A decidable characterization of locally testable tree languages

Thomas Place and Luc Segoufin

INRIA and LSV at ENS Cachan

Abstract. A regular tree language L is locally testable if the membership of a tree into L depends only on the presence or absence of some neighborhoods in the tree. In this paper we show that it is decidable whether a regular tree language is locally testable.

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. For instance, we do not possess yet a decidable characterization of the tree languages definable in $\mathrm{FO}(\leq)$, 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 $\mathrm{FO}(\leq)$ known as *Locally Testable* (LT). A language is in LT if its membership 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 obtained 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.

Decidable characterizations are usually obtained by exhibiting a set of closure properties that holds exactly for the languages in the class under investigation. It is therefore necessary to have a formalism for expressing these properties. This formalism must also come with some tools for proving that the properties do characterize the class, typically with induction mechanisms, but also for proving the decidability of those properties.

Over words one formalism turned out to be successful for characterizing many class of regular languages. The closure properties are expressed as identities on the syntactic monoid of the regular language. The syntactic monoid of a regular language being the transition monoid of its minimal deterministic automata. For instance the class of languages definable in $\mathrm{FO}(\leq)$ is characterized by the fact that their syntactic monoid is aperiodic. The later 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 monoid [BS73,McN74,BP89,TW85].

Over trees the situation is more complex and right now there is no formalism that can easily express all the closure properties of the classes for which we have a decidable characterization. The most successful formalism is certainly forest algebra [BW07]. For instance, forest algebra was 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 algebra for characterizing the class LTT over trees and a different formalism was used for obtaining a decidable characterization for this class [BS09].

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

There is another technique that worked on words for deciding the class LT. It is based on the "delay theorem" [Str85,Til87] for computing the expected size of the neighborhoods: Given a regular language L, a number k can be computed 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 and decidable 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 expected 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 exists several formalisms that were 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.

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 decidable and necessary for being in LT. In Section 4 we show that for the languages satisfying the necessary conditions the expected 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. Due to space limitations several proofs are missing.

2 Notations and preliminaries

We first prove our result for 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. A *language* is a set of trees. We use standard notations for trees. For instance by the *descendant* (resp. ancestor) relation we mean the reflexive transitive closure of the child (resp. inverse of child) relation.

Given a tree t and a node x of t the subtree of t rooted at x, consisting of all the nodes of t which are descendants 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 concatenating 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 $C = t[x, y]$ is defined from $t_1 = t|_x$ by replacing $t_1|_y$ by a port. In this case we say that C is a context *occurring* in 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 in t is the (isomorphism type of the) restriction of $t|_x$ to the set of nodes of t at distance at most k from x. A k-type τ *occurs* in a tree t if there exists a node in t of k-type τ. If C is a context occurring in a tree t then the k-type of a node of C is the k-type of that node in t. Notice that the k-type of a node of C depends on the surrouding tree t, in particular the port of C has a k-type.

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 a context occurring in some tree 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 finite index.

A language L is said to be κ-locally testable (is in LT_κ) 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 words this says that 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.

The problem. We want an algorithm deciding if a given regular language is in LT. If complexity does not matter, we can assume that the given language L is given as a MSO formula. Another option would be to start with a bottom-up tree automata for L or, even better, the minimal deterministic bottom-up tree automata that recognizes L. The main difficulty is to compute a bound on κ, the size of the neighborhood, whenever such a κ exists.

The string 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 string case independently by [BS73,McN74]. It is based on a characterization of the syntactic semigroup of the language by means of 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 on 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 1 *It is decidable whether a regular tree language is in LT.*

Our strategy for proving Theorem 1 is as follows. In a first step we provide necessary, and decidable, conditions for a language to be in LT. In a second step we show that if a language verifies those conditions then we can compute a κ such that it is in LT iff it is in LT_κ. Finally we show that once κ is fixed, it is decidable whether a regular language is a finite union of classes of $\simeq_\kappa$.

Before starting providing the proof details we note that there exists examples showing that the necessary conditions are not sufficient, see the end of Section 3. This is not immediate to see and goes beyond the scope of this paper. We also note that the problem of finding κ whenever such a κ exists is a special case of the *delay-theorem*. In the case of LT, the delay theorem says that if a finite state automaton A recognizes a language in LT then this language must be in LT_κ for a κ computable from A. This theorem holds over strings [Str85,Til87] 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. Our second step can be seen as a particular case of the delay theorem for languages satisfying the conditions provided by the first step.

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 [BS09] 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.

Fig. 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). 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.

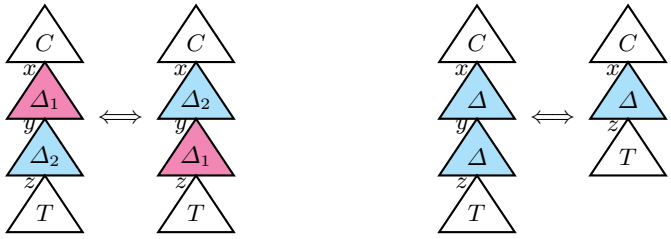

Fig. 2. Vertical Swap (left) and Vertical Stutter (right)

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 operations preserves $(k+1)$-types.

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 1 *If L is in LT then L is tame.*

We now turn to the decision procedure for testing tameness. If k is fixed, it is simple to check whether L is k-tame, see for instance [BS09]. The following proposition shows that k can be assumed to be bounded by a simple function of the size of any bottom-up tree automata recognizing L. It can be shown using a pumping argument.

Proposition 2 *Given a regular language L, it is decidable whether L is tame. Moreover, if this is the case, a k such that L is k-tame can be effectively computed.*

Example 1 *Over strings tameness characterizes exactly LT as vertical swap and vertical stutter are exactly the extensions to trees of the equations given in*

Section 2 for LT. Over trees this is no longer the case as the following example shows. For simplifying the presentation we assume that nodes may have between 0 to three children. 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* $\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$*. One branch, excepted for its leaf, must be of the form* b^*cd^**, another one of the form* b^*cd^* *and the last one of the form* $b^*c'd^*$*, where* **b**, **c**, **c'** *and* **d** *are distinct labels. The reader can verify that L is tame. It is not in LT because replacing exactly one node of label* **c** *by a node of label* **c'** *in a tree with long branches yields a tree with the same neighborhood but not in L.*

4 Deciding LT for tame languages

In this section we show that it is decidable whether a regular tree language is in LT. In view of Proposition 1 and Proposition 2 it is sufficient to show that it is decidable whether a tame regular tree language is in LT. Hence Theorem 1 follows from the following proposition.

Proposition 3 *Assume L is a tame regular tree language. Then it is decidable whether there exists a* κ *such that L is in* LT_κ*.*

Proof. Assume L is tame. Then from Proposition 2 one can effectively compute a k such that L is k-tame. The proof of the proposition is then divided in two steps. The first one shows that for k-tame languages, if such a κ exists then κ is at most exponential in k. The second step, Lemma 1 below, shows that when κ is fixed then being a union of $\simeq_\kappa$ is decidable. The proof of the second step is straightforward and is left to the reader.

Lemma 1 Let L be a regular tree language and κ a number. It is decidable whether L is in LT_κ or not.

The rest of this section is devoted to the proof of the first step showing that for k-tame regular tree language a bound on κ can be obtained. This is a consequence of the lemma below. Recall that for each k, the number of k-types is finite. Let β_k be this number.

Lemma 2 Let L be a k-tame regular tree language. Set $l = \beta_k + 1$. Then for all $l' > l$ and any two trees t, t' if $t \simeq_l t'$ 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. $T \simeq_{l'} T'$

To see that the first step follows from Lemma 2, assume that L is a k-tame regular tree language in LT. Then, by definition of LT, L is in $\mathrm{LT}_{l'}$ for some l'. If $l' > \beta_k + 1$ then, from Lemma 2, L is also in LT_l. Hence for testing membership

in LT it is sufficient to test membership in LT_l for $l = \beta_k + 1$ which is decidable by Lemma 1.

Before proving Lemma 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. Note that if C is a loop of k-type τ and x is a node of type τ in a tree t then inserting C at node x does not modify the k-type of the nodes of t (it may add new k-types, coming from the nodes of C). In particular the k-type of x is unchanged. 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 of t are unchanged by this operation.

Proof (of Lemma 2). Suppose that L is k-tame. Recall that the number of k-types is β_k. Therefore, by choice of l, in every branch of a l-type one can find at least one k-type that is repeated. This provides many k-loops that can be moved around whenever necessary in order to obtain similar bigger types.

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

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' concatenation 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 inserting simultaneously a copy of the context C_τ at all nodes of type $\tau \in B$ of t.

We now show that T and T' have the desired properties. The third property is easy to verify (proof omitted in this abstract).

Claim 1 $T \simeq_{l'} T'$

The other two properties, $t \in L$ iff $T \in L$ and $t' \in L$ iff $T' \in L$, are not obvious at all. This is the difficult part of the proof and it requires tameness of the language. It is a consequence of Lemma 3 below. □

In order to conclude the proof of Proposition 3 it remains to show the following lemma. This lemma shows a key consequence of the fact that L is k-tame: the insertion of k-loops that don't introduce new $(k+1)$-types preserves membership into L.

Lemma 3 Let L be a k-tame regular tree language. 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 loop of k-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 $(k+1)$-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 x, y of some tree T, we say that x is a **l**-ancestor of y if y is a descendant of the left child of x. 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 stutter 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. 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 we 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 left child x' of x and notice that because y_n is a child of y_{n-1} and x_{n-1} has the same $(k+1)$-type than 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 x as the left child of x_{n-1} and we are done.

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 for all $0 \leq i < n$, x_i is of type τ_i.

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 that 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}. Notice that the context between the appropriate child of x_{i-1} and x_i is a k-loop. Using k-guarded vertical swap 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}. 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' in x. The proof of the following claim is omitted in this abstract.

Claim 2 *$T' \in L$ iff $t \in L$.*

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 x_i and T'_i be the subtree branching out the principal path of C' at x_i. The claim below shows that replacing T'_i with T_i does not affect membership into L. Hence a repeated use of that claim eventually shows that $T' \in L$ iff $T \in L$.

Claim 3 *Let u and u' be two trees. Assume s and s' are subtrees respectively of u and u' such that the roots of s and s' have the same k-type. Consider the context D such that $u = Ds$.*

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

Proof (sketch). 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 root of s. Let τ be the $(k+1)$-type of the root of s'. Because $s' \preccurlyeq_{k+1} D$ we know that there exists a node y in D of type τ. We consider two cases depending on the relation between x and y.

- If y is an ancestor of x, let E be $u[y, x]$ and notice that x and y have the same k-type. Hence we can duplicate E using a k-guarded vertical stutter. The resulting tree is DEs and because L is k-tame, $DEs \in L$ iff $Ds \in L$. Let z be the root of E in DEs. Notice that by construction z is of type τ. Let s_1 be the subtree of DEs rooted at the left child of z and let s'_1 be the subtree of s' rooted at the left child of the root of s'. By construction $s_1 \preccurlyeq_{k+1} D$, $s'_1 \preccurlyeq_{k+1} D$. Because their parent have the same $(k+1)$-type, the roots of s_1 and s'_1 have the same k-type. As the depth of s'_1 is strictly smaller than the depth of s', by induction we can replace s_1 by s'_1 without affecting membership into L. Similarly we do the same for the right child and we are done.
- Assume now that x and y are not related by the descendant relationship. We know that x, y have the same k-type and that $s \preccurlyeq_{k+1} D$. Let s'' be the subtree of Ds rooted at y. It can be shown that, as a consequence of tameness (this is where k-guarded horizontal transfer is used), replacing Ds by Ds'' does not affect membership in L . As y'' is of type τ, we can proceed by induction as above and replace the left and right subtrees of s'' by their corresponding subtrees of s' to get the desired result. □

This concludes the proof of the decision algorithm in the case of trees. We note that in the case of strings Lemma 3 is extremely powerful and is sufficient

for showing that tameness implies membership in LT. This is due to the fact that, on strings, any two nodes with the same type induce a loop and therefore Lemma 3 applies to this loop. This lemma can then be used for transforming by induction a string to any other one with the same occurrences of types. However over trees this no longer work as the two nodes may be incomparable.

5 Unranked trees

In this section we consider unranked unordered trees, where each node has an arbitrary number of children but no order is assumed on these children. Our goal is to extend the result of the previous section and provide a decidable characterization of Locally Testable languages of unranked trees. Due to space limitations, we mostly only mention here our results, their proofs will appear in the journal version of this paper.

The first issue is to find an appropriate notion of LT for unranked trees. Recall that a language of binary trees is LT if its membership depends only on the presence or absence of neighborhoods of a certain size. With unranked trees, there may be infinitely many different possible neighborhoods of a certain size and hence this definition does not imply that the language is regular[1]. The idea is to replace isomorphism types with FO definable types such that for each isomorphism type there are only finitely many FO definable types.

We will use the following notion of type. The definition of k-type remains unchanged: it is the isomorphism type of tree induced by nodes at depth at most k. As mentioned above there are now infinitely many k-types. We therefore introduce a more flexible notion that depends on one extra parameter l that restricts the horizontal information. It is defined by induction on k. Consider an unordered tree t. For $k = 0$, the (k, l)-type of t is just the label of the root of t. For $k > 0$ the (k, l)-type of t is the label of its root together with, for each $(k-1, l)$-type, the number, up to threshold l, of children of the root of t of this type.

From this we define two classes of Locally Testable languages. The most general one, denoted ALT (A for *Aperiodic*), is defined as follows. Two trees are (k, l)-equivalent if they have the same occurrences of (k, l)-types and their roots have the same (k, l)-type. A language L is in ALT if there is a k and a l such that L is a union of (k, l)-equivalence classes. In the framework of forest algebra [BW07], languages in ALT have a syntactic forest algebra whose horizontal monoid satisfies $h^\omega = h^{\omega+1}$.

The second one, denoted ILT in the sequel (I for *Idempotent*), assumes $l = 1$: A language L is in ILT if there is a k such that L is a union of $(k, 1)$-equivalence classes. In the framework of forest algebra languages in ILT have a syntactic forest algebra whose horizontal monoid satisfies $h + h = h$.

The main result of this section is that we can extend the decidable characterization obtained for ranked trees to both ILT and ALT.

[1] In this section, by regular we mean definable in MSO using the child predicate. There is also an equivalent automata model.

Theorem 2 *It is decidable whether a regular unranked tree language is ILT.*
It is decidable whether a regular unranked tree language is ALT.

The notions of k-tame and (k,l)-tame are defined as in Section 3 using k-types and (k,l)-types. For unranked unordered trees both notions are identical:

Lemma 4 For every regular unordered tree language L and every k there is a number l, computable from k and L, such that if L is k-tame, then L is (k,l)-tame.

In the idempotent case we can completely characterize ILT. It corresponds to tameness together with an extra closure property denoted *horizontal stutter*. 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 into L.

Theorem 3 *A regular unordered tree language is in ILT iff it is tame and closed under horizontal stutter.*

This immediately implies the ILT part of Theorem 2 as both tameness and closure under horizontal stutter are decidable properties.

Proof (sketch). That the right-hand side conditions are necessary is obvious. For the other direction we first use Lemma 4 to compute l from k. We then prove the adaptation of Claim 3 to forests. Given two forests h and h', $h+h'$ denotes the forest consisting of the trees of h followed by the trees of h'. Given a forest h, $a(h)$ is the tree whose root has label a and whose children are the trees of h.

Consider now two trees t and t' that are (k,l)-equivalent. Then $t = a(h)$ and $t' = a(h')$ for some a and forests h and h' that are $(k-1,l)$-equivalent. Assume now that $t \in L$. By horizontal stutter, $a(h+h)$ is also in L. Because h and h' are $(k-1,l)$-equivalent we can use Claim 3 and replace one copy of h by h' and have $a(h+h') \in L$. Claim 3 applies again and yields $a(h'+h') \in L$. By horizontal stutter this implies that $t' \in L$.

Hence L is a union of (k,l)-equivalence classes. From closure under horizontal stutter this implies that L is a union of $(k,1)$-equivalence classes and is in ILT. □

For ALT we follow the lines of the binary tree case and Theorem 2 follows from the unranked variant of Proposition 3:

Proposition 4 *Assume L is a tame regular unordered tree language. Then it is decidable whether there exists a κ such that L is in ALT_κ.*

6 Discussion

We have provided a recursive procedure for testing whether a regular tree language is locally testable.

Our characterization extends to unranked unordered trees. For ordered trees, we believe that tameness together with a property that essentially say that the horizontal monoid is in LT should provide a decision procedure for an intuitive notion of LT over ordered unranked trees. Note that in this setting it is no longer clear whether tameness is decidable or not. We leave this case for future work.

From the minimal deterministic automata defining a regular tree language our procedure yields a multi exponential algorithm. On words this test for LT can be done in polynomial time. Note that testing whether a tree language is tame requires only polynomial time on the minimal deterministic bottom-up tree automata. A better complexity for testing LT could be obtained by exhibiting a nice set of identities for the class of LT. This is left for future work.

References

[Boj07a] M. Bojańczyk. A new algorithm for testing if a regular language is locally threshold testable. *Inf. Process. Lett.*, 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 Math.*, 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.

[BS09] M. Benedikt and L. Segoufin. Regular languages definable in FO and FOmod. *ACM Trans. Of Computational Logic*, 2009. To appear.

[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.

[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. *Math. Syst. Theor.*, 8:60–76, 1974.

[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. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.

[Til87] B. Tilson. Categories as algebra: an essential ingredient in the theory of monoids. *J. Pure Appl. 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.