A DECIDABLE CHARACTERIZATION OF LOCALLY TESTABLE TREE LANGUAGES

THOMAS PLACE AND LUC SEGOUFIN

INRIA and ENS Cachan, LSV

Abstract. A regular tree language L is locally testable if membership of a tree in L depends only on the presence or absence of some fix set of neighborhoods in the tree. In this paper we show that it is decidable whether a regular tree language is locally testable. The decidability is shown for ranked trees and for unranked unordered trees.

1. Introduction

This paper is part of a general program trying to understand the expressive power of first-order logic over trees. We say that a class of regular tree languages has a decidable characterization if the following problem is decidable: given as input a finite tree automaton, decide if the recognized language belongs to the class in question. Usually a decision algorithm requires a solid understanding of the expressive power of the corresponding class and is therefore useful in any context where a precise boundary of this expressive power is crucial. In particular we do not possess yet a decidable characterization of the tree languages definable in FO($\leq$), the first-order logic using a binary predicate $\leq$ for the ancestor relation.

We consider here the class of tree languages definable in a fragment of FO($\leq$) known as *Locally Testable* (LT). A language is in LT if membership in the language depends only on the presence or absence of neighborhoods of a certain size in the tree. A closely related family of languages is the class LTT of *Locally Threshold Testable* languages. Membership in such languages is determined by counting the number of neighborhoods of a certain size up to some threshold. The class LT is the special case where no counting is done, the threshold is 1. In this paper we provide a decidable characterization of the class LT over trees.

The standard approach for deriving a decidable characterization is to first exhibit a set of closure properties that hold exactly for the languages in the class under investigation and then show that these closure properties can be automatically tested. This requires a formalism for expressing the desired closure properties but also some tools, typically induction mechanisms, for proving that the properties do characterize the class, and for proving the decidability of those properties.

1998 ACM Subject Classification: F.4.3.

Key words and phrases: Regular Tree Languages, Locally Testable, Characterization.

LOGICAL METHODS
IN COMPUTER SCIENCE

DOI:10.2168/LMCS-???

Over words one formalism turned out to be successful for characterizing many classes of regular languages. The closure properties are expressed as identities on the syntactic monoid or syntactic semigroup of the regular language. The syntactic monoid or syntactic semigroup of a regular language is the transition monoid of its minimal deterministic automaton including or not the transition induced by the empty word. For instance the class of word languages definable in $\mathrm{FO}(\leq)$ is characterized by the fact that the syntactic monoid of any such languages is aperiodic. The latter property corresponds to the identity $x^\omega = x^{\omega+1}$ where ω is the size of the monoid. This equation is easily verifiable automatically on the syntactic monoid. Similarly, the classes LTT and LT have been characterized using decidable identities on the syntactic semigroup [BS73, McN74, BP89, TW85].

Over trees the situation is more complex and right now there is no formalism that can easily express all the known closure properties for the classes for which we have a decidable characterization. The most successful formalism is certainly the one introduced in [BW07] known as forest algebras. For instance, these forest algebras were used for obtaining decidable characterizations for the classes of tree languages definable in EF+EX [BW06], EF+F^{-1} [Boj07b, Pla08], BC-$\Sigma_1(<)$ [BSS08, Pla08], $\Delta_2(\leq)$ [BS08, Pla08]. However it is not clear yet how to use forest algebras in a simple way for characterizing the class LTT over trees and a different formalism was used for obtaining a decidable characterization for this class [BS10].

We were not able to obtain a reasonable set of identities for LT either by using forest algebras or the formalism used for characterizing LTT. Our approach is slightly different.

There is another technique that was used on words for deciding the class LT. It is based on the "delay theorem" [Str85, Til87] for computing the required size of the neighborhoods: Given an automaton recognizing the language L, a number k can be computed from that automaton such that if L is in LT then it is in LT by investigating the neighborhoods of size k. Once this k is available, deciding whether L is indeed in LT or not is a simple exercise. On words, a decision algorithm for LT (and also for LTT) has been obtained successfully using this approach [Boj07a]. Unfortunately all efforts to prove a similar delay theorem on trees have failed so far.

We obtain a decidable characterization of LT by combining the two approaches mentioned above. We first exhibit a set of necessary conditions for a regular tree language to be in LT. Those conditions are expressed using the formalism introduced for characterizing LTT. We then show that for languages satisfying such conditions one can compute the required size of the neighborhoods. Using this technique we obtain a characterization of LT for ranked trees and for unranked unordered trees.

Other related work. There exist several formalisms that have been used for expressing identities corresponding to several classes of languages but not in a decidable way. Among them let us mention the notion of preclones introduced in [EW05] as it is close to the one we use in this paper for expressing our necessary conditions.

Finally we mention the class of frontier testable languages, not expressible in $\mathrm{FO}(<)$, that was given a decidable characterization using a specific formalism [Wil96].

Organization of the paper. We start with ranked trees and give the necessary notations and preliminary results in Section 2. Section 3 exhibits several conditions and proves they are necessary for being in LT. In Section 4 we show that for the languages satisfying the

necessary conditions the required size of the neighborhoods can be computed, hence concluding the decidability of the characterization. Finally in Section 5 we show how our result extends to unranked trees.

2. Notations and preliminaries

We first investigate the case of binary trees. The case of unranked unordered trees will be considered in Section 5.

Trees. We fix a finite alphabet Σ, and consider finite binary trees with labels in Σ. All the results presented here extend to arbitrary ranks in a straightforward way. In the binary case, each node of the tree is either *a leaf* (has no children) or has exactly two *children*, the left child and the right child. We use the standard terminology for trees. For instance by the *descendant* (resp. ancestor) relation we mean the reflexive transitive closure of the child (resp. inverse of child) relation and by *distance* between two nodes we refer to the length of the shortest path between the two nodes. A *language* is a set of trees.

Given a tree t and a node x of t the *subtree of t rooted at x*, consisting of all the nodes of t that are descendant of x, is denoted by $t|_x$. A *context* is a tree with a designated (unlabeled) leaf called its *port* which acts as a hole. Given contexts C and C', their concatenation $C \cdot C'$ is the context formed by identifying the root of C' with the port of C. A tree $C \cdot t$ can be obtained similarly by combining a context C and a tree t. Given a tree t and two nodes x, y of t such that y is a descendant (not necessarily strict) of x, *the context of t between x and y*, denoted by $t[x, y]$, is defined by keeping all the nodes of t that are descendants of x but not descendants of y and by placing the port at y. We say that a context C *occurs* in t if C is the context of t between x and y for some nodes x and y of t.

Types. Let t be a tree and x be a node of t and k be a positive integer, the *k-type* of x is the (isomorphism type of the) restriction of $t|_x$ to the set of nodes of t at distance at most k from x. When k will be clear from the context we will simply say *type*. A k-type τ *occurs* in a tree t if there exists a node of t of type τ. If C is the context $t[x, y]$ for some tree t and some nodes x, y of t, then the k-type of a node of C is the k-type of the corresponding node in t. Notice that the k-type of a node of C depends on the surrounding tree t, in particular the port of C has a k-type, the one of y in t.

Given two trees t and t' we denote by $t \preccurlyeq_k t'$ the fact that all k-types that occur in t also occur in t'. Similarly we can speak of $t \preccurlyeq_k C$ when t is a tree and C is $t'[x, y]$ for some tree t' and some nodes x, y of t'. We denote by $t \simeq_k t'$ the property that the root of t and the root of t' have the same k-type and t and t' agree on their k-types: $t \preccurlyeq_k t'$ and $t' \preccurlyeq_k t$. Note that when k is fixed the number of k-types is finite and hence the equivalence relation $\simeq_k$ has a finite number of equivalence classes. This property is no longer true for unranked trees and this is why we will have to use a different technique for this case.

A language L is said to be κ-locally testable if L is a union of equivalence classes of $\simeq_\kappa$. A language is said to be *locally testable* (is in LT) if there is a κ such that it is κ-locally testable. In other words, in order to test whether a tree t belongs to L it is enough to check for the presence or absence of κ-types in t, for some big enough κ.

Regular Languages. We assume familiarity with tree automata and regular tree languages. The interested reader is referred to [CGJ+07] for more details. Their precise definitions are not important in order to understand our characterization. However pumping arguments will be used in the decision algorithms.

The problem. We want an algorithm deciding if a given regular language is in LT. When the complexity is not an issue, we can assume that the language L is given as a MSO formula. Another option would be to start with a bottom-up tree automaton for L or, even better, the minimal deterministic bottom-up tree automaton that recognize L. We will come back to the complexity issues in Section 7. The main difficulty is to compute a bound on κ, the size of the neighborhood, whenever such a κ exists.

The word case is a special case of the tree case as it corresponds to trees of rank 1. A decision procedure for LT was obtained in the word case independently by [BS73] and [McN74]. A language L is in LT if and only if its syntactic semigroup satisifies the equations $exe = exexe$ and $exeye = eyexe$, where e is an arbitrary idempotent ($ee = e$) while x and y are arbitrary elements of the semigroup. The equations are then easily verified after computing the syntactic semigroup.

In the case of trees, we were not able to obtain a reasonably simple set of identities for characterizing LT. Nevertheless we can show:

Theorem 2.1. *It is decidable whether a regular tree language is in LT.*

Our strategy for proving Theorem 2.1 is as follows. In a first step we provide necessary conditions for a language to be in LT. In a second step we show that if a language L verifies those necessary conditions then we can compute from an automaton recognizing L a number κ such that if L is in LT then L is κ-locally testable. The last step is simple and show that once κ is fixed, it is decidable whether a regular language is κ-locally testable. This last step follows immediately from the fact that once κ is fixed, there are only finitely many κ-locally testable languages and hence one can enumerate them and test whether L is equivalent to one of them or not.

Given a regular language L, testing whether L is in LT is then done as follows: (1) compute from L the κ of the second step and (2) test whether L is κ-locally testable using the third step. The first step implies that this algorithm is correct.

Before starting providing the proof details we note that there exist examples showing that the necessary conditions are not sufficient. Such an example will be provided in Section 6. We also note that the problem of finding κ whenever such a κ exists is a special case of the delay theorem mentioned in the introduction. When applied to LT, the delay theorem says that if a finite state automaton A recognizes a language in LT then this language must be κ-locally testable for a κ computable from A. The delay theorem was proved over words in [Str85] and can be used in order to decide whether a regular language is in LT as explained in [Boj07a]. We were not able to prove such a general theorem for trees.

3. Necessary conditions

In this section we exhibit necessary conditions for a regular language to be in LT. These conditions will play a crucial role in our decision algorithm. These conditions are expressed using the same formalism as the one used in [BS10] for characterizing LTT.

Guarded operations. Let t be a tree, and x, x' be two nodes of t such that x and x' are not related by the descendant relationship. The *horizontal swap* of t at nodes x and x' is the tree t' constructed from t by replacing $t|_x$ with $t|_{x'}$ and vice-versa, see Figure 1 (left). A horizontal swap is said to be *k-guarded* if x and x' have the same k-type.

Let t be a tree and x, y, z be three nodes of t such that x, y, z are not related by the descendant relationship and such that $t|_x = t|_y$. The *horizontal transfer* of t at x, y, z is the tree t' constructed from t by replacing $t|_y$ with a copy of $t|_z$, see Figure 1 (right). A horizontal transfer is k-guarded if x, y, z have the same k-type.

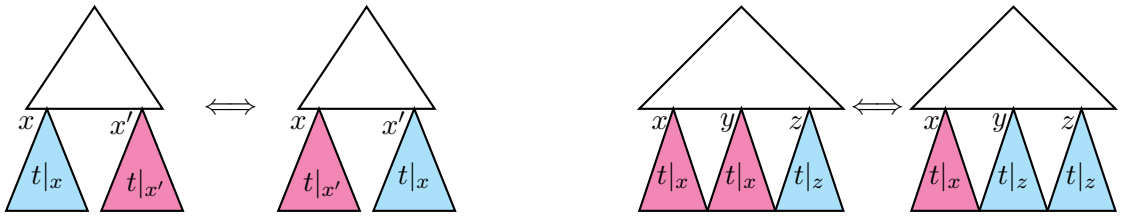


Figure 1: Horizontal Swap (left) and Horizontal Transfer (right)

Let t be a tree of root a, and x, y, z be three nodes of t such that y is a descendant of x and z is a descendant of y. The *vertical swap* of t at x, y, z is the tree t' constructed from t by swapping the context between x and y with the context between y and z, see Figure 2 (left). More formally let $C = t[a, x]$, $\Delta_1 = t[x, y]$, $\Delta_2 = t[y, z]$ and $T = t|_z$. We then have $t = C \cdot \Delta_1 \cdot \Delta_2 \cdot T$. The tree t' is defined as $t' = C \cdot \Delta_2 \cdot \Delta_1 \cdot T$. A vertical swap is *$k$-guarded* if x, y, z have the same k-type.

Let t be a tree of root a, and x, y, z be three nodes of t such that y is a descendant of x and z is a descendant of y such that $\Delta = t[x, y] = t[y, z]$. The *vertical stutter* of t at x, y, z is the tree t' constructed from t by removing the context between x and y, see Figure 2 (right). A vertical stutter is k-guarded if x, y, z have the same k-type.

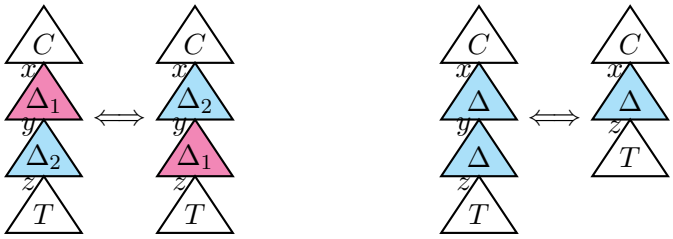


Figure 2: Vertical Swap (left) and Vertical Stutter (right)

Let L be a tree language and k be a number. If X is any of the four constructions above, horizontal or vertical swap, or vertical stutter or horizontal transfer, we say that L is *closed under k-guarded X* if for every tree t and every tree t' constructed from t using k-guarded X then t is in L iff t' is in L. Notice that being closed under k-guarded X implies being closed under k'-guarded X for $k' > k$. An important observation is that each of the k-guarded operation does not affect the set of $(k+1)$-types occurring in the trees.

If L is closed under all the k-guarded operations described above, we say that L is *k-tame*. A language is said to be *tame* if it is k-tame for some k.

The following simple result shows that tameness is a necessary condition for LT.

Proposition 3.1. *If L is in LT then L is tame.*

Proof. Assume L is in LT. Then there is a κ such that L is κ-locally testable. We show that L is κ-tame. This is a straightforward consequence of the fact that all the κ-guarded operations above preserve $(\kappa+1)$-types and hence preserve κ-types. □

A simple pumping argument shows that if L is tame then it is k-tame for k bounded by a polynomial in the size of the minimal deterministic bottom-up tree automaton recognizing L.

Proposition 3.2. *Given a regular language L and A the minimal deterministic bottom-up tree automaton recognizing L, we have L is tame iff L is k_0-tame for $k_0 = |A|^3 + 1$.*

Proof. We prove that if X is one of the four operations that defines tameness, then if L is closed under k-guarded X for $k > k_0$, then L is closed under k_0-guarded X. This will imply that if L is k-tame then it is k_0-tame.

Consider the case of k-guarded horizontal transfer and assume L is closed under k-guarded horizontal transfers. We show that L is closed under k_0-guarded horizontal transfers. Let t be a tree and x, y, z three nodes of t having the same k_0-type and not related by the descendant relation such that $t|_x = t|_y$. We need to show that replacing $t|_y$ by a copy of $t|_z$ does not affect membership in L.

We do this in three steps, first we transform t by pumping in parallel in the subtrees of x, y and z until x, y, z have the same k-type, then we use the closure of L under k-guarded horizontal transfer in order to replace $t|_y$ by a copy of $t|_z$, and finally we backtrack the initial pumping phase in order to recover the initial subtrees.

We let $t_1 = t|_x$ and $t_2 = t|_z$ and we assume for now on that $t_1 \neq t_2$. By *position* we denote a string w of $\{0,1\}^*$. A position w is *realized* in a tree t if there is a node x of t such that if $x_1, \cdots, x_n = x$ is the sequence of nodes in the path from the root of t to x then for all $i \leq n$ the i^{th} bit of w is zero if x_i is a left child and it is one if x_i is a right child. We order positions by first comparing their respective length and then using the lexicographical order.

By hypothesis t_1 and t_2 are identical up to depth at least k_0. Let w be the first position such that t_1 and t_2 differ at that position. That can be either because w is realized in t_1 but not in t_2, or vice versa, or w is realized in both trees but the labels of the corresponding nodes differ. We know that the length n of w is strictly greater than k_0. If $n > k$, we are done with the first phase. We assume now that $n \leq k$.

Consider the run r of A on t. The run assigns a state q to each node of t. From r we assign to each position $w' < w$ a pair of states (q, q') such that q is the state given by r at the corresponding node in t_1 while q' is the state given by r at the corresponding node in t_2. Because $n > k_0 > |A|^2$, there must be two prefixes w_1 and w_2 of w that were assigned the same pair of states. Consider the context $C_1 = t_1[v_1, v_2]$ where v and v' are the nodes of t_1 at position w_1 and w_2 and the context $C_2 = t_2[v'_1, v'_2]$ where v and v' are the nodes of t_2 at position w_1 and w_2. Without affecting membership in L, we can therefore at the same time duplicate C_1 in the two copies of t_1 rooted at x and y and C_2 in the copy of t_2 rooted at z.

Let t'_1 and t'_2 be the subtrees of the resulting tree, rooted respectively at x and z. The reader can verify that t'_1 and t'_2 now differ at a position strictly greater than w.

Performing this repeatedly, we eventually arrive at a situation where the subtree t'_1 rooted at x and y agree up to depth k with the subtree rooted at z. We can now apply

k-guarded horizontal transfer and replace one occurrence of t'_1 by a copy of t'_2. We can then replace t'_1 by t_1 and both copies of t'_2 by t_2 without affecting membership in L.

The other operations are done similarly. For the horizontal swap, we pump the subtrees at positions x and x' simultaneously, which is possible because $k_0 > |A|^2$. For vertical swap, we pump the subtrees at the positions x, y and z simultaneously, and that requires $k_0 > |A|^3$. Finally, for vertical stutter, we pump the subtrees at the positions x, y and z simultaneously, which again requires $k_0 > |A|^3$. □

Once k is fixed, a brute force algorithm can check whether L is k-tame or not. Indeed, as L is regular, when testing for closure under k-guarded X, it is enough to consider all relevant states and appropriate transition functions of the automata instead of all trees and all contexts. See for instance Lemma 12 and Lemma 13 in [BS10].

Therefore Proposition 3.2 implies that tameness is decidable. However for deciding LT we will only need the bound on k_0 given by the proposition.

4. Deciding LT

In this section we show that it is decidable whether a regular tree language is in LT. This is done by showing that if a regular language L is in LT then there is a κ computable from an automaton recognizing L such that L is in fact κ-locally testable. Recall that once this κ is computed the decision procedure simply enumerates all the finitely many κ-locally testable languages and tests whether L is one of them.

Assume L is in LT. By Proposition 3.1, L is tame. Even more, from Proposition 3.2, one can effectively compute a k such that L is k-tame. Hence Theorem 2.1 follows from the following proposition.

Proposition 4.1. *Assume L is a k-tame regular tree language then L is in LT iff L is κ-locally testable where κ is computable from k.*

Recall that for each k the number of k-types is finite. Let β_k be this number. Proposition 4.1 is an immediate consequence of the following proposition.

Proposition 4.2. *Let L be a k-tame regular tree language. Set $\kappa = \beta_k + k + 1$. Then for all $l > \kappa$ and any two trees t, t' if $t \simeq_\kappa t'$ then there exist two trees T, T' with*

1. *$t \in L$ iff $T \in L$*
2. *$t' \in L$ iff $T' \in L$*
3. *$T \simeq_l T'$*

Proof of Proposition 4.1 using Proposition 4.2. Assume L is k-tame and let κ be defined as in Proposition 4.2. We show that L is in LT iff L is κ-locally testable. Assume L is in LT. Then L is l-locally testable for some $l \in \mathbb{N}$. We show that L is actually κ-locally testable. For this it suffices to show that for any pair of trees t and t', if $t \simeq_\kappa t'$ then $t \in L$ iff $t' \in L$. Let T and T' be the trees constructed for l from t and t' by Proposition 4.2. We have $T \simeq_l T'$ and therefore $T \in L$ iff $T' \in L$. As we also have $t \in L$ iff $T \in L$ and $t' \in L$ iff $T' \in L$, the proposition is proved. □

Before proving Proposition 4.2 we need some extra terminology. A non-empty context C occurring in a tree t is a *loop of k-type τ* if the k-type of its root and the k-type of its port is τ. A non-empty context C occurring in a tree t is a k-loop if there is some k-type τ such that C is a loop of k-type τ. Given a context C we call the path from the root of C to its port the *principal path of C*. Finally, the result of the *insertion* of a k-loop C at a node x of a tree t is a tree T such that if $t = D \cdot t|_x$ then $T = D \cdot C \cdot t|_x$. Typically an insertion will occur only when the k-type of x is τ and C is a loop of k-type τ. In this case the k-types of the nodes initially from t and of the nodes of C are unchanged by this operation.

Proof of Proposition 4.2. Suppose that L is k-tame. We start by proving two lemmas that will be useful in the construction of T and T'. Essentially these lemmas show that even though being k-tame does not imply being $(k+1)$-locally testable (recall the remark after Theorem 2.1) some of the expected behavior of $(k+1)$-locally testable languages can still be derived from being k-tame. The first lemma shows that given a tree t, without affecting membership in L, we can replace a subtree of t containing only $(k+1)$-types occurring elsewhere in t by any other subtree satisfying this property and having the same k-type as root. The second lemma shows the same result for contexts by showing that a k-loop can be inserted in a tree t without affecting membership in L as soon as all the $(k+1)$-types of the k-loop were already present in t. After proving these lemmas we will see how to combine them for constructing T and T'.

Lemma 4.3. *Assume L is k-tame. Let $t = Ds$ be a tree where s is a subtree of t. Let s' be another tree such that the roots of s and s' have the same k-type.*

If $s \preccurlyeq_{k+1} D$ and $s' \preccurlyeq_{k+1} D$ then $Ds \in L$ iff $Ds' \in L$.

Proof. We start by proving a special case of the Lemma when s' is actually another subtree of t. We will use repeatedly this particular case in the proof.

Claim 4.4. *Assume L is k-tame. Let t be a tree and let x, y be two nodes of t not related by the descendant relationship and with the same k-type. We write $s = t|_x$, $s' = t|_y$ and C the context such that $t = Cs$. If $s \preccurlyeq_{k+1} C$ then $Cs \in L$ iff $Cs' \in L$.*

Proof. The proof is done by induction on the depth of s and makes crucial use of k-guarded horizontal transfer.

Assume first that s is of depth less than k. Since x and y have the same k-type, we have $s = s'$ and the result follows.

Assume now that s is of depth greater than k.

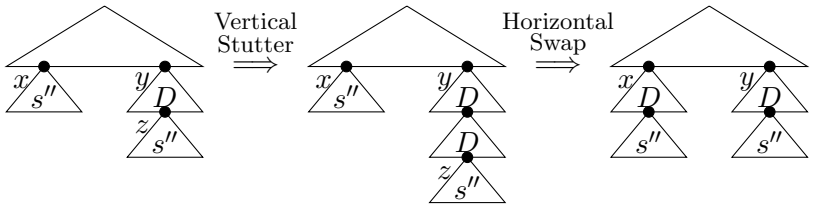
Let τ be the $(k+1)$-type of x. We assume that s is a tree of the form $a(s_1, s_2)$. Notice that the k-type of the roots of s_1 and s_2 are completely determined by τ. Since $s \preccurlyeq_{k+1} C$, there exists a node z in C of type τ. We write $s'' = t|_z$.

We consider several cases depending on the relationship between x, y and z. We first consider the case where x and z are not related by the descendant relationship, then we reduce the other cases to this case.

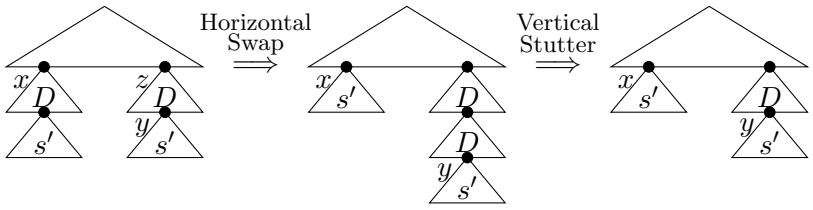
Assume that x and z are not related by the descendant relationship. Since s'' is of type τ, it is of the form $a(s_1'', s_2'')$ where the roots of s_1'' and s_2'' have the same k-type as respectively the roots of s_1 and s_2. By hypothesis all the $(k+1)$-types of s_1 and s_2 already appear in C and hence we can apply the induction hypothesis to replace s_1 by s_1'' and s_2 by s_2'' without affecting membership in L. Notice that the resulting tree is Cs'', that $t = Cs \in L$ iff $Cs'' \in L$, and that Cs'' contains two copies of the subtree s'', one at position x and one at position z. We now show that we can derive Cs' from Cs'' using k-guarded operations.

Since L is k-tame it will follow that that $Cs'' \in L$ iff $Cs' \in L$ and thus $Cs \in L$ iff $Cs' \in L$. Let $t'' = Cs''$ and we distinguish between three cases depending on the relationship between z and y in t'':

(1) If z is a descendant of y, let $D = t''[y, z]$ and notice that $s' = Ds''$. Since x, y and z have the same k-type, we use k-guarded vertical stutter to duplicate D and a k-guarded horizontal swap to move the new copy of D at position x (see the picture below). The resulting tree is Cs' as desired.



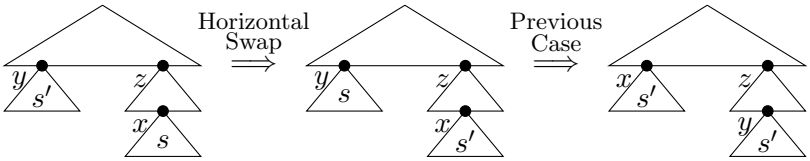
(2) If z is an ancestor of y, let $D = t''[z, y]$ and notice that $s'' = Ds'$. Since y and x have the same k-type, we use k-guarded horizontal swap followed by a k-guarded vertical stutter to delete the copy of D (see the picture below). The resulting tree is Cs' as desired.



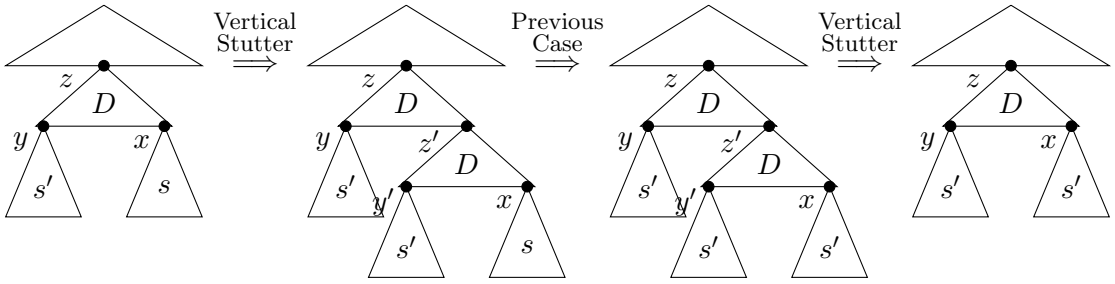
(3) If z and y are not related by the descendant relation, then x, y and z have the same k-type and $t''|_x = t''|_z$. We use k-guarded horizontal transfer to replace $t''|_x$ with $t''|_y$ as depicted below.

This concludes the case where x and z are not related by the descendant relationship in t. We are left with the case where x is a descendant of z (recall that z is outside s and therefore not a descendant of x). We reduce this problem to the previous case by considering two subcases:

- If y, z are not related by the descendant relationship, we use a k-guarded horizontal swap to replace s by s' and vice versa. This reverses the roles of x and y and as y and z are not related by the descendant relationship and position y now has $(k+1)$-type τ we can apply the previous case.



- If z is an ancestor of both x and y we use k-guarded vertical stutter to duplicate the context between z and x. This introduces a new node z' of type τ that is not related to y by the descendant relationship and we are back in the previous case.



□

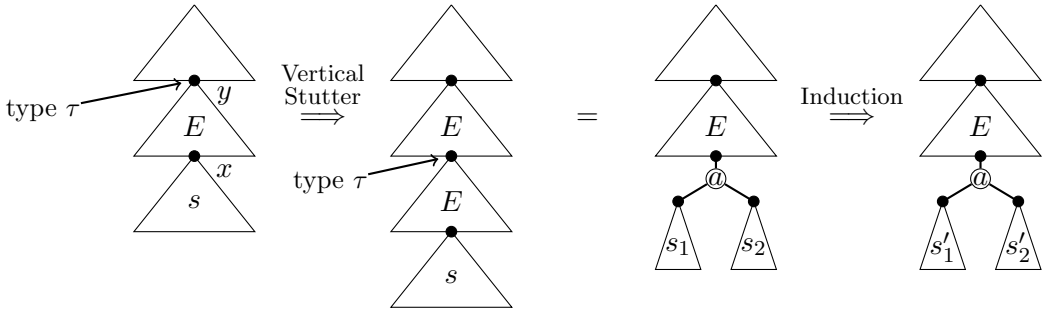
We now turn to the proof of Lemma 4.3. The proof is done by induction on the depth of s'. The idea is to replace s with s' node by node.

Assume first that s' is of depth less than k. Then because the k-type of the roots of s and s' are equal, we have $s = s'$ and the result follows.

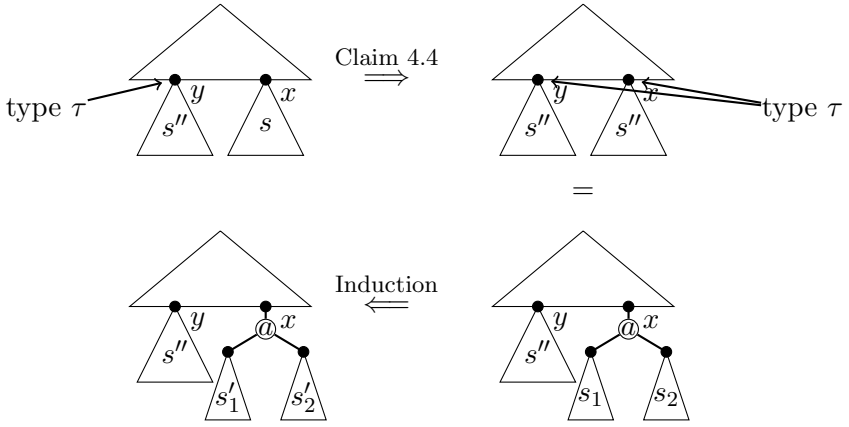
Assume now that s' is of depth greater than k.

Let x be the node of t corresponding to the root of s. Let τ be the $(k+1)$-type of the root of s'. We assume that s' is a tree of the form $a(s'_1, s'_2)$. Notice that the k-type of the roots of s'_1 and s'_2 are completely determined by τ. By hypothesis $s' \preccurlyeq_{k+1} D$, hence there exists a node y in D of type τ. We consider two cases depending on the relationship between x and y.

- If y is an ancestor of x, let E be $t[y, x]$ and notice that x and y have the same k-type. This case is depicted below. Hence applying a k-guarded vertical stutter we can duplicate E obtaining the tree DEs. Because L is k-tame, $DEs \in L$ iff $t = Ds \in L$. Now the root of Es in DEs is of type τ and therefore of the form $a(s_1, s_2)$ where the roots of s_1 and s_2 have the same k-type as respectively the roots of s'_1 and s'_2. By construction all the $(k+1)$-types of s_1 and s_2 already appear in D and hence we can apply the induction hypothesis to replace s_1 by s'_1 and s_2 by s'_2 without affecting membership in L. Altogether this gives the desired result.



- Assume now that x and y are not related by the descendant relationship. This case is depicted below. Let s'' be the subtree of Ds rooted at y. By hypothesis all the $(k+1)$-types of s are already present in D and the roots of s and s'' have the same k-type. Hence we can apply Claim 4.4 and we have $Ds \in L$ iff $Ds'' \in L$. Now the root of s'' is by construction of type τ. Hence s'' is of the form $a(s_1, s_2)$ where s_1 and s_2 have all their $(k+1)$-types appearing in D and their roots have the same k-type as respectively s'_1 and s'_2. Hence by induction s_1 can be replaced by s'_1 and s_2 by s'_2 without affecting membership in L. Altogether this gives the desired result.



□

We now prove a similar result for k-loops.

Lemma 4.5. *Assume L is k-tame. Let t be a tree and x a node of t of k-type τ. Let t' be another tree such that $t \simeq_{k+1} t'$ and C be a k-loop of type τ in t'. Consider the tree T constructed from t by inserting a copy of C at x. Then $t \in L$ iff $T \in L$.*

Proof. The proof is done in two steps. First we use the k-tame property of L to show that we can insert a k-loop C' at x in t such that the principal path of C is the same as the principal path of C'. By this we mean that there is a bijection from the principal path of C' to the principal path of C that preserves the child relation and $(k+1)$-types. In a second step we replace one by one the subtrees hanging from the principal path of C' with the corresponding subtrees in C.

First some terminology. Given two nodes y, y' of some tree T, we say that y' is a **l**-ancestor of y if y is a descendant of the left child of y'. Similarly we define **r**-ancestorship.

Consider the context C occurring in t'. Let $y_0, \cdots, y_n$ be the nodes of t' on the principal path of C and $\tau_0, \cdots, \tau_n$ be their respective $(k+1)$-type. For $0 \leq i < n$, set c_i to **l** if y_{i+1} is a left child of y_i and **r** otherwise.

From t we construct using k-guarded swaps and k-vertical stutters a tree t_1 such that there is a sequence of nodes $x_0, \cdots, x_n$ in t_1 with for all $0 \leq i < n$, x_i is of type τ_i and x_i is an c_i-ancestor of x_{i+1}. The tree t_1 is constructed by induction on n (note that this step do not require that C is a k-loop). If $n = 0$ then this is a consequence of $t \simeq_{k+1} t'$ that one can find in t a node of type τ_0. Consider now the case $n > 0$. By induction we have constructed from t a tree t'_1 such that $x_0, \cdots, x_{n-1}$ is an appropriate sequence in t'_1. By symmetry it is enough to consider the case where y_n is the left child of y_{n-1}. Because all k-guarded operations preserve $(k+1)$-types, we have $t \simeq_{k+1} t'_1$ and hence there is a node x' of t'_1 of type τ_n. If x_{n-1} is a **l**-ancestor of x' then we are done. Otherwise consider the

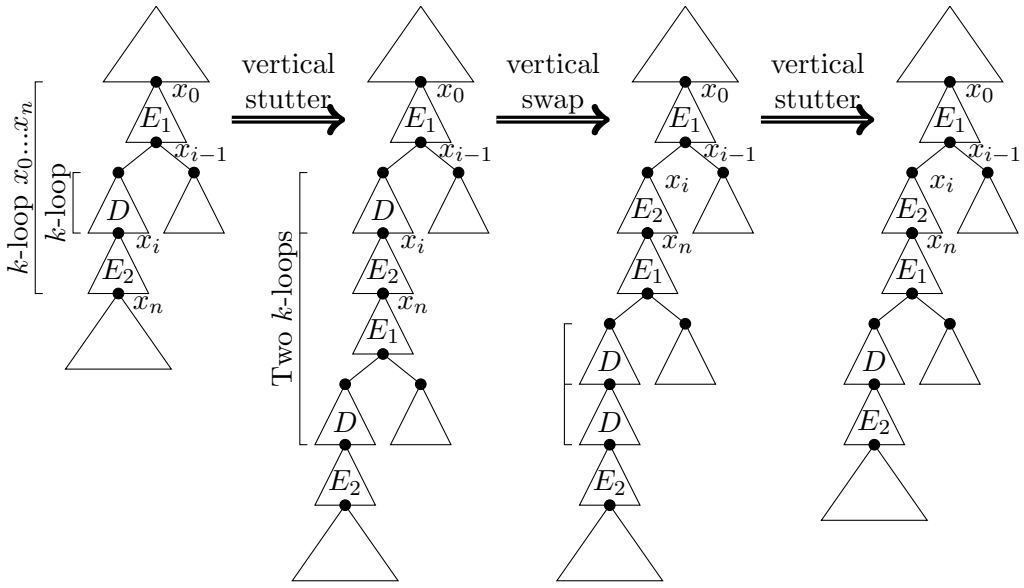


Figure 3: The construction of t_2, eliminating the context D between x_{i-1} and x_i

left child x'' of x_{n-1} and notice that because y_n is a child of y_{n-1} and x_{n-1} has the same $(k+1)$-type as y_{n-1} then x'', y_n and x' have the same k-type.

We know that x' is not a descendant of x''. There are two cases. If x' and x'' are not related by the descendant relationship then by k-guarded swaps we can replace the subtree rooted in x'' by the subtree rooted in x' and we are done. If x' is an ancestor of x'' then the context between x' and x'' is a k-loop and we can use k-guarded vertical stutter to duplicate it. This places a node having the same $(k+1)$-type as x' as the left child of x_{n-1} and we are done.

This concludes the construction of t_1. From t_1 we construct using k-guarded swaps and k-guarded vertical stutter a tree t_2 such that there is a path $x_0, \cdots, x_n$ in t_2 with x_i is of type τ_i for all $0 \leq i < n$.

Consider the sequence $x_0, \cdots, x_n$ obtained in t_1 from the previous step. Recall that the k-type of x_0 is the same as the k-type of x_n. Hence using k-guarded vertical stutter we can duplicate in t_1 the context rooted in x_0 and whose port is x_n. Let t_1' the resulting tree. We thus have two copies of the sequence $x_0, \cdots, x_n$ that we denote by the *top copy* and the *bottom copy*. Assume x_i is not a child of x_{i-1}. By symmetry it is enough to consider the case where x_{i-1} is a **l**-ancestor of x_i. Notice then that the context between the left child of x_{i-1} and x_i is a k-loop. Using k-guarded vertical swap (see Figure 3) we can move the top copy of this context next to its bottom copy. Using k-guarded vertical stutter this extra copy can be removed. We are left with an instance of the initial sequence in the bottom copy, while in the top one x_i is a child of x_{i-1}. This construction is depicted in figure 3.

Repeating this argument yields the desired tree t_2.

Consider now the context $C' = t_2[x_0, x_n]$. It is a loop of k-type τ. Let T' be the tree constructed from t by inserting C' at x.

Claim 4.6. *$T' \in L$ iff $t \in L$.*

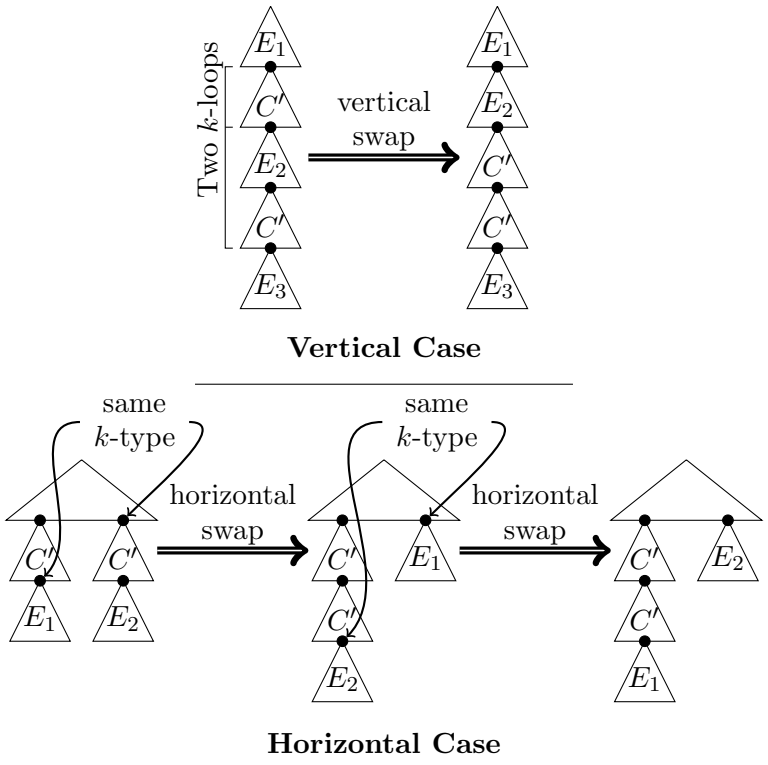


Figure 4: Bringing copies of the k-loop C' together in Claim 4.6

Proof. Consider the sequence of k-guarded swaps and k-guarded vertical stutter that was used in order to obtain t_2 from t. Because L is k-tame, $t \in L$ iff $t_2 \in L$.

We can easily identify the nodes of t with the nodes of T' outside of C'. Consider the same sequence of k-guarded operations applied to T'. Observe that this yields a tree T_2 corresponding to t_2 with possibly several extra copies of C'. As C' is a k-loop, each of the roots and the ports of these extra copies have the same k-type. Hence, using appropriate vertical k-swaps or appropriate horizontal k-swaps, depending on whether two copies are related or not by the descendant relation, they can be brought together. Two examples of such operation is given in Figure 4.

Then, using k-guarded vertical stutter all but one copy can be eliminated resulting in t_2. Hence $T' \in L$ iff $t_2 \in L$ and the claim is proved. See figure 5. □

It remains to show that $T' \in L$ iff $T \in L$. By construction of T' we have $C' \preccurlyeq_{k+1} t$. Consider now a node x_i in the principal path of C'. Let T_i be the subtree branching out the principal path of C at y_i and T'_i be the subtree branching out the principal path of C' at x_i. By construction x_i and y_i are of $(k+1)$-type τ_i. Therefore the roots of T_i and T'_i have the same k-type. Because $C' \preccurlyeq_{k+1} t$ all the $(k+1)$-types of T'_i already appear in the part of T' outside of C'. By hypothesis we also have $T_i \preccurlyeq_{k+1} t$. Hence we can apply Lemma 4.3 and replacing T'_i with T_i does not affect membership in L. A repeated use of that lemma eventually shows that $T' \in L$ iff $T \in L$. □

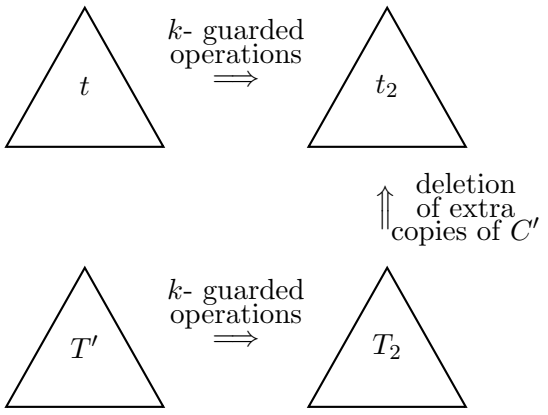


Figure 5: Relation with t_2

We return to the proof of Proposition 4.2. Recall that we have two trees t, t' such that $t \simeq_\kappa t'$ for $\kappa = \beta_k + k + 1$. For $l > \kappa$, we want to construct T, T' such that:

(1) $t \in L$ iff $T \in L$
(2) $t' \in L$ iff $T' \in L$
(3) $T \simeq_l T'$

Recall that the number of k-types is β_k. Therefore, by choice of κ, in every branch of a κ-type one can find at least one k-type that is repeated. This provides many k-loops that can be used using Lemma 4.5 for obtaining bigger types.

Take $l > \kappa$, we build T and T' from t and t' by inserting k-loops in t and t' without affecting their membership in L using Lemma 4.5.

Let $B = \{\tau_0, ..., \tau_n\}$ be the set of k-types τ such that there is a loop of k-type τ in t or in t'. For each $\tau \in B$ we fix a context C_τ as follows. Because $\tau \in B$ there is a context C in t or t' that is a loop of k-type τ. For each $\tau \in B$, we fix arbitrarily such a C and set C_τ as $\underbrace{C \cdot \ldots \cdot C}_{l}$, l concatenations of the context C. Notice that the path from the root of C_τ to its port is then bigger than l.

We now describe the construction of T from t. The construction of T' from t' is done similarly. The tree T is constructed by simultaneously inserting, for all $\tau \in B$, a copy of the context C_τ at all nodes of t of type τ.

We now show that T and T' have the desired properties.

The first and second properties, $t \in L$ iff $T \in L$ and $t' \in L$ iff $T' \in L$, essentially follow from Lemma 4.5. We only show that $t \in L$ iff $T \in L$, the second property is proved symmetrically. We view T as if it was constructed from t using a sequence of insertions of some context C_τ for $\tau \in B$. We write $s_0, ..., s_m$ the sequence of intermediate trees with $s_0 = t$ and $s_m = T$. We call C_i the context inserted to get s_{i+1} from s_i. We show by induction on i that (i) $s_i \simeq_{k+1} t$ and (ii) $s_i \in L$ iff $s_{i+1} \in L$. This will imply $t \in L$ iff $T \in L$ as desired. (i) is clear for $i = 0$. We show that for all i (i) implies (ii). Recall that C_i is the concatenation of l copies of a k-loop present either in t or in t'. We suppose without generality that the k-loop is present in t. Let s be the tree constructed from t by duplicating the k-loop l times. Hence s is a tree containing C_i and by construction $s \simeq_{k+1} t$. Because $t \simeq_\kappa t'$ with $\kappa > k + 1$ and $s_i \simeq_{k+1} t$ we have $s \simeq_{k+1} s_i$. By Lemma 4.5 this implies that

$s_{i+1} \in L$ iff $s_i \in L$. By construction we also have $s_{i+1} \simeq_{k+1} s_i$ and the induction step is proved.

We now show the third property:

Lemma 4.7. $T \simeq_l T'$

Proof. We need to show that $T \preccurlyeq_l T'$, $T' \preccurlyeq_l T$ and that the roots of T and T' have the same l-type. It will be convenient for proving this to view the nodes of T as the union of the nodes of t plus some nodes coming from the k-loops that were inserted. To do this more formally, if x is a node of t of k-type not in B, then x is identified with the corresponding node of T. If x is a node of t whose k-type is in B then x is identified in T with the port of the copy of C_τ that was inserted at node x. We start with the following claim.

Claim 4.8. *Take two nodes x in t and x' in t', such that x and x' have the same κ-type. Let y and y' be the corresponding nodes in T and T'. Then y and y' have the same l-type.*

Proof. Let ν the κ-type of x and x'. Consider a branch of ν of length κ. By the choice of κ we know that in this branch one can find two nodes z and z' with the same k-types τ, with z an ancestor of z' and such that the k-type τ of z is determined by ν (z is at distance $\geq k$ from the leaves of ν). Hence τ is in B. Note that because the k-type of z is included in ν, the presence of a node of type ν induces the presence of a node of type τ at the same relative position than z. Hence a copy of C_τ is inserted simultaneously at the same position relative to y and y' during the construction of T and T'. Because this is true for all branches of ν and because all C_τ have depth at least l, then y and y' have the same l-type. □

From claim 4.8 it follows that the roots of T and T' have the same l-type. By symmetry we only need to show that $T \preccurlyeq_l T'$. Let y be a node of T and μ be its l-type. We show that there exists $y' \in T'$ with type μ. We consider two cases:

- y is not a node of a loop inserted during the construction of T. Let x be the corresponding position in t and let ν be its κ-type. Since $t \simeq_\kappa t'$, there is a node x' of t' of type ν. Let y' be the node of T' corresponding to y'. By Claim 4.8 y and y' have the same l-type.
- y is a node inside a copy of C_τ inserted to construct T. Let x be the node of t where this loop was inserted. Let ν be the κ-type of x (the k-type of x is τ). Since $t \simeq_\kappa t'$, there is a node x' of t' of type ν. Since $\kappa > k$, x and x' have the same k-type, a copy of C_τ was also inserted in t' at position x' during the construction of T'. From Claim 4.8, x and x', when viewed as nodes of T and T' have the same l-type. Let y' be the node of T' in the copy of C_τ inserted at x' that corresponds to the position y. Since y and y' are ancestors of x and x' that have the same l-type, and since the context from y to x is the same as the context from y' to x', then y and y' must have the same l-type.

□

This concludes the proof of Proposition 4.2. □

5. Unranked trees

In this section we consider unranked unordered trees with labels in Σ. In such trees, each node may have an arbitrary number of children but no order is assumed on these children.

In particular even if a node has only two children we can not necessarily distinguish the left child from the right child.

Our goal is to adapt the result of the previous section and provide a decidable characterization of locally testable languages of unranked unordered trees.

In this section by *regular language* we mean definable in the logic MSO using only the child predicate and unary predicates for the labels of the nodes. There is also an equivalent automata model that we briefly describe next. A tree automaton A over unordered unranked trees consists essentially of a finite set of states $Q = \{q_1, \cdots, q_k\}$, an integer m denoted as the *counter threshold* in the sequel, and a transition function δ associating a unique state to any pair consisting of a label and a tuple $(q_1, \gamma_1) \cdots (q_k, \gamma_k)$ where $\gamma_i \in \{= i \mid i < m\} \cup \{\geq m\}$. The meaning is straightforward via bottom-up evaluation: A node of label **a** get assigned a state q if for all i, the number of its children, up to threshold m, that were assigned state q_i is as specified by δ. In the sequel we assume without loss of generality that all our tree automata are deterministic.

In the unranked tree case, there are several natural definitions of LT. Recall the definition of k-type: the k-type of a node x is the isomorphism type of the subtree induced by the descendants of x at distance at most k from x. With unranked trees this definition generates infinitely many k-types. We therefore introduce a more flexible notion of type, (k, l)-type, based on one extra parameter l restricting the horizontal information. It is defined by induction on k. Consider an unordered tree t and a node x of t. For $k = 0$, the (k, l)-type of x is just the label of x. For $k > 0$ the (k, l)-type of x is the label of x together with, for each $(k-1, l)$-type, the number, up to threshold l, of children of x of this type. The reader can verify that over binary trees, the $(k, 2)$-type and the k-type of x always coincide. As in the previous section we say that two trees are (k, l)-equivalent, and denote this using $\simeq_{(k,l)}$, if they have the same occurrences of (k, l)-types and their roots have the same (k, l)-type. We also use $t \preccurlyeq_{(k,l)} t'$ to denote the fact that all (k, l)-types of t also occur in t'.

Based on this new notion of type, we define two notions of locally testable languages. The most expressive one, denoted ALT (A for *Aperiodic*), is defined as follows. A language L is in (κ, λ)-ALT if it is a union of (κ, λ)-equivalence classes. A language L is in ALT if there is a κ and a λ such that L is in (κ, λ)-ALT.

The second one, denoted ILT in the sequel (I for *Idempotent*), assumes $\lambda = 1$: A language L is in ILT if there is a κ such that L is a union of $(\kappa, 1)$-equivalence classes.

The main result of this section is that we can decide membership for both ILT and ALT.

Theorem 5.1. *It is decidable whether a regular unranked unordered tree language is ILT. It is decidable whether a regular unranked unordered tree language is ALT.*

Tameness. The notion of k-tame is defined as in Section 3 using the same k-guarded operations requiring that the swapped nodes have identical k-type. We also define a notion of (k, l)-tame which corresponds to our new notion of (k, l)-type. Consider the four operations of tameness defined in Section 3. A horizontal swap is said to be (k, l)-guarded if x and x' have the same (k, l)-type, a horizontal transfer is (k, l)-guarded if x, y, z have the same (k, l)-type, a vertical swap is (k, l)-guarded if x, y, z have the same (k, l)-type and a vertical stutter is (k, l)-guarded if x, y, z have the same (k, l)-type. Let L be a regular unranked unordered tree language and let m be the counting threshold of the minimal automaton

recognizing L, we say that L is (k,l)-tame iff it is closed under (k,l)-guarded horizontal swap, horizontal transfer, vertical swap and vertical stutter and $l > m$ (we assume $l > m$ in order to make the statements of the results similar to those used in the binary setting). We first prove that over unordered trees being k-tame is the same as being (k,l)-tame.

Proposition 5.2. *Let L be an unordered unranked regular tree language, then for all integers k, L is k-tame iff there exists l such that L is (k,l)-tame. Furthermore, such an l can be computed from any automaton recognizing L.*

Proof. If there exists l such that L is (k,l)-tame then L is obviously k-tame. Suppose that L is k-tame, and let m be the counting threshold of the minimal automaton A recognizing L. We show that there exists l' such that L is closed under (k,l')-guarded operations. This implies the result as one can then take $l = max(m+1, l')$.

We need to show that L is closed under (k,l')-guarded vertical swap, vertical stutter, horizontal swap and horizontal transfer. The proof is similar to the proof of *Proposition 1* in [BS10]. We will use the following claim which is proved in [BS10] using a simple pumping argument:

Claim 5.3. [BS10] *For every tree automaton A there is a number l', computable from A, such that for every k if a tree t_1 is (k,l')-equivalent to a tree t_2, then there are trees t'_1, t'_2 with t'_1 and t'_2 k-equivalent such that A reaches the same state on t'_i as on t_i for $i = 1, 2$.*

We use this claim to prove that L is closed under horizontal transfer. Let l' be the number computed from A by Claim 5.3. We prove that L is closed under (k,l')-guarded horizontal transfer. Consider a tree t and three nodes x, y, z of t not related by the descendant relationship and such that $t|_x = t|_y$ and such that x, y and z have the same (k,l')-type. Let t' be the horizontal transfer of t at x, y, z. Let $t_1 = t|_x$ and $t_2 = t|_z$ and t'_1, t'_2 obtained from t_1, t_2 using Claim 5.3. Let s be the tree obtained from t by replacing $t|_x$ and $t|_y$ with t'_1 and $t|_z$ with t'_2, and let s' be the tree obtained from t' by replacing $t'|_x$ with t'_1 and $t'|_y$ and $t'|_z$ with t_2. From Claim 5.3 it follows that $t \in L$ iff $s \in L$ and $t' \in L$ iff $s' \in L$. Since L is k-tame, it is closed under k-guarded horizontal transfer, therefore we have $s \in L$ iff $s' \in L$, it follows that $t \in L$ iff $t' \in L$.

The closure under horizontal swap is proved using the same claim. The proofs for vertical swap and vertical stutter uses a claim similar to Claim 5.3 but for contexts: For every tree automaton A there is a number l computable from A such that for every k if the context C_1 is (k,l)-equivalent to the context C_2 (by this we mean that their roots have the same (k,l)-type), then there are contexts C'_1, C'_2 with C'_1 k-equivalent to C'_2 such that C'_i induces the same function on the states of A as C_i for $i = 1, 2$. □

From this lemma we know that a regular language over unranked unordered trees is tame iff it is k-tame for some k iff it is (k,l)-tame for some k, l. Moreover, as in the binary setting, if a regular language is tame then it is (k,l)-tame for some k and l computable from an automaton recognizing L. The bound on k can be obtained by a straightforward adaptation of Proposition 3.2. The bound on l then follows from Proposition 5.2. Hence we have:

Proposition 5.4. *Let L be a regular language and let A be its minimal deterministic bottom-up tree automaton, we have L is tame iff L is (k_0, l_0)-tame for $k_0 = |A|^3 + 1$ and some l_0 computable from A.*

5.1. **Decision of ALT.** We now turn to the proof of Theorem 5.1. We begin with the proof for *ALT* as both the decision procedure and its proof are obtained as in the case of binary trees. Assuming tameness we obtain a bound on κ and λ such that a language is in ALT iff it is in (κ, λ)-ALT. Once κ and λ are known, it is easy do decide if a language is (κ, λ)-ALT since the number of such languages is finite. The bounds on κ and λ are obtained following the same proof structure as in the binary cases, essentially replacing k-tame by (k, l)-tame, but with several technical modifications. Therefore, we only sketch the proofs below and only detail the new technical material. Our goal is to prove the following result.

Proposition 5.5. *Assume L is a (k, l)-tame regular tree language and let A be its minimal automaton. Then L is in ALT iff L is in (κ, λ)-ALT where κ and λ are computable from k, l and A.*

Notice that for each k, l the number of (k, l)-types is finite, let $\beta_{k,l}$ be this number. Proposition 5.5 is now a simple consequence of the following proposition.

Proposition 5.6. *Let L be a (k, l)-tame regular tree language and let A be the minimal automaton recognizing L. Set $\lambda = |A|l + 1$ and $\kappa = \beta_{k,l} + k + 1$. Then for all $\kappa' > \kappa$, all $\lambda' > \lambda$ and any two trees t, t' if $t \simeq_{(\kappa,\lambda)} t'$ then there exists two trees T, T' with*

(1) *$t \in L$ iff $T \in L$*
(2) *$t' \in L$ iff $T' \in L$*
(3) *$T \simeq_{(\kappa',\lambda')} T'$.*

Before proving Proposition 5.6 we adapt the extra terminology we used in the proof of Proposition 4.2 to the unranked setting. A non-empty context C occurring in a tree t is a *loop of (k, l)-type τ* if the (k, l)-type of its root and the (k, l)-type of its port is τ. A non-empty context C occurring in a tree t is a (k, l)-loop if there is some (k, l)-type τ such that C is a loop of (k, l)-type τ. Given a context C we call the path from the root of C to its port the *principal path of C*. Finally, the result of the *insertion* of a (k, l)-loop C at a node x of a tree t is a tree T such that if $t = D \cdot t|_x$ then $T = D \cdot C \cdot t|_x$. Typically an insertion will occur only when the (k, l)-type of x is τ and C is a loop of (k, l)-type τ. In this case the (k, l)-types of the nodes initially from t and of the nodes of C are unchanged by this operation.

Proof of Proposition 5.6. Suppose that L is (k, l)-tame. As we did for the proof of the binary case we first prove two lemmas that are crucial for the construction of T and T'. They show that subtrees can be replaced and contexts can be inserted as long as this does not change the $(k + 1, l)$-equivalence class of the tree. They are direct adaptations of the corresponding lemmas for the ranked setting: Lemmas 4.3 and 4.5. We start with subtrees.

Lemma 5.7. *Assume L is (k, l)-tame. Let $t = Ds$ be a tree where s is a subtree of t. Let s' be another tree such that the roots of s and s' have the same (k, l)-type.*

If $s \preccurlyeq_{(k+1,l)} D$ and $s' \preccurlyeq_{(k+1,l)} D$ then $Ds \in L$ iff $Ds' \in L$.

Proof sketch. As in the binary setting the proof is done by first proving a restricted version where s' is actually another subtree of t. Before doing that, we state a new claim, specific to the unranked setting, that will be useful later in the induction bases of our proofs. In the binary setting, two trees that had the same k-type at their root and were of depth smaller than k were equal. This obviously does not extend to unranked trees and (k, l)-types. However it is simple to see that equality can be replaced by indistinguishability by the minimal tree automaton recognizing L.

Claim 5.8. *Let A be a tree automaton and m be its counting threshold. Let t and t' be two trees of depth smaller than k and whose roots have the same (k,m)-type. Then t and t' evaluate to the same state of A.*

Proof. This is done by induction on k. If $k = 0$, t and t' are leaves, it follows from their $(0,m)$-type that $t = t'$.

Otherwise we know that t and t' have the same (k,m)-type at their root therefore they have the same root label. Let s and s' be two trees that are children of the root of t or of t' and have the same $(k-1,m)$-type at their root. The depth of s and s' is smaller than $k-1$, therefore by induction hypothesis s and s' evaluate to the same state of A. Now, because the roots of t and t' have the same (k,m)-type, for each $(k-1,m)$-type τ, they have the same number of children of type τ up to threshold m. From the previous remark this implies that for each state q of A, they have the same number of children in state q up to threshold m. It follows from the definition of A that t and t' evaluate to the same state of A. □

We are now ready to state and prove the lemma in the restricted case.

Claim 5.9. *Assume L is (k,l)-tame. Let t be a tree and let x, y be two nodes of t not related by the descendant relationship and with the same (k,l)-type. We write $s = t|_x$, $s' = t|_y$ and C the context such that $t = Cs$. If $s \preccurlyeq_{(k+1,l)} C$ then $Cs \in L$ iff $Cs' \in L$.*

Proof sketch. This proof only differs from its binary tree counterpart Claim 4.4 in the details of the induction step. It is done by induction on the depth of s.

Assume first that s is of depth less than k. Since x and y have the same (k,l)-type and since $l \geq m$ it follows from Claim 5.8 that s and s' evaluate to the same state on the automaton A recognizing L. Hence we can replace s with s' without affecting membership in L.

Assume now that s is of depth greater than k.

Let τ be the $(k+1,l)$-type of x. We write $s_1, ..., s_n$ for the children of s and a the label of its root. Since $s \preccurlyeq_{(k+1,l)} C$, there exists a node z in C of type τ. We write $s'' = t|_z$.

We now do a case analysis depending on the descendant relationships between x, y and z. As for binary trees, all cases reduce to the case when x and z are not related by the descendant relationship by simple (k,l)-tameness operations. Therefore we only consider this case here.

Assume that x and z are not related by the descendant relationship. We show only that $Cs \in L$ iff $Cs'' \in L$. The proof that $Cs' \in L$ iff $Cs'' \in L$ is then done exactly as for binary trees.

Since x and z are of same $(k+1,l)$-type τ, the roots of s' and s'' have the same label a. Let $s''_1, \ldots, s''_{n'}$ be the children of the root of s''. As in the binary case we want to replace the trees $s_1, \ldots, s_n$ with these children by induction since the depth of the trees $s_1, \ldots, s_n$ is smaller than the depth of s. Unfortunately for each (k,l)-type τ_i, the number of trees whose root has type τ_i among the children of x and among the children of z might not be the same. However we know that in this case both numbers are greater than l. We overcome this difficulty in two steps, first we modify the children of x, without affecting membership in L, so that if s_i and s_j have the same (k,l)-type then $s_i = s_j$, then we use the fact that $l > m$ in order to delete or duplicate children of x until for each (k,l)-type τ_i the number of trees of root of type τ_i among the children of x and among the children of z is the same. By definition of A, this does not affect membership in L. Finally we replace the s_i by the s''_i by induction as in the binary case.

For the first step notice that any of the s_i is by definition of depth smaller than s therefore by the induction hypothesis we can replace it with any of its siblings having the same (k,l)-type at its root without affecting membership in L. □

We now turn to the proof of Lemma 5.7 in its general statement. The proof is done by induction on the depth of s'. The idea is to replace s with s' node by node.

Assume first that s' is of depth smaller than k. Then because the (k,l)-types of the roots of s and s' are the same we are in the hypothesis of Claim 5.8 and it follows that s and s' evaluate to the same state of A. The result follows.

Assume now that s' is of depth greater than k.

Let x be the node of t corresponding to the root of s. Let τ be the $(k+1,l)$-type of the root of s'. In the binary tree case we used a sequence of tame operations to reduce the problem to the case where x has $(k+1,l)$-type τ. Using the same operations we can also reduce the problem to this case in the unranked setting. Then we use the induction hypothesis to replace the children of x by the children of the root of s'. As in the proof of Claim 5.9, the problem is that the number of children might not match but this is solved exactly as in the proof of Claim 5.9. □

As in the binary tree case, we now prove a result similar to Lemma 5.9 but for (k,l)-loops.

Lemma 5.10. *Assume L is (k,l)-tame. Let t be a tree and x a node of t of (k,l)-type τ. Let t' be another tree such that $t \simeq_{(k+1,l)} t'$ and C be a (k,l)-loop of type τ in t'. Consider the tree T constructed from t by inserting a copy of C at x. Then $t \in L$ iff $T \in L$.*

Proof sketch. The proof is done using the same structure as Lemma 4.5 for the binary case. First we use the (k,l)-tame property of L to show that we can insert a (k,l)-loop C' at x in t such that the principal path of C is the same as the principal path of C'. By this we mean that there is a bijection from the principal path of C' to the principal path of C that preserves the child relation and $(k+1,l)$-types. In a second step we replace one by one the subtrees hanging from the principal path of C' with the corresponding subtrees in C.

Let T' be the tree resulting from inserting C' at position x. We do not detail the first step as it is done using exactly the same sequence of tame operations we used for this step in the proof of Lemma 4.5. This yields: $t \in L$ iff $T' \in L$. We turn to the second step showing that $T' \in L$ iff $T \in L$.

By construction of T' we have $C' \preccurlyeq_{(k+1,l)} t$. Consider now a node x'_i in the principal path of C' and x_i the corresponding node in C. As in the binary tree case we replace the subtrees branching out of the principal path of C' with the corresponding trees branching out of the principal path of C using Lemma 5.7. As in the previous proof, the problem is that the numbers of children might not match. This is solved exactly as in the proof of Lemma 5.7. □

We now turn to the construction of T and T' and prove Proposition 5.6.

The construction is similar to the one we did in the binary tree case. We insert (k,l)-loops in t and t' using Lemma 5.10 for obtaining bigger types. However inserting loops only affects the depth of the types. Therefore we need to do extra work in order to also increase the width of the types.

Assuming $t \simeq_{(k,l)} t'$ we first construct two intermediate trees T_1 and T'_1 that have the following properties:

- $t \in L$ iff $T_1 \in L$

- $t' \in L$ iff $T_1' \in L$
- $T_1 \simeq_{(\kappa',\lambda)} T_1'$

This construction is the same as in the binary tree setting so we only briefly describe it. Let $B = \{\tau_0, ..., \tau_n\}$ be the set of (k,l)-types τ such that there is a loop of (k,l)-type τ in t or in t'. For each $\tau \in B$ we fix a context C_τ as follows. Because $\tau \in B$ there is a context C in T_1 or T_1' that is a loop of (k,l)-type τ. For each $\tau \in B$, we fix arbitrarily such a C and set C_τ as $\underbrace{C \cdot \ldots \cdot C}_{\kappa'}$, κ' concatenations of the context C. Notice that the path from the root of C_τ to its port is then bigger than κ'.

T_1 is constructed from t as follows (the construction of T_1' from t' is done similarly). The tree T_1 is constructed by simultaneously inserting, for all $\tau \in B$, a copy of the context C_τ at all nodes of t of type τ. By Lemma 5.10 it follows that $t \in L$ iff $T_1 \in L$ and $t' \in L$ iff $T_1' \in L$. Using the same proof as that of Proposition 4.2 for the binary tree setting, we obtain $T_1 \simeq_{(\kappa',\lambda)} T_1'$.

We now describe the construction of T from T_1, the construction of T' from T_1' is done similarly. It will be convenient for us to view the nodes of T_1 as the union of the nodes of t plus some extra nodes coming from the loops that were inserted.

Let n be the maximum arity of a node of T_1 or of T_1'. We duplicate subtrees in T_1 and T_1' as follows. Let x be a node of T_1, that is not in a loop we inserted when constructing T_1 from t. For each $(\kappa'-1,\lambda)$-type τ, if x has more than λ children of type τ we duplicate one of the corresponding subtrees until x has exactly n children of type τ in total. This is possible without affecting membership in L because $\lambda > m|A|$. Indeed, because $\lambda > m|A|$, for at least one state q of A, there exists more than m subtrees of x of type τ for which A assigns that state q at their root, and by definition of A any of these subtrees can be duplicated without affecting membership in L. The tree T is constructed from T_1 by repeating this operation for any node x of T_1 coming from t. By construction we have $T_1 \in L$ iff $T \in L$ and therefore $t \in L$ iff $T \in L$. The same construction starting from T_1' yields a tree T' such that $t' \in L$ iff $T' \in L$.

We now show that $T \simeq_{\kappa'} T'$, it follows that $T \simeq_{(\kappa',\lambda')} T'$ and this concludes the proof.

Lemma 5.11. $T \simeq_{\kappa'} T'$

Proof. We need to show that $T \preccurlyeq_{\kappa'} T'$, $T' \preccurlyeq_{\kappa'} T$ and that the roots of T and T' have the same κ'-type.

Recall that in T_1 we distinguished between two kinds of nodes, those coming from t and those coming from the loops that were inserted during the construction of T_1 from t. We make the same distinction in T by assuming that a node generated after a duplication gets the same kind as its original copy.

Recall the definition of B and of C_τ for $\tau \in B$ that was used for defining T_1 and T_1' from t and t'.

As for the binary tree case it suffices to show that for any node of T coming from t there is a node of T' coming from t' and having the same κ'-type. Hence the result follows from the claim below that is an adaptation of Claim 4.8.

Claim 5.12. *Take two nodes x in t and x' in t', such that x and x' have the same (κ,λ)-type. Let z and z' be the corresponding nodes in T and T'. Then z and z' have the same κ'-type.*

Proof. Let x and x' be two nodes of t and t' with the same (κ, λ)-type. Let x_1 and x'_1 be the corresponding nodes in T_1 and T'_1. The same proof as Claim 4.8 for the binary tree case shows that x_1 and x'_1 have the same (κ', λ)-type.

Let y be a child of x. Let y_1 be the node corresponding to y in T_1. Notice now that the (κ', λ)-type of y_1 in T_1 is completely determined by the $(\kappa - 1, \lambda)$-type ν of y in t. Indeed, by choice of κ, during the construction of T_1, a loop of type $\tau \in B$ will be inserted between y and any descendant of y at distance at most $\beta_{(k,l)} - 1$ from y. As $\kappa > \beta_{(k,l)} + k$, the relative positions below y where such a C_τ is inserted can be read from ν. As the depth of any C_τ is greater than κ', from ν we can compute exactly the descendants of y_1 in T_1 up to depth κ'. Hence ν determines the (κ', λ)-type of y_1.

It follows that two children of x_1 or of x'_1 have the same (κ', λ)-types iff they had the same $(\kappa - 1, \lambda)$-types in t or in t'.

We now construct an isomorphism between the κ'-type of z and the one of z'. Let d be the maximal distance between z and a node that is a descendant of z where a loop was inserted during the construction of T from t. We construct our isomorphism by induction on d.

If $d = 0$ then the (k, l)-type of z is in B and as z and z' have the same (κ', λ)-type with $\kappa' > k$, the (k, l)-type of z' is the same as the one of z'. Therefore the subtrees rooted at z and z' are equal up to depth κ' as they all start with a copy of C_τ and we are done.

Otherwise, as z and z' have the same (κ', λ)-type their roots must have the same labels. Consider now a $(\kappa' - 1, \lambda)$-type μ. By construction of T and T', z and z' must have the same number of occurrences of children of type μ. Indeed from the type these numbers must match if one of them is smaller than λ and by construction they are equal to n otherwise. Hence we have a bijection from the children of z of type μ and the children of z' of type μ. From the text above we know that the (κ', λ)-type of these nodes is determined by the $(\kappa - 1, \lambda)$-type of their copy in t or in t'. Because x and x' have the same (κ, λ)-type, the corresponding $(\kappa - 1, \lambda)$-types are all equal and hence all the nodes of type μ actually have the same (κ', λ)-type. By induction they are isomorphic up to depth κ' and we are done. □

From Claim 5.11 the lemma follows as in the proof of Lemma 4.7 for binary trees. □

This concludes the proof of Proposition 5.6. □

5.2. **Decision of ILT.** In the idempotent case we can completely characterize ILT using closure properties. We show that membership in ILT corresponds to tameness together with an extra closure property denoted *horizontal stutter* reflecting the idempotent behavior. A tree language L is closed under horizontal stutter iff for any tree t and any node x of t, replacing $t|_x$ with two copies of $t|_x$ does not affect membership in L. Theorem 5.1 for ILT is a consequence of the following theorem.

Theorem 5.13. *A regular unordered tree language is in ILT iff it is tame and closed under horizontal stutter.*

Proof. It is simple to see that tameness and closure under horizontal stutter are necessary conditions. We prove that they are sufficient. Take a regular tree language L and suppose that L is tame and closed under horizontal stutter. Then there exists k and l such that L is (k, l)-tame. We show that if $t \simeq_{(k+1,1)} t'$ then $t \in L$ iff $t' \in L$. It follows that L is in ILT. We first show a simple lemma stating that if two trees contain the same $(k + 1, 1)$-types,

then we can pump them without affecting membership in L into two trees that contain the same $(k+1,l)$-types.

Lemma 5.14. *Let L closed under horizontal stutter and let s and s' two trees such that $s \simeq_{(k+1,1)} s'$. Then there exist two trees S and S' such that:*

- $s \in L$ *iff* $S \in L$.
- $s' \in L$ *iff* $S' \in L$.
- $S \simeq_{(k+1,l)} S'$

Proof. S is constructed from s via a bottom-up procedure. Let x be a node of s. For each subtree rooted at a child of x, we duplicate it l times using horizontal stutter. This does not affect membership in L. After performing this for all nodes x of s we obtain a tree S with the desired properties. □

Let T and T' be constructed from t and t' using Lemma 5.14. Let $T_1, \ldots, T_n$ the children of the root of T and $T'_1, \ldots, T'_{n'}$ the children of the root of T'. Let T'' be the tree whose root is the same as T and T' and whose children is the sequence of trees $T_1, \ldots, T_n, T'_1, \ldots, T'_{n'}$. We show that $T'' \in L$ iff $T \in L$ and $T'' \in L$ iff $T' \in L$. It will follow that $T \in L$ iff $T' \in L$ and by Lemma 5.14 that $t \in L$ iff $t' \in L$ which ends the proof.

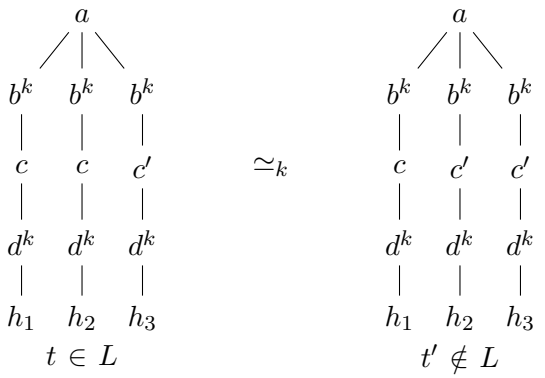
To show that $T'' \in L$ iff $T \in L$ we use horizontal stutter and Lemma 5.7. As the roots of T and T' have the same $(k+1,l)$-type, for each T'_i, there exists a tree T_j such that its root has the same (k,l)-type as T'_i. Fix such a pair (i,j). Let S be the tree obtained by duplicating T_j in T. By closure under horizontal stutter $T \in L$ iff $S \in L$. But now $S = DT_j$ for some context D such that $T \preccurlyeq_{(k+1,l)} D$. Altogether we have that: the roots of T'_i and T_j have the same (k,l)-type (by choice if i and j), $T'_i \preccurlyeq_{(k+1,l)} D$ (as $T'_i \preccurlyeq_{(k+1,l)} T'$ and $T \simeq_{(k+1,l)} T'$) and $T_j \preccurlyeq_{(k+1,l)} D$ (as T_j is part of T hence of D). We can therefore apply Lemma 5.7 and $DT'_i \in L$ iff $DT_j \in L$.

Repeating this argument for all i eventually yields the tree T''. This proves that $T'' \in L$ iff $T \in L$. By symmetry we also have $T'' \in L$ iff $T' \in L$ which concludes the proof. □

6. Tameness is not sufficient

Over strings tameness characterizes exactly LT as vertical swap and vertical stutter are exactly the extensions to trees of the known equations for LT (recall Section 2). Over trees this is no longer the case. In this section we provide an example of a language that is tame but not LT. For simplifying the presentation we assume that nodes may have between 0 and three children; this can easily be turned into a binary tree language. All trees in our language L have the same structure consisting of a root of label **a** from which exactly three sequences of nodes with only one child (strings) are attached. The trees in L have therefore exactly three leaves, and those must have three distinct labels among $\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$. The labels of two of the branches, not including the root and the leaf, must form a sequence in the language $\mathbf{b}^*\mathbf{cd}^*$. The third branch must form a sequence in the language $\mathbf{b}^*\mathbf{c'd}^*$. We assume that **b**, **c**, **c'** and **d** are distinct labels. Note that the language does not specify which leaf label among $\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$ is attached to the branch containing **c'**.

The reader can verify that L is 1-tame. We show that L is not in LT. For all integer k, the two trees t and t' depicted below are such that $t \in L$, $t' \notin L$, while $t \simeq_k t'$.



7. Discussion

We have shown a decidable characterization for the class of locally testable regular tree languages both for ranked trees and unranked unordered trees.

Complexity. The decision procedure for deciding membership in LT as described in this paper requires a time which is a tower of several exponentials in the size of the deterministic minimal automaton recognizing L. This is most likely not optimal. In comparison, over strings, membership in LT can be performed in polynomial time [Pin05]. Essentially our procedure requires two steps. The first step shows that if a regular language L is in LT then it is κ-locally testable for some κ computable from the minimal deterministic automaton A recognizing L. The κ obtained in Proposition 4.1 is doubly exponential in the size of A. In comparison, over strings, this κ can be shown to be polynomial. For trees we did not manage to get a smaller κ but we have no example where even one exponential would be necessary.

Our second step tests whether L is κ-locally testable once κ is fixed. This was easy to do using a brute force algorithm requiring several exponentials in κ. It is likely that this can be optimized but we didn't investigate this direction.

However for unranked unordered trees we have seen in Theorem 5.13 that in the case of ILT it is enough to test for tameness. The naive procedure for deciding tameness is exponential in the size of A. But the techniques presented in [BS10] for the case of LTT, easily extend to the closure properties of tameness, and provide an algorithm running in time polynomial in the size of A. Hence membership in ILT can be tested in time polynomial in the size of the minimal deterministic bottom-up tree automaton recognizing the language.

Logical characterization. There is a logical characterization of languages that are locally testable. It corresponds to those languages definable by formulas containing the temporal predicates **G** and **X** where **G** stands for "everywhere in the tree" while **X** stands for "child". In the binary tree case, we also require two predicates distinguishing the left child from the right child. In the unranked unordered setting the logic above is closed under bisimulation and therefore corresponds to ILT. This shows that in a sense ILT is the natural extension of LT to the unranked setting.

Open problem. It would be interesting to obtain a different characterization of LT based on a finite number of conditions such as the ones characterizing tameness. This would be a more satisfying result and would most likely provide a more efficient algorithm for deciding LT.

References

[Boj07a] M. Bojańczyk. A new algorithm for testing if a regular language is locally threshold testable. *Information Processing Letter*, 104(3):91–94, 2007.

[Boj07b] M. Bojańczyk. Two-way unary temporal logic over trees. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 121–130, 2007.

[BP89] D. Beauquier and J-E. Pin. Factors of words. In *Intl. Coll. on Automata, Languages and Programming (ICALP)*, pages 63–79, 1989.

[BS73] J. A. Brzozowski and I. Simon. Characterizations of locally testable languages. *Discrete Mathematics*, 4:243–271, 1973.

[BS08] M. Bojańczyk and L. Segoufin. Tree languages defined in first-order logic with one quantifier alternation. In *Intl. Coll. on Automata, Languages and Programming (ICALP)*, 2008.

[BS10] M. Benedikt and L. Segoufin. Regular languages definable in FO and FOmod. *ACM Trans. Of Computational Logic*, 11(1), 2010.

[BSS08] M. Bojańczyk, L. Segoufin, and H. Straubing. Piecewise testable tree languages. In *IEEE Symposium on Logic in Computer Science (LICS)*, 2008.

[BW06] M. Bojańczyk and I. Walukiewicz. Characterizing EF and EX tree logics. *Theoretical Computer Science*, 358(255-272), 2006.

[BW07] M. Bojańczyk and I. Walukiewicz. Forest algebras. In *Automata and Logic: History and Perspectives*, pages 107 – 132. Amsterdam University Press, 2007.

[CGJ+07] Hubert Comon, Max Dauchet Remy Gilleron, Florent Jacquemard, Christof Löding, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available electronically at `http://tata.gforge.inria.fr/`, 2007.

[EW05] Z. Esik and P. Weil. Algebraic characterization of regular tree languages. *Theoretical Computer Science*, 340:291–321, 2005.

[McN74] R. McNaughton. Algebraic decision procedures for local testability. *Mathematical System Theory*, 8(1):60–76, 1974.

[Pin05] J-E. Pin. The expressive power of existential first order sentences of Büchi's sequential μ calculus. *Discrete Mathematics*, 291:155–174, 2005.

[Pla08] T. Place. Characterization of logics over ranked tree languages. In *Conference on Computer Science Logic (CSL)*, pages 401–415, 2008.

[Str85] H. Straubing. Finite semigroup varieties of the form V∗D. *J. of Pure and Applied Algebra*, 36:53–94, 1985.

[Til87] B. Tilson. Categories as algebra: an essential ingredient in the theory of monoids. *J. of Pure and Applied Algebra*, 48:83–198, 1987.

[TW85] D. Thérien and A. Weiss. Graph congruences and wreath products. *J. Pure and Applied Algebra*, 36:205–215, 1985.

[Wil96] T. Wilke. An algebraic characterization of frontier testable tree languages. *Theoretical Computer Science*, 154(1):85–106, 1996.