Università degli studi di Udine

Bounded repairability for regular tree languages

Original

Availability:
This version is available http://hdl.handle.net/11390/1174098 since 2021-03-19T16:00:06Z

Publisher:

Published
DOI:10.1145/2898995

Terms of use:
The institutional repository of the University of Udine (http://air.uniud.it) is provided by ARIC services. The aim is to enable open access to all the world.

Publisher copyright

(Article begins on next page)

12 October 2024

A

Bounded Repairability for Regular Tree Languages

Pierre Bourhis, CNRS – CRIStAL UMR 9189, pierre.bourhis@univ-lille1.fr
Gabriele Puppis, CNRS – LaBRI, University of Bordeaux, gabriele.puppis@labri.fr
Cristian Riveros, Dep. of Computer Science, Pontificia Universidad Católica de Chile, cristian.riveros@uc.cl
Sławek Staworko, INRIA Lille-Nord Europe, University of Lille & LFCS, University of Edinburgh, slawomir.staworko@inria.fr

We study the problem of bounded repairability of a given *restriction* tree language R into a *target* tree language T. More precisely, we say that R is *bounded repairable* w.r.t. T if there exists a bound on the number of standard tree editing operations necessary to apply to any tree in R in order to obtain a tree in T. We consider a number of possible specifications for tree languages: bottom-up tree automata (on curry encoding of unranked trees) that capture the class of XML Schemas and DTDs. We also consider a special case when the restriction language R is universal, i.e., contains all trees over a given alphabet.

We give an effective characterization of bounded repairability between pairs of tree languages represented with automata. This characterization introduces two tools, *synopsis trees* and a *coverage relation* between them, allowing one to reason about tree languages that undergo a bounded number of editing operations. We then employ this characterization to provide upper bounds to the complexity of deciding bounded repairability and we show that these bounds are tight. In particular, when the input tree languages are specified with arbitrary bottom-up automata, the problem is CONEXP-complete. The problem remains CONEXP-complete even if we use deterministic non-recursive DTDs to specify the input languages. The complexity of the problem can be reduced if we assume that the alphabet, the set of node labels, is fixed: the problem becomes PSPACE-complete for non-recursive DTDs and CONP-complete for deterministic non-recursive DTDs. Finally, when the restriction tree language R is universal, we show that the bounded repairability problem becomes EXP-complete if the target language is specified by an arbitrary bottom-up tree automaton and becomes tractable (P-complete, in fact) when a deterministic bottom-up automaton is used.

Categories and Subject Descriptors: H.2.3 [**Database Management**]: Languages - *Data manipulation languages*; H.2.5 [**Database Management**]: Heterogeneous Databases - *Data translation*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: XML, XML Schema, DTD, Repair, Edit distance

ACM Reference Format:
ACM Trans. Embedd. Comput. Syst. V, N, Article A (January YYYY), 50 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

The first author was supported by CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020. The research leading to these results has been partly funded by the EPSRC (UK) grant EP/G004021/1 and by the EU project FOX (FP7-ICT-233599). The second author was partially supported by ExStream project (ANR-13-JS02-0010). The third author was supported by CONICYT + PAI / Concurso Nacional Apoyo al Retorno de Investigadores/as desde el extranjero – Convocatoria 2013 + 821320001 and by the Millenium Nucleus Center for Semantic Web Research under grant NC120004. The fourth author has been supported by the EU FP7 DIACHRON project.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© YYYY ACM 1539-9087/YYYY/01-ARTA $15.00
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

A basic problem in data management is to ensure that data is valid, namely, satisfies all integrity constraints associated with a schema [Bertossi 2011]. Validation of data with respect to a schema is crucial in any database system: if data does not satisfy the integrity constraints, then one cannot guarantee that the output produced by the system is correct. Nevertheless, when data does not satisfy constraints, a natural approach is to attempt a *repair*, that is, to modify the data minimally so that it becomes valid [Arenas et al. 1999; Afrati and Kolaitis 2009]. We may want to perform this transformation on the data, or may be merely interested in knowing how difficult it is to perform the transformation in case of need – that is, determining how far a given collection of data is from satisfying the specification. For example, some applications may retain input data even when this contains a few errors, where "few" could be interpreted as a user-defined bound to the total number of errors or to the fraction of errors over the size of the input [Grahne and Thomo 2004].

On relational data, this problem has been extensively studied under the notion of *constraint repair* (see e.g. [Arenas et al. 1999; Afrati and Kolaitis 2009]): in this case the specifications are given by relational integrity constraints, such as keys and foreign keys, and the problem asks to determine how much a database needs to be modified in order to satisfy a given constraint. This approach has been investigated for a variety of integrity constraints, starting with classical functional and inclusion dependencies [Arenas et al. 1999] and continuing with more expressive constraints such as tuple generating dependencies [Afrati and Kolaitis 2009]. Also, a number of different repair operators have been considered in the relational case, including insertions, deletions, and modifications of tuples. Besides finding repairs of relations, this line of research also focuses on querying inconsistent documents via their minimal repairs.

In the XML framework, malformed or non-conformant documents are more the rule than the exception [Chen et al. 2005; Ofuonye et al. 2010]. Indeed, a recent study [Grijzenhout and Marx 2013] shows that, although most XML documents are well-formed (more than 85%), only 25% of them reference a downloaded schema. Even worse, in this study it is shown that less than 10% of XML documents satisfy their downloaded schema. This means that most of the XML data on the web can be read, but only a small part of it can be processed automatically. In this scenario, it is natural to look for automatic repairing XML data with respect to a target schema. The idea is that an automatic repair process receives an invalid XML document and produces the best sequence of edit operations that results in a document satisfying the target schema. The edit operations should respect the nested structure of the XML document and modify the document in a minimal way.

The notion of repair for XML data is defined in a natural way when considering documents as trees: in this case, a repair can be simply defined as the *tree edit distance* [Tai 1979] between the input tree and the repaired tree, that is, the number of atomic edit operations that are needed to get from one tree to another. An atomic edit operation here amounts at inserting, deleting, or modifying a single node in a tree [Bille 2005]. Edit distance can then be lifted to a measure of distance $\mathsf{dist}(t, T)$ of a tree t from a specification T: this is nothing but the minimal distance of t to any tree satisfying T. In our setting, t can be seen as an XML document and T as an XML schema, and hence $\mathsf{dist}(t, T)$ measures how difficult it is to repair the data t so as to satisfy T. Furthermore, $\mathsf{dist}(t, T)$ can be computed efficiently when T is a regular language (e.g. DTD or XSD) specified by means of an automaton [Wagner 1974; Aho and Peterson 1972].

In this paper, we take the next step in repairing XML documents or trees: given two regular specifications S and T over trees, we aim at calculating how difficult it is, in the worst-case, to transform an object satisfying S into an object satisfying T. The

problem is motivated by considering S to be a *source*, i.e. a constraint that the input is guaranteed to satisfy, and T to be a *target*, i.e. a constraint that needs to be enforced. More precisely, we consider the worst-case over all trees t satisfying S of the minimum number of edit operations needed to transform t into some tree in T, that is,

$$\mathsf{cost}(S,T) \;=\; \sup_{t\in S} \mathsf{dist}(t,T) \;=\; \sup_{t\in S} \min_{t'\in T} \mathsf{dist}(t,t').$$

Of course, the above cost may be infinite. In this work, we isolate the pairs of schemas S and T such that $\mathsf{cost}(S,T)$ is finite and we give optimal procedures to decide when this happens, namely, when schema S is "almost contained" in schema T. Specifically, we say that S is *bounded repairable* into T when $\mathsf{cost}(S,T) < \infty$, that is, when every document t in S can be repaired to a document t' in T by applying a finite, uniformly bounded number of edits.

The notion of bounded repair is also motivated by the schema matching problem [Rahm and Bernstein 2001]: we would like to identify whether two schemas are semantically related. In our setting, the semantic relation between two schemas is considered at a very low level, namely, each schema is seen as a set of documents and not as a set of rules. Then the bounded repair problem states that a source schema S is related to a schema T if any XML document satisfying S can be transformed with a finite, uniformly bounded number of operations into a document satisfying T. Here, it is important to notice that our repair operations are designed to only consider the structural part of the data. Further research needs to be done to include in the analysis the data itself, e.g., the constraints between attribute values in XML documents, as this is a very important aspect to take into account when reasoning on and transforming XML documents.

The following examples give an account of some of the difficulties of telling whether one schema is bounded repairable into another.

Example 1.1. Recall that languages of unranked trees can be specified by means of DTDs, that is, by sets of rules of the form $a \to L_a$, where L_a is a regular language describing the possible sequences of children of an a-labeled node. For the sake of brevity, we will often omit from DTDs the rules of the form $a \to \varepsilon$, which denote the fact that a-labeled nodes are leaves.

Consider the following DTDs:

$$\begin{array}{rcl} S: r & \to & d\,c^* \\ d & \to & a^*\,b^* \end{array} \qquad\qquad \begin{array}{rcl} T: r & \to & a^*\,e \\ e & \to & b^*\,c^* \end{array}$$

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

Example 1.2. Consider the following DTDs:

$$\begin{array}{rcl} S': r & \to & a \\ a & \to & b^* \end{array} \qquad\qquad \begin{array}{rcl} T': r & \to & a \\ a & \to & b^*\,c \end{array}$$

It is easy to see that S' is bounded repairable into T': any tree $r(a(b,\ldots,b))$ in S' 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 S' and T' the rule $r \to a$

with the rule $r \rightarrow a^*$, we obtain a new pair of languages S'' and T'' such that S'' is not bounded repairable into T''. This example suggests that bounded repairability depends on some interplay between the rules of DTDs and, more generally, between the specifications of the labellings of the nodes at different levels of the trees.

We will deal with the notion of bounded repairability for schemas that are more general than DTDs, e.g., schemas that are given by regular tree languages [Schwentick 2007], and which capture the structural part of the W3C's XML schema [Fallside and Walmsley 2004]. We will formalize the edit distance between regular tree languages, and from this define the *bounded repair problem*, that is, the problem of deciding bounded repairability between two given tree languages S and T. Our main result is that it is decidable whether or not S can be repaired into T with a uniformly bounded number of edits.

For regular languages of words, the bounded repair problem was resolved in [Benedikt et al. 2013]. 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 more complex, both in terms of complexity and in terms of proof techniques that are required to solve it. We will provide a characterization of bounded repairability that exploits a suitable notion of component of a stepwise tree automaton [Carme et al. 2004], a form of automaton that turns out to be particularly convenient for analyzing repairs. An additional complication for the tree case is that we need to consider structures of connected components of stepwise tree automata that take the form of trees, rather than chains. Our characterization of the bounded repairability of S into T requires that every component structure of S 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 S and T.

With an effective characterization at hand, we can decide the bounded repairability problem, and with some additional optimizations we can give tight complexity bounds. It turns out that, differently from the string setting, the bounded repairability problem is equally complex no matter whether the tree languages are given by non-deterministic automata, deterministic automata, DTDs, or even non-recursive DTDs. Indeed, for all these cases the bounded repair problem is coNExp-complete. We then look for tractable cases that are obtained by further restricting the tree specifications. For example, we show that the bounded repairability problem becomes much simpler when the source alphabet is fixed and the languages are given by deterministic DTDs, or when the source language is assumed to be trivial, namely, the set of all trees over a given finite alphabet.

New material in this paper. Preliminary versions of some of the results in this paper appeared in [Anonymous]. However, this paper contains substantial new material. We include a full proof of the main characterization result (Theorem 5.7). The upper bounds to the number of repairs (Lemma 5.5 and Proposition 5.8) are also new. As concerns the complexity of deciding bounded repairability, the paper provides new complexity bounds that are moreover tight. In [Anonymous], it was shown that the bounded repair problem for regular tree languages is decidable in Π_2^{Exp} and is Exp-hard. Here, we show that the problem is actually coNExp-complete (Theorems 7.3 and 7.4). We finally include new examples and proofs of other claims that were omitted in [Anonymous].

Organization. The paper is organized as follows. In Section 2 we discuss some related work. In Section 3 we give some preliminaries on trees and regular tree languages and in Section 4 we define the bounded repairability problem for tree languages. In Section 5 we give the formal statement of our main result, that is, a characterization of those pairs of schemas that are bounded repairable. Section 6 gives a detailed proof of the characterization. In Section 7 we analyze in detail the complexity of the bounded repairability problem. In Section 8 we give another, simple characterization of bounded repairability for the case where the source language is universal, and we accordingly derive new complexity results. Finally, in Section 9 we give our conclusions and future work.

2. RELATED WORK

Ever since their conception, computers required the input data to follow a set of strict structural and semantic rules and the failure to do so typically resulted in operations producing unpredictable outputs, a well-know phenomenon of *Garbage In, Garbage Out* [Babbage 1864]. While initially this phenomenon was attributed to situations of erroneous manual data entry, with the ever increasing number of applications exchanging data the phenomenon has gained a new meaning describing potential problems occurring when two applications attempt to communicate with incompatible protocols [Lidwell et al. 2010]. While our research aims at solving the problems of the latter scenario, very early in the development of computer science we did see solutions to the problems resulting from erroneous manual data entry. One prominent example is the work on error-correcting parser for context-free languages [Aho and Peterson 1972], where a malformed input string is repaired by applying a (minimal) number of editing operations that make it conform to the given grammar. In [Korn et al. 2013] Kron et al., consider a very similar problem for XML where a serialization of an XML document is not well-formed (e.g., mismatching opening and closing tags or misspelled tag names) and is repaired to allow parsing into an XML tree. A slightly different variant of the problem is repairing well-formed XML with respect to a given schema, in the form of a DTD [Suzuki 2005; Staworko and Chomicki 2006] or XML Schema [Staworko et al. 2008]. The validity of the XML document is restored using a minimal set of editing operations (insertion, deletion, and renaming of nodes). Adding a move operation that can change the relative order of elements is challenging because of fundamental computational limitations [Cormode and Muthukrishnan 2007] and approximate measures have been studied for this operation [Boobna and de Rougemont 2004]. Furthermore, repairing XML documents with respect to analogues of classical relational constraints (key and inclusion dependencies) has also been studied [Flesca et al. 2005]. HTML documents often violate the syntactic rules of well-formedness and the structural rules imposed by the HTML standard, and consequently, repairing them requires methods that diligently combine the approaches of editing the textual serialization and editing the tree representation of the input document [Chen et al. 2005; Ofuonye et al. 2010].

The problem we study is, however, more general than repairing a single input XML document w.r.t. a given schema, because we are interested in repairing any input document drawn from a possibly infinite set of documents with a number of operations that is independent on the size of the XML document. One could attempt to approximate the bound on the number of required editing operations by randomly generating the input document [Antonopoulos et al. 2013] and then computing their edit distance to the target regular language. Because in our setting, the input document is drawn from a regular language and is repaired w.r.t. another regular language, the problem is in fact a generalization of a well known and thoroughly-studied problem of containment of two schemas [Comon et al. 2007; Colazzo et al. 2013; Martens et al. 2009]. Closely

related is the problem of measuring similarity between two schemas based on a notion of embeddings studied in [Fan and Bohannon 2008]. There are, however, significant differences: on the one hand, our semantic characterization of bounded reparability is stronger than the structural similarity determined with embeddings but on the other hand, the framework of [Fan and Bohannon 2008] introduces an additional challenge: it requires the embeddings to be information preserving i.e., for any query that can be evaluated on a document from the source schema there exists an equivalent query over the corresponding document from the target schema.

The preservation of information expressed with queries is the essence of data exchange source-to-target mappings and finding (target) solutions for a given source document is a difficult problem in the context of XML [Arenas and Libkin 2008; Amano et al. 2009]. In this setting, the problem of absolute consistency [Bojańczyk et al. 2011], checking that a solution exists for any possible source instance, bears strong resemblance to the problem of reparability except that it does not call for using editing operations, and consequently, it does not impose any limit on the number of editing operations but merely inquires the possibility of always finding a solution. In fact, the authors propose a solution that uses a notion of a *kind* in a manner analogous to the connected components used in our approach.

3. REGULAR LANGUAGES OF TREES

In this paper, we will work with finite unranked ordered trees whose nodes are labeled over a finite alphabet Σ. Formally, the set of *finite unranked ordered trees over* Σ (hereafter, simply *trees*) is inductively defined as follows: (1) every symbol $a \in \Sigma$ is a tree; (2) if $a \in \Sigma$, $n \in \mathbb{N}$, and $t_1, \ldots, t_n$ are trees, then $a(t_1, \ldots, t_n)$ is a tree. A sequence of Σ-trees $t_1 \cdot \ldots \cdot t_n$ is called a *forest*. As an example, the left-hand side of Figure 1 shows a tree over the alphabet $\Sigma = \{r, a, b, c, d\}$. We denote by $\mathcal{T}_\Sigma$ the set of all trees over Σ. A (*tree*) *language over* Σ is any subset L of $\mathcal{T}_\Sigma$.

It is useful to identify nodes of an unranked tree with sequences of positive natural numbers. Given a tree t of the form $a(t_1, \ldots, t_n)$, its domain is the subset of $\mathbb{N}^*$ that is formally defined as $\mathsf{nodes}(t) = \{\varepsilon\} \cup \{i \cdot x \mid x \in \mathsf{nodes}(t_i) \wedge 1 \leq i \leq n\}$. Note that the root of a tree is represented by ε. Finally, for every node $x \in \mathsf{nodes}(t)$, we denote by $t(x)$ the label of x in t.

Given a tree t, we introduce two partial orders on the domain $\mathsf{nodes}(t)$, which are called *ancestor order* and *post-order* and are denoted by $\preccurlyeq_t^{\mathsf{anc}}$ and $\preccurlyeq_t^{\mathsf{post}}$, respectively. The ancestor order $\preccurlyeq_t^{\mathsf{anc}}$ is nothing but the prefix order on the sequences of positive natural numbers that identify the nodes of t, that is, $x \preccurlyeq_t^{\mathsf{anc}} y$ if and only if x is a prefix of y. The post-order $\preccurlyeq_t^{\mathsf{post}}$ is the total ordering on the nodes of t (i.e. sequences of natural numbers) defined by $x \preccurlyeq_t^{\mathsf{post}} y$ if and only if y is a prefix of x or there exist $z, x', y' \in \mathbb{N}^*$ and $i, j \in \mathbb{N}$ such that $x = z \cdot i \cdot x'$, $y = z \cdot j \cdot y'$, and $i \leq j$.

DTDs. We will manipulate regular languages of trees mainly by means of automaton-based specifications, which will be formally defined in the next paragraphs. However, we will use less expressive specifications, such as XML Document Type Definitions, to give examples of simple tree languages.

An *XML Document Type Definition* (*DTD* for short) is defined as a tuple $D = (\Sigma, d, I)$, where Σ is a finite alphabet, d is a function that maps symbols from Σ to regular expressions over Σ, and $I \subseteq \Sigma$ is the set of initial symbols [Comon et al. 2007]. A tree t satisfies the DTD D if $t(\varepsilon) \in I$ and, for every $x \in \mathsf{nodes}(t)$, the word $t(x \cdot 1) \cdot \ldots \cdot t(x \cdot n)$ belongs to the language defined by the regular expression $d(t(x))$, where n is the number of children of x in t. We denote by $\mathscr{L}(D)$ the language of trees satisfying the DTD D. We will often omit the rules of the form $a \to \varepsilon$, as well as the set of initial symbols from the definition of a DTD when this set is understood from the context (e.g.

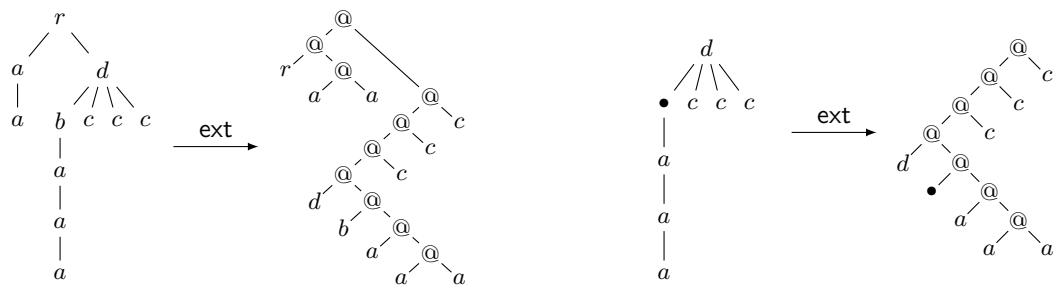


Fig. 1. Curry encodings of an unranked tree and a context.

when it consists of a single symbol r). As an example, consider the DTD:

$$D: \quad r \to a\,d \qquad d \to b\,c^*$$
$$a \to a + \varepsilon \qquad b \to a$$

One can easily check that the left-hand side tree of Figure 1 satisfies the above DTD D.

It is known that DTDs define a proper subclass of regular tree languages [Martens et al. 2006; Martens and Niehren 2007]. Quite interestingly, most of our complexity lower bounds for the bounded repair problem hold even for languages defined by DTDs (see Section 7 for more details).

We will also consider languages defined by non-recursive deterministic DTDs, which are often used in practice and have been studied extensively in [Segoufin and Vianu 2002; Segoufin and Sirangelo 2007]. Let us define the dependency graph of a DTD $D = (\Sigma, d, I)$ as the directed graph whose nodes are the letters in Σ and whose edges connect any letter a to a letter b whenever b occurs in the language specified by the regular expression $d(a)$. A DTD D is called *non-recursive* if its dependency graph is acyclic. A DTD D is called *deterministic* if each regular expression $d(a)$ is one-unambiguous (namely, it can be equally seen as a deterministic finite state automaton) [Brüggemann-Klein and Wood 1998] and the set of initial symbols is a singleton.

Curry encoding. To ease the definitions of the automaton model and the reasoning on tree repairs, we introduce here the notion of *curry encoding*, also known as extension encoding, of a tree [Carme et al. 2004; Martens and Niehren 2007]. According to this encoding any unranked tree over Σ is seen as a binary tree with leaves labeled over Σ and internal nodes labeled by a distinguished symbol @. Formally, the *curry encoding* is the function ext that injectively maps unranked trees to binary trees as follows:

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

To ease readability, we use the symbol @ as a binary, infix, left-associative operator: for instance, $\mathsf{ext}(a(t_1, \ldots, t_n)) = a@\mathsf{ext}(t_1)@\ldots@\mathsf{ext}(t_n)$. The left-hand side of Figure 1 illustrates the encoding of an unranked tree. The inverse ext^{-1} of the encoding is defined by providing the symbol @ with the semantics of the extension operator on unranked trees 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 observe that there is a one-to-one correspondence between the nodes of an unranked tree and the leaves of the curried encoding. In particular, the root node of an unranked tree corresponds to the left-most leaf of its curry 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 corresponding 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.

Hereafter, we will identify unranked trees with their curry encodings. In particular, by a slight abuse of notation, we will denote by $\mathcal{T}_\Sigma$ the set of all curry encodings of trees over Σ, and by $\mathsf{nodes}(t)$ the domain of the curry encoding of a tree t.

Contexts. We now fix another special symbol $\bullet \notin \Sigma$ that will be used as a placeholder for contexts. Formally, a (curried) *context* over Σ is the curry encoding of a tree over the alphabet $\Sigma \cup \{\bullet\}$, with a single node labeled by $\bullet$ (note that, in the curry encoding, the symbol $\bullet$ must occur in a leaf). We denote by $\mathcal{C}_\Sigma$ the set of all contexts over Σ. The *empty* context is the context $\bullet$ having exactly one node. The right-hand side of Figure 1 illustrates the encoding of a context. A context C is *horizontal* if the placeholder $\bullet$ is the leftmost leaf of C. We point out that a horizontal context has the form $\bullet @ t_1 @ \ldots @ t_n$ and represents the *forest* – i.e. a sequence of trees – $(\mathsf{ext}^{-1}(t_1), \ldots, \mathsf{ext}^{-1}(t_n))$. Note that the empty context is horizontal.

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 $\mathcal{C}_\Sigma$). The composition of two horizontal contexts is also horizontal and corresponds to concatenation of the corresponding forests of unranked trees. Note, however, that the difference in the order of context composition and the order of forest concatenation: if $C = \bullet @ t_1 @ \ldots @ t_n$ and $C' = \bullet @ t'_1 @ \ldots @ t'_m$, then $C \circ C' = \bullet @ t'_1 @ \ldots @ t'_m @ t_1 @ \ldots @ t_n$, which represents the forest $(\mathsf{ext}^{-1}(t'_1), \ldots, \mathsf{ext}^{-1}(t'_m), \mathsf{ext}^{-1}(t_1), \ldots, \mathsf{ext}^{-1}(t_n))$.

Stepwise tree automata. We use stepwise tree automata to specify regular tree languages. These are essentially bottom-up tree automata running on the curry encodings of trees [Carme et al. 2004; Martens and Niehren 2007; Comon et al. 2007]. 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 the automaton $\mathcal{A}$ 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 3.1. The following will serve as our running example. Consider two DTDs:

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

The following two stepwise automata capture (modulo the curry encoding) the languages defined by the previous DTDs (the underlined states are final and each rule

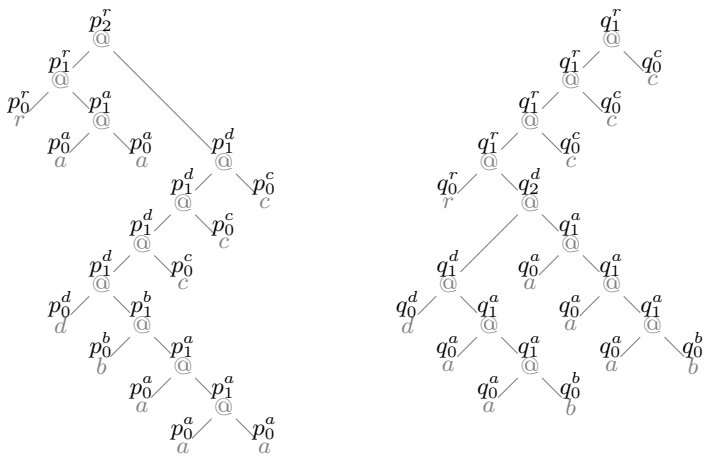


Fig. 2. Runs of two stepwise tree automata over curried trees.

with $q^a_?$ translates to two rules with q^a_0 and q^a_1):

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

Figure 2 presents the (accepting) runs of the automata $\mathcal{S}$ and $\mathcal{T}$ on some curried trees.

Stepwise automata capture exactly the class of regular (unranked) tree languages [Carme et al. 2004] and they are more succinct than other models of automata [Martens and Niehren 2007]. Even though other equivalent models of automata, such as unranked tree automata, are more frequently used in practice, these can be converted into stepwise tree automata in polynomial time. This means that algorithms for analyzing stepwise automata provide the same complexity bounds for unranked tree automata – in particular, all of our complexity results apply to stepwise tree automata as well as to unranked tree automata. The main advantage of using stepwise automata in our proofs is due to their ability of capturing in a uniform way the "cyclic behavior" of a regular tree language (as we will see in Section 5, this cyclic behavior is defined in terms of strongly connected components of automata).

In the sequel, we will assume that our stepwise tree automata are trimmed, namely, they contain only states that appear in valid accepting runs. Formally, 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 $\mathcal{A}$ on t such that $\rho(x) = q$ for some $x \in \mathsf{nodes}(t)$. Every stepwise tree automaton can be trimmed in linear time [Comon et al. 2007]. Given that all problems considered in this paper are at least P-hard, the assumption that all stepwise tree automata are trimmed is without loss of generality.

As it 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,

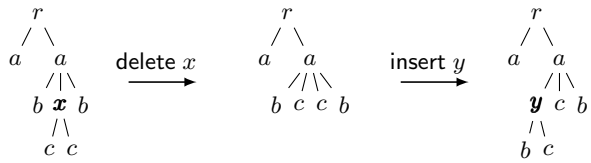


Fig. 3. Edit operations on unranked trees.

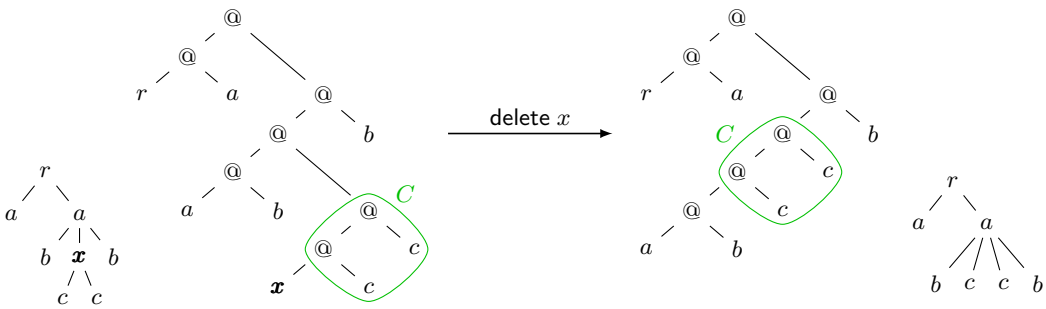


Fig. 4. Deleting a node in the curry encoding.

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). In particular, we have $\delta^*_\bullet(q, \bullet) = \{q\}$. By an abuse of notation, we will denote δ^* and $\delta^*_\bullet$ simply by δ.

4. THE BOUNDED REPAIR PROBLEM FOR TREES

We repair trees by using the standard set of edit operations over nodes [Tai 1979; Bille 2005]. We briefly recall the definitions of the standard edit operations on unranked trees which are extensions of the edit operations over words. The first operation, called *deletion*, removes a distinguished (non-root) node x from a tree t and promotes the subtrees x as children of its parent. The second operation, called *insertion*, adds a new node x in an unranked tree t, with a possible adoption of a list of consecutive children of the parent of x whose original position immediately follows the position of x. Figure 3 gives an example of these two operations. The last operation, called *relabeling*, modifies the label of a node x to a new label in Σ. These three operations are the standard edit operations that are used to define the edit-distance between trees (see [Bille 2005] for a survey). 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'. 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. Therefore, allowing or not the use of the operation of relabeling has an impact on the edit distance between two trees. However, the bounded repair problem is equivalent following that the relabeling is allowed or simulated by insertion and deletion operations.

We identify a close correspondence between the two basic edit operations on unranked trees and two operations on the corresponding curry encodings. Because the operations on unranked trees involve changing the parent of a sequence of consecutive subtrees (forests), the operations on curry encodings involve moving corresponding horizontal contexts. Deleting an inner node x in an unranked tree (cf. Figure 4) corresponds to deleting the corresponding leaf node x in the curry encoding, identifying

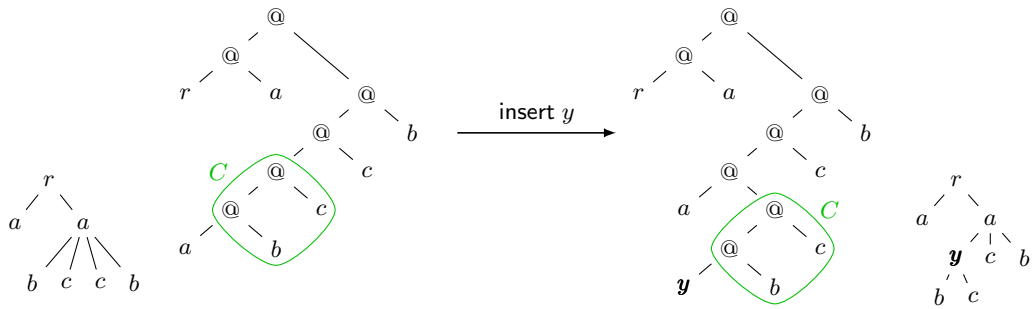


Fig. 5. Inserting a node in the curry encoding.

the horizontal context C that represents the sequence of children of the deleted node, and replacing the parent @ of C by C. Note that the context C is uniquely defined in the curry encoding and always has a parent @ because we apply the deletion operation to an inner node in the unranked tree. Conversely, inserting a node y in an unranked tree (cf. Figure 5) corresponds to identifying a context C representing the forest of the adopted children, placing a new @ node in place of C while attaching C as the right child of the new node @, and finally, substituting the placeholder of C by the new node y. Here the context C is also uniquely defined by the sequence of consecutive children adopted by the inserted node.

We are interested in studying the *bounded repairability problem*. This problem was studied for strings by [Benedikt et al. 2013], so we extend their setting from strings to trees. We consider two finite alphabets Σ and Δ and regular languages $S \subseteq \Sigma^*$ and $T \subseteq \Delta^*$, called the *source* and *target* languages, respectively. Furthermore, we define a *repair strategy* as any function from trees in S to trees in T for any tree languages S and T.

We are now ready to introduce the problem we are mainly interested in:

Definition 4.1. Given two regular tree languages S and T, let

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

be the *worst-case cost of repairing S into T* – note that this can be equally defined as the minimum of $\max_{t \in S} \mathsf{dist}(t, f(t))$ over all repair strategies f from S to T.

If $\mathsf{cost}(S,T)$ is finite, then we say that S *is bounded repairable into* T, and we write $\mathsf{cost}(S,T) < \infty$ for short. Intuitively, this is equivalent to saying that there is a repair strategy f transforming any tree $t \in S$ into a tree $f(t) \in T$ and having $\mathsf{dist}(t, f(t))$ uniformly bounded by a constant.

The *bounded repair problem* amounts at deciding, given two regular tree languages S and T – specified by means of stepwise tree automata or DTDs – whether S is bounded repairable into T.

Example 3.1 (*continued*). Consider the two DTDs D and D' that we introduced in our running example. For the tree languages specified by D and D' one can check $\mathscr{L}(D)$ is bounded repairable into $\mathscr{L}(D')$. Figure 6 shows how to repair a tree satisfying D into one satisfying D' with five edits: first one removes the d-labeled node under the root and its b-labeled child, then one adds a new d-labeled node above the two branches starting with a, finally one adds b-labeled leaves under the $r\ d\ a\ a$ branches. In fact, similar strategies with edit cost at most five can be used to repair any tree $t \in \mathscr{L}(D)$ into a tree $t' \in \mathscr{L}(D')$. In particular, this shows that $\mathsf{cost}(\mathscr{L}(D), \mathscr{L}(D')) < \infty$.

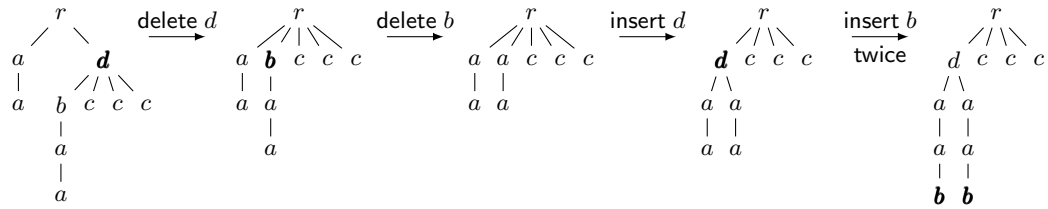


Fig. 6. Example of how to repair an unranked tree satisfying D into one satisfying D'.

It is easy to verify that the bounded repairability relation $\mathsf{cost}(S,T) < \infty$ satisfies the following key properties, which shall be used later:

(1) Subset-subsumption, i.e. $S \subseteq T$ implies $\mathsf{cost}(S,T) < \infty$;
(2) Transitivity, i.e. $\mathsf{cost}(S,T) < \infty$ and $\mathsf{cost}(T,U) < \infty$ imply $\mathsf{cost}(S,U) < \infty$;
(3) Union-compatibility, i.e. $\mathsf{cost}(S,T) < \infty$ and $\mathsf{cost}(S',T') < \infty$ imply $\mathsf{cost}(S \cup S', T \cup T') < \infty$.

This first property, *subset-subsumption*, is trivial to prove given that $\mathsf{cost}(S,T) = 0$ iff $S \subseteq T$. 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 $\mathsf{dist}(t,t'') \leq \mathsf{cost}(S,T) + \mathsf{cost}(T,U)$ given that $\mathsf{dist}(t,t') \leq \mathsf{cost}(S,T)$ and $\mathsf{dist}(t',t'') \leq \mathsf{cost}(T,U)$ for any $t \in S$, $t' \in T$, and $t'' \in U$. Thus, we conclude that $\mathsf{cost}(S,U)$ 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 source into a target language. Indeed, if $\mathsf{cost}(S,T) < \infty$ and $\mathsf{cost}(S',T') < \infty$, then $\mathsf{cost}(S \cup S', T \cup T') \leq \max\{\mathsf{cost}(S,T), \mathsf{cost}(S',T')\}$ and we conclude that $\mathsf{cost}(S \cup S', T \cup T')$ is bounded as well.

5. CHARACTERIZATION OF BOUNDED REPAIRABILITY

In this section we give an effective characterization of the bounded repairability relation between regular tree languages. Similarly to the string setting [Benedikt et al. 2013], this characterization is based on the notion of strongly connected component of the transition graph of a stepwise automaton. In the string case, a suitable coverability relation between chains of components is used to characterize bounded repairability. Because here we work with trees, we need to generalize the notion of coverability to a relation over the so-called synopsis trees, i.e., full binary trees with nodes labeled by strongly connected components.

5.1. 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:

$$E_h = \big\{(q_1,q) \in Q \times Q \;\big|\; \exists q_2.\ q \in \delta(q_1,q_2)\big\},$$
$$E_v = \big\{(q_2,q) \in Q \times Q \;\big|\; \exists q_1.\ q \in \delta(q_1,q_2)\big\}.$$

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 7 depicts the transition graphs of the automata $\mathcal{S}$ and $\mathcal{T}$ of Example 3.1 (dashed arrows represent horizontal edges, solid arrows represent vertical edges).

Recall that a *strongly connected component* of a graph (or simply a *component*) is a maximal set of nodes X such that every two nodes x, y in X are connected by a direct path. By $\mathsf{SCC}(\mathcal{A})$ we denote the set of all strongly connected components in the

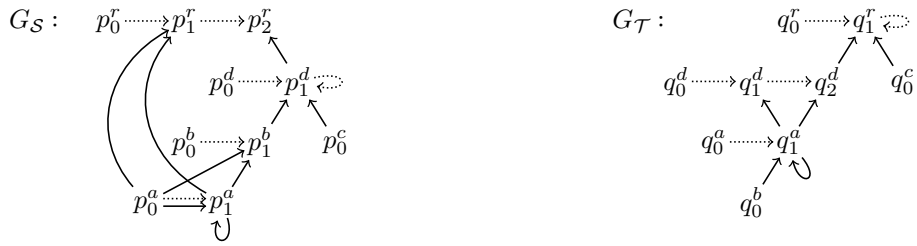
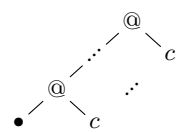


Fig. 7. Transition graphs (dashed and solid arrows represent horizontal and vertical edges, respectively).

transition graph of $\mathcal{A}$. In a way similar to the string setting, 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:

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

For example, the contexts realizable within the component $\{p_1^d\}$ of the automaton of our running example (see also Figure 7) are all of the form



Because editing operations on unranked trees correspond to operations involving horizontal contexts in the curry encodings, we identify strongly connected components of an automaton that yield only horizontal contexts. A proper manipulation of those components translates to performing a fixed number of editing operations regardless of the contexts such components define, which is the basis of characterizing bounded repairability. Formally, a component $X \in \mathsf{SCC}(\mathcal{A})$ is *horizontal* iff $\mathscr{L}(\mathcal{A} \mid X)$ consists of horizontal contexts 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 7. 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 $\bullet$, $a@\bullet$, $a@(a@\bullet)$, ...

5.2. Synopsis trees

We now introduce a suitable structure that eases the characterization of bounded repairability and that we call synopsis tree. The structure is a generalization of the chain of components that is used in Theorem 4.1 [Benedikt et al. 2013] to characterize bounded repairability between string languages. Formally, a *synopsis tree* of an automaton $\mathcal{A}$ is any full binary tree whose nodes are labeled with elements of $\mathsf{SCC}(\mathcal{A})$. The language $[\![\sigma]\!]_{\mathcal{A}}$ of curried trees that is induced by a synopsis tree σ of $\mathcal{A}$ is defined recursively as follows:

$$[\![X]\!]_{\mathcal{A}} \;=\; \big\{C \circ a \;\big|\; C \in \mathscr{L}(\mathcal{A} \mid X),\; a \in \Sigma\big\}$$
$$[\![X(\sigma_1, \sigma_2)]\!]_{\mathcal{A}} \;=\; \big\{C \circ (t_1@t_2) \;\big|\; C \in \mathscr{L}(\mathcal{A} \mid X),\; t_1 \in [\![\sigma_1]\!]_{\mathcal{A}},\; t_2 \in [\![\sigma_2]\!]_{\mathcal{A}}\big\}$$

with $X \in \mathsf{SCC}(\mathcal{A})$. Intuitively, all trees in the language $[\![\sigma]\!]_{\mathcal{A}}$ are obtained by combining, in a suitable order that is compatible with the structure of σ, some contexts which are realizable within the components of σ. Figure 8 contains two synopsis trees σ and τ,

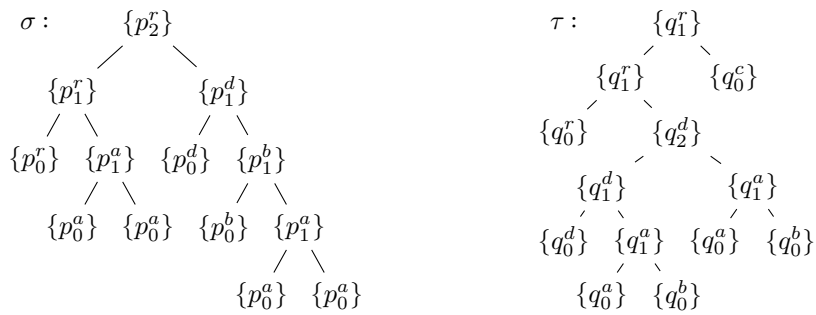


Fig. 8. Synopsis trees for stepwise tree automata $\mathcal{S}$ and $\mathcal{T}$.

respectively for the source automaton $\mathcal{S}$ and the target automaton $\mathcal{T}$ of Example 3.1. An example of tree induced by the synopsis tree on the left is that of Figure 1.

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

Definition 5.1. A *primitive synopsis tree* of an automaton $\mathcal{S} = (\Sigma, Q, \delta, \delta_0, F)$ is a synopsis tree σ of $\mathcal{S}$ such that:

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

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

We observe that the second property stated in Definition 5.1 is equivalent to asking that every component appears at most once in every path of a primitive synopsis tree.

As an example, the tree σ depicted to the left of Figure 8 is a primitive synopsis tree, and it corresponds to the run on the left-hand side of Figure 2 of the automaton $\mathcal{S}$ of Example 3.1. On the other hand, the synopsis tree τ depicted to the right is not primitive.

The idea underlying the notion of primitive synopsis tree is to capture the "cyclic behavior" of the components of the source 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 source language $\mathscr{L}(\mathcal{S})$ is *sound*, in the sense that $\mathscr{L}(\mathcal{S})$ is contained in the union of the languages induced by primitive synopsis trees:

LEMMA 5.2. *For every stepwise tree automaton $\mathcal{S}$, we have* $\mathscr{L}(\mathcal{S}) \subseteq \bigcup_{\sigma \in \mathsf{PST}(\mathcal{S})} [\![\sigma]\!]_{\mathcal{S}}$.

Before entering the details of the proof of Lemma 5.2, we illustrate the main ideas on the example tree t from Figure 1 accepted by the automaton $\mathcal{S}$ from Example 3.1. We use the accepting run of $\mathcal{S}$ on t (cf. Figure 2), and in particular the transitions in it that induce a change of component along both successors, to decompose t into a binary tree structure, where each node represents a context realizable by some component of $\mathcal{S}$. We present this decomposition in Figure 9, where for a better visualization we annotate the states not on the nodes but on the edges above them (this requires adding a virtual edge entering the root).

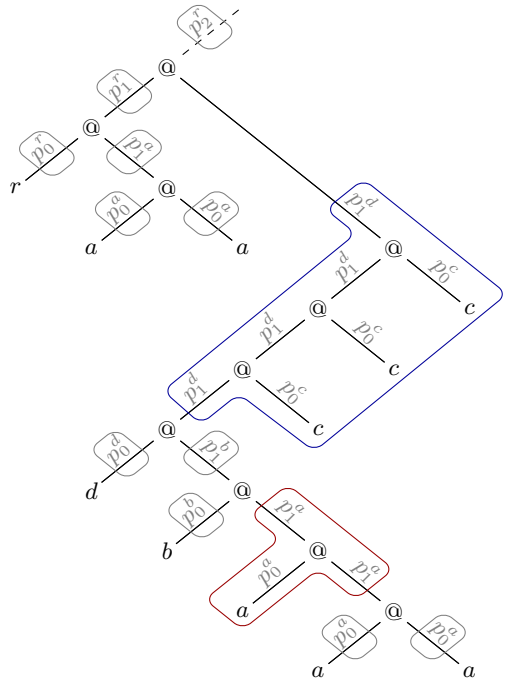


Fig. 9. Decomposition of a curry tree into contexts.

PROOF OF LEMMA 5.2. Fix a curried tree $t \in \mathscr{L}(\mathcal{S})$ and an accepting run ρ of $\mathcal{S}$ on t. We need to construct a primitive synopsis tree σ such that $t \in [\![\sigma]\!]_{\mathcal{S}}$. Recall that a synopsis tree is a tree whose nodes are labeled with strongly connected components of $\mathcal{S}$. To construct σ, we first decompose t into pieces: this will result in a tree-shaped arrangement of contexts, which we call context decomposition of t. We then show how to turn the context decomposition of t into the desired primitive synopsis tree.

Formally, we represent a *context decomposition of* t as a subset D of the nodes of t satisfying the following conditions:

(1) D contains the root of t,
(2) for every node x in D, either x is a leaf of t or there is a descendant y of x in t such that (i) the states $\rho(x)$ and $\rho(y)$ belong to the same strongly connected component of $\mathcal{S}$, (ii) both successors $y \cdot 1$ and $y \cdot 2$ belong to D, and (iii) for every node $z \in D$, if z is a proper descendant of x, then z is a proper descendant of y too.

Note that the node-to-child relation of t induces an analogous structure on any set D of the above form. In particular, we can think of a context decomposition D as a full binary tree. We further associate with each internal node x of D the context $D(x)$ that is obtained by selecting the portion of the tree t that lies between x and the unique node y such that $y \cdot 1$ and $y \cdot 2$ are children of x in D – the node that corresponds to y in the resulting context is labeled with the placeholder $\bullet$. The *flattening* $[\![D]\!]$ of a context decomposition D is the language of trees that is inductively defined as follows:

$$[\![D]\!] = \Sigma \qquad \text{if } D \text{ has a single node}$$
$$[\![D]\!] = \left\{ C_0(t_1 @ t_2) \;\middle|\; t_1 \in [\![D_1]\!],\ t_2 \in [\![D_2]\!] \right\} \qquad \text{if } D \text{ has more than one node}$$

where, in the second line, C_0 is the context associated with the root of D and D_1 and D_2 are the subtrees of D (D_1 and D_2 can be seen as context decompositions of two disjoints subtrees of t). It is easy to see that $t \in [\![D]\!]$ for every context decomposition D of t. We also associate with each decomposition D of t an induced synopsis tree σ_D by replacing every context C labeling an internal node x of D with the strongly connected component X of the state $\rho(x)$ (note that $C \in \mathscr{L}(\mathcal{S} \mid X)$).

Due to the similarity between the definition of the flattening $[\![D]\!]$ and the definition of the language $[\![\sigma_D]\!]_{\mathcal{S}}$ induced by the synopsis tree σ_D, we have that $[\![D]\!] \subseteq [\![\sigma_D]\!]_{\mathcal{S}}$, whence $t \in [\![\sigma_D]\!]$. However, given a generic context decomposition D, there is no guarantee that σ_D is a *primitive* synopsis tree – in particular, it may happen that two consecutive nodes in σ_D are labeled with the same component. To overcome this problem, below we show how to construct a specific context decomposition D of t that satisfies the following additional property:

(3) if x is an internal node of D and $y \cdot 1$ and $y \cdot 2$ are the two immediate successors of x in D, then the component of $\rho(x)$ is different from the components of $\rho(y \cdot 1)$ and $\rho(y \cdot 2)$.

Clearly, the additional property suffices to conclude that σ_D is a primitive synopsis tree, and thus to prove the lemma.

To construct a context decomposition D of t that satisfies the properties (1)–(3), we follow maximal paths within the same component in the run ρ. More precisely, let x be the root of t and let X be the component of $\rho(x)$. We distinguish two cases depending on whether or not there is a leaf y of ρ whose state belongs to the component X. If there is such a leaf y, then we define the context decomposition D of t to be the set containing only the two nodes x and y. In this case, the corresponding synopsis tree σ_D is clearly primitive. Otherwise, if all the states associated with the leaves of ρ are outside X, we choose any maximal path of ρ that starts in x and visits only states within the component X. Let y be the last node of this path. Clearly, y is not a leaf and both states $\rho(y \cdot 1)$ and $\rho(y \cdot 2)$ are outside X. By exploiting a simple inductive argument, we can assume that the two subtrees of t rooted at nodes $y \cdot 1$ and $y \cdot 2$ admit some context decompositions D_1 and D_2 satisfying (1)–(3). We can thus define our context decomposition D of t to be the set $\{x\} \cup D_1 \cup D_2$. It is routine to verify that D satisfies the properties (1)–(3) and induces a primitive synopsis tree σ_D.

Summing up, we constructed from t and ρ a suitable context decomposition D and from this we derived the existence of a primitive synopsis tree σ_D such that $t \in [\![D]\!] \subseteq [\![\sigma_D]\!]_{\mathcal{S}}$. This concludes the proof of the lemma. □

We also observe the following:

Remark 5.3. The height of a primitive synopsis tree of the source automaton $\mathcal{S}$ is bounded by the number of components in $G_{\mathcal{S}}$ and hence by the number of states of $\mathcal{S}$. Consequently, $\mathsf{PST}(\mathcal{S})$ is a finite set and can be represented in exponential space with respect to the size of $\mathcal{S}$.

In order to represent the target language and the possible edited trees, one needs a relaxed version of primitive synopsis tree, called basic synopsis tree, which enforces only the first condition of Definition 5.1. Basic synopsis trees are the analogs of chains of components over $\mathrm{dag}^*(\mathcal{T})$ that were used in Theorem 4.1 [Benedikt et al. 2013] to characterize bounded repairability between string languages.

Definition 5.4. A *basic synopsis tree* of an automaton $\mathcal{T}$ is a synopsis tree τ of $\mathcal{T}$ that respects the transition function of $\mathcal{T}$ (cf. first item of Definition 5.1). We denote by $\mathsf{BST}(\mathcal{T})$ the set of all basic synopsis trees of $\mathcal{T}$.

For example, the tree τ in Figure 8 is a basic synopsis tree that respects the transitions of the run of the automaton $\mathcal{T}$ depicted in the right-hand side of Figure 2.

Differently from primitive synopsis trees, basic synopsis trees may contain repeated occurrences of the same component. This implies that the set $\mathsf{BST}(\mathcal{T})$ of all basic synopsis trees of $\mathcal{T}$ is potentially infinite. However, this set can be finitely presented by means of a deterministic binary bottom-up tree automaton of size polynomial in the size of $\mathcal{T}$.

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

LEMMA 5.5. *For every stepwise tree automaton $\mathcal{T} = (\Delta, Q, \delta, \delta_0, F)$ and every $\tau \in \mathsf{BST}(\mathcal{T})$, we have*

$$\mathsf{cost}\big([\![\tau]\!]_{\mathcal{T}}, \mathscr{L}(\mathcal{T})\big) \;\leq\; (4|\tau| + 1) \cdot 2^{|Q|}$$

where $[\![\tau]\!]_{\mathcal{T}}$ is now seen as a language of unranked trees. In particular, $\mathsf{cost}\big([\![\tau]\!]_{\mathcal{T}}, \mathscr{L}(\mathcal{T})\big) < \infty$.

PROOF. We begin by explaining the main ingredients of the proof. Any tree $t \in [\![\tau]\!]_{\mathcal{T}}$ can be seen as a composition of contexts, precisely, a context C_x for each node x in τ, with $C_x \in \mathscr{L}(\mathcal{T} \mid \tau(x))$. Every such context can be decorated by a partial run of $\mathcal{T}$ that justifies the fact that the context belongs to the language $\mathscr{L}(\mathcal{T} \mid \tau(x))$. These partial runs can be used to construct a complete run of $\mathcal{T}$, but only after the insertion of a small number of small pieces of runs, which provide the necessary connections between the partial run of a context C_x and the partial runs of the successor contexts $C_{x\cdot 1}$ and $C_{x\cdot 2}$. The existence of these pieces is guaranteed by the fact that the synopsis tree τ respects the transition function of $\mathcal{T}$ and by the definition of strongly connected component. We now turn to a more detailed proof.

For every state q of $\mathcal{T}$, we define the automaton $\mathcal{T}^q = (\Delta, Q, \delta, \delta_0, \{q\})$ that recognizes trees via runs that end with state q at the root. We also define $\ell_{p,q} = \min\{|C| \mid q \in \delta(p, C)\}$ for each pair of states p, q, with q reachable from p in $G_{\mathcal{T}}$. We take the maximum of all these values, i.e. $\ell = \max_{p,q \in Q} \ell_{p,q}$, and we observe that $\ell \leq 2^{|Q|}$. We can associate with each pair $p, q \in Q$ such that q is reachable from p a context $C_{p,q}$ of size at most ℓ such that $q \in \delta(p, C_{p,q})$. Below, we exploit an induction on the size of a basic synopsis tree τ to prove that, for all states $q \in \tau(\varepsilon)$,

$$\mathsf{cost}\big([\![\tau]\!]_{\mathcal{T}}, \mathscr{L}(\mathcal{T}^q)\big) \;\leq\; 4|\tau| \cdot \ell$$

Note that to get from the above statement to the claim of the lemma it is sufficient to recall that $\mathcal{T}$ is trimmed and hence any tree in $\mathscr{L}(\mathcal{T}^q)$ can be repaired into $\mathscr{L}(\mathcal{T})$ by simply inserting a context $C_{q,q'}$, where q' is some state from F.

To prove the statement in the base case $\tau = X$, we consider a generic tree $t \in [\![X]\!]_{\mathcal{A}}$ and we observe that $t = C \circ a$ for some context $C \in \mathscr{L}(\mathcal{T} \mid X)$ and some letter $a \in \Delta$. Clearly, there exist $p', q' \in X$ such that $q' \in \delta(p', C)$. Since $\mathcal{T}$ is trimmed, there exist a symbol $b \in \Delta$ and a state $q_0 \in \delta_0(b)$ such that p' is reachable from q_0. We define the tree

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

and we observe that t' can be obtained from t ($= C \circ a$) by replacing the leaf node a with the tree $C_{q_0,p'} \circ b$ and by adding the context $C_{q',q}$ at the top. Clearly, $t' \in \mathscr{L}(\mathcal{T}^q)$ and the overall cost of transforming t to t' is at most 4ℓ.

Now, for the inductive step, suppose that $\tau = X(\tau_1, \tau_2)$ and consider $t \in [\![X(\tau_1, \tau_2)]\!]_{\mathcal{T}}$. By definition, we have $t = C \circ (t_1 @ t_2)$ for some context $C \in \mathscr{L}(\mathcal{T} \mid X)$, and some trees $t_1 \in [\![\tau_1]\!]_{\mathcal{T}}$ and $t_2 \in [\![\tau_2]\!]_{\mathcal{T}}$. Further let $p', q' \in X$ be two states such that $q' \in \delta(p', C)$. Since the synopsis tree τ respects the transition function of $\mathcal{T}$, we know that there exist $p \in X$,

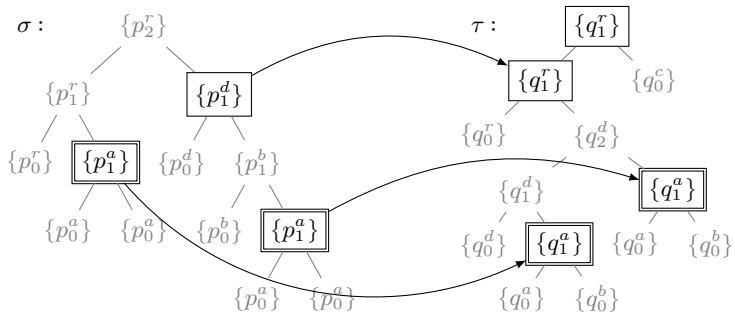


Fig. 10. Covering of a primitive synopsis tree by a basic synopsis tree.

$p_1 \in \tau_1(\varepsilon)$, and $p_2 \in \tau_2(\varepsilon)$ such that $p \in \delta(p_1, p_2)$. By inductive hypothesis, there exists $t'_1 \in \mathscr{L}(\mathcal{T}^{p_1})$ (resp., $t'_2 \in \mathscr{L}(\mathcal{T}^{p_2})$) such that $\mathsf{dist}(t_1, t'_1) \leq 4|\tau_1| \cdot \ell$ (resp., $\mathsf{dist}(t_1, t'_1) \leq 4|\tau_1| \cdot \ell$). We can then define

$$t' \;=\; C_{q',q} \circ C \circ C_{p,p'} \circ (t'_1 \mathbin{@} t'_2)$$

and claim that $t' \in \mathscr{L}(()\mathcal{T}^q)$. Moreover, the above tree t' can be obtained from the original tree $t = C \circ (t_1 \mathbin{@} t_2)$ by first transforming the subtrees t_1 and t_2 into t'_1 and t'_2, respectively, then inserting the context $C_{p,p'}$ between $t'_1 \mathbin{@} t'_2$ and C, and finally adding the context $C_{q',q}$ at the top. Overall, this transformation costs at most $4|\tau_1| \cdot \ell + 4|\tau_2| \cdot \ell + 2\ell \;\leq\; 4|\tau| \cdot \ell$. $\square$

5.3. Coverings

In the previous section, we introduced the concepts of primitive and basic synopsis trees and we showed that they correspond roughly (i.e. up to boundedly many edits) to trees accepted by the source and target automata, respectively. The remaining part of the puzzle is to relate each primitive synopsis tree of the source automaton $\mathcal{S}$ to some basic synopsis tree of the target automaton $\mathcal{T}$, so as to characterize bounded repairability from $\mathscr{L}(\mathcal{S})$ to $\mathscr{L}(\mathcal{T})$. This is accomplished by the notion of *covering* between synopsis trees.

Definition 5.6. Given two stepwise tree automata $\mathcal{S}$ and $\mathcal{T}$ and two synopsis trees σ of $\mathcal{S}$ and τ of $\mathcal{T}$, we say that σ is *covered* by τ iff there is an *injective* mapping λ from non-trivial nodes of σ to non-trivial nodes of τ such that:

(1) λ maps components in a way that is compatible with the languages of contexts, that is, $\mathscr{L}(\mathcal{S} \mid \sigma(x)) \subseteq \mathscr{L}(\mathcal{T} \mid \tau(\lambda(x)))$ for every non-trivial node x of σ;
(2) λ preserves the post-order of non-trivial nodes, that is, $x \preccurlyeq^{\mathsf{post}}_{\sigma} y$ iff $\lambda(x) \preccurlyeq^{\mathsf{post}}_{\tau} \lambda(y)$ for any two non-trivial nodes x, y of σ;
(3) λ preserves the ancestorship of non-horizontal nodes, that is, $x \preccurlyeq^{\mathsf{anc}}_{\sigma} y$ iff $\lambda(x) \preccurlyeq^{\mathsf{anc}}_{\tau} \lambda(y)$ for every non-horizontal node x of σ and every non-trivial node y of σ.

We call the mapping λ a *covering* from σ to τ and we denote it shortly by $\lambda : \sigma \hookrightarrow \tau$.

Figure 10 presents a covering of a primitive synopsis tree σ of $\mathcal{S}$ by a basic synopsis tree τ of $\mathcal{T}$, where the square boxes represent the non-trivial nodes, and have double borders when the component is non-horizontal.

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

THEOREM 5.7. *Given two stepwise automata $\mathcal{S}$ and $\mathcal{T}$, the language $\mathscr{L}(\mathcal{S})$ is bounded repairable into the language $\mathscr{L}(\mathcal{T})$ iff every primitive synopsis tree σ of $\mathcal{S}$*

is covered by some basic synopsis tree τ of $\mathcal{T}$, namely:

$$\mathsf{cost}(\mathcal{S},\mathcal{T}) < \infty \qquad \textit{iff} \qquad \forall\, \tau \in \mathsf{PST}(\mathcal{S}).\ \ \exists\, \sigma \in \mathsf{BST}(\mathcal{T}).\ \ \tau \hookrightarrow \sigma$$

The proof of the above result is given in Section 6. Here, we briefly explain the main ideas underlying the definition of covering. We begin by observing that a reasonable strategy for repairing $\mathcal{S}$ into $\mathcal{T}$ with uniformly bounded cost applies the edit operations only at the "junctions" of the contexts realized by the non-trivial components. Indeed, since non-trivial components of $\mathcal{S}$ can realize arbitrary large repetitions of the same context, we have that either these repetitions do not need any editing at all, or they need an arbitrary large amount of editing. This observation gives an intuitive account for the first condition of Definition 5.6, which enforces containment relationships between languages of contexts realizable within non-trivial components.

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 $t \in \mathscr{L}(\mathcal{S})$. Let us consider a node x in t that is about to be deleted by the editing. There is a unique way to represent the curry encoding of t together with the distinguished node x as an expression of the form $C \circ \big(t' \mathbin{@} (C' \circ a)\big)$, where a is the label of x and C' is a *horizontal* context representing the forest of subtrees under x. The result of the deletion of node x from t is encoded by the curried tree $C \circ C' \circ t'$ (see Figure 4 for an example). Note that this operation does not allow the deletion of the leftmost leaf node in the curried tree (this would correspond to deleting the root node in an unranked tree, an operation that is typically prohibited). The operation of inserting a new node y in an unranked tree t can be described in a symmetric way via curry encodings and transpositions of horizontal contexts, that is, given an unranked tree t with curry encoding $C \circ C' \circ t'$, where C' is a horizontal context, the curried tree $C \circ \big(t' @ (C' \circ a)\big)$ 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 that we just 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., the context C of Figure 4) with their descendants. These properties are precisely captured by the last two conditions of Definition 5.6.

We conclude the section by mentioning a strengthening of the "if" direction of Theorem 5.7, which gives an upper bound for the cost of an optimal repair strategy from $\mathscr{L}(\mathcal{S})$ to $\mathscr{L}(\mathcal{T})$:

PROPOSITION 5.8. *For all automata $\mathcal{S}$ and $\mathcal{T}$, if every primitive synopsis tree of $\mathcal{S}$ is covered by some basic synopsis tree of $\mathcal{T}$, then*

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

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

6. PROOF OF THE MAIN CHARACTERIZATION

The following subsections are devoted to proving the two directions of the characterization.

6.1. From covering to repair

We begin with the proof of the "if" direction of Theorem 5.7. For the rest of the section, we fix two stepwise automata $\mathcal{S} = (\Sigma, Q, \delta, \delta_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', \delta'_0, F')$ recognizing the source and the target languages, respectively. We then assume that every primitive synopsis tree of $\mathcal{S}$ is covered by some basic synopsis tree of $\mathcal{T}$, and we show how to construct a repair strategy from $\mathscr{L}(\mathcal{S})$ to $\mathscr{L}(\mathcal{T})$ with uniformly bounded cost. The proof

basically follows from a series of containments and bounded repairability relations between languages, which can be summarized as follows:

$$\underbrace{\mathscr{L}(\mathcal{S}) \;\subseteq\; \bigcup_{\sigma\in\mathsf{PST}(\mathcal{S})} [\![\sigma]\!]_{\mathcal{S}}}_{\text{Lemma 5.2}} \;\sqsubseteq\; \underbrace{\bigcup_{\tau\in\mathsf{BST}(\mathcal{T})} [\![\tau]\!]_{\mathcal{T}} \;\subseteq\; \mathscr{L}(\mathcal{T})}_{\text{Lemma 5.5}}$$

where $\sqsubseteq$ denotes the bounded repairability relation. The intermediate bounded repairability relation in the above chain is established by the following lemma:

LEMMA 6.1. *For every synopsis tree σ of $\mathcal{S}$ and every synopsis tree τ of $\mathcal{T}$, if σ is covered by τ, then* $\mathsf{cost}\big([\![\tau]\!]_{\mathcal{S}}, [\![\sigma]\!]_{\mathcal{T}}\big) \;\leq\; 4|\tau| + 4|\sigma|$.

The proof of the above lemma is quite technical and will take the entire subsection.

Before entering the details, we briefly discuss how the "if" direction of Theorem 5.7 follows from it. By Lemma 5.2, the source language $\mathscr{L}(\mathcal{S})$ is contained in the union of the languages $[\![\sigma]\!]_{\mathcal{S}}$ induced by all primitive synopsis trees $\sigma \in \mathsf{PST}(\mathcal{S})$. By the hypothesis, each of these synopsis trees is covered by some basic synopsis tree τ of the target automaton $\mathcal{T}$. Thus, by Lemma 6.1, each language $[\![\sigma]\!]_{\mathcal{S}}$ can be repaired with uniformly bounded cost into the language $[\![\tau]\!]_{\mathcal{T}}$, for some $\tau \in \mathsf{BST}(\mathcal{T})$. By Lemma 5.5 each language $[\![\tau]\!]_{\mathcal{T}}$ can be in turn repaired with uniformly bounded cost into the target language $\mathscr{L}(\mathcal{T})$. 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 that is moreover preserved by finite unions.

The rest of the subsection is devoted to the proof of Lemma 6.1. We begin by extending slightly the definition of synopsis tree and by allowing the use of special nodes labeled with ε that represents *dummy* trivial components. The semantics is extended in the natural way by letting $\mathscr{L}(\mathcal{A} \mid \varepsilon) = \{\bullet\}$ (for any stepwise automaton $\mathcal{A}$). Because all trivial components have the same associated language $\{\bullet\}$, we shall often identify trivial components of automata with the dummy component ε.

For a technical reason (see the proof of Lemma 6.3 below), we also need to assume that the alphabet Σ of the source automaton $\mathcal{S}$ is contained in the alphabet Δ of the target automaton $\mathcal{T}$ – note that this condition can be enforced without loss of generality, that is, without changing the recognized languages.

The first ingredient of the proof shows how to "interpolate" two synopsis trees σ and τ by a third synopsis tree θ of $\mathcal{S}$ in such a way that:

— θ has the same labels (i.e., components) as σ on the non-trivial nodes and it covers σ via a *bijection* between non-trivial nodes of σ and non-trivial nodes of θ that maps any non-trivial node of σ with label $X \in \mathsf{SCC}(\mathcal{S})$ to a non-trivial node of θ with the same label X (we say that σ is *strongly covered* by θ and denote this by $\sigma \hookrightarrow\!\!\!\!\rightarrow \theta$);
— θ has the same domain (i.e., set of nodes) as τ and it is covered by τ via the *identity* function between non-trivial nodes (we say that θ is *embedded* into τ and denote this by $\theta \leftrightharpoons \tau$).

It is not difficult to show that such an interpolating synopsis tree θ exists:

LEMMA 6.2. *We have that*

$$\sigma \hookrightarrow \tau \qquad \textit{implies} \qquad \exists\,\theta.\;\; \sigma \hookrightarrow\!\!\!\!\rightarrow \theta \leftrightharpoons \tau$$

that is, if σ is covered by τ, then there is a synopsis tree θ that strongly covers σ and that is embedded in τ.

PROOF. Let λ be a covering function from σ to τ and recall that λ is injective. Let $\mathsf{range}(\lambda)$ denote the set of nodes of τ of the form $\lambda(x)$, for some $x \in \mathsf{nodes}(\sigma)$. Moreover,

for every $y \in \mathsf{range}(\lambda)$, let $\lambda^{-1}(y)$ denote the unique node x of σ such that $\lambda(x) = y$. The synopsis tree θ has the same domain as τ and the same labels as σ, that is, for all $y \in \mathsf{nodes}(\theta) = \mathsf{nodes}(\tau)$,

$$\theta(y) \;=\; \begin{cases} \sigma(\lambda^{-1}(y)) & \text{if } y \in \mathsf{range}(\lambda), \\ \varepsilon & \text{otherwise.} \end{cases}$$

Note that λ can be seen as a bijection between the non-trivial nodes of σ and the non-trivial nodes of θ (the latter are precisely the nodes of τ that belong to $\mathsf{range}(\lambda)$). It follows that σ is strongly covered by θ via the function λ and that θ is embedded into τ via the identity function. □

A first advantage of considering an interpolating synopsis tree θ which is embedded into τ is that θ and τ have the same structure. As a consequence, we can claim that the language induced by θ is *contained* in (not just bounded repairable into) the language induced by τ:

LEMMA 6.3. *If θ is a synopsis tree of $\mathcal{S}$, τ is a synopsis tree of $\mathcal{T}$, and θ is embedded into τ, then $[\![\theta]\!]_{\mathcal{S}} \subseteq [\![\tau]\!]_{\mathcal{T}}$.*

PROOF. The proof is by structural induction on θ (or, equally, τ). In the base case, where θ consists of a single node x, we consider the components $\theta(x) = X$ and $\tau(x) = Y$. If X is a trivial component, then, since the covering function from θ to τ is a bijection between non-trivial nodes, we deduce that Y is also a trivial component and hence $\mathscr{L}(\mathcal{S} \mid X) = \mathscr{L}(\mathcal{T} \mid Y) = \{\bullet\}$. Now, recall that we assumed that the source alphabet Σ is contained in the target alphabet Δ. From this it follows that $[\![\theta]\!]_{\mathcal{S}} = \Sigma \subseteq \Delta = [\![\tau]\!]_{\mathcal{T}}$. Otherwise, if X is a non-trivial component, then from the fact that θ is covered by τ, we obtain $\mathscr{L}(\mathcal{S} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$. As before we conclude that $[\![\theta]\!]_{\mathcal{S}} \subseteq [\![\tau]\!]_{\mathcal{T}}$.

For the inductive step, we suppose that $\theta = X(\theta_1, \theta_2)$ and $\tau = Y(\tau_1, \tau_2)$. We consider a generic tree $t \in [\![\theta]\!]_{\mathcal{S}}$. By definition, we can write $t = C \circ (t_1 @ t_2)$ for some context $C \in \mathscr{L}(\mathcal{S} \mid X)$ and some trees $t_1 \in [\![\theta_1]\!]_{\mathcal{S}}$ and $t_2 \in [\![\theta_2]\!]_{\mathcal{S}}$. We know from the inductive hypothesis that $t_i \in [\![\tau_i]\!]_{\mathcal{T}}$ for both $i = 1$ and $i = 2$. If X is a trivial component, then C is necessarily the trivial context, which is also realizable within the component Y. Otherwise, if X is a non-trivial component, then it must be mapped to the component Y by the embedding function, and hence $C \in \mathscr{L}(\mathcal{T} \mid Y)$. In both cases we conclude that $t \in [\![\tau]\!]_{\mathcal{T}}$. □

Now, recall from Section 4 that the bounded repairability relation is transitive and it generalizes containment. In particular, the previous results reduce the statement of Lemma 6.1 to the problem of proving that $[\![\sigma]\!]_{\mathcal{S}}$ is bounded repairable into $[\![\theta]\!]_{\mathcal{S}}$. In proving the latter statement, we can take advantage of the fact that σ is strongly covered by θ. In particular, we observe that the strong coverability relation $\hookrightarrow\!\!\!\!\rightarrow$ is an *equivalence*: it is indeed reflexive, symmetric, and transitive (the last two properties follow from the fact that the function that witnesses strong coverability is a bijection between non-trivial nodes that preserves components).

In order to derive bounded repairability from strong coverability, we associate with each synopsis tree σ of $\mathcal{S}$ a suitable *normal form* σ^* that can be used as a *canonical representative* of the equivalence class of σ induced by the strong coverability relation. We will then prove that $[\![\sigma]\!]_{\mathcal{S}}$ is bounded repairable into $[\![\theta]\!]_{\mathcal{S}}$ first by repairing $[\![\sigma]\!]_{\mathcal{S}}$ into $[\![\sigma^*]\!]_{\mathcal{S}}$ and then by repairing $[\![\theta^*]\!]_{\mathcal{S}}$ (= $[\![\sigma^*]\!]_{\mathcal{S}}$) into $[\![\theta]\!]_{\mathcal{S}}$ (recall that σ and θ strongly cover each other and hence $\sigma^* = \theta^*$ by canonicity of the normal form). The repair strategy that witnesses bounded repairability between $[\![\sigma]\!]_{\mathcal{S}}$ and $[\![\sigma^*]\!]_{\mathcal{S}}$ (resp., $[\![\theta^*]\!]_{\mathcal{S}}$ and $[\![\theta]\!]_{\mathcal{S}}$) can be read off the sequence of generic editing operations that takes σ to its normal form σ^* (resp., θ to its normal form θ^*).

In the sequel we only manipulate synopsis trees of the source automaton $\mathcal{S}$. For this reason, we can omit the subscript $\mathcal{S}$ from notations like $[\![\sigma]\!]_{\mathcal{S}}$. We describe below the structure of a synopsis tree *in normal form*.

Definition 6.4. A synopsis tree σ is *in normal form* if one of the following cases holds:

(1) $\sigma = \varepsilon$, namely, σ consists of a single node labeled with a trivial component,
(2) $\sigma = X(\alpha, \varepsilon)$, where X is a non-trivial horizontal component and α is a synopsis tree in normal form,
(3) $\sigma = \varepsilon\big(\alpha, X(\beta, \varepsilon)\big)$, where X is a non-horizontal component and α, 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 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.

LEMMA 6.5. *If σ and σ' are two synopsis trees in normal form that strongly cover each other, then σ and σ' are isomorphic.*

PROOF. Let σ and σ' be two synopsis trees in normal form and let λ be a bijection between the non-trivial nodes of σ and the non-trivial nodes of σ' that witnesses the fact that $\sigma \hookrightarrow\!\!\!\!\rightarrow \sigma'$. In the following, we often identify, for the sake of simplicity, the nodes of the synopsis trees σ and σ' with their labels. The proof is by structural induction and case analysis.

For the base case, suppose that $\sigma = \varepsilon$. Since σ contains only trivial nodes and λ is surjective over non-trivial nodes, σ' contains only trivial nodes too. Since σ' is in normal form, it follows that $\sigma' = \varepsilon$.

For the inductive step, we distinguish two cases depending on whether σ is of the form $X(\sigma_1, \varepsilon)$, where X being a non-trivial horizontal component, or of the form $\varepsilon(\sigma_1, X(\sigma_2, \varepsilon))$, with X being a non-horizontal component.

In the former case, i.e. $\sigma = X(\sigma_1, \varepsilon)$, we recall that the mapping λ is a bijection between non-trivial nodes that preserves the post-order. Because the root X of σ is non-trivial and is the maximal element with respect to the post-order relation, it must be mapped by λ to the root Y of τ. Moreover, since λ preserves the labels of non-trivial nodes, we have that $Y = \lambda(X) = X$. In particular, Y is a non-trivial horizontal component. Since σ' is in normal form, it follows that its right sub-tree is ε and hence λ maps the non-trivial nodes of the left sub-tree σ_1 of σ to the non-trivial nodes of the left sub-tree σ'_1 of σ'. Finally, since both σ_1 and σ'_1 are synopsis trees in normal form, we obtain from the inductive hypothesis that $\sigma_1 = \sigma'_1$ and hence $\sigma = X(\sigma_1, \varepsilon) = Y(\sigma'_1, \varepsilon) = \sigma'$.

We consider the second case, i.e. $\sigma = \varepsilon\big(\sigma_1, X(\sigma_2, \varepsilon)\big)$, where X is a non-horizontal component. For the sake of contradiction, suppose that the root Y of σ' is a non-trivial component. Since Y is the last non-trivial node in the post-order traversal of σ' and λ preserves the post-order of non-trivial nodes, the pre-image $\lambda^{-1}(Y)$ in σ would consist of the last non-trivial node in the post-order traversal of σ', whence $\lambda^{-1}(Y) = X$. However, since X is a non-horizontal component, this would be against the hypothesis that σ' is in normal form (recall that the root of any synopsis tree in normal form is always a horizontal component). Knowing that Y is a trivial node and σ' is in normal form, we obtain $Y = \varepsilon$ and hence σ' is of the form $\varepsilon\big(\sigma'_1, Z(\sigma'_2, \varepsilon)\big)$, for some non-horizontal component Z and two synopsis trees σ'_1 and σ'_2 in normal form. Towards a conclusion, observe

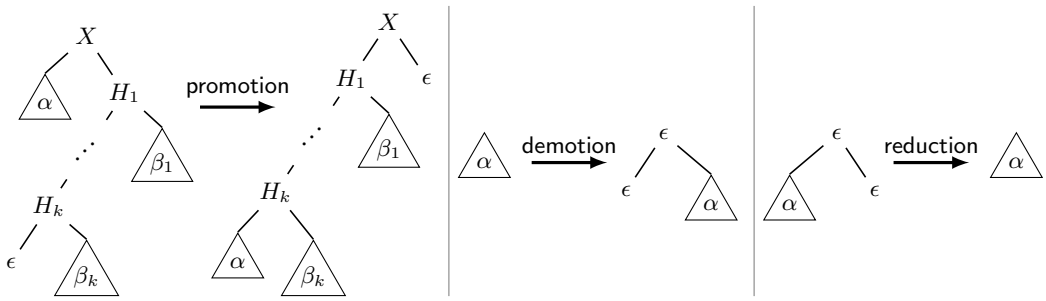


Fig. 11. Editing operations on synopsis trees.

that, in the post-order traversal of σ, the non-trivial nodes of σ_1 precede the non-trivial nodes of σ_2, and the latter are followed by the non-horizontal node X. Since λ is a bijection that 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 σ_1 and σ_1' (resp., σ_2 and σ_2') strongly cover each other. Using the inductive hypothesis, we finally conclude that $\sigma = \varepsilon\big(\sigma_1, X(\sigma_2, \varepsilon)\big) = \varepsilon\big(\sigma_1', Z(\sigma_2', \varepsilon)\big) = \sigma'$. $\square$

Thanks to Lemma 6.5 we can define *the normal form* σ^* of a synopsis tree σ as the unique synopsis tree that is *in normal form* and that *strongly covers* σ, provided that this tree exists.

Our next goal is to prove that the normal form σ^* of σ indeed exists, and that it 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 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 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 $\sigma \rightarrow^*_{\mathsf{op}} \sigma'$ whenever σ' can be obtained from σ by applying a finite sequence of promotion, demotion, and reduction operations. In order to give further intuition about these operations, we remark an analogy between the operations of promotion, depicted in Figure 11, and deletion, depicted in Figure 4 (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 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 that we just described 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. 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 6.6. *For every synopsis tree σ, there is σ^* in normal form such that $\sigma \rightarrow^*_{\mathsf{op}} \sigma^*$. Moreover, the number of operations needed to transform σ into σ^* is bounded by $2|\sigma|$.*

PROOF. The proof goes again by a structural induction on the synopsis tree σ. Intuitively, we first normalize the left and right sub-trees of σ separately using induction. Then we complete the normalization process by applying a suitable series of operations on the basis of the component at the root of σ: if this component is non-horizontal, then we apply a promotion followed by a demotion; if it is horizontal and non-trivial, then

we only apply a promotion operation; if it is trivial, then we apply a promotion followed by a reduction operation.

We can assume without loss of generality that all leaves in the synopsis tree σ are trivial, hence labeled by ε (indeed, we can append ε-labeled nodes to every leaf of σ without changing its equivalence class). This assumption reduces the base case to the situation where the synopsis tree σ consists of a single ε-labeled node. In this case, the synopsis tree is already in normal form and hence the lemma is trivially satisfied by letting $\sigma^* = \varepsilon = \sigma$.

For the inductive step, we assume that $\sigma = X(\alpha, \beta)$. First, we transform the sub-trees α, β of σ into their corresponding normal forms α^*, β^* (this can be done since, by inductive hypothesis, $\alpha \to^*_{\mathsf{op}} \alpha^*$ and $\beta \to^*_{\mathsf{op}} \beta^*$). We consider the intermediate synopsis tree that we just obtained:

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

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

$$\sigma' \;=\; X\big(\alpha^*,\; H_1\big(\ldots H_k(\varepsilon, \beta_k), \ldots, \beta_1\big)\big),$$

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

$$\sigma'' \;=\; X\big(H_1\big(\ldots H_k(\alpha^*, \beta_k), \ldots, \beta_1\big), \varepsilon\big).$$

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

We now distinguish a few cases depending on whether the component X is horizontal, trivial, or non-horizontal:

(1) If the component X is horizontal and non-trivial, then σ'' is already in normal form and we can simply let $\sigma'' = \sigma^*$.
(2) If the component X is trivial, then we "lift" the left sub-tree of σ'' via a reduction operation. This results in a synopsis tree $\sigma^* = H_1(\ldots H_k(\alpha^*, \beta_k), \ldots, \beta_1)$ in normal form.
(3) If the component X is non-horizontal, then we apply a demotion operation to σ'' so as to obtain the synopsis tree

$$\sigma^* \;=\; \varepsilon\big(\varepsilon, X\big(H_1(\ldots H_k(\alpha^*, \beta_k), \ldots, \beta_1\big), \varepsilon\big).$$

We observe that both sub-trees ε and $H_1\big(\ldots H_k(\alpha^*, \beta_k), \ldots, \beta_1\big)$ are in normal form. Moreover, since X is non-horizontal, we know that σ^* is also in normal form.

It remains to prove the upper bound on the minimum number $\#_{\mathsf{op}}(\sigma)$ of operations on synopsis trees that are required to transform σ into σ^*. From previous constructions, we can easily verify that

$$\#_{\mathsf{op}}(\sigma) \;\le\; \#_{\mathsf{op}}(\alpha) + \#_{\mathsf{op}}(\beta) + 2$$

and hence $\#_{\mathsf{op}}(\sigma) \le 2|\sigma|$. $\square$

For the last ingredient of the proof, we show that if $\sigma \to^*_{\mathsf{op}} \sigma'$, then the two languages $[\![\sigma]\!]$ and $[\![\sigma']\!]$ are repairable one into the other by means of a sequence of editing operations of uniformly bounded length. In other words, a single promotion, demotion, or reduction operation to a synopsis tree σ corresponds to a small amount of edits that

are applicable to any generic tree in the language $[\![\sigma]\!]$. The proof of this result is via a simple analysis of the transformations on unranked trees that are induced by the operations of promotion, demotion, and reduction.

LEMMA 6.7. *If σ' is a synopsis tree obtained from another synopsis tree σ via a single promotion, demotion, or reduction operation, then* $\mathsf{cost}\big([\![\sigma]\!],[\![\sigma']\!]\big) \leq 2$ *and* $\mathsf{cost}\big([\![\sigma']\!],[\![\sigma]\!]\big) \leq 2$. *In particular, it follows by induction that* $\mathsf{cost}\big([\![\sigma]\!],[\![\sigma^*]\!]\big) \leq 4|\sigma|$ *and* $\mathsf{cost}\big([\![\sigma^*]\!],[\![\sigma]\!]\big) \leq 4|\sigma|$.

PROOF. It is sufficient to prove that $\mathsf{cost}\big([\![\sigma]\!],[\![\sigma']\!]\big) \leq 2$, as the symmetric bound $\mathsf{cost}\big([\![\sigma']\!],[\![\sigma]\!]\big) \leq 2$ follows from the fact that standard editing operations on trees can be reverted. In the sequel, we assume that all synopsis trees are related to a stepwise automaton $\mathcal{S}$. We apply a case distinction based on the type of operation that transforms σ into σ':

(1) Consider a promotion operation, which takes a synopsis tree of the form $\sigma = X\big(\alpha,\ H_1(\ldots H_k(\varepsilon,\beta_k),\ldots,\beta_1)\big)$ and transforms it into the synopsis tree $\sigma' = X\big(H_1(\ldots H_k(\alpha,\beta_k),\ldots,\beta_1),\varepsilon\big)$. Consider also a generic tree $t \in [\![\sigma]\!]$. This can be written as

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

for some $C \in \mathscr{L}(\mathcal{S} \mid X)$, $s \in [\![\alpha]\!]$, $C_i \in \mathscr{L}(\mathcal{S} \mid H_i)$, $s_i \in [\![\beta_i]\!]$, and $a \in \Sigma$. For the sake of brevity, define the horizontal context

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

in such a way that we can write $t = C \circ (s@(\bar{C} \circ a))$. After deleting the a-labeled node from t, we obtain the tree $t' = C \circ (\bar{C} \circ s)$, and after inserting a b-labeled node, we obtain the tree

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

which clearly belongs to $[\![\sigma']\!]$.

(2) Consider now a demotion operation, which takes a synopsis tree σ and transforms it into the synopsis tree $\sigma' = \varepsilon(\varepsilon,\sigma)$. Let $t \in [\![\sigma]\!]$ and let x be the leftmost leaf in t (note that this corresponds to the root of the unranked tree $\mathsf{ext}^{-1}(t)$). 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 the tree $t' = b \mathbin{@} (C \circ a)$, which clearly belongs to $[\![\sigma']\!]$.

(3) We finally consider a reduction operation, which transforms a synopsis tree $\sigma = \varepsilon(\alpha,\varepsilon)$ into the synopsis tree $\sigma' = \alpha$. We can write any generic tree $t \in [\![\sigma]\!]$ as $t = s \mathbin{@} a$, where $s \in [\![\alpha]\!]$ and $a \in \Sigma$. In this case it suffices to perform one deletion in order to obtain the tree $t' \;=\; s$, which clearly belongs to $[\![\sigma']\!]$.

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

To conclude the proof, we remark that, thanks to Lemma 6.6, the normal form σ^* of any synopsis tree σ can be obtained by applying a sequence of promotions, demotions, and reductions of length at most $2|\sigma|$. A simple induction finally implies that $\mathsf{cost}\big([\![\sigma]\!],[\![\sigma^*]\!]\big) \leq 4|\sigma|$ and $\mathsf{cost}\big([\![\sigma^*]\!],[\![\sigma]\!]\big) \leq 4|\sigma|$. □

We have all the ingredients now to prove Lemma 6.1.

PROOF OF LEMMA 6.1. Let θ be the intermediate synopsis tree such that $\sigma \rightsquigarrow \theta \leftrightharpoons \tau$, whose existence is shown in Lemma 6.2. Recall that Lemma 6.3 implies $[\![\theta]\!]_{\mathcal{S}} \subseteq [\![\tau]\!]_{\mathcal{T}}$. As $|\theta| = |\tau|$, it is sufficient to show that $\mathsf{cost}\big([\![\sigma]\!]_{\mathcal{S}}, [\![\theta]\!]_{\mathcal{S}}\big) \leq 4|\sigma| + 4|\theta|$. This last claim can be proved using Lemma 6.7, which implies $\mathsf{cost}\big([\![\sigma]\!]_{\mathcal{S}}, [\![\sigma^*]\!]_{\mathcal{S}}\big) \leq 4|\sigma|$ and $\mathsf{cost}\big([\![\theta^*]\!]_{\mathcal{S}}, [\![\theta]\!]_{\mathcal{S}}\big) \leq 4|\theta|$, and the fact that $\sigma \rightsquigarrow \theta$, which implies $\sigma^* = \theta^*$. □

We conclude the subsection by proving Proposition 5.8, which essentially gives an upper bound for the cost of an optimal repair strategy from $\mathscr{L}(\mathcal{S})$ to $\mathscr{L}(\mathcal{T})$. In order to prove this proposition, we need to analyze the minimum size of a basic synopsis tree of $\mathcal{T}$ that covers a given primitive synopsis tree of $\mathcal{S}$:

LEMMA 6.8. *Given a primitive synopsis tree σ of $\mathcal{S}$, if σ is covered by some basic synopsis tree of $\mathcal{T}$, then it is covered by one such tree $\tau \in \mathsf{BST}(\mathcal{T})$ that has size at most $(4|\sigma| + 1) \cdot |\mathsf{SCC}(\mathcal{T})|$, where $|\sigma|$ is the number of nodes of σ and $|\mathsf{SCC}(\mathcal{T})|$ is the number of components of $\mathcal{T}$.*

PROOF. Let $\sigma \in \mathsf{PST}(\mathcal{S})$ and $\tau \in \mathsf{BST}(\mathcal{T})$ such that $\sigma \hookrightarrow \tau$ and let λ be the injective function from non-trivial nodes of σ to non-trivial nodes of τ that witnesses $\sigma \hookrightarrow \tau$. We begin by identifying those nodes of τ that belong to the range of λ. Formally, we say that a node y of τ is *used* if $y \in \lambda(x)$ for some non-trivial node x of σ. Below, we show how to restrict τ to a subset of its nodes having size at most $(4|\sigma| + 1) \cdot |\mathsf{SCC}(\mathcal{T})|$ and such that the induced sub-graph is a basic synopsis tree of $\mathcal{T}$ that also covers σ.

We first define the set V that only contains the following nodes:

(1) the root of τ,
(2) the used nodes of τ, and
(3) the nodes of τ whose both sub-trees contain some used nodes of τ.

We claim that the sub-graph of τ induced by V is a tree with at most two children on each node (note that some internal nodes in the induced sub-graph V may contain only one child). Consider two nodes y_1, y_2 in V. Let y be the least common ancestor of y_1 and y_2 in τ. As both sub-trees of y in τ contain used nodes – indeed, they contain y_1 and y_2, respectively – the node y also belongs to V. This shows that V is closed under least-common-ancestor and hence τ restricted to V is a tree with out-degree at most 2.

We can also verify that the size of V is at most $2|\sigma| + 1$. Indeed, the number of used nodes in τ is at most $|\sigma|$, and so is the number of nodes whose both sub-trees contain used nodes.

Next, we extend the set V minimally in such a way that the induced sub-graph of τ is a basic synopsis tree (in particular, it is a binary tree). Formally, we let W be the set of the following nodes:

(4) the nodes in V,
(5) the nodes of τ with one sub-tree containing some used nodes and with label *different* from that of its parent, and
(6) the immediate successors in τ of all previous nodes.

Clearly, the sub-graph of τ induced by W, denoted $\tau|_W$, is a full binary tree, namely, all internal nodes have exactly two children. Essentially, this holds because we included in W the immediate successors of a set of nodes.

It is also easy to see that $\tau|_W$ is a basic synopsis tree. Indeed, consider a node $y \in W$ and its two successors y_1 and y_2 in the induced sub-graph $\tau|_W$. If both y_1 and y_2 are immediate successors of y in τ, then, since τ respects the transition function δ of $\mathcal{T}$, there exist some states $q \in \tau(y)$, $q_1 \in \tau(y_1)$, and $q_2 \in \tau(y_2)$ such that $q \in \delta(q_1, q_2)$. Otherwise, if y_1 is not an immediate successor of y, then all the nodes of τ between

y and the parent of y_1 must have the same label $\tau(y)$ (otherwise, one of these nodes would belong to W, thus contradicting the fact that y_1 is a successor of y in $\tau|_W$). From this, using similar arguments as in the previous case, we conclude that $q \in \delta(q_1, q_2)$ for some states $q \in \tau(y)$, $q_1 \in \tau(y_1)$, and $q_2 \in \tau(y_2)$. The case where y_2 is not an immediate successor of y in τ is just symmetric. Overall, this proves that the induced sub-graph $\tau|_W$ respects the transition function of $\mathcal{T}$ and hence it is a basic synopsis tree.

To prove that $\tau|_W$ covers σ, it suffices to recall that W contains all used nodes of τ, i.e. $\lambda(x) \in W$ for all non-trivial nodes of σ, and hence the same function λ that witnessed $\sigma \hookrightarrow \tau$ can be used to witness $\sigma \hookrightarrow \tau|_W$.

It remains to prove that $|W| \leq (4|\sigma|+1) \cdot |\mathsf{SCC}(\mathcal{T})|$. It is easy to see that every node in $W \setminus V$ either has a descendant which is used or it is the successor of a node with a used descendant. This means that any subset of $W \setminus V$ that contains only nodes that are pairwise incomparable with respect to the ancestor relation has size at most twice the number of used nodes, hence at most $2|\sigma|$. Moreover, if we consider sets of nodes from $W \setminus V$ totally ordered with respect to the ancestor relation, then we observe that such a set has size at most $|\mathsf{SCC}(\mathcal{T})|$: indeed, every two nodes in this set that are consecutive in τ must be labeled with different components. Putting all together, and recalling that $|V| \leq 2|\sigma|+1$, we conclude that $|W| \leq 2|\sigma|+1+2|\sigma| \cdot |\mathsf{SCC}(\mathcal{T})| \leq (4|\sigma|+1) \cdot |\mathsf{SCC}(\mathcal{T})|$. □

We are now ready to derive an upper bound for the cost of an optimal repair strategy from $\mathscr{L}(\mathcal{S})$ to $\mathscr{L}(\mathcal{T})$ under the assumption that all primitive synopsis trees of $\mathcal{S}$ are covered by basic synopsis trees of $\mathcal{T}$.

PROOF OF PROPOSITION 5.8. Let f be a function that maps every primitive synopsis tree σ of $\mathcal{S}$ to a basic synopsis tree τ of $\mathcal{T}$ that covers σ. Following the previous results, the strategy for repairing $\mathscr{L}(\mathcal{S})$ into $\mathscr{L}(\mathcal{T})$ can be obtained from a series of transformations between languages having the following costs:

$$\mathsf{cost}\big(\mathscr{L}(\mathcal{S}),\ \cup_{\sigma \in \mathsf{PST}(\mathcal{S})} [\![\sigma]\!]_{\mathcal{S}}\big) \ = \ 0 \qquad \text{(by Lemma 5.2)}$$

$$\mathsf{cost}\big([\![\sigma]\!]_{\mathcal{S}},\ [\![f(\sigma)]\!]_{\mathcal{T}}\big) \ \leq \ 4|\sigma| + 4|f(\sigma)| \qquad \text{(by Lemma 6.1)}$$

$$\mathsf{cost}\big([\![f(\sigma)]\!]_{\mathcal{T}},\ \mathscr{L}(\mathcal{T})\big) \ \leq \ \big(4|f(\sigma)|+1\big) \cdot 2^{|Q'|} \qquad \text{(by Lemma 5.5)}$$

where Q' is the set of states of $\mathcal{T}$. In particular, we get:

$$\mathsf{cost}\big(\mathscr{L}(\mathcal{S}), \mathscr{L}(\mathcal{T})\big) \ \leq \ \max_{\sigma \in \mathsf{PST}(\mathcal{S})} \big\{4|\sigma| + 4|f(\sigma)| + \big(4|f(\sigma)|+1\big) \cdot 2^{|Q'|}\big\}.$$

As we previously pointed out, any primitive synopsis tree σ of $\mathcal{S}$ is of bounded size, precisely, $|\sigma| \leq 2^{|Q|}$, where Q is the set of states of $\mathcal{S}$. As concerns the minimum size of a basic synopsis tree $f(\sigma)$ that covers σ, by applying Lemma **??** for every $\sigma \in \mathsf{PST}(\mathcal{S})$, there is $\tau \in \mathsf{BST}(\mathcal{T})$ such that $\sigma \hookrightarrow \tau$ and

$$|\tau| \ \leq \ (4|\sigma|+1) \cdot |\mathsf{SCC}(\mathcal{T})|$$

In particular, we can assume, without loss of generality, that $|f(\sigma)| \leq \big(2^{|Q|+2+1}\big) \cdot |\mathsf{SCC}(\mathcal{T})|$. Putting everything together, we obtain:

$$\mathsf{cost}\big(\mathscr{L}(\mathcal{S}), \mathscr{L}(\mathcal{T})\big) \ = \ \mathcal{O}\big(|\mathsf{SCC}(\mathcal{T})| \cdot 2^{|Q|+|Q'|}\big) \ .$$

□

6.2. From repair to covering

In this subsection we prove the "only if" direction of Theorem 5.7. We fix for the rest of the section two stepwise automata $\mathcal{S} = (\Sigma, Q, \delta, \delta_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', \delta'_0, F')$ rec-

ognizing the source and the target languages, respectively. We assume that $\mathscr{L}(\mathcal{S})$ is repairable into $\mathscr{L}(\mathcal{T})$ with uniformly bounded cost and we prove that every primitive synopsis tree of $\mathcal{S}$ is covered by some basic synopsis tree of $\mathcal{T}$.

The general idea is to associate with each primitive synopsis tree σ of $\mathcal{S}$ a suitable tree $t_\sigma \in \mathscr{L}(\mathcal{S})$, called *witness tree* of σ, such that from any optimal repair of t_σ into $\mathscr{L}(\mathcal{T})$ one can extract a basic synopsis tree τ of $\mathcal{T}$ that covers σ. Intuitively, the witness tree t_σ is obtained from the primitive synopsis tree σ by replacing every non-trivial node x with a sufficiently large number of repetitions of a special context in $\mathscr{L}(\mathcal{S} \mid \sigma(x))$, called *fingerprint context*. The number of repetitions of each fingerprint context will depend on the worst-case repair cost $K = \mathsf{dist}\big(\mathscr{L}(\mathcal{S}), \mathscr{L}(\mathcal{T})\big)$. Using the definition of witness tree t_σ and the assumption that t_σ can be repaired into some tree $t'_\sigma \in \mathscr{L}(\mathcal{T})$ with at most K edits, one can then argue that t'_σ contains at least one copy of the fingerprint context associated with each non-trivial node x of σ and, furthermore, the arrangements of these fingerprint contexts inside t_σ and inside t'_σ are the same, 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 t'_σ: this run, together with the structure of the fingerprints inside t'_σ, induces a basic synopsis tree τ of $\mathcal{T}$ and a coverability relation from σ to τ. Below, we illustrate the various definitions and arguments in more detail. We divide up the proof into constructing the witness tree t_σ and building the cover from its repair.

Constructing the witness tree. We begin by giving the following lemma, which defines the so-called fingerprint context of a component of $\mathcal{S}$. Basically, the lemma shows that given a component X of $\mathcal{S}$, one can find a context C_X that can be "pumped" inside the language $\mathscr{L}(\mathcal{S} \mid X)$ (i.e., $C_X \circ \ldots \circ C_X \in \mathscr{L}(\mathcal{S} \mid X)$) and that characterizes the containment of $\mathscr{L}(\mathcal{S} \mid X)$ in $\mathscr{L}(\mathcal{T} \mid Y)$ for every component Y of $\mathcal{T}$ (i.e., $\mathscr{L}(\mathcal{S} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$ iff $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$). 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 6.9. *For all $X \in \mathsf{SCC}(\mathcal{S})$, there is a cyclic context $C_X \in \mathscr{L}(\mathcal{S} \mid X)$ such that, for all $Y \in \mathsf{SCC}(\mathcal{T})$,*

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

PROOF. Let X be a component of $\mathcal{S}$ 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 of $\mathcal{T}$, that is, we prove that for every $0 \leq i \leq m$, there is a cyclic context $C_i \in \mathscr{L}(\mathcal{S} \mid X)$ such that:

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

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

The base case $i = 0$ holds vacuously for $C_i = \bullet$, so we focus on the inductive step. Suppose that we defined a context C_i that satisfies ($\star$) for some index $0 \leq i < m$. To construct a context C_{i+1} that satisfies ($\star$), we need to distinguish two cases, depending on whether $\mathscr{L}(\mathcal{S} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_{i+1})$ or not.

If $\mathscr{L}(\mathcal{S} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_{i+1})$, then we define $C_{i+1} = C_i$ and we observe that ($\star$) holds trivially for $i + 1$.

Otherwise, if $\mathscr{L}(\mathcal{S} \mid X) \nsubseteq \mathscr{L}(\mathcal{T} \mid Y_{i+1})$, we let C be a context in $\mathscr{L}(\mathcal{S} \mid X) \setminus \mathscr{L}(\mathcal{T} \mid Y_{i+1})$. Since C_i is cyclic and $C \in \mathscr{L}(\mathcal{S} \mid X)$, there exist some states $p, q, r \in X$ such that $r \in \delta(r, C_i)$ and $q \in \delta(p, C)$. Let C' and C'' be some other contexts in $\mathscr{L}(\mathcal{S} \mid X)$ such that $r \in \delta(q, C')$ and $p \in \delta(r, C'')$ (such contexts exist since p, q, r are states within the same strongly connected component of $\mathcal{S}$). We then define:

$$C_{i+1} \;=\; C_i \circ C' \circ C \circ C''.$$

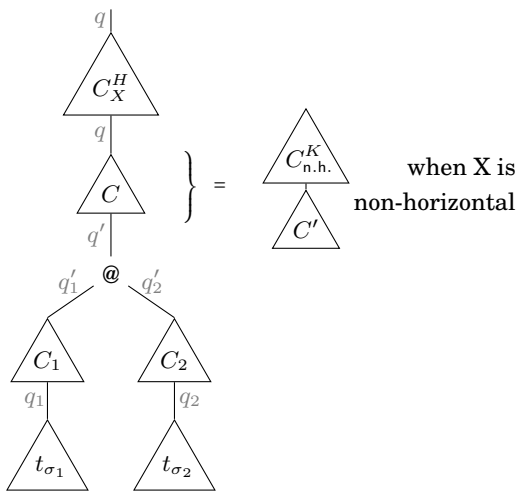


Fig. 12. Construction of the witness tree.

We claim that C_{i+1} is a cyclic context in $\mathscr{L}(\mathcal{S} \mid X)$. Indeed, we have the following runs in the source automaton $\mathcal{S}$:

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

It is also easy to see that $C_{i+1} \notin \mathscr{L}(\mathcal{T} \mid Y_{i+1})$. Indeed, if $C_{i+1} \in \mathscr{L}(\mathcal{T} \mid Y_{i+1})$, then there would exist some states $p', q' \in Y_{i+1}$ such that $q' \in \delta'(p', C)$, which would contradict the fact that $C \notin \mathscr{L}(\mathcal{T} \mid Y_{i+1})$.

We have just constructed a cyclic context $C_{i+1} \in \mathscr{L}(\mathcal{S} \mid X)$ such that $\mathscr{L}(\mathcal{S} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_{i+1})$ iff $C_{i+1} \in \mathscr{L}(\mathcal{T} \mid Y_{i+1})$. To conclude the proof, we recall from the inductive hypothesis that, for all $1 \leq j \leq i$, $\mathscr{L}(\mathcal{S} \mid X) \nsubseteq \mathscr{L}(\mathcal{T} \mid Y_j)$ implies $C_i \notin \mathscr{L}(\mathcal{T} \mid Y_j)$, and hence, since C_{i+1} contains an occurrence of C_i, $C_{i+1} \notin \mathscr{L}(\mathcal{T} \mid Y_j)$. Symmetrically, $\mathscr{L}(\mathcal{S} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_j)$ implies $C_{i+1} = C_i \in \mathscr{L}(\mathcal{T} \mid Y_j)$. All together, this shows that, for all $1 \leq j \leq i+1$, $\mathscr{L}(\mathcal{S} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y_j)$ iff $C_{i+1} \in \mathscr{L}(\mathcal{T} \mid Y_j)$. $\square$

For the rest of this subsection, we fix for each component X of $\mathcal{S}$ a context C_X that satisfies Lemma 6.9 and we call it *fingerprint context*. We also fix an arbitrary primitive synopsis tree σ of $\mathcal{S}$.

Below, we construct the so-called witness tree t_σ by exploiting a structural induction on the primitive synopsis tree σ. In doing so, we will guarantee that there exists a run of $\mathcal{S}$ on t_σ that assigns to the root of t_σ some state that belongs to the same component that labels the root of σ, namely, $\delta(t_\sigma) \cap \sigma(\varepsilon) \neq \varnothing$. We omit the construction for the base case, where σ is a singleton, since it can be easily derived from what follows. We assume that X is the component at the root of σ and that σ_1 and σ_2 are the (non-empty) left and right sub-trees of σ. Suppose that t_{σ_1} and t_{σ_2} are the recursively defined witness trees for σ_1 and σ_2. Moreover, choose arbitrarily some q_1 (resp., q_2) in the non-empty set $\delta(t_{\sigma_1}) \cap \sigma_1(\varepsilon)$ (resp., $\delta(t_{\sigma_2}) \cap \sigma_2(\varepsilon)$). The witness tree t_σ for σ is obtained from the following series of transformations (we suggest the reader to refer to Figure 12 for a graphical representation):

(1) The first transformation merges the trees t_{σ_1} and t_{σ_2} into a single tree that induces a run of $\mathcal{S}$ ending in the component X. We know from the definition of primitive synopsis tree that σ respects the transition function of $\mathcal{S}$. In particular, this means that there exist some states $q' \in X$, $q_1' \in \sigma_1(\varepsilon)$, and $q_2' \in \sigma_2(\varepsilon)$ 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 σ_1 (resp., σ_2), there exist some contexts $C_1 \in \mathscr{L}(\mathcal{S} \mid \sigma_1(\varepsilon))$ and $C_2 \in \mathscr{L}(\mathcal{S} \mid \sigma_2(\varepsilon))$ 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

$$(C_1 \circ t_{\sigma_1}) \mathbin{@} (C_2 \circ t_{\sigma_2})$$

that induces a run of $\mathcal{S}$ ending in state q', within the component X.

(2) The second transformation extends the tree obtained in the previous step in such a way that one can later append 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' via a suitable context $C \in \mathscr{L}(\mathcal{S} \mid X)$ 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:

$$C \circ \big((C_1 \circ t_{\sigma_1}) \mathbin{@} (C_2 \circ t_{\sigma_2})\big).$$

In order to avoid that an editing of the witness tree t_σ could modify the ancestorship of C_X with the nodes of the two sub-trees t_{σ_1} and t_{σ_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 $C_{\mathsf{n.h.}}^K \circ C'$, where $K = \mathsf{dist}\big(\mathscr{L}(\mathcal{S}), \mathscr{L}(\mathcal{T})\big)$, $C', C_{\mathsf{n.h.}} \in \mathscr{L}(\mathcal{S} \mid X)$, $C_{\mathsf{n.h.}}$ is some cyclic *non-horizontal* context, and $C_{\mathsf{n.h.}}^K$ is the K-fold repetition of $C_{\mathsf{n.h.}}$ (recall that the ancestorship of non-horizontal contexts is preserved by the editing operations).

(3) The last transformation adds a sufficiently large repetition of the fingerprint context C_X. For this, we define $H = m \cdot (2K+1)$, where $K = \mathsf{dist}\big(\mathscr{L}(\mathcal{S}), \mathscr{L}(\mathcal{T})\big)$ and m is the number of components of $\mathcal{T}$. We then attach to the tree so far constructed the H-fold repetition C_X^H of the fingerprint context C_X, finally obtaining the desired witness tree:

$$t_\sigma \;=\; C_X^H \circ C \circ \big((C_1 \circ t_{\sigma_1}) \mathbin{@} (C_2 \circ t_{\sigma_2})\big).$$

We observe that, thanks to the above constructions, the automaton $\mathcal{S}$ admits a run on t_σ that ends in state $q \in X$. This shows that the invariant $\delta(t_\sigma) \cap X \neq \varnothing$ is satisfied. We also remark that it may happen that $\delta(t_\sigma) \cap F = \varnothing$ and hence $t_\sigma \notin \mathscr{L}(\mathcal{S})$. Technically speaking, this could violate the claim that one can repair t_σ into $\mathscr{L}(\mathcal{T})$ with at most K edits. However, from the assumption that $\mathcal{S}$ is trimmed it follows that there is a context C_F such that $\delta(q, C_F) \cap F \neq \varnothing$ and hence one can always prolong t_σ to obtain a tree inside the language $\mathscr{L}(\mathcal{S})$. From now on, we assume for the sake of simplicity that $t_\sigma \in \mathscr{L}(\mathcal{S})$.

Building the covering from a repaired witness tree. We now turn towards extracting a covering of σ from a repair of t_σ. We fix, once and for all, the tree t_σ' in the target language $\mathscr{L}(\mathcal{T})$ that is obtained by repairing t_σ with at most K edits.

We recall that the witness tree t_σ contains $H = m \cdot (2K+1)$ copies of the fingerprint context C_X, for each node x in σ, where $X = \sigma(x)$. As a consequence, the repaired tree t_σ' must contain an m-fold repetition C_X^m of each fingerprint context C_X. In the following, we will look at the occurrences of these fingerprint contexts inside the repaired tree t_σ' and compare their post-order and ancestor relationships with those for the analogous occurrences in t_σ. We need some preliminary 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 $\{x\} \cdot \mathsf{nodes}(C) \cap \{x'\} \cdot \mathsf{nodes}(C') = \varnothing$ (resp., if $x \preccurlyeq_t^{\mathsf{post}} x'$, if $x \preccurlyeq_t^{\mathsf{anc}} x'$).

The following lemma shows that the occurrences of the contexts C_X^m inside t_σ are in the same post-order relation as some corresponding occurrences inside t'_σ, and similarly for the ancestor relation when X is a non-horizontal component.

LEMMA 6.10. *One can find a mapping f from the non-trivial nodes x of the primitive synopsis tree σ to the nodes $f(x)$ of the repaired witness tree t'_σ such that:*

- *the context C_X^m, where $X = \sigma(x)$, occurs at node $f(x)$ in t'_σ, for all non-trivial nodes x of σ,*
- *all occurrences $(C_X^m, f(x))$, with $X = \sigma(x)$ and x non-trivial node of σ, are pairwise non-overlapping,*
- *$x \preccurlyeq_\sigma^{\mathsf{post}} y$ iff $f(x) \preccurlyeq_{t'_\sigma}^{\mathsf{post}} f(y)$, for all non-trivial nodes x, y of σ,*
- *$x \preccurlyeq_\sigma^{\mathsf{anc}} y$ iff $f(x) \preccurlyeq_{t'_\sigma}^{\mathsf{anc}} f(y)$, for all non-trivial nodes x, y of σ, with $\sigma(x)$ non-horizontal component.*

PROOF. We begin by establishing a property that concerns the occurrences of contexts in a tree that has been edited. Intuitively, the following claim implies that, if t' is a tree obtained from t by applying at most K edit operations and t contains an occurrence of the $2K+1$-fold repetition of a context C, then t'' contains at least one occurrence of the same context C.

CLAIM 1. *Let t be a 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 n non-overlapping occurrences of the same context C, then t'' contains at least $n-2$ non-overlapping occurrences of C.*

PROOF. We prove the claim for the deletion operation only, as the arguments for the case of an insertion are similar. Let x be the node that is deleted from t. As mentioned in Subsection 5.3, there is a unique way to represent the deletion of x using composition of trees and contexts, namely, we can write

$$t \;=\; C'' \circ \big(t'''@(C' \circ a)\big)$$

where a is the label of node x and C' is the *horizontal* context that represents the forest of sub-trees under x. The result of the deletion of node x gives the curried tree

$$t'' \;=\; C'' \circ (C' \circ t''').$$

Note 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 at least once in t''. In general, suppose that there are n 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 hence t'' contains at least $n-2$ occurrences of C. Finally, it is easy to see that the deletion operation preserves the property of occurrences of being non-overlapping. □

We continue now with the proof of Lemma 6.10. We consider a non-trivial node x of the primitive synopsis tree σ and let $X = \sigma(x)$ be the associated component. By definition of t_σ, we know that the context C_X^H occurs in t_σ. We also recall that $H = m \cdot (2K+1)$. In particular, t_σ contains $(2K+1)$ occurrences of the context C_X^m. As t'_σ is obtained from t_σ by applying at most K edit operations to it, we know from the above claim that the

context C_X^m occurs at least once in t'_σ. We denote by $f(x)$ some node of t'_σ where C_X^m occurs. We have just proved the first part of the lemma.

For second part, let x and y be two distinct non-trivial nodes of σ and let $X = \sigma(x)$ and $Y = \sigma(y)$. Furthermore, let $f(x)$ and $f(y)$ be the nodes in t'_σ where the contexts C_X^m and C_Y^m occurs. Note that the occurrences of these two contexts in t_σ are non-overlapping. Since deletion and insertion operations preserve the property of context occurrences of being non-overlapping, we have that $(C_X^m, f(x))$ and $(C_Y^m, f(y))$ are also non-overlapping occurrences in t'_σ.

Now, suppose that $x \preccurlyeq_\sigma^{\mathsf{post}} y$. Let x' and y' be the nodes in t_σ that carry the corresponding occurrences of the contexts C_X^H and C_Y^H, respectively. It is routine to check, by exploiting the recursive definition of t_σ, that $x' \preccurlyeq_{t_\sigma}^{\mathsf{post}} y'$. In addition, we know that edit operations preserve the post-order relationships between nodes. As previously discussed, at least one occurrence of C_X^m (resp., C_Y^m) inside C_X^H (resp., C_Y^H) is not affected by the edit operations that transform t_σ into t'_σ. This means that the corresponding occurrences $(C_X^m, f(x))$ and $(C_Y^m, f(y))$ in t'_σ are in the same post-order relationship as x and y, namely, $f(x) \preccurlyeq_{t'_\sigma}^{\mathsf{post}} f(y)$. The converse implication follows from the fact that $\preccurlyeq_\sigma^{\mathsf{post}}$ 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 $X = \sigma(x)$ is a non-horizontal component and that $x \preccurlyeq_\sigma^{\mathsf{anc}} y$. As before, let x' and y' be the nodes in t_σ that carry the occurrences of C_X^H and C_Y^H, respectively. Thanks to the construction of t_σ, we have $x' \preccurlyeq_\sigma^{\mathsf{anc}} y'$. Moreover, recall that during the construction of t_σ, we inserted K copies of the non-horizontal context $C_{\mathsf{n.h.}}$ immediately below C_X^H (and thus above C_Y^H). This implies that the path in t_σ that connects the node x' to its descendant y' visits at least K right edges. If we now look at the unranked tree $\mathsf{ext}^{-1}(t_\sigma)$ encoded by t_σ, we observe that there is a bijection between the right edges in t_σ and the vertical edges in $\mathsf{ext}^{-1}(t_\sigma)$, and this bijection preserves the ancestor order. This means that the two portions of the unranked tree $\mathsf{ext}^{-1}(t_\sigma)$ that are encoded by C_X^H and C_Y^H, namely, $\mathsf{ext}^{-1}(C_X^H)$ and $\mathsf{ext}^{-1}(C_Y^H)$, are separated by at least K vertical edges. Finally, each deletion or insertion operation performed on $\mathsf{ext}^{-1}(t_\sigma)$ can only bring two nodes closer by one level at a time. This means that after at most K edit operations, the resulting curried tree t'_σ contains the occurrence of C_X^m at node $f(x)$ is still above the occurrence of C_Y^m at node $f(y)$. We have just proved that $x \preccurlyeq_\sigma^{\mathsf{anc}} y$ implies $f(x) \preccurlyeq_{t'_\sigma}^{\mathsf{anc}} f(y)$. The converse implication follows by symmetric arguments. □

It now remains to show how to extract a basic synopsis tree τ that covers σ from the tree t'_σ. Recall that $t'_\sigma \in \mathscr{L}(\mathcal{T})$ and let ρ be an accepting run of $\mathcal{T}$ on t'_σ. Further let f be the mapping from non-trivial nodes of σ to nodes of t'_σ, as defined in Lemma 6.10. Consider a non-trivial node x of σ and let X be its label. We know that the context C_X^m occurs at node $f(x)$ in t'_σ. Let y be the node of the fingerprint context C_X that is labeled with the placeholder symbol $\bullet$. Clearly, C_X occurs in its m-fold iteration C_X^m at positions ε, y, $y \cdot y$, …, y^{m-1}. Analogous (non-overlapping) occurrences exist in t'_σ, namely, at positions $y_{x,0} = f(x)$, $y_{x,1} = f(x) \cdot y$, $y_{x,2} = f(x) \cdot y \cdot y$, …, $y_{x,m-1} = f(x) \cdot y^{m-1}$. For convenience, let $y_{x,m} = f(x) \cdot y^m$.

Next, we consider the states that occur at the $m+1$ nodes $y_{x,0}, y_{x,1}, \ldots, y_{x,m}$ of the run ρ on t'_σ. By the Pigeonhole Principle, we know that two among these states, say $\rho(y_{x,i})$ and $\rho(y_{x,j})$ for some $0 \leq i < j \leq m$, belong to the same component Y of $\mathcal{T}$. In fact, from the definition of strongly connected component, we can even assume that $j = i+1$, from which we immediately obtain $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$. It is now time to exploit the property of the fingerprint context C_X, shown in Lemma 6.9. In particular, from the fact that $C_X \in \mathscr{L}(\mathcal{T} \mid Y)$ we derive $\mathscr{L}(\mathcal{S} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$.

We have just showed that it is possible to find a mapping from any non-trivial node x of σ to a node y_x ($= y_{x,i}$) in t'_σ such that $\mathscr{L}(\mathcal{S} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$, where $X = \sigma(x)$ and Y is the component of the state $\rho(y_x)$. Thanks to Lemma 6.10, we can also claim that, for all non-trivial nodes x, x' in σ,

— $x \neq x'$ implies $y_x \neq y_{x'}$,
— $x \preccurlyeq^{\mathsf{post}}_{\sigma} x'$ iff $y_x \preccurlyeq^{\mathsf{post}}_{t'_\sigma} y_{x'}$,
— $x \preccurlyeq^{\mathsf{anc}}_{\sigma} x'$ iff $y_x \preccurlyeq^{\mathsf{anc}}_{t'_\sigma} y_{x'}$, provided that the component $\sigma(x)$ is non-horizontal.

Towards a conclusion, we define now the basic synopsis tree τ that covers σ. The domain of τ coincides with the domain of t'_σ, i.e., $\mathsf{nodes}(\tau) = \mathsf{nodes}(t'_\sigma)$. The labeling function of τ maps every trivial node x of τ to the trivial component ε, and every non-trivial node x to the component that contains the state $\rho(y_x)$ associated with the corresponding node y_x in t'_σ. It is easy to see that τ satisfies the properties of a basic synopsis tree (in particular, it respects the transitions of $\mathcal{T}$ because its labeling is essentially the lifting of a valid run ρ of $\mathcal{T}$). It remains to define the mapping λ that witnesses the coverability of σ by τ: for this we simply let $\lambda(x) = y_x$ for every non-trivial node x of σ. The fact that λ satisfies Definition 5.6 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 proves that every primitive synopsis tree of $\mathcal{S}$ is covered by a basic synopsis tree of $\mathcal{T}$.

7. COMPLEXITY ANALYSIS

In this section we investigate the complexity of deciding whether a regular tree language S is bounded repairable into a regular tree language T and we assume that S and T are represented by automata $\mathcal{S}$ and $\mathcal{T}$ respectively. One can propose a straightforward decision procedure following the characterization of bounded repairability with synopsis trees (Theorem 5.7): for every primitive synopsis tree of $\mathcal{S}$ it suffices to guess a covering basic synopsis tree of $\mathcal{T}$. We also recall from Remark 5.3 that the size of a primitive synopsis tree of $\mathcal{S}$ is bounded by a function exponential in the number of strongly connected components of $\mathcal{S}$. Hence, by Lemma 6.8, an analogous bound holds for the size of basis synopsis trees of $\mathcal{T}$. It is also easy to see that testing the covering of a primitive synopsis tree by a basic synopsis tree can be performed efficiently in the size of the synopsis trees. These observations show that deciding bounded repairability for languages represented by tree automata is in EXPSPACE.

We show, however, that a more efficient procedure exists: rather than inspecting individual elements of $\mathsf{PST}(\mathcal{S})$ and verifying that they are covered by elements of $\mathsf{BST}(\mathcal{T})$, it checks inclusion of the sets of *normalized* synopsis trees. More precisely, we first relabel synopsis trees in $\mathsf{BST}(\mathcal{T})$ with compatible connected components of $\mathcal{S}$, and as a result, we deal with synopsis trees labeled with elements of $\mathsf{SCC}(\mathcal{S})$. Next, we define a *serialization* of a synopsis tree, a string representation of the synopsis tree, and show that serialization of a synopsis tree is the same as the serialization of the normal form of the synopsis tree. Naturally, this reduces bounded repairability to the inclusion of serializations of $\mathsf{PST}(\mathcal{S})$ and $\mathsf{BST}(\mathcal{T})$ respectively. Testing this inclusion is not trivial because while both sets $\mathsf{PST}(\mathcal{S})$ and $\mathsf{BST}(\mathcal{T})$ can be captured with tree automata, the serializations versions are string languages that need not be regular. We show, however, that serializations of flattened versions of $\mathsf{PST}(\mathcal{S})$ and $\mathsf{BST}(\mathcal{T})$ can be captured with context-free grammars and, moreover, the context-free grammar for $\mathsf{PST}(\mathcal{S})$ is non-recursive. We then use existing results on testing inclusion of non-recursive context-free grammar in another (possibly) recursive context-free grammar and obtain a CONEXP upper bound. Finally, we show that the CONEXP upper bound

is tight even if we restrict the tree languages provided as input to those definable with deterministic non-recursive DTDs.

7.1. Upper bound

We show that the complexity of testing bounded repairability between two regular tree languages represented with tree automata is in CONEXP. For the reminder of this section we fix automata $\mathcal{S}$ and $\mathcal{T}$ that recognize the source tree language and the target tree language, respectively.

We begin by recalling the notion of embedding from Section 6.1. Given a synopsis tree θ of $\mathcal{S}$ and a synopsis tree τ of $\mathcal{T}$, we say that θ *is embedded into* τ, denoted $\theta \leftrightharpoons \tau$, if θ and τ have the same domain, i.e. $\mathsf{nodes}(\theta) = \mathsf{nodes}(\tau)$, and θ is covered by τ via the identity function, i.e. $\mathscr{L}(\mathcal{S} \mid \theta(x)) \subseteq \mathscr{L}(\mathcal{T} \mid \tau(x))$ for all nodes x. We define the set

$$\mathsf{Emb}_{\mathcal{S}}(\tau) \;=\; \big\{\theta \text{ synopsis tree of } \mathcal{S} \;:\; \theta \leftrightharpoons \tau\big\}$$

of all synopsis trees of $\mathcal{S}$ that are embedded into τ and we extend the notation to any set S of synopsis trees by letting $\mathsf{Emb}_{\mathcal{S}}(S) = \bigcup_{\tau \in S} \mathsf{Emb}_{\mathcal{S}}(\tau)$.

Next, we introduce a variant of the notion of serialization for synopsis trees and we show that this can be used as an alternative representation of the normal form that we introduced in Section 6. Such a serialization takes a synopsis tree θ and produces a well-nested word $\hat{\theta}$ over the alphabet $\mathsf{tags}(\mathcal{S})$ that consists of opening tags of the form $\langle X \rangle$ and closing tags of the form $\langle /X \rangle$, with $X \in \mathsf{SCC}(\mathcal{S})$. It is important to remark that the serialization $\hat{\theta}$ does not represent the specific tree θ, but rather the class of synopsis trees that have the same normal form as θ. Formally, we define the *serialization* $\hat{\theta}$ *of a synopsis tree* θ *of* $\mathcal{S}$ recursively as follows:

$$\hat{\theta} \;=\; \begin{cases} \hat{\theta}_1 \cdot \hat{\theta}_2 & \text{if } \theta = X(\theta_1, \theta_2) \text{ and } X \text{ is a trivial component,} \\ \hat{\theta}_1 \cdot \hat{\theta}_2 \cdot \langle X \rangle \cdot \langle /X \rangle & \text{if } \theta = X(\theta_1, \theta_2) \text{ and } X \text{ is a non-trivial horizontal component,} \\ \langle X \rangle \cdot \hat{\theta}_1 \cdot \hat{\theta}_2 \cdot \langle /X \rangle & \text{if } \theta = X(\theta_1, \theta_2) \text{ and } X \text{ is a non-horizontal component.} \end{cases}$$

Note that the trivial components disappear in the serialization $\hat{\theta}$ of a synopsis tree θ. As usual, we extend serializations to sets of synopsis trees by letting $\hat{U} = \{\hat{\theta} \;:\; \theta \in U\}$.

It is easy to see that serializations are unaffected by the editing operations on synopsis trees that are used to attain the normal form.

LEMMA 7.1. *Given two synopsis trees θ and ζ of $\mathcal{S}$ such that $\theta \rightarrow^*_{\mathsf{op}} \zeta$, we have $\hat{\theta} = \hat{\zeta}$. In particular, we have $\hat{\theta} = \widehat{\theta^*}$, where θ^* is the normal form of θ.*

PROOF. By induction it suffices to show that applying a single editing operation does not change the serialization. A quick inspection of the definitions of the editing operations of promotion, demotion, and reduction (see Figure 11) shows that the claim holds trivially. □

The above lemma, together with the results proven in Section 6.1, implies the following:

COROLLARY 7.2. *The language $\mathscr{L}(\mathcal{S})$ is bounded repairable into the language $\mathscr{L}(\mathcal{T})$ if and only if $\hat{U} \subseteq \hat{V}$, where $U = \mathsf{PST}(\mathcal{S})$ and $V = \mathsf{Emb}_{\mathcal{S}}(\mathsf{BST}(\mathcal{T}))$.*

PROOF. To prove the left-to-right implication, suppose that $\mathscr{L}(\mathcal{S})$ is bounded repairable into $\mathscr{L}(\mathcal{T})$ and consider a primitive synopsis tree $\sigma \in U$. We know from Theorem 5.7 that σ is covered by some basic synopsis tree τ of $\mathcal{T}$. Moreover, by Lemma 6.2, there exists a synopsis tree θ of $\mathcal{S}$ that "interpolates" σ and τ, namely, such that

σ is strongly covered by θ (denoted $\sigma \hookrightarrow\!\!\!\!\rightarrow \theta$) and θ embedded into τ (denoted $\theta \leftrightharpoons \tau$). In particular, we have that θ belongs to the set $V = \mathsf{Emb}_{\mathcal{S}}(\tau)$. Moreover, since $\sigma \hookrightarrow\!\!\!\!\rightarrow \theta$, we know from Lemma 6.5 that the normal forms of σ and θ coincide, and hence by Lemma 7.1 we have $\hat{\sigma} = \hat{\theta}$. We conclude that $\hat{\sigma}$ belongs to $\hat{V}$.

The proof of the converse direction is symmetric, namely, we assume that $\hat{U} \subseteq \hat{V}$, we consider a primitive synopsis tree σ of $\mathcal{S}$, and we prove that σ is covered by some basic synopsis tree of $\mathcal{T}$. Indeed, since $\sigma \in U$ and $\hat{U} \subseteq \hat{V}$, we have $\hat{\sigma} \in \hat{V}$. This means that there exist a synopsis tree θ of $\mathcal{S}$ and a basic synopsis tree τ of $\mathcal{T}$ such that $\theta \leftrightharpoons \tau$ and $\hat{\theta} = \hat{\sigma}$. In particular, Lemma 7.1 implies that θ and σ have the same normal form and hence they strongly cover each other, whence $\sigma \hookrightarrow\!\!\!\!\rightarrow \theta \leftrightharpoons \tau$. We conclude that σ is covered by τ and hence, by Theorem 5.7, $\mathscr{L}(\mathcal{S})$ is bounded repairable into $\mathscr{L}(\mathcal{T})$. □

We conclude the section by showing how to test effectively the inclusion from Corollary 7.2. For this, we introduce some context-free grammars that capture the languages $\hat{U}$ and $\hat{V}$, where $U = \mathsf{PST}(\mathcal{S})$ and $V = \mathsf{Emb}_{\mathcal{S}}(\mathsf{BST}(\mathcal{T}))$. We can define these grammars on the basis of the components of $\mathcal{S}$ and $\mathcal{T}$ and the transitions of $\mathcal{S}$ and $\mathcal{T}$ lifted to these components. More precisely, the grammar $G_{\mathcal{S}}$ that defines the language $\hat{U}$ uses non-terminals $X, X_1, X_2, \ldots$ that correspond to components of $\mathcal{S}$ and rules of the forms

$$
\begin{array}{lll}
X ::= \langle X \rangle \, \langle /X \rangle & & \\
X ::= X_1 \; X_2 & & \text{if } X \text{ is a trivial component,} \\
X ::= X_1 \; X_2 \; \langle X \rangle \, \langle /X \rangle & & \text{if } X \text{ is a non-trivial horizontal component,} \\
X ::= \langle X \rangle \; X_1 \; X_2 \; \langle /X \rangle & & \text{if } X \text{ is a non-horizontal component,}
\end{array}
$$

where $X_1 \neq X \neq X_2$, $q \in \delta(q_1, q_2)$ for some $q \in X$, $q_1 \in X_1$, $q_2 \in X$, and δ is the transition function of $\mathcal{S}$.

The grammar $G_{\mathcal{S},\mathcal{T}}$ that defines the language $\hat{V}$ uses the same non-terminals $X, X_1, X_2 \in \mathsf{SCC}(\mathcal{S})$ and the same rules as above, but instead of enforcing $X_1 \neq X \neq X_2$ and $q \in \delta(q_1, q_2)$ for some $q \in X$, $q_1 \in X_1$, $q_2 \in X$, it requires that there exists some components Y, Y_1, Y_2 of the target automaton $\mathcal{T}$ such that (i) $\mathscr{L}(\mathcal{S} \mid X) \subseteq \mathscr{L}(\mathcal{T} \mid Y)$, (ii) $\mathscr{L}(\mathcal{S} \mid X_1) \subseteq \mathscr{L}(\mathcal{T} \mid Y_1)$, (iii) $\mathscr{L}(\mathcal{S} \mid X_2) \subseteq \mathscr{L}(\mathcal{T} \mid Y_2)$, and (iv) $q \in \gamma(q_1, q_2)$ for some $q \in Y$, $q_1 \in Y_1$, and $q_2 \in Y_2$, where γ is the transition function of $\mathcal{T}$.

Although testing the inclusion of two generic context-free languages is known to be undecidable [Hopcroft and Ullman 1979], here we can exploit the fact that the grammar $G_{\mathcal{S}}$ is non-recursive to decide the inclusion $\mathscr{L}(G_{\mathcal{S}}) \subseteq \mathscr{L}(G_{\mathcal{S},\mathcal{T}})$. Indeed, a non-recursive grammar defines a finite language of words whose lengths are uniformly bounded by an exponent in the size of the grammar. Consequently, a non-deterministic Turing machine can guess a word w of length exponential in the size of $G_{\mathcal{S}}$ and decide the non-containment $\mathscr{L}(G_{\mathcal{S}}) \nsubseteq \mathscr{L}(G_{\mathcal{S},\mathcal{T}})$ by checking that $w \in \mathscr{L}(G_{\mathcal{S}})$ and $w \notin \mathscr{L}(G_{\mathcal{S},\mathcal{T}})$. It is also known that the membership problem of context-free languages can be solved in polynomial time [Younger 1967; Earley 1970]. The only subtlety here is that, although the grammars $G_{\mathcal{S}}$ and $G_{\mathcal{S},\mathcal{T}}$ are of polynomial size with respect to $\mathcal{S}$ and $\mathcal{T}$, this reduction takes exponential time in the size of $\mathcal{S}$ and $\mathcal{T}$: indeed, the the definition of $G_{\mathcal{S},\mathcal{T}}$ requires checking some containment relationships between the languages recognized by the components of $\mathcal{S}$ and $\mathcal{T}$. The latter problem, however, is EXP-complete [Seidl 1990] and hence dominated by the time that is required to guess the word w. We can thus claim the following complexity upper bound to the bounded repair problem.

THEOREM 7.3. *The bounded repair problem between languages represented by stepwise tree automata is in* CONEXP.

7.2. Lower bound

Here we show that the complexity bound established in Theorem 7.3 is tight. More precisely, we prove a matching coNEXP lower bound for the bounded repair problem, which, remarkably, holds even for tree languages represented by non-recursive deterministic DTDs.

We recall the results in [Champavère et al. 2009], in particular, Proposition 4 and Theorem 5, which show that any deterministic DTD can be transformed, in polynomial time, into an equivalent deterministic stepwise automaton. In particular, this means that the complexity lower bound for the bounded repair problem of languages represented by (non-recursive) deterministic DTDs can be immediately transferred to languages represented by deterministic stepwise automata.

We also recall the folklore PSPACE upper bound for the containment problem of non-deterministic DTDs: 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 the useless rules and then checking that, for all letters a in the alphabet of D, the regular language associated with a in D is contained in the regular language associated with a in D'. This upper bound result is tight due to the PSPACE-hardness of containment of regular expressions [Stockmeyer and Meyer 1973]. We finally observe that the complexity of the containment problem lowers to P as soon as deterministic DTDs are considered. Interestingly, the situation is completely different for the complexity of the bounded repair problem.

THEOREM 7.4. *The bounded repair problem between languages represented by non-recursive deterministic DTDs is* coNEXP*-hard.*

PROOF. The proof is by a reduction from the problem of tiling a square grid of exponential size [Boas 1997]. An instance of the latter problem is given by a tuple $I = (n, S, H, V, s_\perp, s_\top)$, where n is a natural number encoded in unary and representing the width 2^n of the square grid, S is a finite set of tiles, $H, V \subseteq S \times S$ are the set of vertical and horizontal constraints, and $s_\perp$ and $s_\top$ are the tiles that should mark the lower left and upper right corners. A tiling is a function f mapping pairs $(i, j) \in \{1, \ldots, 2^n\} \times \{1, \ldots, 2^n\}$ to tiles $f(i, j) \in S$. We say that a tiling f satisfies the constraints of I if the following conditions are satisfied:

(1) $f(1, 1) = s_\perp$ and $f(2^n, 2^n) = s_\top$,
(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 a tiling f that satisfies all the constraints in a given instance I. This problem is known to be NEXP-complete [Boas 1997].

Now, we fix an instance $I = (n, S, H, V, s_\perp, s_\top)$ of the exponential tiling problem and we construct some regular languages S, T of unranked trees such that S is bounded repairable into T if and only if there is *no tiling* satisfying the constraints of I. The basic idea is to let the source language S contain encodings of the possible tilings, and the target language T contain modified encodings that expose violations of some constraint of I. The intended relation between S and T can be phrased as follows: if every tree in S can be transformed into a tree in T with a small (i.e. uniformly bounded) amount of edits, then every tiling of the exponential grid violates some constraint in I, and vice versa. In order to forbid a repair strategy to modify the encoded tiling with a bounded amount of edits, we will allow some redundancy in our encodings. For convenience, we first describe the languages S and T as if they were given by means of stepwise automata of polynomial size with respect to the instance I. Towards the end

of the proof, we will show how to modify the constructions in order to get languages representable by non-recursive deterministic DTDs of polynomial size.

Source language. We begin by describing the trees in the language S. For the sake of the brevity, we let $N = 2^n$ be the width of the grid to be tiled and we consider a generic tiling $f : \{1,\dots,N\} \times \{1,\dots,N\} \to S$. A tree that encodes the tiling f is labeled over an alphabet consisting of tiles in S, separator symbols [and], and a dummy symbol #. Each cell (i,j) in the grid is encoded by a series of consecutive leaves that spell out a word of the form $[[\dots[\ \ f(i,j)\ f(i,j)\ \dots\ f(i,j)\ \ ]\dots]]$, where each symbol $f(i,j)$ occurs at least once and the square brackets are not necessarily well-parenthesized. The repetitions of the symbols $f(i,j)$ are used to ensure robustness to any repair strategy of bounded cost. From now on, such repetitions will be simply represented by a superscript +. The above word encoding a cell (i,j) is called a *cell-block*. Cell-blocks are then juxtaposed to form the frontier of a tree, following the left-to-right bottom-to-top order of the corresponding cells in the grid:

$$\underbrace{[^+ f(1,1)^+\,]^+ \dots [^+ f(1,N)^+\,]^+}_{\text{row } 1}\ \underbrace{[^+ f(2,1)^+\,]^+ \dots [^+ f(2,N)^+\,]^+}_{\text{row } 2}\ \dots\ \underbrace{[^+ f(N,1)^+\,]^+ \dots [^+ f(N,N)^+\,]^+}_{\text{row } N}$$

Finally, #-labeled internal nodes are introduced to guarantee that the frontier is well formed, namely, it contains exactly N rows, each one consisting of N cell-blocks. This can be done by enforcing the existence of $2n+1$ levels above the frontier and by requiring that each internal node at level $\ell = 0 \dots 2n-1$ has exactly two children, and each internal node at level $2n$ has a cell-block as childhood (see Figure 13).

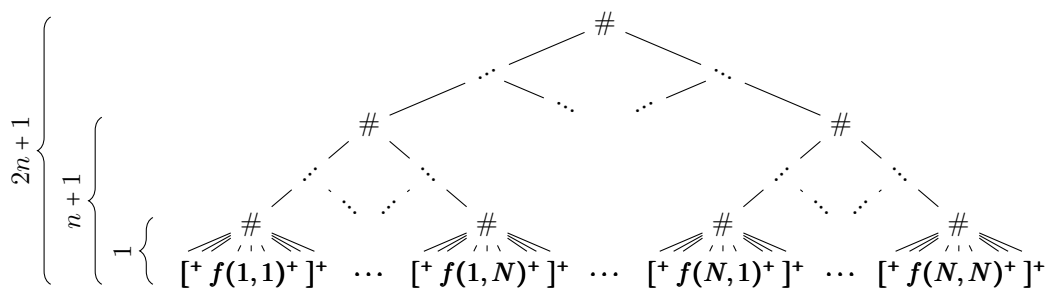


Fig. 13. Redundant encoding of a tiling by an unranked tree.

The source language S is defined as the set of all tree-shaped encodings of tilings f that satisfy the first constraint of I, namely, those tilings f such that $f(1,1) = s_\perp$ and $f(N,N) = s_\top$. The language S is clearly regular. Furthermore, it is not difficult to construct a stepwise automaton $\mathcal{S}$ that recognizes S and has size polynomial in n and $|S|$, and hence also in $|I|$ (we omit the formal definition of such an automaton).

Target language. We now turn to the target language T, which intuitively contains encodings of S modified in a suitable way so as to expose violations of horizontal or vertical constraints, which can then be checked by an automaton of small size.

We begin by analyzing the simpler case of a tiling f that violates a horizontal constraint, say between two tiles $f(i,j-1)$ and $f(i,j)$. Observe that, in the frontier of every tree of S that encodes f, the violating tiles are represented by two consecutive cell-blocks of the form $[^+ f(i,j-1)^+\,]^+$ and $[^+ f(i,j)^+\,]^+$. It is then convenient to expose the violation at the least common ancestor of these two cell-blocks, which must occur at

some level $\ell \in \{n, \ldots, 2n-1\}$. For example, this can be done by relabeling the least common ancestor with the pair $\big(f(i,j-1), f(i,j)\big)$ ($\notin H$). In this case, the modified encoding looks like the unranked tree in Figure 14 (for the sake of clearness, we highlighted the cell-blocks corresponding to the tiles that violate the horizontal constraint).

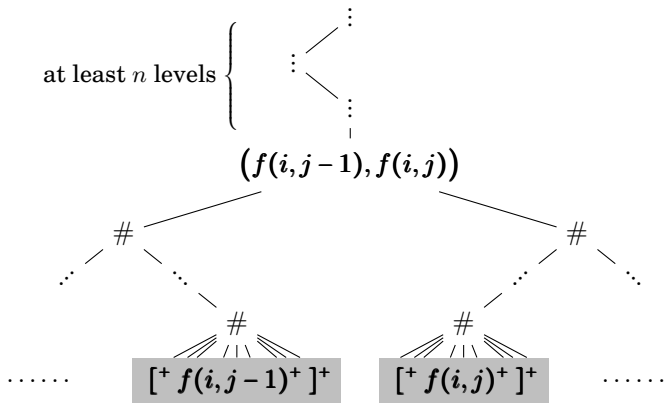


Fig. 14. Exposure of a violation of a horizontal constraint.

We denote by T_H the language of all trees that can be obtained by relabeling a node of some tree in S as described above. We observe that the language T_H is regular and, furthermore, one can construct a stepwise automaton that recognizes T_H and has size polynomial in $|I|$.

We now deal with the case of a tiling t that violates a vertical constraint, say between tiles $f(i-1,j)$ and $f(i,j)$. The basic idea here is to "hide" under a new subtree the factor of the frontier that starts with the corresponding occurrence of the cell-block $[^+ f(i-1,j)^+]^+$ and ends just before the occurrence of the cell-block $[^+ f(i,j)^+]^+$. Note that this factor contains exactly N cell-blocks, so it can be hidden under a complete binary tree of height n, such as the one depicted in Figure 15.

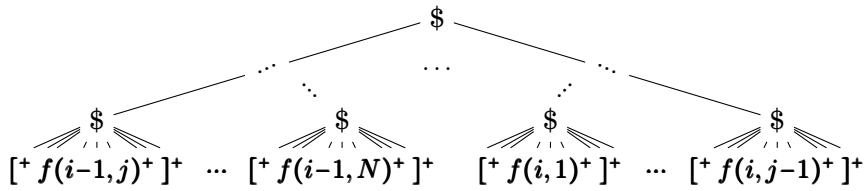


Fig. 15. Factor of a frontier delimited by vertically adjacent tiles.

Similarly, the remaining part of the frontier consists of $N-1$ sequences, each one containing N cell-blocks, so this shape can be enforced using an almost complete binary tree of height $2n$, where exactly one node at level n (e.g. the rightmost one) is a leaf. Putting all together, the modified encoding for the tiling f has the shape depicted in Figure 16 (as before, we highlighted the cell-blocks corresponding to the tiles that violate the vertical constraint).

Accordingly, we define the language T_V of all unranked trees of the above form, for all possible choices of $1 < i \leq N$ and $1 \leq j \leq N$ such that $\big(f(i-1,j), f(i,j)\big) \notin V$. Note that the latter condition can be checked by a small automaton that compares the highlighted

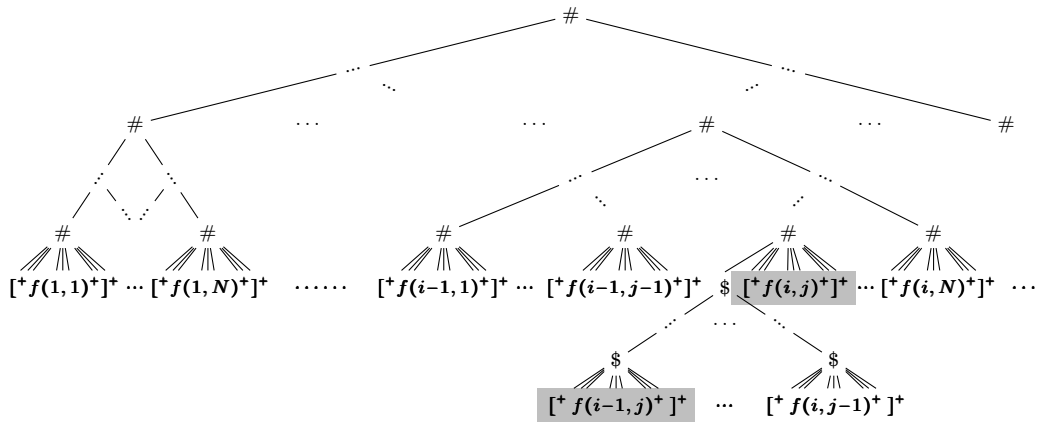


Fig. 16. Exposure of a violation of a vertical constraint.

cell-blocks in the figure. In particular, the language T_V is recognized by a stepwise automaton of size polynomial in $|I|$.

We can finally construct the target language T as the union of T_H and T_V and recall that this is also recognized by an automaton of polynomial size in $|I|$.

Reduction. Now, we need to argue that S is bounded repairable into T iff every tiling of the exponential grid violates some constraint of I. We begin with the easier direction, which assumes that every tiling violates some constrain of I. Consider a generic tree $t \in S$ that encodes a tiling f and let $t' \in T$ be the modified encoding that exposes a violation of a horizontal or vertical constraint, as described above. We observe that the frontiers of t and t' spell out the same sequence of cell-blocks. In particular, t' can be obtained from t by deleting all internal nodes and by inserting new internal nodes. Since the number of internal nodes in t and t' is uniformly bounded by a constant (roughly $\mathcal{O}(2^{2n})$), we know that S is bounded repairable into T.

As for the other direction, suppose that there is a tiling f that satisfies all constraints of I. We fix an arbitrarily large number K and we prove that some tree $t \in S$ requires at least K edits in order to be transformed into a tree of T. The tree t is nothing but the encoding of the tiling f, where each symbol in a cell-block is repeated K times; more precisely, t is the unranked tree of Figure 13 where every superscript + is replaced with K. We observe that every transformation of t consisting of less than K edits preserves at least one occurrence of each symbol in the frontier, and it also preserves the post-order relationships between these occurrences. Furthermore, note that occurrences of symbols [and] ensure that every transformation do not change or mix the order of the cell-block. This means that every such transformation produces a tree whose frontier contains a subsequence that encodes the same tiling f as t. Consider now a generic tree $t' \in T$. We observe that the frontier of t' contains exactly N^2 cell-blocks, so it encodes a tiling f' of the exponential grid. Moreover, by definition of T, the tiling f' must violate some constraint of I. We thus conclude that f and f' must be different tilings, and hence t' cannot be obtained as an editing of t with cost less than K. The above argument holds for any arbitrarily large number K, so this proves that S is not bounded repairable into T.

From automata to DTDs. It remains to show how to modify the languages S and T in such a way that they can be succinctly described by non-recursive deterministic DTDs. The general idea is to annotate the internal nodes of the trees in S and T with enough information so as to ease a deterministic top-down processing. First of all, we need to

annotate the internal nodes of all trees of S and T with their levels: this is possible thanks to the fact that the considered trees have height at most $3n+2$. In addition, we mark the leftmost and rightmost paths of the trees of S with special labels, say 1 and 2 respectively (the marking at the root is irrelevant): this makes it possible to check, by means of a DTD, that the first and last cell-blocks are of the form $[^+ s^+_\perp]^+$ and $[^+ s^+_\top]^+$. As for the trees in T, the crux is to ease the certification of a violation of a horizontal or vertical constraint. To do so, we can promote the information about the violating tiles up to the root. More precisely, on the trees depicted in Figure 14 and Figure 16, we consider the access path to the first highlighted cell-block and we annotate all internal nodes along this path with the corresponding tile; in a similar way, we add a second annotation for the access path to the second highlighted cell-block. For example, the parent of the cell-block $[^+ f(i,j)^+]^+$ of the tree of Figure 16 will be labeled with the tuple $\big(\#, 2n, f(i-1,j), f(i,j)\big)$, where $2n$ is the level of that node, and $f(i-1,j)$ and $f(i,j)$ indicate the tiles corresponding to the first and second highlighted cell-blocks.

The additional information on the labeling of the trees of S and T makes it easy to describe these languages by means of non-recursive deterministic DTDs of size polynomial in $|I|$. Finally, because only internal nodes are affected by the new annotation, the same arguments for the proof of the reduction can be used here. □

Combining Theorems 7.3 and 7.4, we obtain that the bounded repair problem for tree languages represented by all standard specifications [Martens et al. 2006], that is, stepwise tree automata, deterministic stepwise tree automata, XML Schema, DTDs, non-recursive deterministic DTDs, is coNEXP-complete.

7.3. Simpler instances

In order to find sub-cases of the bounded repair problem with a lower complexity, we consider a specialization of the problem where the alphabet Σ of the source language is fixed. We show that, in this case, the problem is PSPACE-complete for languages represented by non-deterministic DTDs, and coNP-complete for languages represented by deterministic DTDs.

Let us first discuss the complexity upper bounds. Suppose that D is a DTD defining a source language over the fixed alphabet Σ. A close inspection to the translation from DTDs to stepwise automata [Champavère et al. 2009] discloses the following crucial property (see the appendix for the proof):

LEMMA 7.5. *Given a non-deterministic (resp., deterministic) DTD D that defines a source language S over an alphabet Σ, one can compute in polynomial time a non-deterministic (resp., a deterministic) stepwise automaton $\mathcal{S} = (\Sigma, Q, \delta, \delta_0, F)$ that recognizes S and whose state space can be partitioned into $k \le 2|\Sigma|$ subsets $Q_1, \ldots, Q_k$ such that*

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

For example, the automaton $\mathcal{S}$ described in Example 3.1 is a deterministic stepwise automaton whose state space can be partitioned into 9 sets that satisfy the first part of the claim: $Q_1 = \{p^a_0\}$, $Q_2 = \{p^a_1\}$, $Q_3 = \{p^b_0\}$, $Q_4 = \{p^b_1\}$, $Q_5 = \{p^c_0\}$, $Q_6 = \{p^d_0\}$, $Q_7 = \{p^d_1\}$, $Q_8 = \{p^r_0, p^r_1\}$, $Q_9 = \{p^r_2\}$.

Lemma 7.5 above implies that any path in the transition graph of $\mathcal{S}$ (see, for instance, the left-hand side graph of Figure 7) traverses at most $2|\Sigma| - 1$ *vertical* edges that connect pairs of states in different components. As a consequence, any primitive

synopsis tree of $\mathcal{S}$ has size at most $|Q|^{2|\Sigma|}$, i.e., polynomial in the size of $\mathcal{S}$ when Σ is fixed.

Putting together Lemma 7.5, Corollary 7.2, and Theorem 5.7 one obtains a PSPACE (resp., CONP) algorithm that decides whether $\mathsf{cost}(S,T) < \infty$, where S and T are languages defined by non-deterministic (resp., deterministic) DTDs and S is over a fixed alphabet Σ. The algorithm has the same structure of the algorithm sketched before Theorem 7.3. Namely, it translates the input DTDs into equivalent stepwise automata $\mathcal{S}$ and $\mathcal{T}$, then it translates $\mathcal{S}$ and $\mathcal{T}$ into the grammars $G_{\mathcal{S}}$ and $G_{\mathcal{S},\mathcal{T}}$, and finally it checks whether $\mathscr{L}(G_{\mathcal{S}}) \subseteq \mathscr{L}(G_{\mathcal{S},\mathcal{T}})$. As previously stated, the last step of the algorithm can be done in CONP by universally-guessing a word w of polynomial size from $\mathscr{L}(G_{\mathcal{S}})$ and checking whether $w \in \mathscr{L}(G_{\mathcal{S},\mathcal{T}})$ in polynomial time. Note that the translation of $\mathcal{S}$ and $\mathcal{T}$ into $G_{\mathcal{S}}$ and $G_{\mathcal{S},\mathcal{T}}$ takes polynomial space for non-deterministic DTDs and polynomial time for deterministic DTDs. The blow-up of the complexity for the former is because one has to check language containment between regular languages which can be done with polynomial space.

PROPOSITION 7.6. *The bounded repair problem between a source language represented by a non-deterministic (resp., deterministic) DTD over a fixed alphabet and a target language represented by a non-deterministic (resp., deterministic) DTD over an arbitrary alphabet is in* PSPACE *(resp., in* CONP*).*

Finally, we show that even strong restrictions, including fixing both alphabets, cannot get us below PSPACE in the non-deterministic case. Indeed, one can easily reduce the containment problem between regular expressions to a bounded repair problem between languages defined by *non-recursive non-deterministic* DTDs, thus showing that the latter problem is PSPACE-hard. To see this, consider two regular expressions E_1 and E_2. Let $\#$ be a fresh symbol and let $E_1^{\#}$ and $E_2^{\#}$ be the expressions obtained from E_1 and E_2, respectively, by substituting every occurrence of a symbol a with the expression $a^* \#$. Let r be another fresh letter reserved for the roots of the trees. Clearly, the language defined by E_1 is contained in the language defined by E_2 if and only if the DTD $r \to E_1^{\#}$ is bounded repairable into the DTD $r \to E_2^{\#}$ (one direction is trivial and the other is easily shown by contraposition). As the latter DTDs are non-recursive (they define trees of height two), this shows that the bounded repair problem between non-recursive non-deterministic DTDs is PSPACE-hard, and this holds even when the alphabets are fixed.

We can also provide a CONP lower bound for the analogous problem when the languages are represented by non-recursive *deterministic* DTDs over fixed alphabets. This lower bound follows easily from a reduction from the validity problem for propositional formulas in disjunctive form. A similar reduction was given in [Benedikt et al. 2013] for languages of words recognized by deterministic finite automata. The additional complication here is that we have to fix the source and the target alphabets; however, the reduction is still possible by encoding the valuation of each variable with a block of nodes labeled over a binary alphabet.

PROPOSITION 7.7. *The bounded repair problem between languages represented by non-recursive non-deterministic (resp., deterministic) DTDs with both source and target alphabets fixed is* PSPACE*-hard (resp.,* CONP*-hard).*

8. THE UNIVERSAL CASE

In this section we consider the so-called universal case of the bounded repairability problem, namely, a variant of the problem where the source 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.

Here we also make use of *deterministic visibly pushdown transducers* [Raskin and Servais 2008; Alur and Madhusudan 2009] 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.

The following result gives equivalent conditions for bounded repairability in the universal case.

PROPOSITION 8.1. *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. Here we only prove that the second item implies the third one (the other two directions are explained in the appendix). 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 $\mathsf{ext}^{-1}(\mathscr{L}(\mathcal{T}))$. Namely, to complete the proof we need to show the following claim.

CLAIM 2. *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 $p \in \delta(\mathsf{ext}(a(u_1, \ldots, u_n, t, v_1, \ldots, v_m)))$.*

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. $\square$

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 EXP: 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., P for deterministic stepwise automata and EXP for non-deterministic stepwise automata) are tight. The hardness proofs can be derived from reductions of the emptiness and universality problems, respectively, on the corresponding classes of automata (see appendix).

PROPOSITION 8.2. *The bounded repair problem in the universal case when the target language is represented by a non-deterministic (resp., deterministic) stepwise automaton is* EXP*-complete (resp.,* P*-complete).*

9. CONCLUSIONS

In the present paper we have investigated the bounded repairability problem for regular tree languages. We have provided an effective characterization of bounded repairability and characterized the complexity of testing whether a given source language S is bounded repairable w.r.t. a given target language T. The characterization can be used with a number of different formalisms for representing the tree languages: tree automata, XML Schemas, DTDs, as well as their non-recursive and deterministic restrictions. While in general the problem is CONEXP-complete, its complexity is considerably reduced for DTDs over fixed alphabets. In the latter case the problem becomes CONP-complete or PSPACE-complete, depending on whether the DTDs are deterministic or not. Finally, we have also considered the variant of the problem when the source language is set to be universal. In this case, we have shown that the problems is EXP-complete in general, and becomes tractable (P-complete, in fact) when a deterministic bottom-up automaton is used.

Several directions of future work can be envisioned. Bounded repairability is essentially a generalization of inclusion between tree languages modulo a bounded number of editing operations. One could attempt to further generalize it allow a number of editing operations that is bounded by a ratio of the size of the input tree. In [Benedikt et al. 2014] it is shown how such a generalized notion of repairability can be computed for regular string languages. It would be interesting to see if the employed methods can be adapted to the setting of regular tree languages. Another direction is bounded repairability in the streaming setting: not only the pair of source and target languages need to be bounded repairable, but, furthermore, the repair must be executable by a transducer, namely, a machine with a possibly infinite state space that makes one pass over the serialization of the input tree while producing a serialization of the output tree. Proposition 8.1 shows that in the unrestricted case visibly pushdown transducers are expressive enough to implement bounded-cost streaming repairs, whenever these exist. In the general case, however, visibly pushdown transducers may be too limiting [Bourhis et al. 2013].

REFERENCES

Foto Afrati and Phokion Kolaitis. 2009. Repair checking in inconsistent databases: algorithms and complexity. In *Proceedings of the 12th International Conference on Database Theory (ICDT)*. ACM, 31–41.

Alfred Aho and Thomas Peterson. 1972. A minimum distance error-correcting parser for context-free languages. *SIAM J. Comput.* 1, 4 (1972), 305–312.

Rajeev Alur and Parthasarathy Madhusudan. 2009. Adding nesting structure to words. *Journal of the ACM (JACM)* 56, 3 (2009), 16.

Shunichi Amano, Leonid Libkin, and Filip Murlak. 2009. XML schema mappings. In *Proceedings of the 28th Symposium on Principles of Database Systems*. 33–42.

Anonymous. details omitted due to double-blind reviewing.

Timos Antonopoulos, Floris Geerts, Wim Martens, and Frank Neven. 2013. Generating, sampling and counting subclasses of regular tree languages. *Theory of Computing Systems* 52, 3 (2013), 542–585.

Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM SIGMOD Symposium on Principles of Database Systems (PODS)*. ACM, 68–79.

Marcelo Arenas and Leonid Libkin. 2008. XML data exchange: consistency and query answering. *J. ACM* 55, 2 (2008).

Charles Babbage. 1864. *Passages from the Life of a Philosopher*. Longman, Green, Longman, Roberts, & Green.

Michael Benedikt, Gabriele Puppis, and Cristian Riveros. 2013. Bounded repairability of word languages. *J. Comput. System Sci.* 79, 8 (2013), 1302–1321.

Michael Benedikt, Gabriele Puppis, and Cristian Riveros. 2014. The per-character cost of repairing word languages. *Theoretical Computer Science* 539 (2014), 38–67.

Leopoldo Bertossi. 2011. Database repairing and consistent query answering. *Synthesis Lectures on Data Management* 3, 5 (2011), 1–121.

Philip Bille. 2005. A survey on tree edit distance and related problems. *Theoretical Computer Science (TCS)* 337, 1 (2005), 217–239.

Peter Van Emde Boas. 1997. The convenience of tilings. In *Complexity, Logic, and Recursion Theory*. Marcel Dekker Inc, 331–363.

Mikołaj Bojańczyk, Leszek A Kołodziejczyk, and Filip Murlak. 2011. Solutions in XML data exchange. In *Proceedings of the 14th International Conference on Database Theory*. ACM, 102–113.

Utsav Boobna and Michel de Rougemont. 2004. Correctors for XML data. In *Database and XML Technologies*. 97–111.

Pierre Bourhis, Gabriele Puppis, and Cristian Riveros. 2013. Which DTDs are streaming bounded repairable?. In *Proceedings of the 16th International Conference on Database Theory (ICDT)*. ACM, 57–68.

Anne Brüggemann-Klein and Derick Wood. 1998. One-unambiguous regular languages. *Information and Computation* 142, 2 (1998), 182–206.

Julien Carme, Joachim Niehren, and Marc Tommasi. 2004. Querying unranked trees with stepwise tree automata. In *Rewriting Techniques and Applications (RTA)*. Springer, 105–118.

Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. 2009. Efficient inclusion checking for deterministic tree automata and XML schemas. *Information and Computation* 207, 11 (2009), 1181–1208.

Shan Chen, Dan Hong, and Vincent Y Shen. 2005. An experimental study on validation problems with existing html webpages. In *Proceedings of the 2005 International Conference on Internet Computing, ICOMP'05*. 373.

Dario Colazzo, Giorgio Ghelli, Luca Pardini, and Carlo Sartiani. 2013. Almost-linear inclusion for XML regular expression types. *ACM Trans. Database Syst.* 38, 3 (2013), 15.

Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. 2007. Tree Automata Techniques and Applications. Available on: http://www.grappa.univ-lille3.fr/tata. (2007).

Grapham Cormode and S. Muthukrishnan. 2007. The String Edit Distance Matching Problem with Moves. *ACM Trans. Algorithms* 3, 1 (2007), 2:1–2:19. DOI: http://dx.doi.org/10.1145/1186810.1186812

Jay Earley. 1970. An efficient context-free parsing algorithm. *Commun. ACM* 13, 2 (1970), 94–102.

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

Wenfei Fan and Philip Bohannon. 2008. Information preserving XML schema embedding. *ACM Transactions on Database Systems (TODS)* 33, 1 (2008), 4.

Sergio Flesca, Filippo Furfaro, Sergio Greco, and Ester Zumpano. 2005. Querying and repairing inconsistent XML data. In *Web Information Systems Engineering (WISE) (LNCS)*, Vol. 3806. Springer, 175–188.

Gösta Grahne and Alex Thomo. 2004. Query answering and containment for regular path queries under distortions. In *Foundations of Information and Knowledge Systems (FOIKS)*. Springer, 98–115.

Steven Grijzenhout and Maarten Marx. 2013. The quality of the XML web. *Web Semantics: Science, Services and Agents on the World Wide Web* 19 (2013), 59–68.

John Hopcroft and Jeffrey Ullman. 1979. *Introduction to automata theory, languages, and computation*. Addison-Wesley.

Flip Korn, Barna Saha, Divesh Srivastava, and Shanshan Ying. 2013. On repairing structural problems in semi-structured data. *Proceedings of the VLDB Endowment* 6, 9 (2013), 601–612.

William Lidwell, Kritina Holden, and Jill Butler. 2010. *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions*. Rockport Publishers.

Wim Martens, Frank Neven, and Thomas Schwentick. 2009. Complexity of Decision Problems for XML Schemas and Chain Regular Expressions. *SIAM J. Comput.* 39, 4 (2009), 1486–1530.

Wim Martens, Frank Neven, Thomas Schwentick, and Geert Jan Bex. 2006. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems (TODS)* 31, 3 (2006), 770–813.

Wim Martens and Joachim Niehren. 2007. On the minimization of XML schemas and tree automata for unranked trees. *Journal of Computer and System Sciences (JCSS)* 73, 4 (2007), 550–583.

Ejike Ofuonye, Patricia Beatty, Scott Dick, and James Miller. 2010. Prevalence and classification of web page defects. *Online Information Review* 34, 1 (2010), 160–174.

Erhard Rahm and Philip A Bernstein. 2001. A survey of approaches to automatic schema matching. *the VLDB Journal* 10, 4 (2001), 334–350.

Jean-François Raskin and Frédéric Servais. 2008. Visibly Pushdown Transducers. In *Automata, Languages and Programming (ICALP)*. LNCS, Vol. 5126. Springer, 386–397.

Thomas Schwentick. 2007. Automata for XML – a survey. *Journal of Computer and System Sciences (JCSS)* 73, 3 (2007), 289–315.

Luc Segoufin and Cristina Sirangelo. 2007. Constant-Memory Validation of Streaming XML Documents Against DTDs. In *Database Theory - ICDT 2007, 11th International Conference, Barcelona, Spain, January 10-12, 2007, Proceedings*. 299–313.

Luc Segoufin and Victor Vianu. 2002. Validating streaming XML documents. In *Proceedings of the 21th ACM SIGMOD Symposium on Principles of Database Systems (PODS)*. ACM, 53–64.

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

Slawomir Staworko and Jan Chomicki. 2006. Validity-sensitive querying of XML databases. In *Current Trends in Database Technology (EDBT workshops)*. LNCS, Vol. 4254. Springer, 164–177.

Slawomir Staworko, Emmanuel Filiot, and Jan Chomicki. 2008. Querying Regular Sets of XML Documents. In *International Workshop on Logic in Databases (LiD)*.

Larry Stockmeyer and Albert Meyer. 1973. Word problems requiring exponential time (Preliminary Report). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 1–9.

Nobutaka Suzuki. 2005. Finding an Optimum Edit Script between an XML Document and a DTD. In *Proceedings of the 2005 ACM symposium on Applied computing*. ACM, 647–653.

Kuo-Chung Tai. 1979. The tree-to-tree correction problem. *Journal of the ACM (JACM)* 26, 3 (1979), 422–433.

Robert Wagner. 1974. Order-n correction for regular languages. *Communications of the ACM (CACM)* 17, 5 (1974), 265–268.

Daniel H. Younger. 1967. Recognition and Parsing of Context-Free Languages in Time nˆ3. *Information and Control* 10, 2 (1967), 189–208.

A. APPENDIX

LEMMA 7.5. *Given a non-deterministic (resp., deterministic) DTD D that defines a restriction language R over an alphabet Σ, one can compute in polynomial time a non-deterministic (resp., a deterministic) stepwise automaton $\mathcal{R} = (\Sigma, Q, \delta, \delta_0, F)$ that recognizes R and whose state space can be partitioned into $k \leq 2|\Sigma|$ subsets $Q_1, \ldots, Q_k$ such that*

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

PROOF. In this proof we follow the translation from DTDs to equivalent stepwise automata given in [Champavère et al. 2009]. Specifically, we first translate a DTD to a special form of stepwise automaton with ε-moves (called factorized automaton) and then we translate the latter automaton to an equivalent stepwise automaton. This translation preserves determinism by a suitable notion of determinism for factorized automata. We recall below the definition of factorized automaton from [Champavère et al. 2009].

A *factorized automaton* is a tuple $\mathcal{A} = (\Sigma, Q_1 \uplus Q_2, \delta_0, \delta, \delta_\varepsilon, F)$, where Σ is an input alphabet used to label leaves of curried trees, Q_1 and Q_2 are two disjoint sets of states, $\delta_0 : \Sigma \to 2^{Q_1 \uplus Q_2}$ is an assignment of labels to states, $\delta : Q_1 \times Q_2 \to 2^{Q_1 \uplus Q_2}$ is a transition function, $\delta_\varepsilon : Q_1 \uplus Q_2 \to 2^{Q_1 \uplus Q_2}$, and $F \subseteq Q_1 \uplus Q_2$ is a set of final states. Note that, according to the above definition, for every transition rule $q_1 @ q_2 \to q$ of a factorized automaton, we have that q_1 belongs to the first set Q_1 of states and q_2 belongs to the second set Q_2 of states. We call this last property *factorization of states*. We shortly write $p \to_\varepsilon q$ (resp., $p \to_\varepsilon^* q$) whenever $q \in \delta_\varepsilon(p)$ (resp., whenever there is a sequence of states $q_0, \ldots, q_n$, with $n \in \mathbb{N}$, such that $q_0 = p$, $q_n = q$, and $q_{i+1} \in \delta_\varepsilon(q_i)$ for all $0 \leq i < n$).

We can extend in a natural way the notions of run and transition graph of a factorized automaton $\mathcal{A}$. The latter consists of states of $\mathcal{A}$, horizontal edges, vertical edges, and ε-edges. We can thus speak of a strongly connected component of a factorized automaton $\mathcal{A}$. In this case, the accessibility relation between states takes into account the horizontal edges, the vertical edges, and the ε-edges of the transition graph of $\mathcal{A}$. Accordingly, we denote by $\mathscr{L}(\mathcal{A} \mid X)$ the set of all contexts realized within a component X of $\mathcal{A}$.

We say that a factorized automaton $\mathcal{A} = (\Sigma, Q_1 \uplus Q_2, \delta_0, \delta, \delta_\varepsilon, F)$ is *deterministic* if

(1) the sub-automaton $(\Sigma, Q_1 \uplus Q_2, \delta_0, \delta, F)$, thought of as a stepwise automaton, is deterministic,
(2) for all states $p \in Q_1 \uplus Q_2$ and all states $q, q' \in Q_1$ (resp., $q, q' \in Q_2$), $p \to_\varepsilon^* q$ and $p \to_\varepsilon^* q'$ imply $q = q'$, i.e. the ε-transition relation can be described as a pair of partial functions $\delta_{\varepsilon,1} : Q_1 \uplus Q_2 \rightharpoonup Q_1$ and $\delta_{\varepsilon,1} : Q_1 \uplus Q_2 \rightharpoonup Q_2$.

From now on we fix a DTD $D = (\Sigma, d, S)$, which can be either deterministic or non-deterministic (depending on this, the resulting automata will be deterministic or non-deterministic). Given a symbol a of the alphabet of D, we denote by $\mathcal{A}_a = (\Sigma, P_a, p_{a,0}, \delta_a, F_a)$ the Glushkov automaton [Brüggemann-Klein and Wood 1998] that recognizes the regular language $\mathscr{L}(d(a)) \subseteq \Sigma^*$. Recall that $\mathcal{A}_a$ can be constructed in polynomial time in the size of $d(a)$ and $\mathcal{A}_a$ is deterministic iff $d(a)$ is deterministic [Brüggemann-Klein and Wood 1998].

We recall the translation of the DTD D into an equivalent factorized automaton $\mathcal{R}$ (see, for instance, Theorem 5 in [Champavère et al. 2009]). The states of the factorized automaton $\mathcal{R}$ are given by the disjoint union of the alphabet Σ, whose elements are now thought of as states, and the sets of states of the finite automata $\mathcal{A}_a$, for all symbols

$a \in \Sigma$. The assignment of initial states of $\mathcal{R}$ to labels is given by the function $\delta_0(a) = p_{a,0}$, where $p_{a,0}$ is the initial state of $\mathcal{A}_a$. The transition relation δ of $\mathcal{R}$ contains all rules of the form $p@b \to q$ such that $\delta_a(p,b) = q$, for some $a, b \in \Sigma$ and some $p, q \in P_a$. The ε-transition relation δ_ε connects every final state $p \in F_a$, with $a \in \Sigma$, to the state a of $\mathcal{R}$. Finally, the final states of $\mathcal{R}$ are the states in F_a, for any initial symbol a of the DTD. It is easy to check that the defined object $\mathcal{R}$ is a *deterministic* factorized automaton if each automaton $\mathcal{A}_a$ is deterministic.

As shown in [Champavère et al. 2009] (Proposition 4), the factorized automaton $\mathcal{R}$ can be turned in polynomial time into an equivalent stepwise automaton $\mathcal{R}'$, having the same set $Q = \Sigma \uplus \biguplus_{a \in \Sigma} P_a$ of states and possibly more transitions. This is done by replacing each transition $p@b \to q$ in $\mathcal{R}$ by the set of all transitions $p'@r \to q$ for which there exist two sequences of ε-moves $p' \to_\varepsilon^* p$ and $r \to_\varepsilon^* b$ in $\mathcal{R}$. Observe that if $\mathcal{R}$ is deterministic, then the factorization of states of $\mathcal{R}$ guarantees that the resulting stepwise automaton $\mathcal{R}'$ is also deterministic.

We make another important remark related to the structure of the automaton $\mathcal{R}'$: the translation from $\mathcal{R}$ to $\mathcal{R}'$ defined above preserves the set of states and the accessibility relation in the corresponding transition graph and hence it also preserves the strongly connected components. Therefore, in order to complete the proof of the lemma, it is sufficient to show that the following claim hold for the factorized automaton $\mathcal{R}$:

CLAIM 3. *The set of states of $\mathcal{R}$ can be partitioned into $k \le 2|\Sigma|$ subsets $Q_1, \ldots, Q_k$ such that*

— *every component of $\mathcal{R}$ is contained in some set Q_i,*
— *for every transition $p@b \to q$ and for every sequence of ε-moves $r \to_\varepsilon^* b$ in $\mathcal{R}$, if r and q are in different components, then $r \in Q_i$ and $q \in Q_j$ for some $1 \le i < j \le k$.*

PROOF. Let $Q_2, Q_4, \ldots, Q_{2h}$ be the strongly connected components of $\mathcal{R}$ that contain at least one state from Σ. Furthermore, for every $1 \le i \le h$, let Q_{2i-1} be the union of the sets $P_b \setminus Q_{2i}$, for any $b \in \Sigma \cap Q_{2i}$. Recall that P_b is the state space of the word automaton $\mathcal{A}_b$, which is contained in the state space of $\mathcal{R}$. Below we show that the sets $Q_1, Q_2, \ldots, Q_k$, where $k = 2h$, define the desired partition.

First of all, note that $h \le |\Sigma|$ and hence $k \le 2|\Sigma|$. It is also easy to check that the sets $Q_1, \ldots, Q_k$ form a partition of the state space of $\mathcal{R}$. Indeed, every state of $\mathcal{R}$ is either a symbol in Σ and hence it belongs to some set among $Q_2, Q_4, \ldots, Q_{2h}$, or it is a state of some automaton $\mathcal{A}_b$ and hence it belongs to some set among $Q_1, Q_3, \ldots, Q_{2h-1}$.

We also claim that every component of $\mathcal{R}$ is contained in some set Q_i, with $1 \le i \le k$. Indeed, any component of $\mathcal{R}$ either contains a state from Σ and hence it coincides with some set among $Q_2, Q_4, \ldots, Q_{2h}$, or it consists only of states from a single automaton $\mathcal{A}_b$ and hence it is contained into some set among $Q_1, Q_3, \ldots, Q_{2h-1}$.

We now check the second item of the claim. We can assume, without loss of generality, that the sets $Q_1, \ldots, Q_k$ have been listed according to a topological order that respects the accessibility relation of the transition graph of $\mathcal{R}$. Let us consider a transition $p@b \to q$ and a sequence of ε-moves $r \to_\varepsilon^* b$ in $\mathcal{R}$ and suppose that r and q are in different components of $\mathcal{R}$. We distinguish between two cases, depending on whether r and b are in the same component or not. If r and b are in the same component, then we know that this component must coincide with a set among $Q_2, Q_4, \ldots, Q_{2h}$, say Q_i. Recall that q belongs to a component different from Q_i, say Q_j. Since the sets $Q_1, \ldots, Q_k$ were listed according to a topological order, we immediately obtain $i < j$. In the second case, r and b belong to different components. From the definition of ε-moves of $\mathcal{R}$, we know that r is a final state of $\mathcal{A}_b$ and hence we have $r \in Q_{2i-1}$ and $b \in Q_{2i}$, for some $1 \le i \le h$. Finally, recall that q belongs to a component different from Q_{2i}, say Q_j. Again,

since the sets $Q_1, \ldots, Q_k$ were listed according to a topological order, we obtain $2i < j$. □

In view of the previous arguments, Claim 3 immediately implies our lemma. □ □

PROPOSITION 8.1. *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 4. *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. $\square$

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 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 $(C^{\text{prefix}}, C^{\text{suffix}})$ of sequences of opening and closing tags such that, for every tree t, $C^{\text{prefix}} \cdot t \cdot C^{\text{suffix}}$ 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

$$C^{\text{prefix}} \cdot \langle a \rangle \cdot u_1 \cdot \ldots \cdot u_n \cdot t \cdot v_1 \cdot \ldots \cdots v_m \cdot \langle a/ \rangle \cdot C^{\text{suffix}}.$$

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. $\square$

PROPOSITION 8.2. *The bounded repair problem in the unrestricted case when the target language is represented by a non-deterministic (resp., deterministic) stepwise automaton is* EXP*-complete (resp.,* P*-complete).*

PROOF. From Proposition 8.1 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 P for deterministic stepwise automata and in EXP for non-deterministic stepwise automata.

For the lower bounds, we show here that the problem is EXP-hard for non-deterministic stepwise automata by a reduction of the universality problem for stepwise automata. For the P 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 P-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:

$$\delta' = \delta \cup (F \times \{q_\#\} \times \{q_f\}) \cup (\{q_\#\} \times F \times \{q_f\}) \cup (\{q_\#\} \times \{q_\#\} \times \{q_\#\}) \cup (Q' \times \{q_f\} \times \{q_f\}) \cup (\{q_f\} \times Q' \times \{q_f\})$$

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 8.1, 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 EXP-hard as well. This completes the proof. □