proposition 2.4.4. *Let R be a fixed restriction language. The problem of deciding, given a DFA $\mathcal{T}$, whether $\mathsf{cost}(R, \mathscr{L}(\mathcal{T})) < \infty$ is in PTIME.*

Proof. The proof is similar to the part of the proof of Theorem 2.4.2 related to the coNP upper bound. Let R be a fixed restriction language and let $\mathcal{R}$ be a DFA that recognizes R. Moreover, let $\mathcal{T}$ be a given DFA recognizing the target language T. From Theorem 2.3.1, deciding whether $\mathsf{cost}(R, \mathscr{L}(\mathcal{T})) < \infty$ amounts at checking that every path π in $\mathrm{dag}(\mathcal{R})$ is covered by some path in $\mathrm{dag}^*(\mathcal{T})$. In virtue of Lemma 2.4.3 and Lemma 2.3.4, coverability of a given path in $\mathrm{dag}(\mathcal{R})$ by paths in $\mathrm{dag}^*(\mathcal{T})$ can be decided in PTIME. Since the number of paths in $\mathrm{dag}(\mathcal{R})$ is fixed, this shows that the problem in decidable in polynomial time. □

It is more difficult to show that the bounded repair problem is tractable when we fix the target automaton. In this case, instead of guessing a path π in $\mathrm{dag}(\mathcal{R})$ and then checking whether π is covered by some path in $\mathrm{dag}^*(\mathcal{T})$, we perform a top-down visit of $\mathrm{dag}(\mathcal{R})$ and compute, for each SCC X of $\mathcal{R}$, the set of all SCCs Y of $\mathcal{T}$ such that there exist a path π in $\mathrm{dag}(\mathcal{R})$ that ends in X and a path τ in $\mathrm{dag}^*(\mathcal{T})$ that ends in Y and covers π.

Proposition 2.4.5. *Let T be a fixed target language. The problem of deciding, given an NFA $\mathcal{R}$, whether $\mathsf{cost}(\mathscr{L}(\mathcal{R}), T) < \infty$ is in PTIME.*

Proof. The polynomial-time solution to the bounded repair problem under a fixed target language $T = \mathscr{L}(\mathcal{T})$ uses a dynamic programming approach and the characterization of Theorem 2.3.1 (hence

it is similar to the proof of Theorem 2.4.2). However, instead of guessing a path π in $\mathrm{dag}(\mathcal{R})$ and then checking whether π is covered by some path π' in $\mathrm{dag}^*(\mathcal{T})$, we directly compute a set representing all instances of the coverability relation. More precisely, we compute the set P of all pairs of the form (X, F), where X is a SCC of the NFA $\mathcal{R}$ and $F = \{Y_1, \ldots, Y_n\}$ is a set of SCCs of the DFA $\mathcal{T}$, for which there is a path π in $\mathrm{dag}(\mathcal{R})$ that ends in X and that is covered by n different paths $\tau_1, \ldots, \tau_n$ in $\mathrm{dag}^*(\mathcal{T})$, with each τ_i ending in Y_i. We call such a path π a *witness* of the pair (X, F). Notice that each pair (X, F) is representing a tuple $(X, F, \pi, \tau_1, \ldots, \tau_n)$, but we are blurring all these tuples by using (X, F) in order to have only polynomially many elements in P. Clearly, if $F \neq \varnothing$ for all pairs $(X, F) \in P$, then we know that every path π in $\mathrm{dag}(\mathcal{R})$ is covered by some path in $\mathrm{dag}^*(\mathcal{T})$ (otherwise, since $\mathcal{R}$ is trimmed, we know that there is a path π in $\mathrm{dag}(\mathcal{R})$ that witnesses a pair $(X, \varnothing)$ and hence π is not covered by any path in $\mathrm{dag}^*(\mathcal{T})$). The pairs (X, F) of the set P can be recursively computed by exploiting an induction on the length of the witnessing paths. Algorithm 2.4.2 below implements such an idea.

Algorithm 2.4.2: AllPathsCoverability($\mathcal{R}$, $\mathcal{T}$)

```
let dag(R) = (V, E)
let dag*(T) = (V', E')
P ← ∅
for all X ∈ V
do { F ← ∅
     for all Y ∈ V'
     do { if CheckContainment(R, T, X, Y)
          then F ← F ∪ {Y} }
     P ← P ∪ {(X, F)} }
for k ← 2 to |V|
do { for all (X, F) ∈ P and (X, X') ∈ E
     do { F' ← ∅
          for all Y ∈ F and (Y, Y') ∈ E'
          do { if CheckContainment(R, T, X', Y')
               then F' ← F' ∪ {Y'} }
          P ← P ∪ {(X', F')} }
return (∀ (X, F) ∈ P. F ≠ ∅)
```

The proof that the algorithm is correct, namely, that it terminates successfully iff every path π in $\mathrm{dag}(\mathcal{R})$ is covered by some path in $\mathrm{dag}^*(\mathcal{T})$, relies on the following invariant: at each iteration of the loop on k, the set P contains all pairs (X, F) that are witnessed by a path π in $\mathrm{dag}(\mathcal{R})$ of length at most k. Moreover, we observe that every instruction used in the pseudo-code of the algorithm (including the calls to the subroutine CheckContainment($\mathcal{R}$, $\mathcal{T}$, X, Y)) requires time polynomial in the size of the arguments (recall, for instance, Lemma 2.3.4). Finally, since the set P has size at most $|\mathcal{R}| \times 2^{|\mathcal{T}|}$ and $\mathcal{T}$ is fixed, we conclude that the algorithm runs in polynomial time with respect to the size of $\mathcal{R}$. □

2.4.2 The bounded repair problem in the streaming case

DFA vs DFA. The characterization of Theorem 2.3.3 shows that the problem of deciding the existence of a streaming repair strategy with uniformly bounded (aggregate or edit) cost for two languages specified by DFA $\mathcal{R}$ and $\mathcal{T}$ amounts at solving a reachability game over a suitable (acyclic) arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. In particular, we observe that the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ can be computed from $\mathcal{R}$ and $\mathcal{T}$ in polynomial time, and that checking containment of languages recognized by SCCs of automata is in PTIME. This shows that the bounded repair problem for DFA in the streaming case is in PTIME:

Corollary 2.4.6. *The bounded repair problem in the streaming case, where the restriction and target languages are specified by DFA, is in PTIME.*

NFA vs NFA. Of course, the problem becomes more difficult when the languages are specified by NFA. In this case we are not able to provide tight complexity bounds, and we only claim that the complexity of the bounded repair problem for NFA in the streaming setting is between PSPACE and EXPTIME. The lower bound will follow from Corollary 2.4.10 below and the upper bound from the standard subset construction on NFA (see [HU79] for a survey):

Corollary 2.4.7. *The bounded repair problem in the streaming case, where the restriction and target languages are specified by NFA, is in EXPTIME and it is PSPACE-hard.*

DFA vs NFA. Here we analyze the complexity of the bounded repair problem in the streaming setting where one of the two input automata is a DFA and the other is an NFA. In these cases, we can improve the upper bounds from EXPTIME (Corollary 2.4.7) to PSPACE:

Theorem 2.4.8. *The bounded repair problem in the streaming case, where the restriction language is specified by a DFA and the target language is specified by an NFA, is PSPACE-complete.*
The bounded repair problem in the streaming case, where the restriction language is specified by an NFA and the target language is specified by a DFA, is in PSPACE.

Proof. In both scenarios, we make use of the characterization Theorem 2.3.3.

We first deal with the case where the restriction language is given by a DFA $\mathcal{R}$ and the target language is given by an NFA $\mathcal{T}$. As a preliminary remark, observe that, in this case, PSPACE-hardness will follow again from Corollary 2.4.10 (below). As for the PSPACE upper bound, we denote by $\det(\mathcal{T})$ the DFA obtained from $\mathcal{T}$ by applying the standard subset construction (see [HU79] for more details), and we recall that, by Theorem 2.3.3, there is a streaming repair strategy for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$ $(= \mathscr{L}(\det(\mathcal{T})))$ of uniformly bounded aggregate/edit cost iff Eve wins the reachability game over the arena $\mathcal{G}_{\mathcal{R},\det(\mathcal{T})}$. The crucial observation is that the longest collection of moves of Adam in the arena $\mathcal{G}_{\mathcal{R},\det(\mathcal{T})}$ is linear in the size of $\mathrm{dag}(\mathcal{R})$. This implies that the length of any play

is at most $|\mathrm{dag}(\mathcal{R})|$ and hence we can simulate the reachability game over $\mathcal{G}_{\mathcal{R},\det(\mathcal{T})}$ by an alternating polynomial-time procedure [Pap94]. More precisely, we can keep track of the configuration of the reachability game by maintaining the current SCC X of $\mathcal{R}$ and (a symbolic representation of) the current SCC Y of $\det(\mathcal{T})$ (note that a SCC of $\det(\mathcal{T})$ can be specified by a single state of $\det(\mathcal{T})$ or, equivalently, by a set of states of $\mathcal{T}$). At each round of the reachability game, we need to check a language containment $\mathscr{L}(\mathcal{R}|X) \subseteq \mathscr{L}(\det(\mathcal{T})|Y)$: this can be done using the characterization given in Lemma 2.3.5 and a PSPACE subroutine based on symbolic reachability analysis. What we have described is an alternating polynomial-time procedure that simulates the reachability game over $\mathcal{G}_{\mathcal{R},\det(\mathcal{T})}$ using a PSPACE subroutine for language containment. Overall, the resulting complexity is in PSPACE.

We now turn to the case where the restriction language is given by an NFA $\mathcal{R}$ and the target language is given by a DFA $\mathcal{T}$. As in the previous proof, we have to simulate the reachability game over the arena $\mathcal{G}_{\det(\mathcal{R}),\mathcal{T}}$ that results from the main characterization result. However, we cannot obtain a polynomial bound on the length of the plays since $\mathrm{dag}(\det(\mathcal{R}))$ has potentially exponential height. The idea here is that it is possible to modify the definition of the arena $\mathcal{G}_{\det(\mathcal{R}),\mathcal{T}}$ (and thus the resulting reachability game) by allowing Adam to move down the graph $\mathrm{dag}(\det(\mathcal{R}))$ using shortcuts, namely, by allowing Adam to move from any SCC of $\det(\mathcal{R})$ to some *descendant* of it (rather than simply a successor of it). On the one hand, allowing this freedom in the new reachability game clearly makes it easier for Adam to win. On the other hand, if Adam wins in the modified arena, then he can also win in the original arena via longer plays. We now argue that, if Adam wins the modified reachability game, then he can do so with a polynomial number of moves. Indeed, a winning strategy of Adam consists in pushing Eve towards a sink node; this however can be done in at most n rounds, where n is the height of the DAG of SCCs of $\mathcal{T}$, by properly choosing shortcut moves. The above arguments show that the two versions of the reachability games are equivalent and, furthermore, one can bound the length of the plays to a polynomial in the size of the DFA $\mathcal{T}$. Therefore, the bounded repair problem for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$ in the streaming setting can be solved by an alternating polynomial-time procedure similar to the one described above. This shows that the problem is in PSPACE. □

2.4.3 The bounded repair problem in the unrestricted case

We now consider the unrestricted case of the bounded repair problem, namely, the case where the restriction language is assumed to be Σ^* and the target language T is specified by a finite state automaton.

The following result adapts the characterization theorems presented in Section 2.3 to give a necessary and sufficient condition for bounded repairability in the unrestricted case. This result, which can be viewed as a special case of both Theorem 2.3.1 and Theorem 2.3.3, also shows that

there is no difference between the non-streaming and the streaming settings when the restriction language is universal.

Corollary 2.4.9. *Given an alphabet Σ and an NFA $\mathcal{T}$, the following conditions are equivalent:*

1. $\mathsf{cost}(\Sigma^*, \mathscr{L}(\mathcal{T})) < \infty$
2. *there exists a SCC Y of $\mathcal{T}$ such that $\Sigma^* \subseteq \mathscr{L}(\mathcal{T}|Y)$,*
3. $\mathsf{cost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\Sigma^*, \mathscr{L}(\mathcal{T})) \leq 2|\mathcal{T}|$.

Using the above characterization, one can easily devise an NLOGSPACE algorithm that solves the bounded repair problem for DFA in the unrestricted (streaming or non-streaming) case. Indeed, if the target automaton $\mathcal{T}$ is a DFA and Y is a component of $\mathcal{T}$, then we have $\Sigma^* \subseteq \mathscr{L}(\mathcal{T}|Y)$ iff for every symbol $a \in \Sigma$ and every state $r \in Y$, $\mathcal{T}$ contains a transition of the form (r, a, r'), with $r' \in Y$. Checking this property amounts to performing a standard NLOGSPACE reachability analysis over $\mathcal{T}$. Conversely, NLOGSPACE-hardness follows from the fact that the emptiness problem for DFA is reducible to the bounded repair problem: given a DFA $\mathcal{A}$ over an alphabet Σ, we have that $\mathscr{L}(\mathcal{A}) \neq \varnothing$ iff Σ^* is repairable into $\mathscr{L}(\mathcal{A}')$ with uniformly bounded cost, where $\mathcal{A}'$ is a DFA that recognizes the language $\Sigma^* \cdot \mathscr{L}(\mathcal{A})$ and that can be constructed from $\mathcal{A}$ in logarithmic space.

In a similar way, one can show that the bounded repair problem for NFA in the unrestricted case is PSPACE-complete. This follows from Corollary 2.4.9 and from suitable reductions to/from the universality problem for NFA [SM73]. Indeed, checking whether a target NFA $\mathcal{T}$ has a SCC Y such that $\Sigma^* \subseteq \mathscr{L}(\mathcal{T}|Y)$ is equivalent to the problem of deciding whether Σ^* is repairable into $\mathscr{L}(\mathcal{T})$ with uniformly bounded cost, and it is clearly reducible to the universality problem for NFA. As for the PSPACE-hardness, we observe that a given NFA $\mathcal{A}$ recognizes the universal language Σ^* iff $(\Sigma \cup \{\#\})^*$ is repairable into $(\mathscr{L}(\mathcal{A}) \cdot \{\#\})^*$ with uniformly bounded cost and $\# \notin \Sigma$. Notice that a finite automaton $\mathcal{A}'$ recognizing the language $(\mathscr{L}(\mathcal{A}) \cdot \{\#\})^*$ can be computed in linear time from $\mathcal{A}$.

We thus conclude the following:

Corollary 2.4.10. *The bounded repair problem in the unrestricted case, where the target language is specified by a DFA (resp., NFA) is NLOGSPACE-complete (resp., PSPACE-complete).*

Another consequence of Corollary 2.4.9 is the following. Suppose that a target language T is recognized by a DFA $\mathcal{T}$ that is *complete* over the target alphabet Δ, namely, for every symbol $a \in \Delta$ and every state r of $\mathcal{T}$, $\mathcal{T}$ contains a transition from r labeled by a (here we do not assume that $\mathcal{T}$ is trimmed). Let us consider a restriction alphabet Σ contained in Δ and suppose that Σ^* is *not* repairable into T with uniformly bounded cost. Let us consider a SCC Y of $\mathcal{T}$ that is reachable from the initial state and terminal, namely, with no outgoing edges. We know that Y does not contain any

final state (otherwise, Y would be a final SCC and hence, by Corollary 2.4.9, $\mathsf{cost}(\Sigma^*, \mathscr{L}(\mathcal{T})) < \infty$). In this case, however, the same component Y in the *complement* DFA $\mathcal{T}^{\mathsf{C}}$ would be final and hence Σ^* would be repairable into $\mathscr{L}(\mathcal{T}^{\mathsf{C}})$ $(= \Delta^* \setminus T)$ with uniformly bounded cost. This shows that:

Corollary 2.4.11. *Given an alphabet Σ and a regular language $T \subseteq \Delta^*$, with $\Sigma \subseteq \Delta$, then either $\mathsf{cost}(\Sigma^*, T) < \infty$ or $\mathsf{cost}(\Sigma^*, \Delta^* \setminus T) < \infty$.*

2.4.4 The threshold problem in the non-streaming case

We now consider the problem of calculating the exact repair cost in the non-streaming setting. If the restriction and target languages are specified by DFA, then we know from Theorem 2.4.2 that we can decide in coNP whether the worst-case repair cost is finite or infinite. Furthermore, Theorem 2.3.1 tells us that if the cost is finite it must be bounded by a polynomial in the input size. Thus, to determine the exact repair cost in the case where it is finite, it suffices to test whether the cost is above or below a given threshold ν in unary, since then we can try every number ν below the polynomial bound. Perhaps surprising, this problem is harder than the bounded repair problem, although still within polynomial space:

Theorem 2.4.12. *The problem of determining whether $\mathsf{cost}(R, T) \leq \nu$, given two languages R and T specified by DFA and given a number ν, is PSPACE-complete. The same holds when R and T are specified by NFA.*

Proof. We first give the upper bound, assuming that R and T are recognized by some NFA $\mathcal{R}$ and $\mathcal{T}$. First, we recall that, by Theorem 2.3.1, we have either $\mathsf{cost}(R, T) \leq (1 + |\operatorname{dag}(\mathcal{R})|) \cdot |\mathcal{T}|$ or $\mathsf{cost}(R, T) = \infty$. Thus, without loss of generality, we can assume that the threshold $\nu \in \mathbb{N}$ is represented in unary notation. We can then construct in polynomial time an NFA $\mathcal{R}_\nu$ that recognizes the language

$$R_\nu \;=^{\text{def}}\; \{u' \in \Delta^* \;:\; \exists\, u \in R.\, \mathsf{dist}(u, u') \leq \nu\}.$$

The automaton $\mathcal{R}_\nu$ is defined by a suitable 'juxtaposition' of $\nu + 1$ disjoint copies of $\mathcal{R}$:

- the states of $\mathcal{R}_\nu$ are the pairs (q, i), where q is a state of $\mathcal{R}$ and $0 \leq i \leq \nu$;
- the transitions of $\mathcal{R}_\nu$ are the triples $\big((q, i), b, (q', j)\big)$ that satisfy one of the following conditions
  1. $i = j$, $b \in \Sigma$, and (q, b, q') is a transition of $\mathcal{R}$,
  2. $i < j$, $b \in \Delta$, and $q = q'$,
  3. $i < j$, $b \in \Delta$, and there is a transition (q, a, q') of $\mathcal{R}$, with $a \in \Sigma$,
  4. $i < j$, $b \in \Sigma$, and there are two transitions (q, a, q'') and (q'', a', q') of $\mathcal{R}$, with $a, a' \in \Sigma$ and $b \in \{a, a'\}$;

- the initial (resp., final) states of $\mathcal{R}_\nu$ are the pairs (q, i), where q is an initial (resp., final) state of $\mathcal{R}$ and $0 \le i \le \nu$.

Note that the automaton $\mathcal{R}_\nu$ can be constructed in polynomial time from $\mathcal{R}$ and ν when ν is represented in unary. Moreover, $\mathcal{R}_\nu$ accepts all and only the words that are at distance at most ν from some words in R, and hence $\mathscr{L}(\mathcal{R}_\nu) = R_\nu$. Finally, it is easy to see that

$$\mathsf{cost}(R, T) \le \nu \qquad \text{iff} \qquad R_\nu \subseteq T.$$

This reduces the threshold problem $\mathsf{cost}(R, T) \le \nu$ to a containment problem between languages recognized by NFA, which is known to be in PSPACE [SM73].

We now discuss the PSPACE lower bound, which holds even for languages specified by DFA. It is via reduction from the problem of tiling a corridor of polynomial width (and unbounded height). An instance of the latter problem is given by a number n (i.e., the width of the corridor, represented in unary notation), a set S of available tiles, some sets $H, V \subseteq S \times S$ of vertical and horizontal constraints, and two tiles $t_\perp$ and $t_\top$ for the bottom and top rows. A *tiling of height* N (for an instance $I = (n, S, H, V, t_\perp, t_\top)$) is a mapping g from the pairs $(i, j) \in [1, N] \times [1, n]$ to the tiles in S such that $g(1, j) = t_\perp$ and $g(N, j) = t_\top$ for all $1 \le j \le n$. We say that the tiling g *satisfies the constraints* of I if

1. $\big(g(i, j), g(i, j+1)\big) \in H$ for all $1 \le i \le N$ and all $1 \le j < n$,
2. $\big(g(i, j), g(i+1, j)\big) \in V$ for all $1 \le i < N$ and all $1 \le j \le n$.

The corridor tiling problem is the problem of deciding, given a tiling instance $I = (n, S, H, V, t_\perp, t_\top)$, whether there is a tiling g of some height $N \ge 1$ that satisfies the constraints in I. It is known (see, for instance, [Boa97]) that this problem is PSPACE-complete.

Hereafter, we fix an instance $I = (n, S, H, V, t_\perp, t_\top)$ of the corridor tiling problem and a threshold $\nu \ge 1$ (note that for $\nu = 0$ the threshold problem becomes a containment problem between DFA, which is clearly solvable in PTIME). Moreover, we assume that the threshold ν is represented in unary notation (clearly this does not make the threshold problem more difficult). Below, we reduce the corridor tiling problem for the instance I to a threshold problem for two DFA $\mathcal{R}$ and $\mathcal{T}$.

We let the restriction alphabet Σ consist of pairs of the form $\langle t, j \rangle$, where t is a tile from S and j is a number in $\{1, \ldots, n\}$. The restriction language R contains "$(\nu+1)$-redundant" encodings of tilings, namely, words of the form

$$\langle g(1,1), 1\rangle^{\nu+1} \ \ldots\ \langle g(1,n), n\rangle^{\nu+1} \ \ldots\ldots\ \langle g(N,1), 1\rangle^{\nu+1} \ \ldots\ \langle g(N,n), n\rangle^{\nu+1}$$

where N is a positive natural number and $g : [1, N] \times [1, n] \to S$ is a tiling of height N that satisfies the horizontal constraints and the constraints on the bottom and top rows (but possibly not the

vertical constraints). We claim that the above language is recognized by a DFA $\mathcal{R}$ of size polynomial in I and in ν: indeed, the automaton $\mathcal{R}$ needs to check that the input word is well-formed, which requires enforcing the horizontal constraints and the initial and final tile requirements (which clearly can be done with a fixed number of states) and counting up to $\nu+1$ and n (which clearly can be done with a number of states linear in ν and n).

We now turn to the definition of the target language. Its alphabet Δ contains all symbols of the restriction alphabet Σ plus marked symbols $\underline{\langle t,j\rangle}$, with $t \in S$ and $1 \le j \le n$. The marked symbols $\underline{\langle t,j\rangle}$ are used to highlight a violation of the vertical constraints in such a way that it becomes easy for an automaton to certify it. Intuitively, a violation of a vertical constraint is certified on a word $v \in \Delta^*$ when v contains a marked symbol of the form $\underline{\langle t,j\rangle}$ at some position x and a symbol $\langle t',j\rangle$, with $(t,t') \notin V$, at position $x+(\nu+1)\cdot n$ (this would imply that the tile t' is adjacent to t along the vertical axis). More precisely, the target language is defined as follows:

$$T \;=^{\text{def}} \bigcup_{\substack{(t,t')\notin V \\ 1\le j\le n}} \Sigma^* \; \underline{\langle t,j\rangle} \; \underbrace{\langle t,j\rangle^{\nu} \; \Sigma^{(\nu+1)\cdot(n-1)} \; \langle t',j\rangle}_{(\nu+1)\cdot n \text{ positions}} \; \langle t',j\rangle^{\nu} \; \Sigma^*.$$

It is easy to construct a DFA $\mathcal{T}$ of size polynomial in $|I|$ and ν that recognizes the language T.

We now argue that there there is a repair strategy for R into T with cost uniformly bounded by ν iff every tiling $g : [1,N] \times [1,n] \to S$ violates the constraints in I. On the one hand, suppose that every tiling $g : [1,N] \times [1,n] \to S$ violates the constraints in I. Let us consider a word $u \in R$ of the form

$$\langle g(1,1),1\rangle^{\nu+1} \; \ldots \; \langle g(1,n),n\rangle^{\nu+1} \; \ldots\ldots \; \langle g(N,1),1\rangle^{\nu+1} \; \ldots \; \langle g(N,n),n\rangle^{\nu+1}$$

where g is a tiling of $[1,N] \times [1,n] \to S$. From the previous assumptions, we know that g violates the vertical constraints in I. This implies that there are $1 \le i < N$ and $1 \le j \le n$ such that $\big(g(i,j), g(i+1,j)\big) \notin V$, and hence there are two positions x $(= j+(i-1)\cdot(\nu+1)\cdot n)$ and $x+(\nu+1)\cdot n$ $(= j+(i-1)\cdot(\nu+1)\cdot n+(\nu+1)\cdot n)$ in u that contain the symbols $\langle g(i,j),j\rangle$ and $\langle g(i+1,j),j\rangle$, respectively. Marking the first of these two occurrences (i.e., replacing the symbol $\langle g(i,j),j\rangle$ at position x with the symbol $\underline{\langle g(i,j),j\rangle}$) will bring us into the target language T. This shows that there is a repair strategy for R and T with worst-case cost at most 1 $(\le \nu)$.

On the other hand, suppose that there is a tiling $g : [1,N] \times [1,n] \to S$ that satisfies the constraints in I. Let u be the corresponding word:

$$\langle g(1,1),1\rangle^{\nu+1} \; \ldots \; \langle g(1,n),n\rangle^{\nu+1} \; \ldots\ldots \; \langle g(N,1),1\rangle^{\nu+1} \; \ldots \; \langle g(N,n),n\rangle^{\nu+1}.$$

Clearly, u belongs to the restriction language R. We claim that no matter how we repair u by at most ν edits, the resulting sequence will not belong to the target language T. Consider a word v produced by such an edit. If there are no occurrences of marked symbols in v, then v cannot belong to T, and similarly if there are more than one occurrence of a marked symbol. Suppose that v contains

exactly one occurrence of a marked symbol, say $v[x] = \underline{\langle t, j\rangle}$, and let $v[x + (\nu+1)\cdot n] = \langle t', j'\rangle$. We distinguish between the following cases:

1. $v[x+i] \neq \langle t, j\rangle$ for some $1 \leq i \leq \nu$,
2. $j' \neq j$ or $v[x + (\nu+1)\cdot n + i] \neq \langle t', j'\rangle$ for some $1 \leq i \leq \nu$,
3. $j' = j$ and $v[x+i] = \langle t, j\rangle$ and $v[x + (\nu+1)\cdot n + i] = \langle t', j\rangle$ for all $1 \leq i \leq \nu$.

In the first two cases, v cannot belong to T. In the third case, since v was obtained from u by applying at most ν edit operations, we know that u contains a sub-string of the form

$$\langle t, j\rangle^{\nu+1}\ \Sigma^{(\nu+1)\cdot(n-1)}\ \langle t', j\rangle^{\nu+1}.$$

Since u is an encoding of a valid tiling g, we have that $(t, t') \in V$. This shows again that v cannot belong to the target language T. □

2.4.5 The threshold problem in the streaming case

For the streaming setting, if we consider 0-lookahead repair strategies with aggregate cost, the threshold problem becomes solvable in polynomial time (hence it is as difficult as the bounded repair problem). Indeed, one can easily reduce this threshold problem to a reachability game over a suitable arena that is constructed in polynomial time from the restriction and target DFA $\mathcal{R}$ and $\mathcal{T}$ and from the threshold ν. It follows from this reduction that one can efficiently compute in polynomial time a streaming repair strategy for $\mathcal{R}$ and $\mathcal{T}$ whose aggregate cost does not exceed the threshold ν.

Theorem 2.4.13. *The problem of determining whether* $\mathsf{cost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(R, T) \leq \nu$, *given two languages* R *and* T *specified by DFA and given a numbers* ν, *is in PTIME.*
Moreover, if $\mathsf{cost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(R, T) \leq \nu$, *then one can compute in polynomial time a streaming* 0*-lookahead repair strategy for* R *and* T *that has aggregate cost at most* ν.

Proof. Let $\mathcal{R} = (\Sigma, Q, \delta, q_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', r_0, F')$ be two DFA, let R and T be the recognized languages, and let ν be the threshold for the aggregate cost. As a preliminary remark, we observe that, without loss of generality, we can assume that ν is represented in unary: indeed, we know from Theorem 2.3.3 and Corollary 2.3.6 that either there is a streaming repair strategy for R and T with aggregate cost uniformly bounded by $(1 + |\operatorname{dag}(\mathcal{R})|)\cdot|\mathcal{T}|$ (i.e., a polynomial in the size of $\mathcal{R}$ and $\mathcal{T}$), or all streaming repair strategies for R and T have unbounded aggregate cost.

We define a reachability game over an arena $\mathcal{G}^{\nu}_{\mathcal{R},\mathcal{T}}$ that characterizes the threshold problem for $\mathcal{R}$ and $\mathcal{T}$ in the streaming case. The nodes of the arena are the pairs (q, r, c) and (q, r, c, a), with $q \in Q$, $r \in R$, $c \in \{0, \ldots, \nu\}$, and $a \in \Sigma$. The former nodes are owned by Adam (i.e., the player entitled to emit a word in the restriction language) and the latter nodes are owned by Eve (i.e.,

the player entitled to repair the given word into the target language). The arena $\mathcal{G}^{\nu}_{\mathcal{R},\mathcal{T}}$ has an edge $(q,r,c) \to (q',r,c,a)$ if $\delta(q,a) = q'$, and it has an edge $(q,r,c,a) \to (q,r',c')$ if r' is reachable from r in $\mathcal{T}$ and $c' = c + \min\{\mathsf{dist}(a,v) : v \in \mathscr{L}(\mathcal{T}_{r,r'})\}$ (provided that $c' \leq \nu$). Adam plays first, starting from the node $(q_0, r_0, 0)$. The player who cannot move loses. Infinite plays, which are feasible in this type of game, are won by Eve.

Now, we show that Eve has a winning strategy in $\mathcal{G}^{\nu}_{\mathcal{R},\mathcal{T}}$ iff there is a streaming 0-lookahead repair strategy for R and T with aggregate cost at most ν. Easily, assume that Eve has a strategy f for winning the reachability game over $\mathcal{G}^{\nu}_{\mathcal{R},\mathcal{T}}$. Without loss of generality, we can assume that the strategy f is positional. It is then easy to construct, using Eve's positional strategy f, a transducer $\mathcal{Z}$ with at most $|Q| \cdot |Q'| \cdot (\nu + 1)$ states that repairs R into T with aggregate cost $\mathsf{cost}^{\mathsf{aggr}}_{\mathcal{Z}}(R,T) \leq \nu$. For the other direction, suppose that there is a transducer $\mathcal{Z} = (\Sigma, \Delta, Z, \kappa, z_0, \Omega)$ that repairs every word from $\mathcal{R}$ into $\mathcal{T}$ with aggregate cost less than or equal to ν. It is straightforward to define a winning strategy for Eve using the transducer $\mathcal{Z}$. Indeed, we only need to maintain the current state s of $\mathcal{Z}$ and move from each node (q,r,c,a) to a successor node (q,r',c') such that $r' = \delta(r,v)$, $c' = c + \mathsf{dist}(a,v)$, and $\kappa(z,a) = (v,z')$.

To conclude the proof we need show that, assuming $\mathsf{cost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(R,T) \leq \nu$, one can compute in polynomial time a streaming 0-lookahead repair strategy for R and T that has aggregate cost at most ν. However, this follows immediately from the fact that (i) given the arena $\mathcal{G}^{\nu}_{\mathcal{R},\mathcal{T}}$ whose reachability game is won by Eve, one can construct in polynomial time a winning positional strategy f for Eve, and (ii) using Eve's winning strategy f, one can construct in polynomial time a repair strategy $\mathcal{Z}$ for R and T such that $\mathsf{cost}^{\mathsf{aggr}}_{\mathcal{Z}}(R,T) \leq \nu$ (for this we use again the above arguments). □

Note that in the above result we deal with the model of *aggregate* cost, which is very different the model of edit cost (the main characterization result, however, shows that one is uniformly bounded iff the other is). We do not know if finding the exact *edit* cost for two languages specified by DFA is even tractable. Similarly, we do not know whether the threshold problem for streaming repair strategies with *k-lookahead* is tractable, where the parameter k is represented in binary.

2.5 Connections to distance automata and games

Both non-streaming and streaming repair problems correspond to special cases of prior problems studied in automata and games. The non-streaming bounded repair problem corresponds to the *limitedness problem* for distance automata, while the streaming variant corresponds to *energy games*. We explain the correspondences in detail now. In each case, however, we find that the results for the more general framework do not give tight complexity bounds.

Non-streaming repairs and distance automata. Intuitively, a *distance automaton* is a transducer $\mathcal{D}$ that receives as input a finite word u and outputs a corresponding cost $\mathcal{D}(u)$ in

$\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$. Formally, a distance automaton is a transducer of the form $\mathcal{D} = (\Sigma, Q, E, I, F)$, where Σ is the input alphabet, Q is a finite set of states, $E \subseteq Q \times \Sigma \times \mathbb{N}^\infty \times Q$ is the transition relation, and $I, F : Q \to \mathbb{N}^\infty$ are the initial and final cost functions. The cost $\mathcal{D}(u)$ on input $u = a_1 \dots a_n \in \Sigma^*$ is obtained by taking the minimum among the costs of the runs of $\mathcal{D}$ on u, where the cost of a run $q_0 \xrightarrow{a_1/c_1} q_1 \xrightarrow{a_2/c_2} \dots \xrightarrow{a_n/c_n} q_n$ is defined as $I(q_0) + \sum_{i=1}^{n} c_i + F(q_n)$. We let $\mathcal{D}(u) = \infty$ if $\mathcal{D}$ admits no successful run on w.

The main problem that has been studied for distance automata is the *limitedness problem*, which consists of deciding whether the cost function computed by a given distance automaton $\mathcal{D}$ is uniformly bounded on all words $u \in \Sigma^*$ with cost $\mathcal{D}(u) < \infty$. This problem was shown decidable by Hashigushi [Has90] and later in [LP04] was shown to be PSPACE-complete. Distance automata have been related to edit-distance problems in several prior works – see Section 2.6 for further discussion of the connections. Here we note only a simple reduction of the bounded repair problem to limitedness.

Given two NFA $\mathcal{R} = (\Sigma, Q, E, I, F)$ and $\mathcal{T} = (\Delta, Q', E', I', F')$, one can construct a distance automaton $\mathcal{D}$ that computes the cost of repairing any word from $\mathscr{L}(\mathcal{R})$ into a word from $\mathscr{L}(\mathcal{T})$. First of all, one associates with each symbol $a \in \Sigma$ a matrix $M(a)$ whose entries $M(a)[p, q]$ are indexed over the pairs of states p, q of $\mathcal{T}$ and give the minimum edit-distance between the symbol a and a word $v \in \Delta^*$ such that $\mathcal{T}$ can move from p to q consuming v (if q is not reachable from p, then one simply lets $M(a)[p, q] = \infty$). One then defines the distance automaton $\mathcal{D}$ as the quadruple $(\Sigma, Q \times Q', E^M, I^M, F^M)$, where E^M is the set of all transitions of the form $\big((p, p'), a, c, (q, q')\big)$, with $a \in \Sigma$, $(p, a, q) \in E$, and $c = M(a)[p', q']$. The initial cost function I^M of $\mathcal{D}$ is defined by letting $I^M(p')$ be the length of the minimum word that can be parsed by $\mathcal{T}$ when moving from an initial state to the state p' (if no such word exist, then $I^M(p') = \infty$). Similarly, $F^M(p)$ is defined to be the length of the minimum word that can be parsed by $\mathcal{T}$ when moving from the state p to a final state (or ∞ if no such word exists). It is easy to see that the cost function computed by $\mathcal{D}$ maps a word $u \in \mathscr{L}(\mathcal{R})$ to the cost of an optimal non-streaming repair of u into $\mathscr{L}(\mathcal{T})$. Moreover, the distance automaton $\mathcal{D}$ has size polynomial in the size of $\mathcal{R}$ and $\mathcal{T}$. This reduces the bounded repair problem for NFA in the non-streaming case to the limitedness problem for distance automata (see Section 3.2 in Chapter 3 for more details of this construction). Combining this reduction with the PSPACE upper bound for the limitedness problem, we see that the bounded repair problem for NFA is in PSPACE.

The same reduction technique can be applied to solve the bounded repair problem for DFA. In this case, however, the resulting complexity bound is not optimal: the bounded repair problem for DFA is in fact in coNP (cf. Theorem 2.4.2). Roughly speaking, the reason why the bounded repair problem for DFA is easier than the limitedness problem for distance automata is that the distance automata emerging from bounded repair problems have a more restricted structure (specifically, they are

deterministic on the 0-cost moves). In addition to not giving tight bounds, approaches via distance automata give less insight into the problems. We invite the reader, for example, to compare the PSPACE upper bound that we derive from our characterization of bounded repairability, Theorem 2.3.1, with the PSPACE upper bound given in [LP04].

Streaming repairs and energy games. Just as non-streaming repair problems can be seen within the framework of distance automata, bounded repair problems in the streaming setting are special cases of games on graphs with quantitative objectives. An interesting family of such games is that of *energy games* studied in [CDAHS03], which are played on finite weighted arenas. The game is played between an energy player, who wants to keep the running sum of the weights (i.e. the energy) always positive, and her opponent. A variant of energy games allows the parameterization by an initial credit of energy; the higher the credit the more possibility for the energy player to win.

It is well known that the problem of determining whether there is a finite initial credit so that the energy player has a winning strategy is in NP $\cap$ coNP [CD12], but the exact complexity is still unknown. Furthermore, this problem can be solved in time polynomial in the size of the arena and the largest weight in absolute value. As a matter of fact, the latter complexity result implies that energy games can be solved in polynomial time with respect to the size of the arena, provided that the weights are represented in unary.

One can easily reduce the bounded repair problem in the streaming setting, under the aggregate cost model for languages recognized by DFA, to the finite initial credit problem for energy games. Informally, the choice of the opponent in the energy game corresponds to the letters emitted by the restriction, while the edits correspond to choices of the energy player. More formally, we have a node in the arena for each pair of states of the restriction DFA $\mathcal{R}$ and of the target DFA $\mathcal{T}$ – call this node a *Restriction Player Node*. We also have a node for each combination of restriction state, target state, and letter played – call this a *Target Player Node*. The former represents the pair of states reached by the restriction and target automata after parsing the unedited and edited words, respectively; the latter adds the last letter emitted by the restriction. There is an edge of weight 0 going from a Restriction Player Node (p, p') to any Target Player Node (q, p', a) whenever (p, a, q) is a valid transition of $\mathcal{R}$. Similarly, there is an edge of weight $-c$ going from a Target Player Node (q, p', a) to a Restriction Player Node (q, q') whenever there is a word v at distance c from a (i.e. $\mathsf{dist}(a, v) = c$) such that $\mathcal{T}$ can move from p' to q' consuming v. Clearly, the energy player wins the game using some initial credit of energy if and only if the cost of repairing $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ is uniformly bounded. Note that this reduction provides a PTIME upper bound on the complexity of the bounded streaming repair problem for DFA, given that the size of the resulting arena is polynomial in the size of the restriction and target DFA and, moreover, the weights are bounded by the size of the target DFA.

R \ T	Fixed	DFA	NFA
Universal	LOGSPACE	NLOGSPACE	PSPACE
Fixed	Const	PTIME	PSPACE
DFA	PTIME	coNP	PSPACE
NFA	PTIME	coNP	PSPACE

Table 2.1: Bounded repair problem in the non-streaming case

Our characterization result (see Theorem 2.3.3) gives analogous (tight) complexity bounds for bounded repairability languages recognized by DFA and, moreover, it proves that the bounded repair problem in the streaming setting is not sensitive to the models of aggregate/edit cost. They also provide tight bounds for special cases of the problem, which cannot naturally be captured in the setting of energy games. It is also worth mentioning that the repair strategy that arises from our characterization result can be seen as a special case of the notion of good-for-energy strategy, which is introduced in [CD12] to solve energy parity games.

Despite the connections mentioned above, many concepts and problems concerning repair do not have natural analogs in the game setting, and vice versa. For instance, in the game setting one could allow lookahead for one player, but it is not as natural as in the repair setting. Moreover, while the aggregate cost metric fits the game setting naturally, our usual cost function does not. Conversely, the binary weights that are allowed in the game setting have no natural analog in the context of edits. Our characterization finally allows us to easily isolated special cases of lower complexity that are not easily seen from the embedding into energy games.

2.6 Conclusions

In this chapter we have investigated the problem of repairing documents between different regular specifications. We gave a characterization of those pairs of regular languages such that one can repair any string in the first language to a string in the second, using a uniformly bounded number of edits and repair processors that either read the entire string offline and then edit, or have the form of sequential real-time transducers. We then used the characterizations to provide complexity bounds for the considered bounded repair problems. These complexity bounds are summarized in Table 2.1 and Table 2.2 – in the non-streaming setting the bounds are tight (indicated by a single class), while in the streaming setting we have several gaps (where a cell gives lower and upper bounds). We omit the corresponding table for computing the exact cost: in the case of non-streaming repair we can derive tight bounds in all cases, and also in the case of streaming repair for aggregate cost. In the latter case we also know the complexity of computing the optimal streaming repair processor.

R \ T	**Fixed**	**DFA**	**NFA**
Universal	LOGSPACE	NLOGSPACE	PSPACE
Fixed	Const	PTIME	PSPACE
DFA	PTIME	PTIME	PSPACE
NFA	PTIME-PSPACE	PTIME-PSPACE	PSPACE-EXPTIME

Table 2.2: Bounded repair problem in the streaming case

Our characterizations of bounded repairability (Theorem 2.3.1 and 2.3.3) strongly use the fact that languages are *positively* defined by a specification (in our setting by traces of an NFA). It would be interesting to study the bounded repair problem when the restriction or target languages are given by finite automata with universal transitions or, more general, by alternating finite automata [BL80, CKS81]. In both cases, it is not clear how to extend our characterization to these classes of specifications and how to derive tight complexity bounds. We left this as an open problem.

Related Work. The problem of finding the minimal distance of a string to a regular language was first considered by Wagner in [Wag74], who showed that the problem could be solved by adapting the dynamic programming approach, giving a polynomial time algorithm. Several authors have extended the definition to deal with distances between languages. Mohri [Moh03] defines a distance function between two languages, and more generally between string distributions: in the case of languages, this is the minimum distance between two strings in the two respective languages, which is appropriate for many applications. Konstantinidis [Kon07] focuses on the minimum distance between distinct strings within the same language, giving tractable algorithms for computing it. Our notion of "distance" is quite distinct from this, since it is asymmetric in the two languages, focusing on the maximum of the distance of a string in one language to the other language.

Grahne and Thomo [GT04] consider a related problem of "approximate containment" of regular expressions. Expressions are evaluated with respect to an edge-labeled graph and are given a numerical semantics by a "distortion", a generalization of the notion of edit distance. Approximate containment of T_1 and T_2 means, roughly speaking, that for every input graph R and every word w generated by R, the distance to target T_1 is bounded by the distance to T_2. Grahne and Thomo also study "k-containment" (distance to T_1 is at most k more than T_2) and "approximate-containment" (k-containment for some k), relying primarily on a reduction to the limitedness problem for distance automata. Their problem differs in several fundamental respects from ours: they are interested in bounding the difference over all words, not just the worst-case; in addition, they quantify over all restrictions (databases, in their terminology).

An entire line of research in XML data management has dealt with comparisons and matching algorithms between schema languages. Many of these lift edit distance between trees to the level

of schemas (i.e. languages) – see, for example, [DR02]. However, in those cases the lifting is done by looking at the syntactic structure of the schema description, rather than at the instance level (distance between documents in each schema).

Property testing [AKNS01, Fis01, FMDR10] is a new line of research that studies restricted randomized algorithms for model-checking problems. Specifically, for a property L and an integer ϵ they want to study whether there exists a randomized algorithm (called ϵ-*tester*) which always accepts a structure w if w satisfies L and rejects it with high probability (namely, with probability strictly greater than $1/2$) if w has to be modified in at least $\epsilon \cdot |w|$ to satisfy L. Furthermore, the randomized algorithm is restricted to query at most $O(1/\epsilon)$ *positions* of w. In the particular case when L is a regular language and w is a string, an ϵ-tester always accepts when $w \in L$ and rejects with high probability when $\mathsf{dist}(w, L) \geq \epsilon \cdot |w|$. This last problem over regular languages was studied in [AKNS01] where a first ϵ-*tester* was proposed. One can easy note that the existence of an ϵ-tester is different from the bounded repair problem. However, some connections can be done between both problems. For example, if Σ^* is bounded repairable into L (with $L \subseteq \Sigma^*$) then this trivially implies that every ϵ-tester never rejects with high probability for words w with length greater than $2 \cdot |\mathcal{A}|/\epsilon$ where $\mathcal{A}$ is a finite automata that accepts L. Just notice that $\mathsf{dist}(w, L) \leq 2 \cdot |\mathcal{A}|$ whenever Σ^* is bounded repairable into L (Corollary 2.4.9). Thus, if an ϵ-tester wants to reject with high probability when $\mathsf{dist}(w, L) \geq \epsilon \cdot |w|$, then w has to satisfy $|w| \leq 2 \cdot |\mathcal{A}|/\epsilon$ which is only fulfilled by a finite number of words in L. This implies that if Σ^* is bounded repairable into L then L is trivially ϵ-*testable* for every $\epsilon \geq 0$. We think that this connection with the bounded repair problem is of special interest if property testing wants to be used in practice.

Chapter 3

Asymptotic repairing of strings

In the previous chapter, we considered the basic question of whether one can get from a string in R to a string in T with a finite, uniformly bounded number of edits. Such a bound, when it exists, shows that the language R is "quite close to being a subset of T" – the gap between strings in R and strings in T is small. However, having a uniform bound on the number of edits is a strong requirement. In this chapter we look not at the absolute number of edits required to get from R to T, but rather at the *percentage* of letters that need to be edited. More specific, we measure the gap from a restriction language R to a target language T via the worst case, over all strings $u \in R$, of the number of edits needed to bring u into T divided by the length of u. Since we want the definition to be robust to a finite number of outliers, we take the limit of this quantity as the strings are of larger and larger length – this is the *asymptotic repair cost* in getting from R to T. This gives us a measure of the distortion needed to get from R to T, lying always between 0 and 1.

In Chapter 2 we have given algorithms for determining when the absolute cost of repairing R into T is uniformly bounded. Similarly, the main result of this chapter is an algorithm that computes the asymptotic cost of repairing R into T. The techniques used for the asymptotic cost analysis relies on the connection of this problem with the theory of *distance automata* [Sim88] (presented in Section 2.5), and in particular on an application of determinization of distance automata, closely related to Mohri's determinization procedure [Moh97].

Similarly to the bounded repair problem, we also study the asymptotic repair cost in the setting where every repair is required to be done in streaming fashion, producing the edits immediately on seeing the input letter. We measure a streaming repair processor by the number of edits per character it requires to get from any string in R to a string in T, again looking at the limit as the string length gets large. We accordingly define the *streaming asymptotic cost* to be the optimal cost of a streaming processor. Interestingly, we show that this quantity can also be calculated effectively, using techniques from mean-payoff games.

This chapter is structured as follows: in Section 3.1 we define the basic problem, namely, the asymptotic repair cost between two regular languages. Then in Section 3.2 we study the problem

of computing the asymptotic cost in the non-streaming case, while in Section 3.3 we deal with the streaming case. Towards the end of the chapter (Section 3.4), we give some concluding remarks and related work.

3.1 Asymptotic repair cost

In the previous chapter, we study the pairs (R, T) of regular languages for which there exist a repair strategy f whose absolute cost is finite and uniformly bounded (see Section 2.1 for the definition of regular languages and regular specifications). Since the worst case of this is often infinite, we will not always concern with the absolute cost of repairing a word. Instead, we consider a notion of repair cost between words that looks at the *percentage* of symbols in a word that need to be edited. More precisely, given a repair strategy f for R and T and given a word $u \in R$, we define the *normalized cost of f on u* as the ratio between the cost of f on u and the length of u (see Section 2.2 for the definition of repair strategy and $\mathsf{cost}(u, f)$):

$$\mathsf{ncost}(u, f) \;=^{\text{def}}\; \frac{\mathsf{cost}(u, f)}{|u|}.$$

In order to measure the asymptotic behavior of the normalized cost, we define the *asymptotic cost of f* as the *limit superior* of the normalized cost when the length of words in the restriction language tends to infinity:

$$\mathsf{acost}(R, f) \;=^{\text{def}}\; \lim_{n\to\infty} \sup_{\substack{u\in R \\ |u|\geq n}} \mathsf{ncost}(u, f).$$

Accordingly, we define the *asymptotic repair cost* $\mathsf{acost}(R, T)$ for two languages R and T as the *minimum* of $\mathsf{acost}(R, f)$ taken over all repair strategies f for R and T. Note that $\mathsf{acost}(R, T)$ can be equally defined by

$$\mathsf{acost}(R, T) \;=\; \lim_{n\to\infty} \sup_{\substack{u\in R \\ |u|\geq n}} \min_{v\in T} \frac{\mathsf{dist}(u, v)}{|u|}.$$

Example 6. *Consider the languages $R = (a+b)^*$ and $T = (ab)^*$. Roughly, for any pair of consecutive occurrences of the letter a in the input, we will have to perform one edit in order to ensure alternation in the output. In particular, the number of edits required to get from a string in R to a string in T is unbounded. On the other hand, it is clear that, in the worst case (i.e., a^{2n}), one needs to edit approximately half of the letters in order to produce a string in T – this shows that the asymptotic repair cost is equal to $\frac{1}{2}$ (i.e. $\mathsf{acost}(R, T) = \frac{1}{2}$).*

Remark 1. *It is easy to see that the asymptotic repair cost $\mathsf{acost}(R, T)$ of any pair of languages ranges over the interval $[0, 1]$ of the real numbers. Indeed, for large words in the restriction language R, one can modify and delete the letters to create shorter words in the target language T, and thus the resulting editing cost is always less than the length of the input word.*

Ideally, we are interested in computing the asymptotic cost $\mathsf{acost}(R,T)$ for any pair of regular languages R and T, provided that this number is rational. We will indeed show that this is the case and describe a procedure that computes the asymptotic cost.

Streaming vs non-streaming. Similar to the bounded repair problem, we also study the asymptotic cost restricted to a streaming setting. We will mainly focus on the model of aggregate cost (see Section 2.2), as for the model of edit cost we do not have any interesting result. Formally, we define the *asymptotic* (*normalized aggregate*) *cost* of a k-lookahead streaming strategy $\mathcal{Z}$ for R and T, as

$$\mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(R,T) \;=^{\text{def}}\; \lim_{n\to\infty} \sup_{\substack{u\in R\\ |u|\geq n}} \frac{\mathsf{cost}^{\mathsf{aggr}}_{\mathcal{Z}}(u)}{|u|}$$

where $\mathsf{cost}^{\mathsf{aggr}}_{\mathcal{Z}}(u)$ is defined above. Similarly, we define the *asymptotic k-lookahead streaming cost of R and T*, denoted $\mathsf{acost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}(R,T)$, as the *infimum* of $\mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(R,T)$ taken over all k-lookahead streaming repair strategies $\mathcal{Z}$ for R and T.

We remark that, a priori, the infimum in the previous definition cannot be replaced by a minimum: it is conceivable that the asymptotic aggregate costs of the k-lookahead streaming repair strategies for R and T are arbitrary close to $\mathsf{acost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}(R,T)$, but never achieve this value.

Example 7. *Consider again the languages $R = (a+b)^*$ and $T = (ab)^*$ of Example 6 and recall that $\mathsf{acost}(R,T) = \frac{1}{2}$. We show that the asymptotic 0-lookahead streaming aggregate cost $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(R,T)$ is higher than the asymptotic cost in the nonstreaming case. Suppose that $\mathcal{Z}$ is a 0-lookahead streaming repair strategy for R and T. One can inductively construct arbitrarily long words $u_n = a_1\, a_2\, \ldots\, a_n \in R$ such that if*

$$z_0 \xrightarrow{a_1/v_1} z_1 \xrightarrow{a_2/v_2} \cdots \xrightarrow{a_n/v_n} z_n$$

is a partial run of $\mathcal{Z}$ on u_n, then each next letter a_n is equal to the last letter of the prefix $v_1\, v_2\, \ldots\, v_{n-1}$ of the output of $\mathcal{Z}$ (if $v_1 v_2 \ldots v_{n-1}$ is empty, then a_n can be chosen arbitrarily). It is easy to see that the aggregate cost induced by the run of $\mathcal{Z}$ on u_n is at least $n-1$, whence $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(R,T) = 1$. However, if we consider the model of edit cost (see Section 2.2), then we have $\mathsf{acost}^{\mathsf{edit}}_{0-\mathsf{lookahead}}(R,T) = \frac{1}{2}$ (in fact, any streaming strategy for R and T achieves this asymptotic edit cost).

Observe that, in general, the k-lookahead streaming asymptotic cost $\mathsf{acost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}(R,T)$ associated with two languages R and T is a non-increasing function of the lookahead parameter $k \in \mathbb{N}$ and it is bounded from below by the non-streaming asymptotic cost $\mathsf{acost}(R,T)$.

3.2 Asymptotic cost in the non-streaming case

In this section, we study the problem of computing the asymptotic cost in the non-streaming setting. We begin by recalling some background on distance automata, which will play a key role in the main characterization result.

3.2.1 Distance automata computing the edit cost

Distance automata were informally introduced in Section 2.5. There, it was shown the construction of a distance automaton $\mathcal{D}_{\mathcal{R},\mathcal{T}}$ that computes the cost of repairing any string in $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ for any two NFAs $\mathcal{R}$ and $\mathcal{T}$. Given that $\mathcal{D}_{\mathcal{R},\mathcal{T}}$ is one of the main building blocks of our algorithm, we give here the formal definition of distance automata and also the formal construction of $\mathcal{D}_{\mathcal{R},\mathcal{T}}$ already presented in Section 2.5.

A *distance automaton* is a tuple $\mathcal{D} = (\Sigma, Q, E, I, F)$, where Q is a finite set of states, $E \subseteq Q \times \Sigma \times \mathbb{N} \times Q$ is a finite transition relation, I and F are some initial and final conditions described by partial functions from Q to $\mathbb{N}$ and representing the costs of beginning and ending a run with certain states. A *run* of $\mathcal{D}$ on a word $u \in \Sigma^*$ is a sequence

$$\gamma \;=\; (q_0, a_1, c_1, q_1)\ (q_1, a_2, c_2, q_2)\ \ldots\ (q_{n-1}, a_n, c_n, q_n)$$

of pairwise adjacent transitions in E (i.e. $p'_i = p_{i+1}$ for each pair of consecutive transitions (p_i, a_i, c_i, p'_i) and $(p_{i+1}, a_{i+1}, c_{i+1}, p'_{i+1})$) that spell the input word $u = a_1 a_2 \ldots a_n$. The *cost* of the run γ is naturally defined by

$$\mathsf{cost}(\gamma) \;=^{\text{def}}\; \textstyle\sum_{1 \le i \le n} c_i.$$

We denote by $\mathcal{D}(u)$ the *minimum* value $I(q_0) + \mathsf{cost}(\gamma) + F(q_n)$ among all states q_0 in the domain $\mathsf{dom}(I)$ of I, all states q_n in the domain $\mathsf{dom}(F)$ of F, and all runs γ of $\mathcal{D}$ on u that start in q_0 and end in q_n. We let $\mathcal{D}(u) = \infty$ if there are no such states q_0 and q_n, or if there is no run from q_0 to q_n.

When considering the edit distance of a word $u \in \Sigma^*$ to a regular language $T \subseteq \Delta^*$, it is fairly natural to express this value in terms of the cost computed by a distance automaton. By default, we assume that the target language T is recognized by an NFA $\mathcal{T} = (\Delta, Q, E, I, F)$. Given two states p, q of $\mathcal{T}$, we let $\mathcal{T}_{p,q}$ be the NFA obtained from $\mathcal{T}$ by letting p be the new initial state and q the new unique final state. The distance automaton that computes the edit distance of a word $u \in \Sigma^*$ to the target language $\mathscr{L}(\mathcal{T})$ is defined as $\mathcal{D}^{\mathsf{edit}}_{\mathcal{T}} = (\Sigma, Q, E^{\mathsf{edit}}, I^{\mathsf{edit}}, F^{\mathsf{edit}})$, where

- E^{edit} is the set of all transitions of the form (p, a, c, q), with $p, q \in Q$, $a \in \Sigma$, q reachable from p in $\mathcal{T}$, and $c = \min\{\mathsf{dist}(a, v) \;:\; v \in \mathscr{L}(\mathcal{T}_{p,q})\}$,
- I^{edit} is the partial function that maps a state $q \in Q$ to the minimum among the values $\mathsf{dist}(\epsilon, v)$, with $v \in \bigcup_{p \in I} \mathscr{L}(\mathcal{T}_{p,q})$ (if q is not reachable from some initial state of $\mathcal{T}$, then $I^{\mathsf{edit}}(q)$ is undefined),

- F^{edit} is the partial function that maps a state $p \in Q$ to the minimum among the values $\mathsf{dist}(\epsilon, v)$, with $v \in \bigcup_{q \in F} \mathscr{L}(\mathcal{T}_{p,q})$ (if p cannot reach a final state of $\mathcal{T}$, then $F^{\mathsf{edit}}(p)$ is undefined).

One can easily show that $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$ computes exactly the edit distance of a word $u \in \Sigma^*$ to the regular language $\mathscr{L}(\mathcal{T})$:

Proposition 3.2.1. *For every word $u \in \Sigma^*$, we have*

$$\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}(u) \;=\; \min_{v \in \mathscr{L}(\mathcal{T})} \mathsf{dist}(u, v).$$

Proof. Let $\mathcal{T} = (\Delta, Q, E, I, F)$ be an NFA and let $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}} = (\Sigma, Q, E^{\mathsf{edit}}, I^{\mathsf{edit}}, F^{\mathsf{edit}})$ be the corresponding distance automaton, as defined above. For this proof, it is convenient to introduce the notion of locally optimal run. We say that a run $\gamma = (q_1, a_1, c_1, q_2) \;\ldots\; (q_n, a_n, c_n, q_{n+1})$ of $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$ is *locally optimal* if it has the minimum cost among all runs of $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$ on the same word $u = a_1 \ldots a_n$ that start in q_1 and end in q_{n+1}. Note that there can exist several locally optimal runs on the same word u with different costs (and, of course, with different beginning and ending states). We have the following characterization of the minimum cost:

Claim 1. *For every word $u \in \Sigma^*$ and every locally optimal run γ of $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$ on u that starts in p and ends in q, we have that*

$$\mathsf{cost}(\gamma) \;=\; \min_{v \in \mathscr{L}(\mathcal{T}_{p,q})} \mathsf{dist}(u, v).$$

Proof. The proof of the above claim is by induction on the length of the word u. If $u = \epsilon$, then the claim follows easily. As for the inductive step, let us assume that the claim holds for u and let us prove it for $u\,a$, with $a \in \Sigma$. Let γ be a locally optimal run of $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$ on $u\,a$ that starts in p and ends in q. Moreover, for every state $r \in Q$, let γ_r be a locally optimal run of $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$ on u from p to r. Since γ is locally optimal, we have that its cost is the minimum among the cost of a run γ_r, with $r \in Q$, plus the cost of an a-labeled transition from r to q. From the inductive hypothesis, we have that

$$\mathsf{cost}(\gamma_r) \;=\; \min_{v \in \mathscr{L}(\mathcal{T}_{p,r})} \mathsf{dist}(u, v).$$

This shows that

$$\begin{aligned} \mathsf{cost}(\gamma) \;&=\; \min_{(r,a,c,q) \in E^{\mathsf{edit}}} \mathsf{cost}(\gamma_r) + c \\ &=\; \min_{(r,a,c,q) \in E^{\mathsf{edit}}} \; \min_{v \in \mathscr{L}(\mathcal{T}_{p,r})} \mathsf{dist}(u, v) + c. \end{aligned}$$

We now look at the definition of the transition relation E^{edit}. It contains all quadruples (r, a, c, q) such that $c = \min\{\mathsf{dist}(a, w) \;:\; w \in \mathscr{L}(\mathcal{T}_{r,q})\}$. This shows that

$$\begin{aligned} \mathsf{cost}(\gamma) \;&=\; \min_{r \in Q} \; \min_{v \in \mathscr{L}(\mathcal{T}_{p,r})} \; \min_{w \in \mathscr{L}(\mathcal{T}_{r,q})} \mathsf{dist}(u, v) + \mathsf{dist}(a, w) \\ &=\; \min_{v\,w \in \mathscr{L}(\mathcal{T}_{p,q})} \mathsf{dist}(u\;a,\; v\;w). \end{aligned}$$

□

To complete the proof of the proposition, it is sufficient to recall that for every word $u \in \Sigma^*$, $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}(u)$ is the minimum among the values $\mathsf{cost}(\gamma) + I^{\mathsf{edit}}(p) + F^{\mathsf{edit}}(q)$, for all states $p \in \mathsf{dom}(I^{\mathsf{edit}})$ and $q \in \mathsf{dom}(F^{\mathsf{edit}})$ and all (locally optimal) runs γ of $\mathcal{D}^{\mathsf{edit}}$ on u that start from p and end in q. We also recall that $I^{\mathsf{edit}}(p) = \min\left\{\mathsf{dist}(\epsilon, v) \ : \ p' \in I, \ v \in \mathscr{L}(\mathcal{T}_{p',p})\right\}$ and $F^{\mathsf{edit}}(q) = \min\left\{\mathsf{dist}(\epsilon, v) \ : \ q' \in F, \ v \in \mathscr{L}(\mathcal{T}_{q,q'})\right\}$. This implies that $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}(u)$ is the minimum among the values:

$$\mathsf{dist}(\epsilon, v_I) + \mathsf{dist}(u, v) + \mathsf{dist}(\epsilon, v_F) \ = \ \mathsf{dist}(u, \ v_I \ v \ v_F),$$

where $v_I \in \bigcup_{p' \in I} \mathscr{L}(\mathcal{T}_{p',p})$, $v \in \mathscr{L}(\mathcal{T}_{p,q})$, $v_F \in \bigcup_{q' \in F} \mathscr{L}(\mathcal{T}_{q,q'})$, (hence $v_I \ v \ v_F \in \mathscr{L}(\mathcal{T})$), and $p, q \in Q$. This concludes the proof of the proposition. □

3.2.2 Shortcut property and determinizable components

Distance automata of the form $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$ are a proper sub-class of all distance automata. In particular, they satisfy the *shortcut property*, formalized just below. Given a symbol $a \in \Sigma$ and two states p, q of a distance automaton $\mathcal{D}$, we write $p \xrightarrow{a} q$ to denote the existence in $\mathcal{D}$ of a transition (p, a, c, q) with some cost $c \in \mathbb{N}$.

Definition 3.2.2. *A distance automaton $\mathcal{D}$ satisfies the* shortcut property *if for all symbols a, b and all states p, q, r, $p \xrightarrow{a} q \xrightarrow{b} r$ implies $p \xrightarrow{a} r$ and $p \xrightarrow{b} r$.*

Lemma 3.2.3. *For every DFA $\mathcal{T}$, $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$ satisfies the shortcut property.*

Proof. The proof follows almost immediately from the definition of $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$. Let us consider two consecutive transitions (p, a, c, q) and (q, b, c', r) in $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$. We know from the definition of the transition relation of $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$ that there exist some words $v \in \mathscr{L}(\mathcal{T}_{p,q})$ and $w \in \mathscr{L}(\mathcal{T}_{q,r})$. It follows that $v \ w \in \mathscr{L}(\mathcal{T}_{p,r})$. Again from the definition of $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$, we derive the existence of a transition (p, a, c'', r), for some $c'' \leq \mathsf{dist}(a, v \ w)$. Similarly, it follows that (p, b, c''', r) is a transition in $\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}}$, for some $c'' \leq \mathsf{dist}(b, v \ w)$. □

As with NFA, we call a *strongly connected component* (SCC) of a distance automaton $\mathcal{D}$ any maximal set of mutually reachable states. Given a SCC X of $\mathcal{D}$, we denote by $\mathcal{D}|X$ the sub-automaton obtained from $\mathcal{D}$ by restricting the set of states and transitions to X and by letting the initial and final conditions map any state of X to 0. Note that the transition graph of $\mathcal{D}|X$ is a clique when $\mathcal{D}$ satisfies the shortcut property.

A crucial property entailed by the shortcut property is the following one. Consider two runs γ and γ' of $\mathcal{D}|X$ that spell the same word u, but end in different states q and q'. If γ and γ' have optimal cost among all runs on $\mathcal{D}|X$ on u that end in q and q' respectively, then one can show that the difference in cost between γ and γ' is uniformly bounded by a constant. This implies that we can determinize $\mathcal{D}|X$ by using a subset construction, maintaining the difference between the optimal cost of reaching each state q and the overall optimal cost (the same idea underlies Mohri's determinization

procedure [Moh97]). Since this difference is always uniformly bounded by a constant, we obtain a finite state deterministic distance automaton:

Proposition 3.2.4. *For every distance automaton $\mathcal{D}$ that satisfies the shortcut property and every SCC X of $\mathcal{D}$, the sub-automaton $\mathcal{D}|X$ can be determinized.*

Proof. Let $\mathcal{D} = (\Sigma, Q, E, I, F)$ be a distance automaton satisfying the shortcut property and let X be a SCC of it. As a preliminary remark, we observe that, by definition, $\mathcal{D}|X = (\Sigma, X, E', I', F')$, where E' is obtained from E by restricting the set of states to X and $I'(q) = F'(q) = 0$ for all $q \in X$. Below, we consider runs of the distance automaton $\mathcal{D}|X$ on a given word u that end in a given state q and have the minimum cost among all runs of the same type. We call these runs (u, q)-*optimal* (note that these are similar to the locally optimal runs used in the proof of Proposition 3.2.1, with the only exception that the starting state is not fixed). We also say that a run is u-*optimal* if it is (u, q)-optimal for some state $q \in X$.

The basic idea underlying the determinization of the sub-automaton $\mathcal{D}|X$ stems from the following property:

Claim 1. *Given a word $u \in \Sigma^*$, the costs of any two u-optimal runs of $\mathcal{D}|X$ differ for at most c_{max}, where c_{max} is the maximum cost that appears in the transitions of $\mathcal{D}|X$.*

Proof. Let us fix a word $u = a_1 \dots a_n$ and let us consider two u-optimal runs of the form $\gamma = (q_1, a_1, c_1, q_2) \dots (q_n, a_n, c_n, q_{n+1})$ and $\gamma' = (q'_1, a_1, c'_1, q'_2) \dots (q'_n, a_n, c'_n, q'_{n+1})$ of $\mathcal{D}|X$ on it. Observe that the states q_{n+1} and q'_{n+1} belong to the same SCC X of $\mathcal{D}$ and, in particular, there exists $v \in \Sigma^*$ such that:

$$q'_{n+1} \xrightarrow{v} q_{n+1}$$

where $\xrightarrow{v}$ denotes the natural extension of the transition relation $\xrightarrow{a}$ from symbols to words (i.e., $p \xrightarrow{v} q$ iff $v = \epsilon$ and $p = q$, or $v = v' \cdot a$, $p \xrightarrow{v'} r$, $r \xrightarrow{a} q$, for some $v' \in \Sigma^*$ and some $r \in X$). Using a basic induction on $|v|$ and the shortcut property, one can prove that $\mathcal{D}$ contains a transition of the form $(q'_n, a_n, c'', q_{n+1})$, for some $c'' \in \{0, \dots, c_{\mathsf{max}}\}$. This implies that the following is also a run of $\mathcal{D}|X$ on u:

$$\gamma'' \ =^{\text{def}} \ (q'_1, a_1, c'_1, q'_2)\ (q'_2, a_1, c'_2, q'_3)\ \dots\ (q'_{n-1}, a_{n-1}, c'_1, q'_n)\ (q'_n, a_n, c'', q_{n+1})$$

Using the u-optimality of γ', we derive

$$\mathsf{cost}(\gamma') \ \leq \ \mathsf{cost}(\gamma'') \ \leq \ \mathsf{cost}(\gamma) + c_{\mathsf{max}}.$$

Using symmetric arguments, one derives the inequality $\mathsf{cost}(\gamma) \ \leq \ \mathsf{cost}(\gamma') + c_{\mathsf{max}}$. □

We now construct a deterministic distance automaton $\mathcal{D}'$ that turns out to be equivalent to $\mathcal{D}|X$. Intuitively, $\mathcal{D}'$ parses an input word u and it outputs the minimal cost of an u-optimal run, keeping track, at the same time, of the differences between this cost and the costs of the (u,q)-optimal runs, for any $q \in X$. These differences are called *residual costs* and, in view of the previous claim, are uniformly bounded. We formally define the deterministic distance automaton $\mathcal{D}'$ equivalent to $\mathcal{D}|X$ as the tuple $(\Sigma, Q', \delta, \bar{r}_0, F')$, where

- Q' is the set of vectors with entries indexed by states in X and values ranging over the finite set $\{0, \ldots, c_{\mathsf{max}}\}$, where c_{max} is the maximum cost that appears in the transitions of $\mathcal{D}|X$ (intuitively, these vectors represent the residual costs of (u,q)-locally optimal runs, for each state $q \in X$ and for some fixed word u);
- δ is the partial function from $Q' \times \Sigma$ to $\mathbb{N} \times Q'$ defined by $\delta(\bar{r}, a) = (c, \bar{r}')$, where $c = \min\left\{\bar{r}[p] + c' \;:\; p \in X,\ (p,a,c',q) \in E\right\}$ and $\bar{r}'[q] = \min\left\{\bar{r}[p] + c' - c \;:\; p \in X,\ (p,a,c',q) \in E\right\}$ (if there is no transition $(p,a,c',q) \in E$ on input symbol a, then we let $\delta(\bar{r},a)$ be undefined);
- $\bar{r}_0$ is the initial vector defined by $\bar{r}_0[q] = 0$ for all $q \in X$;
- F' is the constant function that maps any vector $\bar{r} \in Q'$ to 0 (note that there always exist $q \in X$ for which the corresponding residual $\bar{r}[q]$ in $\bar{r}$ is 0).

We argue that $\mathcal{D}'$ is equivalent to $\mathcal{D}|X$, namely, $\mathcal{D}'(u) = \mathcal{D}|X(u)$ for all $u \in \Sigma^*$. Let us consider a word $u = a_1 \ldots a_n$. By exploiting a simple induction on the length of u, one can prove that

- there is a run of $\mathcal{D}'$ on u if, and only if, there is a run of $\mathcal{D}|X$ on u,
- if $\rho = (\bar{r}_1, a_1, c_1, \bar{r}_2) \ldots (\bar{r}_n, a_n, c_n, \bar{r}_{n+1})$ is the (unique) run of $\mathcal{D}'$ on $u = a_1 \ldots a_n$ (recall that $\mathcal{D}'$ is deterministic), then, for every state $q \in X$, the cost of ρ augmented with the residual $\bar{r}_{n+1}[q]$ is precisely the cost of a (u,q)-optimal run of $\mathcal{D}|X$ on u.

We omit the formal proof of these properties. To conclude the proof, it is sufficient to observe that the second condition implies that the cost output by the deterministic distance automaton $\mathcal{D}'$ on u is precisely the cost output by $\mathcal{D}|X$ on u. □

The above result allows us to denote by $\det(\mathcal{D}|X)$ some deterministic distance automaton equivalent to $\mathcal{D}|X$, namely, $\det(\mathcal{D}|X)(u) = \mathcal{D}|X(u)$ for all $u \in \Sigma^*$. The automaton $\det(\mathcal{D}|X)$ can be computed from $\mathcal{D}|X$ by a direct exponential-time algorithm [Moh97].

Example 8. *Consider the distance automaton $\mathcal{D}$ of Figure 3.1, which computes the edit distance of any word to the target language $T = (ab + b)^*\, a^*$. As $\mathcal{D}$ satisfies the shortcut property and consists of two SCCs X_1 and X_2, the two sub-automata $\mathcal{D}|X_1$ and $\mathcal{D}|X_2$ can be turned into equivalent deterministic distance automata $\det(\mathcal{D}|X_1)$ and $\det(\mathcal{D}|X_2)$, depicted to the right of Figure 3.1.*

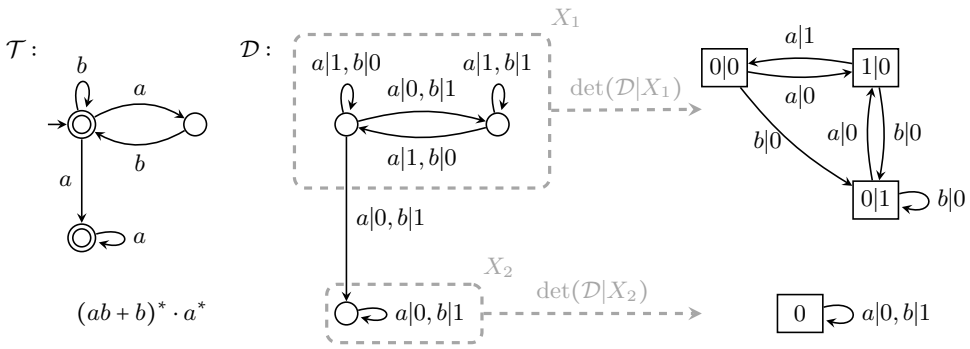


Figure 3.1: A distance automaton with two SCCs and its determinized sub-automata.

We remark that the above result does not imply that the entire distance automaton $\mathcal{D}$ is determinizable. Consider, for instance, a distance automaton $\mathcal{D}$ that computes the edit distance of a word u to the target language $T = a^* + b^*$. This distance is given by the symmetric difference between the number of occurrences of a and the number of occurrences of b and hence any deterministic device that computes $\mathsf{dist}(u, \mathscr{L}(\mathcal{T}))$ must use unbounded memory.

The above result also points out a subclass of distance automata (i.e. the component of a distance automaton satisfying the shortcut property) that is determinizable. This subclass of distance automata cannot be derived from the known determinizable subclasses for which Mohri's algorithm terminates. These determinizable subclasses usually have to satisfy the so-called "twins property" (see [Moh97] for a formal definition). Interestingly, one can find distance automata satisfying the shortcut property which components do not satisfy the "twins property". For example, one can easily show that the component X_1 of Figure 3.1 satisfy the shortcut property but does not satisfy the "twins property".

3.2.3 Asymptotic cost in the unrestricted case

Thanks to Proposition 3.2.1 and Lemma 3.2.3, we can reduce the problem of computing the asymptotic repair cost $\mathsf{acost}(\Sigma^*, \mathscr{L}(\mathcal{T}))$ in the unrestricted case to the problem of computing the *asymptotic cost* of a distance automaton $\mathcal{D}$ satisfying the shortcut property:

$$\mathsf{acost}(\mathcal{D}) \stackrel{\text{def}}{=} \lim_{n\to\infty} \sup_{\substack{u\in\Sigma^* \\ |u|\geq n}} \frac{\mathcal{D}(u)}{|u|}.$$

This section is devoted to providing an effective characterization of the asymptotic cost $\mathsf{acost}(\mathcal{D})$ that will imply that the value is rational and computable from $\mathcal{D}$.

Before turning to the characterization, we prove that, in general, it is not possible to compute the asymptotic cost $\mathsf{acost}(\mathcal{D})$ for an arbitrary distance automaton $\mathcal{D}$:

Proposition 3.2.5. *The problem of deciding, given an arbitrary distance automaton $\mathcal{D}$, whether or not $\mathsf{acost}(\mathcal{D}) \leq \frac{1}{2}$ is undecidable.*

Proof. We use the undecidability of the $\frac{1}{2}$-*threshold problem* for normalized costs induced by distance automata [Kro94], which consists of deciding, given a distance automaton $\mathcal{D}$, whether $\frac{\mathcal{D}(u)}{|u|} \leq \frac{1}{2}$ holds for all words $u \in \Sigma^*$.

Let us consider a distance automaton $\mathcal{D} = (\Sigma, Q, E, I, F)$. We compute a variant of the Kleene closure of $\mathcal{D}$, denoted $\mathcal{D}^*_\#$, by introducing a fresh symbol $\# \notin \Sigma$ and by adding 0-cost #-labeled transitions from all states $p \in \mathsf{dom}(F)$ to all states $q \in \mathsf{dom}(I)$. The new automaton $\mathcal{D}^*_\#$ satisfies the following property:

$$\forall\, m \in \mathbb{N}.\ \forall\, u_1, \ldots, u_m \in \Sigma^*. \qquad \mathcal{D}^*_\#(u_1 \# \ldots \# u_m) \;=\; \textstyle\sum_{1 \leq i \leq m} \mathcal{D}(u_i).$$

Clearly, this implies that $\mathsf{acost}(\mathcal{D}^*_\#) \;\geq\; \sup_{u \in \Sigma^*} \frac{\mathcal{D}(u)}{|u|}$. As for the converse inequality, we consider a family of words $u^{(n)} = u_1^{(n)} \# \ldots \# u_{m_n}^{(n)}$ of length precisely n such that $\lim_{n \to \infty} \dfrac{\mathcal{D}^*_\#(u^{(n)})}{n} \;=\; \mathsf{acost}(\mathcal{D}^*_\#)$ and we observe that

$$\frac{\mathcal{D}^*_\#(u^{(n)})}{n} \;=\; \frac{\sum_{1 \leq i \leq m_n} \mathcal{D}(u_i^{(n)})}{\sum_{1 \leq i \leq m_n} |u_i^{(n)}| + m_n - 1} \;\leq\; \sup_{u \in \Sigma^*} \frac{\mathcal{D}(u)}{|u|}.$$

In particular, the above inequalities imply that $\mathsf{acost}(\mathcal{D}^*_\#) \;=\; \sup_{u \in \Sigma^*} \dfrac{\mathcal{D}(u)}{|u|}$ and hence they reduce the $\frac{1}{2}$-threshold problem for $\mathcal{D}$ to the problem of deciding whether $\mathsf{acost}(\mathcal{D}^*_\#) \leq \frac{1}{2}$. From previous remarks about the undecidability of the $\frac{1}{2}$-threshold problem, it follows that it is not possible to compute the asymptotic cost for generic distance automata.

It is also worth remarking that the reduction we have given above does not preserve the shortcut property – indeed, the distance automaton $\mathcal{D}^*_\#$ has a single SCC, which, in general, cannot be determinized. This suggests the fact that, even though we can compute the asymptotic cost for the sub-class of distance automata satisfying the shortcut property, the decidability of the analogous $\frac{1}{2}$-threshold problem for the same sub-class cannot be immediately devised from that (we leave open this problem). □

Next we explain how the shortcut property helps in computing the asymptotic cost. One can show that the problem of computing $\mathsf{acost}(\mathcal{D})$ for a distance automaton $\mathcal{D}$ that is *deterministic* is reducible to the problem of computing normalized costs of simple cycles. Formally, a *simple cycle* is a run that is a cycle (i.e., that starts and ends in the same state) but that does not contain proper sub-cycles. It is then easy to show that for a deterministic distance automaton $\mathcal{D}$, $\mathsf{acost}(\mathcal{D})$ coincides with the maximum of $\frac{\mathsf{cost}(L)}{|L|}$ among all simple cycles L of $\mathcal{D}$, where $\mathsf{cost}(L)$ denotes the cost of the simple cycle L and $|L|$ its length (i.e., number of transitions in it). Thus by Proposition 3.2.4, calculation with simple cycles suffices to compute the asymptotic cost of any distance automaton satisfying the shortcut property and having a *single* SCC.

We now consider the more general case of a distance automaton $\mathcal{D}$ satisfying the shortcut property and having many SCCs, say $X_1, \ldots, X_k$. The situation in this case is slightly more complicated, as

$\mathsf{acost}(\mathcal{D})$ cannot be expressed as a function of $\mathsf{acost}(\mathcal{D}|X_1), \ldots, \mathsf{acost}(\mathcal{D}|X_k)$. For this we define $\bar{\mathcal{D}}$ as the deterministic *multi-distance* automaton obtained from the synchronous product of $\det(\mathcal{D}|X_1)$, ..., $\det(\mathcal{D}|X_k)$ and we denote by $L_1, \ldots, L_m$ the simple cycles of $\bar{\mathcal{D}}$. Moreover, given $1 \leq i \leq m$ and $1 \leq j \leq k$, we denote by $\mathsf{cost}_j(L_i)$ the cost of the projection of the simple cycle L_i into the j-th component of $\bar{\mathcal{D}}$. Assuming that $\mathcal{D}$ is *trimmed*, we can characterize the asymptotic cost of $\mathcal{D}$ as follows:

Theorem 3.2.6. *For every distance automaton $\mathcal{D}$ satisfying the shortcut property,*

$$\mathsf{acost}(\mathcal{D}) \;=\; \max_{\alpha_1,\ldots,\alpha_m \geq 0} \; \min_{1 \leq j \leq k} \; \frac{\sum_{1 \leq i \leq m} \alpha_i \cdot \mathsf{cost}_j(L_i)}{\sum_{1 \leq i \leq m} \alpha_i \cdot |L_i|} \tag{3.1}$$

where $X_1, \ldots, X_k$ are the SCCs of the distance automaton $\mathcal{D}$, $L_1, \ldots, L_m$ are the simple cycles of the multi-distance automaton $\bar{\mathcal{D}} = \det(\mathcal{D}|X_1) \times \ldots \times \det(\mathcal{D}|X_k)$, and $\mathsf{cost}_j(L_i)$ is the cost of the projection of the simple cycle L_i into the j-th component of $\bar{\mathcal{D}}$.

The idea underlying the above characterization is that the asymptotic cost $\mathsf{acost}(\mathcal{D})$ is achieved by repetitions of simple cycles in $\bar{\mathcal{D}}$. Indeed, the parameters $\alpha_1, \ldots, \alpha_m$ represent a correlation between the numbers of repetitions of the various simple cycles, and the index j represents the SCC of $\mathcal{D}$ that optimizes the normalized cost of these repetitions. Before turning to the proof of this characterization, we illustrate its use by means of an example.

Example 9. *Consider again the distance automaton $\mathcal{D}$ of Figure 3.1, with the two SCCs X_1 and X_2. The determinized sub-automaton $\det(\mathcal{D}|X_1)$ has four different simple cycles: one spelling aa with cost 1, one spelling ab with cost 0, one spelling b with cost 0, and one spelling aab with cost 1. Similarly, the determinized sub-automaton $\det(\mathcal{D}|X_2)$ has two simple cycles: one spelling a with cost 0, and the other spelling b with cost 1. Hence $(aa)^n$ is a family of words achieving a worst-case asymptotic cost of $\lim \frac{n}{2n} = \frac{1}{2}$ for the sub-automaton $\mathcal{D}|X_1$, and b^n is a family of words achieving a worst-case asymptotic cost of $\lim \frac{n}{n} = 1$ for the sub-automaton $\mathcal{D}|X_2$. However, a^{2n} is not a worst-case for $\mathcal{D}|X_2$ (as it can be repaired with asymptotic cost 0) and, symmetrically, b^n is not a worst-case for $\mathcal{D}|X_1$. This means that the asymptotic cost for $\mathcal{D}$ is achieved by a suitable combination of both families, namely, $(aab)^n$ (note that the parameters $\alpha_{aa} = 1$ and $\alpha_b = 1$ maximize Equation 3.1). This gives the asymptotic cost $\mathsf{acost}(\mathcal{D}) = \lim \frac{n}{3n} = \frac{1}{3}$.*

The proof of Theorem 3.2.6 consists of establishing two inequalities, which are given by Lemma 3.2.8 and Lemma 3.2.9 below.

For the first inequality, we argue that all words can be approximated in cost by repetitions of simple cycles, and that the cost of parsing these words is at most the cost of a "homogeneous run", i.e., a run lying entirely inside a single component of $\mathcal{D}$. The first part of the proof relies on the following property, which is also present in [EM79]. We state it in a graph-theoretic setting, in such a way that it can be later reused in several proofs – for the moment, the reader can understand the

graph $\mathcal{G}$ as the multi-distance automaton $\bar{\mathcal{D}}$ and the path ρ in $\mathcal{G}$ as a run of $\bar{\mathcal{D}}$. Intuitively, this property eases the calculation of the cost within a SCC X_j of a run ρ of $\bar{\mathcal{D}}$ that does not show any particular 'cyclic' structure.

Lemma 3.2.7 (Simple cycle decomposition [EM79])**.** *Let $\mathcal{G}$ be a finite graph and let $L_1, \ldots, L_m$ be all the simple cycles in it. Given a path ρ (i.e., a sequence of connected edges) in $\mathcal{G}$, one can find a partition the domain of ρ into (possibly non-convex) subsets $U_0, U_1, \ldots, U_m$ such that*

1. *$|U_0| \leq K$, where K is the number of vertices of $\mathcal{G}$,*
2. *for all $1 \leq i \leq m$, the sub-sequence $\rho|U_i$ is a repetition of L_i.*

Proof. We find the sets $U_0, U_1, \ldots, U_m$ by exploiting an induction. At the beginning we define $U_{0,0}$ to be the entire domain of the path ρ and $U_{0,i} = \varnothing$ for all $1 \leq i \leq m$. At each induction step on $n \in \mathbb{N}$, we subtract a suitable convex subset V_n from $U_{n,0}$ and we add it to one of the subsets $U_{n,i}$, with $1 \leq i \leq m$. More precisely, if $|U_{n,0}| \leq K$, where K is the number of vertices of $\mathcal{G}$, then we terminate the induction with the current sets $U_{n,0}, U_{n,1}, \ldots, U_{n,m}$. Otherwise, we continue the induction by specifying the sets $U_{n+1,0}$, $U_{n+1,1}$, …, $U_{n+1,m}$ in terms of $U_{n,0}$, $U_{n,1}$, …, $U_{n,m}$ as follows. We first claim that there is an interval V_n contained in $U_{n,0}$ for which the sub-sequence $\rho|V_n$ is an occurrence of a simple cycle L_i, for some $1 \leq i \leq m$. Indeed, since the length of the sub-sequence $\rho|U_{n,0}$ exceeds the number K of states of $\mathcal{G}$, we know that $\rho|U_{n,0}$ contains two repeated occurrences of the same state, and hence a cycle L. In its turn, the cycle L must contain an occurrence of a simple cycle among $L_1, \ldots, L_m$ (this follows from the fact that the containment relation between cycles is a well-founded partial order). We choose such an occurrence of the simple cycle L_i in $\rho|U_{n,0}$ and we denote by V_n the set of the positions in $U_{n,0}$ that carry the chosen occurrence. Accordingly, we define

- $U_{n+1,0} = U_{n,0} \setminus V_n$,
- $U_{n+1,i} = U_{n,i} \cup V_n$,
- $U_{n+1,i'} = U_{n,i'}$ for all indices $1 \leq i' \leq m$ different from i.

If n is the last step of the induction, then we define $U_i = U_{n,i}$ for all $0 \leq i \leq m$. Note that we have $|U_0| = |U_{n,0}| \leq K$. Moreover, it is easy to verify (e.g., by induction on n) that each sub-sequence $\rho|U_{n,i}$ (and hence, in particular, the sub-sequence $\rho|U_i$) is a repetition of the corresponding simple cycle L_i. This concludes the proof of the claim. □

Hereafter, for the sake of brevity, we tacitly assume that $X_1, \ldots, X_k$ are the SCCs of the distance automaton $\mathcal{D}$ and $L_1, \ldots, L_m$ are the simple cycles of the multi-distance automaton $\bar{\mathcal{D}} = \det(\mathcal{D}|X_1) \times \ldots \times \det(\mathcal{D}|X_k)$. Moreover, we say that a run $\gamma = (q_0, a_1, c_1, q_1) \ldots (q_{n-1}, a_n, c_n, q_n)$ of $\mathcal{D} = (\Sigma, Q, E, I, F)$ is *accepting* if it starts in a state $q_0 \in \mathsf{dom}(I)$ and it ends in a state $q_n \in \mathsf{dom}(F)$.

Lemma 3.2.8. *For every distance automaton $\mathcal{D}$ satisfying the shortcut property,*

$$\mathsf{acost}(\mathcal{D}) \;\leq\; \max_{\alpha_1,\ldots,\alpha_m \geq 0} \; \min_{1\leq j\leq k} \; \frac{\sum_{1\leq i\leq m} \alpha_i \cdot \mathsf{cost}_j(L_i)}{\sum_{1\leq i\leq m} \alpha_i \cdot |L_i|}.$$

Proof. Let $\left(u^{(n)}\right)_{n\in\mathbb{N}}$ be a family of words over the alphabet Σ such that

$$\mathsf{acost}(\mathcal{D}) \;=\; \limsup_{n\to\infty} \; \frac{\mathcal{D}(u^{(n)})}{|u^{(n)}|}.$$

Without loss of generality, we can assume that the limit of the sequence $\frac{\mathcal{D}(u^{(n)})}{|u^{(n)}|}$, for arbitrarily large numbers n, exists, and hence it coincides with $\mathsf{acost}(\mathcal{D})$. Indeed, if this were not the case, we could restrict ourselves to a proper sub-family $\left(u^{(n)}\right)_{n\in N}$ of words, where N is a an infinite subset of the natural numbers, in such a way that the sequence $\frac{\mathcal{D}(u^{(n)})}{|u^{(n)}|}$ converges for n ranging over N. Assuming that $\lim\limits_{n\to\infty} \frac{\mathcal{D}(u^{(n)})}{|u^{(n)}|}$ is defined will allow us to further restrict, if necessary, to sub-families of words without compromising the above equality. To prove the lemma, it is sufficient to find some parameters $\alpha_1, ..., \alpha_m \geq 0$ that satisfy the following inequality:

$$\limsup_{n\to\infty} \; \frac{\mathcal{D}(u^{(n)})}{|u^{(n)}|} \;\leq\; \min_{1\leq j\leq k} \; \frac{\sum_{1\leq i\leq m} \alpha_i \cdot \mathsf{cost}_j(L_i)}{\sum_{1\leq i\leq m} \alpha_i \cdot |L_i|}. \tag{3.2}$$

Let us fix $n \in \mathbb{N}$ and denote by $\rho^{(n)}$ the (unique) accepting run of $\bar{\mathcal{D}}$ on the word $u^{(n)}$, and by $\rho_j^{(n)}$ the projection of it into the j-th component X_j, for any $1 \leq j \leq k$.

First, we compare the cost $\mathcal{D}(u^{(n)})$ with the cost $\mathcal{D}|X_j(u^{(n)})$ for each SCC X_j. Consider an accepting run $\gamma^{(n)}$ of $\mathcal{D}$ on $u^{(n)}$ which starts in some state p and ends in some state q and that minimizes the value $\mathsf{cost}(\gamma^{(n)}) + I(p) + F(q)$. Clearly, we have $\mathcal{D}(u^{(n)}) \leq \mathsf{cost}(\gamma^{(n)}) + I_{\mathsf{max}} + F_{\mathsf{max}}$, where I_{max} is the maximum value taken by the initial condition of $\mathcal{D}$ and F_{max} is the maximum value taken by the final condition of $\mathcal{D}$. Now, consider a run $\gamma_j^{(n)}$ of $\mathcal{D}|X_j$ on the same word $u^{(n)}$, but entirely inside the SCC X_j, which starts in p' and ends in q' and that minimizes the relative cost. Since the initial and final conditions of $\mathcal{D}|X_j$ map every state to 0, we have $\mathcal{D}|X_j(u^{(n)}) = \mathsf{cost}(\gamma_j^{(n)})$. Moreover, since $\mathcal{D}$ is trimmed, we know that p' is reachable from p and q is reachable from q'. Thus, using the shortcut property, one can easily verify that $\mathsf{cost}(\gamma^{(n)}) \leq \mathsf{cost}(\gamma_j^{(n)}) + 2c_{\mathsf{max}}$, where c_{max} is the maximum cost that appears in the transitions of $\mathcal{D}$ (the additive constant $2c_{\mathsf{max}}$ accounts for the cost discount in considering a run of $\mathcal{D}|X_j$ rather than a run of $\mathcal{D}$). This shows that

$$\begin{aligned}
\mathcal{D}(u^{(n)}) \;&\leq\; \mathsf{cost}(\gamma^{(n)}) \;+\; I_{\mathsf{max}} + F_{\mathsf{max}} \\
&\leq\; \min_{1\leq j\leq k} \mathsf{cost}(\gamma_j^{(n)}) \;+\; 2c_{\mathsf{max}} + I_{\mathsf{max}} + F_{\mathsf{max}} \\
&=\; \min_{1\leq j\leq k} \mathcal{D}|X_j(u^{(n)}) \;+\; 2c_{\mathsf{max}} + I_{\mathsf{max}} + F_{\mathsf{max}}
\end{aligned}$$

From Proposition 3.2.4, we also know that $\mathcal{D}|X_j(u^{(n)}) = \det(\mathcal{D}|X_j)(u^{(n)})$. Moreover, since $\det(\mathcal{D}|X_j)$ is a deterministic distance automaton, the projection $\rho_j^{(n)}$ of $\rho^{(n)}$ can be viewed as the unique run of $\det(\mathcal{D}|X_j)$ on $u^{(n)}$. We thus obtain

$$\mathcal{D}|X_j(u^{(n)}) \;=\; \det(\mathcal{D}|X_j)(u^{(n)}) \;=\; \mathsf{cost}(\rho_j^{(n)}).$$

Below, we explicitly compute the cost of each run $\rho_j^{(n)}$ using the costs of the simple cycles L_i in the component X_j of $\mathcal{D}$. The problem is that the run $\rho^{(n)}$ may not contain factors consisting of entire repetitions of these simple cycles. We overcome this problem by viewing the multi-distance automaton $\bar{\mathcal{D}}$ as a finite graph and the run $\rho^{(n)}$ of $\bar{\mathcal{D}}$ as a path in it. The simple cycle decomposition Lemma 3.2.7 then implies the existence of a partition of the domain of $\rho^{(n)}$ into (possibly non-convex) subsets $U_0^{(n)}, U_1^{(n)}, \ldots, U_m^{(n)}$ such that

1. $|U_0^{(n)}|$ is uniformly bounded by the number K of states of $\bar{\mathcal{D}}$,

2. for all $1 \leq i \leq m$, the sub-sequence $\rho^{(n)}|U_i^{(n)}$ is a repetition of the simple cycle L_i of $\bar{\mathcal{D}}$.

For every index $1 \leq i \leq m$, we denote by $\mathsf{occ}_i^{(n)}$ the number of repetitions of the simple cycle L_i in the sub-sequence $\rho^{(n)}|U_i^{(n)}$ (by a slight abuse of terminology we say that these are also 'repetitions' in the run $\rho^{(n)}$). We are now ready to bound the cost of $\rho^{(n)}$ in the component X_j in terms of the costs of the 'repetitions' of each simple cycle L_i in $\rho^{(n)}$.

The first, straightforward, inequality is as follows (recall that the sets $U_0^{(n)}$, $U_1^{(n)}$, ..., $U_m^{(n)}$ form a partition of the domain of $\rho^{(n)}$ and $|U_0^{(n)}| \leq K$):

$$\begin{aligned} \mathsf{cost}(\rho_j^{(n)}) &= \textstyle\sum_{1\leq i\leq m} \mathsf{cost}(\rho_j^{(n)}|U_i^{(n)}) + \mathsf{cost}(\rho_j^{(n)}|U_0^{(n)}) \\ &\leq \textstyle\sum_{1\leq i\leq m} \mathsf{cost}(\rho_j^{(n)}|U_i^{(n)}) + K \cdot c'_{\mathsf{max}} \end{aligned}$$

where c'_{max} is the maximum cost that appears in the transitions of $\bar{\mathcal{D}}$. Moreover, it easily follows from the fact that the sub-sequence $\rho^{(n)}|U_i^{(n)}$ is an $\mathsf{occ}_i^{(n)}$-fold repetition of the simple cycle L_i, that

$$\mathsf{cost}(\rho_j^{(n)}|U_i^{(n)}) = \mathsf{occ}_i^{(n)} \cdot \mathsf{cost}_j(L_i).$$

Now, in order to find the parameters $\alpha_1, ..., \alpha_m \geq 0$ that satisfy Equation 3.2, we consider the asymptotic behaviour of the sequence $\frac{\mathsf{occ}_i^{(n)}}{n}$. Without loss of generality, we can assume that the limit of $\frac{\mathsf{occ}_i^{(n)}}{n}$ for arbitrarily large numbers $n \in \mathbb{N}$ exists. Indeed, we can always find an infinite set N of natural numbers such that the sequence $\frac{\mathsf{occ}_i^{(n)}}{n}$ converges for n ranging over N. Note that restricting to the corresponding sub-family of words $u^{(n)}$ and runs $\gamma^{(n)}$ and $\rho^{(n)}$, for $n \in N$, does not affect the previously established equalities (in particular, $\mathsf{acost}(\mathcal{D}) = \lim_{n\to\infty} \frac{\mathsf{cost}(\gamma^{(n)})}{|u^{(n)}|}$). For the sake of simplicity, we shall not explicitly mention the set N hereafter and we use, for instance, $\lim_{n\to\infty} f(n)$ to denote the limit of a certain function f for arbitrarily large numbers $n \in N$. Accordingly, for every index $1 \leq i \leq m$, we define

$$\alpha_i =^{\text{def}} \lim_{n\to\infty} \frac{\mathsf{occ}_i^{(n)}}{n}.$$

Clearly, we have that

$$\mathsf{occ}_i^{(n)} \cdot \mathsf{cost}_j(L_i) = \left(n \cdot \alpha_i + \mathsf{occ}_i^{(n)} - n \cdot \alpha_i\right) \cdot \mathsf{cost}_j(L_i) = n \cdot \alpha_i \cdot \mathsf{cost}_j(L_i) + g_{i,j}(n)$$

where $g_{i,j}(n) =^{\text{def}} (\mathsf{occ}_i^{(n)} - n \cdot \alpha_i) \cdot \mathsf{cost}_j(L_i)$ is a function whose limit tends to 0 (hence $g_{i,j} \in O(1)$, using the 'big-O' notation).

Putting everything together, we get an upper bound on the cost of the run $\gamma^{(n)}$:

$$\begin{aligned}
\mathcal{D}(u^{(n)}) &\leq \min_{1\leq j\leq k} \mathcal{D}|X_j(u^{(n)}) + 2c_{\mathsf{max}} + I_{\mathsf{max}} + F_{\mathsf{max}} \\
&= \min_{1\leq j\leq k} \mathsf{cost}(\rho_j^{(n)}) + 2c_{\mathsf{max}} + I_{\mathsf{max}} + F_{\mathsf{max}} \\
&\leq \min_{1\leq j\leq k} \left(\sum_{1\leq i\leq m} \mathsf{cost}(\rho_j^{(n)}|U_i^{(n)}) + K \cdot c'_{\mathsf{max}} \right) + 2c_{\mathsf{max}} + I_{\mathsf{max}} + F_{\mathsf{max}} \\
&= \min_{1\leq j\leq k} \sum_{1\leq i\leq m} \left(\mathsf{occ}_i^{(n)} \cdot \mathsf{cost}_j(L_i) \right) + 2c_{\mathsf{max}} + I_{\mathsf{max}} + F_{\mathsf{max}} + K \cdot c'_{\mathsf{max}} \\
&= \min_{1\leq j\leq k} \sum_{1\leq i\leq m} \left(n \cdot \alpha_i \cdot \mathsf{cost}_j(L_i) + g_{i,j}(n) \right) + 2c_{\mathsf{max}} + I_{\mathsf{max}} + F_{\mathsf{max}} + K \cdot c'_{\mathsf{max}} \\
&= n \cdot \min_{1\leq j\leq k} \textstyle\sum_{1\leq i\leq m} \alpha_i \cdot \mathsf{cost}_j(L_i) + O(1).
\end{aligned}$$

Similarly, we obtain a lower bound on the length of the word $u^{(n)}$:

$$\begin{aligned}
|u^{(n)}| &\geq \textstyle\sum_{1\leq i\leq m} u^{(n)}|U_i^{(n)} \\
&= \textstyle\sum_{1\leq i\leq m} \mathsf{occ}_i^{(n)} \cdot |L_i| \\
&= n \cdot \textstyle\sum_{1\leq i\leq m} \alpha_i \cdot |L_i| - O(1).
\end{aligned}$$

Towards a conclusion, we prove Equation 3.2 as follows:

$$\begin{aligned}
\limsup_{n\to\infty} \frac{\mathcal{D}(u^{(n)})}{|u^{(n)}|} &\leq \limsup_{n\to\infty} \frac{n \cdot \min_{1\leq j\leq k} \sum_{1\leq i\leq m} \alpha_i \cdot \mathsf{cost}_j(L_i) + O(1)}{n \cdot \sum_{1\leq i\leq m} \alpha_i \cdot |L_i| - O(1)} \\
&= \min_{1\leq j\leq k} \frac{\sum_{1\leq i\leq m} \alpha_i \cdot \mathsf{cost}_j(L_i)}{\sum_{1\leq i\leq m} \alpha_i \cdot |L_i|}.
\end{aligned}$$

□

For the converse inequality, we present a large family of words for which the optimal runs are nearly homogeneous (in the sense that they lie almost entirely inside a single component of $\mathcal{D}$). The words will consist of nested repetitions of simple cycles in such a way that any optimal run stabilizes in the same component.

Lemma 3.2.9. *For every distance automaton $\mathcal{D}$ satisfying the shortcut property,*

$$\mathsf{acost}(\mathcal{D}) \geq \max_{\alpha_1,\ldots,\alpha_m\geq 0} \min_{1\leq j\leq k} \frac{\sum_{1\leq i\leq m} \alpha_i \cdot \mathsf{cost}_j(L_i)}{\sum_{1\leq i\leq m} \alpha_i \cdot |L_i|}.$$

Proof. Let us fix arbitrarily some parameters $\alpha_1, \ldots, \alpha_m \geq 0$. We prove the above inequality by constructing a family of 'cyclic' words $u^{(n)}$ that depend on $\alpha_1, \ldots, \alpha_m$ and n and such that the normalized cost of any run of $\mathcal{D}$ on $u^{(n)}$ dominates, in the limit, the cost $\frac{\sum_{1\leq i\leq m} \alpha_i \cdot \mathsf{cost}_j(L_i)}{\sum_{1\leq i\leq m} \alpha_i \cdot |L_i|}$.

We fix (i) a run σ_0 of $\bar{\mathcal{D}}$ that starts from the initial state of $\bar{\mathcal{D}}$ and ends in the first/last state of L_1, (ii) a run σ_m of $\bar{\mathcal{D}}$ that starts from the first/last state of L_m and ends in the first/last state of L_1, and (iii) for all $1 \leq i < m$, a run σ_i of $\bar{\mathcal{D}}$ that starts from the first/last state of L_i and ends in the first/last state of L_{i+1}. Without loss of generality, we can assume that the lengths of the runs $\sigma_0, \sigma_1, \ldots, \sigma_m$ do not exceed the number K of states of $\bar{\mathcal{D}}$. Now, we think of each simple cycle L_i as a run of the multi-distance automaton $\bar{\mathcal{D}}$ that starts and ends in the same state and we construct, for every natural number n, a 'cyclic' run $\rho^{(n)}$ of $\bar{\mathcal{D}}$ as follows:

$$\rho^{(n)} \quad =^{\text{def}} \quad \sigma_0 \ (\rho^{(n)}_{\mathsf{cycles}})^n$$

$$\text{where} \qquad \rho^{(n)}_{\mathsf{cycles}} \quad =^{\text{def}} \quad L_1^{\lceil n \cdot \alpha_i \rceil} \ \sigma_1 \ L_2^{\lceil n \cdot \alpha_i \rceil} \ \ldots \ \sigma_{m-1} \ L_m^{\lceil n \cdot \alpha_i \rceil} \ \sigma_m.$$

Accordingly, we denote by $u^{(n)}$ the word spelled out by the run $\rho^{(n)}$.

To prove the claim of the lemma, it is sufficient to prove that the following inequality holds for all $n \in \mathbb{N}$ and for all choices of runs $\gamma^{(n)}$ of $\mathcal{D}$ on $u^{(n)}$:

$$\limsup_{n \to \infty} \ \frac{\mathsf{cost}(\gamma^{(n)})}{|u^{(n)}|} \ \geq \ \min_{1 \leq j \leq k} \ \frac{\sum_{1 \leq i \leq m} \alpha_i \cdot \mathsf{cost}_j(L_i)}{\sum_{1 \leq i \leq m} \alpha_i \cdot |L_i|}. \tag{3.3}$$

Let us now fix further a run $\gamma^{(n)}$ of $\mathcal{D}$ on $u^{(n)}$ for each $n \in \mathbb{N}$. We introduce some additional notation. Given $n \in \mathbb{N}$, $1 \leq i \leq m$, and $1 \leq j \leq k$, we define:

- $U_j^{(n)}$ to be the set of positions of $\gamma^{(n)}$ that carry occurrences of transitions whose states belong to the same SCC X_j (note that $U_j^{(n)}$ is an interval);
- $V_j^{(n)}$ to be the maximal subset of $U_j^{(n)}$ such that the sub-run $\rho^{(n)}|V_j^{(n)}$ is a repetition of the block $\rho^{(n)}_{\mathsf{cycles}}$ (note that $V_j^{(n)}$ is also an interval);
- $W_{j,i}^{(n)}$ to be the maximal subset of $V_j^{(n)}$ such that $\rho^{(n)}|W_{j,i}^{(n)}$ is a repetition of the simple cycle L_i (note that the sets $W_{j,1}^{(n)}$, ..., $W_{j,m}^{(n)}$ form a partition of $V_j^{(n)}$ and they contain possibly non-contiguous positions).
- $\mathsf{occ}_j^{(n)}$ to be the number of repetitions of $\rho^{(n)}_{\mathsf{cycles}}$ in the sub-run $\rho^{(n)}|V_j^{(n)}$, namely, $\mathsf{occ}_j^{(n)} = \frac{|V_j^{(n)}|}{|\rho^{(n)}_{\mathsf{cycles}}|}$ (note that this implies $\rho^{(n)}|W_{j,i}^{(n)} = L_i^{\lceil n \cdot \alpha_i \rceil \cdot \mathsf{occ}_j^{(n)}}$).

The first inequality is straightforward (the sets $W_{j,i}^{(n)}$ are pairwise disjoint):

$$\mathsf{cost}(\gamma^{(n)}) \ \geq \ \sum_{1 \leq j \leq k} \ \sum_{1 \leq i \leq m} \mathsf{cost}(\gamma^{(n)}|W_{j,i}^{(n)}).$$

Given an index $1 \leq j \leq k$, we also denote by $\rho_j^{(n)}$ the projection of $\rho^{(n)}$ into the j-th component (we can think of it as a run of the deterministic distance automaton $\det(\mathcal{D}|X_j)$ on $u^{(n)}$). Below, we fix some indices $1 \leq i \leq m$ and $1 \leq j \leq k$ and we compare the cost of each sub-sequence $\gamma^{(n)}|W_{j,i}^{(n)}$ with the cost of the corresponding sub-run $\rho_j^{(n)}|W_{j,i}^{(n)}$ of $\mathcal{D}|X_j$.

We observe that $\gamma^{(n)}|W_{j,i}^{(n)}$ is not necessarily a run of $\mathcal{D}|X_j$, since the set $W_{j,i}^{(n)}$ is not an interval of the domain of $\gamma^{(n)}$ (we can still compute its cost though). We first turn $\gamma^{(n)}|W_{j,i}^{(n)}$ into a run $\widetilde{\gamma}_{j,i}^{(n)}$ of $\mathcal{D}|X_j$ on $u^{(n)}|W_{j,i}^{(n)}$ of similar cost, as follows. Suppose that there exist two positions $x < y$ in $W_{j,i}^{(n)}$ such that $z \notin W_{j,i}^{(n)}$ for all $x < z < y$ and the corresponding transitions $\gamma^{(n)}[x] = (p_x, a_x, c_x, q_x)$ and $\gamma^{(n)}[y] = (p_y, a_y, c_y, q_y)$, which are consecutive in $\gamma^{(n)}|W_{j,i}^{(n)}$, do not match (i.e., $q_x \neq p_y$). We call such a pair (x, y) of positions a *gap* of $W_{j,i}^{(n)}$. The states q_x and p_y belong to the same SCC X_j of $\mathcal{D}$ and hence it follows from the shortcut property that $\mathcal{D}|X_j$ contains a transition of the form (p_x, a_x, c'_x, p_y), for some $c'_x \in \mathbb{N}$, connecting p_x to p_y. The described operation of connecting states through shortcuts can be applied to every gap (x, y) of $W_{j,i}^{(n)}$, thus resulting in a correct run $\widetilde{\gamma}_{j,i}^{(n)}$ of $\mathcal{D}|X_j$ on the sub-word $u^{(n)}|W_{j,i}^{(n)}$. We observe two crucial properties about this construction. First, the number of required operations is at most $\mathsf{occ}_j^{(n)}$ (this follows from the fact that the gaps of $W_{j,i}^{(n)}$ can only appear between the positions that correspond to two non-consecutive occurrences of $L_i^{\lceil n \cdot \alpha_i \rceil}$ in $\rho^{(n)}$ and from the fact that there exist at most $\mathsf{occ}_j^{(n)}$ such occurrences). Second, the difference in cost that results from one application of this operation never exceeds the maximum cost c_{max} of the transitions in $\mathcal{D}$. In view of these properties, we have

$$\mathsf{cost}(\gamma^{(n)}|W_{j,i}^{(n)}) \;\geq\; \mathsf{cost}(\widetilde{\gamma}_{j,i}^{(n)}) \;-\; c_{\mathsf{max}} \cdot \mathsf{occ}_j^{(n)}.$$

From the fact that $\widetilde{\gamma}_{j,i}^{(n)}$ is a correct run of $\mathcal{D}|X_j$ on $u^{(n)}|W_{j,i}^{(n)}$ and the initial and final conditions of the sub-automaton $\mathcal{D}|X_j$ map every state in X_j to the cost 0, we derive

$$\mathsf{cost}(\widetilde{\gamma}_{j,i}^{(n)}) \;\geq\; \mathcal{D}|X_j(u^{(n)}|W_{j,i}^{(n)}).$$

Proposition 3.2.4 then implies that

$$\mathcal{D}|X_j(u^{(n)}|W_{j,i}^{(n)}) \;=\; \det(\mathcal{D}|X_j)(u^{(n)}|W_{j,i}^{(n)}).$$

Now, consider the j-th projection $\rho_j^{(n)}|W_{j,i}^{(n)}$ of the sub-run $\rho^{(n)}|W_{j,i}^{(n)}$ of the deterministic multi-distance $\bar{\mathcal{D}}$. By construction, $\rho_j^{(n)}|W_{j,i}^{(n)}$ is a run of $\det(\mathcal{D}|X_j)$ on the sub-word $u^{(n)}|W_{j,i}^{(n)}$. Note that $\rho_j^{(n)}|W_{j,i}^{(n)}$ can start from a state that is different from the initial state of $\det(\mathcal{D}|X_j)$ and hence it is not guaranteed to have optimal cost. However, the first state of $\rho_j^{(n)}|W_{j,i}^{(n)}$ is reachable from the initial state of $\det(\mathcal{D}|X_j)$ by a path $\tau_{j,i}^{(n)}$ of length at most K, where K is the number of states of $\bar{\mathcal{D}}$. For the sake of brevity, we denote by $x_{j,i}^{(n)}$ be the word spelled out by $\tau_{j,i}^{(n)}$ and by $z_{j,i}^{(n)}$ the sub-word $u^{(n)}|W_{j,i}^{(n)}$ (which is spelled out by $\rho^{(n)}|W_{j,i}^{(n)}$). Clearly, we have $\mathsf{cost}(\rho_j^{(n)}|W_{j,i}^{(n)}) \leq \mathsf{cost}(\tau_{j,i}^{(n)}) + \mathsf{cost}(\rho_j^{(n)}|W_{j,i}^{(n)}) = \det(\mathcal{D}|X_j)(x_{j,i}^{(n)}\; z_{j,i}^{(n)}) = \mathcal{D}|X_j(x_{j,i}^{(n)}\; z_{j,i}^{(n)})$. Let us now consider two optimal runs $\alpha_{j,i}^{(n)}$ and $\beta_{j,i}^{(n)}$ of $\mathcal{D}|X_j$ on $x_{j,i}^{(n)}$ and $z_{j,i}^{(n)}$, respectively. Clearly, we have $\mathcal{D}|X_j(x_{j,i}^{(n)}) = \mathsf{cost}(\alpha_{j,i}^{(n)})$ and $\mathcal{D}|X_j(z_{j,i}^{(n)}) = \mathsf{cost}(\beta_{j,i}^{(n)})$. Moreover, since $\mathcal{D}|X_j$ satisfies the shortcut property, we have that the juxtaposition of the two runs $\alpha_{j,i}^{(n)}$ and $\beta_{j,i}^{(n)}$ of $\mathcal{D}|X_j$ can be turned into a valid run $\lambda_{j,i}^{(n)}$ of $\mathcal{D}|X_j$ on $x_{j,i}^{(n)}\; z_{j,i}^{(n)}$, having cost at most $\mathsf{cost}(\alpha_{j,i}^{(n)}) + \mathsf{cost}(\beta_{j,i}^{(n)}) + c_{\mathsf{max}}$, where c_{max} is the maximum cost of

the transitions in $\mathcal{D}$. This shows that $\mathcal{D}|X_j(x_{j,i}^{(n)}\ z_{j,i}^{(n)}) \leq \mathsf{cost}(\lambda_{j,i}^{(n)}) \leq \mathsf{cost}(\alpha_{j,i}^{(n)}) + \mathsf{cost}(\beta_{j,i}^{(n)}) + c_{\mathsf{max}} = \mathcal{D}|X_j(x_{j,i}^{(n)}) + \mathcal{D}|X_j(z_{j,i}^{(n)}) + c_{\mathsf{max}} \leq K \cdot c_{\mathsf{max}} + \det(\mathcal{D}|X_j)(z_{j,i}^{(n)}) + c_{\mathsf{max}}$. Overall, this shows that

$$\det(\mathcal{D}|X_j)(u^{(n)}|W_{j,i}^{(n)}) \ \geq\ \mathsf{cost}(\rho_j^{(n)}|W_{j,i}^{(n)}) \ -\ (K+1)\cdot c_{\mathsf{max}}.$$

Now, we explicitly compute the cost of $\rho_j^{(n)}|W_{j,i}^{(n)}$ as follows. Since $\rho_j^{(n)}|W_{j,i}^{(n)}$ is an $(\lceil n\cdot\alpha_i\rceil\cdot\mathsf{occ}_j^{(n)})$-fold repetition of the simple cycle L_i, we have

$$\mathsf{cost}(\rho_j^{(n)}|W_{j,i}^{(n)}) \ =\ \lceil n\cdot\alpha_i\rceil\cdot\mathsf{occ}_j^{(n)}\cdot\mathsf{cost}_j(L_i).$$

Another crucial step amounts at showing that the sum of the numbers $\mathsf{occ}_j^{(n)}$ over all indices $1 \leq j \leq k$ is almost equal (i.e., equal up to a constant) to the total number n of repetitions of the block $\rho_{\mathsf{cycles}}^{(n)}$ in $\rho^{(n)}$. The first inequality follows trivially by construction:

$$\sum_{1\leq j\leq k}\mathsf{occ}_j^{(n)} \ \leq\ n.$$

As for the converse equality, we recall that each set $V_j^{(n)}$ is defined as the maximal set of positions of $\gamma^{(n)}$ that contain states from the SCC X_j and such that the sub-run $\rho^{(n)}|V_j^{(n)}$ is a repetition of the block $\rho_{\mathsf{cycles}}^{(n)}$. This implies that there exist at most k occurrences of the block $\rho_{\mathsf{cycles}}^{(n)}$ in $\rho^{(n)}$ that are not entirely covered by some set $V_j^{(n)}$, for any $1 \leq j \leq k$. From this and from the definition of $\mathsf{occ}_j^{(n)}$, we derive the following inequalities:

$$\sum_{1\leq j\leq k}\mathsf{occ}_j^{(n)} = \sum_{1\leq j\leq k}\frac{|V_j^{(n)}|}{|\rho_{\mathsf{cycles}}^{(n)}|} \geq \frac{|\rho^{(n)}| - k\cdot|\rho_{\mathsf{cycles}}^{(n)}|}{|\rho_{\mathsf{cycles}}^{(n)}|} \geq \frac{n\cdot|\rho_{\mathsf{cycles}}^{(n)}| - k\cdot|\rho_{\mathsf{cycles}}^{(n)}|}{|\rho_{\mathsf{cycles}}^{(n)}|} \geq n-k.$$

Putting together all the inequalities (and using some basic rewriting), we obtain a lower bound on the cost of the run $\gamma^{(n)}$:

$$\begin{aligned}
\mathsf{cost}(\gamma^{(n)}) \ &\geq\ \sum_{1\leq j\leq k}\sum_{1\leq i\leq m}\mathsf{cost}(\gamma^{(n)}|W_{j,i}) \\
&\geq\ \sum_{1\leq j\leq k}\sum_{1\leq i\leq m}\mathsf{cost}(\widetilde{\gamma}_{j,i}^{(n)}) \ -\ m\cdot c_{\mathsf{max}}\cdot\sum_{1\leq j\leq k}\mathsf{occ}_j^{(n)} \\
&\geq\ \sum_{1\leq j\leq k}\sum_{1\leq i\leq m}\det(\mathcal{D}|X_j)(u^{(n)}|W_{j,i}^{(n)}) \ -\ m\cdot c_{\mathsf{max}}\cdot\sum_{1\leq j\leq k}\mathsf{occ}_j^{(n)} \\
&\geq\ \sum_{1\leq j\leq k}\sum_{1\leq i\leq m}\Big(\mathsf{cost}(\rho_j^{(n)}|W_{j,i}^{(n)}) - (K+1)\cdot c_{\mathsf{max}}\Big) \ -\ m\cdot c_{\mathsf{max}}\cdot n \\
&\geq\ \sum_{1\leq j\leq k}\sum_{1\leq i\leq m}\Big(\lceil n\cdot\alpha_i\rceil\cdot\mathsf{occ}_j^{(n)}\cdot\mathsf{cost}_j(L_i)\Big) \ -\ m\cdot k\cdot(K+1)\cdot c_{\mathsf{max}} - m\cdot c_{\mathsf{max}}\cdot n \\
&\geq\ \sum_{1\leq j\leq k}\Big(\mathsf{occ}_j^{(n)}\cdot n\cdot\sum_{1\leq i\leq m}\alpha_i\cdot\mathsf{cost}_j(L_i)\Big) \ -\ O(n) \\
&\geq\ n\cdot\Big(\sum_{1\leq j\leq k}\mathsf{occ}_j^{(n)}\Big)\cdot\min_{1\leq j\leq k}\sum_{1\leq i\leq m}\Big(\alpha_i\cdot\mathsf{cost}_j(L_i)\Big) \ -\ O(n)
\end{aligned}$$

$$\geq\ n \cdot \Big(n-k\Big) \cdot \min_{1\leq j\leq k} \sum_{1\leq i\leq m} \Big(\alpha_i \cdot \mathsf{cost}_j(L_i)\Big)\ -\ O(n)$$

$$=\ n^2 \cdot \min_{1\leq j\leq k} \sum_{1\leq i\leq m} \alpha_i \cdot \mathsf{cost}_j(L_i)\ -\ O(n).$$

Similarly, we easily obtain an upper bound on the length of the word $u^{(n)}$:

$$\begin{aligned} |u^{(n)}| &= n \cdot \sum_{1\leq i\leq m} \Big(\lceil n\cdot\alpha_i\rceil \cdot |L_i|\Big) + |\sigma_0| + n \cdot \sum_{1\leq i\leq m} |\sigma_i| \\ &\leq n^2 \cdot \sum_{1\leq i\leq m} \Big(\alpha_i \cdot |L_i|\Big) + n \cdot \sum_{1\leq i\leq m} |L_i| + K + n\cdot m\cdot K \\ &= n^2 \cdot \sum_{1\leq i\leq m} \alpha_i \cdot |L_i| + O(n). \end{aligned}$$

Towards a conclusion, we prove Equation 3.3 as follows:

$$\begin{aligned} \limsup_{n\to\infty} \frac{\mathsf{cost}(\gamma^{(n)})}{|u^{(n)}|} &\geq \limsup_{n\to\infty} \frac{n^2 \cdot \min_{1\leq j\leq k} \sum_{1\leq i\leq m} \alpha_i \cdot \mathsf{cost}_j(L_i)\ -\ O(n)}{n^2 \cdot \sum_{1\leq i\leq m} \alpha_i \cdot |L_i|\ +\ O(n)} \\ &= \min_{1\leq j\leq k} \frac{\sum_{1\leq i\leq m} \alpha_i \cdot \mathsf{cost}_j(L_i)}{\sum_{1\leq i\leq m} \alpha_i \cdot |L_i|}. \end{aligned}$$

□

In virtue of Lemma 3.2.8 and Lemma 3.2.9, the proof of Equation 3.1 from Theorem 3.2.6 becomes trivial.

We make a few remarks related to the effectiveness of the characterization. First of all, we observe that the right handside term of Equation 3.1 can be rewritten as the following instance of a linear programming problem:

$$\begin{array}{llll} \texttt{maximize} & y & \texttt{subject to} & \sum_{1\leq i\leq m} c_{i,j}\cdot x_i \geq y \qquad \forall 1\leq j\leq k \\ & & & \sum_{1\leq i\leq m} x_i \leq 1,\quad x_i \geq 0 \qquad \forall 1\leq i\leq m. \end{array}$$

where, for every $1\leq i\leq m$ and every $1\leq j\leq k$, $c_{i,j} = \frac{\mathsf{cost}_j(L_i)}{|L_i|}$. Intuitively, the variables $x_1,\ldots,x_m$ represent the values $\alpha_1\cdot|L_1|$, …, $\alpha_m\cdot|L_m|$ normalized in such a way that they sum up to 1, and the variable y represents an under-approximation of the value of the right handside term of the equation. It is also known [MG06] that the optimal choices for the parameters $x_1,\ldots,x_m,y$ can be found at the 'corners' of the $(m+1)$-dimensional polyhedron that results from the intersection of the finitely many half-spaces defined by the above linear inequalities. This explains why we put $\max_{\alpha_1,\ldots,\alpha_m\geq 0}$ instead of $\sup_{\alpha_1,\ldots,\alpha_m\geq 0}$ in Equation 3.1. Moreover, it also implies that the asymptotic cost $\mathsf{acost}(\mathcal{D})$ is a rational number.

Regarding the complexity of the problem of computing $\mathsf{acost}(\mathcal{D})$, we observe that (i) the size $|\bar{\mathcal{D}}|$ of the multi-distance automaton $\bar{\mathcal{D}}$ is exponential in $|\mathcal{D}|$, (ii) each simple cycle L_i has length at most linear in $|\bar{\mathcal{D}}|$, (iii) the number m of all simple cycles of $\bar{\mathcal{D}}$ is exponential in $|\bar{\mathcal{D}}|$, and (iv) each constant $c_{i,j} = \frac{\mathsf{cost}_j(L_i)}{|L_i|}$ can be computed in time polynomial in $|\bar{\mathcal{D}}|$ and $|L_i|$. Overall, the problem

of computing the asymptotic cost of $\mathcal{D}$ is reduced, in time doubly exponential, to an instance of a linear programming problem. The latter problem is known to be in PTIME [Kar84], which proves that $\mathsf{acost}(\mathcal{D})$ can be computed in doubly exponential time.

The algorithm above can even be improved to single-exponential time by rewriting Equation 3.1 in terms of transitions and then reducing it to a linear programming problem. Denote by $t_1, \ldots, t_l$ the transitions of $\bar{\mathcal{D}}$ and by $\mathsf{cost}_j(t_i)$ the cost of the projection of the transition t_i into the j-th component of $\bar{\mathcal{D}}$. Furthermore, for each state q of $\bar{\mathcal{D}}$ let $\mathrm{in}(q)$ and $\mathrm{out}(q)$ be subsets of $\{1, \ldots, l\}$ such that $i \in \mathrm{in}(q)$ ($i \in \mathrm{out}(q)$) iff q is the initial (final) state of t_i. To rewrite Equation 3.1 in terms of transitions instead of simple cycles, one can assign a number $\beta_i \geq 0$ to each transition t_i that counts the number of times each transition is taken or, in other words, how much "flow" is passing through it. Clearly, not every combination of $\beta_1, \ldots, \beta_l$ represents the number of times α_i that each simple cycle L_i is taken. Nevertheless, we can force $\beta_1, \ldots, \beta_l$ to represent a valid combination of simple cycles by adding linear constraints expressing the *Kirchhoff's Law* [Die12]. This law basically says that "the total amount of flow into a state has to be equal the total amount of flow out of it" [Die12]. Thus, we can rewrite Equation 3.1 as follows:

$$\begin{aligned} \mathsf{acost}(\mathcal{D}) \;=\; & \max_{\beta_1,\ldots,\beta_l \geq 0} \; \min_{1 \leq j \leq k} \; \frac{\sum_{1 \leq i \leq l} \beta_i \cdot \mathsf{cost}_j(t_i)}{\sum_{1 \leq i \leq l} \beta_i} \\ & \texttt{subject to} \quad \sum_{i \in \mathrm{in}(q)} \beta_i \;=\; \sum_{i \in \mathrm{out}(q)} \beta_i \qquad \forall q \in \bar{Q} \end{aligned} \tag{3.4}$$

where $\bar{Q}$ denotes the states of $\bar{\mathcal{D}}$. Notice that the restrictions imposed over the ingoing and outgoing "flow" of each state allow $\beta_1, \ldots, \beta_l$ to represent a valid combination of simple cycles in $\bar{\mathcal{D}}$ by Kirchhoff's law. The advantage of representing Equation 3.1 in terms of transition instead of simple cycles stems from the fact that the number of variables $\beta_1, \ldots, \beta_l$ is polynomial in the size of $\bar{\mathcal{D}}$ in contrast to $\alpha_1, \ldots, \alpha_m$ which is exponential.

Similar than for Equation 3.1, we can now represent Equation 3.4 with the following linear programming problem:

$$\begin{array}{llll} \texttt{maximize} \quad z \quad \texttt{subject to} & \sum_{1 \leq i \leq l} d_{i,j} \cdot y_i \geq z & & \forall 1 \leq j \leq k \\ & \sum_{1 \leq i \leq l} y_i \leq 1, \quad y_i \geq 0 & & \forall 1 \leq i \leq l \\ & \sum_{i \in \mathrm{in}(q)} y_i = \sum_{i \in \mathrm{out}(q)} y_i & & \forall q \in \bar{Q}. \end{array}$$

where, for every $1 \leq i \leq l$ and every $1 \leq j \leq k$, $d_{i,j} = \mathsf{cost}_j(t_i)$. The linear programming problem above is of single exponential size with respect to the size of $\mathcal{D}$. Therefore, the problem of computing the asymptotic cost of $\mathcal{D}$ can be reduced in exponential time to an instance of a linear programming problem. Given that the latter problem is known to be in PTIME [Kar84], this proves that $\mathsf{acost}(\mathcal{D})$ can be computed in single exponential time.

If we consider the threshold problem for the asymptotic cost, that is, the problem of deciding whether $\mathsf{acost}(\mathcal{D}) \leq \nu$ for a given a distance automaton $\mathcal{D}$ satisfying the shortcut property and a

given rational number ν, then the upper-bound that we derive with the above analysis is EXPTIME, namely, by computing the asymptotic cost of $\mathcal{D}$ and checking whether is less than ν. As a consequence, we have that the complexity of the threshold problem for the asymptotic repair cost for a universal restriction language and a target language represented by a NFA is between PSPACE and EXPTIME. Note that the PSPACE lower-bound follows directly from the lower-bound for the bounded repair problem in the unrestricted case where $\mathsf{acost}(\Sigma^*, \mathscr{L}(\mathcal{T})) \leq 0$ iff $\mathsf{cost}(\Sigma^*, \mathscr{L}(\mathcal{T})) < \infty$ for any NFA $\mathcal{T}$. The latter problem was shown to be PSPACE-hard in Corollary 2.4.10.

Proposition 3.2.10. *The problem of deciding, given an alphabet Σ, an NFA $\mathcal{T}$, and a rational number ν, whether $\mathsf{acost}(\Sigma^*, \mathscr{L}(\mathcal{T})) \leq \nu$ is in EXPTIME and it is PSPACE-hard.*

3.2.4 Asymptotic cost in the general case

Now, we show how to generalize the characterization of the asymptotic cost in the unrestricted case to our original repair problem, which involves the presence of both a restriction and a target language. We first modify the definition of asymptotic cost for a distance automaton to include the presence of a restriction language $\mathscr{L}(\mathcal{R})$ recognized by an NFA $\mathcal{R}$:

$$\mathsf{acost}(\mathcal{R}, \mathcal{D}) \ =^{\text{def}} \ \lim_{n \to \infty} \ \sup_{\substack{u \in \mathscr{L}(\mathcal{R}) \\ |u| \geq n}} \ \frac{\mathcal{D}(u)}{|u|}.$$

Thanks to Proposition 3.2.1, we have that the asymptotic cost $\mathsf{acost}(R, T)$ for two regular languages R and T recognized by NFA $\mathcal{R}$ and $\mathcal{T}$ is equal to $\mathsf{acost}(\mathcal{R}, \mathcal{D}^{\mathsf{edit}}_{\mathcal{T}})$.

As usual, given a distance automaton $\mathcal{D}$ satisfying the shortcut property, we denote by $\bar{\mathcal{D}}$ the multi-distance automaton $\det(\mathcal{D}|X_1) \times \ldots \times \det(\mathcal{D}|X_k)$, where $X_1, \ldots, X_k$ are all the SCCs of $\mathcal{D}$. Moreover, given an NFA $\mathcal{R}$ and a SCC Y of it, we consider the synchronized product $\bar{\mathcal{D}} \times (\mathcal{R}|Y)$ of the multi-distance automaton $\bar{\mathcal{D}}$ and the sub-automaton $\mathcal{R}|Y$, which is obtained from $\mathcal{R}$ by restricting the set of states to Y (it does not matter which state is chosen to be initial/final in $\mathcal{R}|Y$). We then denote by $L^Y_1, \ldots, L^Y_{m^Y}$ all the simple cycles of $\bar{\mathcal{D}} \times (\mathcal{R}|Y)$. Finally, given a simple cycle L^Y_i of $\bar{\mathcal{D}} \times (\mathcal{R}|Y)$ and a SCC X of $\mathcal{D}$, we denote by $\mathsf{cost}_X(L^Y_i)$ the cost of the projection of L^Y_i into the component X of $\bar{\mathcal{D}} \times (\mathcal{R}|Y)$. The generalized characterization result is as follows:

Theorem 3.2.11. *For every NFA $\mathcal{R}$ and every distance automaton $\mathcal{D}$ satisfying the shortcut property,*

$$\mathsf{acost}(\mathcal{R}, \mathcal{D}) \ = \ \max_{\substack{\tau = Y_1 \ldots Y_h \ \in \ \mathrm{dag}(\mathcal{R}) \\ \alpha^{Y_1}_1, \ \ldots, \ \alpha^{Y_1}_{m^{Y_1}} \geq 0 \\ \ldots\ldots \\ \alpha^{Y_h}_1, \ \ldots, \ \alpha^{Y_h}_{m^{Y_h}} \geq 0}} \ \min_{\pi = X_1 \ldots X_h \ \in \ \mathrm{dag}(\mathcal{D})} \ \frac{\sum\limits_{\substack{1 \leq l \leq h \\ 1 \leq i \leq m^{Y_l}}} \alpha^{Y_l}_i \cdot \mathsf{cost}_{X_l}(L^{Y_l}_i)}{\sum\limits_{\substack{1 \leq l \leq h \\ 1 \leq i \leq m^{Y_l}}} \alpha^{Y_l}_i \cdot |L^{Y_l}_i|} \tag{3.5}$$

Proof sketch. The proof is very similar to the proof of Theorem 3.2.6. In particular, we prove two inequalities between the asymptotic cost $\mathsf{acost}(\mathcal{R},\mathcal{D})$ and the right handside expression.

Let us consider first the inequality

$$\mathsf{acost}(\mathcal{R},\mathcal{D}) \;\leq\; \max_{\substack{\tau=Y_1\dots Y_h \,\in\, \mathrm{dag}(\mathcal{R}) \\ \alpha_1^{Y_1},\,\dots,\,\alpha_{m^{Y_1}}^{Y_1}\geq 0 \\ \cdots\cdots \\ \alpha_1^{Y_h},\,\dots,\,\alpha_{m^{Y_h}}^{Y_h}\geq 0}} \;\min_{\pi=X_1\dots X_h \,\in\, \mathrm{dag}(\mathcal{D})} \frac{\displaystyle\sum_{\substack{1\leq l\leq h \\ 1\leq i\leq m^{Y_l}}} \alpha_i^{Y_l}\cdot \mathsf{cost}_{X_l}(L_i^{Y_l})}{\displaystyle\sum_{\substack{1\leq l\leq h \\ 1\leq i\leq m^{Y_l}}} \alpha_i^{Y_l}\cdot |L_i^{Y_l}|}$$

In order to prove this inequality, one needs to fix, as in Lemma 3.2.8, a family of words $(u^{(n)})_{n\in\mathbb{N}}$ from the restriction language $\mathscr{L}(\mathcal{R})$ such that

$$\mathsf{acost}(\mathcal{D}) \;=\; \limsup_{n\to\infty} \frac{\mathcal{D}(u^{(n)})}{|u^{(n)}|}.$$

By possibly restricting to sub-families of words, one can replace lim sup by lim in the above equation.

One new ingredient is the following. Without loss of generality, we can also assume that all words $u^{(n)}$ induce accepting runs on the NFA $\mathcal{R}$ following the same path of SCCs of $\mathcal{R}$. More precisely, we denote by $\sigma^{(n)}$ some accepting run of $\mathcal{R}$ on $u^{(n)}$, and by $\tau^{(n)}$ the path in $\mathrm{dag}(\mathcal{R})$ that consists of the sequence of SCCs visited by $\sigma^{(n)}$. Then, since there are only a finite number of paths in $\mathrm{dag}(\mathcal{R})$, we can restrict ourselves to suitable sub-families of words and runs in such a way that all paths $\tau^{(n)}$ are the same. We denote them simply by $\tau = Y_1\dots Y_h$.

The proof then continues as follows. We partition the domain of each run $\sigma^{(n)}$ of the NFA $\mathcal{R}$ into some intervals $V_1^{(n)},\dots,V_h^{(n)}$ (recall that h is the number of SCCs in the path τ) in such a way that each sub-sequence $\sigma^{(n)}|V_l^{(n)}$, for $1\leq l\leq h$, is a run of the sub-automaton $\mathcal{R}|Y_l$ on the sub-word $u^{(n)}|V_l^{(n)}$ (in fact the sets $V_1^{(n)},\dots,V_h^{(n)}$ do not form a partition of the entire domain of $\sigma^{(n)}$, since there can be transitions crossing different SCCs in $\mathcal{R}$; however, the number of these transitions is at most the number of SCCs in $\mathcal{R}$, and thus their cost is negligible for n that tends to ∞). One then considers the (unique) accepting run $\rho^{(n)}$ of $\bar{\mathcal{D}}$ on the word $u^{(n)}$. Given an index $1\leq l\leq h$ and a SCC X of $\mathcal{D}$, we denote by $\rho_{l,X}^{(n)}$ the projection of the sub-run $\rho^{(n)}|V_l^{(n)}$ into the component X. Every sequence $\rho_{l,X}^{(n)}$ can be viewed as a run of $\det(\mathcal{D}|X)$ on the sub-word $u^{(n)}|V_l^{(n)}$. This run has cost almost equal (up to additive constants) to the cost of some optimal run $\gamma_{l,X}^{(n)}$ of $\mathcal{D}|X$ on $u^{(n)}|V_l^{(n)}$. Moreover, given a path $\pi = X_1\dots X_h$ in $\mathrm{dag}(\mathcal{D})$, one can construct a run $\gamma_\pi^{(n)}$ of $\mathcal{D}$ on $u^{(n)}$ by 'concatenating' the runs $\gamma_{1,X_1}^{(n)},\dots,\gamma_{h,X_h}^{(n)}$ (this requires the use of the shortcut property to correct the possible pairs of consecutive transitions that have unmatched states). This shows that

$$\begin{aligned}
\mathcal{D}(u^{(n)}) \;&\leq\; \min_{\pi=X_1\dots X_h \,\in\, \mathrm{dag}(\mathcal{D})} \mathsf{cost}(\gamma_\pi^{(n)}) \;+\; O(1) \\
&=\; \min_{\pi=X_1\dots X_h \,\in\, \mathrm{dag}(\mathcal{D})} \textstyle\sum_{1\leq l\leq h} \mathsf{cost}(\gamma_{l,X_l}^{(n)}) \;+\; O(1) \\
&=\; \min_{\pi=X_1\dots X_h \,\in\, \mathrm{dag}(\mathcal{D})} \textstyle\sum_{1\leq l\leq h} \mathsf{cost}(\rho_{l,X_l}^{(n)}) \;+\; O(1).
\end{aligned}$$

Given the above inequality, the rest of the proof is similar to that of Lemma 3.2.8. That is, we decompose each run $\rho_{l,X_l}^{(n)}$ into simple cycles and we approximate its cost up to additive constants.

We now turn to the converse inequality:

$$\mathsf{acost}(\mathcal{R},\mathcal{D}) \;\geq\; \max_{\substack{\tau=Y_1\ldots Y_h \,\in\, \mathrm{dag}(\mathcal{R}) \\ \alpha_1^{Y_1},\,\ldots,\,\alpha_{m^{Y_1}}^{Y_1}\geq 0 \\ \cdots\cdots \\ \alpha_1^{Y_h},\,\ldots,\,\alpha_{m^{Y_h}}^{Y_h}\geq 0}} \;\min_{\pi=X_1\ldots X_h \,\in\, \mathrm{dag}(\mathcal{D})} \frac{\sum\limits_{\substack{1\leq l\leq h \\ 1\leq i\leq m^{Y_l}}} \alpha_i^{Y_l}\cdot \mathsf{cost}_{X_l}(L_i^{Y_l})}{\sum\limits_{\substack{1\leq l\leq h \\ 1\leq i\leq m^{Y_l}}} \alpha_i^{Y_l}\cdot |L_i^{Y_l}|}$$

As in the proof of Lemma 3.2.9, the first step is to fix a path $\tau = Y_1 \ldots Y_h$ in $\mathrm{dag}(\mathcal{R})$ and some parameters $\alpha_i^{Y_l} \geq 0$ for each $1 \leq l \leq h$ and each $1 \leq i \leq m^{Y_l}$, where m^{Y_l} denotes the number of simple cycles of the automaton $\bar{\mathcal{D}} \times (\mathcal{R}|Y_l)$.

One proves the inequality by defining the following family of runs of $\bar{\mathcal{D}} \times \mathcal{R}$:

$$\rho^{(n)} \;=^{\mathrm{def}}\; \sigma_0 \left(\rho_{1,\mathsf{cycles}}^{(n)}\right)^n \;\ldots\; \sigma_{h-1} \left(\rho_{h,\mathsf{cycles}}^{(n)}\right)^n \sigma_h$$

where, for all $1 \leq l \leq h$,

$$\rho_{l,\mathsf{cycles}}^{(n)} \;=^{\mathrm{def}}\; (L_1^{Y_l})^{\lceil n\cdot\alpha_1^{Y_l}\rceil}\, \sigma_{l,1}\, (L_2^{Y_l})^{\lceil n\cdot\alpha_2^{Y_l}\rceil} \;\ldots\; \sigma_{l,m^{Y_l}-1}\, (L_{m^{Y_k}}^{Y_l})^{\lceil n\cdot\alpha_{m^{Y_l}}^{Y_l}\rceil}\, \sigma_{l,m^{Y_l}}$$

and $\sigma_0, \sigma_1, \ldots, \sigma_h, \sigma_{l,1}, \ldots, \sigma_{l,m^{Y_l}}$ are suitable runs of $\bar{\mathcal{D}}$ of bounded length that connect the various simple cycles $L_i^{Y_l}$. Accordingly, one defines $u^{(n)}$ to be the word spelled out by the run $\rho^{(n)}$. Observe that, by construction (and under the assumption that the automaton $\mathcal{R}$ is trimmed), this word belongs to the language recognized by the NFA $\mathcal{R}$.

It is not difficult then to generalize the arguments used in Lemma 3.2.9 (we thus omit the details). □

Using arguments similar to the complexity analysis of the unrestricted case, we obtain a EXPTIME algorithm that decides whether the asymptotic repair cost $\mathsf{acost}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))$ (= $\mathsf{acost}(\mathcal{R},\mathcal{D}_{\mathcal{T}}^{\mathsf{edit}})$) associated with two NFA $\mathcal{R}$ and $\mathcal{T}$ is less than or equal to a certain threshold $\nu \in \mathbb{Q}$:

Corollary 3.2.12. *The problem of deciding, given two NFA $\mathcal{R}$ and $\mathcal{T}$ and a rational number ν, whether $\mathsf{acost}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})) \leq \nu$ is in EXPTIME.*

3.3 Asymptotic cost in the streaming case

Here we characterize the asymptotic repair (aggregate) cost in the streaming setting in terms of the value of a mean-payoff game [EM79]. Throughout this section we assume some familiarity with turn-based games over finite structures (see [GTW03] for a survey).

3.3.1 Mean-payoff games

A *mean-payoff game* is an infinite, turn-based game played over an arena $\mathcal{G} = (V, E, v_0)$, where V is the union of two disjoint finite sets of vertices, V_{Adam} (owned by player Adam) and V_{Eve} (owned by player Eve), $E \subseteq V \times \mathbb{N} \times V$ is a finite set of weighted edges, and $v_0 \in V$ is an initial vertex. The game starts at v_0 and, at each round, the player who owns the current vertex v moves along an edge $(v, c, v') \in E$. The reward for Adam (resp., the penalty for Eve) in an infinite play $\pi = (v_0, c_1, v_1)\ (v_1, c_2, v_2)\ \ldots$ is given by the value $\nu^{\pi}_{\mathsf{Adam}}$ (resp., ν^{π}_{Eve}), where

$$\nu^{\pi}_{\mathsf{Adam}} =^{\text{def}} \liminf_{n\to\infty} \frac{\sum_{i=1}^{n} c_i}{n} \qquad \nu^{\pi}_{\mathsf{Eve}} =^{\text{def}} \limsup_{n\to\infty} \frac{\sum_{i=1}^{n} c_i}{n}.$$

Intuitively, Adam wants to maximize his reward $\nu^{\pi}_{\mathsf{Adam}}$ while Eve wants to minimize her penalty ν^{π}_{Eve}.

It is known from [EM79] that, in any mean-payoff game, the best reward that can be enforced by Adam coincides with the least penalty that can be enforced by Eve, and, furthermore, these values can be achieved by positional strategies:

Theorem 3.3.1 (Ehrenfeucht and Mycielski [EM79])**.** *We can associate with each mean-payoff game $\mathcal{G}$ a value $\nu_{\mathcal{G}}$ such that Adam (resp., Eve) has a positional strategy that guarantees $\nu^{\pi}_{\mathsf{Adam}} \geq \nu_{\mathcal{G}}$ (resp., $\nu^{\pi}_{\mathsf{Eve}} \leq \nu_{\mathcal{G}}$) for all plays π that respect his (resp., her) strategy.*

In view of the above theorem, we can denote by $\nu_{\mathcal{G}}$ the value of a mean-payoff game over the arena $\mathcal{G}$ and we can restrict ourselves to positional strategies for both Adam and Eve. We will represet a positional strategy for Adam (resp., Eve) as a function from Adam's vertices (resp., Eve's vertices) to outgoing edges.

3.3.2 Characterization of asymptotic streaming cost

Let R and T be the languages recognized by two DFA $\mathcal{R} = (\Sigma, Q, \delta, q_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', r_0, F')$, respectively. To compute the asymptotic cost $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(R, T)$ for streaming (0-lookahead) repair strategies we construct the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, where Adam's vertices are pairs of the form (q, r), with $q \in Q$ and $r \in Q'$, and Eve's vertices are pairs of the form (q, r, a), with $q \in Q$, $r \in Q'$, and $a \in \Sigma$. The edges of the arena are triples of the form $\big((q, r), 0, (q', r, a)\big)$, where $q' = \delta(q, a)$, or of the form $\big((q, r, a), c, (q, r')\big)$, where $r' \in Q'$ and $c = \min\big\{\mathsf{dist}(a, v) \ :\ v \in \mathscr{L}(\mathcal{T}_{r,r'})\big\}$. Recall that $\mathcal{T}_{r,r'}$ is the DFA obtained from $\mathcal{T}$ by letting r be the initial state and r' be the unique final state. The initial vertex of the arena is the pair (q_0, r_0) (so Adam moves first). Observe that the final states of $\mathcal{R}$ and $\mathcal{T}$ do not play any relevant role in this definition: this is because $\mathcal{R}$ and $\mathcal{T}$ are assumed to be trimmed and the costs of moving from non-final states to final states are irrelevant for the asymptotic behavior. Furthermore, note that the game alternates between Adam and Eve, and only the second player can incur positive costs.

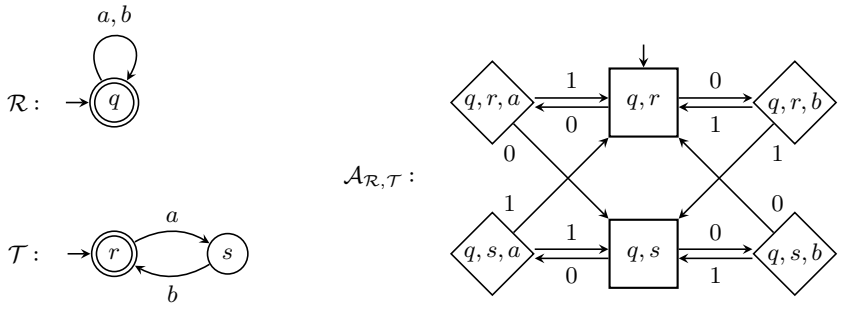


Figure 3.2: Two DFAs and the arena for the associated mean-payoff game.

Remark 2. *In order to avoid that players get stuck at some vertices of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ that have no outgoing edges, we tacitly assume that all states of the DFA $\mathcal{R}$ and $\mathcal{T}$ can reach non-transient states (i.e., states contained in some cycles). In particular, as the automata are also trimmed, we have that for all states $q \in Q$ and $r \in Q'$, both languages $\mathscr{L}(\mathcal{R}_{q,F})$ and $\mathscr{L}(\mathcal{T}_{r,F'})$ contain infinitely many words. Note that it is safe to make this assumption when considering the streaming asymptotic cost, as this cost is preserved when we remove the states that can only reach transient states.*

Below, we show that the value of the mean-payoff game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, multiplied by 2, coincides with the asymptotic aggregate cost in the streaming setting.

Theorem 3.3.2. *For all DFA $\mathcal{R}$ and $\mathcal{T}$, we have*

$$\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) \;=\; 2 \cdot \nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}$$

where $\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}$ is the value of the mean-payoff game over arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Moreover, it holds that $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T}))$ is rational.

Example 10. *Consider the restriction and target languages $R = (a + b)^*$ and $T = (ab)^*$, whose automata $\mathcal{R}$ and $\mathcal{T}$ and mean-payoff arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ are shown in Figure 3.2 (diamond nodes are owned by Eve and square nodes are owned by Adam). One can easily see that an optimal positional strategy for Adam is to play $(q, r) \xrightarrow{\mathsf{Adam}} (q, r, b)$ and $(q, s) \xrightarrow{\mathsf{Adam}} (q, s, a)$. With this optimal strategy we get that the value $\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}$ of the mean-payoff game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ is equal to $\frac{1}{2}$ and thus $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(R, T) = 1$. This value definitely contrasts with the non-streaming asymptotic cost between R and T, which is equal to $\frac{1}{2}$.*

Even if it seems natural that the value of the mean-payoff game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ determines the asymptotic cost $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T}))$, the proof of the Theorem 3.3.2 is not trivial. Indeed, the mean-payoff game corresponds directly to a version of the streaming repair problem where the input to the repair strategy is a sequence of prefixes of a *single infinite word* spelled by a run of $\mathcal{R}$. The core of the proof is to show a correspondence between the infinitary version of the streaming

repair problem and the original problem as stated in Section 3.1. This is done by proving two inequalities. In one case (Lemma 3.3.3) we show that $(\star)$ for an optimal strategy $\mathcal{Z}$ of Eve in the mean-payoff game, one can construct a streaming repair strategy $\mathcal{Z}'$ for $\mathcal{R}$ and $\mathcal{T}$ such that $\frac{\mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}'}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))}{2}$ does not exceed the penalty for Eve induced by her strategy $\mathcal{Z}$. Intuitively, the repair strategy $\mathcal{Z}'$ mimics Eve's strategy $\mathcal{Z}$ until the string terminates, at which point it performs additional insertions to get to a final state. For the other direction (Lemma 3.3.4) we consider a streaming repair strategy $\mathcal{Z}'$ for $\mathcal{R}$ and $\mathcal{T}$ with asymptotic cost $\mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}'}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))$ and we show that no strategy $\mathcal{Z}$ for Adam can guarantee a reward of more than $\frac{\mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}'}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))}{2}$. By the result from [EM79] mentioned above, this shows that Eve can enforce a penalty less than or equal to this amount. The limit on Adam's ability is shown by combating his strategy $\mathcal{Z}$ using the repair strategy $\mathcal{Z}'$. Putting these two directions together, we see that the optimal streaming repair strategy is produced by first computing Eve's optimal strategy, and then applying the transformation $(\star)$ described above; one can then argue that this strategy can be computed in polynomial time from $\mathcal{R}$ and $\mathcal{T}$.

Lemma 3.3.3. *For all DFA $\mathcal{R}$ and $\mathcal{T}$ and all streaming repair strategies $\mathcal{Z}$ for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$, we have $2 \cdot \nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}} \leq \mathsf{acost}^{aggr}_{\mathcal{Z}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))$.*

Proof. Let us fix two DFA $\mathcal{R} = (\Sigma, Q, \delta, q_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', r_0, F')$ and a transducer $\mathcal{Z} = (\Sigma, \Delta, Z, \kappa, z_0, \Omega)$ that implements a streaming repair strategy for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$. Let us also fix an optimal positional strategy $f : Q \times Q' \to E$ for Adam, where E is the set of edges of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

On the basis of the transducer $\mathcal{Z}$ and Adam's positional strategy f, we inductively construct (i) an infinite play π on $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, (ii) an infinite word $u \in \Sigma^\omega$, and (iii) an infinite run ρ of $\mathcal{Z}$ on u, as follows. The first edge of the play π is given by Adam's move $f(v_0) = (v_0, 0, v_1)$, where $v_0 = (q_0, r_0)$. Accordingly, the first symbol of the word u is the symbol a_1 that is contained in the vertex v_1 (note that $v_1 \in Q \times Q' \times \Sigma$). The run ρ of $\mathcal{Z}$ at the beginning is the empty sequence. As for the induction step, we first extend ρ and π, using the transducer $\mathcal{Z}$, and then we extend π and u, using Adam's strategy again. Formally:

- Given a prefix $(v_0, 0, v_1)\ (v_1, c_1, v_2)\ \dots\ (v_{2n}, 0, v_{2n+1})$ of π that ends in a vertex $v_{2n+1} = (q_{n+1}, r_n, a_{n+1})$ owned by Eve and given the corresponding prefix $a_1 \dots a_{n+1}$ of u, we extend the prefix of ρ from $z_0 \xrightarrow{a_1/w_1} \dots \xrightarrow{a_n/w_n} z_n$ to $z_0 \xrightarrow{a_1/w_1} \dots \xrightarrow{a_n/w_n} z_n \xrightarrow{a_{n+1}/w_{n+1}} z_{n+1}$, where $\kappa(z_n, a_{n+1}) = (w_{n+1}, z_{n+1})$. Accordingly, we extend the prefix of π by adding the edge $(v_{2n+1}, c_{n+1}, v_{2n+2})$, where $v_{2n+2} = (q_{n+1}, r_{n+1})$, r_{n+1} is the state of $\mathcal{T}$ reached from r_n after consuming the word w_{n+1}, and $c_{n+1} = \min\left\{\mathsf{dist}(a_{n+1}, w) \ :\ w \in \mathscr{L}(\mathcal{T}_{r_n, r_{n+1}})\right\}$.
- Similarly, given a prefix $(v_0, 0, v_1)\ (v_1, c_1, v_2)\ \dots\ (v_{2n+1}, c_{n+1}, v_{2n+2})$ of π that ends in a vertex $v_{2n+2} = (q_{n+1}, r_{n+1})$ owned by Adam, we extend it using Adam's positional strategy f, namely,

by adding the edge $f(v_{2n+2}) = (v_{2n+2}, 0, v_{2n+3})$. Accordingly, we extend the prefix of u from $a_1 \ \ldots \ a_{n+1}$ to $a_1 \ \ldots \ a_{n+1} \ a_{n+2}$, where a_{n+2} is the symbol contained in the vertex v_{2n+3}.

It is easy to check that the above definitions lead to an infinite play

$$\pi \ = \ (v_0, 0, v_1) \ (v_1, c_1, v_2) \ (v_2, 0, v_3) \ \ldots$$

over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, and an infinite run

$$\rho \ = \ z_0 \xrightarrow{a_1/w_1} z_1 \xrightarrow{a_2/w_2} \ \ldots$$

of the transducer $\mathcal{Z}$ on the word $u = a_1 \ a_2 \ \ldots$ such that, for every $n \in \mathbb{N}$, $c_n \leq \mathsf{dist}(a_n, w_n)$. The play π clearly respects Adam's optimal strategy and hence, by Theorem 3.3.1, we have $\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}} \leq \nu^{\pi}_{\mathsf{Adam}}$. Moreover, observe that every prefix $a_1 \ldots a_n$ of the infinite word u can be extended to a word $a_1 \ \ldots \ a_n \ w'_n$ that belongs to the restriction language $\mathscr{L}(\mathcal{R})$, where w'_n has length at most $|Q|$. By applying the various definitions and some basic rewriting, we easily obtain:

$$\begin{aligned}
2 \cdot \nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}} \ &\leq \ 2 \cdot \nu^{\pi}_{\mathsf{Adam}} \\
&= \ 2 \cdot \liminf_{n \to \infty} \ \frac{\sum_{i=1}^{n} c_i}{2 \cdot n} \\
&\leq \ \liminf_{n \to \infty} \ \frac{\sum_{i=1}^{n} \mathsf{dist}(a_n, w_n)}{n} \\
&\leq \ \limsup_{n \to \infty} \ \frac{\sum_{i=1}^{n} \mathsf{dist}(a_n, w_n)}{n} \\
&= \ \limsup_{n \to \infty} \ \frac{\mathsf{cost}^{\mathsf{aggr}}(a_1 \ \ldots \ a_n \ w'_n, \ \mathcal{Z})}{n + |w'_n|} \\
&\leq \ \mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})).
\end{aligned}$$

□

Lemma 3.3.4. *For all DFA $\mathcal{R}$ and $\mathcal{T}$, there is a transducer $\mathcal{Z}$, whose states are pairs of states of $\mathcal{R}$ and $\mathcal{T}$, that implements a streaming repair strategy for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$ and such that $\mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) \leq 2 \cdot \nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}$.*

Proof. We fix two DFA $\mathcal{R} = (\Sigma, Q, \delta, q_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', r_0, F')$ and an optimal positional strategy $g : Q \times Q' \times \Sigma \to E$ for Eve, where E is the set of edges of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. We then construct from that a transducer $\mathcal{Z} = (\Sigma, \Delta, V_{\mathsf{Adam}}, \kappa, v_0, \Omega)$ as follows:

- $V_{\mathsf{Adam}} = Q \times Q'$ is the set of vertices of $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ owned by Adam;
- κ is the function that maps any pair $(v, a) \in V_{\mathsf{Adam}} \times \Sigma$, with $v = (q, r)$, to the (unique) pair $(w, v) \in \Delta^* \times V_{\mathsf{Adam}}$, with $w = (q', r')$, that satisfies

– $g(v_a) = (v_a, c, v')$, with $v_a = (\delta(q,a), r, a)$ (note that $v_a \in V_{\mathsf{Eve}}$),

– $w \in \mathscr{L}(\mathcal{T}_{r,r'})$, with $\mathsf{dist}(a,w) = c$ (note that since (v_a, c, v') is an edge in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, there exist such a word w).

- $v_0 = (q_0, r_0)$ is the initial vertex of $\mathcal{G}_{\mathcal{R},\mathcal{T}}$,
- Ω is the function that maps any vertex $v = (q,r) \in V_{\mathsf{Adam}}$ to a word w from the language $\bigcup_{r' \in F'} \mathscr{L}(\mathcal{T}_{r,r'})$ (since $\mathcal{T}$ is pruned, there always exists such a word).

Observe that $\mathcal{Z}$ implements a streaming strategy for repairing $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ (it basically differs from Eve's strategy only in the use of the final output function Ω, which guarantees that the edited words belong to the target language $\mathscr{L}(\mathcal{T})$). Moreover, by definition, the states of the transducer $\mathcal{Z}$ range over the set $V_{\mathsf{Adam}} = Q \times Q'$.

Let us now consider a family of words $\left(u^{(n)}\right)_{n \in \mathbb{N}}$ from the restriction language $\mathscr{L}(\mathcal{R})$ such that

$$\mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) \;=\; \limsup_{n \to \infty} \; \frac{\mathsf{cost}^{\mathsf{aggr}}_{\mathcal{Z}}(u^{(n)})}{|u^{(n)}|}.$$

Moreover, let $L_1, \ldots, L_m$ be all the simple cycles of the transition graph of $\mathcal{Z}$ and let $\rho^{(n)}$ be the run of $\mathcal{Z}$ on the word $u^{(n)}$. We use the simple cycle decomposition Lemma 3.2.7 from Section 3.2 to find a partition of the domain of $\rho^{(n)}$ into (possibly non-convex) subsets $U_0^{(n)}, U_1^{(n)}, \ldots, U_m^{(n)}$ such that

- $|U_0^{(n)}|$ is uniformly bounded by the number K of states of $\mathcal{Z}$,
- for all $1 \leq i \leq m$, the sub-sequence $\rho^{(n)}|U_i^{(n)}$ is a repetition of the simple cycle L_i of $\mathcal{Z}$.

As usual, we denote by $\mathsf{occ}_i^{(n)}$ the number of repetitions of the simple cycle L_i in the sub-sequence $\rho^{(n)}|U_i^{(n)}$, namely, $\mathsf{occ}_i^{(n)} = \frac{|U_i^{(n)}|}{|L_i|}$. We have that

$$\limsup_{n \to \infty} \; \frac{\mathsf{cost}^{\mathsf{aggr}}_{\mathcal{Z}}(u^{(n)})}{|u^{(n)}|} \;\leq\; \limsup_{n \to \infty} \; \frac{\sum_{1 \leq i \leq m} \mathsf{occ}_i^{(n)} \cdot \mathsf{cost}(L_i) \;+\; K \cdot c_{\mathsf{max}}}{\sum_{1 \leq i \leq m} \mathsf{occ}_i^{(n)} \cdot |L_i| \;+\; K}$$

where $\mathsf{cost}(L_i)$ denotes the sum of the costs of the transitions in the simple cycle L_i and c_{max} denotes the maximum cost of a transition of $\mathcal{Z}$. Note that the additive terms $K \cdot c_{\mathsf{max}}$ and K above can be ignored when considering the limit for n tending to infinity. Moreover, it is easy to see that the following inequality holds

$$\limsup_{n \to \infty} \; \frac{\sum_{1 \leq i \leq m} \mathsf{occ}_i^{(n)} \cdot \mathsf{cost}(L_i)}{\sum_{1 \leq i \leq m} \mathsf{occ}_i^{(n)} \cdot |L_i|} \;\leq\; \max_{1 \leq i \leq m} \; \frac{\mathsf{cost}(L_i)}{|L_i|}.$$

For the sake of brevity, we denote by L some simple cycle among $L_1, \ldots, L_m$ that maximizes the ratio $\frac{\mathsf{cost}(L)}{|L|}$, namely,

$$\max_{1 \leq i \leq m} \; \frac{\mathsf{cost}(L_i)}{|L_i|} \;=\; \frac{\mathsf{cost}(L)}{|L|}.$$

We now construct a strategy for Adam in the mean-payoff game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ by following the simple cycle L. We denote by u_L be the word that forms the input of the simple cycle L and by u_0 any word that makes the DFA $\mathcal{R}$ move from its initial state q_0 to the state that appears in the first/last position of L (recall that the states of $\mathcal{Z}$ are pairs of states from $\mathcal{R}$ and $\mathcal{T}$). Clearly, the infinite word $u_0\ u_L^\omega$ induces an infinite run inside the automaton $\mathcal{R}$. Adam's strategy will follow precisely this infinite word, choosing at each round n the edge in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ that corresponds to the correct transition that consumes the n-th symbol of $u_0\ u_L^\omega$.

Pairing Adam's strategy given above with Eve's strategy, we obtain an infinite 'cyclic' play $\pi = (v_0, 0, v_1)\ (v_1, c_1, v_2)\ \ldots$ over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. By construction, the average cost incurred by Eve in following the play π' coincides with $\frac{\mathsf{cost}(L)}{2\cdot|L|}$, namely,

$$\frac{\mathsf{cost}(L)}{2\cdot|L|} = \nu^{\pi}_{\mathsf{Eve}}$$

Finally, recall that Eve's strategy was assumed to be optimal and hence, by Theorem 3.3.1,

$$\nu^{\pi}_{\mathsf{Eve}} \leq \nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}.$$

Summing up, we just proved that

$$\begin{aligned}
\mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})) &= \limsup_{n\to\infty} \frac{\mathsf{cost}^{\mathsf{aggr}}_{\mathcal{Z}}(u^{(n)})}{|u^{(n)}|} \\
&\leq \limsup_{n\to\infty} \frac{\sum_{1\leq i\leq m} \mathsf{occ}_i^{(n)}\cdot\mathsf{cost}(L_i)}{\sum_{1\leq i\leq m} \mathsf{occ}_i^{(n)}\cdot|L_i|} \\
&\leq \max_{1\leq i\leq m} \frac{\mathsf{cost}(L_i)}{|L_i|} \\
&= 2\cdot\nu^{\pi}_{\mathsf{Eve}} \\
&\leq 2\cdot\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}.
\end{aligned}$$

□

We now turn to the proof of Theorem 3.3.2.

Proof of Theorem 3.3.2. Let $\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}$ be the value of the mean-payoff game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. We know from Lemma 3.3.3 that for every streaming repair strategy $\mathcal{Z}$ for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$, $2\cdot\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}} \leq \mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))$. Since the asymptotic repair cost $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))$ in the streaming case is defined as the infimum of $\mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))$ over all streaming repair strategies $\mathcal{Z}$, we have

$$2\cdot\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}} \leq \mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})).$$

Conversely, we know from Lemma 3.3.4 that there is a streaming strategy $\mathcal{Z}$ for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$ such that $\mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})) \leq 2\cdot\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}$ and hence, since $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})) \leq \mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))$, we have

$$\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})) \leq 2\cdot\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}.$$

We have just shown that $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})) = 2\cdot\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}$. □

Regarding the complexity of solving mean-payoff games, we recall that the problem of deciding whether the value of an arbitrary mean-payoff game $\mathcal{G} = (V, E, v_0)$ is below a certain threshold is in coNP ∩ NP [GTW03]. Much recent work has focused on improving the exponential bounds on deterministic algorithms; for example, it can be done in $O(|V|^2\cdot|E|\cdot c_{\mathsf{max}})$, where c_{max} is the maximum weight of an edge of $\mathcal{G}$ [ZP96]. Even though the parameter c_{max} is exponential when the weights are represented in binary notation, when restricting to arenas of the form $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, this value never exceeds the total number of states of the DFA $\mathcal{T}$. This gives a polynomial time bound on the complexity of the problem of computing the value of the mean-payoff game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

Corollary 3.3.5. *The streaming asymptotic cost* $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))$ *can be computed in polynomial time for all DFA* $\mathcal{R}$ *and* $\mathcal{T}$*. Furthermore, this cost is achieved by a streaming transducer for* $\mathscr{L}(\mathcal{R})$ *and* $\mathscr{L}(\mathcal{T})$ *which can also be computed in polynomial time.*

Proof. Recall that from the results in [ZP96] the value $\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}$ is rational and it can be computed by a deterministic procedure that runs in time $O(|V|^2\cdot|E|\cdot c_{\mathsf{max}})$, where V is the set of vertices of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ (hence $|V| \le |Q|\cdot|Q'|\cdot(|\Sigma|+1)$), E is the set of edges of $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ (hence $|E| \le |V|^2$), and c_{max} is the maximum weight of an edge in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Since c_{max} never exceeds the number $|Q'|$ of states of the target automaton $\mathcal{T}$, this gives a polynomial time procedure for computing the value $\nu_{\mathcal{G}_{\mathcal{R},\mathcal{T}}}$ of the mean-payoff game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ (and hence the asymptotic cost $\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T}))$). Finally, in [ZP96] it was also shown a polynomial time procedure for computing positional optimal strategies of mean-payoff games in time $O(|V|^4\cdot|E|\cdot\log(\frac{|E|}{|V|})\cdot c_{\mathsf{max}})$. By the proof of Lemma 3.2.9, this positional strategy for $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ can easily be converted into a streaming transducer $\mathcal{Z}$ such that:

$$\mathsf{acost}^{\mathsf{aggr}}_{0-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})) = \mathsf{acost}^{\mathsf{aggr}}_{\mathcal{Z}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})).$$

□

We conclude this subsection by mentioning some natural generalizations of Theorem 3.3.2 and Corollary 3.3.5 related to streaming repair strategies with lookahead. Observe that in order to compute the asymptotic cost of an optimal streaming repair strategy with k-lookahead, where $k \in \mathbb{N}$ is a given parameter, it is sufficient to modify the definition of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ in such a way that Adam plays $(k+1)$-character windows representing substrings of an infinite word. This requires extending the set of vertices of the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ from $(Q\times Q')\cup(Q\times Q'\times\Sigma)$ to $(Q\times Q'\times\Sigma^k)\cup(Q\times Q'\times\Sigma^{k+1})$ and letting the game start from any vertex of the form (p_0,q_0,u_0), where p_0 is the initial state of $\mathcal{R}$, q_0 is the initial state of $\mathcal{T}$, and $u_0 \in \Sigma^k$. We denote by $\mathcal{G}^{k-\mathsf{lookahead}}_{\mathcal{R},\mathcal{T}}$ the new arena and by $\nu_{\mathcal{G}^{k-\mathsf{lookahead}}_{\mathcal{R},\mathcal{T}}}$ the value of the mean-payoff game associated with it. Following the same arguments of the proof of Theorem 3.3.2, one shows that:

$$\mathsf{acost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})) = 2\cdot\nu_{\mathcal{G}^{k-\mathsf{lookahead}}_{\mathcal{R},\mathcal{T}}}.$$

Following similar ideas presented in Corollary 3.3.5, one can show that $\mathsf{acost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T}))$ can be computed in polynomial time when the amount of lookahead k is fixed.

3.4 Conclusions

In this chapter, we have addressed the problem of computing the asymptotic cost of repairing regular languages in the non-streaming and streaming settings. It is surprising that the asymptotic cost in both settings is rational and computable.

In the non-streaming setting, we proved that the threshold problem for the asymptotic cost is between PSPACE and EXPTIME. This result can be made stronger [BPRb] by showing that the threshold problem for the asymptotic cost is PSPACE-hard even for regular languages specified by DFAs. Unfortunately, we leave as an open problem whether the algorithms for computing asymptotic cost in the non-streaming setting are optimal.

In contrast to the above results, we derive optimal online algorithms for editing one language into another in the streaming setting, where a finite lookahead is given, which are quite distinct from traditional edit distance algorithms based on dynamic programming. We think that this last result could be considered in practical scenarios when an optimal strategy with respect to the streaming repair cost could be needed.

We also left open the application of the asymptotic cost in other scenarios. Many papers (see related work below or in Section 2.6) have proposed distance functions between two sets of strings, which are appropriate for their own applications. These distance functions are *local* with respect to the languages or, more specific, they are intrinsically affected by singular instances of the set failing to see both languages as a whole. In our case, the asymptotic cost considers only the difference of the two languages for very long words given an overall picture of the two sets. However, it is worth noticing that the asymptotic cost is not a distance function and is asymmetric with respect to the restriction and target language which restrict its application. Despite this, it will be interesting to see further applications of the asymptotic cost in other scenarios.

Related work. The related work presented in Section 2.6 is also relevant for the asymptotic repair cost studied in this chapter. Here, we would like to highlight some connections of the asymptotic cost with *property testing* [AKNS01, Fis01, FMDR10]. As was previously mentioned, *property testing* is a line of research that studies the existence of ϵ-testers for model-checking problems. Recall that an ϵ-tester for a regular language L and an integer ϵ is a randomized algorithm that for every word w it only queries at most $O(1/\epsilon)$ *positions* of w and accepts with probability 1 whenever $w \in L$ and rejects with high probability whenever $\mathsf{dist}(w, L) \geq \epsilon \cdot |w|$. Although the existence of an ϵ-tester for the model-checking problem of regular languages was solved in [AKNS01], as far as we know no one has pointed out whether an ϵ-tester is always *meaningful* for every ϵ. For example, one can easily

show that if $\epsilon > \mathsf{acost}(\Sigma^*, L)$, then every ϵ-tester never rejects with high probability for words w with length greater than $C/(\epsilon - \mathsf{acost}(\Sigma^*, L))$ where C is a constant that only depends on L and not on the size of w. This means that if $\epsilon > \mathsf{acost}(\Sigma^*, L)$, then any ϵ-tester is not meaningful in the sense that it will only reject a finite number of words with high probability. Therefore, this gives an interval $(\mathsf{acost}(\Sigma^*, L), 1)$ for ϵ values that should not be considered if an ϵ-tester wants to be used in practice.

A similar problem related to the asymptotic cost of a distance automata was studied in [CD13]. In this paper, the authors studied the problem of deciding the *ϵ-approximation* of functions specified by distance automata. Specifically, the paper shows a partial algorithm that, for every $\epsilon \leq 1$ and for every two distance automata f and g, outputs:

- "yes" whenever $f(w) \leq (1-\epsilon) \cdot g(w)$ for all $w \in \Sigma^*$,
- "no" whenever $f(w) \not\leq g(w)$ for some $w \in \Sigma^*$, and
- either "yes" or "no", otherwise.

The algorithm is a partial solution to the problem of deciding whether $f \leq g$ for any distance automata f and g (which is known to be undecidable [Kro94]). Although the algorithm presented in [CD13] could possibly be used to compute the asymptotic cost of a subclass of distance automata, it is not clear how to exploit this algorithm given its incomplete behaviour. Furthermore, this algorithm is susceptible to "small" words and, in contrast, the asymptotic cost depends only on the behaviour of the distance automata in the long run. Finally, it is worth to notice that the techniques used in [CD13] are mostly algebraic and very different form the techniques used in this chapter.

It is important to note that some connections between online algorithm and determinization of distance automata have been studied by Aminof et all in [AKL10]. In this paper, the authors considers distance automata and their determinization in order to find online algorithms with a given competitive ratio with their offline versions. This work can be seen as a more general scenario of our streaming setting. Despite of this, in this work the authors have not considered the computation of the asymptotic ratio of the cost of a distance automata and the connection between the streaming setting and games.

We want to highlight that some interesting results presented in [BPRb] that relate the streaming asymptotic cost $\mathsf{acost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}$ with the lookahead parameter $k \in \mathbb{N}$ where not included here. Intuitively, the k-lookahead streaming asymptotic cost $\mathsf{acost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}(R, T)$ (associated with two languages R and T) is a non-increasing function of k and it is bounded from below by the non-streaming asymptotic cost $\mathsf{acost}(R, T)$. Then it will be interesting to compute the limit of the streaming asymptotic cost $\mathsf{acost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}(R, T)$ as the lookahead parameter k gets bigger. In [BPRb] we were not able to compute the exact value of this limit, nor to prove that $\mathsf{acost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}(R, T)$ stabilizes for

sufficiently large k, as it seems reasonable. However, it was shown how to solve a simpler problem, that consists of deciding given two DFA R and T and a rational threshold ν whether there is $k \in \mathbb{N}$ such that $\mathsf{acost}^{\mathsf{aggr}}_{k-\mathsf{lookahead}}(R,T) < \nu$.

Chapter 4

Bounded repairing of trees

In this chapter, we turn our attention from repairing strings to repairing trees. We consider the bounded repair problem between tree languages which can be specified by XML schemas or, more general, by regular tree languages [Sch07]. By formalizing the edit distance between unranked ordered trees, we will define the *bounded repair problem*, that is, the problem of deciding bounded repairability between two given tree languages R and T. Our main result shows that is decidable whether or not R can be repaired into T with a uniformly bounded number of edits.

For regular languages of strings, the bounded repair problem was resolved in Chapter 2: there it was shown that the problem is coNP-complete when the languages are represented by deterministic finite state automata, and a characterization of bounded repairability was given using a coverability relation between chains of connected components of the automata. In the case of tree languages, the problem turns out to be much more complex, both in terms of complexity and in terms of proof techniques that are needed to resolve it. We will provide a characterization of bounded repairability that exploits a suitable notion of component of a stepwise tree automaton [CNT04], i.e., a form of automaton that turns out to be particularly convenient for analysing repairs. An additional complication for the tree case is that we need to consider structures of components of stepwise tree automata that take the form of trees, rather than chains. Our characterization of the bounded repairability of R into T requires that every component structure of R can be "covered" by a component structure of T. The notion of covering is subtle, and the proof that it captures bounded repairability requires lifting the notion of edit from the level of the individual trees to the level of the component trees associated with the automata for R and T.

Once we have our characterization, we can apply it to get decidability of the bounded repair problem, and with some additional optimization we can give tight complexity bounds. It turns out that, in contrast with the string setting, deciding the bounded repair problem for regular tree languages is equally complex no matter whether the finite tree automata is deterministic or non-deterministic, or whether it is given by a DTD or even by a non-recursive DTD. We show that for all these cases the bounded repair problem is coNEXPTIME-complete. For the sake of positive results,

we study further restrictions over tree specifications and show that bounded repairability is much simpler to check when the restriction language R is trivial (i.e., the class of all trees).

We organize this chapter as follows: in Section 4.1 we give all the background needed on trees and regular tree languages. Then we use this background in Section 4.2 to define the bounded repair problem over trees. In Section 4.3 we give the formal statement of our main result, that is, a characterization of exactly which pairs of schemas are bounded repairable. We take up Section 4.4 to present the proof of this result. Then we use our characterization in Section 4.5 to study the complexity of the bounded repair problem over trees, while in Section 4.6 we give a different characterization of bounded repairability for the case where the restriction language is trivial. Finally, in Section 4.7 we present our conclusions and show some related work.

4.1 Regular tree specifications

In this chapter we study the bounded repair problem over *finite unranked ordered trees* whose nodes are labelled over a finite alphabet Σ (see Chapter 8 in [CDG+07] for a survey). Formally, the set of *unranked ordered* Σ*-trees* (or just trees) is inductively defined as follows: (1) every $a \in \Sigma$ is a Σ-tree; and (2) if $a \in \Sigma$ and $t_1, \ldots, t_n$ are Σ-trees for some $n \in \mathbb{N}$, then $a(t_1, \ldots, t_n)$ is a Σ-tree. Furthermore, we call a sequence of σ-trees $t_1 \cdot \ldots \cdot t_n$ a Σ*-forest* or just a forest. Notice that we do not impose any a priori bound on the number of children of a node in a Σ-tree or Σ-forest. As an example, the left-hand side of Figure 4.1 shows a Σ-tree for $\Sigma = \{r, a, b, c, d\}$. We denote by $\mathcal{T}_\Sigma$ the set of all Σ-trees. We usually call a subset $L \subseteq \mathcal{T}_\Sigma$ a *tree language* over Σ.

It is useful to identify nodes of an unranked tree/forest by sequences of positive natural numbers. Formally, for every $t \in \mathcal{T}_\Sigma$ we define the set $\mathsf{nodes}(t) \subseteq \mathbb{N}^*$ recursively as follows: if $t = a(t_1, \ldots, t_n)$ with $a \in \Sigma$, $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$ and $n \geq 0$, then $\mathsf{nodes}(t) = \{\epsilon\} \cup \{i \cdot x \mid x \in \mathsf{nodes}(t_i) \wedge 1 \leq i \leq n\}$. One can easily extend this definition from trees to forests: $\mathsf{nodes}(f) = \{i \cdot x \mid x \in \mathsf{nodes}(t_i) \wedge 1 \leq i \leq n\}$ for a forest $f = t_1 \cdot \cdots \cdot t_n$ and $n \geq 0$. Notice that the roots of a forest are represented by singleton sequences. In particular, the empty sequence does not belong to the nodes of a forest. Finally, for any node $x \in \mathsf{nodes}(t)$ we denote by $t(x)$ the label of x on t.

Given a Σ-tree t and a node $x \in \mathsf{nodes}(t)$, we highlight some particular set of nodes related to the position of x on t. We denote the *parent of* x by $\mathsf{parent}_t(x) \in \mathsf{nodes}(t)$ such that $x = \mathsf{parent}_t(x) \cdot i$ for some $i \in \mathbb{N}$. Notice that for the root node ϵ the parent node $\mathsf{parent}_t(\epsilon)$ is not defined. We also denote the set of children of x by $\mathsf{children}_t(x) = \{x \cdot i \in \mathsf{nodes}(t) \mid i \in \mathbb{N}\}$. Furthermore, we denote the first and last child of x by $\mathsf{first\text{-}child}_t(x)$ and $\mathsf{last\text{-}child}_t(x)$, respectively (i.e. $\mathsf{first\text{-}child}_t(x) = x \cdot 1$ and $\mathsf{last\text{-}child}_t(x) = x \cdot n$ for $n \in \mathbb{N}$ such that $x \cdot (n+1) \notin \mathsf{children}_t(x)$). We usually omit t on $\mathsf{parent}_t(x)$ and $\mathsf{children}_t(x)$ if t is understood from the context. For example, in Figure 4.1 we can check that for the d-node 2 it holds that $\mathsf{parent}(2) = \epsilon$ and $\mathsf{children}(2) = \{2 \cdot 1, 2 \cdot 2, 2 \cdot 3, 2 \cdot 4\}$.

Finally, we introduce the partial orders $\preccurlyeq_t^{\mathsf{anc}}$ and $\preccurlyeq_t^{\mathsf{post}}$ (also known as post-order traversal) on the nodes of a tree t which are induced by the standard *ancestor relation* and *post-order* of t, respectively. The ancestor relation $\preccurlyeq_t^{\mathsf{anc}}$ is defined as the prefix order on sequences over $\mathbb{N}$, that is, $x \preccurlyeq_t^{\mathsf{anc}} y$ iff x is a prefix of y for every $x, y \in \mathsf{nodes}(t)$. In addition, we define $x \preccurlyeq_t^{\mathsf{post}} y$ iff y is a prefix of x or there exist prefixes x' and y' for x and y, respectively, such that $|x'| = |y'|$ and $x' \leq y'$ where $\leq$ is the alphabetical order on sequences over $\mathbb{N}$. Notice that $\preccurlyeq_t^{\mathsf{post}}$ is a total order over $\mathsf{nodes}(t)$. For example, in Figure 4.1 it holds that $1 \preccurlyeq_t^{\mathsf{post}} 2 \cdot 1$ where 1 is the only a-child of r and $2 \cdot 1$ is the only b-node.

DTDs. In this chapter we consider regular tree specifications to represent tree languages. In general, we focus in the model of finite tree automata which defines the class of *regular tree languages* (see below). We also consider less expressive tree specifications such as XML Document Type Definitions in order to study sub-classes of tree languages or just to give simple examples of regular tree languages.

An *XML Document Type Definitions (DTDs)* is defined as a tuple $D = (\Sigma, d, S)$ where Σ is a finite alphabet, d is a function that maps Σ-symbols to regular expressions over Σ, and $S \subseteq \Sigma$ is the set of initial symbols [CDG+07]. A tree t satisfies D if $t(\epsilon) \in S$ and every $x \in \mathsf{nodes}(t)$ satisfies $t(x \cdot 1) \cdots t(x \cdot n) \in \mathscr{L}(d(t(x)))$ where $\mathsf{last\text{-}child}_t(x) = x \cdot n$ for some $n \in \mathbb{N}$. We denote by $\mathscr{L}(D) \subseteq \mathcal{T}_\Sigma$ the language of Σ-trees satisfying D. We usually omit the initial symbol in the definition of a DTD if this symbol is understood from the context (e.g. it is the first symbol in the list for rules). As an example, consider the DTD:

$$
\begin{array}{lrcl}
D: & r & \to & a\, d \\
 & a & \to & a + \epsilon \\
 & d & \to & b\, c^* \\
 & b & \to & a \\
 & c & \to & \epsilon
\end{array}
$$

One can easily check that the tree at the left-hand side of Figure 4.1 satisfies D.

It is well known that DTDs defines a subclass of regular tree languages [MN07]. We assume this result throughout this chapter. Interestingly, most of our lower complexity bounds holds for the class of tree languages defined by DTDs (see Section 4.5 for more details).

We also consider the classes of non-recursive and deterministic DTDs that has been extensively studied before [SV02, MNSB06]. A DTD $D = (\Sigma, d, S)$ is *non-recursive* if its dependency graph (i.e., the graph that connects a letter a to a letter b whenever b occurs in the language $d(a)$) is acyclic. Similarly, a DTD D is *deterministic* if each $d(a)$ is represented in terms of DFAs for all $a \in \Sigma$. Deterministic DTDs are known in the literature as one-unambiguous DTDs and are the closer formalism to the DTD-specifications considered in practice [BKW98].

Curried trees and contexts. Throughout this chapter we use the *curry encoding* to represent unranked trees. The curry encoding [CNT04, MN07], also known as the extension encoding, represents an unranked tree labeled with elements of Σ as a binary tree whose inner nodes are labeled

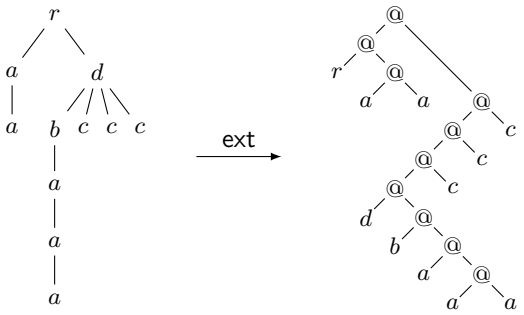


Figure 4.1: Curry encoding of an unranked tree.

with a distinguished symbol @ and leaves with elements of Σ. The encoding, denoted ext, is defined as follows (with $a \in \Sigma$):

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

To simplify the notations, we write the symbol @ as an infix, left-associative operator. Figure 4.1 illustrates the encoding of an unranked tree. The inverse ext^{-1} of the encoding is defined by providing the symbol @ with the semantics of the extension operator on unranked tree and by evaluating the expression in a bottom-up fashion, i.e., $\mathsf{ext}^{-1}(a) = a$ and $\mathsf{ext}^{-1}(a@t_1@\ldots@t_n) = a(\mathsf{ext}^{-1}(t_1), \ldots, \mathsf{ext}^{-1}(t_n))$.

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

Observe that the root node of an unranked tree corresponds to the left-most leaf in the corresponding curried tree. Other remarks follow [CNT04, MN07]. We point out that ext is a one-to-one mapping between the set of unranked trees and the set of curried trees. We also note that there is a one-to-one correspondence between the nodes of an unranked tree and the leaves of the curried encoding. Moreover, the *yield* of a curried tree, i.e., the sequence of leaves taken from left to right, corresponds to the standard left-to-right pre-order traversal of the unranked tree. Another observation follows from the semantics of the extension operator: the inner nodes of a curried tree, labeled with @, correspond to the edges of the unranked tree.

By the previous discussion, in the sequel of this chapter we use the curry encoding to represent Σ-trees. Furthermore, by a slight abuse of notation we denote by $\mathsf{nodes}(t)$ the nodes of the curry encoding of a Σ-tree t (i.e. $\mathsf{nodes}(\mathsf{ext}(t))$) and by $\mathcal{T}_\Sigma$ the set of all *curried trees* over Σ.

Stepwise tree automata. We use stepwise tree automata [CNT04, MN07] to specify regular tree languages. These are essentially bottom-up tree automata running over the curry encodings of trees [CDG+07]. Formally, a *stepwise automaton* is a tuple $\mathcal{A} = (\Sigma, Q, \delta, \delta_0, F)$, where:

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

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

A *run* of a stepwise automaton $\mathcal{A} = (\Sigma, Q, \delta_0, \delta, F)$ on a tree $t \in \mathcal{T}_\Sigma$ is a function $\rho : \mathsf{nodes}(t) \to Q$ such that (1) for every leaf node x, $\rho(x) \in \delta_0(t(x))$, and (2) for every inner node x, $\rho(x) \in \delta(\rho(x \cdot 1), \rho(x \cdot 2))$ (recall that we represent t with its curry encoding). A run ρ is *accepting* if $\rho(\varepsilon) \in F$. The language recognized by $\mathcal{A}$, denoted $\mathscr{L}(\mathcal{A})$, is the set of all trees $t \in \mathcal{T}_\Sigma$ on which $\mathcal{A}$ has an accepting run.

Example 11. *As a running example, consider the two DTDs:*

$$
\begin{array}{lrcl}
D: & r & \to & a\,d \\
 & a & \to & a + \epsilon \\
 & d & \to & b\,c^* \\
 & b & \to & a \\
 & c & \to & \epsilon
\end{array}
\qquad\qquad
\begin{array}{lrcl}
D': & r & \to & d\,c^* \\
 & d & \to & a\,a \\
 & a & \to & a + b \\
 & b & \to & \epsilon \\
 & c & \to & \epsilon
\end{array}
$$

The following two stepwise automata capture (modulo the curry encoding) the languages defined by the previous DTDs (the underlined states are final and any rule using $q^a_?$ translates to two rules using q^a_0 and q^a_1):

$$
\begin{array}{lrcl}
\mathcal{R}: & r & \to & p^r_0 \\
 & a & \to & p^a_0 \\
 & d & \to & p^d_0 \\
 & b & \to & p^b_0 \\
 & c & \to & q^c_0
\end{array}
\qquad
\begin{array}{lrcl}
 & p^r_0 @ p^a_? & \to & p^r_1 \\
 & p^r_1 @ p^d_1 & \to & \underline{p^r_2} \\
 & p^a_0 @ p^a_? & \to & p^a_1 \\
 & p^d_0 @ p^b_1 & \to & p^d_1 \\
 & p^d_1 @ p^c_0 & \to & p^d_1 \\
 & p^b_0 @ p^a_? & \to & p^b_1
\end{array}
\qquad\qquad
\begin{array}{lrcl}
\mathcal{T}: & r & \to & q^r_0 \\
 & d & \to & q^d_0 \\
 & a & \to & q^a_0 \\
 & b & \to & q^b_0 \\
 & c & \to & q^c_0
\end{array}
\qquad
\begin{array}{lrcl}
 & q^r_0 @ q^d_2 & \to & \underline{q^r_1} \\
 & \underline{q^r_1} @ q^c_0 & \to & \underline{q^r_1} \\
 & q^d_0 @ q^a_1 & \to & q^d_1 \\
 & q^d_1 @ q^a_1 & \to & q^d_2 \\
 & q^a_0 @ q^a_1 & \to & q^a_1 \\
 & q^a_0 @ q^b_0 & \to & q^a_1
\end{array}
$$

Figure 4.2 presents the (accepting) runs of the automata $\mathcal{R}$ and $\mathcal{T}$ on the curry encodings of some trees t and t', respectively.

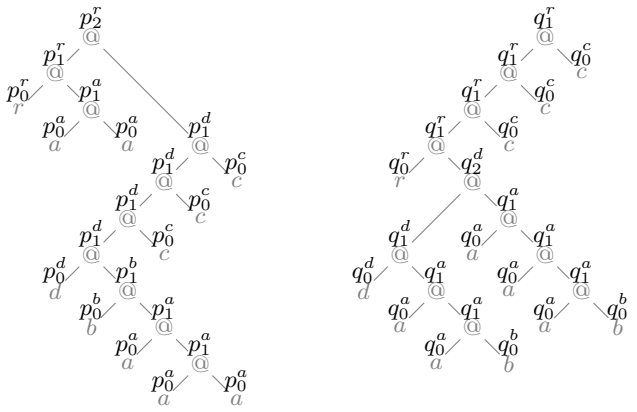


Figure 4.2: Runs of two stepwise tree automata over curry trees.

Stepwise automata capture exactly the class of regular (unranked) tree languages [CNT04] and they are more succinct than many other classes of tree automata [MN07]. Other models that capture the same class of languages, such as *unranked tree automata* are more frequently used (see [Sch07] for a survey). Since unranked tree automata can be converted into stepwise tree automata in polynomial time, algorithms for analyzing stepwise automata provide the same complexity bounds for unranked tree automata – all of our theorems will apply to unranked tree automata as well as stepwise tree automata. The main advantage of using stepwise automata in our proofs is due to their ability of capturing the cyclic behavior of a regular tree language, defined via a suitable notion of strongly connected component (see Section 4.3 for more details).

In the sequel, we will work with *trimmed* stepwise tree automata only. We assume that every state of an automaton appears in some accepting run. That is, a stepwise tree automaton $\mathcal{A} = (\Sigma, Q, \delta, \delta_0, F)$ is *trimmed* if for every state $q \in Q$ there exist $t \in \mathcal{T}_\Sigma$ and an accepting run ρ of $\mathcal{A}$ over t such that $\rho(x) = q$ for some $x \in \mathsf{nodes}(t)$. Observe that every stepwise tree automaton can be trimmed in linear time [CDG+07]. Given that all problems considered in this chapter are at least PTIME-hard, we can assume without loss of generality that all stepwise tree automata in the sequel are trimmed.

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

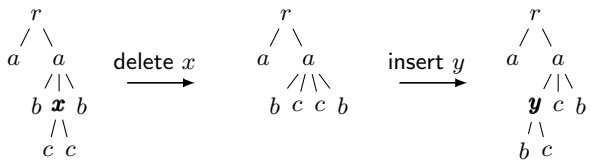


Figure 4.3: Edit operations on unranked trees.

4.2 The bounded repair problem for trees

We repair trees by using the standard set of edit operations over nodes [Tai79, Bil05]. We briefly recall the definitions of the standard edit operations on unranked trees which are extensions of the edit operations over words (see Section 2.2). The first operation, called *deletion*, consists of removing a distinguished (non-root) node x from a tree t and promoting its subtrees as children of its parent. The second operation, called *insertion*, consists of adding a new node x in an unranked tree t, with a possible adoption of a list of subsequent children from the parent of x. Figure 4.3 gives an example of these two operations. The last operation, called relabelling, consists of modifying the label of a node x to a new label in Σ. These are the standard edit-operations that are used to define the edit-distance between trees (see [Bil05] for a survey). Note that the operation of *relabeling* a node in an unranked tree, which is sometimes used as a standard edit operation, is subsumed by the insertion and deletion of nodes. We denote by $\mathsf{dist}(t, t')$ the minimum number of edits operations that are needed for transforming t into t' given two unranked trees t and t'.

Similar to the string case, we are interested in studying the *bounded repair problem* for regular tree languages. First of all, we extend the setting presented in Chapter 4 from strings to trees. We consider two finite alphabets Σ and Δ and regular languages $R \subseteq \Sigma^*$ and $T \subseteq \Delta^*$, called the *restriction* and *target* languages, respectively. Furthermore, we redefine a *repair strategy* as any function from trees in R to trees in T for any tree languages R and T. For a repair strategy f and a tree $t \in R$, we define $\mathsf{cost}(t, f) = \mathsf{dist}(t, f(t))$ to be the (absolute) *cost of* f *on* t.

We study the *bounded repair problem* for regular tree languages, which consists of deciding, given two regular tree languages R and T, whether there exists a repair strategy that transforms any tree $t \in R$ into a tree $t' \in T$ using a finite and uniformly bounded number of edits. Formally, given two regular tree languages R and T, we define

$$\mathsf{cost}(R, T) \ =^{\text{def}} \ \sup_{t \in R} \ \min_{t' \in T} \ \mathsf{dist}(t, t').$$

to be the *worst-case cost of repairing* R *into* T. Note that $\mathsf{cost}(R, T)$ can be equally described as the minimum over all repair strategies f for R and T of the worst-case cost of f. If $\mathsf{cost}(R, T)$ is finite, then we say that R *is bounded repairable into* T and we write $\mathsf{cost}(R, T) < \infty$ for short. During this chapter, we will study how to decide the bounded repair problem for input languages specified by means of stepwise tree automata and DTDs.

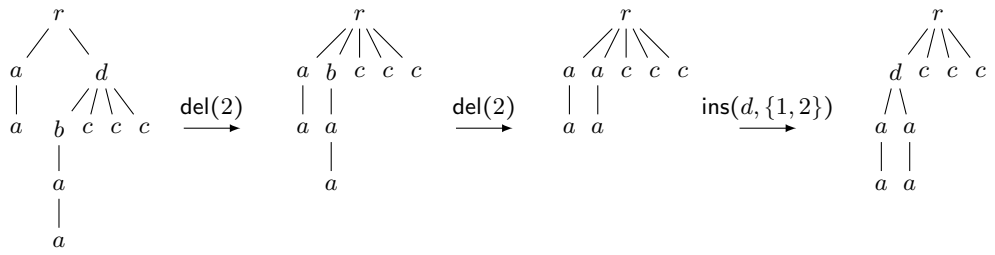


Figure 4.4: Example of how to repair an unranked tree satisfying D into D'.

Example 11 (Continued)**.** *Consider the two DTDs D and D' that were presented previously. For the tree languages specified by D and D' one can check $\mathscr{L}(D)$ is bounded repairable into $\mathscr{L}(D')$. Figure 4.4 shows how to repair a tree satisfying D into D' with only three edits. Actually, any tree $t \in \mathscr{L}(D)$ can be repaired into a tree $t' \in \mathscr{L}(D')$ with at most three edit operations and, thus, $\mathsf{cost}(\mathscr{L}(D), \mathscr{L}(D')) < \infty$. Indeed, we have to delete from t the only b- and d-nodes and then insert a d-node over the two chain sequences of a-nodes under r. It can be checked (by also following Figure 4.4) that the result of these three operations is a tree $t' \in \mathscr{L}(D')$.*

Example 12. *Consider the following DTDs:*

$$R: \begin{array}{lcl} r & \to & a \\ a & \to & b^* \\ b & \to & \epsilon \end{array} \qquad T: \begin{array}{lcl} r & \to & a \\ a & \to & b^*\, c \\ b & \to & \epsilon \\ c & \to & \epsilon \end{array}$$

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

Note that the bounded repair relation $\mathsf{cost}(R, T) < \infty$ satisfies some key properties that are straightforward to prove and which will be used later on. These properties are:

1. Subset-subsumption, i.e. $R \subseteq T$ implies $\mathsf{cost}(R, T) < \infty$.

2. Transitivity, i.e. $\mathsf{cost}(R, T) < \infty$ and $\mathsf{cost}(T, S) < \infty$ imply $\mathsf{cost}(R, S) < \infty$, and

3. Union-compatibility, i.e. $\mathsf{cost}(R, T) < \infty$ and $\mathsf{cost}(R', T') < \infty$ imply $\mathsf{cost}(R \cup R', T \cup T') < \infty$.

This first property, *subset-subsumption*, is trivial to prove given that $\mathsf{cost}(R, T) = 0$ iff $R \subseteq T$. For the transitivity property, one can easily check that function dist is a metric over trees and then satisfies the triangle inequality (i.e. $\mathsf{dist}(t, t') \leq \mathsf{dist}(t, t'') + \mathsf{dist}(t'', t')$ for any $t, t', t'' \in \mathcal{T}_\Sigma$). This implies that $\mathsf{dist}(r, s) \leq \mathsf{cost}(R, T) + \mathsf{cost}(T, S)$ given that $\mathsf{dist}(r, t) \leq \mathsf{cost}(R, T)$ and $\mathsf{dist}(t, s) \leq \mathsf{cost}(T, S)$ for

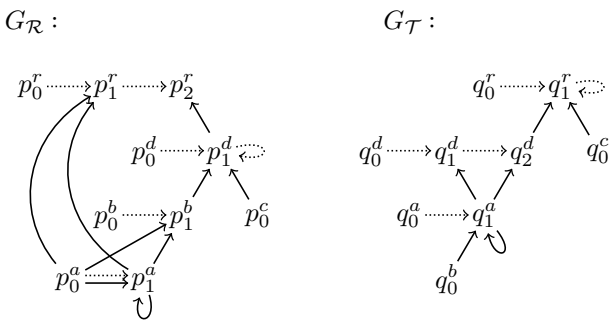


Figure 4.5: Transition graphs of two stepwise tree automata.

any $r \in R$, $t \in T$, and $s \in S$. Thus, we conclude that $\mathsf{cost}(R, S)$ is also bounded and the transitivity property is proved. Finally, the union-compatibility follows directly from the definition of worst-case cost of repairing a restriction into a target language. Indeed, if $\mathsf{cost}(R, T) < \infty$ and $\mathsf{cost}(R', T') < \infty$, then $\mathsf{cost}(R \cup R', T \cup T') \leq \min\{\mathsf{cost}(R, T), \mathsf{cost}(R', T')\}$ and we conclude that $\mathsf{cost}(R \cup R', T \cup T')$ is bounded as well.

4.3 Characterization of bounded repairability

In this section we give an effective characterization of the bounded repairability relation between regular tree languages. Similarly to the word case in Chapter 2, this characterization is based on the notion of strongly connected component of the transition graph of a stepwise automaton. In the word case, a suitable coverability relation between chains of components is used to characterize bounded repairability. Because here we work with trees, we need to generalize the notion of coverability to a relation over the so-called synopsis trees, i.e., full binary trees with nodes labeled by strongly connected components.

4.3.1 Components of stepwise automata

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

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

We call the edges in E_v *vertical* and the edges in E_h *horizontal*. Note that an edge may be both vertical and horizontal. As an example, Figure 4.5 depicts the transition graphs of the automata $\mathcal{R}$ and $\mathcal{T}$ of Example 11 (dashed arrows represent horizontal edges, solid arrows represent vertical edges).

Recall that a set X of nodes of a graph $G = (V, E)$ is a *strongly connected component* (or simply a *component*) iff X is maximal such that for every two $x, y \in X$, there exists a path from x to y visiting nodes only in X. By $\mathsf{SCC}(\mathcal{A})$ we denote the set of all strongly connected components in the

transition graph of $\mathcal{A}$. Similar to the word case, we associate with each component $X \in \mathsf{SCC}(\mathcal{A})$ the language $\mathscr{L}(\mathcal{A} \mid X)$ of contexts that are realizable within X:

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

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

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

As an example, consider again the transition graphs of Figure 4.5. All components except $\{p_1^d\}$, $\{p_1^a\}$, $\{q_1^r\}$, and $\{q_1^a\}$ are trivial. The components $\{p_1^d\}$ and $\{q_1^r\}$ are non-trivial horizontal, since they both realize the contexts $\bullet$, $\bullet@c$, $(\bullet@c)@c$, ... The components $\{p_1^a\}$ and $\{q_1^a\}$ are non-horizontal, since they both realize the contexts $\bullet$, $a@\bullet$, $a@(a@\bullet)$, ...

4.3.2 Synopsis trees

We now introduce a suitable structure that eases the characterization of bounded repairability, namely, a synopsis tree. This structure is the generalization of the chain of components used in Theorem 2.3.1 to characterize bounded repairability in the word case. Formally, a *synopsis tree* of an automaton $\mathcal{A}$ is a full binary tree whose nodes are labeled with elements of $\mathsf{SCC}(\mathcal{A})$. The tree language $[\![\tau]\!]_\mathcal{A}$ of a synopsis tree τ of $\mathcal{A}$ is the language of curried trees recursively defined as follows:

$$\begin{aligned}
[\![X]\!]_\mathcal{A} &= \Big\{C \circ a \;\Big|\; C \in \mathscr{L}(\mathcal{A} \mid X),\; a \in \Sigma\Big\} \\
[\![X(\tau_1, \tau_2)]\!]_\mathcal{A} &= \Big\{C \circ (t_1@t_2) \;\Big|\; C \in \mathscr{L}(\mathcal{A} \mid X),\; t_1 \in [\![\tau_1]\!]_\mathcal{A},\; t_2 \in [\![\tau_2]\!]_\mathcal{A}\Big\}
\end{aligned}$$

with $X \in \mathsf{SCC}(\mathcal{A})$. Figure 4.6 contains two synopsis trees τ and σ, respectively for the automata $\mathcal{R}$ and $\mathcal{T}$ of Example 11. Note that the tree language $[\![\tau]\!]_\mathcal{A}$ defined by a synopsis tree τ of $\mathcal{A}$ are all the combination of contexts in $\bigcup_{u \in \mathsf{nodes}(\tau)} \mathscr{L}(\mathcal{A} \mid \tau(u))$ that follows the "macro" structure of τ.

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

Definition 4.3.1. *A* primitive synopsis tree *of an automaton* $\mathcal{R} = (\Sigma, Q, \delta, \delta_0, F)$ *is a synopsis tree* τ *of* $\mathcal{R}$ *such that:*

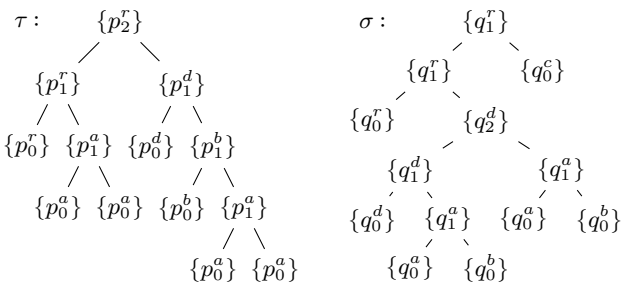


Figure 4.6: Synopsis trees for stepwise tree automata $\mathcal{R}$ and $\mathcal{T}$.

1. *τ respects the transition function of $\mathcal{R}$, i.e., for all nodes x, $x \cdot 1$, and $x \cdot 2$ in τ, there exist some states $q \in \tau(x)$, $q_1 \in \tau(x \cdot 1)$, and $q_2 \in \tau(x \cdot 2)$ such that $q \in \delta(q_1, q_2)$;*

2. *every internal node of τ has label different from the labels of its children, i.e., for all nodes x, $x \cdot 1$, and $x \cdot 2$ in τ, $\tau(x \cdot 1) \neq \tau(x) \neq \tau(x \cdot 2)$.*

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

Basically, a primitive synopsis tree τ of $\mathcal{A}$ follows the SCC-structure of $G_{\mathcal{A}}$ and is reduced, i.e. all labels following a branch of τ are different. The tree τ depicted in Figure 4.6 is a primitive synopsis tree. In particular, this tree respects the transitions of the run of the automaton $\mathcal{R}$ on the left-hand side of Figure 4.2.

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

Lemma 4.3.2. *For any stepwise tree automaton $\mathcal{A}$, it holds that:*

$$\mathscr{L}(\mathcal{A}) \subseteq \bigcup_{\tau \in \mathsf{PST}(\mathcal{A})} [\![\tau]\!]_{\mathcal{A}}.$$

Before going into the technical proof of Lemma 4.3.2, we illustrate the main arguments of this proof on the tree t from Figure 4.1 and the automaton $\mathcal{R}$ from Example 11. For this we consider the accepting run of $\mathcal{R}$ of Figure 4.2 and we use it to decompose t into a binary tree of contexts, where each context is realized by some SCC of $\mathcal{R}$. We present this decomposition in Figure 4.7, where for better visualization we place the states of the run not on the nodes but on the edges above them and we add a virtual edge for the root.

The decomposition procedure works in a recursive manner and begins at the root node. When executed at a node x with a state q (which belongs to some component $X \in \mathsf{SCC}(\mathcal{R})$), the procedure

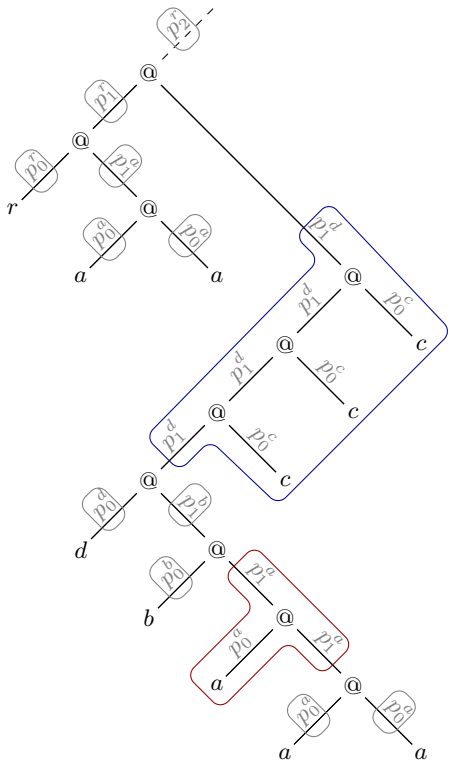


Figure 4.7: Decomposition of a curry tree into contexts.

creates an empty context $C = \bullet$ which spans the edge above the node. If a child of x has a state that belongs to the same component X, then the procedure moves to this node and expands the context C to span the edge above the node child and the whole subtree rooted at the opposite child. This step is repeated iteratively until the procedure reaches a leaf node or a node whose both children have states not in X. In the latter case, the procedure is called recursively on both children nodes, creating two separate children context nodes.

Clearly, the contexts created during the decomposition are realized by some components of $\mathcal{R}$ and the structure of these components takes the form of a synopsis tree. For instance, for the decomposition of Figure 4.7 the resulting synopsis tree is τ in Figure 4.6. Naturally, this synopsis tree respects the transitions of the run of $\mathcal{R}$ on the input tree. Furthermore, since every context has been maximally expanded, every node of the synopsis tree does not share the same label with any of its children. This shows that the constructed synopsis tree is primitive.

Proof of Lemma 4.3.2. Fix a stepwise automaton $\mathcal{A} = (\Sigma, Q, \delta, \delta_0, F)$, a tree $t \in \mathcal{L}(\mathcal{A})$, and the accepting run ρ of $\mathcal{A}$ on t. Below, we define a primitive synopsis tree τ from t and ρ such that $t \in [\![\tau]\!]_{\mathcal{A}}$. We construct τ inductively by starting with an initial synopsis tree τ_0 matching t and then iteratively apply a simple contraction operation until we obtain a primitive synopsis tree. In this process, we also maintain a matching of τ against t and ρ: essentially a structural proof that $t \in [\![\tau]\!]_{\mathcal{A}}$

that additionally shows that nodes of τ witness the transition function of $\mathcal{A}$ over t. Intuitively, a matching essentially represents the tree t as a composition of contexts that are realized by the corresponding nodes of τ. We shall identify a context within t using two anchor nodes: one for the root of the context and the other one for the placeholder $\bullet$. Formally, a *matching* of τ against t and ρ is a function $\vartheta : \mathsf{nodes}(\tau) \to \mathsf{nodes}(t) \times \mathsf{nodes}(t)$ that:

1. for every $x \in \mathsf{nodes}(\tau)$ if $\vartheta(x) = (y, y')$, then $y \preccurlyeq_t^{\mathsf{anc}} y'$;
2. $\vartheta(\varepsilon) = (\varepsilon, y)$;
3. for every $x, x \cdot i \in \mathsf{nodes}(\tau)$ if $\vartheta(x) = (y, y')$, then $\vartheta(x \cdot i) = (y' \cdot i, y'')$;
4. for every leaf node x of τ if $\vartheta(x) = (y, y')$, then y' is a leaf of t;
5. for every $x \in \mathsf{nodes}(\tau)$ if $\vartheta(x) = (y, y')$, then $\rho(y), \rho(y') \in \tau(x)$ (recall that $\tau(x) \in \mathsf{SCC}(\mathcal{A})$).

We first show that matchings are sufficient to get membership in the corresponding language, for any synopsis tree, primitive or not.

Claim 1. *If there exists a matching ϑ of τ against t and ρ, then $t \in [\![\tau]\!]_{\mathcal{A}}$.*

Proof. First, we need to introduce some notation: for a node x of t by $t{\downarrow_x}$ we denote the subtree of t rooted at x and for another node x' such that $x \preccurlyeq_t^{\mathsf{anc}} x'$ by $t{\downarrow_{x,x'}}$ we denote the context obtained by replacing in $t{\downarrow_x}$ the subtree rooted at the node (corresponding to) x' by the placeholder $\bullet$. Note that $t{\downarrow_{x,x}}$ is the empty context. Clearly, for any $x \in \mathsf{nodes}(\tau)$ if $\vartheta(x) = (y, y')$, then $\rho(y) \in \delta(\rho(y'), t{\downarrow_{y,y'}})$ (Condition 1) and then $t{\downarrow_{y,y'}} \in \mathscr{L}(\mathcal{A} \mid \tau(x))$ (Condition 5). This last observation allows us to prove with an simple induction over the height of $x \in \mathsf{nodes}(\tau)$ that $t{\downarrow_y} \in [\![\tau{\downarrow_x}]\!]_{\mathcal{A}}$ where $\vartheta(x) = (y, y')$. Indeed, we have that $\vartheta(x{\cdot}1) = (y'{\cdot}1, y'')$ and $\vartheta(x{\cdot}2) = (y'{\cdot}2, y''')$ whenever $\vartheta(x) = (y, y')$ (Condition 3). By inductive hypothesis, we know that $t{\downarrow_{y'\cdot 1}} \in [\![\tau{\downarrow_{x\cdot 1}}]\!]_{\mathcal{A}}$ and $t{\downarrow_{y'\cdot 2}} \in [\![\tau{\downarrow_{x\cdot 2}}]\!]_{\mathcal{A}}$. By the definition of $[\![\tau{\downarrow_x}]\!]_{\mathcal{A}}$, we conclude that $t{\downarrow_y} \in [\![\tau{\downarrow_x}]\!]_{\mathcal{A}}$ (the base case of this induction can be easily proved by using Condition 4). Combining this last fact with Condition 2, we prove that $t{\downarrow_\epsilon} \in [\![\tau{\downarrow_\epsilon}]\!]_{\mathcal{A}}$ and, thus, $t \in [\![\tau]\!]_{\mathcal{A}}$. □

We show next that if we match a synopsis tree τ against t and ρ, then τ will automatically satisfy the first condition to be a primitive synopsis tree (Definition 4.3.1).

Claim 2. *If there exists a matching ϑ of τ against t and ρ, then τ respects the transition function of $\mathcal{A}$, i.e., for all nodes x, $x \cdot 1$, and $x \cdot 2$ in τ, there exist some states $q \in \tau(x)$, $q_1 \in \tau(x \cdot 1)$, and $q_2 \in \tau(x \cdot 2)$ such that $q \in \delta(q_1, q_2)$.*

Proof. Take any $x, x \cdot 1, x \cdot 2 \in \mathsf{nodes}(\tau)$ and let $\vartheta(x) = (y, y')$, $\vartheta(x \cdot 1) = (y' \cdot 1, y'')$, and $\vartheta(x \cdot 2) = (y'{\cdot}2, y''')$. By Condition 5 of matching, we have $\rho(y') \in \tau(x)$, $\rho(y'{\cdot}1) \in \tau(x \cdot 1)$, and $\rho(y'{\cdot}2) \in \tau(x \cdot 2)$. Because ρ is a run of $\mathcal{A}$ over t, we get $\rho(y') \in \delta(\rho(y' \cdot 1), \rho(y' \cdot 2))$. □

Now, we have all the ingredients to derive a primitive synopsis tree τ from t such that $t \in [\![\tau]\!]_{\mathcal{A}}$. We start from an initial synopsis tree τ_0 and by iteratively applying a contraction operation over τ_0, we will derive τ. The initial synopsis tree τ_0 of t is obtained from t by relabeling every node n of t to the component $X \in \mathsf{SCC}(\mathcal{A})$ that contains the state of x in the run ρ (i.e. $\rho(x) \in X$). The matching of τ against t and ρ is simply $\vartheta_0(x) = (x, x)$ and it is easy to verify that it satisfies Conditions 1 through 5. By the claim above, the initial synopsis tree must respect the transition function of $\mathcal{A}$. It remains to transform the initial synopsis tree in such a way to preserve the existence of a matching against t and ρ, while achieving the second condition required for a primitive synopsis tree.

We present a simple contraction operation on a synopsis tree τ and a matching ϑ. It is applied to a node whose parent has the same label and it removes the parent node together with the subtree rooted at the other child. There are two variants of this operation depending on whether the node is the left or the right child of its parent node. We deal with the left variant only; the treatment of the right variant is analogous. The operation is presented in Figure 4.8 (τ_0 and τ_1 may be empty if the lower X is a leaf node). Note that the child node $x \cdot 1$ in τ becomes x in the new synopsis tree τ'

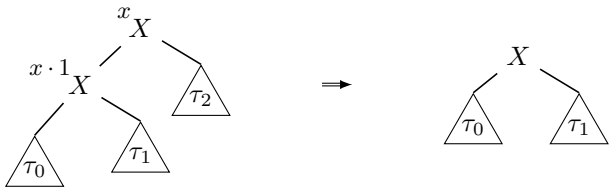


Figure 4.8: Contraction operation over synopsis trees.

(analogously, the nodes of τ_0 and τ_1 need to be offset). The new matching ϑ' is obtained as follows. If $\vartheta(x) = (y, y')$ and $\vartheta(x \cdot 1) = (y' \cdot 1, y'')$, then $\vartheta'(x) = (y, y'')$. On all other nodes ϑ' is identical to ϑ except that the nodes of τ_0 and τ_1 that need to be offset after the deletion and the nodes of τ_2 are dropped altogether. It is easy to verify that ϑ' satisfies Conditions 1 through 5.

We observe that every application of a contraction operation reduces the size of the synopsis tree and so this iterative process terminates and the resulting synopsis tree τ satisfies both conditions of Definition 4.3.1. □

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

In order to represent the target language and the possible edited trees, one needs a relaxed version of primitive synopsis trees, namely, *basic synopsis trees*. These synopsis trees are analogous to the chain of components over the reflexive and transitive closure of $\mathrm{dag}(\mathcal{T})$ (i.e. $\mathrm{dag}^*(\mathcal{T})$) used in Theorem 2.3.1 to characterize bounded repairability in the word case.

Definition 4.3.3. *A* basic synopsis tree *of an automaton $\mathcal{T}$ is a synopsis tree σ of $\mathcal{T}$ that respects the transition function of $\mathcal{T}$ (cf. Definition 4.3.1). We denote by $\mathsf{BST}(\mathcal{T})$ the set of all basic synopsis trees of $\mathcal{T}$.*

For example, the tree σ in Figure 4.6 is a basic synopsis tree that respects the transitions of the run of the automaton $\mathcal{T}$ depicted on the right-hand side of Figure 4.2.

Notice that a basic synopsis tree, in contrast with a primitive synopsis tree, could have internal nodes with the same label of one of its children. This implies that the set $\mathsf{BST}(\mathcal{T})$ of all basic synopsis trees of $\mathcal{T}$ is potentially infinite. Nevertheless, the set of basic synopsis trees of an automaton $\mathcal{T}$ can be represented by a deterministic binary bottom-up tree automaton of size polynomial in the size of $\mathcal{T}$.

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

Lemma 4.3.4. *For any stepwise tree automaton $\mathcal{A} = (\Sigma, Q, \delta, \delta_0, F)$, it holds that:*

$$\mathsf{cost}(\llbracket\sigma\rrbracket_{\mathcal{A}}, \mathscr{L}(\mathcal{A})) \;\leq\; (4 \cdot |\sigma| + 1) \cdot 2^{|Q|}$$

for any $\sigma \in \mathsf{BST}(\mathcal{A})$. In particular, $\mathsf{cost}(\llbracket\sigma\rrbracket_{\mathcal{A}}, \mathscr{L}(\mathcal{A})) < \infty$.

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

Proof of Lemma 4.3.4. We fix a stepwise automaton $\mathcal{A} = (\Sigma, Q, \delta, \delta_0, F)$ and a basic synopsis tree σ of $\mathcal{A}$. For any $q \in Q$ we denote by $\mathcal{A}^q$ the automaton $\mathcal{A}^q = (\Sigma, Q, \delta, \delta_0, \{q\})$ recognizing trees having a run with q at the root node. Also, let $N_{p,q} = \min\{|C| \mid q \in \delta(p, C) \wedge C \neq \bullet\}$ for any $p, q \in Q$ such that q is reachable from p in $G_{\mathcal{A}}$. We aggregate these values with $N = \max_{p,q \in Q} N_{p,q}$. Note that $N \geq 0$ by definition. Furthermore, notice that for any two $p, q \in Q$ such that q is reachable from p, there exists a context C of size bounded by N which satisfies $q \in \delta(p, C)$. For any pair $p, q \in Q$, we denote by $C_{p,q}$ this context whenever q is reachable from p.

We show next that $\mathsf{cost}(\llbracket\sigma\rrbracket_{\mathcal{A}}, \mathscr{L}(\mathcal{A}^q)) \leq 4 \cdot N \cdot |\sigma|$ by induction over the size of σ for any $q \in \sigma(\epsilon)$. Clearly, this will imply that $\mathsf{cost}(\llbracket\sigma\rrbracket_{\mathcal{A}}, \mathscr{L}(\mathcal{A})) \leq (4 \cdot |\sigma| + 1) \cdot 2^{|Q|}$ given that $N \leq 2^{|Q|}$.

For the base case $\sigma = X$, take any $t \in \llbracket X \rrbracket_{\mathcal{A}}$ and let $t = C \circ a$ for some $C \in \mathscr{L}(\mathcal{A} \mid X)$ and $a \in \Sigma$. Then there exists $p', q' \in X$ such that $q' \in \delta(p', C)$. Let $b \in \Sigma$ and $q_0 \in Q$ be a pair such that $q_0 \in \delta_0(b)$ and p' is reachable from q_0 (such a pair exists because we assume that $\mathcal{A}$ is trimmed). Define a tree:

$$t' \;=\; C_{q',q} \circ C \circ C_{q_0,p'} \circ b$$

which can be obtained from $t = C \circ a$ by (1) deleting the leaf node a, (2) inserting the leaf node b, (3) inserting the fragment $C_{q_0,p'}$, and (4) inserting the fragment $C_{q',q}$. Naturally, $t' \in \mathcal{L}(\mathcal{A}^q)$ and overall cost of transforming t to t' is bounded by $4 \cdot N$.

Now, for the inductive case take any $t \in \llbracket X(\sigma_1, \sigma_2) \rrbracket_{\mathcal{A}}$ and let $t = C \circ (t_1 @ t_2)$ for some $C \in \mathscr{L}(\mathcal{A} \mid X)$, $t_1 \in \llbracket \sigma_1 \rrbracket_{\mathcal{A}}$, and $t_2 \in \llbracket \sigma_2 \rrbracket_{\mathcal{A}}$. Also, let $p', q' \in X$ be two states such that $q' \in \delta(p', C)$. Since a basic synopsis tree respects the transition function of $\mathcal{A}$ (Definition 4.3.3), there exists a transition $p \in \delta(p_1, p_2)$ such that $p \in X$, $p_1 \in \sigma_1(\epsilon)$, and $p_2 \in \sigma_2(\epsilon)$. By inductive hypothesis, there exists $t_i' \in \mathscr{L}(\mathcal{A}^{p_i})$ such that $\mathsf{dist}(t_i, t_i') \leq 4 \cdot N \cdot |\sigma_i|$ for $i \in \{1, 2\}$. Define then:

$$t' \;=\; C_{q',q} \circ C \circ C_{p,p'} \circ (t_1' \;@\; t_2').$$

By following the previous construction, one can easily check that $t' \in \mathcal{L}(\mathcal{A}^q)$. Also, observe that t' can be obtained from t by (1) transforming t_1 into t_1' with cost less than $4 \cdot N \cdot |\sigma_1|$, (2) transforming t_2 into t_2' with cost less than $4 \cdot N \cdot |\sigma_2|$, (3) inserting $C_{p,p'}$, and (4) inserting $C_{q',q}$ where both insertions have cost less than N. Together the overall cost is bounded by $2 \cdot N + 4 \cdot N \cdot (|\sigma_1| + |\sigma_2|) \;<\; 4 \cdot N \cdot |\sigma|$. □

4.3.3 Coverings

The remaining part of the puzzle is to identify how to connect each primitive synopsis tree of the restriction automaton $\mathcal{R}$ to some basic synopsis tree of the target automaton $\mathcal{T}$. This is accomplished by the notion of *covering* between synopsis trees.

Definition 4.3.5. *Given two stepwise tree automata $\mathcal{R}$ and $\mathcal{T}$ and two synopsis trees τ of $\mathcal{R}$ and σ of $\mathcal{T}$, we say that τ is is* covered *by σ (denoted by $\tau \to \sigma$) iff there exists an* injective *mapping λ from non-trivial nodes of τ to non-trivial nodes of σ such that:*

1. *λ maps components in a compatible manner, i.e., $\mathscr{L}(\mathcal{R} \mid \tau(x)) \subseteq \mathscr{L}(\mathcal{T} \mid \sigma(\lambda(x)))$ for every non-trivial $x \in \mathsf{nodes}(\tau)$;*

2. *λ preserves the post-order of non-trivial nodes, i.e., $x \preccurlyeq_{\tau}^{\mathsf{post}} y$ iff $\lambda(x) \preccurlyeq_{\sigma}^{\mathsf{post}} \lambda(y)$ for any two non-trivial $x, y \in \mathsf{nodes}(\tau)$;*

3. *λ preserves the ancestorship of non-horizontal nodes, i.e., $x \preccurlyeq_{\tau}^{\mathsf{anc}} y$ iff $\lambda(x) \preccurlyeq_{\sigma}^{\mathsf{anc}} \lambda(y)$ for every non-horizontal $x \in \mathsf{nodes}(\tau)$ and every non-trivial node $y \in \mathsf{nodes}(\tau)$.*

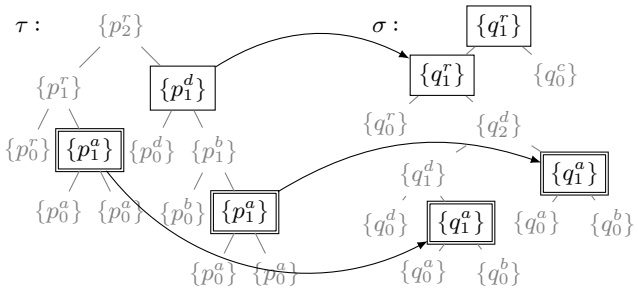


Figure 4.9: Covering of a primitive synopsis tree by a basic synopsis tree.

We call the mapping λ *a* covering *from* τ *to* σ*.*

Figure 4.9 presents a covering of a primitive synopsis tree τ of $\mathcal{R}$ by a basic synopsis tree σ of $\mathcal{T}$ where square boxes represent non-trivial nodes, and they have double borders when the component is non-horizontal.

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

Theorem 4.3.6. *Given two regular tree languages specified by stepwise automata* $\mathcal{R}$ *and* $\mathcal{T}$*,* $\mathscr{L}(\mathcal{R})$ *is bounded repairable into* $\mathscr{L}(\mathcal{T})$ *iff every primitive synopsis tree* τ *of* $\mathcal{R}$ *is covered by some basic synopsis tree* σ *of* $\mathcal{T}$*, that is,*

$$\mathsf{cost}(\mathcal{R},\mathcal{T}) < \infty \quad \textit{iff} \quad \forall\, \tau \in \mathsf{PST}(\mathcal{R}).\ \exists\, \sigma \in \mathsf{BST}(\mathcal{T}).\ \ \tau \to \sigma$$

Before turning to the proof of the above result, we explain the main ideas underlying the notion of covering. As a first remark, we observe that, given $\mathcal{R}$ and $\mathcal{T}$ such that $\mathsf{cost}(\mathscr{L}(\mathcal{R}),\mathscr{L}(\mathcal{T})) < \infty$, any reasonable strategy for repairing $\mathcal{R}$ into $\mathcal{T}$ with a uniformly bounded number cost will apply the edit operations only at the "junctions" of contexts realized by different components. Intuitively, this property holds because the non-trivial components of $\mathcal{R}$ can realize arbitrary large repetitions of the same context – these repetitions either do not need any editing at all, or they need an arbitrary large amount of editing. This gives an intuitive account for enforcing containments between languages recognized by components in the first condition of Definition 4.3.5 (not surprisingly, a similar condition was introduced in Chapter 2 for characterizing bounded repairability between regular languages of words).

As for the other two conditions, it is worth looking at the effect of an edit operation on the curry encoding of an unranked tree. Let us consider an unranked tree t with a distinguished node x that has to be deleted. There exists a unique way to represent the curry encoding of t together with the node x as an expression of the form $C \circ (t' \mathbin{@} (C' \circ a))$, where C' is a *horizontal* context that represents the forest of subtrees under x and a is the label of x. The result of the deletion of x from t is encoded by the curried tree $C \circ (C' \circ t')$ (see Figure 4.10 for an example). Note that this operation does not allow the deletion of the leftmost leaf node in the curried tree (this would

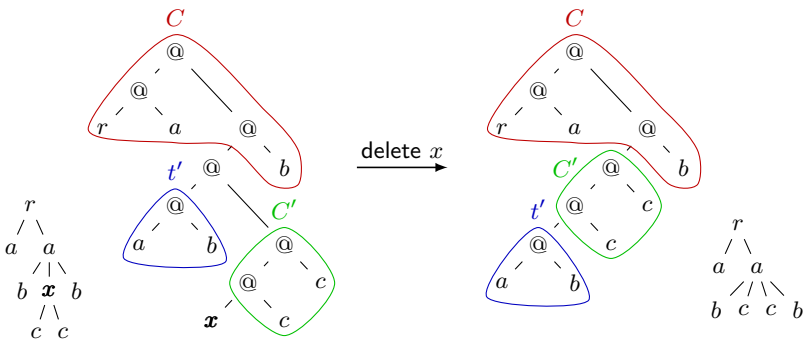


Figure 4.10: Deleting a node in the curry encoding.

correspond to deleting the root node in an unranked tree, an operation that is typically prohibited). The operation of inserting a new node y in an unranked tree t can be described in a similar way using curry encodings and horizontal contexts. Given an unranked tree t with curry encoding $C \circ (C' \circ t')$, where C' is a horizontal context, the curried tree $C \circ (t' @ (C' \circ a))$ represents the unranked tree that results from the insertion of a new a-labeled node y in t having as children the forest represented by C'. We now observe that the transformations on curried trees described above satisfy two crucial properties: (i) they preserve the post-order of the nodes and (ii) they preserve the ancestorship of non-horizontal contexts (e.g., C in Figure 4.10) with their descendants. These properties are precisely captured by the last two conditions of Definition 4.3.5.

As a corollary of Theorem 4.3.6, we obtain the following bound on the cost of a repair strategy.

Corollary 4.3.7. *Given two regular tree languages specified by stepwise automata $\mathcal{R}$ and $\mathcal{T}$, if $\mathscr{L}(\mathcal{R})$ is bounded repairable into $\mathscr{L}(\mathcal{T})$, then:*

$$\mathsf{cost}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) \leq O(|\mathsf{SCC}(\mathcal{T})| \cdot 2^{|Q|+|Q'|})$$

where Q and Q' are the set of states of $\mathcal{R}$ and $\mathcal{T}$, respectively.

4.4 Proof of the main characterization

The following subsections are devoted to proving the two directions of the characterization given in Theorem 4.3.6.

4.4.1 From covering to repair

We begin with the proof of the "if" direction of Theorem 4.3.6 by showing how to construct a repair strategy from $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with bounded cost, under the assumption that every primitive synopsis tree of $\mathcal{R}$ is covered by some basic synopsis tree of $\mathcal{T}$.

In addition to Lemmas 4.3.2 and 4.3.4, we need the following crucial property:

Lemma 4.4.1. *For any synopsis tree τ of $\mathcal{R}$ and any synopsis tree σ of $\mathcal{T}$, it holds that:*

$$\mathsf{cost}([\![\tau]\!]_{\mathcal{R}}, [\![\sigma]\!]_{\mathcal{T}}) \;\leq\; 4 \cdot |\tau| + 4 \cdot |\sigma|$$

whenever τ is covered by σ. In particular, $\mathsf{cost}([\![\tau]\!]_{\mathcal{R}}, [\![\sigma]\!]_{\mathcal{T}}) < \infty$ when τ is covered by σ.

Before giving the proof of Lemma 4.4.1, we explain how the "if" direction of Theorem 4.3.6 follows from it. By Lemma 4.3.2, the restriction language is contained in the union of the languages induced by the semantics of the primitive synopsis trees of $\mathcal{R}$. By hypothesis, each of these trees is covered by a basic synopsis tree of the target automaton $\mathcal{T}$. Thus by Lemma 4.4.1 their languages can be transformed into the languages induced by the semantics of some basic synopsis trees, using a bounded number of edits. By Lemma 4.3.4 the languages of the basic synopsis trees can be in turn repaired into the target language. The result now follows from the fact that there are only finitely many primitive synopsis trees and the fact that bounded repairability is a transitive relation and is preserved under finite unions.

We now turn to the proof of Lemma 4.4.1. For a technical reason, we need to slightly extend the definition of synopsis tree by allowing the use of a special node labeled ϵ that represents a dummy trivial component. The semantics is extended in the natural way by letting $\mathscr{L}(\mathcal{A} \mid \epsilon) = \{\bullet\}$ (for any stepwise automaton $\mathcal{A}$). Because all trivial components have the same semantics (i.e., they all recognize the language $\{\bullet\}$), we shall often identify a trivial component of an automaton with the dummy component ϵ.

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

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

Formally, we say that a synopsis tree θ of $\mathcal{R}$ *strongly covers* τ ($\tau \twoheadrightarrow \theta$) if there exists a bijective mapping λ from non-trivial nodes of τ to non-trivial nodes of θ such that $\tau(x) = \theta(\lambda(x))$. Furthermore, we say that θ is embedded in σ ($\theta \mapsto \sigma$) if $\mathsf{nodes}(\theta) = \mathsf{nodes}(\sigma)$ and $\theta \to \sigma$ with the identity mapping $\lambda : \mathsf{nodes}(\theta) \to \mathsf{nodes}(\sigma)$. Intuitively, θ is a middle-step in order to bounded repair $[\![\tau]\!]_{\mathcal{R}}$ into $[\![\sigma]\!]_{\mathcal{T}}$. Indeed, it is not difficult to show that such an interpolating synopsis tree θ exists.

Lemma 4.4.2. *If τ is covered by σ, then there is an synopsis tree θ of $\mathcal{R}$ such that τ is strongly covered by θ and θ is embedded in σ. That is:*

$$\tau \to \sigma \quad \textit{then} \quad \tau \twoheadrightarrow \theta \;\; \textit{and} \;\; \theta \mapsto \sigma$$

for some synopsis tree θ of $\mathcal{R}$.

Proof. Let λ be a covering function of τ by σ and recall that λ is injective. The synopsis tree θ has the same structure as σ (i.e. $\mathsf{nodes}(\theta) = \mathsf{nodes}(\sigma)$) and its labels are the labels of τ:

$$\theta(x) = \begin{cases} \tau(\lambda^{-1}(x)) & \text{if } x \in \mathit{Range}(\lambda), \\ \epsilon & \text{otherwise.} \end{cases}$$

Because λ is an injective function, its inverse $\lambda^{-1}(x)$ is unique for $x \in \mathit{Range}(\lambda)$, and therefore, the labels of θ are properly defined. It follows easily from the construction that σ embeds θ and θ strongly covers τ. □

The first advantage of considering an interpolating synopsis tree θ between τ and σ is that θ has the same labels than τ and they are connected through a bijective covering. In contrast, θ has the same structure than σ but the set of labels are different. Despite of this difference, the language $[\![\theta]\!]_{\mathcal{R}}$ is closer to $[\![\sigma]\!]_{\mathcal{T}}$ than to $[\![\tau]\!]_{\mathcal{T}}$. In fact, the following result shows that $[\![\theta]\!]_{\mathcal{R}} \subseteq [\![\sigma]\!]_{\mathcal{T}}$.

Lemma 4.4.3. *For a synopsis tree θ of $\mathcal{R}$ and a synopsis tree σ of $\mathcal{T}$, if θ is embedded in σ, then:*

$$[\![\theta]\!]_{\mathcal{R}} \subseteq [\![\sigma]\!]_{\mathcal{T}}.$$

Proof. This is proved by induction on the structure of θ (which has the same structure as σ). The base case is when θ consists a single node x. If $\theta(x) = X$, $\sigma(x) = Y$, and X is non-trivial, then from the covering of θ by σ, we have that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$ and consequently $[\![\theta]\!]_{\mathcal{R}} \subseteq [\![\sigma]\!]_{\mathcal{T}}$. The case when X is a trivial component follows from the fact that the language of every component, and Y in particular, contains the trivial context.

For the inductive step, let $\theta = X(\theta_1, \theta_2)$ and $\sigma = Y(\sigma_1, \sigma_2)$. Take any $t \in [\![\theta]\!]_{\mathcal{R}}$ and define $t = C \circ (t_1 @ t_2)$, where C is a context in $\mathscr{L}(\mathcal{R} \mid X)$ and $t_i \in [\![\theta_i]\!]_{\mathcal{R}}$ for $i \in \{1, 2\}$. By inductive hypothesis we have that $t_i \in [\![\sigma_i]\!]_{\mathcal{T}}$ for $i \in \{1, 2\}$. Moreover, we observe that from the covering relation we have that $C \in \mathscr{L}(\mathcal{T} \mid Y)$ (if X is trivial, then C is the necessarily the trivial context, which belongs to the language of any component). Consequently, $t \in [\![\sigma]\!]_{\mathcal{T}}$. □

Recall that $[\![\theta]\!]_{\mathcal{R}} \subseteq [\![\sigma]\!]_{\mathcal{T}}$ implies $\mathsf{cost}([\![\theta]\!]_{\mathcal{R}}, [\![\sigma]\!]_{\mathcal{T}}) < \infty$ and that the bounded repairability relation is transitive. We thus reduced the proof of Lemma 4.4.1 to the problem of showing that $\mathsf{cost}([\![\tau]\!]_{\mathcal{R}}, [\![\theta]\!]_{\mathcal{R}}) < \infty$. Towards this goal, we can take advantage of the notion of strong coverability and, in particular, of the fact that this is an *equivalence relation*, i.e. reflexive, symmetric, and transitive. Note that symmetry and transitivity follow from the fact that the mapping that witness strong coverability is a bijection between non-trivial nodes and preserves the labeling. For the sake of simplification, in the sequel we only work with synopsis trees of the automaton $\mathcal{R}$. This allows us to remove the automaton $\mathcal{R}$ from the notation that follows (e.g. we write $[\![\tau]\!]$ instead of $[\![\tau]\!]_{\mathcal{R}}$).

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

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

Definition 4.4.4. *A synopsis tree ζ is* in normal form *iff one of the following cases holds:*

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

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

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

Lemma 4.4.5. *If τ and ζ are two synopsis trees in normal form that strongly cover each other, then τ and ζ are isomorphic.*

Proof. Let τ and ζ be two synopsis trees in normal form and let λ be a bijection between the non-trivial nodes of τ and the non-trivial nodes of ζ that witnesses the strong coverability relationship between τ and ζ. For the sake of simplicity, in the following we shall identify the nodes of the synopsis trees τ and ζ with their labelings. Below we use an induction on the structure of τ to prove that τ and ζ are isomorphic synopsis trees.

For the base case, suppose that $\tau = \epsilon$. Since τ contains only trivial nodes and λ is surjective over non-trivial nodes, ζ contains only trivial nodes too. Since ζ is in normal form, it follows that $\zeta = \epsilon$.

For the inductive step, we distinguish two cases depending on whether τ is of the form $X(\tau', \epsilon)$, with X non-trivial horizontal component, or of the form $\epsilon(\tau', X(\tau'', \epsilon))$, with X non-horizontal component.

In the former case (i.e. $\tau = X(\tau', \epsilon)$), we recall that the mapping λ is a bijection between non-trivial nodes and preserves the post-order of non-trivial nodes. From this, one can easily show that λ must map the root X of τ to the root Y of σ. Moreover, since λ preserves the labels of the non-trivial nodes, we have that $Y = \lambda(X) = X$ and, in particular, Y is a non-trivial horizontal component. Since ζ is in normal form, it follows that its right sub-tree is ϵ and hence λ must map the non-trivial nodes of the left sub-tree τ' of τ to the non-trivial nodes of the left sub-tree ζ' of ζ. Finally, since both τ' and ζ' are synopsis trees in normal form, we obtain from the the inductive hypothesis that $\tau' = \zeta'$ and hence $\tau = X(\tau', \epsilon) = Y(\zeta', \epsilon) = \zeta$.

We now consider the latter case (i.e. $\tau = \epsilon(\tau', X(\tau'', \epsilon))$) where X is a non-horizontal component. For the sake of contradiction, suppose that the root Y of ζ is a non-trivial component. Since Y is the last non-trivial node in the post-order visit of ζ and λ preserves the post-order of non-trivial nodes, we would have that the pre-image $\lambda^{-1}(Y)$ in τ would be the last non-trivial node in the post-order visit of ζ, whence $\lambda^{-1}(Y) = X$. However, since X is a non-horizontal component, this would be against the hypothesis that ζ is in normal form (note that the root of any synopsis tree in normal form is always a horizontal component). Knowing that Y is a trivial node and ζ is in normal form, we obtain $Y = \epsilon$ and hence $\zeta = \epsilon(\zeta', Z(\zeta'', \epsilon))$, for some non-horizontal component Z and two synopsis trees ζ' and ζ'' in normal form. Towards a conclusion, observe that, in the post-order visit of τ, the non-trivial nodes of τ' are followed by the non-trivial nodes of τ'', and the latter are followed by the non-horizontal node X. As λ preserves the post-order of non-trivial nodes and the ancestorship relation with non-horizontal nodes, we conclude that $\lambda(X) = Z$ and that the synopsis trees τ' and ζ' (resp., τ'' and ζ'') strongly cover each other. Using the inductive hypothesis, we finally conclude that $\tau = \epsilon(\tau', X(\tau'', \epsilon)) = \epsilon(\zeta', Z(\zeta'', \epsilon)) = \zeta$. □

Thanks to Lemma 4.4.5, we can define *the normal form* ζ^* of a synopsis tree ζ as the unique synopsis tree that is *in normal form* and that *strongly covers* ζ, provided that this tree exists.

Our next goal is to prove that the normal form ζ^* of ζ indeed exists, and that can be attained by a finite sequence of generic editing operations on synopsis trees. These operations are called *promotion*, *demotion*, and *reduction*, and are presented in Figure 4.11. There, ϵ represents a trivial component, X represents an arbitrary component, $H_1, \ldots, H_k$ represent horizontal (possibly trivial) components, and $\alpha, \beta_1, \ldots, \beta_k$ represent arbitrary synopsis trees. Note that the figure describes the case where the promotion, demotion, and reduction operations are applied at the root of a synopsis tree – in general, these operations can be applied to any sub-tree of a synopsis tree. We write $\zeta \to_{\mathsf{op}}^* \zeta'$ whenever ζ' can be obtained from ζ by applying a finite sequence of promotion, demotion, and reduction operations. In order to give further intuition about these operations, we remark an

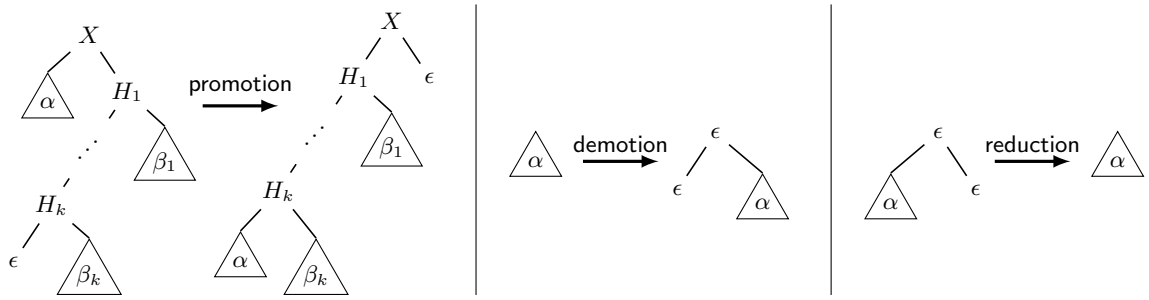


Figure 4.11: Macro-operations over synopsis trees.

analogy between the operations of promotion, depicted in Figure 4.11, and deletion, depicted in Figure 4.10 (a similar correspondence holds between the operations of demotion and insertion of a new root). In this case, the root X of the synopsis tree is acting as the the context C of the curried tree, the sub-tree α is acting as the curried sub-tree t', and the sub-tree rooted at H_1 is acting as the horizontal context C'.

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

Lemma 4.4.6. *For any synopsis tree ζ, we have $\zeta \to_{\mathsf{op}}^{*} \zeta^{*}$. Moreover, the number of operations needed to transform ζ into ζ^{*} is bounded by $2 \cdot |\zeta|$.*

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

Proof of Lemma 4.4.6. The proof is by structural induction on the synopsis tree ζ. For the base case, we can assume without loss of generality that all the leaves of the synopsis tree ζ are always ϵ-nodes. In fact, we can start our normalization procedure by hanging two ϵ-nodes to each ζ-leave without changing the equivalence class of ζ. This reduces all base cases to the case when the synopsis tree is a single ϵ-node. Then we have that $\zeta = \epsilon$, which means that ζ is already in normal form and hence trivially $\zeta \to_{\mathsf{op}}^{*} \zeta^{*}$.

For the inductive step, we assume that ζ is of the form $X(\alpha, \beta)$ and later we distinguish whether the component X at the root of ζ is horizontal, trivial, or non-horizontal.

First, we transform the left and right sub-trees of ζ to the corresponding normal forms α^* and β^* (this can be done since, by inductive hypothesis, $\alpha \to_{\mathsf{op}}^* \alpha^*$ and $\beta \to_{\mathsf{op}}^* \beta^*$). The resulting synopsis tree is of the form:

$$\zeta' \;=\; X(\alpha^*, \beta^*).$$

Moreover, since the right sub-tree β^* is in normal form, its leftmost branch consists of horizontal components only. We can thus write:

$$\zeta' \;=\; X\big(\alpha^*,\; H_1(\dots H_k(\epsilon, \beta_k), \dots, \beta_1)\big),$$

for some horizontal (possibly trivial) components $H_1, \dots, H_k$ and some synopsis trees $\beta_1, \dots, \beta_k$. We now can perform a promotion operation at the root of ζ' and obtain the synopsis tree:

$$\zeta'' \;=\; X\big(H_1(\dots H_k(\alpha^*, \beta_k), \dots, \beta_1), \epsilon\big).$$

Using a simple induction on $i = k, \dots, 1$ and the fact that the sub-trees $H_i(\dots H_k(\epsilon, \beta_k), \dots, \beta_1)$ of β^* are in normal form, one can easily verify that the sub-trees $H_i(\dots H_k(\alpha^*, \beta_k), \dots, \beta_1)$ of ζ'' are also in normal form. In particular, this shows that the left sub-tree of ζ'' is in normal form.

Now, we distinguish whether the component X is horizontal, trivial, or non-horizontal.

- If the component X at the root of ζ'' is horizontal and non-trivial, then ζ'' is already in normal form and hence $\zeta'' = \zeta^*$ (recall that the operations of promotion, demotion, and reduction preserve strong coverability).

- If the component X is trivial, then we select the left sub-tree from ζ'' via a reduction operation. This would result in the normalized tree $H_1(\dots H_k(\alpha^*, \beta_k), \dots, \beta_1)$ and we are done.

- Otherwise, if the component X is non-horizontal, we apply a demotion operation to ζ'' and obtain:

  $$\zeta''' \;=\; \epsilon(\epsilon, X\big(H_1(\dots H_k(\alpha^*, \beta_k), \dots, \beta_1), \epsilon\big).$$

  Note that the sub-trees ϵ and $H_1(\dots H_k(\alpha^*, \beta_k), \dots, \beta_1)$ are in normal form. Moreover, since X is non-horizontal, we know that ζ''' satisfies the definition of normal form and hence $\zeta''' = \zeta^*$ is in normal form.

Finally, define $\#_{\mathsf{op}}(\zeta)$ the minimum number of operations needed to transform ζ into ζ^*. From the previous inductive proof, one can obtain that $\#_{\mathsf{op}}(\zeta) \leq \#_{\mathsf{op}}(\alpha) + \#_{\mathsf{op}}(\beta) + 2$ where ζ is of the form $X(\alpha, \beta)$. By solving the previous recursive equation, we can show that $\#_{\mathsf{op}}(\zeta) \leq 2 \cdot |\zeta|$ and Lemma 4.4.6 is shown. □

For the last ingredient of the proof, we show that if $\zeta \to_{\mathsf{op}}^* \zeta'$, then the two languages $[\![\zeta]\!]$ and $[\![\zeta']\!]$ are repairable one into each other with a uniformly bounded number of edits. In other words, the

application of a promotion, demotion, or reduction operation to a synopsis tree ζ corresponds to a small amount of edits over $[\![\zeta]\!]$. The proof of this result is via a simple analysis of the transformations on languages of unranked trees that are induced by the operations of promotion, demotion, and reduction.

Lemma 4.4.7. *For any pair of synopsis trees ζ and ζ', if $\zeta \to_{\mathsf{op}}^{*} \zeta'$, then $\mathsf{cost}([\![\zeta]\!], [\![\zeta']\!]) < \infty$ and $\mathsf{cost}([\![\zeta']\!], [\![\zeta]\!]) < \infty$. In particular, $\mathsf{cost}([\![\zeta]\!], [\![\zeta^*]\!]) \leq 4 \cdot |\zeta|$.*

Proof. As a preliminary remark, we observe that it is sufficient to consider a single operation of promotion, demotion, or reduction that takes a synopsis tree ζ to another synopsis tree ζ' and prove that $\mathsf{cost}([\![\zeta]\!], [\![\zeta']\!]) < \infty$ (the case of longer sequences of operations can be dealt with by a simple inductive argument and the symmetric relationship $\mathsf{cost}([\![\zeta']\!], [\![\zeta]\!]) < \infty$ can be derived from the fact that standard editing operations can always be reverted). In the sequel, we assume that all synopsis trees are related to a single stepwise automaton $\mathcal{A}$.

Promotion. We begin by analyzing a promotion operation that takes a synopsis tree $\zeta = X\big(\alpha,\ H_1(\ldots H_k(\epsilon, \beta_k), \ldots, \beta_1)\big)$ to the synopsis tree $\zeta' = X\big(H_1(\ldots H_k(\alpha, \beta_k), \ldots, \beta_1), \epsilon\big)$. Let us consider a generic tree $t \in [\![\zeta]\!]$, which can be written as:

$$t \ = \ C \circ (s@(C_1 \circ (\ldots C_k \circ (a@s_k) \ldots @s_1))),$$

for some $C \in \mathscr{L}(\mathcal{A} \mid X)$, $s \in [\![\alpha]\!]$, $C_i \in \mathscr{L}(\mathcal{A} \mid H_i)$, $s_i \in [\![\beta_i]\!]$, and $a \in \Sigma$. Let us define the horizontal context:

$$\bar{C} \ = \ C_1 \circ (\ldots C_k \circ (\bullet@s_k) \ldots @s_1).$$

In this way the tree t can be written as $t = C \circ (s@(\bar{C} \circ a))$. After deleting a from t we obtain the tree $t' = C \circ (\bar{C} \circ s)$ and after we insert a b-node we get:

$$t'' \ = \ C \circ ((\bar{C} \circ s)@b) \ = \ C \circ ((C_1 \circ ((\ldots C_k \circ (s@s_k) \ldots)@s_1))@b),$$

which belongs to $[\![\zeta']\!]$.

Demotion. We now consider a demotion operation that takes a synopsis tree ζ to the synopsis tree $\zeta' = \epsilon(\epsilon, \zeta)$. We let $t \in [\![\zeta]\!]$ and we denote by x the leftmost leaf in t (note that this corresponds to the root of the unranked tree $\mathsf{ext}^{-1}(t)$). In this way we can write $t = C \circ a$ where a is the label of the leftmost leaf x of t and C is the horizontal context obtained from t by relabeling x with a placeholder. By applying an insertion operation to t, we obtain $t' = b \ @ \ (C \circ a)$ which clearly belongs to $[\![\zeta']\!]$.

Reduction. We finally consider a reduction operation that takes a synopsis tree $\zeta = \epsilon(\alpha, \epsilon)$ to the synopsis tree $\zeta' = \alpha$. We can write any generic tree $t \in [\![\zeta]\!]$ as $t = s \ @ \ a$ where $s \in [\![\alpha]\!]$ and $a \in \Sigma$. Here, it suffices one deletion operation to obtain the tree $t' \ = \ s$ which clearly belongs to $[\![\zeta']\!]$.

Towards the end of the proof, it is sufficient to observe that the repair strategies defined above can be lifted to trees under any given context C. More specific, if a tree $t \in [\![\zeta]\!]$ is transformed with editing operations into a tree $t' \in [\![\zeta']\!]$, then the tree $C \circ t$ can be edited into $C \circ t'$ using the same strategy. This observation is important because the operations of promotion, demotion, and reduction may be applied at arbitrary nodes of synopsis trees.

Finally, from the previous arguments one can check that $\mathsf{cost}([\![\zeta]\!], [\![\zeta']\!]) \leq 2$ whenever $\zeta \to_{\mathsf{op}} \zeta'$. This fact, together with Lemma 4.4.6, proves that $\mathsf{cost}([\![\zeta]\!], [\![\zeta^*]\!]) \leq 4 \cdot |\zeta|$. □

Proof of Lemma 4.4.1. We have all the ingredients to prove Lemma 4.4.1. Remember that by Lemmas 4.4.2 and 4.4.3 it only remains to show that $\mathsf{cost}([\![\tau]\!]_{\mathcal{R}}, [\![\theta]\!]_{\mathcal{R}}) < \infty$. The latter claim can be proved by combining Lemmas 4.4.5, 4.4.6, and 4.4.7. Indeed, we know from Lemma 4.4.6 that τ and θ can be converted into the normal forms τ^* and θ^*, respectively, by applying sequences of promotions, demotions, and reductions. Lemma 4.4.7 implies that $\mathsf{cost}([\![\tau]\!]_{\mathcal{R}}, [\![\tau^*]\!]_{\mathcal{R}}) \leq 4 \cdot |\tau|$ and, symmetrically, $\mathsf{cost}([\![\theta^*]\!]_{\mathcal{R}}, [\![\theta]\!]_{\mathcal{R}}) \leq 4 \cdot |\theta$. Since τ strongly covers θ, Lemma 4.4.5 implies $\tau^* = \theta^*$. We thus conclude that $\mathsf{cost}([\![\tau]\!]_{\mathcal{R}}, [\![\theta]\!]_{\mathcal{R}}) \leq 4 \cdot |\tau| + 4 \cdot |\theta|$ and, thus, $\mathsf{cost}([\![\tau]\!]_{\mathcal{R}}, [\![\sigma]\!]_{\mathcal{T}}) \leq 4 \cdot |\tau| + 4 \cdot |\sigma|$ from the transitivity of the bounded repairability relation. □

Towards the end of this subsection, we show the upper-bound given in Corollary 4.3.7. Assume then that $\mathcal{R}$ is bounded repairable into $\mathcal{T}$. Following the full repair process from $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ (Lemma 4.3.2, 4.4.1, and 4.3.4), we obtain the following inequalities between $\mathcal{R}$ and $\mathcal{T}$:

$$\begin{aligned}
\mathsf{cost}\big(\mathscr{L}(\mathcal{R}), \textstyle\bigcup_{\tau \in \mathsf{PST}(\mathcal{R})} [\![\tau]\!]_{\mathcal{R}}\big) &= 0 \\
\mathsf{cost}([\![\tau]\!]_{\mathcal{R}}, [\![\sigma]\!]_{\mathcal{T}}) &\leq 4 \cdot |\tau| + 4 \cdot |\sigma| \qquad \text{, whenever } \tau \to \sigma \\
\mathsf{cost}([\![\sigma]\!]_{\mathcal{T}}, \mathscr{L}(\mathcal{T})) &\leq (4 \cdot |\sigma| + 1) \cdot 2^{|Q|}
\end{aligned}$$

where Q' is the set of states of $\mathcal{T}$. Let f be a function from $\mathsf{PST}(\mathcal{R})$ to $\mathsf{BST}(\mathcal{T})$ such that $\tau \to f(\tau)$ for every $\tau \in \mathsf{PST}(\mathcal{R})$. Putting all the previous upper-bounds together, we can show that:

$$\mathsf{cost}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) \leq \max_{\tau \in \mathsf{PST}(\mathcal{R})} \left\{ 4 \cdot |\tau| + 4 \cdot |f(\tau)| + (4 \cdot |f(\tau)| + 1) \cdot 2^{|Q|} \right\}.$$

As was previously pointed out, any primitive synopsis tree τ of $\mathcal{R}$ is of bounded size. Furthermore, it is straightforward to show that the size of any primitive synopsis tree τ is bounded by $2^{|Q|}$ where Q is the set of states in $\mathcal{R}$. Indeed, any $\tau \in \mathsf{PST}(\mathcal{R})$ is a full binary tree and its height is bounded by $|\mathsf{SCC}(\mathcal{R})|$ which implies $|\tau| \leq 2^{|Q|}$. With respect to the size of $f(\tau)$, in Lemma 4.5.1 below we will show that there exists a basic synopsis tree σ' such that τ is covered by σ' and $|\sigma'| \leq 2 \cdot |\tau| \cdot |\mathsf{SCC}(\mathcal{T})|$ whenever τ is covered by some $\sigma \in \mathsf{BST}(\mathcal{R})$. Thus, without loss of generality we can assume that $f(\tau) \leq |\mathsf{SCC}(\mathcal{T})| \cdot 2^{|Q|+1}$. Putting everything together, one can easily prove Corollary 4.3.7, that is, $\mathsf{cost}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T})) \leq O(|\mathsf{SCC}(\mathcal{T})| \cdot 2^{|Q|+|Q'|})$.

4.4.2 From repair to covering

We now show the proof of the "only if" direction of Theorem 4.3.6. We fix for the rest of the section two stepwise automata $\mathcal{R} = (\Sigma, Q, \delta_0, \delta, F)$ and $\mathcal{T} = (\Delta, Q', \delta'_0, \delta', F')$ recognizing the restriction and the target languages, respectively. We assume that $\mathscr{L}(\mathcal{R})$ is bounded repairable into $\mathscr{L}(\mathcal{T})$ and we prove that every primitive synopsis tree of $\mathcal{R}$ is covered by some basic synopsis tree of $\mathcal{T}$.

The idea of the proof is to associate with any primitive synopsis tree τ of $\mathcal{R}$ a suitable tree $t_\tau \in \mathscr{L}(\mathcal{R})$, called a *witness tree* of τ, such that from any optimal repair of t_τ into $\mathscr{L}(\mathcal{T})$ one can extract a basic synopsis tree σ of $\mathcal{T}$ that covers τ. Intuitively, the witness tree t_τ is obtained from the primitive synopsis tree τ by replacing every non-trivial node x with a sufficiently large number of repetitions of a special context in $\mathscr{L}(\mathcal{R} \mid \tau(x))$, called a *fingerprint context*. The number of repetitions of each fingerprint context will depend on the worst-case repair cost $N = \mathsf{dist}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T}))$. Using the definition of the witness tree t_τ and the assumption that t_τ can be repaired into some tree $s_\tau \in \mathscr{L}(\mathcal{T})$ with at most N edits, one can then argue that s_τ contains at least one copy of the fingerprint context associated with each non-trivial node x of τ and, furthermore, the arrangements of the occurrences of these fingerprints inside t_τ and inside s_τ coincide both with respect to the post-order relation and with respect to the ancestorship of the non-horizontal components. One finally looks at some run of $\mathcal{T}$ that accepts the tree s_τ: this run, together with the structure of the fingerprints inside s_τ, induces a basic synopsis tree σ of $\mathcal{T}$ and a coverability relation from τ to σ. We recall that similar ideas were used in the proof of Theorem 2.3.1 and, thus, this direction can be considered as a generalization of the word case.

Below, we illustrate the argument in more detail. We divide up the proof into construction of the witness tree t_τ and building the cover from its repair.

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

Lemma 4.4.8. *For every $X \in \mathsf{SCC}(\mathcal{R})$, there exists a cyclic context $C_X \in \mathscr{L}(\mathcal{R} \mid X)$ such that, for every $Y \in \mathsf{SCC}(\mathcal{T})$,*

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

Proof. Let X be a component of $\mathcal{T}$ and let $Y_1, \ldots, Y_m \in \mathsf{SCC}(\mathcal{T})$ be all the components of $\mathcal{T}$. We construct the cyclic context C_X by exploiting an induction over the number m of components in $\mathcal{T}$. That is, we prove that for every $1 \leq i \leq m$, there is a cyclic context $C_i \in \mathscr{L}(\mathcal{R} \mid X)$ such that:

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

Clearly, the lemma follows from $(\star)$ when we let $C_X = C_m$.

For the base step $i = 1$, we distinguish two cases, depending on whether $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_1)$ or not. If $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_1)$, then we define C_1 as to be the trivial context, i.e., $C_1 = \bullet$. We thus have that $C_1 \in \mathscr{L}(\mathcal{T} \mid Y_1)$, C_1 is cyclic, and $(\star)$ holds. Otherwise, if $\mathscr{L}(\mathcal{R} \mid X) \notin \mathscr{L}(\mathcal{T} \mid Y_1)$, then we let C be any context in $\mathscr{L}(\mathcal{R} \mid X)$, but not in $\mathscr{L}(\mathcal{T} \mid Y_1)$ and $p, q \in Q$ be two states such that $q \in \delta(p, C)$. Since X is a strongly connected component, there exists a context C' such that $p \in \delta(q, C')$. We thus define $C_1 = C \circ C'$. By construction, we have that C_1 is cyclic. Furthermore, it is easy to see that $C_1 \notin \mathscr{L}(\mathcal{T} \mid Y_1)$ and hence $(\star)$ holds.

For the inductive step, let $i < m$ and suppose that there exists a context C_i that satisfies $(\star)$. We prove that $(\star)$ holds for $i + 1$ as well. Again, we distinguish between 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 $C_{i+1} = C_i$, in such a way that $(\star)$ holds trivially. Otherwise, let C be a context in $\mathscr{L}(\mathcal{R} \mid X)$ but not in $\mathscr{L}(\mathcal{T} \mid Y_{i+1})$. Since C_i is cyclic and $C \in \mathscr{L}(\mathcal{R} \mid X)$, that there exist some states $r, p, q \in X$ such that $r \in \delta(r, C_i)$ and $q \in \delta(p, C)$. Let C' and C'' be some contexts in $\mathscr{L}(\mathcal{R} \mid X)$ such that $r \in \delta(q, C')$ and $p \in \delta(r, C'')$ (clearly, such contexts C' and C'' exist since X is a strongly connected component and $r, p, q \in X$). Now, define $C_{i+1} = C_i \circ (C' \circ (C \circ C''))$. One can easily show that C_{i+1} is a cyclic context in $\mathscr{L}(\mathcal{R} \mid X)$ by following the transitions:

$$r \xrightarrow{C''} p \xrightarrow{C} q \xrightarrow{C'} r \xrightarrow{C_i} r.$$

It is also easy to show that C_{i+1} is not in $\mathscr{L}(\mathcal{T} \mid Y_{i+1})$. Indeed, if $C_{i+1} \in \mathscr{L}(\mathcal{T} \mid Y_{i+1})$, then there would exist states $p', q' \in Y_{i+1}$ such that $q' \in \delta'(p', C)$. This is a contradiction since $C \notin \mathscr{L}(\mathcal{T} \mid Y_{i+1})$. Finally, we know from the inductive hypothesis that for every $1 \leq j \leq i$, if $C_{i+1} \in \mathscr{L}(\mathcal{T} \mid Y_j)$, then $C_i \in \mathscr{L}(\mathcal{T} \mid Y_j)$ and $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_j)$. The converse implication follows in a similar way. We conclude that, for every $1 \leq j \leq i + 1$, $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_j)$ iff $C_{i+1} \in \mathscr{L}(\mathcal{T} \mid Y_j)$. This proves the inductive step for $(\star)$. □

Thanks to Lemma 4.4.8, in the following we associate with each component X a context C_X that satisfies the above lemma.

We now fix a primitive synopsis tree τ of $\mathcal{R}$ and we define the corresponding witness tree t_τ by using an induction on τ. In doing so, we will guarantee that there exists a run of $\mathcal{R}$ on the witness tree t_τ that reaches a state of the component at the root of τ, namely, $\delta(t_\tau) \cap \tau(\varepsilon) \neq \varnothing$. We omit the construction for the base case, where τ is a singleton, since it can be easily derived from what follows. Thus, we assume that X is the component at the root of τ and that τ_1 and τ_2 are the non-empty left and right sub-trees of τ. By induction hypothesis, we can denote by t_{τ_1} and t_{τ_2} the witness trees of τ_1 and τ_2, respectively. Moreover, we can fix a state q_1 (resp., q_2) in the non-empty set $\delta(t_{\tau_1}) \cap \tau_1(\varepsilon)$ (resp., $\delta(t_{\tau_2}) \cap \tau_2(\varepsilon)$). The construction of the witness tree t_τ is done in a bottom-up way using the three steps described below (the reader can also refer to Figure 4.12).

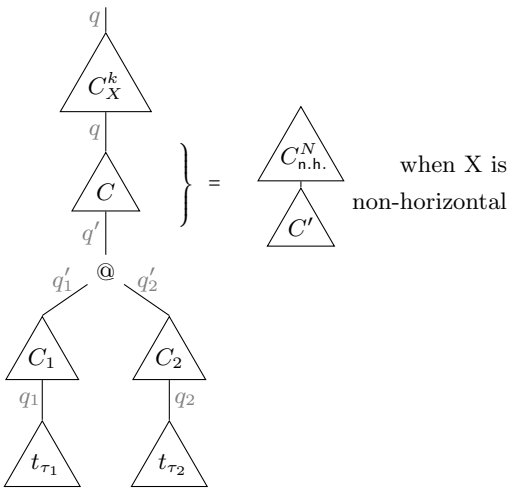


Figure 4.12: Construction of the witness tree.

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

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

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

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

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

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

The last step consists of plugging in a sufficiently long repetition of the fingerprint context C_X. For this, we define $k = m \cdot (2N + 1)$, where $N = \mathsf{dist}(\mathscr{L}(\mathcal{R}), \mathscr{L}(\mathcal{T}))$ and m is the number of SCCs

of $\mathcal{T}$. We then attach the k-fold repetition C_X^k of the fingerprint context C_X to the tree so far constructed, thus obtaining the witness tree:

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

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

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

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

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

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

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

Lemma 4.4.9. *One can find a mapping f from the non-trivial nodes x of the primitive synopsis tree τ to the nodes $f(x)$ of the repaired witness tree s_τ such that:*

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

- $x \preccurlyeq_{\tau}^{\mathsf{anc}} y$ *iff* $f(x) \preccurlyeq_{s_\tau}^{\mathsf{anc}} f(y)$*, for all pairs of non-trivial nodes* x, y *of* τ*, with* $\tau(x)$ *non-horizontal component.*

Proof. First we need the following technical result:

Claim 1. *Let* t *be a generic curried tree and let* t' *be the curried tree obtained from* t *after a deletion or an insertion of a single node. If* t *contains at least* i *non-overlapping occurrences of the same context* C*, then* t' *contains at least* $i-2$ *non-overlapping occurrences of* C*.*

Proof. Given that insertion is the inverse operation of deletion, it is sufficient to prove the claim for a deletion only. Let x be the node that is deleted from t. As mentioned in Subsection 4.3.3, there is a unique way to represent the operation of deletion using composition of trees and contexts. More precisely, t can be written as an expression of the form $C'' \circ (t''@(C' \circ a))$, where C' is a *horizontal* context that represents the forest of subtrees under x and a is the label of x. The result of the deletion of node x in $\mathsf{ext}^{-1}(t)$ gives the curried tree $t' = C'' \circ (C' \circ t'')$. Notice that the deletion operation performed on curry encodings removes exactly two nodes: the a-labeled leaf that corresponds to x and the @-labeled node y that connects t' to $C' \circ a$. All other nodes are preserved (but possibly re-arranged) by this transformation. In particular, this means that if (C, z) is an occurrence of the context C in t that does not overlap with the a-labeled node x nor with the @-labeled node y, then C occurs in either C'', C', or t''. This shows that C occurs in t'. Now, suppose that there are i non-overlapping occurrences of C in t. Since the deletion operation affects only two nodes, x and y, we have that, in the worst-case, all but two of these occurrences of C can be found in t' and t' contains at least $i-2$ occurrences of C. Finally, it is easy to see that the deletion operation preserve the property of occurrences of being non-overlapping. □

Basically, the above claim shows that if C^{2n+i} occurs in a tree t, then C^i occurs in any tree t' obtained from t after applying n edit operations.

Now we continue with the proof of the lemma. Consider a non-trivial node x of τ and let $X = \tau(x)$ be the associated component. By construction of t_τ, we know that the context C_X^k occurs in t_τ. Also recall that $k = m \cdot (2n+1)$. That is, we have $(2 \cdot N + 1)$-copies of the context C_X^M embedded inside t_τ. As s_τ is obtained from t_τ by applying at most n edit operations to it, we know from the above claim that the context C_X^m still occurs in s_τ. We denote by $f(x)$ some node of s_τ where C_X^m occurs. We have just proved the first part of the lemma.

For second part, let x and y be two non-trivial nodes of τ and let $X = \tau(x)$ and $Y = \tau(y)$. Further let $f(x)$ and $f(y)$ be the nodes of s_τ where the contexts C_X^m and C_Y^m occurs. Again, since deletion and insertion operations preserve the property of occurrences of being non-overlapping, we have that $(C_X^m, f(x))$ and $(C_Y^m, f(y))$ are non-overlapping occurrences in s_τ.

Now, suppose that $x \preccurlyeq_{\tau}^{\mathsf{post}} y$. Let x' and y' be the nodes in t_τ that carry the occurrences of C_X^k and C_Y^k, respectively. It is routine to check, by exploiting the recursive definition of t_τ, that

$x' \preccurlyeq^{\mathsf{post}}_{t_\tau} y'$. In addition, we know that edit operations preserves the post-order relationships between nodes. As previously discussed, at least one occurrence of C^m_X (resp., C^m_Y) inside C^k_X (resp., C^k_Y) is left untouched by the edit operations that transform t_τ into s_τ. This means that the corresponding occurrences $(C^m_X, f(x))$ and $(C^m_Y, f(y))$ in s_τ are in the same post-order relationship as x and y, namely, $f(x) \preccurlyeq^{\mathsf{post}}_{s_\tau} f(y)$. The converse implication follows from the fact that $\preccurlyeq^{\mathsf{post}}_{\tau}$ is a total order (hence it is sufficient to swap the roles of x and y above).

We finally check the last condition. Suppose that $X = \tau(x)$ is a non-horizontal component and that $x \preccurlyeq^{\mathsf{anc}}_{\tau} y$. Let x' and y' be the nodes in t_τ that carry the occurrences of C^k_X and C^k_Y, respectively. Thanks to the construction of t_τ, we have $x' \preccurlyeq^{\mathsf{anc}}_{\tau} y'$. Moreover, recall that during the construction of t_τ, we inserted n copies of the non-horizontal context C_v immediately below C^k_X (thus above C^k_Y). This implies that the path in t_τ that connects the node x' to its descendant y' visits at least n right edges. Similarly, in the corresponding unranked tree $\mathsf{ext}^{-1}(t_\tau)$, the two occurrences $\mathsf{ext}^{-1}(C^k_X)$ and $\mathsf{ext}^{-1}(C^k_Y)$ are far away from each other by at least n levels. Moreover, each deletion operation on $\mathsf{ext}^{-1}(t_\tau)$ can only bring the two occurrences closer by one level at a time, and insertion operations cannot do better. This means that after at most edit operations, the occurrence of $\mathsf{ext}^{-1}(C^m_X)$ in $\mathsf{ext}^{-1}(t_\tau)$ is still above the occurrence of $\mathsf{ext}^{-1}(C^m_Y)$. Formally, we have that $f(x) \preccurlyeq^{\mathsf{anc}}_{s_\tau} f(y)$. The converse implication follows by symmetric arguments. □

It only remains to show how to extract a basic synopsis tree σ that covers τ from an accepting run of $\mathcal{T}$ on s_τ. Let ρ be a run of $\mathcal{T}$ that accepts s_τ, let x be a non-trivial node of τ with label X, and let $f(x)$ be the corresponding node in s_τ induced by the mapping f, as defined in Lemma 4.4.9. By the previous arguments, the context C^m_X occurs at node $f(x)$ in s_τ. Let y be the position of the placeholder $\bullet$ in the fingerprint context C_X and observe that C_X occurs m times at nodes ε, y, $y \cdot y$, …, y^{m-1} of C^m_X. Let $y_{x,0} = f(x)$, $y_{x,1} = f(x) \cdot y$, …, $y_{x,m-1} = f(x) \cdot y^{m-1}$, and $y_{x,m} = f(x) \cdot y^m$. Clearly, $(C_X, y_{x,0})$, …, $(C_X, y_{x,m-1})$ are non-overlapping occurrences of the context C_X inside s_τ.

Consider the states that occur at the $m+1$ nodes $y_{x,0}, y_{x,1}, \ldots, y_{x,m}$ of the run ρ. By the Pigeonhole Principle, we know that there exist two nodes $y_{x,i}$ and $y_{x,j}$, with $0 \leq i < j \leq m$, and a component Y of $\mathcal{T}$ such that both states $\rho(y_{x,i})$ and $\rho(y_{x,j})$ belong to Y. In fact, by the definition of strongly connected component, we can assume $j = i+1$ and hence $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$. Notice that, until this point, we have not used the property of fingerprint contexts (Lemma 4.4.8). In particular, the fact that C_X is a fingerprint context of X and $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$ implies that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$.

What we have just showed is that it is possible to find a mapping from any non-trivial node x of τ to a node $y_x = y_{x,i}$ of s_τ such that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$, where $X = \tau(x)$ and Y is the component of the state $\rho(y_x)$. Thanks to Lemma 4.4.9, we can also claim that, for all non-trivial nodes x, x' in τ,

- $x \neq x'$ implies $y_x \neq y_{x'}$,

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

Now, we are ready to define the basic synopsis tree σ that covers τ. The domain of σ coincides with the domain of s_τ, i.e., $\mathsf{nodes}(\sigma) = \mathsf{nodes}(s_\tau)$. The labeling $\sigma(x)$ of a node x of σ is given by the component X that contains the corresponding state $\rho(x)$, where ρ is a run of $\mathcal{T}$ that accepts s_τ. Notice that σ trivially satisfies the properties of a basic synopsis tree, as its labellings respects the transitions of $\mathcal{T}$ given by the run ρ. In a similar way, we can define the mapping λ that witnesses the coverability of τ by σ: for this we simply let λ map any non-trivial node x of τ to the node y_x of σ. The fact that λ satisfies Definition 4.3.5 follows easily from the properties described by the three items above (for instance, the fact that λ is injective follows from the first item). This completes the proof that τ is covered by a basic synopsis tree of $\mathcal{T}$.

4.5 Complexity analysis

In this section we will exploit the characterization given in Theorem 4.3.6 to obtain tight complexity bounds for the bounded repair problem. We will first consider the general case where the regular tree languages are specified by non-deterministic stepwise tree automata and show a straightforward Π_2^P-upperbound for the bounded repair problem than can be obtained from Theorem 4.3.6. In order to find the precise complexity, we have to exploit Theorem 4.3.6 further. Actually, we show an coNEXPTIME-algorithm for deciding bounded repairability. This algorithm follows from a polynomial time reduction of the coverability problem between synopsis trees to the containment of non-recursive context-free grammars. For this reduction the characterization given in Theorem 4.3.6 will be crucial. Later on, we will concentrate on less succinct and less expressive representations such as *non-recursive deterministic DTDs* (see Section 4.1). We will show that even for these specifications the bounded repair problem is coNEXPTIME-hard. Thus, we will conclude that for all regular specifications consider in this chapter the bounded repair problem is coNEXPTIME-complete.

We start by analysing the size of the primitive and basic synopsis trees in $\mathcal{R}$ and $\mathcal{T}$, respectively. It follows easily from Definition 4.3.1 that any primitive synopsis tree of $\mathcal{R}$ has height less than the number of components of $\mathcal{R}$, and hence size at most $2^{|\mathsf{SCC}(\mathcal{R})|}$. As for the minimal size of the basic synopsis trees of $\mathcal{T}$ that cover a given primitive synopsis tree of $\mathcal{R}$, we can prove the following:

Lemma 4.5.1. *Given two stepwise automata $\mathcal{R}$ and $\mathcal{T}$ and two synopsis trees $\tau \in \mathsf{PST}(\mathcal{R})$ and $\sigma \in \mathsf{BST}(\mathcal{T})$, if σ covers τ, then there is $\sigma' \in \mathsf{BST}(\mathcal{T})$ that covers τ and has size at most $2{\cdot}|\tau|{\cdot}|\mathsf{SCC}(\mathcal{T})|$, where $|\tau|$ is the number of nodes of τ and $|\mathsf{SCC}(\mathcal{T})|$ is the number of components of $\mathcal{T}$.*

Proof. In this proof we abstract any primitive synopsis tree τ of $\mathcal{R}$ by its set $\mathcal{X}$ of non-trivial nodes. By a slight abuse of notation, we will use $\mathscr{L}(\mathcal{R} \mid x)$ to denote the language of contexts realized by

the component associated with the node x in the underlying primitive synopsis tree of $\mathcal{R}$. Given a basic synopsis tree σ of $\mathcal{T}$, we say that σ *weakly covers* $\mathcal{X}$ via an injective function $\lambda : \mathcal{X} \to \mathsf{nodes}(\sigma)$ if $\mathscr{L}(\mathcal{R} \mid x) \subseteq \mathscr{L}(\mathcal{T} \mid \sigma(\lambda(x)))$ holds for all $x \in \mathcal{X}$. We also say that a component $Y \in \mathsf{SCC}(\mathcal{T})$ has *height* h if the maximum number of distinct components visited by a path in the transition graph of $\mathcal{T}$ is h. Note that the height of a component $Y \in \mathsf{SCC}(\mathcal{T})$ never exceeds the number n of components of $\mathcal{T}$. We now prove the following property, which immediately implies the claim of the lemma:

Claim 1. *Given a set $\mathcal{X}$ of non-trivial nodes of a primitive synopsis tree of $\mathcal{R}$ and a basic synopsis tree σ of $\mathcal{T}$ that weakly covers $\mathcal{X}$, there is a basic synopsis tree σ' of $\mathcal{T}$ that weakly covers $\mathcal{X}$ and such that*

- *the root of σ' is labelled with the same component of the root of σ, i.e., $\sigma'(\varepsilon) = \sigma(\varepsilon)$,*
- *the number of nodes in σ' is at most $2h \cdot |\mathcal{X}| - 1$, where h is the height of the component at the root of σ.*

We prove the above claim by exploiting an induction on the size of the set $\mathcal{X}$ and the height h of the component at the root of σ. For the base case, let $\mathcal{X}$ be a singleton and $h = 1$. Further, suppose that σ is a basic synopsis tree that covers the set $\mathcal{X} = \{x\}$ via a function $\lambda : \mathcal{X} \to \mathsf{nodes}(\sigma)$. Let π be the access path of the node $y = \lambda(x)$ in σ. We define σ' as the tree obtained from σ by selecting only the nodes along the path π and their immediate descendants. Clearly, σ' is a full binary tree of size $2|\pi| - 1 \leq 2h - 1 = 2h \cdot |\mathcal{X}| - 1$.

As for the inductive case, we fix a non-singleton set $\mathcal{X}$ of non-trivial nodes and a basic synopsis tree σ of $\mathcal{T}$ that weakly covers $\mathcal{X}$ via a function $\lambda : \mathcal{X} \to \mathsf{nodes}(\sigma)$, and we assume that the above claims holds on every proper subset $\mathcal{X}'$ of $\mathcal{X}$ and on every sub-tree σ' of σ. Let h be the height of the component at the root ε of σ, let σ_1 and σ_2 be the left and right sub-trees rooted at ε, let $\mathcal{X}_1 = \lambda^{-1}(\mathsf{nodes}(\sigma_1))$ and $\mathcal{X}_2 = \lambda^{-1}(\mathsf{nodes}(\sigma_2))$, and finally let h_1 and h_2 be the heights of the components at the roots of σ_1 and σ_2, respectively. Note that $h_1, h_2 \leq h$. We distinguish between the following cases:

1. If the root of σ is in the range of λ (i.e., $\varepsilon \in \lambda(\mathcal{X})$) or both $\mathcal{X}_1$ and $\mathcal{X}_2$ are non-empty, then $|\mathcal{X}_1|, |\mathcal{X}_2| < |\mathcal{X}|$ follows. We can thus apply the inductive hypothesis on $\mathcal{X}_i$ and σ_i, for both $i = 1$ and $i = 2$. From that we obtain that there exist two basic synopsis trees σ'_1 and σ'_2 of $\mathcal{T}$ such that (i) the component at the root of σ'_1 (resp., σ'_2) coincides with the component at the root of σ_1 (resp., σ_2) and (ii) the number of nodes in σ'_1 (resp., σ'_2) is at most $2h_1 \cdot |\mathcal{X}_1| - 1$ (resp., $2h_2 \cdot |\mathcal{X}_2| - 1$). By reconnecting the root of σ to the trees σ'_1 and σ'_2 one obtains a new basic synopsis tree σ' of $\mathcal{T}$ that weakly covers $\mathcal{X}$ and such that (i) the component at the root of σ' coincides with the component at the root of σ and (ii) the number of nodes in σ' is

$$|\sigma'| \;=\; 1 + |\sigma'_1| + |\sigma'_2| \;\leq\; 1 + 2h_1 \cdot |\mathcal{X}_1| - 1 + 2h_2 \cdot |\mathcal{X}_2| - 1 \;\leq\; 2h \cdot (|\mathcal{X}_1| + |\mathcal{X}_2|) - 1 \;\leq\; 2h \cdot |\mathcal{X}| - 1.$$

This proves the inductive step in the case where the root of σ is in the range of λ and in the case where both $\mathcal{X}_1$ and $\mathcal{X}_2$ are non-empty.

2. If the root of σ is not in the range of λ and either $\mathcal{X}_1$ or $\mathcal{X}_2$ is empty, then we have $\mathcal{X} = \mathcal{X}_1$ or $\mathcal{X} = \mathcal{X}_2$. Assume, without loss of generality, that $\mathcal{X} = \mathcal{X}_1$ (a symmetric argument can be applied to the case $\mathcal{X} = \mathcal{X}_2$) and consider the left sub-tree σ_1 of σ, which weakly covers $\mathcal{X}$. Without loss of generality, we can assume that the component at the root of σ_1 is different from the component at the root of σ. Otherwise, we could simply reduce the tree σ to σ_1 and apply inductively the above arguments. As σ and σ_1 have different components at their roots, we know that $h_1 < h$ and hence we can apply the inductive hypothesis to $\mathcal{X}$ and σ_1. This results in a basic synopsis tree σ_1' that weakly covers $\mathcal{X}$, has the same component at the root as σ_1, and has size at most $2h_1 \cdot |\mathcal{X}| - 1$. By reconnecting the root of σ to the tree σ_1', and the root Using the root of σ, the tree σ_1', and the root of σ_2 (without its descendants), one can obtain a new basic synopsis tree σ' of $\mathcal{T}$ that weakly covers $\mathcal{X}$ and and has size

$$|\sigma'| \;=\; 1 + |\sigma_1'| + 1 \;\leq\; 1 + 2h_1 \cdot |\mathcal{X}| - 1 + 1 \;\leq\; 1 + 2(h-1) \cdot |\mathcal{X}| \;\leq\; 2h \cdot |\mathcal{X}| - 1.$$

This completes the proof of the claim. □

The above lemma, together with Theorem 4.3.6, gives a Π_2^{EXP} algorithm that receives two non-deterministic stepwise automata $\mathcal{R}$ and $\mathcal{T}$ and decides whether $\mathscr{L}(\mathcal{R})$ is bounded repairable into $\mathscr{L}(\mathcal{T})$. The algorithm universally guesses a primitive synopsis tree τ of $\mathcal{R}$ of size at most $2^{|\mathsf{SCC}(\mathcal{R})|}$, then it existentially guesses a basic synopsis tree σ of $\mathcal{T}$ of size at most $2^{|\mathsf{SCC}(\mathcal{R})|+1} \cdot |\mathsf{SCC}(\mathcal{T})|$ (thanks to Lemma 4.5.1) and a function λ from non-trivial nodes of τ to non-trivial nodes of σ, and finally it checks that λ is a covering of τ by σ (this amounts at deciding language inclusion, which can be done in exponential time [Sei90]).

To find the exact complexity of the bounded repair problem between regular tree languages we have to exploit Theorem 4.3.6 further by serializing synopsis trees and reducing the coverability problem of synopsis trees to the containment of non-recursive context-free grammars. Specifically, we represent the set of all primitive synopsis trees of $\mathcal{R}$ by a non-recursive context-free grammar $G_{\mathcal{R}}$ and we encode all possible ways of covering primitive synopsis trees with basic synopsis trees of $\mathcal{T}$ by a grammar $G_{\mathcal{R},\mathcal{T}}$. Then we show that $\mathcal{R}$ is bounded repairable into $\mathcal{T}$ iff $\mathscr{L}(G_{\mathcal{R}}) \subseteq \mathscr{L}(G_{\mathcal{R},\mathcal{T}})$, i.e. the language represented by $G_{\mathcal{R}}$ is contained in the language represented by $G_{\mathcal{R},\mathcal{T}}$. This implies that we can decide bounded repairability between $\mathcal{R}$ and $\mathcal{T}$ by checking the containment of two context-free grammars. This last problem can be decided in coNEXPTIME which implies a coNEXPTIME upper-bound for the bounded repair problem. Actually, we show later that bounded repairability of regular tree languages is coNEXPTIME-complete. In the following, we explain the reduction of the bounded repair problem into context-free grammars.

A context-free grammar (or just grammar) is a tuple $G = (\Sigma, V, P, I)$ where Σ is a finite alphabet (called terminals), V is a finite set of variables (called non-terminals) such that $V \cap \Sigma = \varnothing$, P is a finite set of productions, and $I \subseteq V$ is the initial set of variable. Each production in P is of the form $A \to \beta$ where $A \in V$ and $\beta \in (V \cup \Sigma)^*$. We say that $\alpha A \gamma \to_G \alpha\beta\gamma$ with $A \in V$ and $\alpha, \beta, \gamma \in (V \cup \Sigma)^*$ is a derivation of G whenever there exists a production $(A \to \beta) \in P$. Furthermore, we define the relation $\to_G^+$ to be the transitive closure of $\to_G$. The language defined by G is given by $\mathscr{L}(G) = \{w \in \Sigma^* \mid A \to_G^+ w \text{ and } A \in I\}$. We say that a context-free grammar G is *non-recursive* if $A \not\to_G^+ \alpha A \beta$ for every $A \in V$ and $\alpha, \beta \in (V \cup \Sigma)^*$. In particular, non-recursive context-free grammars define only finite languages.

In the sequel, it will be useful to represent $\to_G^+$-derivations by *derivation trees*. A derivation tree d of G is a finite ranked tree where internal nodes are labelled with non-terminals symbols (i.e. V), leafs nodes are labelled with ϵ or terminal symbols (i.e. Σ), and for every internal node $x \in \mathsf{nodes}(d)$ there exists a production $(A \to \beta) \in P$ such that $d(x) = A$ and $d(x \cdot 1) \cdot d(x \cdot 2) \cdot \cdots \cdot d(x \cdot k) = \beta$ where k is the number of children of x in d. We say that d is a derivation tree of G over w if d is a derivation tree of G, $d(\epsilon) \in I$, and the yield of d, i.e. the sequence of leaves taken from left to right, is equal to w. Note that $w \in \mathscr{L}(G)$ iff there exists a derivation tree of G over w.

Now, fix two stepwise automata $\mathcal{R} = (\Sigma, Q, \delta, \delta_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', \delta'_0, F')$. In the following, we define two context free grammars $G_\mathcal{R}$ and $G_{\mathcal{R},\mathcal{T}}$ such that $\mathscr{L}(G_\mathcal{R})$ represents the serialization of all primitive synopsis trees of $\mathcal{R}$ and $\mathscr{L}(G_{\mathcal{R},\mathcal{T}})$ encodes all possible ways of covering primitive synopsis trees of $\mathcal{R}$ with basic synopsis trees of $\mathcal{T}$. To serialize primitive synopsis trees of $\mathcal{R}$ we use the XML-symbols $\langle\, X \,\rangle$, $\langle / \, X \,\rangle$, $\langle\, X \,/\rangle$ for each $X \in \mathsf{SCC}(\mathcal{R})$. In XML, $\langle\, X \,\rangle$ and $\langle / \, X \,\rangle$ are called open-tag and close-tag, respectively, and $\langle\, X \,/\rangle$ is called a leaf-tag (see [BPSM+08] for a detailed specification of the standard). We define the set of all tags over $\mathsf{SCC}(\mathcal{R})$ by $\mathsf{tags}(\mathcal{R}) = \{\langle\, X \,\rangle, \langle / \, X \,\rangle, \langle\, X \,/\rangle \mid X \in \mathsf{SCC}(\mathcal{R})\}$.

The first step of the reduction is to represent a synopsis tree τ with a string over $\mathsf{tags}(\mathcal{R})$. This representation uses tags $\langle\, X \,\rangle$ and $\langle / \, X \,\rangle$ to encode non-horizontal components and leaf-tags $\langle\, X \,/\rangle$ to encode horizontal components of τ. We use nesting pairs $\langle\, X \,\rangle$ and $\langle / \, X \,\rangle$ to simulate the ancestor relationship of non-horizontal components in τ and the sequential order of a string to naturally simulate the post-order of nodes in τ. Formally, for any synopsis tree τ of $\mathcal{R}$ we define the *serialization* of τ (denoted by $\hat{\tau}$) inductively as follows. Suppose that $\tau = X(\tau_1, \tau_2)$ where τ_1, τ_2 are synopsis trees of $\mathcal{R}$ and $X \in \mathsf{SCC}(\mathcal{R})$, then:

- $\hat{\tau} = \hat{\tau_1} \cdot \hat{\tau_2} \cdot \langle\, X \,/\rangle$ whenever X is a non-trivial and horizontal component of $\mathcal{R}$,
- $\hat{\tau} = \langle\, X \,\rangle \cdot \hat{\tau_1} \cdot \hat{\tau_2} \cdot \langle / \, X \,\rangle$ whenever X is a non-horizontal component of $\mathcal{R}$, and
- $\hat{\tau} = \hat{\tau_1} \cdot \hat{\tau_2}$ whenever X is a trivial component.

The base case (i.e. $\tau = X$) can be easily derived from the previous cases. Notice that by serializing τ we are loosing some structural information of the tree. That is, there exist synopsis trees τ_1 and τ_2 such that $\hat{\tau_1} = \hat{\tau_2}$ but $\tau_1 \neq \tau_2$. Nevertheless, $\hat{\tau}$ preserves the post-order between non-trivial nodes and the ancestor-ship relation between non-horizontal and non-trivial nodes in τ which is all the structure that we need to check coverability between synopsis trees.

The next step is to represent all serialized primitive synopsis trees of $\mathcal{R}$ with a non-recursive context-free grammar over $\mathsf{tags}(\mathcal{R})$. We define the context-free grammar:

$$G_{\mathcal{R}} \;=\; (\,\mathsf{tags}(\mathcal{R}),\, \mathsf{SCC}(\mathcal{R}),\, P_{\mathcal{R}},\, \mathsf{SCC}(\mathcal{R})\,)$$

where $\mathsf{tags}(\mathcal{R})$ is the set of terminal symbols, $\mathsf{SCC}(\mathcal{R})$ is the set of variables and initial variables, and $P_{\mathcal{R}}$ are the set of productions such that, for every $q \in \delta(q_1, q_2)$ and $[q_1] \neq [q] \neq [q_2]$, it holds that (we denote $X = [q]$ for the sake of readability):

- $X \to [q_1] \cdot [q_2] \cdot \langle\, X \,/\rangle$ and $X \to \langle\, X \,/\rangle$ are productions in $P_{\mathcal{R}}$ whenever X is a non-trivial and horizontal component of $\mathcal{R}$,
- $X \to \langle\, X \,\rangle \cdot [q_1] \cdot [q_2] \cdot \langle /\, X \,\rangle$ and $X \to \langle\, X \,\rangle \cdot \langle /\, X \,\rangle$ are productions in $P_{\mathcal{R}}$ whenever X is a non-horizontal component of $\mathcal{R}$, and
- $X \to [q_1] \cdot [q_2]$ and $X \to \epsilon$ are productions in $P_{\mathcal{R}}$ whenever X is a trivial component of $\mathcal{R}$.

It is straightforward to show that $\hat{\tau} \in \mathscr{L}(G_{\mathcal{R}})$ for every primitive synopsis tree τ of $\mathcal{R}$. Moreover, if $w \in \mathscr{L}(G_{\mathcal{R}})$, then there exists $\tau \in \mathsf{PST}(\mathcal{R})$ such that $\hat{\tau} = w$. For this last fact, notice that for every derivation tree d of $G_{\mathcal{R}}$ over $w \in \mathscr{L}(G_{\mathcal{R}})$ one can easily obtain a primitive synopsis tree τ_d of $\mathcal{R}$ such that $\hat{\tau}_d = w$. Indeed, construct τ_d from d by pruning all leaves of d, i.e. nodes labelled with elements in $\mathsf{tags}(\mathcal{R})$ or with ϵ. Then for all nodes $x \in \mathsf{nodes}(\tau_d)$ it holds that $\tau_d(x) \in \mathsf{SCC}(\mathcal{R})$. Furthermore, all internal nodes $x \in \mathsf{nodes}(\tau_d)$ satisfy $\tau_d(x \cdot 1) \neq \tau_d(x) \neq \tau_d(x \cdot 2)$ and $q \in \delta(q_1, q_2)$ for some states $q \in \tau_d(x)$, $q_1 \in \tau_d(x \cdot 1)$, and $q_2 \in \tau_d(x \cdot 2)$. That is, τ_d is a primitive synopsis tree of $\mathcal{R}$ and, thus, $\hat{\tau}_d = w$. In the following, we denote by τ_d the primitive synopsis tree obtained from a derivation tree d of $G_{\mathcal{R}}$.

We can also define a context-free grammar $G_{\mathcal{R},\mathcal{T}}$ that represents all serialized primitive synopsis tree of $\mathcal{R}$ that are covered by some basic synopsis tree of $\mathcal{T}$. Let

$$G_{\mathcal{R},\mathcal{T}} \;=\; (\,\mathsf{tags}(\mathcal{R}),\, \mathsf{SCC}(\mathcal{T}),\, P_{\mathcal{R},\mathcal{T}},\, \mathsf{SCC}(\mathcal{T})\,)$$

where $\mathsf{tags}(\mathcal{R})$ is the set of terminal symbols, $\mathsf{SCC}(\mathcal{T})$ is the set of variables and initial variables, and for all $q' \in \delta'(q'_1, q'_2)$ and $X \in \mathsf{SCC}(\mathcal{R})$ (we denote $Y = [q']$ for the sake of readability):

- $Y \to [q'_1] \cdot [q'_2] \cdot \langle\, X \,/\rangle$ and $Y \to \langle\, X \,/\rangle$ are productions in $P_{\mathcal{R},\mathcal{T}}$ whenever X is an horizontal component of $\mathcal{R}$ and $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$,

- $Y \to \langle\, X \,\rangle \cdot [q_1'] \cdot [q_2'] \cdot \langle /\, X \,\rangle$ and $Y \to \langle\, X \,\rangle \cdot \langle /\, X \,\rangle$ are productions in $P_{\mathcal{R},\mathcal{T}}$ whenever X is a non-horizontal component of $\mathcal{R}$ and $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$, and
- $Y \to [q_1'] \cdot [q_2']$ and $Y \to \epsilon$ are always productions in $P_{\mathcal{R},\mathcal{T}}$.

Despite the similarities between both grammars, $G_{\mathcal{R},\mathcal{T}}$ uses the structure of $\mathcal{T}$ to generate all potential primitive synopsis trees of $\mathcal{R}$ that are covered by some basic synopsis tree of $\mathcal{T}$. Similar to $G_{\mathcal{R}}$, we can obtain a basic synopsis tree $\sigma_d \in \mathsf{BST}(\mathcal{T})$ from any derivation tree d of $G_{\mathcal{R},\mathcal{T}}$ by pruning all its leaves. Thus, we can denote by σ_d the basic synopsis tree obtained from a derivation tree d of $G_{\mathcal{R},\mathcal{T}}$.

Intuitively, each word $w \in \mathscr{L}(G_{\mathcal{R},\mathcal{T}})$ is representing a possible basic synopsis tree of $\mathcal{T}$ that can cover a primitive synopsis tree τ of $\mathcal{R}$ such that $\hat{\tau} = w$. The following lemma makes this intuition precise.

Lemma 4.5.2. *Every primitive synopsis tree of $\mathcal{R}$ is covered by a basic synopsis tree of $\mathcal{T}$ iff*

$$\mathscr{L}(G_{\mathcal{R}}) \;\subseteq\; \mathscr{L}(G_{\mathcal{R},\mathcal{T}}).$$

Proof. Assume first that every primitive synopsis tree of $\mathcal{R}$ is covered by a basic synopsis tree of $\mathcal{T}$ and consider any $w \in \mathscr{L}(G_{\mathcal{R}})$. We know that there exists a derivation tree d of $G_{\mathcal{R}}$ over w that witnesses that $w \in \mathscr{L}(G_{\mathcal{R}})$. Let τ_d be the primitive synopsis tree of $\mathcal{R}$ obtained from d. Then there exists $\sigma \in \mathsf{BST}(\mathcal{T})$ and a mapping λ from non-trivial nodes of τ_d to non-trivial nodes of σ such that τ_d is covered by σ with λ. In the following, we show how to construct a derivation tree of $G_{\mathcal{R},\mathcal{T}}$ over w starting from σ and λ. Clearly, this will prove that $w \in \mathscr{L}(G_{\mathcal{R},\mathcal{T}})$.

For the sake of simplification, we need to first introduce some notation. Given that λ is an injective mapping from non-trivial nodes of τ_d to non-trivial nodes of σ, we can define the unique inverse mapping $\lambda^{-1} : \mathsf{img}(\lambda) \to \mathsf{nodes}(\tau_d)$ where $\mathsf{img}(\lambda)$ is the image of λ. Furthermore, for any trees t, t' and function $f : A \to B$ where $A \subseteq \mathsf{nodes}(t)$ and $B \subseteq \mathsf{nodes}(t')$ we denote by $f|x$ the function $f|x : A|x \to B$ for every $x \in A$ such that $A|x = \{y \in \mathbb{N}^* \mid xy \in A\}$ and $f|x(y) = f(xy)$. That is, $f|x$ is equal to f restricted to suffixes of A with x as the common prefix.

We define the derivation tree $F(\sigma, \lambda)$ of $G_{\mathcal{R},\mathcal{T}}$ over w inductively over the structure of σ. From the construction, it will easily follow that $F(\sigma, \lambda)$ is a derivation tree of $G_{\mathcal{R},\mathcal{T}}$ over w. Let $\sigma = Y(\sigma_1, \sigma_2)$ where $Y \in \mathsf{SCC}(\mathcal{T})$ and $\sigma_1, \sigma_2 \in \mathsf{BST}(\mathcal{T})$. Define the derivation tree $F(\sigma, \lambda)$ inductively as follows:

- $F(\sigma, \lambda) = Y(F(\sigma_1, \lambda|1), F(\sigma_2, \lambda|2))$ whenever $\epsilon \notin \mathsf{img}(\lambda)$,
- $F(\sigma, \lambda) = Y(F(\sigma_1, \lambda|1), F(\sigma_2, \lambda|2), \langle\, \tau_d(\lambda^{-1}(\epsilon)) \,/\rangle)$ whenever $\epsilon \in \mathsf{img}(\lambda)$ and $\lambda^{-1}(\epsilon)$ is a horizontal component of $\mathcal{R}$,
- $F(\sigma, \lambda) = Y(\langle\, \tau_d(\lambda^{-1}(\epsilon)) \,\rangle, F(\sigma_1, \lambda|1), F(\sigma_2, \lambda|2), \langle /\, \tau_d(\lambda^{-1}(\epsilon)) \,\rangle)$ whenever $\epsilon \in \mathsf{img}(\lambda)$ and $\lambda^{-1}(\epsilon)$ is a non-horizontal component of $\mathcal{R}$.

The base case $\sigma = Y$ can be easily derived from the previous cases by omitting nodes $F(\sigma_1, \lambda|1)$ and $F(\sigma_2, \lambda|2)$ from the final result. In the specific case where $\epsilon \notin \mathsf{img}(\lambda)$, we define $F(\sigma, \lambda) = Y(\epsilon)$.

In order to show that $F(\sigma, \lambda)$ is a derivation tree of $G_{\mathcal{R},\mathcal{T}}$ over w, recall that λ preserves the post-order between non-trivial nodes and the ancestor-ship relation between non-horizontal and non-trivial nodes. From this last property and the construction of $F(\sigma, \lambda)$, it is a routine exercise to show that $F(\sigma, \lambda)$ is a derivation tree of $G_{\mathcal{R},\mathcal{T}}$ over w.

For the only-if direction, assume that $\mathscr{L}(G_{\mathcal{R}}) \subseteq \mathscr{L}(G_{\mathcal{R},\mathcal{T}})$ and consider a primitive synopsis tree τ. We will show a tree $\sigma \in \mathsf{BST}(\mathcal{T})$ and a covering λ such that σ covers τ with λ. First of all, remember that $\hat{\tau} \in \mathscr{L}(G_{\mathcal{R}})$ which implies that $\hat{\tau} \in \mathscr{L}(G_{\mathcal{R},\mathcal{T}})$. Hence let d be a derivation tree of $G_{\mathcal{R}}$ over $\hat{\tau}$ such that $\tau_d = \tau$, i.e. d is equal to τ by pruning all its leaves. Further, let d' be the derivation tree of $G_{\mathcal{R},\mathcal{T}}$ over $\hat{\tau}$ and $\sigma = \sigma_{d'}$ the basic synopsis tree obtained from d'. Note that the only difference between τ and d (or σ and d') are the terminal nodes at the leaves of d. In particular, we have that $\mathsf{nodes}(\tau) \subseteq \mathsf{nodes}(d)$ and $\mathsf{nodes}(\sigma) \subseteq \mathsf{nodes}(d')$. Therefore, in the rest of the proof we will use nodes in τ as nodes of d and vice-versa.

We devote the rest of the proof to showing that σ covers τ by using the derivation trees d and d'. Notice first that given two derivation trees d and d' over the same word, we can always find an injective mapping that maps terminal leaves of d (i.e. leaves labelled with elements in $\mathsf{tags}(\mathcal{R})$) to terminal leaves of d' preserving the same label and yield-order between leaves nodes. Denote then by λ' the injective mapping that maps terminals leaves in d to terminal leaves in d' that preserves the same label and post-order between leaves, i.e. $x \preccurlyeq_d^{\mathsf{post}} y$ iff $\lambda'(x) \preccurlyeq_{d'}^{\mathsf{post}} \lambda'(y)$ where $d(x), d(y) \in \mathsf{tags}(\mathcal{R})$. Now, translate the mapping λ' between terminal leaves of d and d' to a mapping λ between non-trivial nodes of τ and σ as follows: for every $x \in \mathsf{nodes}(\tau)$ and $y \in \mathsf{nodes}(\sigma)$ define $\lambda(\mathsf{parent}(x)) = \mathsf{parent}(y)$ whenever $\lambda'(x) = y$. That is, the function λ is defined by promoting λ' to the parents of its domain and image which are nodes in τ and σ, respectively. From the previous definition of λ, one can easily derive the following axiom for any $x \in \mathsf{dom}(\lambda)$:

$$\lambda(x) \;=\; \mathsf{parent}(\lambda'(\mathsf{last\text{-}child}_d(x))). \qquad (*)$$

We claim that λ is a covering mapping between τ and σ which concludes this proof. To start, notice that λ is an injective mapping between non-trivial nodes of τ to non-trivial nodes of σ. Indeed, λ' is an injective mapping and hence λ is also injective. Furthermore, if $x \in \mathsf{nodes}(d)$ or $x \in \mathsf{nodes}(d')$ are labelled with elements in $\mathsf{tags}(\mathcal{R})$, then $\tau(\mathsf{parent}(x))$ or $\sigma(\mathsf{parent}(x))$ are non-trivial components of $\mathcal{R}$ or $\mathcal{T}$, respectively. This implies that λ is an injective mapping between non-trivial nodes of τ to non-trivial nodes of σ. To conclude, we show that λ also satisfies properties 1), 2), and 3) of Definition 4.3.5:

1. For any non-trivial $x \in \mathsf{nodes}(\tau)$, we know that $\mathsf{last\text{-}child}_d(x)$ is labelled either with $\langle / \tau(x) \rangle$ or with $\langle \tau(x) / \rangle$. Without lost of generality, suppose that $d(\mathsf{last\text{-}child}_d(x)) = \langle \tau(x) / \rangle$. Then

$\lambda'(\mathsf{last\text{-}child}_d(x))$ is defined and $d'(\lambda'(\mathsf{last\text{-}child}_d(x))) = \langle\, \tau(x)\, /\rangle$. Recall that, by definition of $G_{\mathcal{R},\mathcal{T}}$, a node x in d' has a last-child labelled with $\langle\, X\, /\rangle$ for some $X \in \mathsf{SCC}(\mathcal{R})$ iff $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid d'(x))$. This implies that $\mathscr{L}(\mathcal{R} \mid \tau(x)) \subseteq \mathscr{L}(\mathcal{T} \mid \sigma(\lambda(x)))$ whenever $\lambda(x)$ has a last-child labelled with $\langle\, \tau(x)\, /\rangle$. By Equation (*), we have that $\lambda'(\mathsf{last\text{-}child}_d(x))$ is the last child of $\lambda(x)$ and is labelled with $\langle\, \tau(x)\, /\rangle$. This shows the first requirement of Definition 4.3.5.

2. For any non-trivial nodes $x, x' \in \mathsf{nodes}(\tau)$, we derive the following sequence of implications that proves the second requirement of Definition 4.3.5:

$$\begin{aligned} x \preccurlyeq^{\mathsf{post}}_{\tau} x' \quad \text{iff} \quad & \mathsf{last\text{-}child}_d(x) \preccurlyeq^{\mathsf{post}}_{d} \mathsf{last\text{-}child}_d(x') \\ \text{iff} \quad & \lambda'(\mathsf{last\text{-}child}_d(x)) \preccurlyeq^{\mathsf{post}}_{d'} \lambda'(\mathsf{last\text{-}child}_d(x')) \\ \text{iff} \quad & \mathsf{parent}(\lambda'(\mathsf{last\text{-}child}_d(x))) \preccurlyeq^{\mathsf{post}}_{\sigma} \mathsf{parent}(\lambda'(\mathsf{last\text{-}child}_d(x'))) \\ \text{iff} \quad & \lambda(x) \preccurlyeq^{\mathsf{post}}_{\sigma} \lambda(x'). \end{aligned}$$

3. Similar to the previous case, for any non-horizontal $x \in \mathsf{nodes}(\tau)$ and any non-trivial $x' \in \mathsf{nodes}(\tau)$ we derive the following sequence of implications that shows the last requirement of Definition 4.3.5:

$$\begin{aligned} x \preccurlyeq^{\mathsf{anc}}_{\tau} x' \quad \text{iff} \quad & \mathsf{first\text{-}child}_d(x) \preccurlyeq^{\mathsf{post}}_{d} \mathsf{last\text{-}child}_d(x') \preccurlyeq^{\mathsf{post}}_{d} \mathsf{last\text{-}child}_d(x) \\ \text{iff} \quad & \lambda'(\mathsf{first\text{-}child}_d(x)) \preccurlyeq^{\mathsf{post}}_{d'} \lambda'(\mathsf{last\text{-}child}_d(x')) \preccurlyeq^{\mathsf{post}}_{d'} \lambda'(\mathsf{last\text{-}child}_d(x)) \text{ and} \\ & \mathsf{parent}(\lambda'(\mathsf{first\text{-}child}_d(x))) = \mathsf{parent}(\lambda'(\mathsf{last\text{-}child}_d(x))) \\ \text{iff} \quad & \lambda(x) \preccurlyeq^{\mathsf{anc}}_{\sigma} \lambda(x'). \end{aligned}$$

□

Thanks to Lemma 4.5.2, we can reduce the problem of checking coverability between synopsis trees to deciding containment between two context-free grammars. Moreover, combining Theorem 4.3.6 and Lemma 4.5.2 we can decide whether $\mathscr{L}(\mathcal{R})$ is bounded repairable into $\mathscr{L}(\mathcal{T})$ by reducing $(\mathcal{R}, \mathcal{T})$ into $(G_{\mathcal{R}}, G_{\mathcal{R},\mathcal{T}})$ and then checking whether $\mathscr{L}(G_{\mathcal{R}}) \subseteq \mathscr{L}(G_{\mathcal{R},\mathcal{T}})$. Thus, the decidability of the bounded repair problem is reduced to the containment of two context-free grammars over $\mathsf{tags}(\mathcal{R})$.

In general, containment of context-free languages is undecidable (see [HU79] for a survey). However, in this particular case $G_{\mathcal{R}}$ is non-recursive and, further, each word is of at most exponential size with respect to $|G_{\mathcal{R}}|^2$. Therefore, we can decide whether $\mathscr{L}(G_{\mathcal{R}}) \subseteq \mathscr{L}(G_{\mathcal{R},\mathcal{T}})$ by universally-guessing a word $w \in \mathscr{L}(G_{\mathcal{R}})$ of size at most $2^{O(|G_{\mathcal{R}}|^2)}$ and then check if $w \in \mathscr{L}(G_{\mathcal{R},\mathcal{T}})$. The last step can be done efficiently in polynomial time [Ear70]. Hence, one can check containment of a non-recursive context-free grammar into a context-free grammar in coNEXPTIME.

To summarize, we have proved the following upper-bound for the bounded repair problem between regular tree languages.

Proposition 4.5.3. *The bounded repair problem between tree languages represented by non-deterministic stepwise automata is in coNEXPTIME.*

It is important to notice that, although $G_{\mathcal{R}}$ and $G_{\mathcal{R},\mathcal{T}}$ are of polynomial size with respect to $\mathcal{R}$ and $\mathcal{T}$, this reduction takes exponential time in the size of $\mathcal{R}$ and $\mathcal{T}$. Recall that the definition of $G_{\mathcal{R},\mathcal{T}}$ requires checking the containment between regular tree languages, which is EXPTIME-complete [Sei90]. Given that checking whether $\mathscr{L}(G_{\mathcal{R}}) \subseteq \mathscr{L}(G_{\mathcal{R},\mathcal{T}})$ already takes more than exponential time, the complexity of the reduction does not affect the running time of the full algorithm.

We now analyze the complexity of the bounded repair problem when tree languages are represented by DTDs. Recall that a DTD $D = (\Sigma, d, S)$ contains a function d that maps any letter $a \in \Sigma$ into a regular expression $d(a)$ over Σ (see Section 4.1). Furthermore, we call a DTD D *non-recursive* if its dependency graph is acyclic and *deterministic* if each $d(a)$ is given by DFAs. From results in [CGLN09] (in particular, from Proposition 4 and Theorem 5) it follows that any deterministic DTD can be turned into an equivalent deterministic stepwise automaton in polynomial time.

As a first remark, we recall that the containment problem for non-deterministic DTDs is PSPACE-complete. The lower bound follows easily from the PSPACE-hardness of containment of regular expressions [SM73]. The upper bound is a folklore result: given two DTDs D and D', one can decide whether the language defined by D is contained in the language defined by D' by first removing useless rules and then checking that the language $D(a)$ is contained in the language $D'(a)$ for every a in the alphabet of D. Although the problem is PSPACE-complete for general DTDs, the complexity lowers to PTIME whenever deterministic DTDs are considered.

Interestingly, the next result shows that the story is different in the case of the bounded repair problem. Proposition 4.5.4 below gives a strong result by showing that the bounded repair problem between languages specified by non-recursive deterministic DTDs is coEXPTIME-hard. Thus, this result collapses the complexity of the bounded repair problem to be coNEXPTIME-complete for all regular tree specifications considered in this work. The proof of this result is done by a technical reduction from the corridor tiling problem [Boa97].

Proposition 4.5.4. *The bounded repair problem between languages represented by deterministic DTDs is coNEXPTIME-hard, even for non-recursive DTDs.*

Proof. The proof is a reduction from the problem of tiling a square grid of exponential size. An instance of the latter problem is given by a tuple $I = (n, S, H, V, s_\perp, s_\top)$ where n is a positive number (i.e. the width of the square: 2^n), S is a finite set of available tiles, $H, V \subseteq S \times S$ are the set of vertical and horizontal constraints, and $s_\perp$ and $s_\top$ are the initial and final tiles. An exponential-tiling for I is a mapping f from the pairs $(i, j) \in \{1, \ldots, 2^n\} \times \{1, \ldots, 2^n\}$ to tiles in S. We say that a tiling f satisfies the constraints of I if the following conditions are satisfied:

1. $f(1,1) = s_\perp$ and $f(2^n, 2^n) = s_\top$,

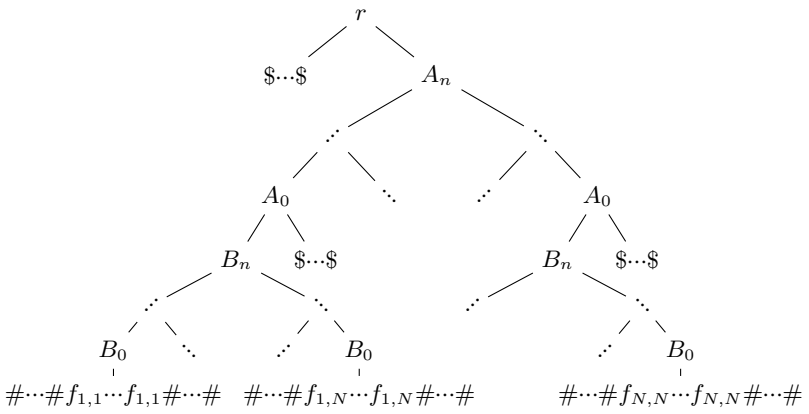


Figure 4.13: Encoding of a tiling function by an unranked tree.

2. $\big(f(i,j-1), f(i,j)\big) \in H$ for all $1 \leq i \leq 2^n$ and all $1 < j \leq 2^n$,

3. $\big(f(i-1,j), f(i,j)\big) \in V$ for all $1 < i \leq 2^n$ and all $1 \leq j \leq 2^n$.

The exponential tiling problem is the problem of deciding whether there exists an exponential-tiling f that satisfies the constraints in I given a tiling instance $I = (n, S, H, V, s_\bot, s_\top)$. It is known that this problem is NEXPTIME-complete [Boa97].

Hereafter, we fix an instance $I = (n, S, H, V, t_\bot, t_\top)$ of the exponential tiling problem and we construct two non-recursive deterministic DTDs R and T such that $\mathscr{L}(R)$ is bounded repairable into $\mathscr{L}(T)$ iff there is *no* tiling of the corridor that satisfies the constraints in I. This would imply that the bounded repair problem for non-recursive deterministic DTDs is coNEXPTIME-hard.

We start by defining the DTD for the restriction language R. The idea is to let R represent the set of all (redundant encodings of) exponential tilings. For the sake of brevity, we let $N = 2^n$. We let the restriction alphabet be $\Sigma = \{r, A_0, B_0, \ldots, A_n, B_n\} \uplus S \uplus \{\#, \$\}$, where $\#$ is used as a tile separator and $\$$ as a row separator. The non-recursive deterministic DTD R is given by the following set of rules:

$$
\begin{array}{lrcll}
R: & r & \to & \$^*\, A_n & \\
 & A_{i+1} & \to & A_i\, A_i & \forall i \in \{0, \ldots, n-1\} \\
 & A_0 & \to & B_n\, \$^* & \\
 & B_{i+1} & \to & B_i\, B_i & \forall i \in \{0, \ldots, n-1\} \\
 & B_n & \to & \#^*\, s^*\, \#^* & \forall s \in S
\end{array}
$$

Given any exponential-tiling $f : \{1, \ldots, N\} \times \{1, \ldots, N\} \to S$, we can always define an unranked tree that satisfies R and encodes f (see Figure 4.13). Notice that we repeat each tile $s \in S$ and each separator $\$$ and $\#$ in order to force a bounded repair strategy to keep them after the repair is done. In the sequel, we say that a string of the form $\#^* s^* \#^*$ for all $s \in S \cup \{\$\}$ is a *block*. For example, each tree in $\mathscr{L}(R)$ contains at most $M = 2^{2n} + 2^n + 1$ blocks.

One can check that if we delete all nodes labelled with elements in $\{A_0, B_0, \cdots, A_n, B_n\}$ from a tree in $\mathscr{L}(\mathcal{R})$, then the result is a tree rooted at r with a sequence of children of the form:

$$\$^* \#^* f^*_{1,1} \#^* f^*_{1,2} \#^* \cdots \#^* f^*_{1,N} \#^* \$^* \cdots \$^* \#^* f^*_{N,1} \#^* \cdots \#^* f^*_{N,N} \#^* \$^* \qquad (*)$$

Notice that we can separate the above string into M blocks: 2^{2n} blocks for all tiles, 2^n \$-blocks for each row, and one \$-block at the beginning.

The target language T is defined in a similar way and contains all redundant encodings of tilings that do not satisfy the constraints in I. Namely, a tree in T will contain M blocks hanging as leaves but distributed in such a way that a violation of a constraint in I is pointed out following a clever arrangement of the blocks in the tree. We first give the definition of T to later explain the main purpose of its structure. Consider the target alphabet

$$\Delta \;=\; \{r, A_0, B_0, \ldots, A_{2n}, B_{2n}, B'_0, \ldots, B'_n\} \uplus \{\mathsf{Err}_{HV}, \mathsf{Err}_V, \mathsf{Err}_H, \mathsf{Err}_\bot, \mathsf{Err}_\top\} \uplus S \uplus \{\#, \$\}.$$

In T, symbols A_i, B_i, B'_i are used to allocate blocks in the leaves of the tree and Err_l for $l \in \{HV, H, V, \bot, \top\}$ are used to indicate a violation of an l-constraint in I. The set of rules of T is defined as follows:

$$
\begin{array}{lrcll}
T: & r & \to & \mathsf{Err}_\bot \mid \mathsf{Err}_\top \mid \mathsf{Err}_{HV} & \\
& A_i & \to & A_{i-1}\, A_{i-1} & \forall i \in \{1, \ldots, 2n-1\} \\
& A_i & \to & \#^*\, s^*\, \#^* & \forall i \in \{0, 1, \ldots, 2n-1\}, \\
& & & & \forall s \in S \cup \{\$\} \\
& \mathsf{Err}_\bot & \to & \$^*\, \#^*\, s^*\, \#^*\, A_0\, A_1 \cdots A_{n-1}\, A_{2n} & \forall s \in S, s \neq s_\bot \\
& \mathsf{Err}_\top & \to & A_0\, A_1 \cdots A_{n-1}\, A_{2n}\, \#^*\, s^*\, \#^*\, \$^* & \forall s \in S, s \neq s_\top \\
& \mathsf{Err}_{HV} & \to & A_{2n-1}\, B_{2n-1} \mid B_{2n-1}\, A_{2n-1} & \\
& B_i & \to & A_{i-1}\, B_{i-1} \mid B_{i-1}\, A_{i-1} & \forall i \in \{1, \ldots, 2n-1\} \\
& B_0 & \to & \mathsf{Err}_V \mid \mathsf{Err}_H & \\
& \mathsf{Err}_V & \to & \#^*\, s^*_1\, \#^*\, A_n\, \#^*\, s^*_2\, \#^* & \forall (s_1, s_2) \notin V \\
& \mathsf{Err}_H & \to & A_{n-1}\, B'_{n-1} \mid B'_{n-1}\, A_{n-1} & \\
& B'_i & \to & A_{i-1}\, B'_{i-1} \mid B'_{i-1}\, A_{i-1} & \forall i \in \{1, \ldots, n-1\} \\
& B_0 & \to & A_0\, \#^*\, s^*_1\, \#^*\, s^*_2\, \#^* \mid \#^*\, s^*_1\, \#^*\, s^*_2\, \#^*\, A_0 & \forall (s_1, s_2) \notin H
\end{array}
$$

First of all, consider the rules for symbols $A_0, \ldots, A_{2n}$. The complete derivation of each node A_i for $i \in \{0, \ldots, 2n\}$ can produce a full binary tree of height $i+1$ whose leaves are blocks of the form $\#^* s^* \#^*$ for $s \in S \cup \{\$\}$. Hence, in the following keep in mind that each subtree starting at an A_i-node can allocate at most 2^i blocks.

Focus now on the root node r. There one can choose between three different child-nodes representing three possible sets of errors in a tiling: an error in the initial tile ($\mathsf{Err}_\bot$), in the final tile ($\mathsf{Err}_\top$), or in a vertical or horizontal constraint (Err_{HV}). Consider first an error in the initial tile $\mathsf{Err}_\bot$ (the case $\mathsf{Err}_\top$ is analogous). The third production in T includes two blocks of the form $\$^*$ and

$\#^* s^* \#^*$, and a sequence of sub-trees $A_0, \ldots, A_{n-1}, A_{2n}$ which can allocate $2^0, \ldots, 2^{n-1}, 2^{2n}$ number of blocks. By summing all these blocks, the number of blocks in a tree with an $\mathsf{Err}_\perp$-node can include: $2 + \sum_{i=0}^{n-1} 2^i + 2^{2n} = 2 + (2^n - 1) + 2^{2n} = M$. Therefore, to point out an error in the initial tile, a repair strategy can delete all the structural nodes, produce a string like (*), and then insert A-nodes to distribute all the blocks in sub-trees $A_0, \ldots, A_{n-1}, A_{2n}$. Note that the number of repairs is bounded for any tree-encoding of an exponential-tiling with the first tile different from $s_\perp$.

In case that a violation of a vertical or horizontal constraint in I happens, we distribute the blocks below an Err_{HV}-node and point to the two tiles that are violating a horizontal or vertical constraint. Suppose that a tiling f is violating a vertical constraint at tiles $(i-1, j)$ and (i, j). Then the two blocks:

$$\#^* f(i-1, j)^* \#^* \quad \text{and} \quad \#^* f(i, j)^* \#^*$$

represented in the sequence (*) have (1) 2^n blocks in between, (2) N_1 blocks to the left, and (3) N_2 blocks to the right for some $N_1, N_2 \in \mathbb{N}$ such that $N_1 + N_2 = 2^{2n} - 1$ (recall that the total sum of blocks must be $M = 2^n + 2 + N_1 + N_2$). Clearly, we do not know in advance the exact positions $(i-1, j)$ and (i, j) which determine the values N_1 and N_2. However, we know that both values satisfy $N_1 + N_2 = 2^{2n} - 1$ which implies a specific pattern in the binary encoding of both numbers. The binary representation of the value $2^{2n} - 1$ is a sequence of $2n$ ones (i.e. $1 \overset{2n}{\cdots} 1$). Then, in order to satisfy $N_1 + N_2 = 2^{2n} - 1$, the binary representation of N_1 must be the opposite representation of N_2. That is, if $\mathsf{bin}(N_1) = b_0 b_1 \ldots b_{2n-1}$, then $\mathsf{bin}(N_2) = \bar{b}_0 \bar{b}_1 \ldots \bar{b}_{2n-1}$ where $\mathsf{bin}(\cdot)$ denotes the binary representation of a number (starting from the less significant bit), $b_0, \ldots, b_{2n-1} \in \{0, 1\}$, and $\bar{b}_k = 0$ iff $b_k = 1$ for all $k \in \{0, \ldots, 2n-1\}$. Indeed, if $\mathsf{bin}(N_1)$ and $\mathsf{bin}(N_2)$ coincide in a position, then the value of this position in $\mathsf{bin}(N_1 + N_2)$ must be equal to 0 which is a contradiction.

Coming back to the encoding of a vertical violation in T, we follow the binary representation of N_1 and N_2 to distribute the $2^{2n} - 1$ blocks at both sides of $(i-1, j)$ and (i, j). Figure 4.14 shows a possible representation of f pointing out the vertical violation in tiles $f(i-1, j)$ and $f(i, j)$. Note that the total number of blocks allocated in Figure 4.14 is equal to M. Furthermore, the tree uses the pattern of $\mathsf{bin}(N_1)$ and $\mathsf{bin}(N_2)$ to distribute the $2^{2n} - 1$ blocks at both sides of the violation at positions $(i-1, j)$ and (i, j). The (k)-bit of $\mathsf{bin}(N_1)$ ($\mathsf{bin}(N_2)$) is equal to 1 iff the children of B_k are equal to $A_{k-1} B_{k-1}$ ($B_{k-1} A_{k-1}$, resp.) for every $k \in \{1, \ldots, 2n\}$. One can easily check that the total number of blocks on the left-side (right-side) of Err_V is equal to N_1 (N_2, resp.). We conclude that all M blocks can be distributed in a unique way (with respect to the structure of T) in order to point out a vertical violation at positions $(i-1, j)$ and (i, j).

A similar representation is used to point out a violation of a horizontal constraint in I. In this case, the number of blocks on the left and right side of two consecutive tiles sums to $2^{2n} + 2^n - 1$. For this representation a sequence of nodes $B'_0, \ldots, B'_n$ is used in addition to the previous sequence $B_0, \ldots, B_{2n-1}$ that was considered for a vertical error.

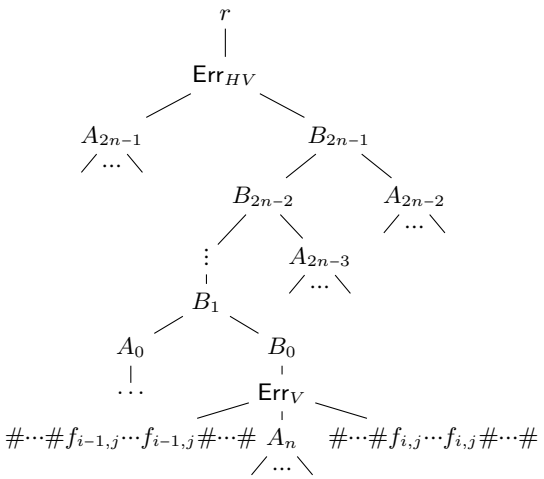


Figure 4.14: Encoding of a vertical error in a tiling.

Note that at most $2^{2n} + 2^n + 1$ blocks can be allocated inside a tree satisfying T for the four possible scenarios (i.e. $\mathsf{Err}_\perp$, $\mathsf{Err}_\top$, $(\mathsf{Err}_{HV}, \mathsf{Err}_V)$, and $(\mathsf{Err}_{HV}, \mathsf{Err}_H)$). Hence, a bounded repair strategy is restricted to redistribute all M-blocks in order to show a possible violation in a tiling and it cannot cheat by either adding or removing blocks from a tree satisfying R. In fact, giving any exponential tiling encoded in a tree $t \in \mathscr{L}(R)$ with a violation of a constraint in I, we can modify t by deleting all nodes $A_0, B_0, \ldots, A_n, B_n$ and then inserting the necessary structure (i.e. $\mathsf{Err}_\perp$, $\mathsf{Err}_\top$, $(\mathsf{Err}_{HV}, \mathsf{Err}_V)$) in order to satisfy T depending on the type of violation. One can easily check that the converse is also true. That is, if there exists an exponential tiling f satisfying I, then we can construct a sequence of trees encoding f satisfying R such that any repair strategy will need an unbounded number of edit-operations to satisfy T. Finally, one can easily check that both DTDs R and T can be defined by non-recursive deterministic DTDs. □

Combining Proposition 4.5.3 and Proposition 4.5.4 we obtain that the complexity of the bounded repair problem for all standard specifications of regular tree languages [MNSB06] is coNEXPTIME-complete. Furthermore, this result holds even for non-recursive deterministic DTDs. For the sake of completeness, we highlight this result in the following theorem.

Theorem 4.5.5. *The bounded repair problem between languages represented by either (1) stepwise tree automata, (2) deterministic stepwise tree automata, or (3) non-recursive deterministic DTDs is coNEXPTIME-complete.*

4.6 The unrestricted case

In this section we consider the so-called unrestricted case of the bounded repairability problem, namely, a variant of the problem where the restriction language is assumed universal (i.e., equal to

$\mathcal{T}_\Sigma$) and the target language is represented by a stepwise automaton $\mathcal{T}$.

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

In this section we also make use of *deterministic visibly pushdown transducers* [RS08, AM09] as suitable devices that transform unranked trees in a streaming fashion. These devices receive the serialized version of an unranked tree and output the serialized version of another unranked tree. By a slight abuse of notation we identify unranked trees with their serializations (see Chapter 5 for more details).

The following result gives equivalent conditions for bounded repairability in the unrestricted case.

Proposition 4.6.1. *Given an alphabet Σ and an automaton $\mathcal{T} = (\Delta, Q, \delta, \delta_0, F)$, the following conditions are equivalent:*

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

Proof. First observe that the third item trivially implies the first one. Below, we prove that the first item implies the second one, by contraposition. Suppose that $\mathcal{T} = (\Delta, Q, \delta, \delta_0, F)$ is a (trimmed) stepwise automaton that is not complete over Σ. Let t_0 be an unranked tree over Σ such that $\delta(\mathsf{ext}(t_0)) = \varnothing$, let a be any symbol from Σ, and, for every $n \geq 1$, let

$$t_n \;=\; a(\underbrace{t_0, \ldots, t_0}_{n \text{ times}}).$$

Clearly, $\mathsf{ext}(t_n) \in \mathcal{T}_\Sigma$. Moreover, at least n edit operations are required to repair the tree t_n into the language $\mathsf{ext}^{-1}(\mathscr{L}(\mathcal{T}))$. This shows that $\mathcal{T}_\Sigma$ is not bounded repairable into $\mathscr{L}(\mathcal{T})$.

Finally, we prove that the second item implies the third one. Suppose that $\mathcal{T} = (\Delta, Q, \delta, \delta_0, F)$ is a (trimmed) stepwise automaton that is complete over Σ. It is not difficult to show that from the fact that $\mathcal{T}$ is complete over Σ it follows that $\mathcal{T}_\Sigma$ is bounded repairable into $\mathscr{L}(\mathcal{T})$. The interesting result is that, when we identify unranked trees with their serializations, the repair strategy of $\mathcal{T}_\Sigma$ into $\mathscr{L}(\mathcal{T})$ can be implemented by a deterministic visibly pushdown transducer. More specifically, the deterministic visibly pushdown transducer outputs, at the very first step and independently of the input, a fixed prefix of a serialized unranked tree (this represents a portion of the repaired tree); then it copies the input t as a continuation of the prefix formerly constructed, mimicking at the same time the computation of the stepwise automaton $\mathcal{T}$ on $\mathsf{ext}(t)$; finally, the transducer terminates

by outputting a suitable suffix in such a way that the corresponding repaired tree belongs to the language $\mathsf{ext}^{-1}(\mathscr{L}(\mathcal{T}))$. The difficult part of this proof is to show that there is a single prefix that, no matter how it is prolonged, can be completed into a serialized tree that belongs to the language $\mathsf{ext}^{-1}(\mathscr{L}(\mathcal{T}))$. To show this, we prove the following claim under the assumption that the automaton $\mathcal{T}$ is complete:

Claim 1. *There are a symbol $a \in \Sigma$, a state $p \in Q$, and a sequence of unranked trees $u_1, \ldots, u_n$ over Σ such that for every unranked tree t over Σ, there is a sequence of unranked trees $v_1, \ldots, v_m$ over Σ satisfying $p \in \delta(\mathsf{ext}(a(u_1, \ldots, u_n, t, v_1, \ldots, v_m)))$.*

Proof. We prove the claim by contraposition. Suppose that $(\star)$ for every symbol $a \in \Sigma$, every state $p \in Q$, and every sequence $u_1, \ldots, u_n$ of trees, there exists a tree t such that, for every sequence of trees $v_1, \ldots, v_m$, $p \notin \delta(\mathsf{ext}(a(u_1, \ldots, u_n, t, v_1, \ldots, v_m)))$. We fix an arbitrary symbol $a \in \Sigma$ and an enumeration $p_1, \ldots, p_N$ of all states in Q. Then, by applying the hypothesis $(\star)$ to the symbol a, to each state $p \in \{p_1, \ldots, p_N\}$, and to increasing sequences of trees $u_1, \ldots, u_n$, we construct a tree t' over Σ on which $\mathcal{T}$ has no valid run (this would imply that $\mathcal{T}$ is not complete over Σ). First, we let $p = p_1$ and $n = 0$, and we obtain from $(\star)$ that there is a tree t_1 such that $p_1 \notin \delta(\mathsf{ext}(a(t_1, v_1, \ldots, v_m)))$ for all sequences of trees $v_1, \ldots, v_m$. Similarly, if we let $p = p_2$, $n = 1$, and $u_1 = t_1$, we know from $(\star)$ that there is a tree t_2 such that $p_2 \notin \delta(\mathsf{ext}(a(t_1, t_2, v_1, \ldots, v_m)))$ for all sequences of trees $v_1, \ldots, v_m$. By applying a simple inductive argument, we can construct a sequence of trees $t_1, \ldots, t_N$ such that for every index $1 \leq i \leq N$, $p_i \notin \delta(\mathsf{ext}(a(t_1, \ldots, t_N)))$. Since $p_1, \ldots, p_N$ are all and only the states of $\mathcal{T}$, we derive that $\mathcal{T}$ has no valid run on $\mathsf{ext}(a(t_1, \ldots, t_N))$. This shows that $\mathcal{T}$ is not complete over Σ. □

Turning back to the main proof, we can use the above claim to construct a deterministic visibly pushdown transducer that repairs any tree over Σ into a tree in the language $\mathsf{ext}^{-1}(\mathscr{L}(\mathcal{T}))$ with uniformly bounded number of edit operations. Let a, p, $u_1, \ldots, u_n$ be as in the claim above. Since $\mathcal{T}$ is trimmed, there is a (curried) context C over Δ such that $\delta(p, C) \cap F \neq \varnothing$ and the automaton $\mathcal{T}$ accepts the context C when the placeholder is assigned the state p. From now on we identify the unranked trees $u_1, \ldots, u_n$ over Σ with the corresponding serializations over Σ. Similarly, we can represent the context C with a pair $(C^{\mathrm{prefix}}, C^{\mathrm{suffix}})$ of sequences of opening and closing tags such that, for every tree t, $C^{\mathrm{prefix}} \cdot t \cdot C^{\mathrm{suffix}}$ is the serialization of the unranked tree $\mathsf{ext}^{-1}(C \circ \mathsf{ext}(t))$ (note that the sequences C^{prefix} and C^{suffix} need not to be well-parenthesized). In particular, any curried tree of the form $C \circ \mathsf{ext}(a(u_1, \ldots, u_n, t, v_1, \ldots, v_m))$ is represented by the word

$$C^{\mathrm{prefix}} \cdot \langle a \rangle \cdot u_1 \cdot \ldots \cdot u_n \cdot t \cdot v_1 \cdot \ldots \cdots v_m \cdot \langle a/ \rangle \cdot C^{\mathrm{suffix}}.$$

Moreover, from the previous claim and assumptions, it follows that for every tree t, there is a sequence of trees $v_1, \ldots, v_m$ such that $C \circ \mathsf{ext}(a(u_1, \ldots, u_n, t, v_1, \ldots, v_m))$ is accepted by $\mathcal{T}$. Therefore, the deterministic visibly pushdown transducer that implements the bounded repair strategy of $\mathcal{T}_\Sigma$ into

$\mathscr{L}(\mathcal{T})$ can be constructed as follows. First, it outputs the prefix $C^{\text{prefix}} \cdot \langle a \rangle \cdot u_1 \cdot \ldots \cdot u_n$, independently of the input; then, it copies the serialized input tree t and it mimics at the same time the computation of $\mathcal{T}$ on $\mathsf{ext}(t)$ (this is possible since stepwise automata can be translated into equivalent deterministic visibly pushdown automata); finally, on the basis of the state reached by $\mathcal{T}$ after parsing $\mathsf{ext}(t)$, the transducer outputs a suitable suffix of the form $v_1 \cdot \ldots \cdot v_m \cdot \langle a/ \rangle \cdot C^{\text{suffix}}$ in such a way that the final output represents a tree inside the language $\mathsf{ext}^{-1}(\mathscr{L}(\mathcal{T}))$ (this is possible thanks to the above claim). To conclude the proof it is sufficient to observe that the tree output by the transducer can be obtained from its input by performing a uniformly bounded number of insert operations. □

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

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

Proposition 4.6.2. *The bounded repair problem in the unrestricted case when the target language is represented by a non-deterministic (resp., deterministic) stepwise automaton is EXPTIME-complete (resp., PTIME-complete).*

Proof. From Proposition 4.6.1 one can derive a polynomial-time algorithm that decides whether $\mathcal{T}_\Sigma$ is bounded repairable into $\mathscr{L}(\mathcal{T})$ when $\mathcal{T}$ is given by a deterministic stepwise automaton. For this it is sufficient to first turn $\mathcal{T}$ into a trimmed automaton $\mathcal{T}' = (\Sigma, Q', \delta', \delta'_0, F')$ over Σ and then check that (i) for every symbol $a \in \Sigma$, $\delta'_0(a) \neq \varnothing$ and (ii) for every pair of states $q_1, q_2 \in Q'$, $\delta'(q_1, q_2) \neq \varnothing$. For a stepwise automaton, this procedure takes polynomial time in the size of $\mathcal{T}$. On the other hand, if $\mathcal{T}$ is non-deterministic, one can determinize $\mathcal{T}$ and then check property (i) and (ii) over its determinization. This algorithm takes exponential time for the non-deterministic case. Therefore, we conclude that the problem is in PTIME for deterministic stepwise automata and in EXPTIME for non-deterministic stepwise automata.

For the lower bounds, we show here that the problem is EXPTIME-hard for non-deterministic stepwise automata by a reduction of the universality problem for stepwise automata. For the PTIME hardness of the deterministic case, the reduction is similar but one has to use the emptiness problem of deterministic tree automata that is well known to be PTIME-complete.

Consider a trimmed stepwise automata $\mathcal{T} = (\Sigma, Q, \delta, \delta_0, F)$ as an instance of the universality problem. We define the stepwise automata $\mathcal{T}' = (\Sigma \cup \{\#\}, Q \cup \{q_\#, q_f\}, \delta', \delta_0 \cup (\#, q_\#), \{q_f\})$ where $\#$ is a new label and $q_\#, q_f$ are new states. For the transition function, we define:

$$\begin{aligned}\delta' \quad = \quad & \delta \cup (F \times \{q_\#\} \times \{q_f\}) \cup (\{q_\#\} \times F \times \{q_f\}) \cup (\{q_\#\} \times \{q_\#\} \times \{q_\#\}) \cup \\ & (Q' \times \{q_f\} \times \{q_f\}) \cup (\{q_f\} \times Q' \times \{q_f\})\end{aligned}$$

where $Q' = Q \cup \{q_\#, q_f\}$ is the complete set of states in $\mathcal{T}'$. In the sequel we show that $\mathscr{L}(\mathcal{T})$ is universal if, and only if, $\mathcal{T}_{\Sigma\cup\{\#\}}$ is bounded repairable into $\mathscr{L}(\mathcal{T}')$.

Suppose that $\mathcal{T}$ contains the universal tree language, namely, $\mathcal{T}_\Sigma \subseteq \mathscr{L}(\mathcal{T})$. By Proposition 4.6.1, we only need to show that $\mathcal{T}'$ is complete over $\Sigma \cup \{\#\}$. First, notice that $\mathcal{T}'$ is already trimmed given that $\mathcal{T}$ is trimmed and one can easily check that states $q_\#$ and q_f appear in some accepting run. To show completeness, let $t \in \mathcal{T}_{\Sigma\cup\{\#\}}$ be any curried tree. If t does not contain label $\#$, then there exists trivially a run of $\mathcal{T}'$ over t given that $\delta \subseteq \delta'$ and $\mathscr{L}(\mathcal{T})$ is universal. Now suppose that t contains at least one node labeled by $\#$. If t only contains $\#$-labels, then it is straightforward to show a run of $\mathcal{T}'$ over t. Otherwise, let t' be the smallest subtree of t such that $t' = t_1 @ t_2$ and either $t_1 \in \mathcal{T}_{\{\#\}}$ and $t_2 \in \mathcal{T}_\Sigma$ or vice versa (one can easily show that t' always exists in t). Without loss of generality, take $t_2 \in \mathcal{T}_\Sigma$. Since $\mathscr{L}(\mathcal{T})$ is universal, we know that there exists an accepting run of $\mathcal{T}$ over t_2 that reaches a final state q. On the other hand we have that $q_\# \in \delta'(t_1)$ and then $q_f \in \delta'(t')$. Finally, one can easily check by the definition of $\mathcal{T}'$ that for every context $C \in \mathcal{C}_{\Sigma\cup\{\#\}}$, it holds that $q_f \in \delta(q_f, C)$. This implies that $q_f \in \delta(t)$ and, therefore, $\mathcal{T}$ is complete over $\Sigma \cup \{\#\}$.

For the other direction, suppose that $\mathcal{T}_\Sigma \nsubseteq \mathscr{L}(\mathcal{T})$ and take a tree $t \notin \mathscr{L}(\mathcal{T})$. Define a new tree $t' = t \,@\, \#$. By the construction of $\mathcal{T}'$, one can easily check that $\delta'(t') = \varnothing$ and we have that $\mathcal{T}'$ is not complete over $\Sigma \cup \{\#\}$. From this, we conclude that the bounded repair problem in the unrestricted case for non-deterministic stepwise automata is EXPTIME-hard as well. This completes the proof. □

It is worth remarking some similarities and differences between the considered problem and the analogous unrestricted bounded repair problem for regular languages of words (Chapter 2). First of all, the characterization given in Proposition 4.6.1 implies that, in the unrestricted case only, if the universal tree language $\mathcal{T}_\Sigma$ is bounded repairable into a regular tree language T, then the repair strategy can be implemented by a deterministic visibly pushdown transducer that works directly on serializations of unranked trees. This is reminiscent of the equivalence between the unrestricted repair problems for regular word languages in the non-streaming and in the streaming settings. We also remark that bounded repairability for tree languages is a much stronger property than that for word languages. In Corollary 2.4.11, it is shown that for every alphabet Σ and every regular word language T, Σ^* is bounded repairable into T or into its complement $T^\complement$. When considering languages of trees the situation is quite different: for every alphabet Σ, there are regular tree languages T such that $\mathcal{T}_\Sigma$ is bounded repairable neither into T nor into its complement $T^\complement$.

4.7 Conclusions

In this chapter we study the bounded repair problem for regular tree specifications. Specifically, we give the first effective characterization for determining whether one regular tree language can be repaired to another regular tree language with a finite number of edits. In contrast to the previous chapters, understanding the bounded repair problem over trees was much more involved and most of the machinery used in the word case was extended in order to capture the bounded repair problem over trees. As an example, there was a clear similarity between the paths of SCCs for finite word automata and the synopsis trees of SCCs for finite tree automata. Furthermore, the coverability relations were very similar in both cases. Although the coverability relation between synopsis trees was more complicated than the coverability relation between paths of components, both preserve the idea that components have to maintain some fix order and this cannot be changed by a bounded number of edit operations. We think that the extension of the characterization from string to trees case is elegant and, further, it captures most of the fundamental concepts needed in order to understand the bounded repair problem between regular tree languages.

A different story is the complexity analysis of the bounded repair problem given in this chapter. As was mentioned in Section 4.5, a straightforward application of the main characterization gives a Π_2^{EXP}-algorithm for deciding whether one regular tree language can be repaired to another regular tree language with a finite number of edits. In order to find the exact complexity of the bounded repair problem, we have to reduce our characterization to the containment of non-recursive context-free languages. With this final step we were able to sketch a coNEXPTIME-algorithm for the bounded repair problem over trees. Furthermore, we provide a tight lower bound of the bounded repair problem by showing that even for non-recursive deterministic DTDs the problem is coNEXPTIME-hard. This last result contrasts with the word setting where the bounded repair problem for deterministic automata has a much lower complexity than the non-deterministic case. In some sense, the containment of non-recursive context free languages is embedded even in the deterministic case and this produces an explosion in the complexity.

This chapter contains new results that were not included in [PRS12]. For example, the best upper-bound for the bounded repair problem given in [PRS12] was a Π_2^{EXP}-algorithm that follows from Lemma 4.5.1 and, therefore, the coNEXPTIME-bounds given in Section 4.5 are new. Finally, it is worth mentioning that in [PRS12] we study further restrictions over tree specifications that were not included in this chapter. There, we show that bounded repairability is much less complex for DTDs when the alphabet is fixed.

Related work. Edit distance between trees has been extensively studied and several polynomial-time algorithms exist based on dynamic-programming (see [Bil05] for a survey). The edit operations presented here was first introduce in [Tai79] as a generalization of the edit distance for strings.

The main advantage of these operations over trees is that they preserve the ancestorship relation and post-order between nodes, which is a desirable property in order to maintain the underlying relations between data on an XML document. In [Sel77] a different edit distance was considered where insertions and deletions are restricted to leaves of the trees. This edit distance is usually called *1-degree edit distance*. Depending on the application, these operations are argued to be more accurate and useful for XML documents [CAM02, SC06].

The problem of computing the distance between a tree and a schema has also been studied before [BdR04]. Under certain conditions, the algorithm proposed in [BdR04] returns a valid document whose distance from the original one is guaranteed to be within a multiplicative constant of the minimum distance. In this context it is also relevant to cite the work in [AP72]. In this paper the edit distance of a string to a context-free language was studied and an algorithm to compute the minimum repair was given. Clearly, if edit operations over strings are applied over the XML encoding of a tree, then theses operations can be lifted from strings and context free languages to XML documents and DTDs.

Regarding similarity between schemas, syntactic similarity between schemas were studied in [FB08] and a similarity set-based measure was proposed in [AGMN11], but both works did not consider edit operations between trees. Approximation of regular tree languages by single-type regular tree languages has been studied in [GIM+13] were the setting is restricted to a particular class of tree languages (single-type) and the approximation is modeled by language containment. To the best of our knowledge, the problem of editing all trees in a schema to trees in another schema has not been studied explicitly.

Distance automata over trees are a natural generalization of distance automata over words [CL10]. In contrast with the string case, in this chapter we did not study any connection between the bounded repair problem and the boundedness for distance automata on trees. Recall that in Chapter 2 and 3 we show that, for every regular language T, one can reduce the function $\mathsf{cost}(u, T) = \min_{v \in T} \mathsf{dist}(u, v)$ into a distance automaton over words in polynomial time. Interestingly, for the tree case it is not clear that for every regular tree language T one can construct (in polynomial time) a tree distance automaton that computes the cost of repairing trees into T. It is not even clear that distance automata are expressible enough to compute this type of functions, that is, whether there exists a tree distance automata $\mathcal{D}_T$ such that $\mathcal{D}_T(t) = \mathsf{cost}(t, T)$ for every regular tree language T. We left this question as an open problem.

Chapter 5

Streaming repairing of trees

In this last chapter, we tackle the question of which pairs of trees specifications are streaming repairable with uniformly bounded cost. Intuitively, a streaming repair is a procedure that inserts, deletes, or modifies the tree structure while reading the tree in pre-order fashion, producing an output that satisfies some constraints. 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 being *bounded streaming repairable* (see Chapter 2). Specifically, we study the problem of determining whether there exists a stream processor that will (i) ensure that any tree satisfying a restriction specification is repaired to a tree satisfying the target specification and (ii) performs a number of edits that is uniformly bounded and independent of the input.

The previous chapter gave a characterization and decision procedure for determining whether a bounded repair strategy exists in the non-streaming setting. In this chapter we give a characterization and decision procedure for determining whether a streaming bounded repair strategy exists, in the important case of DTD schemas, and more generally of 'deterministic top-down schemas'. This is a subclass of tree automata that subsumes not only DTDs but also XML Schemas (XSDs).

Our 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 the previous chapters: we associate a graph with each schema, and then look at the corresponding notion of connected component; such components represent 'repeatable behaviours'. 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 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 tree languages. 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 this game, and thus decidability of a winner.

Both directions of the proof of this characterization are highly non-trivial. In one direction, we manufacture an effective tree 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. Finally, we complement this with a matching lower bound.

An outline of this chapter is the following: we start in Section 5.1 giving the basic definitions on top-down tree automata and their serialization. Then in Section 5.2 we state the streaming repair problem formally, and we give examples that explain the difficulties of the problem and motivate its solution. Our main characterization theorem is stated in Section 5.3, which relies on defining a new kind of game played on stacks of components. We devote Section 5.4 to explaining the first direction of the proof, namely, from the simulation game to the bounded repair problem, and Section 5.5 for the other direction. In Section 5.6 we consider the consequences of our characterization theorem for complexity, while in Section 5.7 we give our concluding remarks.

5.1 Tree specifications and their serializations

We adopt the usual notations for strings and unranked trees considered in the previous chapters (see Sections 2.1 and 4.1 for more details). Also we consider the serialized representation of trees (or *XML* encoding) in order to study streaming repair problems over trees.

In this section, we implicitly use the *first-child-next-sibling* encoding for representing unranked trees instead of the curry encoding considered in the previous section. Interestingly, it turns out that the first-child-next-sibling encoding is more suitable for studying streaming repair problems over trees and closer to top-down tree automata (i.e. the class of tree specifications used in this section) than the curry encoding.

Top-down tree automata. In this chapter we make use of languages of unranked trees, such as those defined by the structural components of DTDs and XSD schemas [MLMK05, MNSB06]. We work with 'deterministic top-down schemas' [MNS08, CLT05], which generalize DTDs and can be seen as typing systems in which the type associated with each internal node of a tree depends uniquely on the type of the parent and the type of the left sibling (if this exists). Equivalently, one can think of these schemas as deterministic top-down binary tree automata running on the standard first-child-next-sibling encoding of an unranked tree. Formally, a *(deterministic) top-down tree automaton* is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, where:

1. Σ is a finite alphabet,

2. Q is a finite set of states,

3. $\delta : Q \times \Sigma \rightharpoonup Q \times Q$ is a partial transition function,

4. $q_0 \in Q$ is an initial state, and

5. $F \subseteq Q$ is a set of final states.

To avoid switching every time between unranked trees and their encodings (like in Chapter 4), we define the runs of our automata directly on unranked trees and unranked forests. Given an unranked tree/forest t, we denote by $\mathsf{nodes}^+(t)$ the *extended domain* of t, which is defined as the set that contains all nodes of t and all sequences $\vec{i} \cdot j \cdot 1 \in \mathsf{nodes}^+(t)$ and $\vec{i} \cdot (j+1) \in \mathsf{nodes}^+(t)$, with $\vec{i} \cdot j \in \mathsf{nodes}(t)$. Intuitively, $\mathsf{nodes}^+(t)$ is the extension of the domain of t that results from adding a new child to each leaf and a new sibling to each node with no right sibling. A *run* of $\mathcal{A}$ on t is a function $\rho : \mathsf{nodes}^+(t) \to Q$ such that $\delta\big(\rho(\vec{i} \cdot j), t(\vec{i} \cdot j)\big) = \big(\rho(\vec{i} \cdot j \cdot 1), \rho(\vec{i} \cdot (j+1))\big)$ for all $\vec{i} \cdot j \in \mathsf{nodes}(t)$. A run ρ is said to be *accepting* if $\rho(1) = q_0$ and $\rho(\vec{i}) \in F$ for all nodes $\vec{i} \in \mathsf{nodes}^+(t) \setminus \mathsf{nodes}(t)$. The language recognized by $\mathcal{A}$ is the set $\mathscr{L}(\mathcal{A})$ of all trees $t \in \mathcal{T}_\Sigma$ that induce an accepting run of $\mathcal{A}$.

Top-down tree automata as defined above are always deterministic. In this chapter we only consider the deterministic version of these tree automata and, thus, in the future we refer to them just as top-down tree automata.

Similar to previous chapters, in the sequel we will work with *trimmed* automata only, namely, all states of our automata appear in some accepting run. Formally, we say that a top-down tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ is *trimmed* if for every $q \in Q$ there exists a tree $t \in \mathcal{T}_\Sigma$ and an accepting run ρ of $\mathcal{A}$ over t such that $\rho(\vec{i}) = q$ for some $\vec{i} \in \mathsf{nodes}^+(t)$. Because useless states of automata can be detected and removed in linear time [CDG+07], this assumption will have no impact on our complexity results.

Top-down tree automata define a proper subset of regular tree languages [MNSB06, MNS08] and then they are less expressive than stepwise-tree automata (consider previously in Chapter 4). We switch to a less expressive specification formalism given that our characterization of streaming tree bounded repairability works only for this restricted fragment of regular tree specifications (see Section 5.2 for further explanations and examples). Despite of the lack of expressibility, top-down tree automata are more expressive than DTDs or even XML Schemas [MNS08] which are the main XML specification language consider in practical applications.

Example 13. *Consider the following DTDs that are slight variations of the DTDs presented in Chapter 4, Example 11:*

$$
\begin{array}{lllll}
D: & r & \to & a\,d \\
 & a & \to & a + \epsilon \\
 & d & \to & b\,c^* \\
 & b & \to & a + \epsilon \\
 & c & \to & \epsilon
\end{array}
\qquad\qquad
\begin{array}{lllll}
D': & r & \to & d\,c^* \\
 & d & \to & a\,a + a \\
 & a & \to & a + \epsilon \\
 & c & \to & \epsilon
\end{array}
$$

The left-hand side of Figure 5.1 gives an example of an unranked tree satisfying the DTDs D.

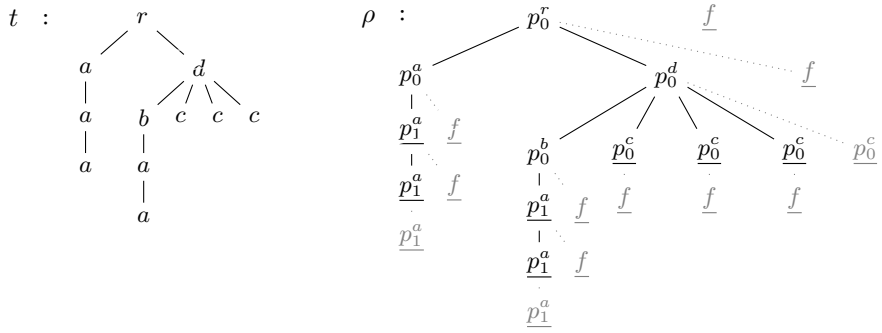


Figure 5.1: Run of a top-down tree automaton over an unranked tree.

We can easily define top-down tree automata automata $\mathcal{R}$ and $\mathcal{T}$ that recognize the languages defined by D and D', respectively. The following are the transitions rules of $\mathcal{R}$ and $\mathcal{T}$ such that $\mathscr{L}(D) = \mathscr{L}(\mathcal{R})$ and $\mathscr{L}(D') = \mathscr{L}(\mathcal{T})$:

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

where p_0^r and q_0^r are the initial states and the final states are underlined. Furthermore, one can easily check that the right-hand side of Figure 5.1 is the accepting run of the automaton $\mathcal{R}$ on the tree t. In this figure, black nodes correspond to elements of t, while gray nodes have no corresponding element in t (i.e. $\mathsf{nodes}^+(t) \setminus \mathsf{nodes}(t)$).

For this model of automata, it is useful to consider a restricted type of context. Recall that a context in Chapter 4 is a curry tree with a leaf labelled by the special symbol $\bullet$. We redefine contexts over Σ and the set of context $\mathcal{C}_\Sigma^\downarrow$ motivated by the tree automata model (i.e. top-down tree automata) considered in this chapter. Formally, a *top-down context* (or just a context) over Σ is an unranked tree or an unranked forest labelled over $\Sigma \uplus \{\bullet\}$, where $\bullet$ occurs 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. Note that top-down contexts are the unranked version of contexts for the standard first-child-next-sibling encoding of an unranked tree. Given a context C and a tree t, we denote by $C \circ t$ the tree obtained from the substitution of $\bullet$ in C by t. The composition $C \circ C'$ of two contexts C and C' is defined similarly and results again in a context. In the sequel, we denote by $\mathcal{C}_\Sigma^\downarrow$ the set of all top-down contexts over the alphabet Σ. Furthermore, all contexts considered in this chapter are top-down contexts in $\mathcal{C}_\Sigma^\downarrow$.

The advantage of restricting contexts to have the $\bullet$-node in "a leaf that has no right sibling" comes to light when we want to extend the transition function δ from states to contexts That is, as

is usual for finite automata on words we derive from the transition function δ of a top-down tree automaton $\mathcal{A}$ a transition function $\hat{\delta}$ on contexts. Formally, we define $\hat{\delta}$ as the partial function from $Q \times \mathcal{C}_\Sigma^{\downarrow}$ to Q by letting $\hat{\delta}(q, C) = q'$ iff the context C is accepted by the automaton $\mathcal{A}$ where the initial state is replaced by q and the transition function δ is extended in such a way that $\delta(q', \bullet) = (f, f)$, with f being a new final state. Notice that the extension of δ into $\hat{\delta}$ cannot be done if arbitrary unranked contexts (i.e. contexts where the $\bullet$-node is at an arbitrary leaf) are considered. Indeed, we cannot choose any accepting run for a context C whenever the $\bullet$-node is put at a leaf $\vec{i} \cdot j$ that has a right sibling $\vec{i} \cdot (j+1)$ in C. The state at position $\vec{i} \cdot (j+1)$ is determined by the state and label at position $\vec{i} \cdot j$, however the label at $\vec{i} \cdot j$ is $\bullet$.

Trees, contexts, and their serializations. It is known that trees, and more generally forests, can be represented by their serializations (XML Documents). Formally, given a finite alphabet Σ, we introduce a disjoint copy of Σ of the form $\bar{\Sigma} = \{\bar{a} \ : \ a \in \Sigma\}$. The elements in Σ represent the *opening tags* of the serializations, while the elements in $\bar{\Sigma}$ represent the *closing tags*. The *serialization* $\hat{t}$ of a tree t is defined recursively by $\hat{t} = \epsilon$ if t is empty, and by $\hat{t} = a \cdot \hat{t}_1 \cdot \ldots \cdot \hat{t}_n \cdot \bar{a}$ if $t = a(t_1, \ldots, t_n)$. The serialization of a forest is the concatenation of the serializations of its trees. Clearly, every serialization of a tree/forest produces a *well-matched* string over $\Sigma \uplus \bar{\Sigma}$ and vice-versa. For example, one can easily check that the serialization of t in Figure 5.1 is equal to:

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

Note that the serialization $\hat{C}$ of a context C is a word that contains a single occurrence of the substring $\bullet\bar{\bullet}$. We will denote by $\hat{C}^{\mathsf{prefix}}$ (resp. $\hat{C}^{\mathsf{suffix}}$) the prefix (resp. suffix) of $\hat{C}$ that ends immediately before the occurrence of $\bullet$ (resp. that starts immediately after the occurrence of $\bar{\bullet}$). Observe also that the composition of contexts with trees/contexts has analogous operations on serializations, that is, $\widehat{C \circ t} = \hat{C}^{\mathsf{prefix}} \cdot \hat{t} \cdot \hat{C}^{\mathsf{suffix}}$ and $\widehat{C \circ C'} = \hat{C}^{\mathsf{prefix}} \cdot \hat{C}' \cdot \hat{C}^{\mathsf{suffix}}$.

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

Tree edits over serializations. Recall the definition of the tree edit operations presented in Section 4.2 which are *insertion*, *deletion*, and *relabelling*. For the setting considered in this chapter, we want to edit trees by editing their serializations – by deleting and inserting tags – in such a way that the resulting operations implement tree edit operations. In order to reach this goal, we can code a sequence of string edits via another string, called alignment. Formally, given two strings $u \in \Sigma^*$ and $v \in \Delta^*$, an *alignment* of u and v is any string e over the alphabet $(\Sigma \uplus \{\epsilon\}) \times (\Delta \uplus \{\epsilon\})$ whose projection over the first (resp. second) component gives u (resp. v). The *cost* of an alignment e,

denoted $\mathsf{cost}(e)$, is the number of occurrences in e that are not of the form (a,a), with $a \in \Sigma$, nor of the form (ϵ,ϵ). Then the *edit-distance* between two strings $u \in \Sigma^*$ and $v \in \Delta^*$ can be defined as the minimum cost of all possible alignments of u and v [WF74]. As an example, the string $\binom{a}{a}\binom{b}{\epsilon}\binom{c}{c}\binom{d}{d}\binom{\epsilon}{e}$ is an alignment between the strings *abcd* and *acde* that achieves optimal cost 2 (hence $\mathsf{dist}(abcd, acde) = 2$).

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

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

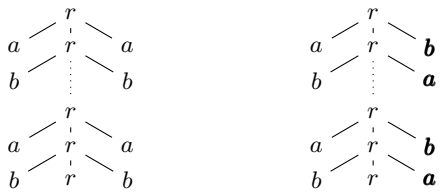
Example 14. *Given two unranked trees* $t = a(a(b),c)$ *and* $t' = a(a(c),b)$ *and their serializations* $\hat{t} = aab\bar{b}ac\bar{c}\bar{a}$ *and* $\hat{t}' = aac\bar{c}ab\bar{b}\bar{a}$*, the following are two possible alignments between* $\hat{t}$ *and* $\hat{t}'$*:*

$$e = \tbinom{a}{a}\tbinom{a}{a}\tbinom{b}{c}\tbinom{\bar{b}}{\bar{c}}\tbinom{\bar{a}}{\bar{a}}\tbinom{c}{b}\tbinom{\bar{c}}{\bar{b}}\tbinom{\bar{a}}{\bar{a}}$$
$$e' = \tbinom{a}{a}\tbinom{a}{a}\tbinom{\epsilon}{c}\tbinom{\epsilon}{\bar{c}}\tbinom{\epsilon}{\bar{a}}\tbinom{b}{b}\tbinom{\bar{b}}{\bar{b}}\tbinom{\bar{a}}{\bar{a}}\tbinom{c}{\epsilon}\tbinom{\bar{c}}{\epsilon}\tbinom{\bar{a}}{\epsilon}.$$

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

It is not difficult to see that, given two trees t and t', there is a sequence of tree edit operations turning t into t' and having cost N iff there is a tree edit alignment between the serializations $\hat{t}$ and $\hat{t}'$ having cost $2N$. Interestingly, the following example shows that the generic notion of alignment between serializations is too powerful to capture the costs of edit operations on trees, even up to multiplicative constants. There exist indeed families of trees on which the costs of alignments are uniformly bounded, while the costs of tree edit operations get arbitrary high [Aku06].

Example 15. *Consider pairs of trees of the same height and of the following forms (some labels are in bold to mark the differences between the left-hand and right-hand side):*



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

$$\tbinom{r}{r}\tbinom{a}{a}\tbinom{\bar{a}}{\bar{a}}\tbinom{r}{r}\tbinom{b}{b}\tbinom{\bar{b}}{\bar{b}}\dots\tbinom{r}{r}\tbinom{a}{a}\tbinom{\bar{a}}{\bar{a}}\tbinom{r}{r}\tbinom{b}{b}\tbinom{\bar{b}}{\bar{b}}\tbinom{r}{r}\cdot$$
$$\tbinom{\bar{r}}{\epsilon}\tbinom{b}{\epsilon}\tbinom{\bar{b}}{\epsilon}\tbinom{\bar{r}}{\bar{r}}\tbinom{a}{a}\tbinom{\bar{a}}{\bar{a}}\tbinom{\bar{r}}{\bar{r}}\dots\tbinom{b}{b}\tbinom{\bar{b}}{\bar{b}}\tbinom{\bar{r}}{\bar{r}}\tbinom{a}{a}\tbinom{\bar{a}}{\bar{a}}\tbinom{\bar{r}}{\bar{r}}\tbinom{\epsilon}{b}\tbinom{\epsilon}{\bar{b}}\tbinom{\epsilon}{\bar{r}}.$$

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

Transducers. A streaming repair process is a machine that consumes parts of the input and produces edits. We capture this with the notion of *transducer*. We recall the definition of transducer (defined previously in Section 2.1) for the sake of completeness. A (sub-sequential) transducer can be described by a tuple of the form $\mathcal{Z} = (\Sigma, \Delta, Z, \kappa, z_0, \Omega)$, where Σ is a finite alphabet for the input, Δ is a finite alphabet for the output, Z is a (possibly infinite) set of states, κ is a partial transition function from $Z \times (\Sigma \uplus \{\epsilon\})$ to $\Delta^* \times Z$, $z_0 \in Z$ is an initial state, and Ω is a final output function from Z to Δ^*. A run of $\mathcal{Z}$ consists of a sequence of transitions of the form

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

where $u_i \in \Sigma \uplus \{\epsilon\}$, $v_i \in \Delta^*$, $v_{n+1} = \Omega(z_n)$, 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 $\delta(z, a)$, with $a \in \Sigma$, and $\delta(z, \epsilon)$ are defined on the same state z.

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

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

where

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

We define the *cost* of a run ρ of $\mathcal{Z}$ as the cost of its canonical alignment $\mathsf{align}(\rho)$. Finally, we define the *cost* of $\mathcal{Z}$ over a word t by $\mathsf{cost}(t, \mathcal{Z}) = \sum_{i=1}^{k} \mathsf{dist}(a_i, b_i)$ where $\mathsf{align}(\rho) = \tbinom{a_1}{b_1}\cdots\tbinom{a_k}{b_k}$ and ρ is a run of $\mathcal{Z}$ over $\hat{t}$.

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

Top-down visibly pushdown automata. Repairing trees in a streaming fashion requires to process the serialization of a tree. Even though top-down tree automata are useful for reasoning about the structure of a tree language, they are not suitable for processing the serialization of trees, at least not in the presentation that we give in the main proofs of this chapter. In the last part of this section we give an alternative view of a top-down tree automata running on the serializations of trees, such that there exists a direct translation between both representations. The definitions introduced in the rest of this section are rather technical and are mostly used in the proof of the main characterization of this chapter. The reader could skip these definitions in a first read.

We represent top-down tree automata via a special form of *visibly pushdown automata* (VPA) [AM04, GNR08, KMV07]. Roughly speaking, VPA run over serializations of trees by using a restricted stack such that the depth of the stack is synchronized with the depth of the tree node being read. VPA are thus a convenient processing model for streaming validation and are equally expressive than arbitrary tree automata [KMV07]. However, we define below a subclass that is rich enough to capture deterministic top-down tree automata that we called top-down VPA. A *top-down VPA* is a tuple $\hat{\mathcal{A}} = (\Sigma \uplus \bar{\Sigma}, Q, \hat{\delta}, q_0, F)$, where

1. Σ (resp., $\bar{\Sigma}$) is a set of opening (resp. closing) tags,
2. Q is a finite set of control states,
3. $\hat{\delta} : Q \times (\Sigma \uplus \bar{\Sigma}) \to Q^*$ is a partial transition function,
4. $p_0 \in P$ is an initial control state, and
5. $F \subseteq P$ is a set of final control states.

We assume that $\hat{\delta}$ is restricted depending on the input letter in Σ or $\bar{\Sigma}$. Specifically, we suppose that $\hat{\delta}(q, a) \in Q^2$ whenever $a \in \Sigma$ and $\hat{\delta}(q, \bar{a}) = \epsilon$ whenever $\bar{a} \in \bar{\Sigma}$. This is the "visibly" restriction imposed to top-down VPAs [KMV07]. We use this restriction to maintain the equivalence between top-down VPA and top-down tree automata as it will be shown later.

A configuration of a VPA $\hat{\mathcal{A}}$ is described by a stack of control states (i.e. a non-empty string $\vec{q} \in Q^+$). The initial configuration of $\hat{\mathcal{A}}$ is the singleton stack q_0. A final configuration is any stack $\vec{q}$ such that $\mathsf{top}(\vec{q}) \in F$. A transition of $\hat{\mathcal{A}}$ on $a \in \Sigma \cup \bar{\Sigma}$ is of the form $q \cdot \vec{q} \xrightarrow{a} \hat{\delta}(q, a) \cdot \vec{q}$ whenever $\hat{\delta}(q, a)$ is defined (otherwise there is no such transition). Note that a transition of $\hat{\mathcal{A}}$ on a closing tag $\bar{a} \in \bar{\Sigma}$ is of the form $q \cdot \vec{q} \xrightarrow{\bar{a}} \vec{q}$ (called *pop*), provided that $\hat{\delta}(q, a)$ is defined. A run of $\hat{\mathcal{A}}$ on a word $w = w_1 \cdots w_n \in \Sigma \uplus \bar{\Sigma}$ is a sequence of transitions of the form

$$\vec{q}_0 \xrightarrow{w_1} \vec{q}_1 \xrightarrow{w_2} \cdots \xrightarrow{w_n} \vec{q}_n.$$

We say that $\hat{\mathcal{A}}$ accepts a (serialized) tree t if it admits run on $\hat{t}$ that starts from the initial singleton stack q_0 and ends in a final singleton stack. We denote by $\mathscr{L}(\hat{\mathcal{A}})$ the language of all (serialized) trees accepted by $\hat{\mathcal{A}}$.

Similar to word automata, we can extend the transition function of a top-down VPA $\hat{\mathcal{A}} = (\Sigma \uplus \bar{\Sigma}, Q, q_0, \hat{\delta}, F)$ to a partial mapping $\hat{\delta} : Q^+ \times (\Sigma \uplus \bar{\Sigma})^* \to Q^+$ such that $\hat{\delta}(\vec{q}, w) = \vec{q}'$ iff there exists a run of $\hat{\mathcal{A}}$ of the form $\vec{q} \xrightarrow{w} \vec{q}'$. As a particular case, for every tree t, we have that $\hat{t} \in \mathscr{L}(\hat{\mathcal{A}})$ iff $\hat{\delta}(q_0, \hat{t})$ is a singleton final stack.

One can easily seen the connection between deterministic top-down tree automata and top-down VPA. Indeed, given a deterministic top-down tree automata $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, define the top-down VPA $\hat{\mathcal{A}} = (\Sigma \uplus \bar{\Sigma}, Q, \hat{\delta}, q_0, F)$ such that $\delta(q, a) = (q_1, q_2)$ iff $\hat{\delta}(q, a) = q_1 \cdot q_2$ for every $a \in \Sigma$ and $\delta(q, \bar{a}) = \epsilon$ iff $q \in F$. It is straightforward to prove that $t \in \mathscr{L}(\mathcal{A})$ iff $\hat{t} \in \mathscr{L}(\mathcal{A})$. We assume this result without proof and we usually switch between the two formalism without explanation. We usually denote by $\mathcal{A}$ and δ for deterministic top-down tree automata and $\hat{\mathcal{A}}$ and $\hat{\delta}$ for top-down VPAs.

We further extend top-down VPA to its transducer version. Formally, a *top-down visibly pushdown transducer* is a tuple of the form $\hat{\mathcal{Z}} = (\Sigma \uplus \bar{\Sigma}, \Delta, Z, \hat{\kappa}, z_0, \Omega)$ where $\Sigma \uplus \bar{\Sigma}$ is a finite alphabet for the input, Δ is a finite alphabet for the output, Z is a finite set of control states, κ is a partial transition function from $Z \times (\Sigma \uplus \bar{\Sigma} \uplus \{\epsilon\})$ to $Z^* \times \Delta^*$, $z_0 \in Z$ is an initial state, and Ω is a final output function from Z to Δ^*. A run of $\mathcal{Z}$ consists of a sequence of transitions of the form

$$\vec{z}_0 \xrightarrow{u_1/v_1} \vec{z}_1 \xrightarrow{u_2/v_2} \dots \xrightarrow{u_n/v_n} \vec{z}_n \xrightarrow{\epsilon/v_{n+1}}$$

where $\vec{z}_i \in Z^+$, $\vec{z}_0 = z_0$, $u_i \in \Sigma \uplus \bar{\Sigma} \uplus \{\epsilon\}$, $v_i \in \Delta^*$, $v_{n+1} = \Omega(\mathsf{top}(\vec{z}_n))$, and for all $1 \leq i \leq n$ there exists $\vec{z}, \vec{z}' \in Z^*$ such that $\delta(\mathsf{top}(\vec{z}_{i-1}), u_i) = (\vec{z}, v_i)$ and $\vec{z}_i = \vec{z} \cdot \vec{z}'$. Given the above run, we say that $v = v_1 \cdot v_2 \cdot \ldots \cdot v_n \cdot v_{n+1}$ is the *output* of $\mathcal{Z}$ on *input* $u = u_1 \cdot u_2 \cdot \ldots \cdot u_n$. In order to guarantee that $\mathcal{Z}$ produces at most one output on each input, we forbid the possibility that both $\delta(z, a)$, with $a \in \Sigma$, and $\delta(z, \epsilon)$ are defined on the same state z. For the sake of simplification, we occasionally write $\hat{\mathcal{Z}}$ as a tuple $\hat{\mathcal{Z}} = (\Sigma \uplus \bar{\Sigma}, \Delta, Z, \hat{\kappa}, z_0, \iota, \Omega)$ such that $\iota \in \Delta^*$ is an initial output released before beginning to read the input.

5.2 Streaming bounded repairing of trees

We focus in this chapter on the *streaming bounded repair problem* for languages of unranked trees. Similar to the previous chapters, the setting is given by two languages R and T of unranked trees, called *restriction* and *target* languages. Trees in R (resp. T) are labelled over a finite alphabet Σ (resp. Δ), and they are encoded by serializations. The languages R and T are represented by means of deterministic top-down tree automata or DTDs.

The goal is to decide whether it is possible to 'repair' any tree $t \in R$ into a tree $t' \in T$, using a number of edits on serializations that is uniformly bounded by a constant. We are particularly

interested in repair strategies that are *streaming*, that is, repairs of serializations of trees that are produced in an online way, by means of tree edit transducers.

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 $\mathcal{Z} \in \mathsf{stream}(R,T)$ define the *worst-case aggregate cost* of $\mathcal{Z}$ as follows:

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

The *tree streaming aggregate cost* $\mathsf{cost}^{\mathsf{aggr}}_{\mathsf{stream}}(R,T)$ for two languages R and T is then defined as the minimum of $\mathsf{cost}^{\mathsf{aggr}}_{\mathcal{Z}}(R,T)$ taken over all tree edit transducers $\mathcal{Z} \in \mathsf{stream}(R,T)$. Note that we could define the *worst-case edit cost* and *tree streaming edit cost* of R and T similar to Section 2.2. However, our results in this chapter are restricted to the aggregate-case and we leave the edit-case for future work.

The *streaming bounded repair problem* consists of deciding, given two regular tree languages R and T whether there exists a tree edit transducer $\mathcal{Z}$ such that (i) on any input $\hat{t}$, with $t \in R$, $\mathcal{Z}$ outputs the serialization $\hat{t}'$ of some tree $t' \in T$, and (ii) the cost of the runs of $\mathcal{Z}$ are uniformly bounded by a constant. That is, $\mathsf{cost}^{\mathsf{aggr}}_{\mathsf{stream}}(R,T) < \infty$. Some examples of positive and negative instances of the streaming bounded repair problem follow.

Example 13 (Continued)**.** *Consider again the languages $R = \mathscr{L}(\mathcal{R})$ and $T = \mathscr{L}(\mathcal{T})$ described in our running example. It is possible to transform every tree $t \in R$ into a tree $t' \in T$ using just* 3 *edit operations: one first deletes the d-labelled child of the root, then relabels the b-labelled node into a, finally one inserts a new d-labelled node as a first child of the root, adopting the two chains of a-labelled nodes as sub-trees (see, for example, Figure 4.4). This strategy can be implemented at the level of serializations by a transducer that first copies the opening tag r from the input, then it produces an opening tag d and copies the portion $a \dots a\,\bar{a} \dots \bar{a}$ of the input; subsequently, it replaces the input string $d\,b$ with a, it copies another portion $a \dots a\,\bar{a} \dots \bar{a}$ of the input, it prepends $\bar{a}\,\bar{e}$ to the next incoming string $c\,\bar{c} \dots c\,\bar{c}$, and finally it erases the closing tag $\bar{d}$ and copies the last symbol $\bar{r}$.*

Example 16. *The following is a variant of an example from Chapter 2, which shows the difference between non-streaming edit strategies and streaming ones. Consider the language R of all trees of the form $r(x,c,\dots,c,y,\dots,y)$, with $x,y \in \{a,b\}$, and the language T of all trees of the form $r(x,c,\dots,c,x,\dots,x)$, with $x \in \{a,b\}$. A simple way to edit a tree of R into a tree of T is to replace the label x of the first child with the label y occurring the rightmost sibling. This strategy has uniformly bounded cost, but it cannot be implemented by a tree edit transducer of similar cost. Indeed, every transducer of bounded cost that parses a serialization of a tree of R has to commit to either preserving or modifying the label x of the first child before seeing the right siblings labelled by y. We have that the language R is* not *streaming repairable into T with uniformly bounded cost.*

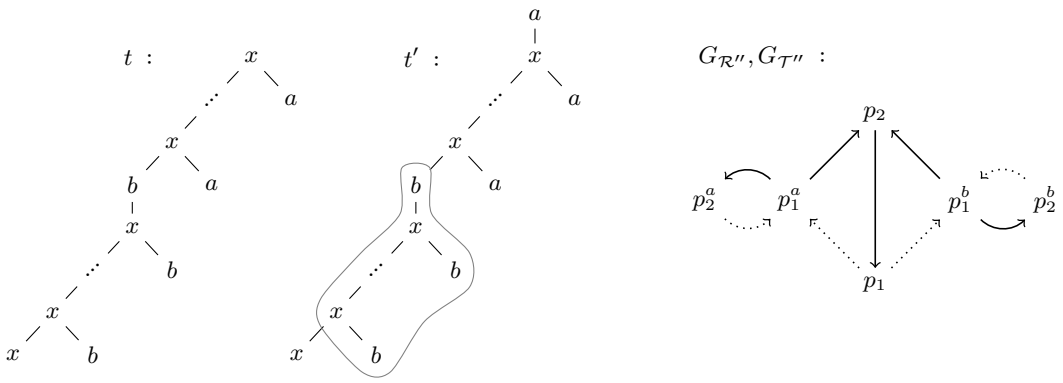


Figure 5.2: Example of trees accepted by $\mathcal{R}''$ and $\mathcal{T}''$, and their transition graphs.

In the next section, we provide a characterization to decide the streaming bounded repair problem for regular tree languages given by deterministic top-down tree automata. Interestingly, the techniques use so far in the previous chapters does not apply for the streaming bounded repair problem for the class of all regular tree languages. For this reason, we leave the general case (i.e. a characterization for all regular tree languages) as an open problem.

The following example shows that the approach of decomposing a tree into strongly connected components does not help for solving the streaming bounded repair problem.

Example 17. *Consider the set of states* $Q = \{p_0^x, p_0^a, p_1^a, p_2^a, p_0^b, p_1^b, p_2^b, p_1, p_2\}$ *and the following transition relations for a stepwise tree automaton over the alphabet* $\Sigma = \{x, a, b\}$*:*

$$
\begin{array}{llll}
\delta_0: & x & \to & p_0^x \\
 & a & \to & p_0^a \\
 & b & \to & p_0^b
\end{array}
\qquad
\begin{array}{llll}
\delta: & p_1 \mathbin{@} p_0^l & \to & p_1^l \quad l \in \{a, b\} \\
 & p_0^x \mathbin{@} p_1^l & \to & p_2^l \quad l \in \{a, b\} \\
 & p_2^l \mathbin{@} p_0^l & \to & p_1^l \quad l \in \{a, b\} \\
 & p_0^l \mathbin{@} p_1^l & \to & p_2 \quad l \in \{a, b\} \\
 & p_0^x \mathbin{@} p_2 & \to & p_1 \\
 & p_0^x \mathbin{@} p_0^x & \to & p_1
\end{array}
$$

Using these transitions functions, define the stepwise tree automata $\mathcal{R}'' = (\Sigma, Q, \delta, \delta_0, \{q_1^a, q_1^b\})$ *and* $\mathcal{T}'' = (\Sigma, Q, \delta, \delta_0, \{q_2\})$*. The left-hand side of Figure 5.2 shows two example of trees accepted by* $\mathcal{R}''$ *and* $\mathcal{T}''$*. Basically, trees in* $\mathscr{L}(\mathcal{R}'')$ *or* $\mathscr{L}(\mathcal{T}'')$ *repeat the context of the form pointed out by the gray line with either repeating a's or b's. The only difference between* $\mathcal{R}''$ *and* $\mathcal{T}''$ *is that trees in* $\mathcal{R}''$ *are labeled by* x *at the root and trees in* $\mathcal{T}$ *"complete" the tree with either an* a*-node or an* b*-node depending if the sub-tree below the root is either of the form* $x(\ldots, a)$ *or* $x(\ldots, b)$*, respectively.*

Note that $\mathcal{R}''$ and $\mathcal{T}''$ have the same structure of states and transition functions, and they only differ in the final states. Furthermore, graphs $G_{\mathcal{R}''}$ and $G_{\mathcal{T}''}$ consist of only one non-trivial SCC. The right-hand side of Figure 5.2 shows the graph $G_{\mathcal{R}''}$ and $G_{\mathcal{T}''}$ (note that $G_{\mathcal{R}''} = G_{\mathcal{T}''}$). We omit vertices p_0^x, p_0^a, and p_0^b from Figure 5.2 given that they are trivial. Further, one can show that

neither $\mathcal{R}''$ nor $\mathcal{T}''$ can be defined by a deterministic top-down tree automaton and then they are intrinsically "bottom-up" regular tree languages.

Clearly, trees in $\mathscr{L}(\mathcal{R}'')$ can be repaired into $\mathscr{L}(\mathcal{T}'')$ with a uniformly bounded number of edit operations. One only needs to insert either an a-node or an b-node over the root depending if the right-child of the root is label by a or b, respectively. We can also check that $\mathsf{cost}(\mathscr{L}(\mathcal{R}''), \mathcal{T}'') < \infty$ by using Theorem 4.3.6 given that $\mathcal{R}''$ and $\mathcal{T}''$ have the same internal structure. However, one can easily check that $\mathscr{L}(\mathcal{R}'')$ is not streaming bounded repairable into $\mathscr{L}(\mathcal{T}'')$. Indeed, the serialization of trees accepted by $\mathcal{R}''$ are of the form:

$$(x\, x \cdots x) \cdots (c\, \bar{c}\, \bar{x} \cdots c\, \bar{c}\, \bar{x}\, c\, \bar{c}\, \bar{x})$$

where $c \in \{a, b\}$. A streaming strategy $\mathcal{Z}$ needs to know in advance if the sequence ends either with a's or b's in order to insert the corresponding a- or b-label at the beginning. If $\mathcal{Z}$ inserts an a at the beginning, then the sequence could end with $(b\, \bar{b}\, \bar{x} \cdots b\, \bar{b}\, \bar{x}\, b\, \bar{b}\, \bar{x})$ and $\mathcal{Z}$ would have to edit an unbounded number of b-labels.

Example 17 shows that, in contrast with the characterizations given in Theorem 2.3.3 or 4.3.6, the SCC-structure of $\mathcal{R}''$ and $\mathcal{T}''$ cannot help us to decide whether $\mathscr{L}(\mathcal{R}'')$ is streaming bounded repairable into $\mathscr{L}(\mathcal{T}'')$. Therefore, we are doomed to fail if we try to develop a characterization for streaming bounded repairability based on the SCCs of the restriction and target languages. In the next section, we show that we can find a game-characterization based on SCCs if we restrict to regular tree languages specified by deterministic top-down tree automata.

5.3 Characterization of streaming repairability

In this section, we present an effective characterization of streaming bounded repairability for the languages recognized by deterministic top-down tree automata (thus including those languages definable by DTDs). This characterization combines ideas both from the solution of the streaming repair problem for regular word languages (Chapter 2) and from the solution of the non-streaming repair problem for regular languages of unranked trees (Chapter 4). For instance, in Chapter 2 streaming bounded repairability for regular word languages was characterized in terms of a simulation game over the directed acyclic graphs of the strongly connected components of the automata – similar concepts are used also in the present chapter. In Chapter 4 special conditions related to the behaviour of tree automata along the vertical (i.e. first-child) axis were taken into account – here we do something similar in the presence of 'vertical' contexts. To describe formally our characterization, we need to first introduce some definitions and notations.

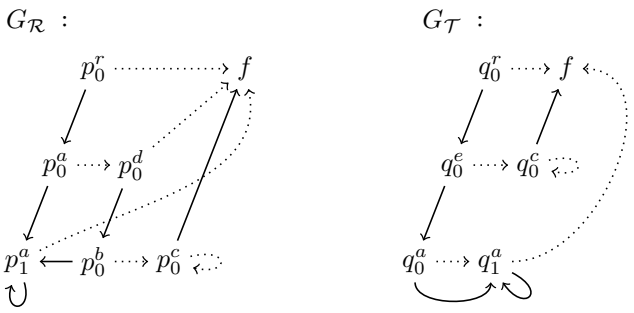


Figure 5.3: Transition graphs of two top-down tree automata.

5.3.1 Components of automata

It is easy to see that any deterministic top-down tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ can be equivalently represented by its transition graph $G_{\mathcal{A}} = (Q, E_h \cup E_v)$ (similar than for stepwise-tree automata), where the nodes are the control states of $\mathcal{A}$ and the edge relations E_v and E_h are defined as follows:

$$E_v = \big\{(q, q_1) \in Q \times Q \mid \exists q_2 \in Q.\ \exists a \in \Sigma.\ \delta(q, a) = (q_1, q_2)\big\},$$
$$E_h = \big\{(q, q_2) \in Q \times Q \mid \exists q_1 \in Q.\ \exists a \in \Sigma.\ \delta(q, a) = (q_1, q_2)\big\}.$$

Intuitively, the edges in E_v, called *vertical edges*, represent transitions of $\mathcal{A}$ from a given node of a tree to its first child, while the edges in E_h, called *horizontal edges*, represent transitions of $\mathcal{A}$ from a given node to its next sibling. Figure 5.3 depicts the transition graphs of the automata $\mathcal{R}$ and $\mathcal{T}$ of Example 13 (dotted arrows represent horizontal edges, solid arrows represent vertical edges).

Using the graph representation of an automaton $\mathcal{A}$, we can derive the notion of *strongly connected component* (or simply a *component*) of $\mathcal{A}$: this is a maximal set X of nodes of $G_{\mathcal{A}}$ such that for all $q, q' \in X$, there is a directed path from q to q' visiting only nodes in X and traversing edges in $E_v \cup E_h$. Observe that for every $q, q' \in X$, there is a context $C \in \mathcal{C}_\Sigma^\downarrow$ such that $\hat{\delta}(q, C) = q'$. Further, we denote by $\mathsf{SCC}(\mathcal{A})$ the set of all components of $\mathcal{A}$ and we distinguish between two types of components $X \in \mathsf{SCC}(\mathcal{A})$:

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

The left-hand side graph of Figure 5.3 contains six horizontal components and only one non-horizontal component (i.e. the one consisting of the single state p_1^a); similarly, the right-hand side graph of Figure 5.3 contains five horizontal components and one non-horizontal component.

Non-trivial components, namely, components that contain at least one edge, represent 'repeatable behaviours' of the automaton. Similar to the characterization in the previous chapters, these components have to be taken into account in the characterization of streaming bounded repairability

because they could generate arbitrary large fragments of trees (namely, contexts) that cannot be edited with uniformly bounded cost. To make this statement more precise, we associate with each component X of an automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ the *language of contexts realizable in* X:

$$\mathscr{L}(\mathcal{A} \mid X) \;=\; \left\{ C \in \mathcal{C}_\Sigma^{\downarrow} \;:\; \exists\, q, q' \in X.\;\; \hat{\delta}(q, C) = q' \right\}.$$

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

As an example, the automaton $\mathcal{R}$ of our running Example 13 contains one non-trivial horizontal component $\{p_0^c\}$, which realizes contexts of the form $c\,c \ldots c\,\bullet$. The other non-trivial component of $\mathcal{R}$ is $\{p_1^a\}$, which is non-horizontal and realizes contexts of the form $a(a(\ldots a(\bullet)\ldots))$.

5.3.2 Prefix-rewriting systems

To understand our characterization result, it is useful to think of a deterministic top-down tree automaton as a device that processes serializations of trees in a single-threaded left-to-right fashion, rather than in parallel. Informally, being able to perform a bounded repair from a restriction automaton $\mathcal{R}$ to a target automaton $\mathcal{T}$, one needs to respond to prefixes u of serializations of trees in $\mathscr{L}(\mathcal{R})$ by prefixes v of serializations of trees in $\mathscr{L}(\mathcal{T})$ in such a way that, at any point, if we take the component of the current state of $\mathcal{R}$ at the end of u (recall that $q \in Q$ is the *current state at the end of* u iff $\hat{\delta}(q_0, C) = q$ and $\hat{C}^{\mathsf{prefix}} = u$ for some context C), the language of contexts realized in this component is covered by the language of contexts realized in the component of the current state of $\mathcal{T}$ at the end of v. In this way, if the prefix u is repeatedly extended in a cyclic way – without exiting the component of the current state – the repair processor can respond by just copying the input symbols, incurring no cost. Of course, it is not feasible to look at all possible prefixes u of serializations of trees in $\mathcal{R}$. Thus our characterization of streaming bounded repairability is based on a sort of simulation game in which *abstractions* of runs of $\mathcal{R}$ are produced by one player, and are countered by abstractions of runs of $\mathcal{T}$, produced by the other player.

The abstractions are stacks of components, representing the states at the frontier of the portion of the tree that is represented by the prefix of the serialization. For example, extending a prefix u of a serialization with a new opening tag a induces a transition of $\mathcal{R}$ from the current state p to two states p_1 and p_2 (one associated with the new a-labelled child, the other associated with a forthcoming right sibling). This transition is abstracted at the level of components by a corresponding push-and-swap move that replaces the component of p at the top of the restriction stack with the components of p_1 and p_2. A key observation is that it is not necessary to mimic all transitions of the restriction automaton, but only those that exit the current component and reach new components with both

successor states. This will keep the length of the plays in the simulation game bounded, allowing us to determine the winner effectively.

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:

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

where X, X_1, X_2 denote single components of $\mathcal{R}$. Note that $\vec{x} \overset{\mathcal{R}}{\mapsto} \vec{x}'$ implies either $|\vec{x}| = |\vec{x}'| + 1$ or $|\vec{x}| + 1 = |\vec{x}'|$. Moreover, according to the above definition, the component X at the top of the stack cannot be rewritten into a copy of it (due to the condition $X_1 \neq X \wedge X_2 \neq X$). That is, the prefix rewriting system $\overset{\mathcal{R}}{\mapsto}$ is non-recursive and every path in the graph defined by $\overset{\mathcal{R}}{\mapsto}$ is finite.

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

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

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

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

Example 18. *Consider the automata $\mathcal{R}$ and $\mathcal{T}$ of our running example (see also Figure 5.3 for a quick reference of the transitions). The following are two valid derivations of the prefix-rewriting systems $\overset{\mathcal{R}}{\mapsto}$ and $\overset{\mathcal{T}}{\mapsto^*}$:*

$$\{p_0^r\} \overset{\mathcal{R}}{\mapsto} \{p_0^a\}\{f\} \overset{\mathcal{R}}{\mapsto} \{p_1^a\}\{p_0^d\}\{f\} \overset{\mathcal{R}}{\mapsto} \{p_0^d\}\{f\}$$
$$\{q_0^r\} \overset{\mathcal{T}}{\mapsto^*} \{q_0^a\}\{q_0^c\}\{f\} \overset{\mathcal{T}}{\mapsto^*} \{q_1^a\}\{q_1^a\}\{q_0^c\}\{f\} \overset{\mathcal{T}}{\mapsto^*} \{f\}.$$

5.3.3 The simulation game

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

To explain the general idea we first consider the simpler case where all components of the restriction automaton $\mathcal{R}$ are horizontal. In this case, the simulation game takes place between two players, called *Generator* and *Repairer*, who control two stacks $\vec{x} \in \mathsf{SCC}(\mathcal{R})^*$ and $\vec{y} \in \mathsf{SCC}(\mathcal{T})^*$ using the prefix-rewriting relations $\overset{\mathcal{R}}{\mapsto}$ and $\overset{\mathcal{T}}{\mapsto^*}$, respectively. The game starts with the initial singleton stacks X_0 and Y_0, where X_0 is the component of the initial state of $\mathcal{R}$ and Y_0 is the component of the initial state of $\mathcal{T}$. Repairer moves first by applying to his stack Y_0 a sequence of prefix-rewriting rules satisfying $\overset{\mathcal{T}}{\mapsto^*}$ (this corresponds to the fact that the repair processor is allowed to insert some initial prefix of the output, prior to any input being received). Generator responds by applying to his stack X_0 a single prefix-rewriting rule satisfying $\overset{\mathcal{R}}{\mapsto}$. Then the game continues in a similar way from the new pair of stacks. Some invariants have to be enforced. Every time Repairer moves, he has to guarantee that the language $\mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}))$ of contexts realizable in the top component of his stack $\vec{y}$ contains the language $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x}))$ of contexts realizable in the top component of the stack $\vec{x}$ of Generator. We will see later in Section 5.4 and 5.5 how this covering property between languages of components eases the repair process. Eventually, one of the two players will not be able to move, in which case the other player wins.

In order to correctly characterize streaming bounded repairability in the presence of non-horizontal components of $\mathcal{R}$, we need to consider a variant of the simulation game where a special separator symbol $\triangleleft$ is prepended to the non-horizontal components of the stacks. For the sake of presentation, it is convenient to describe the variant of the simulation game by introducing a third player, called *Referee*, who handles the occurrences of the separator symbol $\triangleleft$ in the two stacks. The game goes as before by alternating between moves of Repairer and moves of Generator. However, if after a move of Repairer the element at the top of the stack of Generator happens to be a non-horizontal component, then Referee comes into play: he inserts the separator symbol $\triangleleft$ just below the top components of the stacks of Generator and Repairer and he passes the turn to Generator. From there after, neither Generator nor Repairer are allowed to modify the parts of their stacks that

are hidden under a separator. If after a move of Generator the top element of his stack becomes $\triangleleft$, then Referee comes again into play: he removes $\triangleleft$ from the top of the stack of Generator, he pops from the stack of Repairer the top-most separator and all elements above it, and he finally passes the turn to Repairer. We remark that in the above formulation of the game, Referee cannot choose his moves, as these are always determined by the current configuration of the game. This makes the game equivalent to a classical turn-based two-player reachability game, whose winner is known to be determined.

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

Definition 5.3.1. *Let $\mathcal{R}$ and $\mathcal{T}$ be two top-down tree automata and let $\vec{x}, \vec{x}', \vec{x}''$ (resp. $\vec{y}, \vec{y}', \vec{y}''$) denote generic sequences over $\mathsf{SCC}(\mathcal{R}) \uplus \{\triangleleft\}$ (resp. $\mathsf{SCC}(\mathcal{T}) \uplus \{\triangleleft\}$). The arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ for the simulation game is defined as follows:*

- *the positions owned by Generator are the pairs $[\![\vec{x}, \vec{y}]\!]$, where $\mathsf{top}(\vec{x})$ and $\mathsf{top}(\vec{y})$ are components such that $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}))$, and where $\mathsf{top}(\mathsf{tail}(\vec{x})) = \triangleleft$ whenever $\mathsf{top}(\vec{x})$ is non-horizontal;*
- *the positions owned by Repairer are the pairs $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle$, where $\mathsf{top}(\vec{x}) \neq \triangleleft$;*
- *the positions owned by Referee are the pairs $[\![\vec{x}, \vec{y}]\!]$, where $\mathsf{top}(\vec{x})$ and $\mathsf{top}(\vec{y})$ are non-horizontal components, $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}))$, and $\mathsf{top}(\mathsf{tail}(\vec{x})) \neq \triangleleft$, as well as the pairs $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle$, where $\mathsf{top}(\vec{x}) = \triangleleft$;*
- *the initial position is the pair $\langle\!\langle \vec{x}_0, \vec{y}_0 \rangle\!\rangle$, which is owned by Repairer, where $\vec{x}_0$ (resp. $\vec{y}_0$) is the singleton stack that consists of the component of the initial state of $\mathcal{R}$ (resp. $\mathcal{T}$);*
- *the possible moves for Generator are of the form $[\![\vec{x} \cdot \vec{x}'', \vec{y}]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle \vec{x}' \cdot \vec{x}'', \vec{y} \rangle\!\rangle$, where $\vec{x} \overset{\mathcal{R}}{\mapsto} \vec{x}'$ is a single prefix-rewriting rule associated with $\mathcal{R}$ (in particular, $\triangleleft$ occurs neither in $\vec{x}$ nor in $\vec{x}'$);*
- *the possible moves for Repairer are of the form $\langle\!\langle \vec{x}, \vec{y} \cdot \vec{y}'' \rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}, \vec{y}' \cdot \vec{y}'']\!]$, where $\vec{y} \overset{\mathcal{T}}{\mapsto}{}^{*} \vec{y}'$ is a sequence of prefix-rewriting rules associated with $\mathcal{T}$ (in particular, $\triangleleft$ occurs neither in $\vec{y}$ nor in $\vec{y}'$);*
- *the possible moves for Referee are of the form $[\![X \cdot \vec{x}'', Y \cdot \vec{y}'']\!] \overset{\mathsf{Ref}}{\mapsto} [\![X \cdot \triangleleft \cdot \vec{x}'', Y \cdot \triangleleft \cdot \vec{y}'']\!]$, where X is non-horizontal, and those of the form $\langle\!\langle \triangleleft \cdot \vec{x}, \vec{y} \cdot \triangleleft \cdot \vec{y}'' \rangle\!\rangle \overset{\mathsf{Ref}}{\mapsto} \langle\!\langle \vec{x}, \vec{y}'' \rangle\!\rangle$, where $\triangleleft$ does not occur in $\vec{y}$.*

We observe that all plays that could possibly arise from the simulation game over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ are finite: this is because each position of $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ is visited at most once during a play and the set of

Figure 5.4: Examples of unranked trees in $\mathscr{L}(\mathcal{R}')$ and $\mathscr{L}(\mathcal{T}')$.

all reachable positions is finite, due to the restriction on the moves of Generator. Indeed the stacks that could be derived from the prefix-rewriting system $\overset{\mathcal{R}}{\mapsto}$ have length at most $|\mathsf{SCC}(\mathcal{R})|$. This allows us to define the winner of a play as the last player who moved (this must be either Generator or Repairer).

Example 19. *We continue our running example by describing a prefix of possible play over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ (to save space and improve readability, we write the pairs for the positions of the arena vertically):*

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

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

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

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

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

Example 20. *Let $\mathcal{R}'$ and $\mathcal{T}'$ be the deterministic top-down tree automata with the following transitions (p_0 and q_0 are the initial states, all other states are final):*

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

Figure 5.4 shows examples of trees in $\mathscr{L}(\mathcal{R}')$ and in $\mathscr{L}(\mathcal{T}')$. Clearly, $\mathscr{L}(\mathcal{R}')$ is not bounded repairable into $\mathscr{L}(\mathcal{T}')$ (not even with an offline repair strategy). Accordingly, Repairer loses the simulation game over $\mathcal{G}_{\mathcal{R}',\mathcal{T}'}$ in the presence of separator symbols: Generator has a winning strategy that consists of first reaching the restriction stack $\{p_1\}\triangleleft\{f\}$, forcing Repairer to respond with a target stack of the form $\{q_2\}\triangleleft\ldots\{q_3\}\{f\}$, and later rewriting his stack to $\{p_2\}\triangleleft\{f\}$, thus leading to a losing position for Repairer (the component $\{p_2\}$ of $\mathcal{R}'$ is not covered by any component of $\mathcal{T}'$ that is reachable from $\{q_2\}$).

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

We are now ready to state our main characterization result:

Theorem 5.3.2. *Given a pair of deterministic top-down tree automata* $\mathcal{R}$ *and* $\mathcal{T}$*, there exists a streaming repair strategy from* $\mathscr{L}(\mathcal{R})$ *to* $\mathscr{L}(\mathcal{T})$ *with uniformly bounded cost iff Repairer has a strategy to win the simulation game over* $\mathcal{G}_{\mathcal{R},\mathcal{T}}$*.*

The effectiveness of the above characterization is discussed in Section 5.6, together with tight complexity bounds for the streaming bounded repairability problem. In the following sections (Section 5.4 and 5.5) we show the proof of Theorem 5.3.2 in detail. First, in Section 5.4 we show that if Repairer has a strategy to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, then we can use this strategy to define a streaming transducer with bounded cost that repairs every serialized tree from $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$. From this proof, one can effectively construct a tree edit transducer that repairs $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with uniformly bounded cost whenever Repairer wins the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Later, in Section 5.5 we prove the other direction of Theorem 5.3.2. We show specifically that if Generator has a winning strategy to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, then every streaming repair strategy from $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ has to incur with an unbounded cost.

5.4 From simulation games to repairs

This section is devoted to the proof of the if-direction of Theorem 5.3.2. We assume that Repairer wins the game over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, and we show 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 reachability game: Referee cannot really choose his moves, and one of the remaining players, either Generator or Repairer, wins by reaching a position of the arena where the opponent cannot move. It is well known that reachability games are positionally determined (see [GTW03] for a survey), which means that one of the two players can win using a strategy defined only on the basis of the current position of the arena. Thus, we assume that Repairer has a positional strategy for winning the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. This strategy is described 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 $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}', \vec{y}']\!]$. We use W to construct a transducer $\mathcal{Z}$ that implements a repair processor from $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with uniformly bounded cost.

It is convenient to construct $\mathcal{Z}$ incrementally from W, namely, as a cascade composition of fairly simple transducers $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$. Intuitively, the first transducer $\mathcal{Z}_1$ decomposes the input tree t into a uniformly bounded number of contexts, each one realizable within a single component of

$\mathcal{R}$ which may require deleting a small number of nodes in t. The output of $\mathcal{Z}_1$ is formed in such a way that one can easily extract a sequence of prefix-rewriting steps of the form $X \mapsto X_1X_2$ or $X \mapsto \epsilon$. The second transducer $\mathcal{Z}_2$ receives the output of $\mathcal{Z}_1$ and computes the responses of Repairer to the moves of Generator induced by the output of $\mathcal{Z}_1$. For this purpose, we exploit the existence of a winning strategy W for Repairer. Furthermore, $\mathcal{Z}_2$ annotates the contexts of t with partial runs of $\mathcal{T}$. Finally, the third transducer $\mathcal{Z}_3$ receives the output of $\mathcal{Z}_2$ and glues the contexts of t decorated with runs of $\mathcal{T}$ in such a way that forms a complete run over a tree $t' \in \mathscr{L}(\mathcal{T})$. This requires inserting additional contexts of uniformly bounded size, which can be extracted from the moves of Repairer provided by $\mathcal{Z}_2$.

Below, we explain each sub-transducer in a different subsection and in the last subsection we show how they work together. Given that this proof is long and technical, we give here a brief overview of all its stages:

- In Section 5.4.1, we start by defining $\mathcal{Z}_1$ and its output. We first introduce the concept of *$\mathcal{R}$-decomposition tree* of a tree $t \in \mathscr{L}(\mathcal{R})$. The idea is to decompose the serialization $\hat{t}$ into a uniformly bounded number of contexts, each one realized within a component of $\mathcal{R}$. Here, we use the top-down determinism of $\mathcal{R}$ to generate this decomposition in a streaming fashion that will be consumed by the next transducer $\mathcal{Z}_2$. Concurrently, $\mathcal{Z}_1$ extracts from $\hat{t}$ a sequence of prefix-rewriting steps of the stack controlled by Generator in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. This is a valid sequence of moves of Generator and, furthermore, represents a valid play of Generator over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Both, the $\mathcal{R}$-decomposition and the sequence of prefix-rewriting steps, can be obtained from t by simulating the unique run of $\mathcal{R}$ over t and by extracting maximal contexts realized within components of $\mathcal{R}$. This process is similar to the one given in the proof of Lemma 4.3.2 but implemented by a streaming transducer. Note that an $\mathcal{R}$-decomposition of t could be seen analogous to a primitive synopsis tree (Chapter 4) with some additional information that will be used by the next transducers.

- In Section 5.4.2, we give a detail definition of $\mathcal{Z}_2$. Roughly speaking, the first task of $\mathcal{Z}_2$ is to transform the sequence of moves of Generator obtained by $\mathcal{Z}_1$ into moves of Repairer. Given that moves of Generator are produced by $\mathcal{Z}_1$ in a streaming fashion, these moves can be assumed to be a valid play of Generator over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. The duty of $\mathcal{Z}_2$ is to reply to these moves with moves of Repairer by using the strategy W and by preserving the restriction that the top element of Generator's stack $\vec{x}$ is covered by the top element of Repairer's stack $\vec{y}$, namely, $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}))$. The context C from the $\mathcal{R}$-decomposition of t obtained just after a move of Generator and serialized by $\mathcal{Z}_1$ will satisfy $C \in \mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x}))$. Thus, the second task of $\mathcal{Z}_2$ is to run $\mathcal{T}$ over each context C and to find two states $q, q' \in \mathsf{top}(\vec{y})$ such that $\delta(q, C) = q'$ where δ is the transition function of $\mathcal{T}$. Furthermore, $\mathcal{Z}_2$ decorates each

serialized context C with states $q, q' \in \mathsf{top}(\vec{y})$ such that $\delta(q, C) = q'$. These states will be used by the last transducer to glue everything together into a tree in $\mathscr{L}(\mathcal{T})$.

- In Section 5.4.3, we finally turn to the last stage of the processing line, namely, the transducer $\mathcal{Z}_3$ that repairs the output of $\mathcal{Z}_2$ into $\mathscr{L}(\mathcal{T})$. The main idea is that every output w of $\mathcal{Z}_2$ can be turned into a concrete tree in $\mathscr{L}(\mathcal{T})$ by glueing together all contexts inside w. To achieve this it might be necessary to produce additional contexts of small size (bounded by $|\mathcal{T}|$) that connects states q and q' added to each context C by $\mathcal{Z}_2$. This step can be seen similar to the process explained in the proof of Lemma 4.3.4 (Chapter 4). There, components were glued together by following the structure of a basic synopsis tree. Here, the whole synopsis tree is not known in advance and the only guides are the moves of Repairer that are assumed to follow a winning strategy against Generator.

- Finally, in Section 5.4.4 we show the correctness of the composition of $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$. We argue that for every tree $t \in \mathscr{L}(\mathcal{R})$ it holds that $\mathcal{Z}_3 \circ \mathcal{Z}_2 \circ \mathcal{Z}_1(\hat{t}) \in \mathscr{L}(\mathcal{T})$ and together this composition performs a bounded number of edit operations. The statement that $\mathcal{Z}_3 \circ \mathcal{Z}_2 \circ \mathcal{Z}_1$ is a tree edit transducers will be clear from the construction of $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$.

During this proof, we fix two top-down tree automata $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$ and $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ that specify the restriction and target language, respectively. Recall that we denote by $\vec{x}$ (with upper or lower scripts) an arbitrary stack for Generator in $(\mathsf{SCC}(\mathcal{R}) \cup \{\triangleleft\})^*$, by $\vec{y}$ a Repairer stack in $(\mathsf{SCC}(\mathcal{T}) \cup \{\triangleleft\})^*$, and by X and Y a component in $\mathsf{SCC}(\mathcal{R})$ and $\mathsf{SCC}(\mathcal{T})$, respectively. The initial strongly connected component of $\mathcal{R}$ ($\mathcal{T}$) is denoted by X_0 (Y_0 resp.). We denote by $[q]_{\mathcal{A}}$ the equivalence class or strongly connected component of a state q in $\mathcal{A}$, or simply by $[q]$ if $\mathcal{A}$ is clear from the context. Finally, in some sections of the proof we need to also consider unranked forests. We denote by $\mathcal{F}_\Sigma$ the set of all unranked forests over Σ.

5.4.1 Decomposing the input into contexts

As was explained in the introduction of this section, the main purpose of $\mathcal{Z}_1$ within the process of repairing the input tree $t \in \mathscr{L}(\mathcal{R})$ is to run the restriction automata $\mathcal{R}$ over $\hat{t}$ and to use this run to decompose $\hat{t}$ into contexts, while deleting portions of the tree. These contexts will represent the parts of the tree that will remain untouched in the final output. Further, $\mathcal{Z}_1$ extracts from this run moves of Generator over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ that are passed to the next transducer $\mathcal{Z}_2$. We devote this subsection to explaining all these steps of $\mathcal{Z}_1$ in detail.

First, we explain how we decompose a tree t into contexts. Consider an unranked tree t and any node $\bar{n} \in \mathsf{nodes}(t)$. Intuitively, $\bar{n}$ decomposes t into three disjoint parts: (1) the context C that is above $\bar{n}$, (2) the forest of children $h \in \mathcal{F}_\Sigma$ under $\bar{n}$, and (3) the forest of right-siblings $h' \in \mathcal{F}_\Sigma$ at the right of $\bar{n}$. For example, for a tree $t = a(b, c(d, d), b, b)$ the node c decomposes t into three disjoint

parts: the context $a(b, \bullet)$, the children (d, d), and the right-siblings (b, b). We can always recover the initial tree t by hanging h below $\bar{n}$, concatenating this result with h', and composing it with C, that is:

$$t \;=\; C \circ (a(h) \cdot h')$$

where $a = t(\bar{n})$. We can continue this process recursively by picking a new node over h or h', and decomposing h or h' until we reach a leaf without right-siblings. The representation of this recursive procedure is what we call a decomposition tree of t. Formally, a *decomposition tree* D is a binary tree such that its internal nodes are labelled in $\mathcal{C}_\Sigma^\downarrow \times \Sigma$ and its leaves are labelled in $\mathcal{C}_\Sigma^\downarrow$ (where $\mathcal{C}_\Sigma^\downarrow$ is the set of all contexts over Σ). We write the label of an internal node $\bar{n} \in \mathsf{nodes}(D)$ by $D(\bar{n}) = [C, a]$ where we call C the *contexts* and a the *branching* label of $\bar{n}$ in D. We usually write a decomposition tree D inductively by $D = [C, a](D_1, D_2)$ where $[C, a]$ is the label of the root and D_1, D_2 are decomposition sub-trees of D. In order to reconstruct the tree represented by a decomposition tree, we define recursively a tree, or more precise a forest, $\dot{D}$ over Σ represented by a decomposition tree D as follow:

- $\dot{D} = C \circ \epsilon$ whenever $D = C$ is a leaf, and
- $\dot{D} = C \circ (a(\dot{D}_1) \cdot \dot{D}_2)$ whenever $D = [C, a](D_1, D_2)$

We say that D is a decomposition tree of t if D is a decomposition tree and $t = \dot{D}$. Notice that each branching label in D represents a node of t. We say that a node of t coming from a branching label in D is a *branching node* with respect to D.

Clearly, there exists many different decomposition trees for each unranked tree in $\mathcal{T}_\Sigma$. However, we are interested in a special kind of decomposition tree given a top-down tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$. Let $D = [C, a](D_1, D_2)$ be a decomposition tree and $q \in Q$. Further, consider the states $q_1, q_2 \in Q$ such that $\delta(\hat{\delta}(q, C), a) = (q_1, q_2)$ where $\hat{\delta} : Q \times \mathcal{C}_\Sigma^\downarrow \rightharpoonup Q$ is the extension of δ from Σ to $\mathcal{C}_\Sigma^\downarrow$. Then we say that D is an $(\mathcal{A}, q)$-decomposition tree iff:

- $\hat{\delta}(q, C) \in [q]_\mathcal{A}$,
- $[q_1]_\mathcal{A} \neq [q]_\mathcal{A} \neq [q_2]_\mathcal{A}$ whenever D_1 and D_2 are non-empty, and
- D_1 (D_2) is a $(\mathcal{A}, q_1)$-decomposition tree (resp. $(\mathcal{A}, q_2)$-decomposition tree).

Furthermore, we say that D is an $\mathcal{A}$-decomposition tree of t if D is a $(\mathcal{A}, q_0)$-decomposition tree where q_0 is the initial state of $\mathcal{A}$. Intuitively, an $\mathcal{A}$-decomposition of t is a maximal decomposition with respect to the components of $\mathcal{A}$.

For a tree $t \in \mathscr{L}(\mathcal{R})$ and an $\mathcal{A}$-decomposition tree D of t, we want to indicate which are the particular states at the branching nodes of t given the unique run of $\mathcal{A}$ over t. Formally, a run ρ of

$\mathcal{A}$ over an $\mathcal{A}$-decomposition tree D is a function $\rho : \mathsf{nodes}(D) \to Q$ such that $\rho(\epsilon) = q_0$ and for every internal node $\bar{n}$ of D it holds that:

$$\delta(\hat{\delta}(\rho(\bar{n}), C), a) \;=\; (\rho(\bar{n}\cdot 1), \rho(\bar{n}\cdot 2))$$

where $D(\bar{n}) = [C, a]$. Informally, ρ annotates D with the states that are reached by $\mathcal{A}$ at the beginning of each node in the decomposition. Since $\mathcal{A}$ is deterministic, for every $\mathcal{A}$-decomposition tree D of $t \in \mathscr{L}(\mathcal{A})$ there always exists a unique run of $\mathcal{A}$ over D. Furthermore, it is straightforward to prove from the definition that $C \in \mathscr{L}(\mathcal{A} \mid [\rho(\bar{n})]_{\mathcal{A}})$ for every node $\bar{n} \in \mathsf{nodes}(D)$ and $D(\bar{n}) = [C, a]$ where $\mathscr{L}(\mathcal{A} \mid [\rho(\bar{n})]_{\mathcal{A}})$ is the language of contexts of the strongly connected component of $\rho(\bar{n})$ in $\mathcal{A}$.

We have all the ingredients to define the output of the first transducer $\mathcal{Z}_1$. The output of $\mathcal{Z}_1$ on a tree t consists of three main elements: (1) the serialization of an $\mathcal{R}$-decomposition tree D of t, (2) a run ρ of $\mathcal{R}$ over D, and (3) moves of Generator over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ extracted from D and ρ. Before giving the formal definition of the output of $\mathcal{Z}_1$, we define the serialization of these last elements; that is, the string encoding of the rules of Generator and Repairer over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Let $\overset{\mathcal{R}}{\mapsto} \subseteq \mathsf{SCC}(\mathcal{R})^* \times \mathsf{SCC}(\mathcal{R})^*$ be the prefix rewriting system between stacks of strongly connected components of $\mathcal{R}$ defined in Section 5.3.2 by:

$$\begin{array}{lll} X \cdot \vec{x} \;\overset{\mathcal{R}}{\mapsto}\; X_1X_2 \cdot \vec{x} & \text{iff} & X_1 \neq X \neq X_2 \;\wedge \\ & & \exists\, p \in X, p_1 \in X_1, p_2 \in X_2, a \in \Sigma \\ & & \delta(p,a) = (p_1, p_2) \\ X \cdot \vec{x} \;\overset{\mathcal{R}}{\mapsto}\; \vec{x} & & \text{always.} \end{array}$$

For each transition as above we associate an *atomic rule* of the form $X \mapsto X_1X_2$ whenever $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} X_1X_2 \cdot \vec{x}$, or $X \mapsto \epsilon$ whenever $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} \vec{x}$ for some $\vec{x} \in \mathsf{SCC}(\mathcal{R})^*$. We denote by $\mathsf{moves}(\mathcal{R})$ the set of all atomic rules associated with transitions in $\mathcal{R}$. The set of atomic rules $\mathsf{moves}(\mathcal{T})$ are defined analogously for $\mathcal{T}$ and its prefix rewriting system $\overset{\mathcal{T}}{\mapsto}{}^*$. Notice that $\mathsf{moves}(\mathcal{R})$ and $\mathsf{moves}(\mathcal{T})$ are finite sets. Clearly, any atomic rule of the form $X \mapsto \vec{x}$ defines the next move of Generator from a stack $X \cdot \vec{x}' \in \mathsf{SCC}(\mathcal{R})^*$. Then we write $\vec{x} \overset{m}{\mapsto} \vec{x}'$ whenever the atomic rule $m \in \mathsf{moves}(\mathcal{R})$ defines a transition $\vec{x} \overset{\mathcal{R}}{\mapsto} \vec{x'}$. We extend this notation to a sequence of atomic rules $\vec{m} = m_1 \dots m_n$ and we write $\vec{x} \overset{\vec{m}}{\mapsto} \vec{x}'$ if there exists $\vec{x}_1, \dots, \vec{x}_{n-1} \in \mathsf{SCC}(\mathcal{R})^*$ such that $\vec{x} \overset{m_1}{\mapsto} \vec{x}_1 \overset{m_2}{\mapsto} \cdots \overset{m_n}{\mapsto} \vec{x}'$. In other words, we encode sequences of moves of Generator by words over the finite alphabet $\mathsf{moves}(\mathcal{R})$. Finally, atomic rules in $\mathcal{R}$ are naturally extended to stacks in $(\mathsf{SCC}(\mathcal{R}) \cup \{\triangleleft\})^*$. We define $X \cdot \triangleleft \cdot \vec{x} \overset{m}{\mapsto} \vec{x}$ whenever $m = X \mapsto \epsilon$, or $X \cdot \vec{x} \overset{m}{\mapsto} X_1 \cdot \triangleleft \cdot X_2 \cdot \vec{x}$ whenever $m = X \mapsto X_1X_2$ and X_1 is non-horizontal. As we will show next, the output of $\mathcal{Z}_1$ over a serialized tree in $\mathscr{L}(\mathcal{R})$ contains a complete sequence of atomic rules played by Generator.

Let $\Gamma = \Sigma \uplus \bar{\Sigma} \uplus \mathsf{moves}(\mathcal{R}) \uplus P \uplus \{\langle\, , /, \,\rangle\}$ be the output alphabet of $\mathcal{Z}_1$. Namely, the output alphabet Γ is the disjoint union of letters of $\Sigma \uplus \bar{\Sigma}$, atomic rules in $\mathsf{moves}(\mathcal{R})$, states P of $\mathcal{R}$, and a

set of separators $\{\langle\, , /, \,\rangle\}$. These separators are necessary to output macro-tags of the form:

$$\langle\, p\ \hat{C}^{\mathsf{prefix}}\,\rangle, \langle /\, \hat{C}^{\mathsf{suffix}}\,\rangle, \text{ and } \langle\, p\ \hat{C}^{\mathsf{prefix}}\, /\rangle$$

where $p \in P$ and $C \in \mathcal{C}_\Sigma^{\downarrow}$. Notice that we use an XML-encoding style for the output of $\mathcal{Z}_1$, where $\langle\, a\,\rangle$ denotes an opening tag, $\langle /\, a\,\rangle$ a closing tag, and $\langle\, a\, /\rangle$ a leaf tag. For the sake of abstraction for transducers $\mathcal{Z}_2$ and $\mathcal{Z}_3$, we consider these macro-tags as letters of an infinite alphabet. Formally, we define the alphabet of opening macro-tags $\Phi_1^{\mathsf{open}} = \{\langle\, p{\cdot}\hat{C}^{\mathsf{prefix}}\,\rangle \mid p \in P, C \in \mathcal{C}_\Sigma^{\downarrow}\}$, the alphabet of closing macro-tags $\Phi_1^{\mathsf{close}} = \{\langle /\, \hat{C}^{\mathsf{suffix}}\,\rangle \mid C \in \mathcal{C}_\Sigma^{\downarrow}\}$, and the alphabet of leaf macro-tags $\Phi_1^{\mathsf{leaf}} = \{\langle\, p \cdot \hat{C}^{\mathsf{prefix}}\, /\rangle \mid p \in P, C \in \mathcal{C}_\Sigma^{\downarrow}\}$. We denote the alphabet of all macro-tags by $\Phi_1^{\mathsf{tags}} = \Phi_1^{\mathsf{open}} \cup \Phi_1^{\mathsf{close}} \cup \Phi_1^{\mathsf{leaf}}$. Note that an opening macro-tag $\langle\, p\ \hat{C}^{\mathsf{prefix}}\,\rangle$ contains the prefix of a context (plus a state p) and a closing macro-tag $\langle /\, \hat{C}^{\mathsf{suffix}}\,\rangle$ contains its suffix. Leaf macro-tags $\langle\, p\ \hat{C}^{\mathsf{prefix}}\, /\rangle$ are for contexts coming from horizontal components where its suffix $\hat{C}^{\mathsf{suffix}}$ is always empty. By an abuse of notation, for every $u \in \Gamma^*$ and $v \in \Phi_1^{\mathsf{tags}}$ we write "$u = v$" whenever u and v are equivalent by considering v as a word in Γ^*.

We now define the output of $\mathcal{Z}_1$ formally. Fix a tree $t \in \mathscr{L}(\mathcal{R})$ and let $\hat{t}$ be its serialization. The task of $\mathcal{Z}_1$ is to determine in a streaming fashion an $\mathcal{R}$-decomposition tree D of t and a run ρ of $\mathcal{R}$ over D. Then $\mathcal{Z}_1$ outputs a serialization of D, ρ, and atomic rules of Generator obtained from the structure of D and ρ. Formally, given an $\mathcal{R}$-decomposition tree D of t and a run ρ of $\mathcal{R}$ over D, $\mathcal{Z}_1(\hat{t})$ is defined recursively by the function $O(\cdot,\cdot)$ over D and ρ as follows:

- If $D = C$ is a leaf and X is non-horizontal:

$$O(D,\rho) \quad = \quad \langle\, p \cdot \hat{C}^{\mathsf{prefix}}\,\rangle \ \cdot\ X \mapsto \epsilon \ \cdot\ \langle /\, \hat{C}^{\mathsf{suffix}}\,\rangle$$

- If $D = C$ is a leaf and X is horizontal:

$$O(D,\rho) \quad = \quad \langle\, p \cdot \hat{C}^{\mathsf{prefix}}\, /\rangle \ \cdot\ X \mapsto \epsilon$$

- If $D = [C,a](D_1, D_2)$ is an internal node and X is non-horizontal:

$$O(D,\rho) \quad = \quad \langle\, p \cdot \hat{C}^{\mathsf{prefix}}\,\rangle \ \cdot\ X \mapsto X_1X_2 \ \cdot\ O(D_1,\rho_1) \ \cdot\ O(D_2,\rho_2) \ \cdot\ \langle /\, \hat{C}^{\mathsf{suffix}}\,\rangle$$

- If $D = [C,a](D_1, D_2)$ is an internal node and X is horizontal:

$$O(D,\rho) \quad = \quad \langle\, p \cdot \hat{C}^{\mathsf{prefix}}\, /\rangle \ \cdot\ X \mapsto X_1X_2 \ \cdot\ O(D_1,\rho_1) \ \cdot\ O(D_2,\rho_2)$$

where $p = \rho(\epsilon)$, $X = [\rho(\epsilon)]_{\mathcal{R}}$, $X_1 = [\rho(\epsilon{\cdot}1)]_{\mathcal{R}}$, $X_2 = [\rho(\epsilon{\cdot}2)]_{\mathcal{R}}$, and ρ_1 (ρ_2) is the function ρ restricted to D_1 (resp. D_2). Note that for horizontal components the output data $\hat{C}^{\mathsf{suffix}}$ has not been lost, given that this is always empty. Therefore, the output of $\mathcal{Z}_1$ (given t) contains all the serialized data of an $\mathcal{R}$-decomposition D and a run ρ of $\mathcal{R}$ over D, except for the branching nodes of t with

respect to D. These are exactly the nodes of t that are deleted by $\mathcal{Z}_1$ (and more generally by $\mathcal{Z}$). Furthermore, the rest of t apart from its branching nodes will remain unmodified in the remaining steps of the repair.

The first transducer $\mathcal{Z}_1$ works as follows: $\mathcal{Z}_1$ consumes the serialization of a tree $t \in \mathscr{L}(\mathcal{R})$ and determines in a streaming fashion an $\mathcal{R}$-decomposition tree D of t and a run ρ of $\mathcal{R}$ over D. Then it outputs synchronously the function $O(D, \rho)$. That is, $\mathcal{Z}_1(\hat{t}) = O(D, \rho)$ for some $\mathcal{R}$-decomposition tree D of t and a run ρ of $\mathcal{R}$ over D. We postpone the formal definition of $\mathcal{Z}_1$ for the last part of this subsection. Meanwhile, we define recursively when a sequence in $(\Phi_1^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{R}))^*$ is what we call a Generator sequence starting from a component in $\mathsf{SCC}(\mathcal{R})$.

Definition 5.4.1. *Assume that $w \in (\Phi_1^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{R}))^*$. Then w is a* Generator sequence *starting from $X \cdot \vec{x} \in \mathsf{SCC}(\mathcal{R})^+$ if, and only if, one of the following cases holds:*

1. *$w = X \mapsto \epsilon$ and $\vec{x} = \epsilon$, or*
2. *$w = (X \mapsto X_1 X_2) \cdot w'$ and w' is a Generator sequence starting from $X_1 X_2 \cdot \vec{x}$, or*
3. *$w = \langle\, p \cdot \hat{C}^{\mathsf{prefix}} /\rangle \cdot w'$ such that $p \in X$, $\hat{\delta}(p, C) \in X$, and w' is a Generator sequence starting from $X \cdot \vec{x}$, or*
4. *$w = \langle\, p \cdot \hat{C}^{\mathsf{prefix}} \rangle \cdot w' \cdot \langle / \hat{C}^{\mathsf{suffix}} \rangle \cdot w''$ and X is non-horizontal such that $p \in X$, $\hat{\delta}(p, C) \in X$, and w' (w'') is a Generator sequence starting from X ($\vec{x}$ resp.).*

It is obvious from the resemblance between the output of the function $O(\cdot, \cdot)$ and the above definition, that the output $O(D, \rho)$ is a Generator sequence starting from the component $X_0 = [\rho(\epsilon)]_{\mathcal{R}}$ for any $\mathcal{R}$-decomposition tree D and run ρ of $\mathcal{R}$ over D. In fact, by induction over the size of D it is straightforward to show that $O(D, \rho)$ is a Generator sequence starting from the component X_0.

Lemma 5.4.2. *For any tree $t \in \mathscr{L}(\mathcal{R})$, $\mathcal{Z}_1(\hat{t})$ is a Generator sequence starting from X_0 such that X_0 is the initial component of $\mathcal{R}$ (i.e. $p_0 \in X_0$).*

In the sequel of this subsection we give the formal definition of $\mathcal{Z}_1$. This definition and its correctness proof are rather technical. The reader could omit the rest of this subsection and assume that Lemma 5.4.2 is correct. For the sake of completeness, we now give the full description of $\mathcal{Z}_1$.

Fix a tree $t \in \mathscr{L}(\mathcal{R})$ and let $\hat{t}$ be its serialization. The main idea of $\mathcal{Z}_1$ is to run the VPA $\mathcal{R}$ over $\hat{t}$, extract from this run a $\mathcal{R}$-decomposition tree D of $\hat{t}$ and a run ρ of $\mathcal{R}$ over D, and output the function $O(D, \rho)$. In order to achieve these goals, $\mathcal{Z}_1$ runs $\mathcal{R}$ as a top down VPA $\hat{\mathcal{R}}$ over $\hat{t}$ and stores some additional flags inside the stack in order to build the $\mathcal{R}$-decomposition tree in a streaming fashion. Specifically, we define a visible pushdown transducer $\mathcal{Z}_1 = (\Sigma \uplus \bar{\Sigma}, \Sigma \uplus \bar{\Sigma} \uplus \Gamma, Z_1, \kappa_1, z_0, \iota_1, \Omega_1)$ such that:

- $\Sigma \uplus \bar{\Sigma}$ is the input alphabet and $\Sigma \uplus \bar{\Sigma} \uplus \Gamma$ is the output alphabet.

- $Z_1 = P \times \{\mathsf{move}, \neg\mathsf{move}\} \times (\{\mathsf{delete}(p) \mid p \in P\} \cup \{\neg\mathsf{delete}\}) \times \{\,\rangle, /\rangle\}^*$ is the set of stack states where P is the set of control state of $\mathcal{R}$, $\{\mathsf{move}, \neg\mathsf{move}\}$ is a flag where move means that this branch of the run is following the path of a branching state, $(\{\mathsf{delete}(p) \mid p \in P\} \cup \{\neg\mathsf{delete}\})$ is a flag that specifies whether the closing tag $\bar{a} \in \bar{\Sigma}$ (that pops the state from the stack) has to be deleted and p means the state that has to be outputted instead, and $\{\,\rangle, /\rangle\}^*$ is the set of opening and closing parenthesis that are needed for the construction of a closing macro-tags,

- $\kappa_1 : Z_1 \times (\Sigma \uplus \bar{\Sigma}) \to Z_1^* \times (\Sigma \uplus \bar{\Sigma} \uplus \Gamma_1)^+$ is the transition function of $\mathcal{Z}_1$ that we will define later,

- z_0 is the initial state of $\mathcal{Z}_1$ defined by:

$$z_0 = \begin{cases} (p_0, \mathsf{move}, \neg\mathsf{delete}, \rangle) & \text{if } [p_0] \text{ is non-horizontal} \\ (p_0, \mathsf{move}, \neg\mathsf{delete}, \epsilon) & \text{if } [p_0] \text{ is horizontal} \end{cases}$$

- $\iota_1 \in \Sigma \uplus \bar{\Sigma} \uplus \Gamma$ is the initial output of $\mathcal{Z}_1$ such that $\iota_1 = \langle \cdot\, p_0$ where p_0 is the initial control state of $\mathcal{R}$, and

- $\Omega_1 : Z_1 \to \Sigma \uplus \bar{\Sigma} \uplus \Gamma$ is the final output function define only over states of the form $(p, \nu, \neg\mathsf{delete}, u)$ such that:

$$\Omega_1((p, \nu, \neg\mathsf{delete}, u)) = \begin{cases} u & \nu = \neg\mathsf{move} \\ \rangle \cdot [p] \mapsto \epsilon \cdot \langle/ \cdot\, u & \nu = \mathsf{move} \text{ and } [p] \text{ is non-horizontal} \\ \rangle \cdot [p] \mapsto \epsilon \cdot\, u & \nu = \mathsf{move} \text{ and } [p] \text{ is horizontal} \end{cases}$$

Now, we define the transition function κ_1 of $\mathcal{Z}_1$ depending on whether the incoming letter is an opening tag $a \in \Sigma$ or a closing tag $\bar{a} \in \bar{\Sigma}$. First, assume that the incoming letter is $a \in \Sigma$, the top state is $(p, \nu, v, u) \in Z_1$ and $\delta(p, a) = (p_1, p_2)$. Recall that we denote by $[p] \in \mathsf{SCC}(\mathcal{R})$ the strongly connected component of p in $\mathcal{R}$. Then we define $\kappa_1((p, \nu, v, u), a)$ as follows.

1. If $\nu = \neg\mathsf{move}$, then:

$$\kappa_1((p, \neg\mathsf{move}, v, u), a) = ((p_1, \neg\mathsf{move}, \neg\mathsf{delete}, \epsilon) \cdot (p_2, \neg\mathsf{move}, v, u), a).$$

2. If $\nu = \mathsf{move}$ and $[p] = [p_1]$, then:

$$\kappa_1((p, \mathsf{move}, v, u), a) = ((p_1, \mathsf{move}, \neg\mathsf{delete}, \epsilon) \cdot (p_2, \neg\mathsf{move}, v, u), a).$$

3. If $\nu = \mathsf{move}$ and $[p_1] \neq [p] = [p_2]$, then:

$$\kappa_1((p, \mathsf{move}, v, u), a) = ((p_1, \neg\mathsf{move}, \neg\mathsf{delete}, \epsilon) \cdot (p_2, \mathsf{move}, v, u), a).$$

4. If $\nu = \mathsf{move}$, $[p]$ is non-horizontal and $[p_1] \neq [p] \neq [p_2]$, then $\kappa_1((p, \mathsf{move}, v, u), a) = (z_1 \cdot z_2, o)$ such that

$$\begin{aligned} z_1 &= (p_1, \mathsf{move}, \mathsf{delete}(p_2), \rangle^{b_1}) \\ z_2 &= (p_2, \mathsf{move}, v, \rangle^{b_2} \cdot \langle / \cdot u) \\ o &= \rangle \cdot [p] \mapsto [p_1][p_2] \cdot \langle \cdot p_1 \end{aligned}$$

where b_1 (b_2) is equal to 1 if the component $[p_1]$ ($[p_2]$, resp.) is non-horizontal, and 0 otherwise.

5. If $\nu = \mathsf{move}$, $[p]$ is horizontal, and $[p_1] \neq [p] \neq [p_2]$, then $\kappa_1((p, \mathsf{move}, v, u), a) = (z_1 \cdot z_2, o)$ such that

$$\begin{aligned} z_1 &= (p_1, \mathsf{move}, \mathsf{delete}(p_2), \rangle^{b_1}) \\ z_2 &= (p_2, \mathsf{move}, v, \rangle^{b_2} \cdot u) \\ o &= /\rangle \cdot [p] \mapsto [p_1][p_2] \cdot \langle \cdot p_1 \end{aligned}$$

where b_1 (b_2) is equal to 1 if the component $[p_1]$ ($[p_2]$, resp.) is non-horizontal, and 0 otherwise.

Notice how states $p, p_1, p_2 \in P$ at the first component of Z_1 simulates a run of top-down VPA $\hat{\mathcal{R}}$ over the input. In each case, it holds that $\kappa_1((p, \nu, v, u), a) = ((p_1, \nu_1, v_1, u_1) \cdot (p_2, \nu_2, v_2, u_2), o)$. Now, in the first case when $\nu = \neg\mathsf{move}$, no action of $\mathcal{Z}_1$ is needed and the information is just passed (i.e. v and u are copied to the second state) without any modification to the input. Furthermore, cases 2 and 3 only passes the collected information to the next states and marks with the move flag which states are in the same component (only one is chosen arbitrarily). After a branching node is founded (cases 4 and 5), the move flag is passed to both states and the incoming letter a is deleted. In order to delete the closing tag $\bar{a}$ in the future, the topmost state is marked with $\mathsf{delete}(p_2)$ (to understand why p_2 is also stored and how $\mathsf{delete}(p_2)$ is used when the closing tag $\bar{a}$ is seen, we suggest the reader to follow cases 3, 4, or 5 for $\bar{a} \in \bar{\Sigma}$). Strings $\rangle^{b_1}$ and $\rangle^{b_2} \cdot u$ (or $\rangle^{b_2} \cdot \langle / \cdot u$ when $[p]$ is non-horizontal) stored at the fourth component of Z_1 are needed for technical reasons in order to be used in the output of the matching closing macro-tag when $[p_1]$ or $[p_2]$ are non-horizontal components. Finally, the output of the last two cases shows how a prefix rewriting for Generator of the form $[p] \mapsto [p_1][p_2]$ is obtained from the input . After $[p] \mapsto [p_1][p_2]$ is outputted, a new opening macro-tag is opened to be filled with the prefix of a context.

On the other hand, assume that the incoming letter is $\bar{a} \in \bar{\Sigma}$ and the current state at the top of the stack is $(p, \nu, v, u) \in Z_1$. Then we have the following cases:

1. If $\nu = \mathsf{move}$, $[p]$ is non-horizontal, and $v = \neg\mathsf{delete}$, then:

$$\kappa_1((p, \mathsf{move}, \neg\mathsf{delete}, u), \bar{a}) = (\epsilon, \rangle \cdot [p] \mapsto \epsilon \cdot \langle / \cdot u \cdot \bar{a})$$

2. If $\nu = \mathsf{move}$, $[p]$ is horizontal, and $v = \neg\mathsf{delete}$, then:

$$\kappa_1((p, \mathsf{move}, \neg\mathsf{delete}, u), \bar{a}) \;=\; (\epsilon,\ /\rangle \cdot [p] \mapsto \epsilon \cdot u \cdot \bar{a}\ )$$

3. If $\nu = \mathsf{move}$, $[p]$ is non-horizontal, and $v = \mathsf{delete}(p')$, then:

$$\kappa_1((p, \mathsf{move}, \mathsf{delete}(p'), u), \bar{a}) \;=\; (\epsilon,\ \rangle \cdot [p] \mapsto \epsilon \cdot \langle / \cdot u \cdot \langle \cdot p'\ )$$

4. If $\nu = \mathsf{move}$, $[p]$ is horizontal, and $v = \mathsf{delete}(p')$, then:

$$\kappa_1((p, \mathsf{move}, \mathsf{delete}(p'), u), \bar{a}) \;=\; (\epsilon,\ /\rangle \cdot [p] \mapsto \epsilon \cdot u \cdot \langle \cdot p'\ )$$

5. If $\nu = \neg\mathsf{move}$ and $v = \neg\mathsf{delete}$, then:

$$\kappa_1((p, \neg\mathsf{move}, \neg\mathsf{delete}, u), \bar{a}) \;=\; (\epsilon,\ u \cdot \bar{a})$$

6. If $\nu = \neg\mathsf{move}$ and $v = \mathsf{delete}(p')$, then:

$$\kappa_1((p, \neg\mathsf{move}, \mathsf{delete}(p'), u), \bar{a}) \;=\; (\epsilon,\ u \cdot \langle \cdot p')$$

The transition function κ_1 over closing tags in $\bar{\Sigma}$ shows again how $\mathcal{Z}_1$ mimics a run of $\hat{\mathcal{R}}$ over the input tree. In particular, when a closing tag $\bar{a}$ is seen, the top state is popped from the stack like in $\hat{\mathcal{R}}$. In the first cases (1 to 4), the move flag marks the search of a branching node. Therefore, if a closing tag is received, this marks the end of the searching phase and a popping rule $[p] \mapsto \epsilon$ is released. In contrast, if move and delete flag are off (case 5), then $\bar{a}$ is part of a context suffix, namely, embedded in a closing macro-tag of the output. Recall that $\mathsf{delete}(p')$ marks that a closing tag has to be deleted when the current state is popped. This is exactly the behaviour defined in item 3, 4, and 6, where $\bar{a}$ is deleted and replaced by the beginning of an opening macro-tag (i.e. $\langle \cdot p'$). Finally, the elements $\langle /$ and u in the output are needed for technical reasons in order to correctly outputted the closing macro-tags of non-horizontal components.

In the next lemma we show that $\mathcal{Z}_1$ is correct, which means that for any tree $t \in \mathscr{L}(\mathcal{R})$ there exists a $\mathcal{R}$-decomposition tree D of t and a run ρ of $\mathcal{R}$ over D such that $\mathcal{Z}_1(\hat{t}) = O(D, \rho)$.

Lemma 5.4.3. *For any tree $t \in \mathscr{L}(\mathcal{R})$, there exists a $\mathcal{R}$-decomposition tree D of t and a run ρ of $\mathcal{R}$ over D such that*

$$\mathcal{Z}_1(\hat{t}) \;=\; O(D, \rho).$$

Proof. The proof follows by induction on the *partial order* between strongly connected components in $\mathcal{R}$. Formally, we define the partial order $X \preceq_{\mathcal{R}} X'$ between strongly connected components in $\mathsf{SCC}(\mathcal{R})$ iff there exists $p \in X$ and $p' \in X'$ such that $\delta(p', a) = (p, p'')$ or $\delta(p', a) = (p'', p)$. It is straightforward to check that $\preceq_{\mathcal{R}}$ is effectively a partial order. Furthermore, this partial order has

been implicitly used in most of the definition through all the paper. We use the partial order $\preceq_{\mathcal{R}}$ to prove Lemma 5.4.3 by induction.

Let $X \in \mathsf{SCC}(\mathcal{R})$ and $p \in X$. Take a forest $h \in \mathcal{F}_\Sigma$ (where $\mathcal{F}_\Sigma$ denote the set of all forests in Σ) such that there exists a run of $\mathcal{R}$ over h starting from p. We make the following claim.

Claim 1.

- *If X is non-horizontal, then $(p, \mathsf{move}, v, \rangle) \xrightarrow{\hat{h}/w}_{\mathcal{Z}_1} (f, \nu, v, u)$ and:*

$$\langle \cdot p \cdot w \cdot \Omega_1(f, \nu, v, u)$$

  is the serialization of an $\mathcal{R}$-decomposition tree D of h and a run ρ of $\mathcal{R}$ over D.

- *If X is horizontal, then $(p, \mathsf{move}, v, \epsilon) \xrightarrow{\hat{h}/w}_{\mathcal{Z}_1} (f, \nu, v, u)$ and:*

$$\langle \cdot p \cdot w \cdot \Omega_1(f, \nu, v, u)$$

  is the serialization of an $\mathcal{R}$-decomposition tree D of h and a run ρ of $\mathcal{R}$ over D.

From the above claim and the definition of $\mathcal{Z}_1$, one can easily conclude that Lemma 5.4.3 holds. Indeed, if X_0 is the initial component in $\mathcal{R}$ and non-horizontal, then $(p_0, \mathsf{move}, \neg\mathsf{delete}, \rangle)$ is the initial state in $\mathcal{Z}_1$ and for any tree $t \in \mathscr{L}(\mathcal{R})$ it holds that $(p_0, \mathsf{move}, \neg\mathsf{delete}, \rangle) \xrightarrow{\hat{t}/w}_{\mathcal{Z}_1} (f, \nu, \neg\mathsf{delete}, u)$ by the above claim. The final output of $\mathcal{Z}_1$ over $\hat{t}$ is thus equal to $\iota_1 \cdot w \cdot \Omega_1(f, \nu, \neg\mathsf{delete}, u)$ where, by the above claim, is the serialization of an $\mathcal{R}$-decomposition tree D of t and a run ρ of $\mathcal{R}$ over D. The case when X_0 is horizontal follows the same arguments. Therefore, Lemma 5.4.3 can be shown from Claim 1 and the definition of $\mathcal{Z}_1$. As was previously mentioned, we prove Claim 1 by induction over the partial order $\preceq_{\mathcal{R}}$. For the sake of simplification, in each base and inductive case we only consider the case where X is non-horizontal. The case in which X is horizontal is analogous and we leave the proof to the reader.

For the base case, assume that X is a minimal element with respect to $\preceq_{\mathcal{R}}$ and let h be any forest in $\mathcal{F}_\Sigma$. Furthermore, assume that $\hat{h} \neq \epsilon$. Indeed, if $\hat{h} = \epsilon$ we get that $w = \epsilon$ and $\langle \cdot p \cdot \rangle \cdot X \mapsto \epsilon \cdot \langle / \cdot \rangle$ is the serialization of an $\mathcal{R}$-decomposition tree of h. Therefore, let $\hat{h} \neq \epsilon$ and let $w_1 = a_1 \cdots a_n \in \Sigma^*$ be the maximum prefix of $\hat{h}$ that consist of only opening tags in Σ^*. Given that $\hat{h} \neq \epsilon$ and h is a forest, we know that $w_1 \neq \epsilon$. Consider the following partial run of $\mathcal{Z}_1$ over w_1 which can be easily checked by the definition of $\mathcal{Z}_1$:

$$(p, \mathsf{move}, v, \rangle) \xrightarrow{w_1/w_1}_{\mathcal{Z}_1} (p_1, \mathsf{move}, \neg\mathsf{delete}, \epsilon) \cdot (p_2, \neg\mathsf{move}, \neg\mathsf{delete}, \epsilon) \cdots (p_{n+1}, \neg\mathsf{move}, v, \rangle)$$

Denote $(p_2, \neg\mathsf{move}, \neg\mathsf{delete}, \epsilon) \cdots (p_{n+1}, \neg\mathsf{move}, v, \rangle)$ by $\vec{z}$. By the maximality of w_1, the next letter in $\hat{t}$ is in $\bar{\Sigma}$. This implies the transition $(p_1, \mathsf{move}, \neg\mathsf{delete}, \epsilon) \cdot \vec{z} \xrightarrow{\bar{a}/w'}_{\mathcal{Z}_1} \vec{z}$ in $\mathcal{Z}_1$ where $w' = \rangle \cdot X \mapsto \epsilon \cdot \langle / \cdot \bar{a}$. Let $w_2 \in (\Sigma \uplus \bar{\Sigma})^*$ such that $\hat{h} = w_1 \cdot \bar{a} \cdot w_2$. Notice that all states in $\vec{z}$ have the move flag and delete

flag off. Then one can easily check that $\vec{z} \xrightarrow{w_2/w_2}_{\mathcal{Z}_1} (f, \neg\mathsf{move}, v, \rangle)$ for some $f \in F$. Putting everything together we get that $(p, \mathsf{move}, v, \rangle) \xrightarrow{\hat{h}/w}_{\mathcal{Z}_1} (f, \neg\mathsf{move}, v, \rangle)$ where w is equal to:

$$w \quad = \quad w_1 \cdot \rangle \cdot X \mapsto \epsilon \cdot \langle / \cdot \bar{a} \cdot w_2$$

Clearly, one can check that $\langle \cdot p \cdot w_1 \cdot \rangle \cdot X \mapsto \epsilon \cdot \langle / \cdot \bar{a} \cdot w_2 \cdot \rangle$ is the serialization of an $\mathcal{R}$-decomposition tree of h and the base case was proved.

For the inductive case, suppose that Claim 1 holds for every $X' \in \mathsf{SCC}(\mathcal{R})$ such that $X' \prec_{\mathcal{R}} X$, and we prove the claim for X. Let h be any forest in $\mathcal{F}_\Sigma$ such that $\hat{h} \neq \epsilon$. Define w' to be the minimum prefix of $\hat{h}$ such that $\hat{\delta}(p, w') = p' \cdot \vec{p}$ where $p' \in X$ and either:

- $w' \cdot a$ is a prefix of $\hat{h}$ with $a \in \Sigma$ such that $\delta(p', a) = (p_1, p_2)$ and $[p_1] \neq [p'] \neq [p_2]$, or
- $w' \cdot \bar{a}$ is a prefix of $\hat{h}$ with $a \in \bar{\Sigma}$ such that $p' \in F$.

It is easy to show that w' always exists by an induction over the size of h. Then assume that w' exists and, without loss of generality, assume that there exists $a \in \Sigma$ such that $w' \cdot a$ is a prefix of $\hat{h}$, $\delta(p', a) = (p_1, p_2)$, and $[p_1] \neq [p'] \neq [p_2]$. For simplicity, we omit the second case when $w' \cdot \bar{a}$ is a prefix of $\hat{h}$ since it can be proved with the same ideas used in the base case. Moreover, we assume that $X_1 = [p_1]$ is non-horizontal and $X_2 = [p_2]$ is horizontal. All the other combinations where $[p_1]$ or $[p_2]$ can be either non-horizontal or horizontal can be proved similarly. Summing up, we assume that there exists $a \in \Sigma$ and $w' \in \Sigma^*$ such that $w' \cdot a$ is the minimum prefix of $\hat{h}$ and:

- $\hat{\delta}(p, w') = p' \cdot \vec{p}$ where $p' \in X$,
- $\delta(p', a) = (p_1, p_2)$,
- $X_1 = [p_1]$ is non-horizontal,
- $X_2 = [p_2]$ is horizontal, and
- $X_1 \neq X \neq X_2$.

By the previous sequence of properties, the next state and output of $\mathcal{Z}_1$ can be derived from the transitions of $\mathcal{Z}_1$:

$$(p, \mathsf{move}, v, \rangle) \xrightarrow{w'/w'}_{\mathcal{Z}_1} (p', \mathsf{move}, \neg\mathsf{delete}, \epsilon) \cdot (p_1, \neg\mathsf{move}, \neg\mathsf{delete}, \epsilon) \cdots (p_n, \neg\mathsf{move}, v, \rangle)$$

for some $p_1, \ldots, p_n \in P$ and $n \in \mathbb{N}$. Since X is non-horizontal and $[p_1] \neq [p] \neq [p_2]$, we know that:

$$\kappa_1((p', \mathsf{move}, \neg\mathsf{delete}, \epsilon), a) \quad = \quad ((p_1, \mathsf{move}, \mathsf{delete}(p_2), \rangle) \cdot (p_2, \mathsf{move}, \neg\mathsf{delete}, \langle /), \rangle \cdot X \mapsto X_1 X_2 \cdot \langle \cdot p_1).$$

In the following, we apply the inductive hypothesis over X_1 and X_2 over the top states $(p_1, \mathsf{move}, \mathsf{delete}(p_2), \rangle)$ and $(p_2, \mathsf{move}, \neg\mathsf{delete}, \langle /)$. First, let h_1 and h_2 in $\mathcal{F}_\Sigma$ such that $\hat{h} =$

$w' \cdot a \cdot \hat{h}_1 \cdot \bar{a} \cdot \hat{h}_1 \cdot w''$ for some $w'' \in (\Sigma \uplus \bar{\Sigma})^*$. Intuitively, h_1 is the children-forest and h_2 is the right-sibling-forest of the node labelled by a in h. Coming back to the proof, we know that $X_1 \prec X$ and $X_2 \prec X$ and, thus, we can apply the inductive hypothesis over X_1 (X_2), p_1 (p_2), h_1 (h_2), and $(p_1, \mathsf{move}, \mathsf{delete}(p_2), \rangle)$ ($(p_2, \mathsf{move}, \neg\mathsf{delete}, \langle/)$ resp.). By applying the inductive hypothesis, we get the following derivation:

$$\begin{aligned} (p_1, \mathsf{move}, \mathsf{delete}(p_2), \rangle) &\xrightarrow{\hat{h}_1/w_1}_{\mathcal{Z}_1} (f_1, \nu_1, \mathsf{delete}(p_2), u_1) \\ (p_2, \mathsf{move}, \neg\mathsf{delete}, \epsilon) &\xrightarrow{\hat{h}_2/w_2}_{\mathcal{Z}_1} (f_2, \nu_2, \neg\mathsf{delete}, u_2) \end{aligned}$$

such that $\langle \cdot p_1 \cdot w_1 \cdot \Omega_1(f_1, \nu_1, \mathsf{delete}(p_2), u_1)$ and $\langle \cdot p_2 \cdot w_2 \cdot \Omega_1(f_2, \nu_2, \neg\mathsf{delete}, u_2)$ are serialization of $\mathcal{R}$-decomposition trees of h_1 and h_2, respectively. If we continue the run of $\mathcal{Z}_1$, we can derive that $(p, \mathsf{move}, v, \rangle) \xrightarrow{\hat{h}/w}_{\mathcal{Z}_1} (f, \nu, v, u)$ where:

$$w = w' \cdot \rangle \cdot X \mapsto X_1 X_2 \cdot \langle \cdot p_1 \cdot w_1 \cdot \Omega_1(f_1, \nu_1, \mathsf{delete}(p_2), u_1) \cdot \langle \cdot p_2 \cdot w_2 \cdot \Omega_1(f_2, \nu_2, \neg\mathsf{delete}, u_2) \cdot \langle/ \cdot w''$$

It is straightforward to show that $\langle \cdot p \cdot w \cdot \Omega_1(f, \nu, v, u)$ is the serialization of an $\mathcal{R}$-decomposition tree of h. This concludes the proof of Claim 1. □

5.4.2 Playing against the input

We begin this subsection by proving a technical lemma about tree and context languages defined over a top-down tree automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$. This lemma is fundamental for defining the output of $\mathcal{Z}_2$ and for the overall construction of $\mathcal{Z}$. Before going into the statement of the lemma, we need to define the parametrization of the language $\mathscr{L}(\mathcal{R} \mid X)$ with respect to a state $q \in Q$. Specifically, given a component $X \in \mathsf{SCC}(\mathcal{A})$ we parametrize the language of contexts $\mathscr{L}(\mathcal{R} \mid X)$ by an initial state $q \in Q$ as follows:

$$\mathscr{L}(\mathcal{A}|_q X) = \left\{ C \mid \hat{\delta}(q, C) \in X \right\}.$$

Intuitively, this language consists of all contexts realized within component X starting from state q. Clearly, we have:

$$\mathscr{L}(\mathcal{A} \mid X) = \bigcup_{q \in X} \mathscr{L}(\mathcal{A}|_q X).$$

The following lemma is crucial for defining $\mathcal{Z}_2$ and for the uniform cost of $\mathcal{Z}$. An analogous lemma (Lemma 2.3.5) was presented in Chapter 2 for streaming repairing strings and, actually, both proofs follows the same ideas.

Lemma 5.4.4. *Let $X \in \mathsf{SCC}(\mathcal{R})$ and $Y \in \mathsf{SCC}(\mathcal{T})$ such that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$. Then, for all $p \in X$, there is $q \in Y$ such that $\mathscr{L}(\mathcal{R}|_p X) \subseteq \mathscr{L}(\mathcal{T}|_q Y)$.*

Proof. We show this lemma by contrapositive by assuming that there exists $p \in X$ such that $\mathscr{L}(\mathcal{R}|_p X) \notin \mathscr{L}(\mathcal{T}|_q Y)$ for all $q \in Y$, and proving that $\mathscr{L}(\mathcal{R} \mid X) \notin \mathscr{L}(\mathcal{T} \mid Y)$. Moreover, we prove this claim by induction over the set $Y = \{q_1, \ldots, q_n\}$ such that for every $i \leq n$ there exists $C_i \in \mathscr{L}(\mathcal{R} \mid X)$ that satisfies $C_i \notin \mathscr{L}(\mathcal{T}|_{q_j} Y)$ for all $j \leq i$. Clearly, for $i = n$ this implies that $C_n \notin \mathscr{L}(\mathcal{T} \mid Y)$ and the lemma is proved.

Fix two top-down tree automata $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$ and $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ with $X \in \mathsf{SCC}(\mathcal{R})$ and $Y \in \mathsf{SCC}(\mathcal{T})$. First of all, notice that the base case $i = 1$ is trivially true. Indeed, take by hypothesis the context $C_1 \in \mathscr{L}(\mathcal{R}|_p X)$ such that $C_1 \notin \mathscr{L}(\mathcal{T}|_{q_1} Y)$ and the base case is shown. For the inductive case $i < n$, assume that C_i satisfies the induction hypothesis. If $\hat{\gamma}(q_{i+1}, C_i) \notin Y$, then define $C_{i+1} = C_i$. Thus, we have $C_{i+1} \notin \mathscr{L}(\mathcal{T}|_{q_j} Y)$ for all $j \leq i+1$. Otherwise, given that X is strongly connected, take two contexts $C, C' \in \mathcal{C}^{\downarrow}_{\Sigma}$ such that $\hat{\delta}(p, C_i \circ C) = p$ and $C' \notin \mathscr{L}(\mathcal{T}|_{q_{i+1}} Y)$ where $C' \in \mathscr{L}(\mathcal{R}|_p X)$. Then define the context $C_{i+1} = C_i \circ C \circ C'$. Finally, it is straightforward to prove that C_{i+1} satisfies the desire inductive case $i+1$, that is, $C_{i+1} \in \mathscr{L}(\mathcal{R} \mid X)$ but $C_{i+1} \notin \mathscr{L}(\mathcal{T}|_{q_j} Y)$ for all $j \leq i+1$. This was to be shown. □

The output elements of the first transducer $\mathcal{Z}_1$ can be decomposed into three disjoint sets: (1) moves of Generator on the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ of the form $X \mapsto X_1 X_2$ or $X \mapsto \epsilon$, (2) opening and leaf macro-tags $\langle\, p\ \hat{C}^{\mathsf{prefix}} \,\rangle$ and $\langle\, p\ \hat{C}^{\mathsf{prefix}} /\rangle$, and (3) closing macro-tags $\langle / \hat{C}^{\mathsf{suffix}} \rangle$. The main goals of $\mathcal{Z}_2$ are related with the first and second set of output elements. The first goal of $\mathcal{Z}_2$ is to reply to the sequence of moves of Generator with the corresponding moves of Repairer: $\mathcal{Z}_2$ receives moves of Generator sequentially from $\mathcal{Z}_1$ and output immediately the corresponding sequence of moves of Repairer with respect to the strategy W. Since W is a winning strategy for Repairer, we know that W respects the restriction imposed by Referee and, moreover, it always replies to the moves of Generator (i.e. Repairer wins the game). The second goal of $\mathcal{Z}_2$ is to annotate the opening and leaf macro-tags of the form $\langle\, p\ \hat{C}^{\mathsf{prefix}} \,\rangle$ and $\langle\, p\ \hat{C}^{\mathsf{prefix}} /\rangle$, respectively, with states from $\mathcal{T}$ depending on the component at the top of Repairer stack. Formally, suppose that before $\langle\, p\ \hat{C}^{\mathsf{prefix}} \,\rangle$ is consumed, $\mathcal{Z}_2$ determines previously a component $Y \in \mathsf{SCC}(\mathcal{T})$ such that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$ where $X = [p]_{\mathcal{R}}$. By Lemma 5.4.4, we know that there exists a state $q_1 \in Y$ such that $\mathscr{L}(\mathcal{R}|_p X) \subseteq \mathscr{L}(\mathcal{T}|_{q_1} Y)$. Furthermore, we know that $C \in \mathscr{L}(\mathcal{R}|_p X)$ given that $\langle\, p\ \hat{C}^{\mathsf{prefix}} \,\rangle$ comes from a Generator sequence (Lemma 5.4.2) and, thus, $C \in \mathscr{L}(\mathcal{T}|_{q_1} Y)$. Let $q_2 \in Y$ be a state such that $q_2 = \hat{\gamma}(q_1, C)$. Then the second goal of $\mathcal{Z}_2$ is to read $\langle\, p\ \hat{C}^{\mathsf{prefix}} \,\rangle$ and output $\langle\, q_1\ \hat{C}^{\mathsf{prefix}}\ q_2 \,\rangle$. Notice that states q_1 and q_2 are determined by the elements $p \in P$, $C \in \mathcal{C}^{\downarrow}_{\Sigma}$, and $Y \in \mathsf{SCC}(\mathcal{T})$. To formalize this fact, we define the function $V : P \times \mathcal{C}^{\downarrow}_{\Sigma} \times \mathsf{SCC}(\mathcal{T}) \to Q \times Q$ such that $V(p, C, Y) = (q_1, q_2)$ as it was stated above (i.e. q_1 and q_2 satisfies $\mathscr{L}(\mathcal{R}|_p X) \subseteq \mathscr{L}(\mathcal{T}|_{q_1} Y)$ and $q_2 = \hat{\gamma}(q_1, C)$ for $q_1, q_2 \in Y$). If $\mathscr{L}(\mathcal{R} \mid [p]) \notin \mathscr{L}(\mathcal{T} \mid Y)$ or $\hat{\delta}(p, C) \notin [p]$, then V is not defined. We use V in the definition of the second transducer $\mathcal{Z}_2$.

We now give the main technical definitions in order to define the output of $\mathcal{Z}_2$. Recall that by assumption Repairer has a positional strategy for winning the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. We

describe this strategy by a function:

$$W : (\mathsf{SCC}(\mathcal{R}) \uplus \{\triangleleft\})^+ \times (\mathsf{SCC}(\mathcal{T}) \uplus \{\triangleleft\})^+ \to \mathsf{moves}(\mathcal{T})^*$$

such that if $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle$ is the current state of the game and $W(\vec{x}, \vec{y}) = \vec{m}$, then $\vec{y} \overset{\vec{m}}{\mapsto} \vec{y}'$ for some $\vec{y}' \in (\mathsf{SCC}(\mathcal{T}) \uplus \{\triangleleft\})^+$ and $\langle\!\langle \vec{x}, \vec{y} \rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\vec{x}, \vec{y}']\!]$ is the next position of Repairer in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Furthermore, we suppose that if $\vec{x} = \triangleleft \cdot \vec{x}''$ and $\vec{y} = \vec{y}' \cdot \triangleleft \cdot \vec{y}''$, then $W(\vec{x}, \vec{y}) = \vec{m}$ such that $y' \overset{\vec{m}}{\mapsto} \epsilon$, i.e. $\vec{m}$ simulates the popping sequence does by Referee for cleaning the part of the stack of Repairer that is over the top separator $\triangleleft$. In particular, we suppose that for every $\vec{y} \in \mathsf{SCC}(\mathcal{T})^*$ it holds that $W(\epsilon, \vec{y}) = \vec{m}$ where $\vec{m}$ satisfies $y \overset{\vec{m}}{\mapsto} \epsilon$. In the definition of $\mathcal{Z}_2$ we strongly use function W for leading the actions of $\mathcal{Z}_2$.

Let $\Gamma_2 = \Sigma \uplus \bar{\Sigma} \uplus \mathsf{moves}(\mathcal{T}) \uplus Q \uplus \{\langle\, , /, \,\rangle\}$ be the output alphabet of $\mathcal{Z}_2$ such that $\Sigma \uplus \bar{\Sigma}$ is the restriction alphabet, $\mathsf{moves}(\mathcal{T})$ is the set of atomic rules of Repairer, Q is the set of states of $\mathcal{T}$, and $\{\langle\, , /, \,\rangle\}$ is the set of separators. Similar to Γ_1, we define from Γ_2 an infinite alphabet of opening, closing, and leaf macro-tags that help us for abstracting the specification of the third transducer. Hence, we define the alphabet of opening macro-tags $\Phi_2^{\mathsf{open}} = \{\langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} q_2 \,\rangle \mid q_1, q_2 \in Q, C \in \mathcal{C}_\Sigma^{\downarrow}\}$, the alphabet of closing macro-tags $\Phi_2^{\mathsf{close}} = \{\langle / \hat{C}^{\mathsf{suffix}} \,\rangle \mid C \in \mathcal{C}_\Sigma^{\downarrow}\}$, and the alphabet of leaf macro-tags $\Phi_2^{\mathsf{leaf}} = \{\langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,/\rangle \mid q_1, q_2 \in Q, C \in \mathcal{C}_\Sigma^{\downarrow}\}$. Further, we define the alphabet of all macro-tags by $\Phi_2^{\mathsf{tags}} = \Phi_2^{\mathsf{open}} \cup \Phi_2^{\mathsf{close}} \cup \Phi_2^{\mathsf{leaf}}$. Alphabet Φ_2^{tags} together with $\mathsf{moves}(\mathcal{T})$ correspond to the macro alphabet of the output of $\mathcal{Z}_2$.

Now, we have all the ingredients to give the definition of $\mathcal{Z}_2$. We define the second transducer $\mathcal{Z}_2$ formally by the tuple $\mathcal{Z}_2 = (\Phi_1^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{R}), \Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}), Z_2, \kappa_2, \iota_2, \Omega_2)$ such that each component of $\mathcal{Z}_2$ is defined as follows.

- $\Phi_1^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{R})$ and $\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T})$ are the set of input and output alphabet, respectively.
- Z_2 is the set of control states $(\vec{x}, \vec{y}) \in (\mathsf{SCC}(\mathcal{R}) \cup \{\triangleleft\})^* \times (\mathsf{SCC}(\mathcal{T}) \cup \{\triangleleft\})^+$ such that:

  $$(a)\ \ \mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x})) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y})) \quad \text{or} \quad (b)\ \ \mathsf{top}(\vec{x}) = \mathsf{top}(\vec{y}) = \triangleleft.$$

  A state $(\vec{x}, \vec{y}) \in Z_2$ that satisfies (a) can be seen as a Generator position $[\![\vec{x}, \vec{y}]\!]$ (see Definition 5.3.1). On the other hand, a state $(\vec{x}, \vec{y}) \in Z_2$ that satisfies (b) is like a Referee position where the top symbol $\triangleleft$ is waiting to be removed.
- ι_2 is the initial function defined by $\iota((\vec{x}_0, \vec{y})) = \vec{m}_0$ if, and only if, $W(\vec{x}_0, \vec{y}_0) = \vec{m}_0$ and $\vec{y}_0 \overset{\vec{m}_0}{\mapsto} \vec{y}$ for $\vec{y} \in (\mathsf{SCC}(\mathcal{T}) \cup \{\triangleleft\})^+$. In other words, the initial state of $\mathcal{Z}_2$ is the pair of stacks $(\vec{x}_0, \vec{y})$ after Repairer does his first sequence of moves $\vec{m}_0$ in order to satisfy the initial restriction $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x}_0)) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}))$. In fact, $\mathcal{Z}_2$ starts by outputting the sequence $\vec{m}_0$ before receiving any input from $\mathcal{Z}_1$.

- $\kappa_2 : Z_2 \times (\Phi_1^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{R})) \to Z_2 \times (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}))$ is the transition function of $\mathcal{Z}_2$ defined for every state $(\vec{x}, \vec{y}) \in Z_2$ and $e \in \Phi_1^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{R})$ such that:

  – If $e = X \mapsto X_1X_2$ and $\vec{x} = X \cdot \vec{x}'$, then we define:

  $$\kappa_2((X \cdot \vec{x}', \vec{y}), X \mapsto X_1X_2) = ((X_1X_2 \cdot \vec{x}', \vec{y}'), \vec{m})$$

  such that $W(X_1X_2 \cdot \vec{x}', \vec{y}) = \vec{m}$ and $\vec{y} \overset{\vec{m}}{\mapsto} \vec{y}'$. Informally, κ_2 updates the stack of Generator from $X \cdot \vec{x}'$ to $X_1X_2 \cdot \vec{x}'$ with the incoming rule $e = X \mapsto X_1X_2$, outputs the next sequence of rules $\vec{m} = W(X_1X_2 \cdot \vec{x}', \vec{y})$ of Repairer, and finally updates the stack of Repairer from $\vec{y}$ to $\vec{y}'$ by following $\vec{m}$.

  – If $e = X \mapsto \epsilon$ and $\vec{x} = X \cdot \vec{x}'$, then we define:

  $$\kappa_2((X \cdot \vec{x}', \vec{y}), X \mapsto \epsilon) = ((\vec{x}', \vec{y}'), \vec{m})$$

  such that $W(\vec{x}', \vec{y}) = \vec{m}$ and $\vec{y} \overset{\vec{m}}{\mapsto} \vec{y}'$. Notice that this case is analogous to the previous one. Indeed, κ_2 updates the stack of Generator from $X \cdot \vec{x}'$ to $\vec{x}'$ with the incoming rule $e = X \mapsto \epsilon$, outputs the next sequence of rules $\vec{m} = W(\vec{x}', \vec{y})$ of Repairer, and updates the stack of Repairer from $\vec{y}$ to $\vec{y}'$ by following $\vec{m}$. Note that, in the particular case when $\mathsf{top}(\vec{x}') = \triangleleft$, it happen that $\vec{m} = W(\vec{x}', \vec{y})$ is the sequence of moves that cleans the top of the stack of Repairer until the first separator. Furthermore, $\mathcal{Z}_2$ cleans completely the stack of Repairer whenever $\vec{x}' = \epsilon$.

Note that until this point we have not made any direct reference to Referee in κ_2. Moreover we have not shown how to deal with separators in Z_2. The following cases (when either $e = \langle\, p \cdot \hat{C}^{\mathsf{prefix}} \,\rangle$ or $e = \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle$) deal with this issue.

– If $e = \langle\, p \cdot \hat{C}^{\mathsf{prefix}} \,\rangle$ and $(\vec{x}, \vec{y}) = (X \cdot \vec{x}', Y \cdot \vec{y}')$ such that X is non-horizontal and $p \in X$, then we define:

$$\kappa_2((X \cdot \vec{x}', Y \cdot \vec{y}'), \langle\, p \cdot \hat{C}^{\mathsf{prefix}} \,\rangle) = ((X \cdot \triangleleft \cdot \vec{x}', Y \cdot \triangleleft \cdot \vec{y}'), \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,\rangle)$$

such that $V(p, \hat{C}^{\mathsf{prefix}}, Y) = (q_1, q_2)$ (recall the definition of V at the beginning of this subsection after Lemma 5.4.4). This transition has mainly two goals. The first goal is to add separators below the top elements X and Y of $\vec{x}$ and $\vec{y}$, respectively. As was informally shown in the definition of $\mathcal{Z}_1$, opening macro-tags mark exactly the position when this separator has to be added. The second goal of this transition is to annotate the opening macro-tags with the states q_1 and q_2. Notice that the condition $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$ is satisfied by the definition of Z_2 and, thus, function V is correctly defined over p, $\hat{C}^{\mathsf{prefix}}$, and Y.

– If $e = \langle / \hat{C}^{\mathsf{prefix}} \rangle$ and $(\vec{x}, \vec{y}) = (\triangleleft \cdot \vec{x}', \triangleleft \cdot \vec{y}')$, then we define:

$$\kappa_2((\triangleleft \cdot \vec{x}', \triangleleft \cdot \vec{y}'), \langle / \hat{C}^{\mathsf{suffix}} \rangle) = ((\vec{x}', \vec{y}''), \langle / \hat{C}^{\mathsf{suffix}} \rangle \cdot \vec{m})$$

such that $W(\vec{x}', \vec{y}') = \vec{m}$ and $\vec{y}' \overset{\vec{m}}{\mapsto} \vec{y}''$. Similar to the previous case, a closing macro-tag marks the time when a top separator has to be removed. After this operation, the next position $(\vec{x}', \vec{y}')$ should be a position owned by Repairer. Therefore, the transition function κ_2 decides the next sequence of rules $\vec{m}$ given by W, derives the next stack of Generator $\vec{y}' \overset{\vec{m}}{\mapsto} \vec{y}''$, and outputs the closing macro-tag $\langle / \hat{C}^{\mathsf{suffix}} \rangle$ followed by the sequence of rules $\vec{m}$.

– If $e = \langle\, p \cdot \hat{C}^{\mathsf{prefix}} / \rangle$ and $(\vec{x}, \vec{y}) = (X \cdot \vec{x}', Y \cdot \vec{y}')$ such that X is horizontal and $p \in X$, then we define:

$$\kappa_2((X \cdot \vec{x}', Y \cdot \vec{y}'), \langle\, p \cdot \hat{C}^{\mathsf{prefix}} \rangle) = ((X \cdot \vec{x}', Y \cdot \vec{y}'), \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,/ \rangle)$$

such that $V(p, \hat{C}^{\mathsf{prefix}}, Y) = (q_1, q_2)$. This case is analogous to the previous one when the input was an opening macro-tag.

- Ω_2 is the final output function defined only for the pair $(\epsilon, \epsilon) \in Z_2$ such that $\Omega_2((\epsilon, \epsilon)) = \epsilon$. Therefore, the only final state of $\mathcal{Z}_2$ is when the two stacks become empty.

We omit the full specification of transducer $\mathcal{Z}_2$. The complete specification is rather technical and one can easily show from the definition of its output that this can be implemented by a sequential transducer. Indeed, to determine states q_1 and q_2 when a macro-tag $\langle\, p \cdot \hat{C}^{\mathsf{prefix}} \rangle$ (or $\langle\, p \cdot \hat{C}^{\mathsf{prefix}} \,/ \rangle$) is consumed, $\mathcal{Z}_2$ can obtained state q_1 from Lemma 5.4.4. Furthermore, state q_2 can be obtained by running $\mathcal{T}$ starting from q_1 over $\hat{C}^{\mathsf{prefix}}$. One has to notice here that to compute the function $\hat{\gamma}(q_1, C)$ in a streaming fashion only the prefix $\hat{C}^{\mathsf{prefix}}$ is needed. To be more precise, one can run $\mathcal{T}$ as a top-down VPA over $\hat{C}^{\mathsf{prefix}}$ and the top element of the stack will be exactly q_2, that is, $q_2 = \mathsf{top}(\hat{\gamma}(q_1, \hat{C}^{\mathsf{prefix}}))$. Therefore, by the previous arguments $\mathcal{Z}_2$ can be specified by a sequential transducer.

In the last part of this subsection we define recursively when a sequence in $(\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}))^*$ is a *Repairer sequence* starting from a sequence in $\mathsf{SCC}(\mathcal{T})^*$ and next we show that $\mathcal{Z}_2$ always produces valid Repairer sequences starting from the initial component Y_0.

Definition 5.4.5. *Assume that* $v \in (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}))^*$. *Then* v *is a* Repairer sequence *starting from* $\vec{y} = Y \cdot \vec{y}'$ *if, and only if, one of the following cases holds:*

- $v = Y \mapsto \epsilon$ *and* $\vec{y}' = \epsilon$, *or*
- $v = (Y \mapsto Y_1 Y_2) \cdot v'$ *and* v' *is a Repairer sequence starting from* $Y_1 Y_2 \cdot \vec{y}'$, *or*

- $v = \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,/\rangle \cdot v'$ *such that* $q_1, q_2 \in Y$, $\hat{\gamma}(q_1, C) = q_2$, *and* v' *is a Repairer sequence starting from* $\vec{y}$, *or*
- $v = \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,\rangle \cdot v' \cdot \langle\,/\, \hat{C}^{\mathsf{suffix}} \,\rangle \cdot v''$ *such that* $q_1, q_2 \in Y$, $\hat{\gamma}(q_1, C) = q_2$, *and* v' (v'') *is a Repairer sequence starting from* Y $(\vec{y}'$ *resp.).*

In particular, when $v \in \mathsf{moves}(\mathcal{T})^+$ we have that v is a Repairer sequence starting from Y iff v is a correct sequence of atomic rules that starting from Y reaches ϵ (i.e. $Y \overset{v}{\mapsto} \epsilon$). The next lemma shows that $\mathcal{Z}_2$ always produces a Repairer sequence starting from Y_0, the initial component in $\mathcal{T}$. The proof follows easily from the definition of $\mathcal{Z}_2$.

Lemma 5.4.6. *For every* $t \in \mathscr{L}(\mathcal{R})$, $\mathcal{Z}_2(\mathcal{Z}_1(\hat{t}))$ *is a Repairer sequence starting from* Y_0.

Proof. The proof goes by induction over the length of the Generator sequence output by $\mathcal{Z}_1(\hat{t})$. Toward this goal, we begin by pointing out some properties of W and $\mathcal{Z}_2$. First, we notice that one can always assume that W satisfies the following property:

$$W(\vec{x}, \vec{y}) \;=\; W(\vec{x} \cdot \lhd \cdot \vec{x}', \vec{y} \cdot \lhd \cdot \vec{y}')$$

for every $\vec{x} \in \mathsf{SCC}(\mathcal{R})^+$, $\vec{y} \in \mathsf{SCC}(\mathcal{T})^+$, $\vec{x}' \in (\mathsf{SCC}(\mathcal{R}) \cup \{\lhd\})^+$, and $\vec{y}' \in (\mathsf{SCC}(\mathcal{T}) \cup \{\lhd\})^+$. Informally, the sequence of moves for Repairer from a position $\langle\!\langle\, \vec{x} \cdot \lhd \cdot \vec{x}', \vec{y} \cdot \lhd \cdot \vec{y}' \,\rangle\!\rangle$ are determined by the prefixes $(\vec{x}, \vec{y})$. We assume this property without giving an explicit proof. Indeed, it can be easily proved by using the restrictions imposed by the separator $\lhd$ over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Second, we can note that the transition function κ_2 of $\mathcal{Z}_2$ directly depends on the winning strategy W. In particular, by the previous arguments one can easily show that for every $\vec{x} \in \mathsf{SCC}(\mathcal{R})^+$, $\vec{y} \in \mathsf{SCC}(\mathcal{T})^+$, $\vec{x}' \in (\mathsf{SCC}(\mathcal{R}) \cup \{\lhd\})^+$, $\vec{y}' \in (\mathsf{SCC}(\mathcal{T}) \cup \{\lhd\})^+$, it holds that:

$$(\vec{x} \cdot \lhd \cdot \vec{x}', \vec{y} \cdot \lhd \cdot \vec{y}') \xrightarrow{w/v}_{\mathcal{Z}_2} (\lhd \cdot \vec{x}', \lhd \cdot \vec{y}') \quad \text{iff} \quad (\vec{x}, \vec{y}) \xrightarrow{w/v}_{\mathcal{Z}_2} (\epsilon, \epsilon) \tag{5.1}$$

where $w \in (\Phi_1^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{R}))^*$ and $v \in (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}))^*$. We use Equation 5.1 for the inductive step of this proof.

Assume that $w \in (\Phi_1^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{R}))^*$, $\vec{x} \in \mathsf{SCC}(\mathcal{R})^+$, and $\vec{y} \in \mathsf{SCC}(\mathcal{T})^+$. By induction over the size of w, we prove that if w is a Generator sequence starting from $\vec{x}$ and $[\![\vec{x}, \vec{y}]\!]$ is a winning position for Repairer, then $(\vec{x}, \vec{y}) \xrightarrow{w/v}_{\mathcal{Z}_2} (\epsilon, \epsilon)$ and v is a Repairer sequence starting from $\vec{y}$. For the base case, suppose that $|w| = 1$. Given that w is a Generator sequence starting from $\vec{x}$ and $|w| = 1$, the only possibility is that $w = X \mapsto \epsilon$ and $\vec{x} = X$ for some $X \in \mathsf{SCC}(\mathcal{R})^*$. By the definition of κ_2, we have that $\kappa_2((X, \vec{y}), X \mapsto \epsilon) = ((\epsilon, \epsilon), \vec{m})$ where $\vec{m}$ is the sequence of rules in $\mathsf{moves}(\mathcal{T})$ that pops all the elements of $\vec{y}$. We clearly have $(\vec{x}, \vec{y}) \xrightarrow{w/v}_{\mathcal{Z}_2} (\epsilon, \epsilon)$ with $v = \vec{m}$. Hence, we conclude that v is a Repairer sequence starting from $\vec{y}$.

For the inductive case, suppose that the hypothesis holds for every $\vec{x} \in \mathsf{SCC}(\mathcal{R})^+$, $\vec{y} \in \mathsf{SCC}(\mathcal{T})^+$, and $|w| < i$. We next prove that the inductive hypothesis also holds for every $\vec{x} \in \mathsf{SCC}(\mathcal{R})^+$, $\vec{y} \in$

$\mathsf{SCC}(\mathcal{T})^+$, and $|w| = i$. Lets denote $\vec{x} = X \cdot \vec{x}'$ and $\vec{y} = Y \cdot \vec{y}'$. The proof is shown by case analysis on the structure of the Generator sequence w. However, we only show the most interesting cases when $w = \langle\, p \cdot \hat{C}^{\mathsf{prefix}} \,\rangle \cdot w' \cdot \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle \cdot w''$. The other cases can be deduced with the same inductive argument. Therefore, assume that $w = \langle\, p \cdot \hat{C}^{\mathsf{prefix}} \,\rangle \cdot w' \cdot \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle \cdot w''$ and X is non-horizontal such that $p \in X$, $\hat{\delta}(p, C) \in X$, and w' (w'') is a Generator sequence starting from X ($\vec{x}'$ resp.) (Definition 5.4.5, item (4)). For the starting macro-tag $\langle\, p \cdot \hat{C}^{\mathsf{prefix}} \,\rangle$ in w, we have the following transition:

$$\kappa_2((X \cdot \vec{x}', Y \cdot \vec{y}'), \langle\, p \cdot \hat{C}^{\mathsf{prefix}} \,\rangle) = ((X \cdot \triangleleft \cdot \vec{x}', Y \cdot \triangleleft \cdot \vec{y}'), \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,\rangle)$$

We know by definition that w' is a Generator sequence starting from X, $[\![X, Y]\!]$ is a winning position for Repairer, and $|w'| < i$. Then by induction we have that $(X, Y) \xrightarrow{w'/v'}_{\mathcal{Z}_2} (\epsilon, \epsilon)$ and v' is a Repairer sequence starting from $\vec{Y}$. By Equation 5.1 we have:

$$(X \cdot \triangleleft \cdot \vec{x}', Y \cdot \triangleleft \cdot \vec{y}') \xrightarrow{w'/v'}_{\mathcal{Z}_2} (\triangleleft \cdot \vec{x}', \triangleleft \cdot \vec{y}').$$

The next incoming letter is a close macro-tag $\langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle$ and we have the following transition:

$$\kappa_2((\triangleleft \cdot \vec{x}', \triangleleft \cdot \vec{y}'), \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle) = ((\vec{x}', \vec{y}''), \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle \cdot \vec{m})$$

where $\vec{m}$ is the sequence of moves such that $\vec{y}' \overset{\vec{m}}{\mapsto} \vec{y}''$ and $[\![\vec{x}', \vec{y}'']\!]$ is a winning position for Repairer. Now we have w'' is a Generator sequence for $\vec{x}'$, $[\![\vec{x}', \vec{y}'']\!]$ is a winning position for Repairer, and $|w''| < i$. By inductive hypothesis we get that $(\vec{x}', \vec{y}'') \xrightarrow{w''/v''}_{\mathcal{Z}_2} (\epsilon, \epsilon)$ and v'' is a Repairer sequence starting from $\vec{y}''$. Notice that if $\vec{y}' \overset{\vec{m}}{\mapsto} \vec{y}''$ and v'' is a Repairer sequence starting from $\vec{y}''$ then $\vec{m} \cdot v''$ is a Repairer sequence from $\vec{y}'$. Putting everything together, we get that:

$$v = \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,\rangle \cdot v' \cdot \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle \cdot \vec{m} \cdot v''$$

is the output of $\mathcal{Z}_2$ such that $(\vec{x}, \vec{y}) \xrightarrow{w/v}_{\mathcal{Z}_2} (\epsilon, \epsilon)$ and v is a Repairer sequence starting from $\vec{y}$. Thus, we conclude that the inductive hypothesis holds for the case when $w = \langle\, p \cdot \hat{C}^{\mathsf{prefix}} \,\rangle \cdot w' \cdot \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle \cdot w''$.

For the last step of the proof we show that $\mathcal{Z}_2(\mathcal{Z}_1(\hat{t}))$ is a Repairer sequence starting from Y_0 for every $t \in \mathscr{L}(\mathcal{R})$. This easily follows from the above proof and Lemma 5.4.2. Indeed, the initial function ι_2 of $\mathcal{Z}_2$ outputs $\vec{m}_0 \in \mathsf{moves}(\mathcal{T})^*$ such that $Y_0 \overset{\vec{m}_0}{\mapsto} \vec{y}$ and $[\![X_0, \vec{y}]\!]$ is a winning position for Repairer. By the previous result, we know that $(X_0, \vec{y}) \xrightarrow{w/v} (\epsilon, \epsilon)$ where $w = \mathcal{Z}_1(\hat{t})$ and, thus, $\mathcal{Z}_2(\mathcal{Z}_1(\hat{t})) = \vec{m}_0 \cdot v$ is a Repairer sequence starting from Y_0. □

5.4.3 Gluing the output

In this subsection we define the behaviour of the last transducer $\mathcal{Z}_3$. We suppose that $\mathcal{Z}_3$ receives the output of $\mathcal{Z}_2$, namely, a sequence of atomic rules in $\mathsf{moves}(\mathcal{T})$ and macro-tags in Φ_2^{tags}. The purpose of $\mathcal{Z}_3$ is to receive the context chunks of t coming inside macro-tags and glues them in a single serialized tree from $\mathscr{L}(\mathcal{T})$. To do this, $\mathcal{Z}_3$ interprets the atomic rules of Repairer as guides for pasting the output tree.

We specify $\mathcal{Z}_3$ as a transducer with input alphabet $\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T})$ and output alphabet Δ. We first give all the specification of $\mathcal{Z}_3$ and then we prove its correctness. A state of $\mathcal{Z}_3$ is represented by a stack over states in Q, suffixes of contexts over Δ (i.e $\hat{C}^{\mathsf{suffix}}$ for $C \in \mathcal{C}_\Delta^{\downarrow}$), and separators ($\triangleleft$). Formally, we define the states of $\mathcal{Z}_3$ as stacks constructed from the set:

$$Z_3 = Q \uplus \{\hat{C}^{\mathsf{suffix}} \mid C \in \mathcal{C}_\Delta^{\downarrow}\} \uplus \{\triangleleft\}.$$

For example, the following is a stack in $\mathcal{Z}_3$: $u = q \cdot [\bar{a}a\bar{a}\bar{b}] \cdot \triangleleft \cdot q' \cdot [\bar{b}]$. We write suffixes of context over $(\Delta \uplus \hat{\Delta})^*$ between brackets to represent that they are single elements. The meaning of states Q and $\{\triangleleft\}$ inside a stack $\vec{z} \in Z_3^*$ is to store a Repairer stack interleaved inside $\vec{z}$. Formally, for any $\vec{z} \in Z_3^*$ we denote by $[\vec{z}]$ the projection of $\vec{z}$ over $\mathsf{SCC}(\mathcal{T}) \cup \{\triangleleft\}$ where $[\vec{z}] = [z_1]_{\mathcal{T}} \cdots [z_n]_{\mathcal{T}}$ and $z_1 \cdot \cdots \cdot z_n$ is the maximum subsequence of $\vec{z}$ over $Q \cup \{\triangleleft\}$ and $[\triangleleft]_{\mathcal{T}} = \triangleleft$. As an example, for the stack u we have $[u] = [q]_{\mathcal{T}} \cdot \triangleleft \cdot [q']_{\mathcal{T}}$. Note that $[\vec{z}]$ does not contain any suffix in $\{\hat{C}^{\mathsf{suffix}} \mid C \in \mathcal{C}_\Delta^{\downarrow}\}$. Additionally, suffixes of contexts in Z_3^* are necessary to store words that must be output in the future. In particular, when a stack $\vec{z} = [\hat{C}^{\mathsf{suffix}}] \cdot \vec{z}'$ and the top element is popped, $\mathcal{Z}_3$ outputs $\hat{C}^{\mathsf{suffix}}$.

We define the transitions of $\mathcal{Z}_3$ as a function $\kappa_3 : Z_3 \times (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}) \cup \{\epsilon\}) \to Z_3^* \times \Delta^*$. This means that if $\kappa_3(z, a) = (\vec{z}, u)$ and $z \cdot \vec{z}'$ is the current state of $\mathcal{Z}_3$, then when reading $a \in (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}) \cup \{\epsilon\})$ the transducer outputs u and updates its stack to $\vec{z} \cdot \vec{z}'$. Note that we also allow ϵ-moves in the transition function of $\mathcal{Z}_3$. Nevertheless, it would be clear from the definition of $\mathcal{Z}_3$ that its behaviour is deterministic despite the existence of ϵ-moves. In order to define κ_3 formally, we need to introduce some notation. For every $Y \in \mathsf{SCC}(\mathcal{T})$ and every $q_1, q_2 \in Y$ we fix a context $C_{q_1,q_2} \in \mathcal{C}_\Delta^{\downarrow}$ such that $\hat{\gamma}(q_1, C_{q_1,q_2}) = q_2$. Moreover, for every $q \in Q$ we fix a tree t_q such that there exists an accepting run of $\mathcal{T}$ over t_q using q as initial state. We now define the transition function κ_3 over $Z_3 \times (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}) \cup \{\epsilon\})$ as follows.

1. If $q \in Q$ and $(Y \mapsto Y_1 \cdot Y_2) \in \mathsf{moves}(\mathcal{T})$ with $q \in Y$, let $q' \in Y$, $q_1 \in Y_1$, $q_2 \in Y_2$ and $a \in \Delta$ such that $\gamma(q', a) = (q_1, q_2)$ (by definition of $Y \mapsto Y_1 Y_2$). Then:

$$\kappa_3(q, Y \mapsto Y_1 \cdot Y_2) = (\ q_1 \cdot [\bar{a}] \cdot q_2 \cdot [\hat{C}^{\mathsf{suffix}}_{q,q'}],\ \ \hat{C}^{\mathsf{prefix}}_{q,q'} \cdot a\ )$$

2. If $q \in Q$ and $(Y \mapsto \epsilon) \in \mathsf{moves}(\mathcal{T})$ with $q \in Y$, then:

$$\kappa_3(q, Y \mapsto \epsilon) = (\ \epsilon,\ \ \hat{t}_q\ )$$

3. If $q \in Q$ and $\langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2\, \rangle \in \Phi_2^{\mathsf{open}}$ with $q_1, q_2 \in [q]_{\mathcal{T}}$ and $C \in \mathcal{C}_\Sigma^{\downarrow}$, then:

$$\kappa_3(q, \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2\, \rangle) = (\ q_2 \cdot \triangleleft \cdot [\hat{C}^{\mathsf{suffix}}_{q,q_1}],\ \ \hat{C}^{\mathsf{prefix}}_{q,q_1} \cdot \hat{C}^{\mathsf{prefix}}\ )$$

4. If $q \in Q$ and $\langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2\, /\rangle \in \Phi_2^{\mathsf{leaf}}$ with $q_1, q_2 \in [q]_{\mathcal{T}}$ and $C \in \mathcal{C}_\Sigma^{\downarrow}$, then:

$$\kappa_3(q, \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2\, /\rangle) = (\ q_2 \cdot [\hat{C}^{\mathsf{suffix}}_{q,q_1}],\ \ \hat{C}^{\mathsf{prefix}}_{q,q_1} \cdot \hat{C}^{\mathsf{prefix}}\ )$$

5. If $\langle / \hat{C}^{\mathsf{suffix}} \rangle \in \Phi_2^{\mathsf{close}}$ with $C \in \mathcal{C}_\Sigma^\downarrow$, then:

$$\kappa_3(\triangleleft, \langle / \hat{C}^{\mathsf{suffix}} \rangle) = (\ \epsilon,\ \ \hat{C}^{\mathsf{suffix}}\ )$$

6. If $C \in \mathcal{C}_\Delta^\downarrow$, then:

$$\kappa_3([\hat{C}^{\mathsf{suffix}}], \epsilon) = (\ \epsilon,\ \ \hat{C}^{\mathsf{suffix}}\ )$$

Therefore, we define $\mathcal{Z}_3$ formally as a tuple $\mathcal{Z}_3 = (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}), \Delta, Z_3, \kappa_3, q_0, \Omega_3)$ such that $\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T})$ and Δ are the input and output alphabet, respectively, Z_3 is the set of states, κ_3 is the transition function defined as above, q_0 is the initial stack (i.e. initial control state of $\mathcal{T}$), and $\Omega_3 : Z_3^* \to \Delta^*$ is the final output function such that for every $\vec{z} \in \{\hat{C}^{\mathsf{suffix}} \mid C \in \mathcal{C}_\Delta^\downarrow\}^*$ it holds that $\Omega_3(\vec{z}) = \vec{z}'$ where $\vec{z}'$ is equal to $\vec{z}$ viewed as a word over Δ^*.

For the last part of this subsection, we prove that $\mathcal{Z}_3$ outputs valid trees in $\mathscr{L}(\mathcal{T})$. Specifically, if $v \in (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}))^*$ is a Repairer sequence starting from Y_0 (the initial component of $\mathcal{T}$) then $\mathcal{Z}_3(v) \in \mathscr{L}(\mathcal{T})$. This lemma together with Lemma 5.4.2 and Lemma 5.4.6 will show that the cascade composition of $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$ repairs trees from $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$.

Lemma 5.4.7. *For every Repairer sequence $v \in (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}))^*$ starting from Y_0 (the initial component of $\mathcal{T}$):*

$$\mathcal{Z}_3(v) \in \mathscr{L}(\mathcal{T}).$$

Proof. This lemma is proved by induction over the length of v. Towards this goal, we reformulate the statement in two ways. First, in the rest of the proof we see $\mathcal{T}$ as a top-down VPA (see the definition of a top-down VPA in Section 5.1) denoted by $\hat{\mathcal{T}}$ and, second, we parametrize the statement for any component $Y \in \mathsf{SCC}(\mathcal{T})$. Therefore, the following claim proves our lemma.

Claim 1. *Let $Y \in \mathsf{SCC}(\mathcal{T})$ and $q \in Y$. For every Repairer sequence $v \in (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}))^*$ starting from Y, it holds that $q \xrightarrow{v/u}_{\mathcal{Z}_3} \epsilon$ for some $u \in \Delta^*$ and $q \xrightarrow{u}_{\hat{\mathcal{T}}} g$ for some $g \in G$.*

We restate the definition of a Repairer sequence in an equivalent way but only starting from a single element in $\mathsf{SCC}(\mathcal{T})$. It is straightforward to show that both definitions are equivalent.

Definition 5.4.8. *Assume that $v \in (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}))^*$. Then v is a* Repairer sequence *starting from Y if, and only if, one of the following cases holds:*

- *$v = Y \mapsto \epsilon$, or*
- *$v = (Y \mapsto Y_1 Y_2) \cdot v'$ such that $v' = v_1 \cdot v_2$ and v_1 (v_2) is a Repairer sequence starting from Y_1 (Y_2 resp.), or*
- *$v = \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,/\rangle \cdot v'$ such that $q_1, q_2 \in Y$, $\hat{\gamma}(q_1, C) = q_2$, and v is a Repairer sequence from Y, or*

- $v = \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,\rangle \cdot v' \cdot \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle$ *such that* $q_1, q_2 \in Y$, $\hat{\gamma}(q_1, C) = q_2$, *and* v' *is a Repairer sequence starting from* Y.

In the rest of the proof, we use above definition of Repairer sequence to prove our claim.

We show Claim 1 by induction over the length of v. For the base case, for any $Y \in \mathsf{SCC}(\mathcal{T})$ and $q \in Y$, we have that $v = Y \mapsto \epsilon$ is a Repairer sequence starting from Y. By definition of $\mathcal{Z}_3$, we know that $\kappa_3(q, Y \mapsto \epsilon) = (\epsilon, \hat{t}_q)$ such that there exists an accepting run of $\mathcal{T}$ over t_q using q as initial state. In terms of top-down VPA, this means that $q \xrightarrow{\hat{t}_q}_{\hat{\mathcal{T}}} g$ for some $g \in G$ and the claim holds for the base case.

For the inductive case, we suppose that Claim 1 holds for any Repairer sequence v' starting from $Y' \in \mathsf{SCC}(\mathcal{T})$ of length $|v'| < i$ and we prove it for a Repairer sequence v starting from $Y \in \mathsf{SCC}(\mathcal{T})$ of length $|v| = i$. The proof follows by case analysis. For each case, let $Y \in \mathsf{SCC}(\mathcal{T})$ and $q \in Y$.

- If $v = Y \mapsto Y_1 Y_2 \cdot v'$ is a Repairer sequence starting from Y, then we know by Definition 5.4.8 that $v' = v_1 \cdot v_2$ and v_1 (v_2) is a Repairer sequence starting Y_1 (Y_2 resp.) for some $v_1, v_2 \in (\Phi_2^{\mathsf{tags}} \cup \mathsf{moves}(\mathcal{T}))^+$. By definition of $\mathcal{Z}_3$ and $Y \mapsto Y_1 Y_2$, we know that there exists $q' \in Y$, $q_1 \in Y_1$, $q_2 \in Y_2$ and $a \in \Delta$ such that $\gamma(q', a) = (q_1, q_2)$ and:

$$\kappa_3(q, Y \mapsto Y_1 \cdot Y_2) = (\; q_1 \cdot [\bar{a}] \cdot q_2 \cdot [\hat{C}^{\mathsf{suffix}}_{q,q'}],\;\; \hat{C}^{\mathsf{prefix}}_{q,q'} \cdot a \;)$$

For the sake of simplification, denote $m = Y \mapsto Y_1 Y_2$, $C = \hat{C}^{\mathsf{prefix}}_{q,q'}$ and $\bar{C} = \hat{C}^{\mathsf{suffix}}_{q,q'}$. By the definition of C and top-down VPA $\hat{\mathcal{T}}$, we have $\hat{\gamma}(q, C) = q' \cdot \vec{q}$ for some $\vec{q} \in Q^*$ and $\hat{\gamma}(g \cdot \vec{q}, \bar{C}) \in G$ for every $g \in G$. Given that v_k is a Repairer sequence starting from Y_k, $q_k \in Y_k$, and $|v_k| < i$ for $k \in \{1, 2\}$, we have by inductive hypothesis that $q_k \xrightarrow{v_k/u_k}_{\mathcal{Z}_3} \epsilon$ for $u_k \in \Delta^*$ and $q_k \xrightarrow{u_k}_{\hat{\mathcal{T}}} g_k$ for some $g_k \in F$. Then we have the following derivation:

$$\begin{array}{lccccccccccc}
\mathcal{Z}_3: & q & \xrightarrow{m, C \cdot a} & q_1 \cdot [\bar{a}] \cdot q_2 \cdot [\bar{C}] & \xrightarrow{v_1, u_1} & [\bar{a}] \cdot q_2 \cdot [\bar{C}] & \xrightarrow{\epsilon, \bar{a}} & q_2 \cdot [\bar{C}] & \xrightarrow{v_2, u_2} & [\bar{C}] & \xrightarrow{\epsilon, \bar{C}} & \epsilon \\
\hat{\mathcal{T}}: & q & \xrightarrow{C \cdot a} & q_1 \cdot q_2 \cdot \vec{q} & \xrightarrow{u_1} & g_1 \cdot q_2 \cdot \vec{q} & \xrightarrow{\bar{a}} & q_2 \cdot \vec{q} & \xrightarrow{u_2} & g_2 \cdot \vec{q} & \xrightarrow{\bar{C}} & g
\end{array}$$

where $g \in G$ is a final state. We conclude that the claim holds for $v = Y \mapsto Y_1 Y_2 \cdot v'$.

- If $v = \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,\rangle \cdot v' \cdot \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle$ is a Repairer sequence starting from Y then $q_1, q_2 \in Y$, $\hat{\gamma}(q_1, C) = q_2$, and v' is a Repairer sequence starting from Y by Definition 5.4.8. From the transitions of $\mathcal{Z}_3$ we get that:

$$\kappa_3(q, \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,\rangle) = (\; q_2 \cdot \triangleleft \cdot [\hat{C}^{\mathsf{suffix}}_{q,q_1}],\;\; \hat{C}^{\mathsf{prefix}}_{q,q_1} \cdot \hat{C}^{\mathsf{prefix}} \;)$$

In order to simplify the notation, let $l = \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,\rangle$, $\bar{l} = \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle$, $C = \hat{C}^{\mathsf{prefix}}$, $\bar{C} = \hat{C}^{\mathsf{suffix}}$, $B = \hat{C}^{\mathsf{prefix}}_{q,q_1}$, and $\bar{B} = \hat{C}^{\mathsf{suffix}}_{q,q_1}$. Given that we see $\mathcal{T}$ as a top-down VPA, we know that $\hat{\gamma}(q_1, C) = q_2 \cdot \vec{q}$ for some $\vec{q} \in Q^*$ and $\hat{\gamma}(g \cdot \vec{q}, \bar{C}) \in G$ for every $g \in G$. The same holds for B where $\hat{\gamma}(q, B) = q_1 \cdot \vec{q}'$ for some $\vec{q}' \in Q^*$ and $\hat{\gamma}(g' \cdot \vec{q}', \bar{B}) \in G$ for every $g' \in G$. Since v' is a Repairer sequence starting

from Y, $q_2 \in Y$, and $|v'| < i$, we have by the inductive hypothesis that $q_2 \xrightarrow{v'/u}_{\mathcal{Z}_3} \epsilon$ for $u \in \Delta^*$ and $q_2 \xrightarrow{u}_{\hat{\mathcal{T}}} g$ for some $g \in G$. Similar to the previous case, we have the following derivation:

$$\begin{array}{lccccccccc} \mathcal{Z}_3: & q & \xrightarrow{l,B\cdot C} & q_2 \cdot \triangleleft \cdot [\bar{B}] & \xrightarrow{v',u} & \triangleleft \cdot [\bar{B}] & \xrightarrow{\bar{l},\bar{C}} & [\bar{B}] & \xrightarrow{\epsilon,\bar{B}} & \epsilon \\ \hat{\mathcal{T}}: & q & \xrightarrow{B\cdot C} & q_2 \cdot \vec{q} \cdot \vec{q}' & \xrightarrow{u} & g \cdot \vec{q} \cdot \vec{q}' & \xrightarrow{\bar{C}} & g' \cdot \vec{q}' & \xrightarrow{\bar{B}} & g'' \end{array}$$

where $g'' \in G$ is a final state. We conclude that the claim holds for $v = \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,\rangle \cdot v' \cdot \langle\!/\, \hat{C}^{\mathsf{suffix}} \,\rangle$.

- The case where $v = \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,/\rangle \cdot v'$ is similar to the above case. The only difference is that in this case l is a leaf macro-tag and $\bar{l} = \epsilon$ which implies that $\bar{C} = \epsilon$ and $\hat{\gamma}(q_1, C) = q_2$. By the same above arguments, we get the following derivation:

$$\begin{array}{lccccccc} \mathcal{Z}_3: & q & \xrightarrow{l,B\cdot C} & q_2 \cdot [\bar{B}] & \xrightarrow{v',u} & [\bar{B}] & \xrightarrow{\epsilon,\bar{B}} & \epsilon \\ \hat{\mathcal{T}}: & q & \xrightarrow{B\cdot C} & q_2 \cdot \vec{q} & \xrightarrow{u} & g \cdot \vec{q} & \xrightarrow{\bar{B}} & g' \end{array}$$

where $g' \in G$ is a final state. We conclude that the claim also holds for $v = \langle\, q_1 \cdot \hat{C}^{\mathsf{prefix}} \cdot q_2 \,/\rangle \cdot v'$.

For any Repairer sequence the inductive hypothesis holds and, thus, Claim 1 is proved. □

5.4.4 Correctness of the repair strategy

In this last subsection we gather all previous results for $\mathcal{Z}_1$, $\mathcal{Z}_2$, and $\mathcal{Z}_3$ to show that the cascade composition $\mathcal{Z} = \mathcal{Z}_3 \circ \mathcal{Z}_2 \circ \mathcal{Z}_1$ is a tree edit transducer that repairs each tree in $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$ with uniformly bounded cost. First of all, by combining Lemma 5.4.2, Lemma 5.4.6, and Lemma 5.4.7 we can conclude that $\mathcal{Z}$ is a repair transducer from $\mathcal{R}$ into $\mathcal{T}$, that is, $\mathcal{Z}(t) \in \mathscr{L}(\mathcal{T})$ for every $t \in \mathscr{L}(\mathcal{R})$. Furthermore, from the construction of $\mathcal{Z}_1$ and $\mathcal{Z}_3$, one can show that the matching relation between unmodified tags from the input is preserved. In fact, the contexts extracted by $\mathcal{Z}_1$ from the input tree are not modified by $\mathcal{Z}_2$ and, furthermore, they are connected back in $\mathcal{Z}_3$. Therefore, we can conclude that the cascade composition $\mathcal{Z} = \mathcal{Z}_3 \circ \mathcal{Z}_2 \circ \mathcal{Z}_1$ is a tree edit transducer from $\mathcal{R}$ into $\mathcal{T}$.

To complete the proof, we show that the cost of $\mathcal{Z}$ is uniformly bounded for all trees in $\mathcal{R}$. From Lemma 5.4.2 we know that the output of $\mathcal{Z}_1$ is a complete play for Generator starting from X_0. One can easily show from the definition of $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ that the numbers of moves from Generator is bounded by $O(2^{|\mathsf{SCC}(\mathcal{R})|})$. This implies that the number of nodes deleted by $\mathcal{Z}_1$ over an input tree is bounded by $O(2^{|\mathsf{SCC}(\mathcal{R})|})$. Let $|W|$ be the longest sequence of moves output by the winning strategy W in a play over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Then we have that the number of prefix rewriting steps outputted by $\mathcal{Z}_2$ is bounded by $O(2^{|\mathsf{SCC}(\mathcal{R})|} \cdot |W|)$. Each prefix rewriting step read by $\mathcal{Z}_3$ implies adding a context of size bounded by the size of the target language, in fact, it is in $O(2^{|\mathcal{T}|})$. Therefore, the number of nodes added by $\mathcal{Z}_3$ to the final output is bounded by $O(2^{|\mathsf{SCC}(\mathcal{R})|+|\mathcal{T}|} \cdot |W|)$ for any tree $t \in \mathscr{L}(\mathcal{R})$. We conclude that the overall cost of $\mathcal{Z}$ is bounded by $O(2^{|\mathsf{SCC}(\mathcal{R})|+|\mathcal{T}|} \cdot |W|)$. Given that $\mathcal{R}$ and $\mathcal{T}$ are fixed and the winning strategy depends only on $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, we conclude that the cost of $\mathcal{Z}$ is uniformly bounded for all trees in $\mathscr{L}(\mathcal{R})$.

5.5 From repairs to simulation games

For this direction, we assume the existence of a tree edit transducer $\mathcal{Z}$ that implements a streaming repair strategy of $\mathscr{L}(\mathcal{R})$ into $\mathscr{L}(\mathcal{T})$, with uniformly bounded cost, and we derive from that the existence of a strategy for Repairer to win the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. The idea of Repairer's strategy is to construct a sequence of prefixes $u_1, \ldots, u_i$ of a tree $t \in \mathscr{L}(\mathcal{R})$ (i.e. prefixes of the serialization of t) incrementally following the moves of Generator over its prefix-rewriting system in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. That is, given the next move m_i of Generator in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ we extend the prefix u_i with a string u_{m_i} such that $u_{i+1} = u_i \cdot u_{m_i}$ and u_{i+1} is the prefix of some $t \in \mathscr{L}(\mathcal{R})$. Further, after each move m_i of Generator and after defining w_{i+1}, we use the output of $\mathcal{Z}$ over w_{i+1} to decide the next move of Repairer, i.e., we run $\mathcal{T}$ as a top-down VPA over $\mathcal{Z}(w_{i+1})$ and then check the configuration $\vec{y}$ (i.e. the stack) of $\mathcal{T}$ after reaching the end of $\mathcal{Z}(w_{i+1})$. By converting $\vec{y} = q_1 \ldots q_n$ into strongly connected components $Y_1 \ldots Y_n$ of $\mathcal{T}$, we will define $Y_1 \ldots Y_n$ to be the content of the prefix-rewriting system for Repairer in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ and, furthermore, this will define its next move in the simulation game. Since $\mathcal{Z}$ is a bounded streaming repair strategy, this will imply that the informal strategy above is a winning strategy over the simulation game $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

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

$$\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y).$$

Indeed, if this were not the case, then the prefix u could be expanded by an iteration of a context that stays within the same component X and, unless the corresponding output induces a change of component in the target automaton, each context would have to be repaired into Y, thus resulting in unbounded repair cost (see Lemma 5.5.2 for a formal definition). Thanks to the above invariant, one can abstract the output of the transducer $\mathcal{Z}$ into valid plays over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, which turn out to be winning for Repairer.

Before going into the details of the proof, we want to point out that the ideas and structure of this proof are similar to the ones presented in Theorem 2.3.3 and 4.3.6. For example, the idea of extending a context inside a component in order to "stabilize" the output of $\mathcal{Z}$ into a component in the target language were already presented in the proof of Theorem 2.3.3. Also, the construction followed by the strategy of Repairer will be similar to the proof of Theorem 4.3.6 (presented in Section 4.4.2) where fingerprint contexts where glue together in a specific way to force the repair strategy (in this case $\mathcal{Z}$) to give the desire result. We suggest the reader to keep in mind these proofs when following the proof below.

We first need to introduce some new terminology for representing the structure of Repairer's strategy. Given an alphabet Σ and label $\bullet$, we call a tree C with labels in $\Sigma \cup \{\bullet\}$ a *multi-context* whenever $\bullet$ occurs only at leaves that have no right sibling. An example of a multi-context is $a(b(c,\bullet),b,\bullet)$. Notice that trees and contexts are special cases of multi-contexts. We denote by $\mathcal{M}_\Sigma$ the set of all multi-context in Σ and by $\mathcal{M}_\Sigma^n$ the set of all multi-context in Σ with n-placeholders. We define the composition of a multi-context C with a tree t, denoted by $C \circ t$, to be the multi-context obtained from the substitution of the first $\bullet$ (following the preorder traversal of C) with t in C. We can easily extend this definition for the composition between multi-contexts C_1 and C_2 by $C_1 \circ C_2$. In this case, the result is a multi-context with (n_1+n_2-1) $\bullet$-nodes where n_i is the number of $\bullet$-nodes in C_i for $i \in \{1,2\}$. Given two multi-context C_1 and C_2, we say that C_1 is a *prefix* of C_2 iff there exists a sequence of multi-contexts $C'_1,\ldots,C'_k$ such that $(\cdots(C_1 \circ C'_1) \circ \cdots) \circ C'_k = C_2$. Similar to the serialization of a context, for a multi-context C we denote by $\hat{C}^{\mathsf{prefix}}$ the prefix of $\hat{C}$ that ends immediately before the first occurrence of a $\bullet$-node. In case that C does not contain any $\bullet$-node (i.e. C is a tree), we have that $\hat{C}^{\mathsf{prefix}} = \hat{C}$. One can easily check that the composition between multi-contexts is in correspondence with the prefix of $\hat{C}$ in the sense that $\hat{C}^{\mathsf{prefix}}$ is a prefix of $\widehat{C \circ C'}^{\mathsf{prefix}}$ for any two multi-contexts C and C'. Furthermore, if C_1 is a prefix of C_2 then $\hat{C_1}^{\mathsf{prefix}}$ is a prefix of $\hat{C_2}^{\mathsf{prefix}}$. Usually we will use the $(\cdot)^{\mathsf{prefix}}$ operator after serializing a context, that is, after using the $\widehat{(\cdot)}$ operator. In order to simplify the notation, we usually omit the $\widehat{(\cdot)}$ operator before applying the $(\cdot)^{\mathsf{prefix}}$ operator. As an example, we write C^{prefix} instead of $\hat{C}^{\mathsf{prefix}}$ where C is a multi-context. We can also extend the transition function δ of a top-down tree automaton to a multi-context. Formally, we define the function $\delta^n : Q \times \mathcal{M}_\Sigma^n \to Q^n$ such that $\delta^n(q,C) = (q_1,\ldots,q_n)$ iff there exists a run ρ of $\mathcal{A}$ on C such that $\rho(\epsilon) = q$, $\rho(\bar{n_i}) = q_i$ with $\bar{n_i}$ the i-th placeholder of C (following the preorder traversal) and the final conditions are satisfied over each node $\bar{n}$ in t such that $t(\bar{n}) \neq \bullet$. Similarly to context, we make an abuse of notation and denote δ^n by δ if n is understand from the input multi-context.

Similar to the if-direction, we denote by $[q]_\mathcal{A}$ the strongly connected component of $q \in Q$ in $G_\mathcal{A}$ for a top-down tree automaton $\mathcal{A}$. We write $[q]$ for $[q]_\mathcal{A}$ if the tree automata is understood from the context. Furthermore, we extend this function from states to stack of states $q_1\cdots q_n \in Q^+$ by $[q_1\cdots q_n]_\mathcal{A} = [q_1]_\mathcal{A}\cdots[q_n]_\mathcal{A}$. During this proof, we usually use both representations of a top-down deterministic tree automata indistinguishable (i.e. top-down tree automata and top-down VPA). Given a tree automaton $\mathcal{A} = (\Sigma \uplus \bar{\Sigma}, Q, \delta, q_0, F)$, we denote by $\hat{\mathcal{A}} = (\Sigma, Q, \hat{\delta}, q_0, F)$ the top-down VPA equivalent to $\mathcal{A}$ (see Section 5.1). In particular, given a context C we have that $\hat{\delta}(q, \hat{C}^{\mathsf{prefix}}) = q' \cdot \vec{q}$ for some $q, q' \in Q$ and $\vec{q} \in Q^+$ whenever $\delta(q,C) = q'$.

For the rest of the proof, fix two top-down deterministic automata $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$ and $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ where $\hat{\mathcal{R}} = (\Sigma, P, \hat{\delta}, p_0, F)$ and $\hat{\mathcal{T}} = (\Delta, Q, \hat{\gamma}, q_0, G)$ are the VPA-versions of $\mathcal{R}$ and $\mathcal{T}$, respectively. We have all the ingredients to define a winning strategy for Repairer over

$\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Remember that for this direction, we suppose that there exists a tree repair transducer $\mathcal{Z}$ that repairs $\mathcal{R}$ into $\mathcal{T}$ with cost bounded by N (i.e. $\mathsf{cost}(\hat{t},\mathcal{Z}) < N$ for all $t \in \mathscr{L}(\mathcal{R})$) and we show how to define a winning strategy for Repairer over the simulation game $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

In this proof, we consider a simplified version of the simulation game $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ where Referee is not allowed to play. This means that only Generator and Repairer are allowed to make changes in their own stacks and that the $\triangleleft$-symbol does not appear in any of the two stacks. We will argue towards the end of the proof that the winning strategy constructed for Repairer is *safe* in the sense that this strategy does not take advantage on the absence of Referee and that the moves of Repairer implicitly follow the rules of Referee. Now, given a sequence of moves:

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

over the arena $\mathcal{G}_{\mathcal{R},\mathcal{T}}$, we define inductively a sequence of multi-contexts $C_1,\ldots,C_n$ over Σ such that for every $1 \leq i \leq n$ and $k = |\vec{x}_i|$ it holds that:

1. C_{i-1} is a prefix of C_i,
2. C_i has the same number of $\bullet$-nodes than the length of $\vec{x}_i$ ($C_i \in \mathcal{M}^k_\Sigma$),
3. there exists $p^n_1,\ldots p^n_k \in P$ such that $\delta(p_0, C_i) = (p^n_1,\ldots,p^n_k)$ and $\vec{x}_i = [p^n_1 \cdots p^n_k]$, and
4. letting $w = \mathcal{Z}(\hat{C}_i^{\mathsf{prefix}})$, we have $\vec{y}_i = [\hat{\gamma}(q_0, w)]$.

In the previous definition we suppose that C_0 is the empty context when we say that C_0 is a prefix of C_1 (C_0 was not formally in the sequence of multi-contexts). Intuitively, the sequence of multi-contexts $C_1,\ldots,C_n$ are in correspondence with the stacks $(\vec{x}_1,\vec{y}_1),\ldots,(\vec{x}_n,\vec{y}_n)$ in the sense that for each C_i the components of the sequence $\delta(p_0, C_i)$ is equal to $\vec{x}_i$ and the components of the stack of running $\mathcal{T}$ over the output $\mathcal{Z}(\hat{C}_i^{\mathsf{prefix}})$ is equal to $\vec{y}_i$.

Let us first explain how we will define the strategy of Repairer in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ by using the multi-contexts $C_1,\ldots,C_n$. Suppose that the game is in position $(\vec{x}_{n-1},\vec{y}_n)$ and Generator moves $[\![\, \vec{x}_{n-1}\,,\, \vec{y}_n \,]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle\, \vec{x}_n\,,\, \vec{y}_n \,\rangle\!\rangle$. The next step of Repairer is obtained by constructing a multi-context C_{n+1} from C_n and $\vec{x}_n$, and defining the next move of Repairer by $\langle\!\langle\, \vec{x}_n\,,\, \vec{y}_n \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_n\,,\, [\hat{\gamma}(q_0, w)] \,]\!]$ where $w = \mathcal{Z}(\hat{C}_i^{\mathsf{prefix}})$. We show that the move defined above is a valid move of Repairer in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ and that C_{n+1} satisfies the four properties defined above. By showing that C_{n+1} can always be constructed inductively from C_n and $\vec{x}_n$, we will prove that Repairer can always reply to the moves of Generator and, therefore, we will define a winning strategy of Repairer over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$.

Before going into details, we recall the definition of a *fingerprint context* for a component in $\mathcal{R}$ (given in the previous chapters). Similar to the proofs of Theorem 2.3.1 and 4.3.6, these fingerprint contexts are the building blocks in the construction of the multi-contexts $C_1,\ldots,C_n$. Basically, this lemma shows that given a component $X \in \mathsf{SCC}(\mathcal{R})$, one can find a context C_X that can be "pumped"

inside the language $\mathscr{L}(\mathcal{R} \mid X)$ (i.e., $C_X \circ \ldots \circ C_X \in \mathscr{L}(\mathcal{R} \mid X)$) and such that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$ iff $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$, for any component $Y \in \mathsf{SCC}(\mathcal{T})$. Also recall that a context C is *cyclic* for a component X if there exists a state $p \in X$ such that $\delta(p, C) = p$. The proof of this lemma was given in Lemma 2.3.2 for DFAs and in Lemma 4.4.8 for stepwise tree automata. We omit the proof in this case given that it is trivial following the ideas in Lemma 4.4.8.

Lemma 5.5.1. *For every $X \in \mathsf{SCC}(\mathcal{R})$, there exists a cyclic context $C_X \in \mathscr{L}(\mathcal{R} \mid X)$ such that, for every $Y \in \mathsf{SCC}(\mathcal{T})$,*

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

If X is non-horizontal, then C_X is non-horizontal as well.

Next, we construct the sequence of multi-contexts inductively. Before giving this construction, we need the following lemma that synthesizes the main technical part of the induction. This lemma says that if we repeat a context D a specific number of times M, the tree repair transducer $\mathcal{Z}$ will stabilize at some component Y and the language of context $\mathscr{L}(\mathcal{T} \mid Y)$ will contain D.

Lemma 5.5.2. *Suppose that $C \in \mathcal{M}_\Sigma^k$ and $p \in Q$ such that C is a prefix of some tree in $\mathscr{L}(\mathcal{R})$ and $\delta(p_0, C) = (p_1, \ldots, p_k)$ with $p = p_1$. Let D be a cyclic context over the state p, i.e. $\delta(p, D) = p$. Then there exist $M \in \mathbb{N}$ and $Y \in \mathsf{SCC}(\mathcal{T})$ such that for all $k \geq 0$:*

1. $\mathcal{Z}((C \circ D^{M+k})^{\mathsf{prefix}}) = \mathcal{Z}((C \circ D^M)^{\mathsf{prefix}}) \cdot (D^k)^{\mathsf{prefix}}$,
2. $\mathsf{top}(\hat{\gamma}(q_0, w)) \in Y$ *where* $w = \mathcal{Z}((C \circ D^{M+k})^{\mathsf{prefix}})$, *and*
3. $D \in \mathscr{L}(\mathcal{T} \mid Y)$.

Proof. Let $w_i = (C \circ D^i)^{\mathsf{prefix}}$ for every $i \in \mathbb{N}$. Each w_i is the prefix of a serialized tree in $\mathscr{L}(\mathcal{R})$. Indeed, C is a prefix of some tree in $\mathscr{L}(\mathcal{R})$ and one can easily show that $C \circ D^i$ is a prefix of some tree in $\mathscr{L}(\mathcal{R})$ as well. Remember that $\mathcal{Z}$ is a bounded tree repair transducer for serialized tree in $\mathcal{R}$. Then there must exist some $I \in \mathbb{N}$ such that $\mathsf{cost}(w_I, \mathcal{Z}) = \mathsf{cost}(w_i, \mathcal{Z})$ for every $i \geq I$. Otherwise, the cost of $\mathcal{Z}$ over the sequence of words $\{w_i\}_{i \in \mathbb{N}}$ would be unbounded which would lead to a contradiction. Given that the cost remains unchanged for every $i \geq I$, this implies that:

$$\mathcal{Z}(w_{I+k}) = \mathcal{Z}(w_I) \cdot (D^{\mathsf{prefix}})^k \quad \text{for every } k \in \mathbb{N}. \tag{$\star$}$$

Now consider the sequence of states $\{q_i\}_{i \in \mathbb{N}} \subseteq Q$ such that $\hat{\gamma}(q_0, \mathcal{Z}(w_i)) = q_i \cdot \vec{q}_i$ for some $\vec{q}_i \in Q^+$. Given that the sequence of states $\{q_i\}_{i \in \mathbb{N}}$ is infinite, we know by the pigeonhole principle that there must exist i, j such that $i < j \leq |\mathsf{SCC}(\mathcal{T})|$ and $[q_i] = [q_j]$. Further, one can easily show that for every $q \in Q$ and $C \in \mathcal{C}_\Sigma^\downarrow$ it holds that $[\gamma(q, D)] \leq_{\mathsf{SCC}(G_\mathcal{T})} [q]$. In other words, any state reach from a state q is in a lower or equal component than q with respect to the partial order $\leq_{\mathsf{SCC}(G_\mathcal{T})}$ of $G_\mathcal{R}$. Indeed,

this implies that $[q_i] = [q_j]$ for every $i \geq j$. From this argument we deduce that $[q_i] = [q_{i+1}]$ for every $i \geq I + |\mathsf{SCC}(\mathcal{T})|$ and a run of $\mathcal{T}$ over $\mathcal{Z}(w_i)$ stabilizes in the same component over $\mathcal{T}$ after i-repetitions of D with $i \geq I + |\mathsf{SCC}(\mathcal{T})|$. Let us define $M = I + |\mathsf{SCC}(\mathcal{T})|$ and $Y = [q_M]$. By the previous arguments, we can easily check that the first and second statements of the lemma hold.

We now show that $D \in \mathscr{L}(\mathcal{T} \mid Y)$. Considering $\mathcal{T}$ as a VPA, this is the same as showing that there exists $q, q' \in Y$ and $\vec{q} \in Q^+$ such that (a) $\hat{\gamma}(q, \hat{D}^{\mathsf{prefix}}) = q' \cdot \vec{q}$ and (b) $\hat{\gamma}(\vec{q}, \hat{D}^{\mathsf{suffix}}) \in F$. Towards this goal, define $J = I + |\mathsf{SCC}(\mathcal{T})| + (N+1)$ where N is the bound on the cost of $\mathcal{Z}$ and fix a tree $t \in \mathcal{T}_\Sigma$ such that $C \circ D^J \circ t$ is the prefix of a tree in $\mathscr{L}(\mathcal{R})$ or, formally, there is an accepting run of $\mathcal{T}$ over t starting from p. Furthermore, consider the set of words:

$$u_i = (C \circ D^J)^{\mathsf{prefix}} \cdot \hat{t} \cdot (\hat{D}^{\mathsf{suffix}})^i \quad \text{for every } i \leq N+1.$$

Given that the cost of $\mathcal{Z}$ is bounded by N, we must have that $\mathcal{Z}(u_{j+1}) = \mathcal{Z}(u_j) \cdot \hat{D}^{\mathsf{suffix}}$ for some $j \leq N+1$. In fact, if $\mathcal{Z}(u_{j+1}) \neq \mathcal{Z}(u_j) \cdot \hat{D}^{\mathsf{suffix}}$ for all $j \leq N+1$, then the cost of repairing any tree whose prefix is $C \circ D^J \circ t$ with $\mathcal{Z}$ is greater than N, which would lead to a contradiction. Thus, we have that $\mathcal{Z}(u_{j+1}) = \mathcal{Z}(u_j) \cdot \hat{D}^{\mathsf{suffix}}$. Recall that from the above arguments we also have that $\mathcal{Z}(w_{J-j}) = \mathcal{Z}(w_{J-(j+1)}) \cdot \hat{D}^{\mathsf{prefix}}$ (by $(\star)$). Since $\mathcal{Z}$ is a tree repair transducer, this implies that the $(J-j)$-iteration of D remains unmodified in the output of $\mathcal{Z}$. By the second proved statement of the lemma, this implies that there exists $q, q' \in Y$ and $\vec{q} \in Q^+$ such that $\hat{\gamma}(q, \hat{D}^{\mathsf{prefix}}) = q' \cdot \vec{q}$ and $\hat{\gamma}(\vec{q}, \hat{D}^{\mathsf{suffix}}) \in F$. This was to be shown. □

We now show how to construct the sequence of multi-context inductively. For the base case, assume that X is the initial component of $\mathcal{R}$ (i.e. $X = [p_0]$) such that $\vec{x}_0 = X$. By Lemma 5.5.1, we know that there exists a fingerprint context $C_X \in \mathscr{L}(\mathcal{R} \mid X)$ and a state $p \in X$ such that $\delta(p, C_X) = p$, and $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$ iff $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$ for every $Y \in \mathsf{SCC}(\mathcal{T})$. Since p and p_0 are in the same component X, we denote by C the context that connects both states, that is, $\delta(p_0, C) = p$. By considering C as a multi-context such that $\delta(p_0, C) = (p)$ and C_X the cyclic context over p, we know by Lemma 5.5.2 that there exists a constant $M \in \mathbb{N}$ and a component Y that satisfies the three properties stated in the lemma. Thus, we define the multi-context C_1 for the base case:

$$C_1 = C \circ C_X^{M+1}$$

It is trivial to show that C_1 is a prefix of the empty context $C_0 = \bullet$, $C_1 \in \mathcal{M}^1_\Sigma$, and $\vec{x}_0 = [p]$ where $\delta(p_0, C_1) = p$. For the last property of the induction, we denote $w = \mathcal{Z}(\hat{C_1}^{\mathsf{prefix}})$ and define $\vec{y}_1 = [\hat{\gamma}(q_0, w)]$. Notice first that $\vec{y}_0 \overset{\mathcal{T}}{\mapsto}{}^* \vec{y}_1$ is a valid prefix-rewriting rule associated with $\mathcal{T}$. Indeed, one can easily check that if $\hat{\gamma}(q \cdot \vec{q}, a) = q_1 \cdot q_2 \cdot \vec{q}$ (resp. $\hat{\gamma}(q \cdot \vec{q}, \bar{a}) = \vec{q}$) for $q, q_1, q_2 \in Q$ and $\vec{q} \in Q^+$, then $[q \cdot \vec{q}] \overset{\mathcal{T}}{\mapsto} [q_1 \cdot q_2 \cdot \vec{q}]$ ($[q \cdot \vec{q}] \overset{\mathcal{T}}{\mapsto} [\vec{q}]$) is a valid prefix-rewriting rule associated with $\mathcal{T}$. In particular, this implies that $\vec{y}_0 \overset{\mathcal{T}}{\mapsto}{}^* [\hat{\gamma}(q_0, w)] = \vec{y}_1$ is a prefix-rewriting rule associated with $\mathcal{T}$. To show that $\langle\!\langle\, \vec{x}_0, \vec{y}_0 \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_0, \vec{y}_1 \,]\!]$ is a valid move of Repairer, it remains only to argue that $\mathscr{L}(\mathcal{R} \mid$

$\mathsf{top}(\vec{x}_0)) \subseteq \mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{y}_1))$. In fact, we know by the second property of 5.5.2 that $\mathsf{top}(\hat{\gamma}(q_0, w)) \in Y$ which implies that $Y = \mathsf{top}(\vec{y}_1)$. Furthermore, the third property of Lemma 5.5.2 tells us that $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$. By combining this last fact with Lemma 5.5.1, we find that $\mathscr{L}(\mathcal{R} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$. We conclude that $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x}_0)) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}_1))$ given that $X = \mathsf{top}(\vec{x}_0)$ and $Y = \mathsf{top}(\vec{y}_1)$.

For the inductive argument, suppose that $(\vec{x}_{n-1}, \vec{y}_n)$ is the last move on the game, C_n satisfies the inductive hypothesis, and Generator makes a move $[\![\vec{x}_{n-1}, \vec{y}_n]\!] \overset{\mathsf{Gen}}{\mapsto} \langle\!\langle \vec{x}_n, \vec{y}_n \rangle\!\rangle$. Then we have two cases depending on whether the last move of Generator was a push (push case) or a pop (pop-case) over $\vec{x}_{n-1}$:

- Suppose that Generator does a push prefix rewriting rule over $\vec{x}_{n-1}$, that is, $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} X_1X_2 \cdot \vec{x}$ where $\vec{x}_{n-1} = X \cdot \vec{x}$ and $\vec{x}_n = X_1X_2 \cdot \vec{x}$. First, we know by the inductive hypothesis that there exists $p_1^n, \ldots, p_k^n \in P$ such that $\vec{x}_{n-1} = [p_1^n \cdot \ldots \cdot p_k^n]$ and $\delta(p_0, C_n) = (p_1^n, \ldots, p_k^n)$ with $k = |\vec{x}_{n-1}|$. In particular, we have that $p_1^n \in X$. Second, given that $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} X_1X_2 \cdot \vec{x}$ is a valid move, we also know that there must exist $p \in X$, $p_1 \in X_1$, $p_2 \in X_2$, and $a \in \Sigma$ such that $\delta(p, a) = (p_1, p_2)$. Notice that p and p_1^n are in the same component X. Thus, we can find a context C such that $\delta(p_1^n, C) = p$. Third, Lemma 5.5.1 tells us that we can find a fingerprint component C_{X_1} and a state p' such that $\delta(p', C_{X_1}) = p'$. Similar to p and p_1^n, we can find a context C' such that $\delta(p_1, C') = p'$ given that $p_1, p' \in X_1$.

  We have all the ingredients to define the inductive step C_{n+1}. Let us first define the multi-context:

  $$C'_{n+1} \quad = \quad C_n \circ C \circ (a(C', \bullet))$$

  From this definition, it is straightforward to check that the result $\delta(p_0, C_n) = (p_1^n, p_2^n, \ldots, p_k^n)$ can be extended for C'_{n+1} to $\delta(p_0, C'_{n+1}) = (p', p_2, p_2^n, \ldots, p_k^n)$ by just following the run of δ over C_n, C, and C':

  $$p_1^n \xrightarrow{C} p \xrightarrow{a} p_1 \xrightarrow{C'} p' \xrightarrow{C_{X_1}^M} p'$$
  $$p_1^n \xrightarrow{C} p \xrightarrow{a} p_2$$

  Furthermore, given that C_{X_1} is a cyclic context over p' and $\delta(p_0, C'_{n+1}) = (p', p_2, p_2^n, \ldots, p_k^n)$, we know that there exists $M \in \mathbb{N}$ and $Y \in \mathsf{SCC}(\mathcal{T})$ that satisfies the properties stated in Lemma 5.5.2. We can now define the multi-context C_{n+1}:

  $$C_{n+1} = C'_{n+1} \circ C_{X_1}^{M+1}$$

  The first property that one can notice from this construction is that C_{n+1} is derived from C_n by composing C_n with a sequence of contexts in its first $\bullet$-node. This implies that C_n is a prefix of C_{n+1}. A second property that one can easily check is that $(a(C' \circ C_{X_1}^M), \bullet)$ is a multi-context with two $\bullet$-nodes and $C_n \circ C$ is a multi-context with $|\vec{x}_{n-1}|$-nodes labelled by $\bullet$. Hence, C_{n+1} is a multi-context with $|\vec{x}_{n-1}| + 1 = |\vec{x}_n|$ $\bullet$-nodes, that is, $C_{n+1} \in \mathcal{M}_\Sigma^k$ with $k = |\vec{x}_n|$, and the second

induction hypothesis is satisfied. The third condition was already proved when we stated that $\delta(p_0, C'_{n+1}) = (p', p_2, p_2^n, \ldots, p_k^n)$. Actually, this implies that $\delta(p_0, C_{n+1}) = (p', p_2, p_2^n, \ldots, p_k^n)$ by the definition of C_{n+1}. Given that $p' \in X_1$, $p_2 \in X_2$, and $\vec{x}_{n-1} = [p_1^n \cdot \ldots \cdot p_k^n]$, we have $\vec{x}_n = [p' \cdot p_2 \cdot p_2^n \cdot \ldots \cdot p_k^n]$. This shows that the multi-context C_{n+1} satisfies the first three properties of the inductive hypothesis.

We now show that C_{n+1} satisfies the last property of the inductive hypothesis by combining Lemma 5.5.1 and Lemma 5.5.2. Define $w = \mathcal{Z}(\hat{C}_{n+1}^{\mathsf{prefix}})$ and $\vec{y}_{n+1} = [\hat{\gamma}(q_0, w)]$. By the same arguments given in the base case, one can easily show that $\vec{y}_n \overset{\mathcal{T}}{\mapsto}{}^* \vec{y}_{n+1}$ is a valid rule of the prefix rewriting system associated with $\mathcal{T}$. Furthermore, we know by the second and third property of Lemma 5.5.2 that $\mathsf{top}(\vec{y}_n) = Y$ and $C_{X_1} \in \mathscr{L}(\mathcal{T} \mid Y)$. Given that C_{X_1} is the fingerprint context of component X_1, we get by Lemma 5.5.1 that $\mathscr{L}(\mathcal{R} \mid X_1) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$. We conclude that $\mathscr{L}(\mathcal{R} \mid \mathsf{top}(\vec{x}_n)) \subseteq \mathscr{L}(\mathcal{T} \mid \mathsf{top}(\vec{y}_n))$ which shows that $\langle\!\langle\, \vec{x}_n \,,\, \vec{y}_n \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_n \,,\, \vec{y}_{n+1} \,]\!]$ is a valid move of Repairer.

- Suppose now that Generator does a pop prefix rewriting rule over $\vec{x}_{n-1}$, i.e. $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} \vec{x}$ where $\vec{x}_{n-1} = X \cdot \vec{x}$ and $\vec{x}_n = \vec{x}$. Assume that $\vec{x} \neq \epsilon$ and denote by X' the top element of $\vec{x}$ (i.e. $X' = \mathsf{top}(\vec{x})$). If not, then $\vec{x} = \epsilon$ and Repairer wins the game, thus X' is well defined. Similar to the previous case, we know by the inductive hypothesis that there exists $p_1^n, \ldots, p_k^n \in P$ such that $\vec{x}_{n-1} = [p_1^n \cdot \ldots \cdot p_k^n]$ and $\delta(p_0, C_n) = (p_1^n, \ldots, p_k^n)$ with $k = |\vec{x}_{n-1}|$. Lets t be a tree such that there exists an accepting run of $\mathcal{R}$ over t starting from p_1^n and define the multi-context $C'_{n+1} = C_n \circ t$. One can easily check from the definition of C'_{n+1} and $\delta(p_0, C_n)$ that $\delta(p_0, C'_{n+1}) = (p_2^n, \ldots, p_k^n)$ and then $\vec{x}_n = [p_2^n \cdot \ldots \cdot p_k^n]$. Notice that hanging t from the first $\bullet$-node of C_n has an analogy behavior than popping the top element of $\vec{x}_{n-1}$. Now, let $C_{X'}$ be the fingerprint context of X' given by Lemma 5.5.1, $p \in Q$ be the state that satisfies $\delta(p, C_{X'}) = p$, and C be a context that connects p_2^n with p by using δ (i.e. $\delta(p_2^n, C) = p$). Define the multi-context $C''_{n+1} = C'_{n+1} \circ C$. One can easily verify that this new multi-context satisfies $\delta(p_0, C''_{n+1}) = (p, p_3^n, \ldots, p_k^n)$. Given that C''_{n+1}, p, and $C_{X'}$ satisfies the conditions of Lemma 5.5.2, we know that there exist $M \in \mathbb{N}$ and $Y \in \mathsf{SCC}(\mathcal{T})$ that satisfy the three properties stated in Lemma 5.5.2. Then we can define the multi-context C_{n+1} from C''_{n+1}:

$$C_{n+1} = C''_{n+1} \circ C_{X'}^{M+1}$$

Similar to the arguments given in the push case, we can easily check by the construction of C_{n+1} that C_{n+1} is a prefix of C_n, that $C_{n+1} \in \mathcal{M}_\Sigma^{k-1}$ where $|\vec{x}_n| = k - 1$, and $\vec{x}_n = [p \cdot p_3^n \cdot \ldots \cdot p_k^n]$ where $\delta(p_0, C_{n+1}) = (p, p_3^n, \ldots, p_k^n)$. This means that the first three properties of the inductive hypothesis are satisfied. For the last property of the inductive hypothesis, define $w = \mathcal{Z}(\hat{C}_{n+1}^{\mathsf{prefix}})$ and $\vec{y}_{n+1} = [\hat{\gamma}(q_0, w)]$. By following the same lines as in the base case or the push case, one

can check that $\langle\!\langle\, \vec{x}_n\,,\, \vec{y}_n \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}_n\,,\, \vec{y}_{n+1} \,]\!]$ is a valid move of Repairer in $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Therefore, C_{n+1} satisfies the inductive hypothesis and we are done.

We dedicate the last part of the proof to give some intuition why Referee is not needed during the play or, in other words, why the strategy of Repairer implicitly respects the restrictions imposed by the Referee. The first step of our argument is to relax the restrictions imposed by Referee over Repairer in the simulation game $\mathcal{G}_{\mathcal{R},\mathcal{T}}$. Let $S = \vec{y}_1 \overset{\mathcal{T}}{\mapsto} \vec{y}_2 \overset{\mathcal{T}}{\mapsto} \ldots \overset{\mathcal{T}}{\mapsto} \vec{y}_n$ be a sequence of prefix-rewriting rules associated with $\mathcal{T}$. We say that a position i is *repeatable* in S if $\mathsf{top}(\vec{y}_i) = \mathsf{top}(\vec{y}_n)$ and $|\vec{y}_j| \leq |\vec{y}_i|$ for all $j \leq i$. Let $r(S) = \min_{i \leq n}\{i \mid i \text{ is repeatable in } S\}$. Suppose that $\langle\!\langle\, \vec{x}\,,\, \vec{y}_1 \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}\,,\, \vec{y}_n \,]\!]$ is a move of Repairer, $S = \vec{y}_1 \overset{\mathcal{T}}{\mapsto} \ldots \overset{\mathcal{T}}{\mapsto} \vec{y}_n$ are the sequence of rules in $\mathcal{T}$ associated with this move, and suppose that $\mathsf{top}(\vec{x})$ and $\mathsf{top}(\vec{y}_n)$ are non-horizontal. Then we know that the next turn is from Referee who adds a $\triangleleft$-symbol below the top elements of $\vec{x}$ and $\vec{y}_n$. Strictly speaking, Referee could put the $\triangleleft$-symbol in any position before the $(|\vec{y}_n| - |\vec{y}_{r(S)}|)$-elements of $\vec{y}_n$ without changing the outcome of the game. Formally, suppose that $\vec{y}_n = Y_1 Y_2 \ldots Y_m$. Then we know that $\mathsf{top}(\vec{y}_{r(S)}) = Y_1$ and $\mathsf{tail}(\vec{y}_{r(S)})$ is a suffix of $\vec{y}_n$ since $r(S)$ is a repeatable position of S. Let $\vec{y}$ be the prefix of $\vec{y}_n$ such that $\vec{y}_n = \vec{y} \cdot \mathsf{tail}(\vec{y}_{r(S)})$. Notice that $\vec{y}_{r(S)} = Y_1 \cdot \mathsf{tail}(\vec{y}_{r(S)}) \overset{\mathcal{T}}{\mapsto}{}^{*}\ \vec{y} \cdot \mathsf{tail}(\vec{y}_{r(S)}) = \vec{y}_n$. If Repairer wants to access any element in $\vec{y}$ after Referee puts the separator in $\vec{y}_n$, then he can clearly replicate the same moves that he did after $\vec{y}_{r(s)}$, by performing the following move:

$$Y_1 \cdot \triangleleft \cdot Y_2 \ldots Y_m \overset{\mathcal{T}}{\mapsto}{}^{*}\ \vec{y} \cdot \triangleleft \cdot Y_2 \ldots Y_m$$

That is, all stack elements that stay between $\triangleleft$ and the suffix $\mathsf{tail}(\vec{y}_{r(S)})$ of $\vec{y}_n$ are still "virtually" accessible for Repairer. Therefore, we can assume without loss of generality that during the game Referee puts the separator in Repairer's stack in any position below the suffix $\mathsf{tail}(\vec{y}_{r(S)})$. Furthermore, we can suppose that this position can be chosen by the strategy of Repairer without changing the outcome of the game.

We use the above relaxation on the restrictions imposed by Referee to show that the strategy of Repairer constructed from a multi-context satisfies the restriction over non-horizontal components. By inspecting the arguments of the above inductive proof (base case, push case, or pop case), one can easily check that given a position of the game $\langle\!\langle\, \vec{x}\,,\, \vec{y} \,\rangle\!\rangle$ we derive a multi-context C and a fingerprint context D of $\mathsf{top}(\vec{x})$ by Lemma 5.5.1 that satisfy the conditions of Lemma 5.5.2. Then the context D is iterated $(M+1)$-times where M is given by Lemma 5.5.2 in order to construct the next multi-context $C \circ D^{M+1}$. Then we define the next move of Repairer by $\langle\!\langle\, \vec{x}\,,\, \vec{y} \,\rangle\!\rangle \overset{\mathsf{Rep}}{\mapsto} [\![\, \vec{x}\,,\, [\hat{\gamma}(q_0, \mathcal{Z}(w))] \,]\!]$ where $w = (C \circ D^{M+1})^{\mathsf{prefix}}$. Define the two words $w_0 = (C \circ D^{M+0})^{\mathsf{prefix}}$ and $w_1 = (C \circ D^{M+1})^{\mathsf{prefix}}$. If we rephrase the two first properties of Lemma 5.5.2 in terms of w_0 and w_1, we get that:

1. $\mathcal{Z}(w_1) = \mathcal{Z}(w_0) \cdot D^{\mathsf{prefix}}$,

2. $[\mathsf{top}(\hat{\gamma}(q_0, \mathcal{Z}(w_0)))] = [\mathsf{top}(\hat{\gamma}(q_0, \mathcal{Z}(w_1)))]$.

For the sake of simplification, define $W_0 = [\hat{\gamma}(q_0, \mathcal{Z}(w_0))]$ and $W_1 = [\hat{\gamma}(q_0, \mathcal{Z}(w_1))]$. Looking carefully at these two properties, we can check that W_0 is a repeatable position of the prefix rewriting rule $\vec{y} \overset{\mathcal{T}}{\mapsto}{}^* W_1$, formally, $\vec{y} \overset{\mathcal{T}}{\mapsto}{}^* W_0 \overset{\mathcal{T}}{\mapsto}{}^* W_1$ and W_0 is a repeatable position of this sequence of prefix rewriting rules associated with $\mathcal{T}$. Therefore, we can check that $W_1 = \vec{y}' \cdot \mathsf{tail}(W_0)$ for some string $\vec{y}' \in \mathsf{SCC}(\mathcal{T})^+$. By the above relaxation of the game, we can always assume that whenever $\mathsf{top}(\vec{x})$ is a non-horizontal component, then Referee moves:

$$[\![\vec{x}, W_1\,,\ \overset{\mathsf{Ref}}{\mapsto}\ ]\!][\![\mathsf{top}(\vec{x}) \cdot \triangleleft \cdot \mathsf{tail}(\vec{x}), \vec{y}' \cdot \triangleleft \cdot \mathsf{tail}(W_0)\,, .]\!]$$

In addition, if we suppose that $\mathsf{top}(\vec{x})$ is non-horizontal, then D is a non-horizontal context by Lemma 5.5.1. That is, there exists $a \in \Sigma$ and a context D_0 such that $\hat{D} = a \cdot \hat{D_0} \cdot \bar{a}$. Given that $D^{\mathsf{prefix}} = a \cdot D_0^{\mathsf{prefix}}$, then we have that $\mathcal{Z}(w_1) = \mathcal{Z}(w_0) \cdot a \cdot D_0^{\mathsf{prefix}}$. Furthermore, we know that $\mathcal{Z}$ is a tree repair transducer and, thus, the closing element of $\bar{a}$ is not modified. Formally, one can easily check from the definition of a tree repair transducer $\mathcal{Z}$ that for every context C, every tree t and $a \in \Sigma$ it holds that $\mathcal{Z}(C^{\mathsf{prefix}} \cdot a \cdot \hat{t} \cdot \bar{a}) = \mathcal{Z}(C^{\mathsf{prefix}} \cdot a \cdot \hat{t}) \cdot \bar{a}$ whenever $\mathcal{Z}(C^{\mathsf{prefix}} \cdot a) = \mathcal{Z}(C^{\mathsf{prefix}}) \cdot a$. This implies that $\mathsf{tail}(W_0)$ is a suffix of $[\hat{\gamma}(q_0, \mathcal{Z}(w_0 \cdot a \cdot \hat{t})))]$ for every $t \in \mathcal{T}_\Sigma$. Finally, we conclude that the strategy of Repairer (as defined) does not pass the suffix $\triangleleft \cdot \mathsf{tail}(W_0)$ until the $\bar{a}$ is seen, that is, when Generator reaches $\mathsf{tail}(\vec{x})$.

5.6 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 [GTW03], 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 exhaustively searches all plays to determine the winner of the simulation game, and possibly synthesize a winning strategy.

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

Theorem 5.6.1. *The problem of streaming bounded repairability for languages recognized by top-down tree automata is in EXPTIME.*

Proof of Theorem 5.6.1. We use our main characterization result to prove the EXPTIME upper bound by fixing two top-down automata $\mathcal{R} = (\Sigma, P, \delta, p_0, F)$ and $\mathcal{T} = (\Delta, Q, \gamma, q_0, G)$ and showing how to determine the winner of the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ by means of an alternating polynomial-space procedure.

We already mentioned that the stacks controlled by Generator in the simulation game never exceed in length the number of components of $\mathcal{R}$ – this follows from the fact that prefix-rewriting rules of the form $X \cdot \vec{x} \overset{\mathcal{R}}{\mapsto} X_1 X_2 \cdot \vec{x}$ exist 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 – this is mostly due to the existence of prefix-rewriting rules of the form $Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y_1 Y_2 \cdot \vec{y}$, with $Y_1 = Y$, which can be iterated to produce arbitrarily long stacks. To overcome this problem and be able to perform an exhaustive search on the arena in alternating polynomial space, we consider an equivalent version of the simulation game, obtained by modifying the prefix-rewriting rules associated with $\mathcal{T}$ in such a way that no rule of the form $Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y Y_2 \cdot \vec{y}$ is applicable.

We first describe the modified prefix-rewriting system associated with $\mathcal{T}$. For each component Y of $\mathcal{T}$, we introduce a dummy component Y^*, recognizing the empty language of contexts (in particular Y^* does not cover any component of $\mathcal{R}$). By a slight abuse of notation, we denote by $\mathsf{SCC}(\mathcal{T}')$ the new set of components, which includes the original components of $\mathcal{T}$ and the dummy copies. First, we replace each rule of the form $Y_1 \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y Y_2 \cdot \vec{y}$ or $Y_1 \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y_2 Y \cdot \vec{y}$ where $Y_1 \neq Y$ by the rule of the form:

$$Y_1 \cdot \vec{y} \overset{\mathcal{T}'}{\mapsto} Y Y^* Y_2 \cdot \vec{y} \quad \text{or} \quad Y_1 \cdot \vec{y} \overset{\mathcal{T}'}{\mapsto} Y_2 Y Y^* \cdot \vec{y}$$

respectively. In other words, we add Y^* below all the rules that produce Y. Then for each (recursive) rule of the form $Y \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y Y_2 \cdot \vec{y}$ we replace this rule with:

$$Y^* \cdot \vec{y} \overset{\mathcal{T}}{\mapsto} Y_2 Y^* \cdot \vec{y}$$

For the special case when Y is equal to the initial component Y_0, we redefine Y_0^* to be the initial component of $\overset{\mathcal{T}'}{\mapsto}$. We denote by $\overset{\mathcal{T}'}{\mapsto}$ the modified prefix-rewriting system. Observe that the new system does not contain rules of the form $Y \cdot \vec{y} \overset{\mathcal{T}'}{\mapsto} Y Y_2 \cdot \vec{y}$. As usual, we denote by $\overset{\mathcal{T}'}{\mapsto}{}^*$ the reflexive and transitive closure of $\overset{\mathcal{T}'}{\mapsto}$. Below, we prove that the two prefix-rewriting systems $\overset{\mathcal{T}}{\mapsto}{}^*$ and $\overset{\mathcal{T}'}{\mapsto}{}^*$ are essentially equivalent, according to the following definition:

Definition 5.6.2. *We say that two stacks $\vec{y} \in \mathsf{SCC}(\mathcal{T})^+$ and $\vec{z} \in \mathsf{SCC}(\mathcal{T}')^+$ are* bisimilar up to n rounds *if the following conditions hold:*

1. $\mathsf{top}(\vec{y}) = \mathsf{top}(\vec{z})$,
2. *if $n > 0$ and $\vec{y} \overset{\mathcal{T}}{\mapsto}{}^* \vec{y}'$, then $\vec{z} \overset{\mathcal{T}'}{\mapsto}{}^* \vec{z}'$ for some stack $\vec{z}' \in \mathsf{SCC}(\mathcal{T}')^+$ bisimilar to $\vec{y}'$ up to $n-1$ rounds,*
3. *if $n > 0$ and $\vec{z} \overset{\mathcal{T}'}{\mapsto}{}^* \vec{z}'$, then $\vec{y} \overset{\mathcal{T}}{\mapsto}{}^* \vec{y}'$ for some stack $\vec{y}' \in \mathsf{SCC}(\mathcal{T})^+$ bisimilar to $\vec{z}'$ up to $n-1$ rounds.*

We say that the two systems $\overset{\mathcal{T}}{\mapsto}{}^*$ *and* $\overset{\mathcal{T}'}{\mapsto}{}^*$ *are* similar up to n rounds *if given the component* Y_0 *of the initial state of* $\mathcal{T}$*, we have that the singleton stack* Y_0 *is bisimilar to* Y_0^* *up to* n *rounds.*

It is easy to verify, e.g. by induction on the number of rounds, that the two prefix-rewriting systems $\overset{\mathcal{T}}{\mapsto}{}^*$ and $\overset{\mathcal{T}'}{\mapsto}{}^*$ are bisimilar up any finite number of rounds. Furthermore, from the definition of $\overset{\mathcal{T}'}{\mapsto}{}^*$ we can easily check that stacks in $\overset{\mathcal{T}'}{\mapsto}{}^*$ are of size polynomial in $\overset{\mathcal{T}'}{\mapsto}{}^*$. Indeed, each transition is of the form: $Y \cdot \vec{y} \overset{\mathcal{T}'}{\mapsto} Y_1\, Y_2 \cdot \vec{y}$ where $Y_1 \neq Y \neq Y_2$ or $Y \cdot \vec{y} \overset{\mathcal{T}'}{\mapsto} \vec{y}' \cdot Y \cdot \vec{y}$ where $\vec{y}'$ is a sequence of components strictly below Y and, therefore, stacks in $\overset{\mathcal{T}'}{\mapsto}{}^*$ are of size linear in the number of components in $\overset{\mathcal{T}'}{\mapsto}{}^*$.

Now, we denote by $\mathcal{G}_{\mathcal{R},\mathcal{T}'}$ the arena obtained by applying the usual Definition 5.3.1 to the modified prefix-rewriting system $\overset{\mathcal{T}'}{\mapsto}{}^*$. Clearly, using the fact that the two systems $\overset{\mathcal{T}}{\mapsto}{}^*$ and $\overset{\mathcal{T}'}{\mapsto}{}^*$ are bisimilar – in particular the first condition of Definition 5.6.2 – and another simple induction on the rounds, one can deduce that Repairer wins the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ iff he wins the simulation game over $\mathcal{G}_{\mathcal{R},\mathcal{T}'}$.

We conclude the proof of the theorem by exploiting the equivalence of the simulation games over $\mathcal{G}_{\mathcal{R},\mathcal{T}}$ and $\mathcal{G}_{\mathcal{R},\mathcal{T}'}$ and the bounds to the sizes of the reachable stacks of $\mathcal{G}_{\mathcal{R},\mathcal{T}'}$ to derive the existence of an alternating polynomial-space procedure that simulates all possible plays in $\mathcal{G}_{\mathcal{R},\mathcal{T}'}$, eventually determining the winniner of the game. □

In the next theorem, we show that the problem of streaming bounded repairability for top-down tree automata is EXPTIME-hard. In fact, we show that EXPTIME-hardness holds for languages specified by *non-recursive deterministic DTDs* (see Section 4.1). Given that any deterministic DTD can be efficiently translated into an equivalent deterministic top-down tree automaton [MNSB06], the EXPTIME-hardness result can be transferred to languages recognized by deterministic top-down tree automata.

Theorem 5.6.3. *The problem of streaming bounded repairability for languages defined by non-recursive deterministic DTDs is EXPTIME-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. 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 C 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 $1 < j \leq n$,

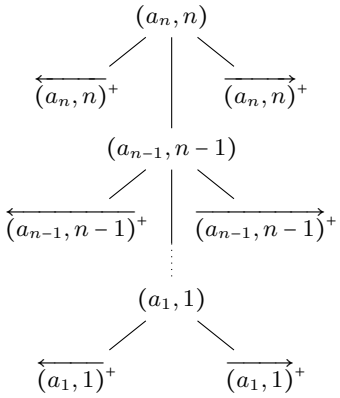


Figure 5.5: Example of the encoding of Adam's rows.

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 the height of the tiling reaches 2^{n+1}. We know from [Boa97] that the problem of deciding the winner of a tiling game is APSPACE-hard (hence EXPTIME-hard).

Below, we construct two DTDs $\mathcal{R}$ and $\mathcal{T}$ of size polynomial in $|I|$ that define two languages R and T such that *R is streaming bounded repairable into T iff Eve wins the instance I of the tiling game*. 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 allow some redundancy in the encodings of Adam rows in order to forbid any repair processor from modifying them with boundedly many edits. Symmetrically, the target DTD $\mathcal{T}$ will require "interleaving" the encodings of rows produced by Adam with new rows, representing Eve responses to Adam. Since we cannot guarantee that the rows generated by the restriction DTD satisfy the vertical constraints, we allow Adam to "cheat" by producing rows that do not match with the previous ones. This freedom is however countered by the possibility of Eve producing an ad hoc repair that "exposes" a violation of the constraints. 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 a family of trees of the form shown in Figure 5.5. We observe that, in the above encoding, the tiles $a_1, \dots, a_n$ of α are paired with the indices 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 are repeated an arbitrary number of times both to the left and to the right of the spine (left and right arrows annotate these repetitions of tiles).

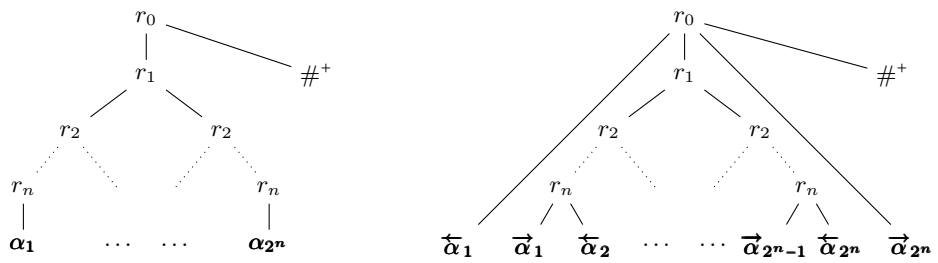
The sequence of all possible rows that could be produced by Adam is encoded inside another tree. As Adam produces exactly 2^n rows, say $\alpha_1, \ldots, \alpha_{2^n}$, it is sufficient to construct a full binary tree of height n and append to its leaves the encodings of $\alpha_1, \ldots, \alpha_{2^n}$. Moreover, for a technical reason that will be clear later, Adam will produce a repetition of a dummy tile # at the end of his rows. The rough intuition is that, when Adam cheats, a suitable repair process can "eat" the dummy tile and correctly get to the target language.

The resulting structure is shown in Figure 5.6(a). For the sake of simplicity, we name the trees that encode the rows produced by Adam with bold letters. 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 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 by removing all nodes labeled with r_0, r_i, and (a_i, i), for all $1 \leq i \leq n$. This results in a sequence of forests $\overleftarrow{\alpha}_1$, $\overrightarrow{\alpha}_1$, $\ldots$, $\overleftarrow{\alpha}_{2^n}$, $\overrightarrow{\alpha}_{2^n}$, 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 another full binary tree, which spans the previously mentioned forests (possibly modified) and groups them pairwise, leaving out just the first and the last forest. This operation is required to induce a shift in the grouping of the forests, so that one can later verify the vertical constraints on consecutive rows. Intuitively, at this stage of the repair process an intermediate tree shown in Figure 5.6(b) 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 $\alpha_\perp = a_\perp \ldots a_\perp$, 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 definable by a DTD. In the following, we will describe further editing operations, which are performed in the case Adam did not cheat right at the beginning.



(a) Decomposition of a tree.

(b) Intermediate stage of a repair.

Figure 5.6: Different stages of a repair strategy.

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 simply mimics the moves of Eve by modifying the 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 responds with the row $\beta_i = b_{i,1} \dots b_{i,n}$. This means that the portion of the serialized input tree disclosed so far ends with a string of the form

$$\overrightarrow{\alpha}_i = \left(\overrightarrow{(a_{i,1}, 1)}\ \overline{\overrightarrow{(a_{i,1}, 1)}} \right)^+ \dots \left(\overrightarrow{(a_{i,n}, n)}\ \overline{\overrightarrow{(a_{i,n}, n)}} \right)^+$$

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

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

(the inserted opening tags are listed in bold and will be closed at the next repair step). After that, the input is resumed and the encoding of the next row $\alpha_{i+1} = a_{i+1,1} \dots a_{i+1,n}$ is consumed:

$$\overleftarrow{\alpha}_{i+1} = \left(\overleftarrow{(a_{i+1,n}, n)}\ \overline{\overleftarrow{(a_{i+1,n}, n)}} \right)^+ \dots \left(\overleftarrow{(a_{i+1,1}, 1)}\ \overline{\overleftarrow{(a_{i+1,1}, 1)}} \right)^+ .$$

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

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

Otherwise, if the row is not correct, namely, if $i + 1 = i^*$, then we know that $(b_{i,j}, a_{i+1,j}) \notin V$ for some column j that witnesses the violation. In this case, the canonical repair strategy performs an editing such as the previous one, with the only difference that the forests encoded by the j-th and

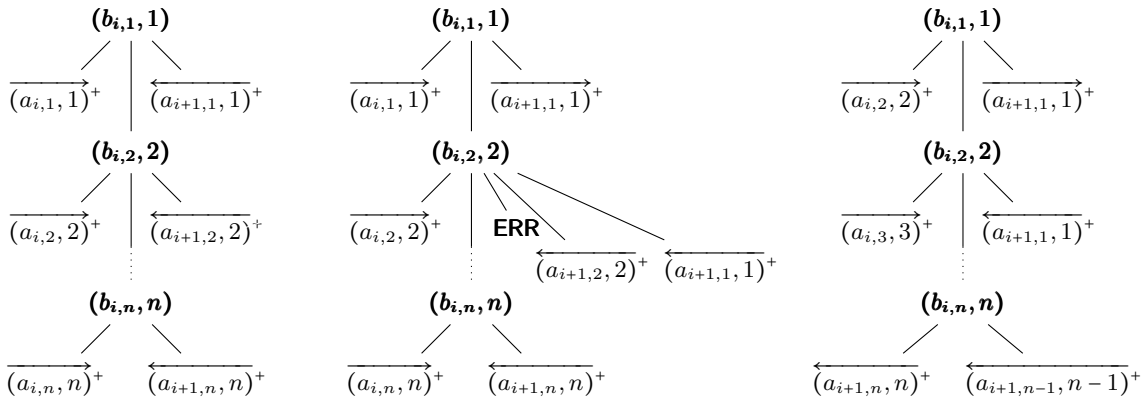


Figure 5.7: Subtrees that could possibly arise from the repair process.

$j-1$-th blocks in $\overleftarrow{\alpha}_{i+1}$ will be gathered under the same node labeled with $b_{i,j}$, and 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 that will propagate up to the root of the output tree, allowing in this way the validation by a target DTD. Thus, the output produced by this repair process is:

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

$$\mathsf{ERR}\ \left(\overleftarrow{(a_{i+1,j},j)}\ \overline{\overleftarrow{(a_{i+1,j},j)}}\right)^+ \left(\overleftarrow{(a_{i+1,j-1},j)}\ \overline{\overleftarrow{(a_{i+1,n},j-1)}}\right)^+ \overline{\boldsymbol{(b_{i,j},j)}}\ \ \ldots$$

$$\left(\overleftarrow{(a_{i+1,1},1)}\ \overline{\overleftarrow{(a_{i+1,1},1)}}\right)^+ \overline{\boldsymbol{(b_{i,2},2)}}\ \left(\overrightarrow{(a_{i+1,1},1)}\ \overline{\overrightarrow{(a_{i+1,1},1)}}\right)^+ \overline{\boldsymbol{(b_{i,1},1)}}.$$

We observe that the last part of the output ends with the closing tag $\overline{(b_{i,2},2)}$, so the $(b_{i,1},1)$-labeled node that was previously inserted has to be closed with the next incoming block, if $i < 2^n$, or with 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 5.7 some subtrees that could possibly arise from the repair process. The left-hand side tree occurred before any violation of the constraints, the tree in the middle occurred exactly at the first violation of the constraints – a violation on column 2 –, and the right-hand side tree occurred 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 checked by a deterministic DTD.

Given the explanations above, it is natural to define the following DTD for the target language:

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

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:

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

- $\mathsf{Left}(b,j)$, 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. In order to prove the correctness of the above reduction, we need to argue that, when Adam wins the tiling game, an unbounded number of edits is required to repair trees from $\mathcal{R}$ to $\mathcal{T}$. This is easily proved by considering the family of all trees that encode rows played by Adam according to his winning strategy. Within this family, each tile can be encoded with an arbitrary amount of redundancy, which makes it impossible for any bounded repair processor to erase or replace a tile in a row. On the other hand, forging new rows that violate the constraints and pretending they were played by Adam does not help the repair process either, as this can be detected in the target language by a change in the number of rows. □

We conclude the section by recalling a result from Chapter 4 that concerns a specific case of the streaming bounded repairability problem. From Propositions 4.6.1 and 4.6.2 of Chapter 4 it follows that the complexity of the streaming bounded repairability problem drops to PTIME when the restriction language contains all trees over a given alphabet Σ.

5.7 Conclusions

In this chapter, 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 most of the concepts introduces in the previous chapters. For example, trees in the restriction and target are abstracted by using strongly connected components (Chapter 4) and games are used to characterize the problem (Chapter 2). One can easily observe here that this two-stack game is basically a reachability game suitably encoded into stacks: Generator is restricted to a finite number of moves and one could easily impose a restriction in the possible moves of Repairer. Indeed, the decidability of the streaming bounded repair problem is strongly based on this property. We use this fact to give an alternating polynomial space algorithm (i.e. EXPTIME algorithm) to decide the streaming bounded repair problem and, furthermore, we show that this algorithm is optimal (EXPTIME-hard).

A weakness of our technique is that we heavily depend on the top-down determinism of the two schemas – for the case of schemas given by arbitrary tree automata, decidability is still open. Recall that Example 17 shows that a general characterization cannot be based on the strongly connected components of tree automata. In this example, the restriction language is not streaming bounded repairable into the target language regardless that both specifications have the same graph structure. We observe that a better understanding of the cyclic behaviour of tree automata is needed in order to effectively characterize the streaming bounded repair problem in general. In this direction, we wants to highlight Proposition 4.6.1 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 winning strategy on the simulation game, one can effectively extract a streaming repair strategy giving by the composition of three different transducers (see Section 5.4). By composing these transducers, one can easily notice that only two stacks are needed in order to run the restriction and target tree automata in parallel. An interesting question is to determine whether two stacks are always needed or whether this can be done with just one stack. Also, we do not know the exact complexity of determining the optimal repair transducer, where optimality is expressed in terms of maximal number of repairs. Further simplifications in the proof are required in order to better understand some open question regarding the optimality of the streaming repair strategy.

We left open some complexity gaps when the restriction and target are represented by non-deterministic specifications. For example, we did not consider the case when the restriction and target are given by general DTDs. Notice here that a DTD specification can be represented by a deterministic top-down tree automata only after determinizing each regular expression on the head of its rules. Thus, one can easily derive from here a double exponential algorithm to decide streaming bounded repairability of general DTDs by applying our EXPTIME algorithm to a "determinized" DTD. Unfortunately, EXPTIME-hardness is the best lower bound that one can get from our results which is very far from the double exponential algorithm argued above. Recall that similar gaps were also left open in the string case (Chapter 2). We think that new insights are required to close these complexity gaps of the streaming bounded repair problem for non-deterministic specifications.

We want to briefly add that some complexity results presented in [BPR13] were not included in this chapter. These results show some sub-cases where the complexity of the streaming bounded repair problem decreases to PSPACE.

Finally, our work highlights the issue of the proper notion of edit processor for trees that have a canonical serialization as a string, as is the case with XML. Example 15 shows that the ability to edit tree serializations is more powerful than emitting tree edits. The example can be used to show that there are XML schemas that can be repaired in streaming fashion with a bounded number of edits on the serialization, but where there is no bounded repair processor of any sort (even non-streaming) that repairs using only tree edits. We do not know if this last phenomena can occur for more limited schemas, such as DTDs.

Chapter 6

Conclusion

In this thesis, we take the next step in repairing strings and trees specified by regular specifications. We study foundational problems such as deciding the bounded repair problem and computing the asymptotic cost for regular languages over strings and trees. The main results can be summarized as follows.

- We give an effective characterization (Theorem 2.3.1) of the bounded repair problem for regular languages given by non-deterministic finite automata. The characterization is based on a covering relation between chains of strongly connected components. We use this characterization to show that the bounded repair problem is PSPACE-complete for NFAs and coNP-complete for DFAs. Table 2.1 in Chapter 2 briefly summarizes the complexity map for the bounded repair problem in the non-streaming setting.

- We characterize the streaming bounded repair problem of DFAs in terms of finite games (Theorem 2.3.3). Our characterization shows that the streaming bounded repair problem can be solved in polynomial time. Furthermore, it shows that streaming bounded repairing is not sensible to the amount of memory or look-ahead needed by the streaming strategy. Several other related problems such as the bounded repair problem in the unrestricted case or the threshold problem are studied and tight complexity bounds are given.

- We provide an algorithm to compute the asymptotic cost for any pair regular languages given by NFAs (Theorem 3.2.6 and 3.2.11). The algorithm shows that the asymptotic cost is always rational and it can be computed in double exponential time. We also connect the computation of the streaming asymptotic cost with mean-payoff games (Theorem 3.3.2). Interestingly, this result shows that the streaming asymptotic cost can be computed in polynomial time in contrast to the double exponential time upper-bound given in the non-streaming case.

- We give an effective characterization of the bounded repair problem over trees in the non-streaming setting (Theorem 4.3.6). The characterization and proof are highly non-trivial and

	Non-streaming	**Streaming**
Strings	coNP	PTIME
Trees	coNEXPTIME	EXPTIME

Table 6.1: Complexity of the bounded repair problem in the non-streaming and streaming case

are inspired on a generalization of the ideas of components and coverability relation used in the string case. We exploit this characterization to show that the problem is coNEXPTIME-complete (Theorem 4.5.5) for all the regular tree specifications considered throughout this thesis (e.g. non-deterministic and deterministic tree automata). We also study sub-cases of this problem over trees: we show that the non-streaming and streaming bounded repair problems coincide when the restriction is universal.

- We show that the streaming bounded repair problem over trees can be effectively characterized in terms of stacks games (Theorem 5.3.2). The characterization is specifically for tree languages given by "top-down deterministic tree automata", a sub-class of regular tree languages that includes DTDs and XML schemas (XSD). Similar to the non-streaming case, the characterization and proof are non-trivial and join most of the ideas used throughout this thesis. A side-effect of the proof is the construction of a streaming repair strategy of bounded cost whenever the pair of tree languages are streaming bounded repairable. With this characterization in hand, we show that the streaming bounded repair problem for top-down tree automata is EXPTIME-complete (Theorem 5.6.1 and 5.6.3).

A concise summary of the main results in this thesis is given by Table 6.1. All bounds in this table are tight and for specifications given by deterministic automata (recall that for streaming trees this was only proved for top-down deterministic tree automata). Interestingly, there is a perfect symmetry between strings versus trees, or streaming versus non-streaming. For example, there is an exponential blow-up in the complexity when we go from strings to trees. Furthermore, if we compare the algorithmic complexity of the non-streaming and streaming bounded repair problem, it is always "easier" to determine whether two languages are streaming bounded repairable rather than non-streaming bounded repairable. We believe that this observation is of particular interest if some of these results wants to be applied in practice.

We have already mentioned some open questions in the conclusion section at the end of each chapter, but we highlight here the main problems that are left open and some directions for future work.

- An interesting problem is to determine the precise complexity of the bounded repair problem in the streaming setting for non-deterministic finite automata. The complexity of this problem

is between PSPACE and EXPTIME. An EXPTIME-hardness proof would show that a game-based characterization (such as Theorem 2.3.3) is only for deterministic automata. That is, if we want to decide streaming bounded repairability for non-deterministic automata, we are forced to determinize our finite automata before applying the polynomial time algorithm obtained by the game-based characterization. We conjecture that the problem is EXPTIME-complete, but we do not have any clue how to prove it.

- Another interesting open problem is to optimize the algorithm given in Chapter 3 for computing the asymptotic cost of two regular languages. Our algorithm runs in EXPTIME and the best lower bound is PSPACE-hard. It is quite surprising that the asymptotic cost is computable. However, it is far from our techniques to understand the underlying complexity.

- The main open problem in this thesis is a characterization for the streaming bounded repair problem on arbitrary regular tree languages. Recall that Example 17 shows that a general characterization cannot be based on the techniques proposed in this thesis. In this example, the restriction language is not streaming bounded repairable into the target language despite that both specifications have the same graph structure. We believe that a better understanding of the cyclic behaviour of (streaming) tree automata is needed in order to provide an effective characterization for this problem.

- This thesis opens new research directions for future work. One possible direction is to study whether it is decidable if there exists a streaming repair strategy that achieves the same cost as an optimal off-line repair strategy. That is, given two regular languages R and T (over strings or trees) whether there exists a streaming strategy $\mathcal{Z}$ such that $\mathsf{cost}(w, \mathcal{Z}) = \mathsf{dist}(w, T)$ for all $w \in R$. The idea behind $\mathcal{Z}$ is that one could always repair any document in R by using $\mathcal{Z}$ instead of the standard dynamic programming algorithm (presented in [Wag74]). Another possibility is to consider whether there exists a streaming strategy $\mathcal{Z}$ and a bound N such that $\mathsf{cost}(w, \mathcal{Z}) \leq \mathsf{dist}(w, T) + N$ for all $w \in R$. In this case, we allow $\mathcal{Z}$ to make some uniformly number of "mistakes" for all documents in R. The ideas presented in [AKL10] are a good starting point towards this direction.

The hope is that this thesis sets the stage for further development on the foundations of regular repair of specifications over strings and trees, which may ultimately help to give practical solutions to this kind of problematic.

Bibliography

[ABC99] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.

[AGMN11] Timos Antonopoulos, Floris Geerts, Wim Martens, and Frank Neven. Generating, sampling and counting subclasses of regular tree languages. In *Proceedings of the 14th International Conference on Database Theory (ICDT)*, pages 30–41, 2011.

[AK09] Foto Afrati and Phokion Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *Proceedings of the 12th International Conference on Database Theory (ICDT)*, pages 31–41, 2009.

[AKL10] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms (TALG)*, 6(2):28, 2010.

[AKNS01] Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6):1842–1862, 2001.

[Aku06] Tatsuya Akutsu. A relation between edit distance for ordered trees and edit distance for euler strings. *Information Processing Letters*, 100(3):105–109, 2006.

[AM04] Rajeev Alur and Parthasarathy Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 202–211, 2004.

[AM09] Rajeev Alur and Parthasarathy Madhusudan. Adding nesting structure to words. *Journal of the ACM (JACM)*, 56(3):16, 2009.

[AP72] Alfred Aho and Thomas Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM Journal on Computing*, 1(4):305–312, 1972.

[BdR04] Utsav Boobna and Michel de Rougemont. Correctors for XML data. In *Database and XML Technologies*, pages 97–111. 2004.

[Bil05] Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science (TCS)*, 337(1):217–239, 2005.

[BKW98] Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

[BL80] Janusz Brzozowski and Ernst Leiss. On equations for regular languages, finite automata, and sequential networks. *Theoretical Computer Science (TCS)*, 10:19–35, 1980.

[Boa97] Peter Van Emde Boas. The convenience of tilings. In *Complexity, Logic, and Recursion Theory*, pages 331–363, 1997.

[BPRa] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Bounded repairability of word languages. *Journal of Computer and System Sciences (JCSS)*. To appear.

[BPRb] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. The per-character cost of repairing word languages. Submited to Theoretical Computer Science (TCS).

[BPR11a] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. The cost of traveling between languages. In *Automata, Languages and Programming (ICALP)*, pages 234–245. 2011.

[BPR11b] Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Regular repair of specifications. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 335–344, 2011.

[BPR13] Pierre Bourghis, Gabriele Puppis, and Cristian Riveros. Which DTDs are streaming bounded repairable? In *Proceedings of the 16th International Conference on Database Theory (ICDT)*, 2013.

[BPSM+08] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Francois Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation, November 2008.

[CAM02] Gregory Cobena, Serge Abiteboul, and Amelie Marian. Detecting changes in XML documents. In *International Conference on Data Engineering (ICDE)*, pages 41–52, 2002.

[CD12] Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theoretical Computer Science (TCS)*, 2012.

[CD13] Thomas Colcombet and Laure Daviaud. Approximate comparison of distance automata. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 574–585, 2013.

[CDAHS03] Arindam Chakrabarti, Luca De Alfaro, Thomas A Henzinger, and Mariëlle Stoelinga. Resource interfaces. In *Embedded Software*, pages 117–133, 2003.

[CDG+07] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. 2007.

[CGLN09] Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Efficient inclusion checking for deterministic tree automata and XML schemas. *Information and Computation*, 207(11):1181–1208, 2009.

[CKS81] Ashok Chandra, Dexter Kozen, and Larry Stockmeyer. Alternation. *Journal of the ACM (JACM)*, 28(1):114–133, 1981.

[CL10] Thomas Colcombet and Christof Loding. Regular cost functions over finite trees. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 70–79. IEEE, 2010.

[CLRS09] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to algorithms (3. ed.)*. MIT press, 2009.

[CLT05] Julien Cristau, Christof Löding, and Wolfgang Thomas. Deterministic automata on unranked trees. In *Fundamentals of Computation Theory*, pages 68–79, 2005.

[CNT04] Julien Carme, Joachim Niehren, and Marc Tommasi. Querying unranked trees with stepwise tree automata. In *Rewriting Techniques and Applications (RTA)*, pages 105–118, 2004.

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[DR02] Hong-Hai Do and Erhard Rahm. COMA: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pages 610–621, 2002.

[Ear70] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[EM79] Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

[FB08] Wenfei Fan and Philip Bohannon. Information preserving XML schema embedding. *ACM Transactions on Database Systems (TODS)*, 33(1):4, 2008.

[FFGZ05] Sergio Flesca, Filippo Furfaro, Sergio Greco, and Ester Zumpano. Querying and repairing inconsistent XML data. In *Web Information Systems Engineering (WISE)*, pages 175–188. 2005.

[Fis01] Eldar Fischer. The art of uninformed decisions. *Bulletin of the EATCS*, 75:97, 2001.

[FMDR10] Eldar Fischer, Frédéric Magniez, and Michel De Rougemont. Approximate satisfiability and equivalence. *SIAM Journal on Computing*, 39(6):2251–2281, 2010.

[GIM+13] Wouter Gelade, Tomasz Idziaszek, Wim Martens, Frank Neven, and Jan Paredaens. Simplifying XML schema: single-type approximations of regular tree languages. *Journal of Computer and System Sciences (JCSS)*, 2013.

[GNR08] Olivier Gauwin, Joachim Niehren, and Yves Roos. Streaming tree automata. *Information Processing Letters*, 109(1):13–17, 2008.

[GT04] Gösta Grahne and Alex Thomo. Query answering and containment for regular path queries under distortions. In *Foundations of Information and Knowledge Systems (FOIKS)*, pages 98–115. 2004.

[GTW03] Erich Grädel, Wolfgang Thomas, and Thomas Wilke. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer, 2003.

[Has90] Kosaburo Hashiguchi. Improved limitedness theorems on finite automata with distance functions. *Theoretical Computer Science (TCS)*, 72(1):27–38, 1990.

[HU79] John Hopcroft and Jeffrey Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 1979.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing (STOC)*, pages 302–311, 1984.

[KMV07] Viraj Kumar, Parthasarathy Madhusudan, and Mahesh Viswanathan. Visibly pushdown automata for streaming XML. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 1053–1062, 2007.

[Kon07] Stavros Konstantinidis. Computing the edit distance of a regular language. *Information and Computation*, 205(9):1307–1316, 2007.

[Kro94] Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4:405–425, 1994.

[LP04] Hing Leung and Viktor Podolskiy. The limitedness problem on distance automata: Hashiguchi's method revisited. *Theoretical Computer Science (TCS)*, 310(1):147–158, 2004.

[MG06] Jiri Matousek and Bernd Gärtner. *Understanding and using linear programming.* Springer, 2006.

[MLMK05] Makoto Murata, Dongwon Lee, Murali Mani, and Kohsuke Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology (TOIT)*, 5(4):660–704, 2005.

[MN07] Wim Martens and Joachim Niehren. On the minimization of XML schemas and tree automata for unranked trees. *Journal of Computer and System Sciences (JCSS)*, 73(4):550–583, 2007.

[MNS08] Wim Martens, Frank Neven, and Thomas Schwentick. Deterministic top-down tree automata: past, present, and future. In *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 505–530, 2008.

[MNSB06] Wim Martens, Frank Neven, Thomas Schwentick, and Geert Jan Bex. Expressiveness and complexity of XML schema. *ACM Transactions on Database Systems (TODS)*, 31(3):770–813, 2006.

[Moh97] Mehryar Mohri. Finite-state transducers in language and speech processing. *Journal of Computational Linguistics*, 23(2):269–311, 1997.

[Moh03] Mehryar Mohri. Edit-distance of weighted automata: general definitions and algorithms. *International Journal of Foundations of Computer Science*, 14(06):957–982, 2003.

[Pap94] Christos H Papadimitriou. *Computational complexity.* Addison Wesley, 1994.

[PRS12] Gabriele Puppis, Cristian Riveros, and Sławek Staworko. Bounded repairability for regular tree languages. In *Proceedings of the 15th International Conference on Database Theory (ICDT)*, pages 155–168, 2012.

[RS08] Jean-François Raskin and Frédéric Servais. Visibly pushdown transducers. In *Automata, Languages and Programming (ICALP)*, pages 386–397. 2008.

[SC06] Slawomir Staworko and Jan Chomicki. Validity-sensitive querying of XML databases. In *EDBT Workshops*, pages 164–177. 2006.

[Sch77] Marcel Paul Schuetzenberger. Sur une variante des fonctions séquentielles. *Theoretical Computer Science (TCS)*, 4(1):47–57, 1977.

[Sch07] Thomas Schwentick. Automata for XML – a survey. *Journal of Computer and System Sciences (JCSS)*, 73(3):289–315, 2007.

[Sei90] Helmut Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.

[Sel77] Stanley Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.

[Sim88] Imre Simon. Recognizable sets with multiplicities in the tropical semiring. In *Mathematical Foundations of Computer Science (MFCS)*, pages 107–120, 1988.

[SM73] Larry Stockmeyer and Albert Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–9, 1973.

[SV02] Luc Segoufin and Victor Vianu. Validating streaming XML documents. In *Proceedings of the 21th ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 53–64, 2002.

[Tai79] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM (JACM)*, 26(3):422–433, 1979.

[Wag74] Robert Wagner. Order-n correction for regular languages. *Communications of the ACM (CACM)*, 17(5):265–268, 1974.

[WF74] Robert Wagner and Michael Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[ZP96] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science (TCS)*, 158(1):343–359, 1996.