Regular tree languages definable in FO and in FO$_{mod}$

Michael Benedikt Luc Segoufin

Abstract

We consider regular languages of labeled trees. We give an effective characterization of the regular languages over such trees that are definable in first-order logic in the language of labeled graphs. These languages are the analog on trees of the "locally threshold testable" languages on strings. We show that this characterization yields a decision procedure for determining whether a regular tree language is first-order definable: the procedure is polynomial time in the minimal automaton presenting the regular language. We also provide an algorithm for deciding whether a regular language is definable in first-order logic supplemented with modular quantifiers.

Keywords: Tree automata, Logic

1 Introduction

This paper is concerned with the relationship between regularity (acceptance by an automaton) and definability in first-order logic and first-order logic with counting quantifiers. Over strings this relationship is well-understood. A fundamental result in formal language theory [Buc60] states that a language of strings is regular – that is, equal to the language of strings accepted by a finite string automaton – exactly when it is definable in monadic second-order logic (MSO) over the vocabulary consisting of the successor relation on strings and the labels. By restricting to first-order logic (FO) rather than MSO, we can obtain two proper subcollections of the family of regular languages. The languages that are definable in first-order logic over the transitive closure of the successor relation and the label predicates, which we denote FO($<$), are exactly the star-free or, equivalently, the aperiodic languages [MP71, Sch65]. The languages that are definable in first-order logic over the successor relation and the label predicates, which we denote by FO, correspond to locally threshold testable languages (see [Tho97]). Using a fundamental result of Thérien and Weiss [TW85], Beauquier and Pin [BP89] gave an algebraic characterization of the FO languages. They are exactly the languages for which the corresponding monoid satisfies certain identities. Put another way, they show that the monoids corresponding to FO-definable languages form a pseudo-variety within the collection of all finite monoids. Both the characterization of FO($<$)-definability via aperiodicity and the characterization of FO-definability of Beauquier and Pin lead to effective algorithms for checking whether a regular language is FO($<$) (resp. FO) definable. Straubing [Str94] provides an analogous characterization for the logic FO$_{mod}$ that extends first-order logic with quantifiers that count modulo a given integer. A complete overview of the string case can be found in [Tho97] or in [Str94].

We now consider the situation over ranked trees – labeled trees with a fixed bound on branching. Regularity is now defined as acceptance by a (non-deterministic top-down or deterministic bottom-up) tree automaton, and regularity is shown to be equivalent to definability in monadic second-order logic in the vocabulary of labeled graphs [Don70, TW68]. In this context we use FO($<$) to denote first-order logic over the labels and the transitive closure of the graph relation (that is, the descendant relation on trees). We use FO to denote first-order logic over the graph relation and the labels, and

FO_{mod} to denote first-order logic with counting quantifiers (modulo an integer) over the graph relation and labels. The notions of aperiodicity and star-freeness have natural extensions to the tree context, but here FO($<$) is strictly weaker than aperiodicity and star-freeness [PT93, Heu91, Pot95]. Finding a decidable characterization of FO($<$) within the regular tree languages is a longstanding open problem; partial results (see below) are given in [EW03, BW04]. As in the string case, FO definability is known to be strictly weaker than FO($<$) definability, but surprisingly an effective characterization of FO-definability was also lacking. [Wil96] gives an algebraic characterization and decision procedure for the frontier testable languages, a subclass of the FO definable languages. [BW04] provides a decision procedure for two fragments of FO($<$) defined using existential path quantification; none of these fragments exactly matches the expressiveness of FO. [EW03] gives a characterization of the FO($<$) definable languages in terms of an algebraic structure (the "syntactic pre-clone") associated with the language; this characterization is not known to be effective. To our knowledge, the decidability of definability in each of these logics was also unresolved over trees.

In this work we give an effective characterization of definability in FO over trees, ranked or unranked. Over ranked trees FO still corresponds to the Local Threshold Testable (LTT) languages, but this characterization does not yield a decision procedure. Our main result is an effective characterization of FO within the regular tree languages that uses a set of equivalences that preserve membership within the language. Unlike the string case, these equivalences include preconditions requiring portions of the tree to be similar "locally". They are thus a midpoint between a definition using locally threshold testability (which characterizes FO over ranked trees, but which is not effective) and a purely algebraic approach. We extend our characterizations to give characterizations of FO-definable languages over unranked tress as well.

As an application of the characterization theorems, we show that over strings, our results yield a new proof of the algebraic characterization of LTT and of the decidability of membership in LTT over strings presented in [BP89, Str94]. The current proofs of the characterization of FO in the string case use either fundamental (and difficult) results in the theory of monoids [BP89] or difficult results within the theory of finite categories [Str94]. Nevertheless several of the technical lemmas remain identical in inspiration if not in notation to the earlier proofs.

We then show that our characterization theorem yields that one can decide whether a regular language of trees is definable in FO, both over ranked and unranked trees. We show in fact that membership in these classes can be decided in polynomial time in the size of a minimal automaton accepting the regular language. Finally, we show that in the ranked tree case our techniques and results also yields a decision procedure for membership of a regular language in FO_{mod}. We also state characterizations for FO_{mod}, both for the ranked case and the unranked case, in the same spirit as those obtained for FO. Those characterizations yields a PTIME algorithm for testing membership in FO_{mod}.

Organization: Section 2 gives the basic notation for this article. Section 3 states and proves the characterization theorem for FO in the case of ranked trees. Section 4 extends to prove the characterization in the unranked case. Section 5 shows how the results for strings follow from the tree case and gives the decision procedures that follow from the characterization theorem. Section 6 provides extensions of the results to first-order logic supplemented with counting quantifiers. Section 7 gives conclusions and open issues.

This paper is a journal version of our STACS'05 paper [BS05]. It contains the full proofs of the decidability results claimed in [BS05]. The exact characterization claimed in [BS05], however, was incorrect.

2 Notation

Trees We fix a finite alphabet Σ, and consider trees with labels in Σ. In this paper we will deal with two settings. In the *ranked* setting, we fix some integer r and consider Σ-labeled trees of rank r; that is, each node has at most r children. In this case, the children of any given node are *ordered*: that is, we can distinguish the first child, second child, and so forth. In the *unranked* setting there is no bound on the number of children and we will always take the children to be *unordered*. Finding a decidable characterization in the unranked ordered case remains an open question.

In both cases, we use standard notation for trees. By the *descendant* (resp. ancestor) relation we mean the reflexive transitive closure of the child (resp. inverse of child) relation. We use $\mathcal{T}(\Sigma, r)$ for the set of trees of rank at most r with labels coming from alphabet Σ, and $\mathcal{T}(\Sigma, \omega)$ for the set of unordered trees of any finite rank with labels from Σ. When the setting is clear, or when we assert something that holds in all settings, we just write $\mathcal{T}$.

For trees t, t', we say that t' is a *subtree* of t if the nodes of t' are a subset of those of t and the edge relation and labeling function of t' are obtained from those of t by restricting to the nodes of t'. Thus if t' is a subtree of t, t' need not contain the root of t, and leaves of t' need not be leaves of t. We say that t' is a *prefix* of t if t' is a subtree of t that contains the root of t.

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 $t|_x$. Let t be a tree and x be a node of t, the *k-spill* of x is the restriction of $t|_x$ to the set of nodes of t at distance at most k from x. Given a tree t and a set S of nodes of t, the minimum subtree of t containing S is the smallest tree whose root r is the least common ancestor of all nodes of S and which contains all nodes of S and their ancestors up to r.

Given two nodes x and x' occurring respectively in the trees t, t' we say that x is depth-k similar to x' if the k-spill of x in t is isomorphic to the k-spill of x' in t'. Similarly two trees t and t' are depth-k similar if their roots are depth-k similar. When we are in the ranked case, isomorphism must preserve the order of children, but in the unranked case it need not.

A *context* is an (ordered or unordered) 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. Concatenation of a context C and a tree t is defined similarly. 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.

Tree automata Regular tree languages will be represented by finite state automata. Over r-ranked trees, a (deterministic bottom-up) tree automaton A is defined in the usual way; it has a finite set of states Q, a set $F \subseteq Q$ of accepting states, and a transition function δ associating a unique state to any pair in $(Q^i \times \Sigma)$ for $i \leq r$.

A tree automaton A over unordered unranked trees consists of a finite set of states Q, a set $F \subseteq Q$ of accepting states, an integer m and a transition function δ associating a unique state to any pair $(\Gamma_m^Q \times \Sigma)$ where $\Gamma_m = \{= i, \geq i \mid i \leq m\}$. We furthermore assume that if there are functions $f, g \in \Gamma^Q$ with f giving a more restrictive constraint than g on every $q \in Q$, then $\delta(f, a) = \delta(g, a)$; that is, A is deterministic. The transition function associates a unique state to any pair in $Q^i \times \Sigma$. The number m is called the *tolerance* of A.

As usual a run τ of A on a tree t is a function from the set of nodes of t to Q. The notion of a valid run for ranked trees is standard. In the case of an unranked tree automaton, a run τ is valid if for any node x of label $a \in \Sigma$, there exists a function $f \in \Gamma^Q$ such that $\delta(f, a) = \tau(x)$ and, for every $q \in Q$, the number of children y of x such that $\tau(y) = q$, is consistent with $f(q)$. Each tree t has a unique valid run. A tree t is accepted by A if the valid run τ of A on t is such that the image under τ of the root of t is in F. Languages accepted by such automata are called *regular languages*. It is folklore that this corresponds to the usual definition over ranked and unranked trees (see also [Tho97]).

An automaton A with set of states Q and a context C induce a function from Q to Q, sending a state q to the state q' reached by A at the root of C assuming state q at its port. For contexts C_1 and C_2, we write $C_1 \simeq C_2$ if they induce the same function. In particular, in this case we have $C_1 \cdot t$ is accepted by A iff $C_2 \cdot t$ is accepted by A.

Logics Monadic Second Order Logic (MSO) and First Order Logic (FO) are defined over trees in the standard way. In the case of r-ranked trees, they will be defined over the signature containing one unary predicate P_a per letter $a \in \Sigma$ and the tree successor relations $E_1 \dots E_r$, where $E_i(x, y)$ holds if y is the i^{th} child of x. In the case of unranked trees, they are defined over the signature containing one unary predicate P_a per letter $a \in \Sigma$, and the tree successor relation. A tree language is said to be *regular* if it is definable in MSO or, equivalently, recognized by a tree automaton.

For any formula $\varphi \in$ FO, its quantifier rank qr(φ) is defined as the nesting depth of the quantifiers of φ as usual. The elementary equivalence up to depth n is denoted by $\equiv^n$: for any two trees $t, t' \in \mathcal{T}$ we say that $t \equiv^n t'$ if t and t' satisfy exactly the same FO sentences of quantifier rank less than n.

The logic FO$_{mod}$ extends FO by allowing formulae to be built up by the rule $\psi(\vec{y}) = \exists^{r,q} x\ \phi(x, \vec{y})$, where r, q are integers with $r < q$. This holds in a structure $(G, \vec{y})$ iff the number of x such that $(G, \vec{y}, x)$ holds is equal to r modulo q. If P is a finite set of integers we let FO$_{mod(P)}$ be the extension of FO with the constructors above, where we restrict q to be in P.

3 Ranked trees

3.1 Statement of the main result

In this section we fix $r \in \mathbb{N}$ and we assume that all trees are in $\mathcal{T}(\Sigma, r)$.

Swaps 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.

Let t be a tree of root a, and x, y, x', y' be four nodes of t such that y is a descendant of x, x' is a descendant of y and y' is a descendant of x'. The *vertical swap* of t between $[x, y)$ and $[x', y')$ is the tree t' constructed from t as depicted in Figure 1. More formally let $C = C_t[a, x)$, $\Delta_1 = C_t[x, y)$, $\Delta_2 = C_t[x', y')$, $\Delta = C_t[y, x')$, $T = t|_{y'}$. Then notice that $t = C \cdot \Delta_1 \cdot \Delta \cdot \Delta_2 \cdot T$. The tree t' is defined as $t' = C \cdot \Delta_2 \cdot \Delta \cdot \Delta_1 \cdot T$.

Guarded swaps Let $k \in \mathbb{N}$, $t \in \mathcal{T}$ and x, y, x', y' be nodes of t. A horizontal swap is said to be *k-guarded* if x and x' are depth-k similar. A vertical swap is said to be *k-guarded* if x and x' are depth-k similar and y and y' are depth-k similar.

Let L be a tree language and k be a number. We say that L is *closed under k-guarded swaps* if for every tree $t \in L$ and every tree t' constructed from t by either a horizontal or a vertical k-guarded swap then t' is in L. Note that being closed under k-guarded swaps implies being closed under k'-guarded swaps for $k' > k$.

A regular tree language L is said to be aperiodic if there exists $l \in \mathbb{N}$ such that for all contexts C, Δ and every tree T, $C\Delta^l T \in L$ iff $C\Delta^{l+1}T \in L$. The least such l is referred to as the *aperiodicity number* of L. This just the classical notion of aperiodicity in the monoid of contexts.

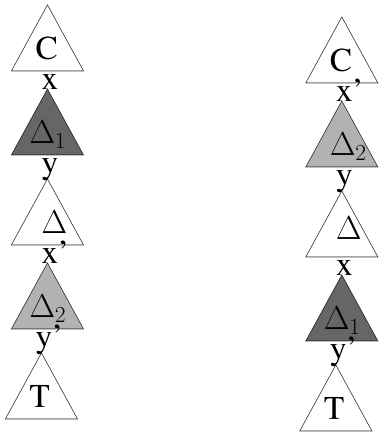


Figure 1: Illustration of the vertical swap

Theorem 1. *Let L be a regular tree language.*

Then L is definable in FO iff L is aperiodic and there exists a k such that L is closed under k-guarded swaps.

The "only if" direction of Theorem 1 is easy. If L is definable in FO , then L is aperiodic [Tho97]. It is also known that for any FO sentence ϕ there is a number k such that the truth of ϕ is determined by the number of k-neighborhoods of each isomorphism type. The least such k is referred to as the *locality rank* of ϕ [Lib04]. A k-neighborhood in a graph G is the set of nodes that are within distance k of some point in G. Since k-guarded swaps preserve the number of k-neighborhoods of every isomorphism type, it follows that if L is definable by an FO sentence ϕ, then L is closed under k-guarded swaps, where k is the locality rank of ϕ.

The opposite direction follows from the following theorem, whose proof will be quite involved:

Theorem 2. *For any regular tree language L which is aperiodic and closed under k-guarded swaps, there exists a K such that for any $s, t \in \mathcal{T}$ we have: $s \equiv^K t \quad \Rightarrow \quad s \in L$ iff $t \in L$.*

Before proving Theorem 2 we show how Theorem 1 follows from it. From Theorem 2 we know that if L is aperiodic and closed under k-guarded swaps then L is a union of equivalence classes of $\equiv^K$ for some K. Standard arguments from finite model theory (see e.g. [Lib04]) show that $\equiv^K$ has only finitely many equivalence classes and that each of them is definable in FO. Therefore L is definable in FO as a disjunction of such formulas for the corresponding classes.

3.2 Proof of Theorem 2

In this section we fix an aperiodic regular tree language L with aperiodicity number l, a number k and assume that L is closed under k-guarded swaps. We also fix a deterministic bottom-up tree automaton A for L.

Because the trees are ranked, there are only finitely many isomorphism types of trees of depth at most k (by the depth of a tree, we mean the maximal length of any path). We denote the set of such isomorphism types by $\mathcal{T}_k$. Given a tree t and a node x of t, we write $T^t_k(x)$ for the isomorphism type of the k-spill of x in t, and denote it as the k-type of x (type of x when k is understood from the context). A crucial observation for the rest of the paper is that the $(k+1)$-type of a node determines the k-types of its children.

For each $\tau \in \mathcal{T}_k$ and any tree t we denote by $|t|_\tau$ the number of occurrences of the type τ in t. Given two trees s, t we write $s =_d^k t$ if for all $\tau \in \mathcal{T}_k$, $|s|_\tau = |t|_\tau$ or $|s|_\tau, |t|_\tau > d$ (s and t have the same number of occurrences of type τ up to threshold d). We write $s \leq^k t$ if for all $\tau \in \mathcal{T}_k$, $|s|_\tau \leq |t|_\tau$, and we write $s \leq_d^k t$ if $s =_d^k t$ and $s \leq^k t$. If for all $\tau \in \mathcal{T}_k$, $|s|_\tau = |t|_\tau$ then we write $s =_\infty^k t$.

Another fact that will be used repeatedly is that if we apply a k-guarded swap move to a tree t, there is an obvious bijection from the nodes of t to the resulting tree t' that preserves $(k+1)$-types; in particular, we have $t =_\infty^{k+1} t'$.

This first lemma shows that if we have the hypothesis of Theorem 2, then we can assume that s and t have the same number of types up to some threshold.

Lemma 1. For each number d, there is a number K_d such that $s \equiv^{K_d} t$ implies that $s =_d^{k+1} t$ and that s, t are depth-$(k+1)$ similar.

Proof. Choose K_d big enough so that we can count the number of satisfiers of any $(k+1)$-type up to threshold d using K_d quantifiers. □

The following lemma refines the previous one by showing that not only can we assume that s and t have the same number of types up to some threshold, but that this number is always bigger in t than in s.

Lemma 2. For each number d' there exists a number d such that if $s =_d^{k+1} t$ then there exists t' such that $s \leq_{d'}^{k+1} t'$, and t, t' are depth-$(k+1)$ similar, and $t' \in L$ iff $t \in L$.

Proof. Assume $s =_d^{k+1} t$ for some large enough d whose value will become apparent during the proof. Let β be the number of $(k+1)$-types τ such that $|t|_\tau < |s|_\tau$. We prove the lemma by induction on β. If $\beta = 0$ this is clear. Otherwise let τ be a $(k+1)$-type that occurs more times in s than in t. By hypothesis τ occurs at least d times in t.

Given two nodes x, y in a tree t with y a strict descendant of x, we write $\tau[x, y)$ for the number of nodes in the context $C_t[x, y)$ that have type τ. A *τ-skeleton of length* n is a sequence $x_i : 0 \leq i \leq n$ such that x_{i+1} is a strict descendant of x_i, and $\tau[x_i, x_{i+1}) \geq 1$ for each $0 \leq i \leq n-1$.

We first show that for every d_1 there is d such that for every tree u, if $|u|_\tau > d$ then there is a τ-skeleton of length d_1 in u. By the *interior* of a pointed tree, we mean all the nodes in it other than the port. Choose $d > (r+1)^{d_1}$. Starting with x_0 being the root of u, we will choose x_i inductively such that the interior of $C_u[x_{i-1}, x_i)$ has at least one node whose type in u is τ and x_i has at least $(r+1)^{d_1-i}$ nodes of type τ below it. Suppose that we have $x_1 \ldots x_i$. Let z_1 be a descendant of x_i of type τ having minimal depth. If there are no nodes of type τ in the interior of $C_u[x_i, z_1)$, then we know that there are at least $(r+1)^{d_1-i}$ nodes of type τ below z_1, including z_1. Hence there is some child of z_1 having at least $(r+1)^{d_1-(i+1)}$ nodes of type τ below it. Set x_{i+1} to be such a child. If there is some node of type τ in the interior of $C_u[x_i, z_1)$, then there is some node z_2 strictly between x_i and z_1 which has more than one child having a node of type τ below it. Taking z_2 to be the highest such node, it is clear that one of the children of z_2 must have at least $(r+1)^{d_1-(i+1)}$ nodes of type τ below it; choose x_{i+1} to be this node. We can verify in either case that this preserves the induction hypothesis.

We apply this to the tree t, for d_1 to be chosen later on, getting a τ-skeleton $x_i : 0 \leq i \leq d_1$. Let q be one more than the product of the number of $(k+1)$-types and the number of states in the automaton. So given a set $x_1 \ldots x_q$ of nodes in t, there must be x_d, x_e with $d \neq e$ where x_d and x_e have the same $(k+1)$-type and such that the automaton run on t reaches the same state at x_d and x_e. states of A. The nodes in the interior of the context between x_i and x_{i+q}, for $i \leq d_1 - q$ with $i = 0$ mod q form a *section* of t. We say that a $(k+1)$-type ν is *safe* if $|t|_\nu \geq d'$. A section is *safe* if it contains only nodes having safe $(k+1)$-types. Because the number of sections is at least $(d_1/q) - 1$, we can choose d_1 big enough so that at least one of them is safe. Given this choice of d_1, fix x_i such that all nodes in the interior of the context between x_i and x_{i+q} are safe. By the choice of q, there are d and e with $i \leq d < e < i + q$ such that the run of automaton A on t reaches the same state at x_d as at x_e, with x_d and x_e having the same $(k+1)$-type. Hence we can replace the context $C_t[x_d, x_e)$ with arbitrarily many copies of itself, without changing membership in L, without changing membership in L. Since x_j and x_k have the same type, performing this replacement does not change the $(k+1)$-types of any node of $C_t[x_j, x_k)$, and the type of any node in $C_t[x_j, x_k)$ within t is the same as the type of each of its copies in the modified tree. Thus we only added copies of safe types. Therefore we have $t' =_{d'}^{k+1} t =_{d'}^{k+1} s$. Now since $C_t[x_j, x_k)$ contains an occurrence of τ, by adding sufficiently many copies of the resulting tree t', we have reduced β by one and we can conclude by induction. □

A tree t is *k-pseudo-included* in a tree t' if there is an injective mapping h from nodes of t to nodes of t', sending the root of t to the root of t', and such that: (i) h preserves types in $\mathcal{T}_k$, and (ii) if x is the i^{th} child of y in t then $h(x)$ is a descendant of the i^{th} child of $h(y)$ in t'. In this case the *h-pseudo-tree* is the minimum prefix of t' which contains $h(t)$.

The next step shows that we can also assume that s is pseudo-included in t. It requires only the closure of L under k-guarded swaps.

Lemma 3. If $s \leq^{k+1} t$ and s, t are depth-$(k+1)$ similar then there exists t' such that s is $(k+1)$-pseudo-included in t', $t' =_{\infty}^{k+1} t$, t, t' are depth-$(k+1)$ similar, and $t' \in L$ iff $t \in L$.

Proof. The proof is by induction. We construct $t_0 \cdots t_n$ such that: t_0 is t and, for all $0 \leq i < n$, t_{i+1} is obtained from t_i using only k-guarded swaps, if s_i is a prefix of s maximal with respect to the property that s_i is $(k+1)$-pseudo-included in t_i, then there exists a node x_i of s that is a child of a leaf of s_i, such that $x_i \in s_{i+1}$. Since s_i cannot keep growing forever, we must eventually have $s_n = s$. This implies the lemma by taking $t' = t_n$, using the fact that k-guarded swaps preserves the number of $(k+1)$-types and the assumption that L is closed under k-guarded swaps.

By hypothesis the root of s and the root of t have the same $(k+1)$-type. Thus we can initiate our process by mapping the root of s to the root of t.

Assume now that we have constructed t_i. Let s_i denote a maximal prefix s_i of s which is $(k+1)$-pseudo-included in t_i by a mapping h such that $h(a) = a'$. If $s_i = s$ we are done. Otherwise let x be a node of s_i such that its p^{th} child y is not in s_i. Let s' be a minimal prefix of s which contains s_i and y. We show how to transform t_i into t_{i+1} so that s' is $(k+1)$-pseudo-included in t_{i+1}.

Let $\tau = T_{k+1}^s(x)$, $\nu = T_{k+1}^s(y)$ and $x' = h(x)$. By hypothesis we know that there is a node y' in t_i outside of $h(s_i)$ such that $T_{k+1}^{t_i}(y') = \nu$. Let z' be the p^{th} child of x'. Note that z' cannot be in the h-pseudo-tree.

We distinguish several possibilities depending on the relative position of x' and y'. By maximality of s_i we know that y' is not below z'.

Assume first that y' is outside the h-pseudo-tree. Then it is either below x' or not related to x' by the descendant relationship. It is crucial here that $h(a) = a'$, as it rules out the case where y' occurs above $h(a)$. Because x and x' agree on their $(k+1)$-types, z' and y' are depth-k similar. We can apply the k-guarded horizontal swap to these two nodes. This yields the desired tree t_{i+1}, as we can now extend h by setting $h(y) = y'$. We can verify that this yields a $(k+1)$-pseudo-inclusion mapping, since the new $(k+1)$-type of y' remains ν.

Assume now that y' is inside the h-pseudo-tree. Let x_1 be the deepest node in s_i such that $x'_1 = h(x_1)$ is an ancestor of y', and x_2 be the highest node in s_i so that y' is an ancestor of $x'_2 = h(x_2)$. Note that the definition of pseudo-inclusion implies that x_2 is uniquely defined. Assume that x_2 is the j^{th} child of x_1 in s and let z'_1 be the j^{th} child of x'_1 in t'_i. Note that z'_1 cannot be in the image of h, since this would contradict the maximality of x_1. There are two cases to consider.

The first case is when x' is a descendant of x'_2 (see Figure 2). Because h preserves $(k+1)$-types, z'_1 and x'_2 are depth-k similar and the same holds for y' and z'. We can thus apply the k-guarded vertical swap between $[z'_1, y')$ and $[x'_2, z')$ and obtain the desired tree t_{i+1}. We can then extend h by setting $h(y) = y'$. It remains to verify that this indeed gives a $(k+1)$-pseudo-inclusion mapping. This is straightforward and left to the reader.

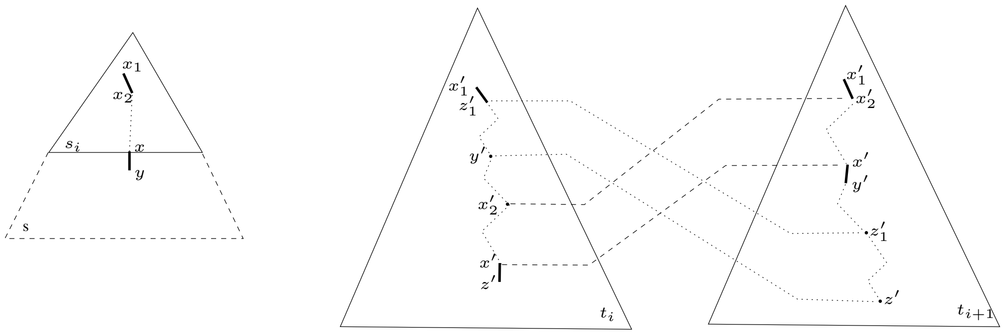


Figure 2: This illustrates the case when x' is a descendant of x'_2. s_i and s are depicted on the left. t_i is depicted in the middle. After applying the k-guarded vertical swap between $[z'_1, y')$ and $[x'_2, z')$, we reach t_{i+1} depicted to the right. The nodes z'_1 and x'_2 have the same k-types and the nodes y' and z' have the same k-types.

If x' is not a descendant of x'_2 we proceed as follows. As above, we know that y' and z' are depth-k similar. If x' is a descendant of y', then it would have to be a descendant of x'_2 as well, since all pseudo-tree elements beneath y' lie beneath x'_2. Hence we know x' cannot be a descendant of y', and so z' is not a descendant of y' either. We can therefore apply the k-guarded horizontal swap to y' and z', obtaining an intermediate tree t'_i. In t'_i, we have that x'_2 and z'_1 are depth-k similar and we can apply again the k-guarded horizontal swap to obtain the desired tree t_{i+1}. The mapping h is extended by sending y to y', and it is immediate to see that this preserves $(k+1)$-types. □

An immediate corollary of Lemma 3 is:

Corollary 1. *If s and t are trees that are depth-$(k+1)$ similar such that $s =_\infty^{k+1} t$ then $s \in L$ iff $t \in L$.*

Proof. Apply Lemma 3 to s and t and notice that the tree t' obtained is isomorphic to s via the $(k+1)$-pseudo-inclusion mapping h, as the hypothesis implies that t' cannot contain any extra nodes. □

At this point, we know that we can transform t into t'' so that s is pseudo-included in t'' by some mapping h. Thus t'' is a copy of s plus extra contexts inserted between elements of $h(s)$. We also know, by the corollary above, that if we could get the types of t'' to match those of s exactly, we would be done. Our next goal will be to add these contexts to s one by one. We will use the crucial observation that all $(k+1)$-types occurring outside of $h(s)$ have strictly more occurrences in t'' than in s, and hence must have many occurrences in s. To make the last step formal we will need further notation and one extra lemma.

Let C be a context where the port is not the same as the root, and let λ be a function assigning a k-type to the port of C and a $(k+1)$-type to each other node of C (in the case where C consists of only a single node, then λ will assign only a $k+1$-type to this node). λ is said to be *consistent* if there exists a tree t such that for every non-port node x in C the $(k+1)$-type of x in $C \cdot t$ matches $\lambda(x)$. An *abstract context* is a context C whose root is not equal to its port, supplemented with a consistent assignment λ. We extend the basic definitions on trees to abstract contexts in the obvious way: if $U = (C, \lambda)$ is an abstract context, we will refer to any node in C as a node of U, and similarly refer to the root of U, child relation on U, etc. Given a node x of U (i.e. a node of C), we will refer to $\lambda(x)$ as the $(k+1)$-type of x. Given two abstract contexts U and V we say that U is *compatible* with V if the $(k+1)$-type of the root of V, when seen as a k-type, is the k-type of the port of U. If U and V are compatible abstract contexts then $U \cdot V$, the concatenation of U and V, is also an abstract context with the obvious consistent assignment. We can also can concatenate an abstract context with a tree. An abstract context U and a tree t are compatible if the $(k+1)$-type of the root of t is the k-type of the port of U. In this case, the concatenation $U \cdot t$ will be a tree.

An abstract context U is an *abstract loop* if U is compatible with U. Thus if U is an abstract loop, then U^n, the concatenation of n copies of U, is also an abstract context for any $n \in \mathbb{N}$.

Loops will play a significant role in reducing t to s. Observe that if h witnesses that s is $(k+1)$-pseudo-included in t and y is the p^{th} child of x in s, then $C_t[z, h(y))$ where z is the p^{th} child of $h(x)$, together with the obvious assignment, is an abstract loop in t.

Given a tree t and an abstract context U, we say that U is $(k+1)$-*included* in t if there is a function from C to t preserving the i^{th} child relation for every $i \leq r$ which also preserves $(k+1)$-types. We say that $U <_{k+1} t$ if the number of occurrences of each $(k+1)$-type in U is strictly less that the number of occurrences of the same $(k+1)$-type in t.

We are now ready to state and prove our last technical lemma. It is very similar in spirit to Lemma 3 and its proof follows exactly the same ideas. However it differs in Lemma 3 in two crucial respects. The hypothesis on the number of types is stronger, as we require strictly more types in t than in U. The conclusion is stronger, as we replaced pseudo-inclusion by inclusion.

Lemma 4. Let t be a tree, k a number, and U an abstract context.

If $U <_{k+1} t$ then there exists t' such that U is $(k+1)$-included in t', $t' =_{\infty}^{(k+1)} t$, and, $t' \in L$ iff $t \in L$.

Proof. The proof is similar to that of Lemma 3. It is done by induction and requires a lengthy case analysis.

An abstract context U is *weakly* $(k+1)$-*pseudo-included* in a tree t' iff there is an injective mapping h from nodes of U to nodes of t' that satisfies the requirements for pseudo-inclusion, except for the requirement that the root of U is mapped to the root of t'. We will likewise talk about weak $(k+1)$-pseudo-inclusion mappings and weak $(k+1)$-pseudo-trees. The first step is to transform t' into t'' so that there is a weak $(k+1)$-pseudo-inclusion of U into t''. Note that we cannot directly apply Lemma 3 as the hypothesis on the root types was crucial. In the proof of Lemma 3 this was reflected in the fact that if y' is not in the h-pseudo-tree then it cannot be above the image under h of the root of s. Without this the proof would not go through. However with our stronger hypothesis on the number of types, this case can now be handled.

Claim 1. *If $U <_{k+1} t$ then there exists a tree t'' such that U is weakly $(k+1)$-pseudo-included into t'', $t'' =_{\infty}^{k+1} t$, and, $t'' \in L$ iff $t \in L$.*

Proof. The proof is done exactly as in the proof of Lemma 3 with the following differences. In the base case, the image of the root of U is now an arbitrary node of $t = t_1$ whose type matches the type of the root of U; an inclusion mapping does not demand preservation of the root. During the induction we have constructed t_i and a prefix U_i of U which is weakly $(k+1)$-pseudo-included in t_i. Let a be the root of U. Recall the proof of Lemma 3. We have two nodes $x, y \in s$ such that y is a child of x and is of type ν. We also have three nodes nodes $x', y', z' \in t_i$, such that $x' = h(x)$, z' is a child of x', and y' is a node outside of $h(U_i)$ of type ν. We are trying to modify t_i in order to put a node of type ν below z'. This is done by a case analysis depending on the relative position of x', y' and z'. All cases are handled as in Lemma 3 but we now need to consider one extra case which was not possible in Lemma 3.

Assume y' is an ancestor of $h(a)$. By hypothesis we know that there is a $y'' \neq y'$ outside of $h(U_i)$ whose type is also ν. If we re-do the case analysis with y'' playing the role of y' we are left again with the case where both y' and y'' are ancestors of $h(a)$. Assume without loss of generality that y'' is a strict descendant of y'. Notice that z', y', and y'' are depth-k similar. We can apply the k-guarded vertical swap to $[y', y'')$ and $[y'', z')$. This yields the desired tree t_{i+1} as y' is now the p^{th} child of x'. Notice that the presence of y'' was crucial for this step. □

Using Claim 1 we can assume without loss of generality that U is weakly $(k+1)$-pseudo-included in t.

Let Δ be the set of vertices y of U such that the parent of y, denoted by x, is in U, and such that $h(x)$ and $h(y)$ are not in a parent/child relation. Note that the nodes in Δ do not necessarily form a subtree. Let $n = |\Delta|$ and $m = \Sigma_{y \in \Delta} d(y)$, where $d(y)$ denotes the depth in U of y. If $n = 0$ we already have a $(k+1)$-inclusion mapping and we are done. If not we show that it is possible to modify h and re-arrange t via swaps and obtain a new weak $(k+1)$-pseudo-inclusion for U mapping with $(n', m') < (n, m)$, where $<$ denotes the lexicographic ordering on pairs. By repeating this argument we eventually get a k-inclusion mapping of U into some tree t'.

Assume that $n \neq 0$ and take x and y such that $y \in \Delta$ is the p^{th} child of x, and consider $x' = h(x)$ and $y' = h(y)$. Let τ be the $(k+1)$-type of x and ν be the $(k+1)$-type of y. By assumption we know that there is another node y'' outside of the image $h(U)$, such that the type of y'' is ν. Let z' be the p^{th} child of x'. Assume first that $z' = y''$. Then the type of z' is ν and we can modify h by setting $h(y)$ to z' while reducing n by 1. Otherwise we have that y', y'', and z' are all distinct nodes. Notice however that z', y' and y'' are depth-k similar. We perform a case analysis depending on the relationship between z', y', and y''.

In the first case, we assume that y'' is an ancestor of z'. We apply the k-guarded vertical swap to $[y'', z')$ and $[z', y')$, obtaining a tree t_1. This case is depicted in Figure 3. Notice that U is still weakly k-pseudo-included in t_1 and that y' is now the p^{th} child of x'. Hence n has decreased by one.

In the second case, we assume that y'' is a descendant of y'. We apply the k-guarded vertical swap to $[z', y')$ and $[y', y'')$, with h being modified so that if before it maps some node w to a node w' that is swapped, it will now map w to the image of w' under the swap. Via this modification U remains weakly k-pseudo-included in the new tree, and in this tree node y' is now the p^{th} child of $h(x)$, thus decreasing n by one.

In the third case, we assume that y'' is not related to z' and y'. We first apply k-guarded horizontal swapping to y'' and z' and again between y'' (i.e. the image of y'' under the previous swap: for brevity we omit this distinction henceforth) and y'. We then modify h by composing with the swap mappings,

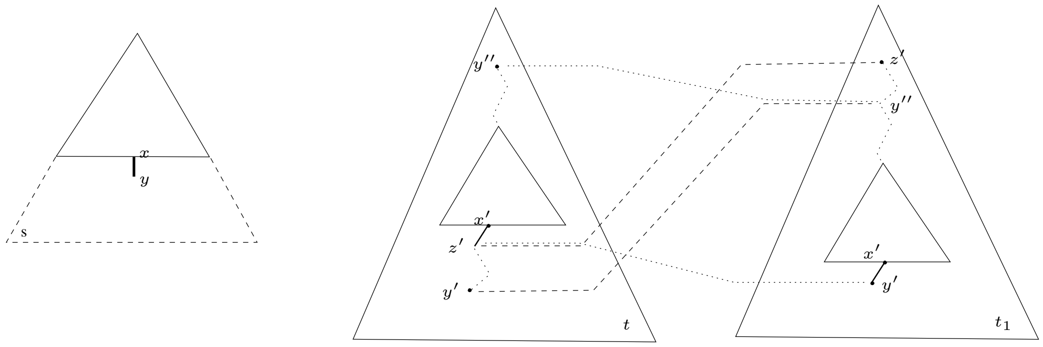


Figure 3: Illustration of the first case. s is depicted on the left, with U the solid triangle within it. t is in the middle, with the weak pseudo-image of U depicted as a solid triangle within it. The tree t_1 resulting from the swap is shown on the right. Notice that y' becomes a child of x' and that no other nodes of the weak pseudo-inclusion are affected.

giving a weak k-pseudo-inclusion in the obvious way. Again we have connected $h(x)$ and y' and decreased n by one.

In the fourth case, we assume that y'' is between z' and y'. But then we can change h so that $h(y) = y''$ and still get a weak k-pseudo-inclusion of U into t' via the new h. We then proceed as in the second case.

The last case is when y'' is a descendant of z' but is not related to y'. If $\nu = \tau$ then let z'' be the p^{th} child of y' and notice that z', y' and z'' are depth-k similar. We apply the k-guarded vertical swap to $[z', y')$ and $[y', z'')$ and the reader can verify that we are done. If $\nu \neq \tau$ then by assumption we know that there is a node x'' outside of $h(U)$ such that the k-type of x'' if τ. Let z'' be the p^{th} child of x''. Notice that z'', z', y' are depth-k similar. Again we have to consider several subcases.

In the first subcase z'' is not related to z'. We apply the k-guarded horizontal swap to z'' and z' followed by a k-guarded horizontal swap applied to z'' and y'. The reader can verify that this yields a tree t_1 with the desired properties: y' is now the p^{th} child of x' and the weak k-pseudo-inclusion mapping is only affected there, thus decreasing n by one.

In the second subcase z'' is a descendant of y'. Then we apply the k-guarded vertical swap to $[z', y')$ and $[y', z'')$ and obtain a tree where n is decreased by one.

The third subcase is when z'' is an ancestor of z'. Then we apply the k-guarded vertical swap to $[z'', z')$ and $[z', y')$. The reader can verify that n is decreased by one.

The fourth subcase is when z'' is below z' but not related to y'. We apply the k-guarded horizontal swap to z'' and y', attaching y' to x''. We now modify h by mapping x to x'' instead of x'. It is easy to verify that h is still a weak k-pseudo-inclusion mapping for U. It is also easy to check that, with this new mapping, n does not increase: y is no longer in Δ but x is now in Δ, with no other nodes moving into Δ. We remark now that with this new mapping m has decreased by one.

The last subcase is when z'' is between z' and y'. Then z', z'' and y'' are related as in the subcase above and we proceed replacing y' with y''. □

We are now ready to complete the proof of Theorem 2.

Proof of Theorem 2:

Let Q be the set of states of A, $\alpha = |Q|$ be the number of states of A. Let $\beta_k = |\mathcal{T}_k|$. Recall that l is the aperiodicity number of L, and thus for every C and T we have $C\Delta^l T \in L$ iff $C\Delta^{l+1} T \in L$.

Let $d' = r^{(\beta_k * r^\alpha + 1)*l} + 1$. Let d be the number required in Lemma 2 for d'. Let K be the number K_d required in Lemma 1 for d. We show that $s \equiv^K t$ implies $s \in L$ iff $t \in L$.

Assume $s \equiv^K t$, we show that $s \in L$ iff $t \in L$. From Lemma 1 we know that $s =_d^{k+1} t$. Therefore by Lemma 2 there is a tree t' such that $t' \in L$ iff $t \in L$ and $s \preceq_{d'}^{k+1} t'$. We can now apply Lemma 3 and obtain t'' such that $t'' \in L$ iff $t' \in L$, s is $(k+1)$-pseudo-included in t'' via a mapping h, and $t'' =_\infty^{k+1} t' =_{d'}^{k+1} s$. Therefore it suffices to prove that $s \in L$ iff $t'' \in L$.

By construction t'' is $h(s)$ plus possibly some extra contexts inserted between elements of $h(s)$. We will consider each of these contexts one by one, with the aim of adding them to s. Let y be the p^{th} child of some node x in s such that $h(y)$ is not a child of $h(x)$. Let z be the p^{th} child of $h(x)$. By the definition of pseudo-inclusion, we see that $h(y)$ is a descendant of z and $C_{t''}[z, h(y))$, together with the obvious assignment is an abstract loop in t''. Let $V_1 \cdots V_n$ be the set of abstract loops that are obtained by this process from $t'' \backslash h(s)$. For each V_i and each $(k+1)$-type τ, we let $|V_i|_\tau$ denote the number of nodes in V_i that have type τ in t''. We will "pump" s until the number of occurrences of each $(k+1)$-type exactly matches the number in t''. To achieve this, by induction, we construct $s_0 \cdots s_n$ such that: (i) s_0 is s, (ii) for all $n \geq i > 0$, for all $\tau \in \mathcal{T}_{k+1}$, $|s_i|_\tau = |s|_\tau + |V_1|_\tau + \cdots + |V_i|_\tau$, (iii) $s_i \in L$ iff $s_{i-1} \in L$.

The base case is immediate. Assume the result for $0 \leq i < n$, and consider $V = V_{i+1}$.

Let f_V be the transition function on states associated to the context V. The first step is to *minimize* the size of V: find an abstract context V' such that the function associated to the underlying context of V' is f_V, V' uses the same $(k+1)$-types as V, and the size of V' is bounded by $r^{\beta_k * r^\alpha + 1}$. This is a pumping argument. To find such a V', label each node x of V with the pair (f, τ) where τ is the $(k+1)$-type of x in V and f is the transition function associated with the context obtained from the underlying context of V by removing all nodes that are not descendants of x (if the port of V is not below x, this will be a constant function). Now, whenever there is a branch in V which contains the same label twice, we prune the section from (and including) the top node to (and excluding) the bottom one, without affecting f_V. This yields an abstract context V' whose depth is bounded by $\beta_k * r^\alpha$. As the rank of V' is bounded by r, the total size of V' is bounded by $r^{\beta_k * r^\alpha + 1}$.

Now set $U = V'^l$. Because V' is an abstract loop, U is well-defined as an abstract context. Moreover its size is bounded by $r^{(\beta_k * r^\alpha + 1)*l} = d' - 1$. Recall the crucial observation that all $(k+1)$-types occurring outside of $h(s)$ have strictly more occurrences in t'' than in s; they thus appear at least d' times in s, and therefore in s_i. In particular, this is true of each $(k+1)$-type of U, hence by the choice of d' we can apply Lemma 4 to U and s_i and obtain $s'_i = \Delta_1 \cdot U \cdot \Delta_2$ for some context Δ_1 and tree Δ_2, such that $s'_i \in L$ iff $s_i \in L$, and $s'_i =_\infty^k s_i$. We can now use the aperiodicity of L and without affecting membership in L obtain a tree $s''_i = \Delta_1 \cdot V'^l \cdot V' \cdot \Delta_2$. Now set $s_{i+1} = \Delta_1 \cdot V'^l \cdot V \cdot \Delta_2$. Since f_V was the same as $f_{V'}$, moving from s''_i to s_{i+1} does not affect membership in L. We can easily see that for every $(k+1)$-type τ, $|s_{i+1}|_\tau = |s_i|_\tau + |V_i|_\tau$, and thus we have all the other desired properties.

This last step is depicted in Figure 4.

Let $s' = s_n$. By construction we have $s' =_\infty^k t''$ and $s' \in L$ iff $s \in L$. Theorem 2 now follows from Corollary 1.

4 Unranked trees

In this section we consider unranked trees. Each node may now have an arbitrary number of children. As mentioned in Section 2 we assume no order among the children of a node; in particular we cannot speak of the first child of a node. As usual we denote by *forest* a set of trees.

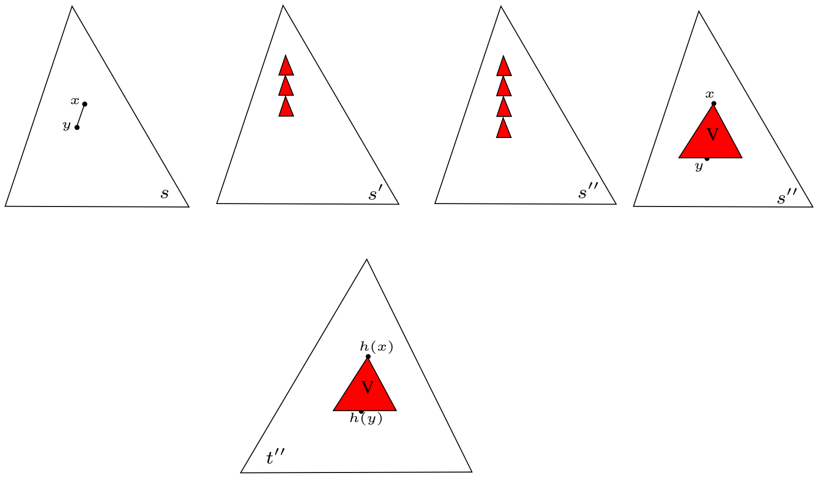


Figure 4: This figure depicts the last step of the proof. The top left tree is s, which is pseudo-included in t'' depicted at the bottom. The extra part of $t'' \backslash h(s)$ is depicted in dark grey and is the abstract context V. The second tree in the top row represents s' after applying Lemma 4, the dark grey parts representing $U = V^l$. Aperiodicity adds one more copy of V and yields the third tree in the top row. But this tree is essentially the initial one with V added between x and y, as depicted by the top right tree.

The new difficulty of the unranked case, compared with the ranked case, is that the number of isomorphism types of a k-spill of a node is infinite. We therefore need to relax the notion of similarity. For any number n we define an equivalence relation $\sim_n^k$ on trees of depth k by induction on k as follows. Let t and t' be two trees of depth k. Let r and r' their respective roots. In the case $k = 0$, $t \sim_n^0 t'$ if r and r' agree on their label. Otherwise $t \sim_n^k t'$ if, for each class **c** of $\sim_n^{k-1}$, the number of children of r in **c** must agree with the number of children of r' in **c** or both numbers must be bigger than n. It is immediate to see that, for each n, k, the equivalence relation $\sim_n^k$ is of finite index. For each node x of a tree t, the $\sim_n^k$-equivalence class of its k-spill is called the (n, k)-*type* of x. When n and k are understood from the context we simply say the *type* of x. Let $U\mathcal{T}_{n,k}$ be the (finite) set of (n, k)-types. If the (n, k)-type of node x in tree t is μ, we write $UT^t_{n,k}(x) = \mu$. For each $\tau \in U\mathcal{T}_{n,k}$ we extend the notation $|t|_\tau$, $=_d^{n,k}$, and $\leq_d^{n,k}$ in the obvious way.

Two nodes x, y are said to be (n, k)-similar if they have the same (n, k)-type. Note that (n, k)-similar implies (n', k')-similar for all $n' \leq n$ and $k' \leq k$. Two trees are said to be (n, k)-similar if their roots are (n, k)-similar.

The (n, k)-guarded swaps are defined as in the ranked case, replacing depth-k similar with (n, k)-similar. Note that if L is closed under (n, k)-guarded swaps then it is closed under (n', k')-guarded swaps for all $n' \geq n$ and $k' \geq k$.

We first focus on proving the following result, from which our main theorem, Theorem 5 below, will follow easily:

Theorem 3. *Let L be a regular language over unranked trees.*

Then L is definable in FO iff L is aperiodic and there exists $n, k \in \mathbb{N}$ *such that L is closed under* (n, k)*-guarded swaps.*

As with the proof of Theorem 1, one direction is easy and the other follows from the following theorem.

Theorem 4. *For any regular tree language L which is aperiodic and closed under (n,k)-guarded swaps, there exists a K such that for any $s,t \in \mathcal{T}$ we have:* $\quad s \equiv^K t \quad \Rightarrow \quad s \in L$ *iff* $t \in L$.

The proof of Theorem 4 will follow along the same lines as the proof of Theorem 2, but differs in the technical details.

We fix an aperiodic regular tree language L, numbers k and n and assume that L is closed under (n,k)-guarded swaps. We also fix a deterministic bottom-up unranked tree automaton A for L. Let m be the tolerance of A. Without loss of generality we can assume that $n \geq m$.

The following is an extension to the unranked setting of a sequence of lemmas that we used in the ranked case. The first one is again immediate from the locality of FO.

Lemma 5. For each pair of numbers d, n', there exists $K_{d,n'}$ such that $s \equiv^{K_{d,n'}} t$ implies $s =_d^{n',k+1} t$ and s,t are $(n',k+1)$-similar.

Lemma 6. For each number d' and each number $n' > n$ there exists a number d such that if $s =_d^{n',k+1} t$ then there exists t' such that $s \leq_{d'}^{n',k+1} t'$, t and t' are $(n',k+1)$-similar, and $t' \in L$ iff $t \in L$.

Proof. Assume $s =_d^{n',k+1} t$ for some large enough d whose value will become apparent during the proof. Let β be the number of $(n',k+1)$-types τ such that $|t|_\tau < |s|_\tau$. We prove the lemma by induction on β. If $\beta = 0$ this is clear. Otherwise let τ be a $(n',k+1)$-type that occurs more times in s than in t. By hypothesis τ occurs at least d times in t. We say that a $(n',k+1)$-type ν is *safe* if $|t|_\nu \geq d'$. A subtree in a tree t is safe if all the nodes in it have safe types within t.

First, let d_1 be big enough so that whenever we have d_1 distinct trees then we can find n' of them and a state q of A such that all the selected subtrees are safe and A reaches state q at the root of each. Let $\#(A)$ be the number of states in A, and let d_2 be big enough so that whenever one has $(d_2/(\#(A)+1))-1$ contexts there is at least one that is safe. Such a d_1 and d_2 can be easily computed from the size of A, d', n', and k. We now claim that any d bigger than $(d_1+1)^{d_2}$ will suffice.

A τ-skeleton of length d is defined as in the ranked case. We have two cases to consider. In the first case, every node in t has at most d_1 children which have a descendant of type τ. In this case, since we have more than $(d_1+1)^{d_2}$ nodes of type τ, we can use the same proof as in the ranked case to construct a τ-skeleton of length d_2 in t. As in the ranked case, the context between x_i and $x_{i+\#(A)}$ in the τ-skeleton of t, including the top node and excluding the bottom one, is called a *section* of t. Using the same argument as in the ranked case and the definition of d_2, we can pump a portion of some τ-section in the skeleton as much as we need to get the number of nodes satisfying type τ in t to be larger than the number in s. This pumping will not change the type of any prior node, so in particular will not impact the type of the root; thus the resulting tree is $(n',k+1)$-similar to t.

In the second case, there is some node x in t that has more than d_1 children which have a descendant of type τ. By the choice of d_1 we can find a state q along with $y_1, \cdots, y_{n'}$ children of x so that for each y_i, $t|_{y_i}$ is safe, $t|_{y_i}$ contains a node of type τ, and the automaton A when run on t reaches q at y_i. Because $n' > n \geq m$, the tolerance of A, it is possible to add an arbitrary number of extra copies of any of $t|_{y_i}$ without affecting membership in L. Each copy adds at least one node of type τ, and we do this until we have enough nodes of type τ. Again it is easy to check that pre-existing types are preserved, so the resulting tree is $(n',k+1)$-similar to t. $\square$

We adapt the notion of pseudo-inclusion to the unranked case. A tree t is $(n,k+1)$-*pseudo-included* in a tree t' if there is an injective mapping h from nodes of t to nodes of t', sending the root of t to the root of t', and such that: (i) h preserves $(n,k+1)$-types, (ii) if y is a child of x in t then $h(y)$ is a descendant of a child z' of $h(x)$ in t' such that z' and y have the same (n,k)-type (notice the switch from $k+1$ to k here), and (iii) if y_1 and y_2 are distinct children of x in t then the least common ancestor of $h(y_1)$ and $h(y_2)$ in t' is $h(x)$ (the children of $h(x)$ associated to y_1 and y_2 are distinct). The *h-pseudo-tree* is the minimum subtree of t' which contains $h(t)$.

The following lemma takes care of the pseudo-inclusion step.

Lemma 7. For all d' there exists n' such that if $s \preceq_{d'}^{n',k+1} t$ and s, t are $(n', k+1)$-similar then there exists t' such that s is $(n, k+1)$-pseudo-included in t', $t' =_{d'}^{n,k+1} t$, and $t' \in L$ iff $t \in L$.

Proof. We say that a type $\tau \in UT_{n,k+1}$ is *safe* it it occurs more than d' times in t. A subtree of t is safe if it contains only safe types. Let n' be computed from d', n, k and m so that whenever one considers a collection of n' pairwise-disjoint subtrees of some tree, then there exists a state q of A and at least m of the subtrees which are safe and for which the automaton reaches state q at the root.

As in the ranked case, the proof is done by induction. We construct $t_0 \cdots t_\alpha$ such that: t_0 is t, for all $0 \leq i \leq \alpha$, $t_{i+1} \in L$ iff $t_i \in L$, if s_i is a maximal prefix of s such that s_i is $(n', k+1)$-pseudo-included in t_i then there exists a node x of s that is a child of a leaf of s_i, such that $x \in s_{i+1}$, $t_{i+1} =_{d'}^{n,k+1} t_i$ and $t_i \preceq_{d'}^{n',k+1} t_{i+1}$. Since s_i cannot keep growing forever, we must eventually have $s_n = s$. This implies the lemma by taking $t' = t_\alpha$

By hypothesis the root of s and the root of t have the same $(n', k+1)$-type. Thus we can initiate our process by mapping the root of s to the root of t.

Assume now that we have constructed t_i, and let s_i be a maximal prefix of s $(n, k+1)$-pseudo-included in t_i by the mapping h. If $s_i = s$ we are done. Otherwise let x be a node of s_i which has a child y that is not in s_i. Let s' be the prefix of s which contains s_i and y. We show how to transform t_i into t_{i+1} so that s' is pseudo-included in t_{i+1}.

Let $\tau = UT^s_{n',k+1}(x)$, $\nu = UT^s_{n',k+1}(y)$ and $x' = h(x)$. By hypothesis we know that there is a node y' in t_i outside of $h(s_i)$ such that $UT^{t_i}_{n',k+1}(y') = \nu$.

Let C be the set of children of x that have an image under h (in particular, y is not in C). Let C' be the set of children of x' having a descendant in $h(C)$. We distinguish two cases depending on whether there exists a child z' of x' which is (n', k)-similar to y' and which is not in C'.

If such a z' exists then we are in a situation similar to the ranked case and can again use a case analysis depending on the relative position of z' and y' provide a sequence of swaps placing y' below z' without affecting the current mapping h.

Unlike in the ranked case, such a z' might not exist. In this case we show that we can expand the number of children of x', without affecting membership in L and violating the induction hypothesis, introducing a node z' (n', k)-similar to y'.

Let $\mu = UT^s_{n',k}(y)$ (note that we move from $k+1$ to k, therefore ν implies μ but not conversely). Since x and x' have the same $(n', k+1)$-type, the number of children of x with (n', k)-type μ must agree with the number of children of x' of (n', k)-type μ or both numbers must be bigger than n'. Let C_μ be the subset of C consisting of children of x of type μ. Let C'_μ be the subset of C' of children of x' of type μ. By definition of $(n', k+1)$-pseudo-inclusion we have that $|C_\mu| = |C'_\mu|$. Because y is also of type μ and $y \notin C_\mu$, the total number of children of x of type μ is strictly bigger than the total number of children of x' of type μ and therefore both numbers are bigger than n'. By the choice of n', this implies that we can find m children of x' of (n', k)-type μ such that their subtrees are safe and such that the automaton A reaches the same state q on all their subtrees. By the choice of m we can arbitrarily duplicate each of these subtrees without affecting membership in L. Because of the safety assumption the resulting tree t'_i is such that $t_i =_{d'}^{n,k+1} t'_i$. Because we have only introduced new nodes in t'_i it is clear that $t_i \preceq^{n',k+1} t'_i$. This has introduced a new node z' of type μ in t'_i and we can proceed as in the first case. □

This is where the unranked case departs in organization from the ranked one. Indeed the proof of Lemma 7 used pumping arguments, while the corresponding lemma in the ranked case used only swaps. Therefore we cannot infer an unranked variant of Corollary 1. Nevertheless, the remainder of the proof will proceed in the same spirit of the ranked proof in that extra nodes left over in t from the image of s under a pseudo-inclusion will be removed. Since we cannot appeal to Corollary 1 as a stopping condition, we will have to preserve the pseudo-inclusion as we perform this removal.

We extend the notion of abstract context to the unranked case. Consider a context C whose root is not equal to its port and a function λ assigning a (n, k)-type to the port of C and a $(n, k+1)$-type to each other node of C. We say λ is *consistent* if there exists a a tree t such that in $C \cdot t$, for every non-port node x in C, the $(n, k+1)$-type of x in $C \cdot t$ matches $\lambda(x)$, and the (n, k)-type of the port x in $C \cdot t$ also matches $\lambda(x)$. An *abstract* $(n, k+1)$*-context* is a context C supplemented with a consistent assignment λ. We will drop $(n, k+1)$ when it is clear from context, referring simply to an abstract context.

The notions of *compatibility* and *loop* are extended to unranked abstract contexts in the obvious way. Given a tree t and an abstract context U, the notion of U being (n, k)*-included* in t is defined as expected, as a mapping which not only preserves the (n, k)-types but also the child relation.

As in the ranked case we will need lemmas that allow us to tightly embed an abstract context into a given tree, so that we can apply aperiodicity to remove them. To do this embedding we need the following technical lemmas.

Given an (n, k)-type τ, a node x satisfying τ in a forest U is said to be (n, k)*-thin* if its k-spill can be embedded into the k-spill of all nodes of type τ. We have that for each $\tau \in U\mathcal{T}_{n,k}$ there is a tree whose root satisfies τ and is (n, k)-thin: choose the k-spill of the root such that when we calculate the (n, k)-type, we never go beyond n when counting the number of occurrences at each depth. A non-port node x in an abstract context of type τ is said to be (n, k)-thin if for any node x' of a tree t with type τ and (n, k)-type ρ, the number of children of x assigned to ρ is no greater than the number of children of x'a satisfying ρ.

Lemma 8. Let U be an abstract context. If $U <_{n,k+1} t$ and each node $x \in U$, is $(n, k+1)$-thin, then there exists t' such that U is $(n, k+1)$-included in t', $t' =^{n,k+1}_{\infty} t$ and, $t' \in L$ iff $t \in L$.

Proof. As was the case for the ranked variant of this lemma, Lemma 4, the first step in the proof is a weak pseudo-inclusion of U in t, which is obtained by a refinement of Lemma 7. Recall that weak pseudo-inclusion is a pseudo inclusion that do not require that the root is mapped to the root.

Claim 2. *If $U <_{k+1} t$ and every node $x \in U$ is $(n, k+1)$-thin, then there exists a tree t'' such that U is weakly $(n, k+1)$-pseudo-included in t'', $t'' =^{n,k+1}_{\infty} t$, and, $t'' \in L$ iff $t \in L$.*

Proof. The proof is done exactly as in the proof of Lemma 7 with some small differences. See also how Claim 1 was proved using a modification of the proof of Lemma 3. Let a be the root of U. In the base case, the image of the root of U is now an arbitrary node of $t = t_1$ whose type agrees with the type of a; an inclusion mapping does not demand preservation of the root. During the induction we have constructed t_i and a prefix U_i of U which is weakly $(k+1)$-pseudo-included in t_i.

Recall the proof of Lemma 7. We have two nodes $x, y \in s$ such that y is a child of x and is of type ν. We also have two nodes nodes $x', y' \in t_i$, such that $x' = h(x)$ and y' is a node outside of $h(U_i)$ of type ν. We are trying to modify t_i in order to put a node of type ν below x'.

In the proof of Lemma 7, we distinguished two cases depending on whether there exists a child z' of x' which is (n, k)-similar to y' and which is not in $h(U_i)$. As we now have that x is thin, x' has at least as many children as x for each type, and therefore such a z' always exists. We are therefore in a situation similar to the ranked case and we can perform a case analysis depending on the relative position of x', y' and z' in order to place a node of type ν below z' using only swaps. As no pumping is necessary, we will eventually also have $t'' =^{n,k+1}_{\infty} t$.

The case analysis is done as in the proof of Lemma 7 and, as in the proof of Claim 1, one extra case needs to be considered when y' is above $h(a)$. This case is treated as in the proof of Claim 1. $\square$

The rest of the proof proceeds exactly as in the ranked case in the proof of Lemma 4. We show that the weak pseudo-inclusion h obtained in Claim 2 can be transformed into a real inclusion step by step. Consider $x, y \in U$ such that y is a child of x and assume that $y' = h(y)$ is not a child of $x' = h(x)$. We also know by assumption that there is a node y'' outside of $h(U)$ which has the same type as y'.

Again, the fact that x is thin implies that there is a child z' of x' which is (n,k)-similar to y'. We are therefore in a situation similar to the ranked case and we can perform the same case analysis depending on the relative position of x', y', y'' and z' in order to replace z' by y' using only swaps and without modifying the rest of the pseudo-inclusion mapping h. The reader is now referred to the proof of Lemma 4. $\square$

We also need a version of this lemma for forests.

Given a tree t and a forest U, the notion of U being (n,k)-*included* in t is defined by the existence of an injective mapping which preserves (n,k)-types and the child relation. For a forest U and a tree t, we say that $U <_{n,k} t$ if every (n,k)-type occurring in U occurs strictly more often in t.

The same argument as Lemma 8 shows:

Lemma 9. Let U be a forest. If $U <_{n,k+1} t$ and each node $x \in U$ is $(n,k+1)$-thin, then there exists t' such that U is $(n,k+1)$-included in t', $t' =^{n,k+1}_{\infty} t$ and, $t' \in L$ iff $t \in L$.

We are now ready for:

Proof of Theorem 4. Let m be the tolerance of A. Let $d' = (m+l) * \alpha + 1$ where α is spelled out in the proof below. Let n' be the number required in Lemma 7 from d', n and k, and d be the number required in Lemma 6 from d' and n'. Let K be the number $K_{d,n'}$ required in Lemma 5 for d.

We show that $s \equiv^K t$ implies $s \in L$ iff $t \in L$. Assume $s \equiv^K t$. From Lemma 5 we know that $s =^{n',k+1}_{d} t$ and that s and t are $(n',k+1)$-similar. By the choice of d we can apply Lemma 6 and construct t' such that $t' \in L$ iff $t \in L$, $t' \preceq^{n',k+1}_{d'} s$, and s and t' are $(n',k+1)$-similar. Now we can apply Lemma 7 and obtain t'' such that $t'' \in L$ iff $t' \in L$, s is $(n,k+1)$-pseudo-included in t'', and $t'' =^{n,k+1}_{d'} t' =^{n,k+1}_{d'} s$. We show that $s \in L$ iff $t'' \in L$.

As in the ranked case we observe that by definition of h, t'' is $h(s)$ plus *abstract loops* inserted between nodes of $h(s)$ and extra *branches* branching off the h-pseudo-tree. The crucial observation is again that all $(n,k+1)$-types which occur outside of $h(s)$ have strictly more occurrences in t'' than in s and therefore appear at least d' times in s. The rest of the proof transforms t'' without affecting membership in L in order to remove all the extra material.

We first show how to remove an extra branch. Let x' be a node of $h(s)$ and y' be a child of x' such that $V = t''|_{y'}$ is disjoint from $h(s)$. Let t_1 be the tree constructed from t'' by removing V. We show that $t_1 \in L$ iff $t'' \in L$, thus we can proceed with t_1 replacing t''. By repeating this argument we can assume that t'' does not contain any extra branch.

We now prove that $t_1 \in L$ iff $t'' \in L$. Let τ be the (n,k)-type of y' and let C_τ be the set of children of x' of type τ in the h-pseudo-tree. Because h preserves $(n,k+1)$-types, C_τ is of cardinality at least n. As in the ranked case, using simple pumping arguments, we can, without affecting membership in L, replace V by a tree V' having the following properties: (i) V' and V evaluate to the same state of

A, (ii) V' has exactly the same $(n, k+1)$-types as V, (iii) V' has all its nodes $(n, k+1)$-thin, and (iv) the number of nodes of V is bounded by a constant α easily computable from n, k and A. Let U be the forest consisting of m copies of V'. Notice that all types occurring in V' occur more frequently in t'' than in s, hence occur at least d' times in s and thus in $h(s)$. Since t_1 still contains $h(s)$, we have that all the types in U occur at least d' times in t_1. By the choice of d', which is strictly greater than $|U|$, we have $U <_{n,k+1} t_1$. Thus we can apply Lemma 9 to U and t_1, obtaining a tree t_2 with m copies of V' $(n, k+1)$-included in it as children of a node z. By Lemma 9 we have $t_1 \in L$ iff $t_2 \in L$. As t'' is t_1 plus V, each operation which transformed t_1 into t_2 can be applied to t'' yielding a tree t_3 such that t_3 is t_2 plus V hanging from node x'. Moreover we have $t_3 \in L$ iff $t'' \in L$. It remains to show that $t_2 \in L$ iff $t_3 \in L$. Recall now that $n > m$ and therefore there exist m siblings of V of type τ. We can then perform (n, k)-guarded swaps in t_3 in order to place the m copies of V' as siblings of V. We thus have $m+1$ siblings evaluating to the same state below x' and thus V can be removed without affecting membership in L. From the tree we just obtained we get t_2 by reversing the last m swaps, showing the desired property.

We now show how to remove an abstract loop in t'' using a similar technique. Let V be an abstract loop in t''. Let t_1 be the tree constructed from t'' by removing V. We show that $t_1 \in L$ iff $t'' \in L$, thus we can proceed with t_1 replacing t''. By repeating this argument we eventually derive that $t'' = s$ showing that $t'' \in L$ iff $s \in L$. Let V' be an abstract loop obtained from V satisfying the properties (ii),(iv) listed in the branch case above together with: (iii') every non-port node is $(n, k+1)$-thin, and (i') V and V' induce the same transition function for A; this abstract loop can be found as in the ranked case. Apply Lemma 8 for the abstract context $U = (V')^l$ and t_1 yielding a tree t_2. Again as t'' is t_1 plus V, the same operations that transformed t_1 into t_2 can be applied in order to transform t'' into t_3. It remains to show that $t_2 \in L$ iff $t_3 \in L$. From t_3, an extra swap appends V after the sequence of l copies of V' and therefore V can be removed without affecting membership in L by applying aperiodicity, yielding t_2. □

The k-guarded swaps are defined as in the ranked case, but requiring isomorphism of the k-spill. We will now show the main result:

Theorem 5. *Let L be a regular language over unranked trees. Then L is definable in FO iff L is aperiodic and there exists $k \in \mathbb{N}$ such that L is closed under k-guarded swaps.*

The theorem will follow immediately from Theorem 3 and the following proposition:

Proposition 1. *For every regular language L and every k there is a number n such that if L is closed under k-guarded swaps, then it is closed under (n, k)-guarded swaps.*

Proof. To prove the proposition, we first show the following claim:

Claim 3. *For every regular language L there is a number n such that for every k if tree t_1 is (n, k)-similar to tree t_2, then there are trees t'_1, t'_2 with t'_1 depth-k similar t'_2 such that A reaches the same state on t'_i as on t_i, for $i = 1, 2$.*

Proof of the claim. Let m be the tolerance of A, and $n = m|Q|$. Notice that n is defined so that if there are n nodes, at least m of them will have the automaton A reach the same state. We show by induction on k that n suffices. For $k = 0$ it is clear. To prove this for $k+1$, consider $(n, k+1)$-similar trees t_1, t_2. Let S be the set of (n, k)-types of the children of the root of t_1 and the children of the root of t_2. For each $\tau \in S$, let $t_1(\tau)$ be the number of children of the root of t_1 that satisfy τ and $t_2(\tau)$ be the number of children of the root of t_2 that satisfy τ. If for every τ we have $t_2(\tau) = t_1(\tau)$. then we can let $t'_1 = t_1$, $t'_2 = t_2$ and we are done. For each τ for which this is not the case, we know that

$t_2(\tau) > n$ and $t_1(\tau) > n$. Assume without loss of generality that $t_2(\tau) > t_1(\tau)$. By the definition of m, there are at least m children of the root of t_1 satisfying τ for which the automaton reaches the same state. Pick one of these nodes x and add $t_2(\tau) - t_1(\tau)$ many copies of the subtree of x as children of the root of t_1. Since m is the tolerance of A, this does not affect the state of the run of A at the root of t_1. Applying induction, we can change t_2 and t_1 without affecting the state at the root so that all the children of the root of t_2 which satisfy τ are depth-k similar to one another, and each of these are depth-k similar to the children of the root of t_1 which satisfy τ. Doing this for each $\tau \in S$, we end up with trees t'_1 and t'_2 with the property that: for every $\tau \in S$ all the children of the root of t_1 with type τ are depth-k similar to one another, they are all depth-k similar to every child of the root of t'_2 with type τ and the number of such children is the same in t'_1 and t'_2. It is clear that t'_1 and t'_2 are depth-$(k+1)$ similar. □

Given the above claim, we show how Proposition 1 follows. Suppose L is closed under k-guarded horizontal swaps. We show that L is closed under (n,k)-guarded horizontal swaps for n given by the claim. Given t_1 (n,k)-similar t_2 , let t'_1, t'_2 be as in the claim above. We know that:

$T[t_1,t_2] \in L \leftrightarrow T[t'_1,t'_2] \in L \leftrightarrow T[t'_2,t'_1] \in L \leftrightarrow T[t_2,t_1] \in L$

The first and third equivalences follow because t'_i and t_i are equivalent in the automaton, and the middle equivalence is from closure under k-guarded swaps.

The proof for vertical swapping is similar; instead of the claim above, we show that: For every regular language L there is a number n such that for every k if context C_1 is (n,k)-similar to context C_2, then there are contexts C'_1, C'_2 with C'_1 depth-k similar C'_2 such that C'_i induces the same state function as C_i for $i = 1, 2$.

Here two contexts are (n,k)-similar if they are (n,k)-similar as trees. The extension of the inductive argument for this is left to the reader. □

5 Decidability

We first show that the characterizations of Theorem 1 and Theorem 5 are generalizations of the string case. Then we show that they lead to decision procedures for membership in FO and FO$_{mod}$.

5.1 The string case

We view a string as a tree in which every node has at most one child. The child corresponds to the successor of a node in a string. With this kind of tree, only the vertical swap can be applied. Theorems 1 and 5 imply that a regular language is definable in FO iff it is aperiodic and closed under k-guarded swaps for some k.

In the string case a similar characterization of FO says that a regular language is definable in FO iff it is aperiodic and closed under *idempotent-guarded swaps*. This result was proved by Beauquier and Pin [BP89]. We will show how this can be derived from our characterization.

We define the notion of *idempotent-guarded swaps*. Fix a regular (string) language L and a minimal deterministic automaton A recognizing L. Recall that a function f is *idempotent* if $f \circ f = f$. A string e is said to be *idempotent* if the transition function it induces is idempotent. A regular string language L is *closed under idempotent-guarded swaps* if for any string of the form $uesfves'fw$, where e and f are idempotents and different from the empty string, we have

$$uesfves'fw \in L \text{ iff } ues'fvesfw \in L.$$

We show that the two notions of guarded swaps are equivalent over strings.

Theorem 6. *A regular language L over strings is closed under idempotent-guarded swaps iff it is closed under k-guarded swaps for some $k \in \mathbb{N}$.*

Proof. One direction is simple: if L is closed under k-guarded swaps then it is closed under idempotent-guarded swaps. Consider a string of the form $uesfves'fw$, where e and f and non-empty idempotents. Then we have:

$$uesfves'fw \in L \quad \text{iff} \quad ue^k sf^k ve^k s' f^k w.$$

Notice now that the two positions right after u and right after v are depth-k similar, and the same holds for the two positions right after s and right after s'. We can then apply k-guarded swaps and get

$$uesfves'fw \in L \quad \text{iff} \quad ue^k s' f^k ve^k sf^k w \ \text{ iff } \ ues'fvesfw \in L.$$

We now turn to the other direction. Assume L is regular, let A be a deterministic automaton for L, and assume L is closed under idempotent-guarded swaps. Let α be the number of states of A and take $k = \alpha^\alpha + 1$. We show that L is closed under k-guarded swaps. Recall that a string w induces a transition function f_w^A on the states of A such that $f_w^A(q) = q'$ if, when started in state q, q' is the state reached by A at the end of w. Two strings w and w' are *equivalent relative to* L if $f_w^A = f_{w'}^A$.

Consider a string w of length greater than k. For $i \leq k$ let v_i be the first i letters of w. By the choice of k there must be $i < j < k$ such that $f_{v_i}^A = f_{v_j}^A$. Let u be the string such that $v_j = v_i u$. Because $f_{v_i}^A = f_{v_j}^A$ we have for every positive integer β, $v_i u^\beta$ and v_j are equivalent relative to L. Notice now that for all strings u there exists β such that u^β is idempotent. Hence in any string of length at least k there is an idempotent e that can be inserted without affecting membership in L.

Now consider a string w and positions x, y, x', y' such that $x < y < x' < y'$ and x, x' are depth-k similar, and y, y' are depth-k similar. First, consider the case in which y is not in the k-spill of x, x' is not in the k-spill of y, and y' is not in the k-spill of x'. Thus w can be decomposed into $w_1 \cdot sv \cdot s'w_2 \cdot sv' \cdot s'w_3$ where s and s' are the k-spill s of x and y and the "dots" mark the position of x, y, x', y'.

We can now apply the technique mentioned in the paragraph above and insert an idempotent e into s and an idempotent f into s' without affecting membership in L. Thus we have

$$w \in L \quad \text{iff} \quad w_1 \cdot s_1 e s_2 v \cdot s_1' f s_2' w_2 \cdot s_1 e s_2 v' \cdot s_1' f s_2' w_3 \in L.$$

We now use the closure of L under idempotent-guarded swaps and obtain (the "dots" now indicate the sections that are swapped):

$$w_1 s_1 \cdot e s_2 v s_1' f \cdot s_2' w_2 s_1 \cdot e s_2 v' s_1' f \cdot s_2' w_3 \in L \quad \text{iff} \quad w_1 s_1 \cdot e s_2 v' s_1' f \cdot s_2' w_2 s_1 \cdot e s_2 v s_1' f \cdot s_2' w_3 \in L.$$

But the latter is precisely $w_1 \cdot sv' \cdot s'w_2 \cdot sv \cdot s'w_3$ as required for k-guarded swapping.

The other cases are handled similarly. We consider the case where y is in the k-spill s of x, x' is not in the k-spill s' of y, and y' is not in the k-spill of x'. Thus w can be decomposed into $w_1 \cdot s_1 \cdot s'w_2 \cdot sv' \cdot s'w_3$ with $s' = s_2 s_3$ and $s = s_1 s_2$.

By the argument above, we know that there is an idempotent f that can be inserted into s' without impacting membership in L. We have two subcases to consider depending whether f falls in s_2 or in s_3. If f can be inserted in s_2 then we write s_2 as $t_1 \cdot t_2$, where $t_1 f t_2$ is equivalent relative to L with $t_1 t_2$. We thus have a decomposition of w as $w_1 \cdot s_1 \cdot t_1 t_2 s_3 w_2 \cdot s_1 t_1 t_2 v' \cdot t_1 t_2 s_3 w_3$.

Using the fact that f can be inserted, we see that:

$$w \in L \quad \text{iff} \quad w_1 s_1 t_1 \cdot f t_2 s_3 w_2 s_1 t_1 f \cdot f t_2 v' t_1 f \cdot f t_2 s_3 w_3 \in L.$$

Applying idempotent-guarded swapping to the blocks between copies of f we get:

$$w \in L \quad \text{iff} \quad w_1 s_1 t_1 \cdot f t_2 v' t_1 f \cdot f t_2 s_3 w_2 s_1 t_1 f \cdot f t_2 s_3 w_3 \in L.$$

Removing f and regrouping we get:

$$w \in L \quad \text{iff} \quad w_1 \cdot s_1 t_1 t_2 v' \cdot t_1 t_2 s_3 w_2 \cdot s_1 \cdot t_1 t_2 s_3 w_3 \in L.$$

This shows that guarded swapping holds.

In the subcase where f can be inserted into s_3, we know there is another idempotent e that can be inserted into $s = s_1 s_2$ prior to the place where f is inserted. We now write w as $w_1 \cdot t_1 e t_2 t_3 f t_4 w_2 \cdot t_1 e t_2 v' \cdot t_3 f t_4 w_3$, and again the a swap gives the desired result.

The rest of the cases are treated similarly. $\square$

Note that Theorem 6 does not generalize to trees. In the tree case it is still true that any k-spill, for a sufficiently large k, will contain an idempotent, but which idempotent this is and where it can be inserted can no longer be computed by looking only at the k-spill.

5.2 Decision Procedure

Let L be a regular tree language, A be a deterministic bottom-up automaton (ranked or unranked) recognizing L, and let Q be the set of states of A. In this subsection we will consider the problem of deciding whether L is in FO. The input of the problem is A, and thus the complexity is relative to the size of A. Without loss of generality, A can be taken to be minimal, since a number that is polynomial in the size of a minimal automaton is clearly polynomial in the size of any automaton.

In the string case deciding whether a regular language L can be defined in FO is PTIME in the size of such an A [Pin05]. Note that this is not immediate, as checking aperiodicity alone is PSPACE-complete [CH91]. It turns out that ideas similar to [Pin05] show that membership in FO can actually be checked in PTIME also in the tree setting. We will show this only for the ranked case; the unranked case is proved along the same lines.

We will first show that the aperiodicity condition in Theorem 1 can be replaced by one that is easier to check. Following the approach of [Pin96]: we replace aperiodicity by the condition $(\dagger) l$: there exists l such that for any contexts s, x, y, any context e that generates an idempotent function on states of the minimal automaton, and any tree s', we have

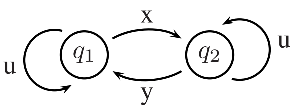
$$s \cdot (e \cdot x \cdot e \cdot y \cdot e)^l \cdot s' \ \in L \quad \text{iff} \quad s \cdot (e \cdot x \cdot e \cdot y \cdot e)^l \cdot e \cdot x \cdot e \cdot s' \ \in L.$$

In [Pin05] it is shown that for a regular string language that is closed under idempotent-guarded swaps, aperiodicity is equivalent to $(\dagger) l$. More precisely, [Pin05] shows that in any finite monoid satisfying $e \cdot u \cdot f \cdot s' \cdot e \cdot v \cdot f = e \cdot v \cdot f \cdot s' \cdot e \cdot u \cdot f$ for idempotents e, f and arbitrary monoid elements u, s', v, we have the following are equivalent: a) there is l such that the identity $u^l = u^{l+1}$ holds and b) there is l such that

$$(e \cdot x \cdot e \cdot y \cdot e)^l = (e \cdot x \cdot e \cdot y \cdot e)^l \cdot e \cdot x \cdot e$$

holds (where again e, f range over idempotents and x, y, u, v, s' range over monoid elements). We apply these results to the monoid of contexts. Observe that the proof of Theorem 6 shows that if a tree language is closed under k-guarded vertical swaps, then it is closed under idempotent-guarded vertical swaps (the converse holds for string languages but not for tree languages). Applying this observation, Theorem 1, and the result of [Pin05] cited above, we see that a regular tree language is definable in FO iff there exists k such that it is closed under k-guarded swapping and there exists l such that $(\dagger)l$ holds.

We will now show that one can decide in PTIME whether or not a regular tree language satisfies $(\dagger)l$. Our argument will rely on the notion of graph pattern matching, which we review here. For the purposes of this section, a *pattern* is a graph whose edges are labeled by variables which range over elements of Γ^* for some finite alphabet Γ. In addition a pattern comes with side conditions stating which nodes of the pattern should be interpreted as distinct nodes. Let G be a graph whose edges are labeled in Γ. Such a graph G *matches* a pattern if there is a mapping f taking each variable in the pattern to a string in Γ^+ and each node of the pattern to a node of G such that for each side constraint $p_1 \neq p_2$, $f(p_1) \neq f(p_2)$, and such that whenever there is an edge from p_1 to p_2 in the pattern labeled with v, there is a path from $f(p_1)$ to $f(p_2)$ in G whose labels yield the string $f(v)$. In [CPP93] it is noted that for every fixed pattern, the problem of determining given a graph, whether the graph matches the pattern, is in PTIME. This result was used to show that FO-definability is in PTIME in the string case. From a minimal automaton A recognizing L one constructs an edge-labeled graph $G_A = (V_A, E_A)$ as follows. The vertex set V_A of G_A is the set of states of A. The transitions $E_A \subseteq V_A \times \Sigma \times V_A$ are labeled with letters of the alphabet Σ of L and correspond to the transitions of A.



Let P be the pattern depicted above together with the condition $q_1 \neq q_2$. It has been shown that [CPP93] L verifies $(\dagger)l$ for some l iff G_A does not match P. Minimality of A is used in the left to right direction.

This result extends to trees as follows. From an automaton A define $G_A = (V_A, E_A)$ as follows. The set of vertices V_A is Q. The set of edges E_A is included in $V_A \times \Lambda \times V_A$ where $\Lambda = \Sigma \times \bigcup_{i<r} Q^i \times \{1 \dots r\}$ where r is the rank. We connect a node p to a node p' via an edge (a, λ, j), where λ is a sequence of (at most) r states, if $\delta(\lambda, a) = p'$ where the j^{th} state of λ is p. That is, an edge represents the inverse of a transition of the automaton. The same proofs as in [Pin05, CPP93] show that:

Lemma 10. L verifies $(\dagger)$l for some l iff G_A does not match P.

Therefore we have:

Proposition 2. *One can decide in* PTIME *(in the size of a deterministic bottom-up automaton) whether or not a regular language L satisfies $(\dagger)l$ for some l.*

We now show that it is decidable in PTIME whether there exists k such that L is closed under k-guarded swaps. We first show that if L is closed under k-guarded swaps then it is closed under k'-guarded swaps where k' is computable from A.

Lemma 11. Let L be a regular language. Let α be the number of states of an automaton for L and $k' = \alpha^2 + 1$. Then L is closed under k-guarded swaps for some k iff it is closed under k'-guarded swaps.

Proof. We show that if L is closed under k-guarded horizontal swaps for any k then it is closed under k'-guarded horizontal swaps. The proof for vertical swaps is similar and is left to the reader.

Let t be a tree and x, y two nodes of t that are depth-k' similar. We first show that we can transform t by pumping in new subtrees so that x, y become k-similar in the new tree, without affecting membership in L. Let $t_1 = t|_x$ and $t_2 = t|_y$. Let $f(t, x, y)$ be the lex-minimum of pairs of integers (n, m) such that there is a leaf node z in the common prefix of t_1 and t_2 such that its level is n and its position is m among nodes of level n: we know that n is no smaller than k'. We will transform t by pumping so that $f(t, x, y)$ increases.

Let P be the common prefix of t_1 and t_2, w be a leaf node in P witnessing $f(t, x, y)$. Consider a run r of A on t. The run assigns a state q to each node of P when running up both t_1 and t_2. We assign to each node of P the pair of states (q, q') such that q is the state of r at the corresponding node in t_1 while q' is the state of r at the corresponding node in t_2. By the choice of k', on the path from the root to w, there must be a pair of states that repeat. Let z and z' be two such nodes. Consider the context C_1 between the copy of z and z' within t_1, including z and replacing z' by a port. Without affecting membership in L, we can duplicate C_1 in t_1 as many times as we wish. Considering the context C_2, between the copy of z and z' within t_2 we can perform the same duplication within t_2. Performing both these duplications, we have now increased the depth of w to be above k without removing any nodes from P, thus increasing $f(t, x, y)$.

Performing this repeatedly, we arrive at a tree t' obtained from t by adding new sections such that $f(t', x, y) = (k, l)$ for a given k. We can now apply k-guarded swapping to switch the subtrees under x and y in t'. We can now remove the extra sections, resulting in a tree formed from the original tree t by swapping the subtrees under x and y, as required. □

From the above it is already clear that one can decide whether L is closed under k-guarded swaps for some k, since one needs to check only that L is closed under k'-guarded swaps, and for a fixed k' checking closure under swaps is easily seen to be decidable. Below we will show a stronger result:

Theorem 7. *There is a* PTIME *algorithm that decides, given a deterministic automaton for a regular language L and a positive integer k, whether or not L is closed under k-guarded swaps.*

We will now prove Theorem 7. We first show that deciding closure under horizontal swapping is in PTIME.

We first note that:

Claim 4. *There is a* PTIME *algorithm that, given a deterministic bottom-up automaton A, positive integer k, and states q_1, q_2 of A, determines whether or not there exist k-similar trees t_1, t_2 such that when A is run on $t_i : i = 1, 2$ it leads to state q_i at the root of t_i. We write $q_1 \sim_k q_2$ when this occurs.*

Proof. The algorithm is simple induction, since $p \sim_k q$ iff there is an alphabet symbol a number $r' \leq r$ and states $p_1 \dots p_{r'}, q_1 \dots q_{r'}$ with $p_i \sim_{k-1} q_i$ for each $i \leq r'$ such that A transitions on a from $p_1 \dots p_{r'}$ to p and from $q_1 \dots q_{r'}$ to q. □

From Claim 4, we can derive our first result:

Lemma 12. Verifying that the language of trees given by a deterministic automaton A is closed under k-guarded horizontal swapping is in PTIME in $|A|$ and k.

Proof. It suffices to show this for a minimal automaton. We prove this in the case of binary trees. The extension to arbitrary ranked trees and to unranked trees is left to the reader.

A 2-context is a tree with two distinguished leaves called ports. An automaton A and a 2-context Δ induce a function Δ_A from $Q \times Q$ to Q simulating a bottom-up evaluation.

By minimality and determinism of A, we need to check that for all states q_1, q_2, if $q_1 \sim_k q_2$ then for every 2-context Δ we have $\Delta_A(q_1, q_2) = \Delta_A(q_2, q_1)$.

From Claim 4 the lemma follows assuming that we have shown: given two states q_1 and q_2 such that $q_1 \sim_k q_2$, we can check in PTIME that a given 2-context Δ satisfies $\Delta_A(q_1, q_2) = \Delta_A(q_2, q_1)$.

Consider the set Z consisting of the sextuples of states $(p, p_1, p_2, q, q_1, q_2)$ such that for some context Δ, $p = \Delta_A(p_1, p_2)$ and $q = \Delta_A(q_1, q_2)$. Z can be computed easily by a fixpoint algorithm. To check the property above, we need only determine whether there is a state p such that $(p, q_1, q_2, p, q_1, q_2)$ is in Z: this can be done by a single iteration over Z. Hence the whole process can be done in PTIME. □

We now turn to deciding closure under vertical swaps, which is checked using similar ideas.

We need the following two relations among states of A. We say that $R_k(r, p_1, q_1, p_2, q_2)$ holds whenever there exists a context Δ and a tree t such that Δ and t agree up to depth k, $\Delta_A(p_1) = q_1$, $\Delta_A(p_2) = q_2$ and r is the state reached by A on t. We say that $S_k(p_1, q_1, p_2, q_2, p'_1, p'_2, q'_1, q'_2)$ holds whenever there exist two contexts Δ and Δ' such that Δ and Δ' agree up to depth k, and $\Delta_A(p_i) = q_i$ and $\Delta'_A(p'_i) = q'_i$ for $i = 1, 2$.

Claim 5. *For each $k \in \mathbb{N}$, both R_k and S_k are computable in* PTIME*.*

Proof. This is done as in Claim 4, with the extra reachability constraints being verifiable in PTIME. □

We are now ready to show:

Lemma 13. There is an algorithm deciding whether or not a language of r-ranked trees given by automaton A is closed under k-guarded vertical swapping which takes polynomial time in $k, |A|$.

Proof. Immediate from Claim 5. k-guarded vertical swapping fails if we can find $(r, p_1, q_1, p_2, q_2) \in R_k$ and $(r, p_1, q_2, s', q_1, s, r, p_2) \in S_k$ with $s \neq s'$. This can be checked in PTIME once S_k and R_k are computed. □

From Lemma 12, Lemma 13, we have Theorem 7.

From Lemma 11, Theorem 7, and Proposition 2, we conclude the following:

Theorem 8. *There is a* PTIME *algorithm that takes a deterministic automaton for a regular language of ranked trees and decides whether or not it is definable in FO.*

The theorem also applies to unranked trees, via easy extensions of Lemma 11, Theorem 7, and Proposition 2.

6 Modulo counting

In this section we extend the previous results to FO_{mod}. We prove that whether a regular language is definable in FO_{mod} is decidable, using Corollary 1 and an idea from [B07]. We also mention a characterization that can be used to give a PTIME algorithm for a decision procedure. We deal here only with the case of trees of fixed rank r, although the results can be extended to the unranked setting.

In [Str94] (See VII.3.1), the result of [BP89] is extended to a characterization of $\text{FO}_{mod(P)}$ on strings, by simply replacing the aperiodicity of L with a *periodicity* condition - that is, that the monoid associated with L is q-periodic. When we apply this to the monoid of contexts generated by a language, we get that a regular tree language L is q-periodic if:

(q-periodicity) $\quad \exists l$ such that $\forall s, u$ contexts, and $\forall t$ tree, $s \cdot u^l \cdot t \ \in L$ iff $s \cdot u^{l+q} \cdot t \ \in L$.

Fix $P \subseteq \mathbb{N}$ finite and let q be the least common multiple of all numbers occurring in P. We show that membership in FO$_{mod(P)}$ is decidable.

We start with giving the notion of locality relevant to FO$_{mod}$. For numbers k and q, and trees s and t, we say $s \cong_{k,n,q} t$ if for every k-type τ, $|s|_\tau = |t|_\tau$ mod q and $|s|_\tau = |t|_\tau$ if either $|s|_\tau < n$ or $|t|_\tau < n$.

It is well-known that an FO$_{mod}$ sentence on bounded-degree structures can only count the number of local neighborhoods up to some modulus and threshold (see, for example, [N00] Theorem 3.4). Applying this within ranked trees, one easily obtains the following:

Proposition 3. *For any fixed rank r and any* FO$_{mod(P)}$ *sentence ϕ, there are numbers k and n computable from ϕ and r such that ϕ cannot distinguish two r-ranked trees s and t with $s \cong_{k,n,q} t$.*

Lemma 14. Let L be a regular tree language over r-ranked trees definable in FO$_{mod(P)}$. Then L is q-periodic and there exists a k such that L is closed under k-guarded swaps.

Proof. Fixing k as in Proposition 3, one sees that L is closed under k-guarded swaps, since these preserve the number of k-types.

The proof that q-periodicity is necessary is done as in [Str94]: one shows by structural induction that all FO$_{mod(P)}$ formula are q-periodic where free variables are treated as sentences in a product alphabet. The base case of atomic formulas follows from the result for FO, while the inductive cases are already shown in the proof of VII.3.1 of [Str94]. ☐

We are now ready for the main result:

Theorem 9. *There is a computable function which takes as input a regular language L of r-ranked trees (presented effectively, for example by a tree automaton) and determines whether or not L is definable in* FO$_{mod(P)}$.

Proof. Let L be a regular tree language. Our decision procedure works as follows. We first check that L is q-periodic and also whether there is a k such that L is closed under k-guarded swaps. Since q-periodicity requires only checking an identity on each class of a computable finite monoid, the first property is decidable. By Theorem 7 and Lemma 11 the second is decidable as well. If this check fails we know from Lemma 14 that L is not definable in FO$_{mod(P)}$.

Because L is closed under k-guarded swaps, we know from Corollary 1 that membership in L depends only on the number of $(k+1)$-types present in the tree. Let n be the number of $(k+1)$-types and fix a linear order $\tau_1, \cdots, \tau_n$ among the $(k+1)$-types. Given a tree t, let $\xi_k(t) \in \mathbb{N}^n$ be the n-tuple $(i_1, \cdots, i_n)$ where $i_l = |t|_{\tau_l}$, the number of occurrences of τ_l in t. Note that when $k = 0$, $\xi_0(t)$ is simply the number of occurrences of each label. Restating Corollary 1, we have that for all trees t and t', $\xi_k(t) = \xi_k(t')$ implies $t \in L$ iff $t' \in L$.

Given a regular tree language M, we denote by $\xi_k(M)$ the set $\{\xi_k(t), t \in L\}$. A vector set $X \subseteq \mathbb{N}^n$ is called *semi-linear* if it is a finite union of sets of the form:

$$\{\bar{x} + i_1\bar{x}_1 + \cdots + i_l\bar{x}_l \mid i_1, \cdots i_l \in \mathbb{N}\}$$

for some $x_1, \cdots x_l \in \mathbb{N}^n$.

A simple extension of Parikh's theorem shows that for any regular tree language M, $\xi_k(M)$ is semi-linear and can be computed from M.

Lemma 15. For any regular tree language M and $k \in \mathbb{N}$, $\xi_k(M)$ is a semi-linear set, whose representation as a union of constraints is computable from M.

Proof. Let T_k be the set of $(k+1)$-types. Given a tree t with labels in Σ we define $[t]_k$ with labels in T_k as the relabeling of t replacing for any node x of t its label by is $(k+1)$-type.

Let $M_k = \{[t]_k \mid t \in M\}$. As $(k+1)$-types can be computed by a tree automaton, M_k is a regular tree language. Given a non-deterministic top-down tree-automaton A, we can create a context free-grammar G_A whose words are in one-to-one correspondence with the trees of A, with occurrences of letters in a word matching the occurrences of labels in the corresponding tree. G_A will have non-terminals for each state of A. Each transition $q \rightarrow_a q_1 \dots q_r$ of A will correspond to a production $q \rightarrow aq_1 \dots q_r$ of G_A, while each final state q of A will generate a production $q \rightarrow \epsilon$.

We apply this to the automaton for M_k, generating a context-free grammar. By Parikh's Theorem [Par66], we conclude that the tuple of word counts for words generated by the grammar, which is exactly $\xi_0(M_k)$, is semi-linear, with a set of linear constraints representing it being computable from the grammar. □

Let $\vec{x}, \vec{y} \in \mathbb{N}^n$. For $d \in \mathbb{N}$, we denote by $\vec{x} \sim_q^d \vec{y}$ the fact that for each component m, either $\vec{x}.m = \vec{y}.m$ or both $\vec{x}.m$ and $\vec{y}.m$ are above d and $\vec{x}.m = \vec{y}.m$ modulo q.

Using Lemma 15, we compute $S = \xi_k(L)$ and $\bar{S} = \xi_k(\bar{L})$, where $\bar{L}$ is the complement of L. The key point for getting decidability is the following lemma:

Lemma 16. Let L be q-periodic and closed under k-swaps. Then L is in FO$_{mod(P)}$ iff there exists d such that for all vectors $\vec{x}$, $\vec{y}$ such that $\vec{x} \sim_q^d \vec{y}$ we don't have $\vec{x} \in S$ and $\vec{y} \in \bar{S}$.

Proof. If L is in FO$_{mod(P)}$, the property holds because, by Proposition 3, a formula of FO$_{mod(P)}$ only counts local neighborhoods up to some threshold or modulo q.

Now suppose that L satisfies the property above. Using the consequence of Corollary 1 mentioned before, we see that membership in L only depends on the count of k-types up to some constant and the modulo q class of each such count. This is easy to express in FO$_{mod}$. □

To conclude the proof of the theorem, notice that the property stated in Lemma 16 can be expressed in Presburger Arithmetic, once q, S and $\bar{S}$ are given. Hence, by decidability of Presburger Arithmetic [Pres], one can decide whether or not this property holds. □

From Theorem 9, the result for FO$_{mod}$ easily follows:

Corollary 2. *There is a computable function which takes as input a regular language L of r-ranked trees (presented effectively, for example by a tree automaton) and determines whether or not L is definable in* FO$_{mod}$.

Proof. Given the prior theorem, it suffices to decide whether a language is q-periodic for some fixed q. But this is just checking q-periodicity in a finite monoid, which is known to be decidable [Str94]. □

The algorithm we have just described relies on a decision procedure for Presburger arithmetic, and this is not optimal. We can actually show a more precise characterization of FO$_{mod(P)}$ which yields a PTIME decision procedure. We only state the result below. The proof follows along the same lines as the characterizations in the previous sections, with the additional technical difficulty that only pieces of size 0 modulo q can be added or removed.

Theorem 10. *Let L be a regular tree language.*

Then L is definable in FO$_{mod(P)}$ *iff L is q-periodic and there exists a k such that L is closed under k-guarded swaps.*

We have the following corollary, which follows from the well-known fact that every finite monoid is q-periodic for some q:

Corollary 3. *Let L be a regular tree language.*
Then L is definable in FO_{mod} *iff there exists a k such that L is closed under k-guarded swaps,*

The approach using the finer characterization above works also in the unranked case. The simpler approach we used above to get decidability in the ranked case does not extend in an obvious way to the unranked setting, since it relied on Corollary 1.

7 Conclusions

The main result presented here is the decidability of FO-definability in ranked trees and unordered unranked trees. The question of characterizing FO-definability in ordered unranked trees is open. Our decidability results for unordered unranked trees extend easily to languages given by sentences of Monadic Second Order Logic with counting modulo quantifiers (CMSO). These languages can be presented by a bottom-up tree automaton whose transitions can count the number of children in a given state modulo p. Again, one can get a decision procedure that is polynomial in the size of a deterministic automaton.

We believe that our characterization (and the decidability results that follow) extends to ω-trees. In addition to giving a decision procedure, the characterization here has been useful for demonstrating that certain queries are first-order; for example, it is has been used to prove that order-invariant first-order queries over trees are first-order expressible.

The class LT of languages is defined as for LTT but without the threshold. That is, one can check the occurrence or absence of a pattern in a string but can no longer count the number of occurrences. We are considering how to modify our axioms to characterize LT.

Acknowledgment: We wish to thank Jean-Eric Pin for many fruitful discussions on the word case and Mikołaj Bojańzyk for his help on an earlier draft of this paper.

References

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

[BS05] M. Benedikt and L. Segoufin. Regular languages definable in FO. In *STACS*, pages 327-339, 2005.

[BW04] M. Bojańczyk and I. Walukiewicz. Characterizing EF and EX tree logics. *Theoretical Computer Science*, 358:255–272, 2006.

[B07] M. Bojańczyk. A new algorithm for testing if a regular language is locally threshold testable. To appear in *Information Processing Letters*, 2007

[Buc60] J. Büchi. Weak second-order logic and finite automata. *S. Math. Logik Gründlagen Math.*, 6:66–92, 1960.

[CH91] S. Cho and D T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116, 1991.

[Cou90] B. Courcelle. The monadic second order logic of graphs I: recognizable sets of tinite traphs. *Information and Computation*, 85:12–75, 1990.

[CPP93] J. Cohen, D. Perrin, and J-E. Pin. On the expressive power of temporal logic for finite words. *Journal of Computer and Science Systems*, 46(1993):271–294, 1993.

[Don70] J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[EW03] Z. Ésik and P. Weil. On logically defined recognizable tree languages. In *FSTTCS*, pages 195-207, 2003.

[Heu91] U. Heuter. First-order properties of trees, star-free expressions, and aperiodicity. *Informatique Théorique et Applications*, 25:125–146, 1991.

[Lib04] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[MP71] R. McNaughton and S. Papert. *Counter-free Automata*. MIT Press, 1971.

[N00] J. Nurmonen. Counting modulo quantifiers on finite structures. *Information and Computation*, 160(1-2):62-87 2000.

[Par66] R. Parikh. On context-free languages. *Journal of the ACM* 13(4), pages 570–581, 1966.

[Pin96] J-E. Pin. The expressive power of existential first order sentences of Büchi's sequential calculus. In *Proc. of Intl. Coll. on Automata, Languages and Programming*, pages 300–311, 1996.

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

[Pot95] A. Potthoff. First-order logic on finite trees. In *Theory and Practice of Software Development (TAPSOFT)*, pages 125–139, 1995.

[PT93] A. Potthoff and W. Thomas. Regular tree languages without unary symbols are star-free. In *Fundamentals of Computation Theory (FCT)*, pages 396–405, 1993.

[Pres] M. Presburger. Über die Vollständingkeit eines gewissen Systems der Arithmetic ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt *Comptes Rendus du I congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.

[Sch65] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

[Str94] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.

[Tho97] W. Thomas. *Handbook of Formal Languages*, volume 3, chapter 7. Springer, 1997.

[TW68] J.W. Thatcher and J.B. Wright. Generalized finite automata with an application to a decision problem of second order logic. *Math. Syst. Theory*, 2:57–82, 1968.

[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.

8 Appendix: proofs of the modulo characterization theorems

Here we will give the proofs of the characterizations of first-order logic with counting quantifiers. As before we start with the ranked case, to illustrate the main idea, and then we move to unranked trees.

Fix $P \subseteq \mathbb{N}$ finite and let q be the lcm of all numbers occurring in P. The goal of this section is to show that replacing aperiodicity with q-periodicity in the theorems for FO gives a characterization of $\mathrm{FO}_{mod(P)}$.

8.1 Ranked trees and modulo counting

We start with the ranked tree case, fixing the rank as r for the whole section. Note first that for ranked trees, $\mathrm{FO}_{mod(P)}$ is included in MSO (this is no longer the case for unranked trees). This is because over ranked trees, a linear order can be defined in MSO and therefore counting quantifiers can be simulated using this order. Thus $\mathrm{FO}_{mod(P)}$ defines only regular languages. We recall the statement of Theorem 10:

Let L be a regular tree language.

Then L is definable in $\mathrm{FO}_{mod(P)}$ *iff L is q-periodic and there exists a k such that L is closed under k-guarded swaps.*

The proof that q-periodicity is necessary is done as in [Str94]: one shows by structural induction that all $\mathrm{FO}_{mod(P)}$ formula are q-periodic where free variables are treated as sentences in a product alphabet. The base case of atomic formulas follows from the result for FO, while the inductive cases are already shown in the proof of VII.3.1 of [Str94].

For the converse, we denote by $t \equiv_q^K t'$ the fact that t and t' agree on all sentences of $\mathrm{FO}_{mod(P)}$ with at most K (first-order or modular) quantifiers. As in the case without modulo quantifiers, Theorem 10 will follow immediately from:

Theorem 11. *For any regular language L which is q-periodic and closed under k-guarded swaps, there exists a K such that for any $t, t' \in \mathcal{T}$ we have: $t \equiv_q^K t' \quad \Rightarrow \quad t \in L \;$ iff $\; t' \in L$.*

We will thus work towards the proof of Theorem 11. We follow the lines of the proof of Theorem 2. Fix L and q, k such that L is q-periodic and closed under k-guarded swaps. Fix a deterministic automaton A for L.

The notion of k-spill and depth-k similar is as in the ranked case for FO. Given two trees s, t we denote by $s =_d^{q,k} t$ the fact that for all $\tau \in \mathcal{T}_k$, $|s|_\tau = |t|_\tau$ or, $|s|_\tau, |t|_\tau > d$ and $|s|_\tau = |t|_\tau$ modulo q. If in addition for all $\tau \in \mathcal{T}_k$, $|s|_\tau \leq |t|_\tau$ then we write $s \leq_d^{q,k} t$. If for all $\tau \in \mathcal{T}_k$, $|s|_\tau = |t|_\tau$ then we write $s =_\infty^{q,k} t$.

The first two lemmas are easy adaptations of Lemma 1 and Lemma 2 to the modulo counting case.

Lemma 17. For each number d, there exists a constant K_d such that $s \equiv_q^{K_d} t$ implies that $s =_d^{q,k+1} t$ and s, t are depth-$(k+1)$ similar.

The lemma follows because the count modulo q of occurrences of a given k-spill within a tree can be expressed via an $\mathrm{FO}_{mod(P)}$ sentence.

Lemma 18. For each number d' there exists a number d such that if $s =_d^{q,k+1} t$ then there exists t' such that $s \leq_{d'}^{q,k+1} t'$, and t, t' are depth-$(k+1)$ similar, and $t' \in L$ iff $t \in L$.

Proof. This is proved by making a slight modification of the proof of Lemma 2. Again we proceed by induction on the number of types τ that are not sufficiently well-represented in t'. In the inductive step, we fix a $(k+1)$-type τ. The proof of Lemma 2 shows that we can find a pair of nodes x_1, x_2 with x_2 a strict descendant of x_1 such that $C_t[x_1, x_2)$ (that is, the set of nodes that are below or equal to x_1 and not below x_2) contains at least one node of type τ, both x_1 and x_2 have the same type, all nodes in $C_t[x_1, x_2)$ have a type that is safe (i.e. occurs at least d' times), and the automaton A for L reaches the same state of x_2 as on x_1. If we now replace $C_t[x_1, x_2)$ by $\alpha \cdot p$ copies of itself, for large enough α, then we will have preserved the counting modulo q of occurences of each type, have not disturbed any unsafe type, and will have made the number of occurrences of τ in t' greater than the number in t. $\square$

Lemma 3 and Lemma 4 are proven using only swapping moves, hence without modifying the occurrences of $(k+1)$-types. Therefore we can make use of them in the modulo counting case. The same holds for Corollary 1. It therefore remains to revisit the last part of the proof of Theorem 2 and adapt it to the modulo counting case.

We are now ready to begin the proof of Theorem 11. Let Q be the set of states of A, $\alpha = |Q|$ be the number of states of A. Let $\beta_k = |\mathcal{T}_{k+1}|$. Let l be the q-periodicity number of L. Let $d' = r^{(\beta_k * r^\alpha + 1) * l} + 1$. Let d be the number required in Lemma 18 for d'. Let K be the number K_d required in Lemma 17 for d. We show that $s \equiv_q^K t$ implies $s \in L$ iff $t \in L$.

Assume $s \equiv^K t$, we show that $s \in L$ iff $t \in L$. From Lemma 17 we know that $s =_d^{q,k+1} t$. Therefore by Lemma 18 there is a tree t' such that $t' \in L$ iff $t \in L$ and $s \leq_{d'}^{q,k+1} t'$. We can now apply Lemma 3 and obtain t'' such that $t'' \in L$ iff $t' \in L$, s is $(k+1)$-pseudo-included in t'' via a mapping h, and $t'' =_\infty^{q,k+1} t' =_{d'}^{q,k+1} s$. Therefore it suffices to prove that $s \in L$ iff $t'' \in L$.

Once more following the proof of Theorem 2, we have by construction that t'' is $h(s)$ plus loops (of size larger than 1) inserted between nodes of $h(s)$. As before, all $(k+1)$-types which occur outside of $h(s)$ have strictly more occurrences in t'' than in s and therefore appear at least d' times in s (and in t''). Notice also that for each $(k+1)$-type τ the total number of occurrences of τ outside of $h(s)$ is zero modulo q. Let $V_1 \cdots V_n$ be the sequence (in arbitrary order) of loops in $t'' \backslash h(s)$ and let V be the forest $\bigcup_{1 \leq i \leq n} V_i$. From the remark above we have $\forall \tau \in \mathcal{T}_{k+1}, |V|_\tau = 0$ modulo q.

Before continuing we need some additional definitions. For number j, a *j-context* is a tree with j designated leaves, which we call (generalizing the notation for contexts) ports. Given a j-context C, an ordering of its ports as $p_1, \ldots, p_j$, and trees $t_1 \ldots t_j$ we let $C[t_1, \ldots, t_j]$ denote the tree obtained by plugging in each t_i into p_i. An "abstract j-context" is a j-context C in which the root is different from each port, supplemented with an assignment λ of $(k+1)$-types (where k is the number fixed above) to each non-port node and a k-type to each port node such that the assignments are *consistent*: there are $t_1 \ldots t_j$ such that for every node x in C, the type assigned to x in $C[t_1, \ldots, t_j]$ matches $\lambda(x)$. Note that this notion of consistency extends that given for abstract contexts in the FO case.

Lemma 19. There exists n' and abstract loops $U_1 \cdots U_{n'}$, with each U_i having cardinality greater than 1, such that if $U = \bigcup_{1 \leq i \leq n'} U_i$ we have $\forall \tau \in \mathcal{T}_{k+1}, q \cdot |U|_\tau = |V|_\tau$.

Proof. For each $(k+1)$-type τ and each k-type ν let $\alpha_\tau(\nu)$ be 1 if ν is induced by τ, 0 otherwise. For any multiset B of $(k+1)$-types and any k-type ν, let $\alpha_B(\nu)$ be $\Sigma_{\tau \in \mathcal{T}_{k+1}} |B(\tau)| \alpha_\tau(\nu)$, where $|B(\tau)|$ is the number of occurrences of τ in B. For τ, ν as above let $\beta_\tau(\nu)$ be the number of children of k-type ν that a node of $(k+1)$-type τ must have, and extend this to a multiset of $(k+1)$-types B by $\beta_B(\nu) = \Sigma_{\tau \in \mathcal{T}_{k+1}} |B(\tau)| \beta_\tau(\nu)$. So α_B measures how often a given k-type occurs in a multiset of $(k+1)$-types, and β_B measures how often a k-type occurs as a child in a given set of $(k+1)$-types. Finally, let $\gamma_B(\nu)$ be $\alpha_B(\nu) - \beta_B(\nu)$.

Let B be the multiset of all $(k+1)$-types occurring in V. Because V contains only abstract loops, we have that for each $\nu \in \mathcal{T}_k$ $\gamma_B(\nu) = 0$: the occurrence of a type in the interior of one of the loops is counted once in both α and β, thus contributing 0 to γ, while the occurrence of a type in the root (and hence in the port) is counted in α, but is balanced by the occurrence of that type as a port in β. Let B' be B where the multiplicity of each type has been divided by q. B' is well-defined because all multiplicities in B are multiples of q. Since for each $\nu \in \mathcal{T}_k$, the multiplicity of each $(k+1)$-type that induces it and the multiplicity of each $(k+1)$-type that had it as a child are both divided by q in going from B to B', $\gamma_{B'}(\nu) = 0$. We will construct abstract loops $U_1 \cdots U_{n'}$ such that the multiset formed with the $(k+1)$-types of nodes of $\bigcup_{1 \le i \le n'} U_i$ is B'.

Assume we have already constructed n_1 abstract loops $U_1 \cdots U_{n_1}$, and a "partially constructed loop" – either an abstract j-context X which we hope to extend into an abstract loop, or the "empty abstract context" (whose underlying context has no nodes). Let B_1 be the multiset of $(k+1)$-types assigned to non-port nodes of $X \cup \bigcup_{1 \le i \le n_1} U_i$. We will assume inductively that B_1 is a sub-multiset of the types in B'. Let $B_2 = B' - B_1$, where difference of multisets is defined in the usual way.

If X is empty and $B_1 = B'$ (i.e. B_2 is empty), then we are done, since $U_1 \dots U_{n_1}$ are the required abstract loops. Otherwise we will extend the construction while decreasing the sum of the multiplicities of types in B_2.

Suppose X is empty and $B_1 \neq B'$. Then B_2 must contain at least one $(k+1)$-type with positive multiplicity. Let τ be such a $(k+1)$-type, and let $\nu_1, \dots, \nu_j$ be the sequence of induced k-types of the children of a node of $(k+1)$-type τ. We set X to be an abstract j-context containing a root node assigned to $(k+1)$-type τ with j children, all of which are ports, with the i^{th} child assigned k-type ν_i. The definition of the sequence $\nu_1, \cdots, \nu_j$ implies that this is a consistent assignment.

If X is not empty and is an abstract loop, then we set $U_{n_1+1} = X$ and continue as above. If X is not empty and has no ports, then $\gamma_{B_1}(\nu) > 0$ for ν the k-type of the root of X. Since $\gamma(B')(\nu) = 0$, we have $\gamma(B_2)(\nu) < 0$ and hence there is some $(k+1)$-type τ in B_2 that requires ν as the i^{th} child. We add this type as a new root of X, appending the old X as the i^{th} subtree while making any other required children into ports. Again, the multiplicity of τ in B_2 is decreased.

Suppose X is not empty and is not an abstract loop. Let τ be the $(k+1)$-type assigned to the root of X and τ' be the k-type induced by τ. Since X is not an abstract loop, either X has a port whose assigned k-type is $\nu \neq \tau'$, or X has more than one port of type τ'.

In the first case, fix such a port p and type ν. Then $\gamma(B_1)(\nu) > 0$, and since $\gamma(B')(\nu) = 0$ this implies $\gamma(B_2)(\nu) < 0$. So there is some $(k+1)$-type ρ with positive multiplicity in B_2 consistent with ν. Let $\eta_1 \dots \eta_s$ be the k-types of children required by ρ. Replace port p with a node x of $(k+1)$-type ρ, where x will have children $p_1 \dots p_s$ that are ports of types $\eta_1 \dots \eta_s$, respectively. The size of the multiplicities of types in B_2 has decreased by 1, and we continue the induction.

In the second case, $\gamma(B_1)(\tau') > 0$, and this implies $\gamma(B_2)(\tau') < 0$. Hence there is a $(k+1)$-type ρ in B_2 consistent with τ'. Replace one of the ports of type τ' with a node x of type ρ, which is again given the port children of the types required by ρ, and continue inductively.

Let $U = \bigcup_{1 \le i \le n'} U_i$. By construction we have $\forall \tau \in \mathcal{T}_{k+1}, q.|U|_\tau = q.|B'|_\tau = |B|_\tau = |V|_\tau$ as required. □

Fix $U = \{U_1 \cdots U_{n'}\}$ as in Lemma 19. By induction we construct $s_0 \cdots s_{n'}$ such that: (i) s_0 is s, (ii) $\forall \tau \in \mathcal{T}_{k+1}$ $|s_i|_\tau = |s|_\tau + q.|U_1|_\tau + \cdots q.|U_i|_\tau$, (iii) $s_i \in L$ iff $s_{i-1} \in L$.

The base case is immediate. The induction is done as in the proof of Theorem 2, first U_i is *reduced* to W_i, whose size is strictly bounded by d'/l, and then W_i^l is inserted to s_{i-1} using Lemma 4. We can now use q-periodicity of L to insert q extra copies of W_i (and therefore U_i) as required. All this is done without affecting membership in L.

Let $s' = s_n$. By construction we have $\forall \tau \in \mathcal{T}_{k+1}, |s'|_\tau = |s|_\tau + q.|U|_\tau = |s|_\tau + |V|_\tau = |t''|_\tau$ and $s' \in L$ iff $s \in L$. Theorem 11 now follows from Corollary 1. $\square$

Theorem 10 immediately implies the following complexity bound:

Corollary 4. *There is a* PTIME *algorithm that, given a deterministic bottom-up ranked tree automaton, checks whether the corresponding tree language is definable in* FO_{mod}.

Proof. By our prior results, and the fact that every regular language is q-periodic for some q, it suffices to check in PTIME that a language is closed under k-guarded swapping for some k. But this was already shown in Theorem 7. $\square$

We can also show the analogous result for $\mathrm{FO}_{mod(P)}$:

Theorem 12. *There is a* PTIME *algorithm that, given a minimal deterministic bottom-up ranked tree automaton, checks whether the corresponding tree language is definable in* $\mathrm{FO}_{mod(P)}$.

Proof. By Theorem 10, we need only show that for languages that satisfy k-guarded swapping, q-periodicity can be checked in PTIME. We use an argument modeled tightly on the string case, from [Pin96]. The following claim was proven for $q = 1$ in [Pin96]:

Claim 6. *For a monoid satisfying $e \cdot u \cdot f \cdot s' \cdot e \cdot v \cdot f = e \cdot v \cdot f \cdot s' \cdot e \cdot u \cdot f$ we have $u^l = u^{l+q}$ holds for some l iff*

$(e \cdot x \cdot e \cdot y \cdot e)^\kappa = (e \cdot x \cdot e \cdot y \cdot e)^\kappa \cdot (e \cdot x \cdot e)^q \qquad (\dagger\kappa, q)$

holds for some κ (where, again, e, f range over idempotents and x, y, u, v, s' range over monoid elements).

Proof. The proof of the claim is a simple generalization of the argument for $q = 1$ in [Pin96]. In one direction, we assume $(\dagger\kappa, q)$, and prove q-periodicity by choosing ω such that for all u, u^ω is idempotent, and substituting $x = u, y = e = u^\omega$. This yields the identity $u^{(4\omega+1)\kappa} = u^{(4\omega+1)\kappa}u^{(2\omega+1)q}$, which implies $u^{\omega+\kappa} = u^{\omega+\kappa+q}$ using idempotence of u^ω. Since u was arbitrary, this shows that q-periodicity holds with $l = \omega + \kappa$. In the other direction, we assume q-periodicity and prove $(\dagger l, q)$. We use the observation, proved in [Pin96], that our additional hypothesis on the monoid implies that for any idempotent e and any monoid elements x, y we have

$$exeye = eyexe \quad (**)$$

From (**) and idempotence of e we can derive $(exeye)^l(exe)^q = (exe)^{l+q}(eye)^l$, by repeatedly applying $(**)$ to rewrite occurrences of $eyexe$ to $exeye$, and collapsing e^2 into e.

Now using q-periodicity we have $(exe)^{l+q}(eye)^l = (exe)^l(eye)^l$, and using (**) and idempotence of e we have $(exe)^l(eye)^l = (exeye)^l$. Thus $(exeye)^l(exe)^q = (exeye)^l$ as required for q-periodicity. $\square$

One can decide whether $(\dagger\kappa, q)$ holds using a pattern P^q obtained from P by adding $q-1$ additional nodes, which with q_2 form a chain of length q, with edges from one element of the chain to the next labeled with x. Each element of the chain has a self-loop labeled with u associated with it, and the last element in the chain is constrained to be distinct from q_1. The notion of a graph formed from an automaton which is to be matched against pattern P^q is the same as the notion for P in the FO case for ranked trees. $\square$

8.2 Unranked trees and modulo counting

In the unranked case, the main difference is that $\mathrm{FO}_{mod(P)}$ is no longer included in MSO but in CMSO. From Courcelle [Cou90] we know that the following family of automata has exactly the same expressive power as CMSO. The automaton is defined as for unranked trees, but each transition, in addition to counting the number of states, up to some threshold m, reached for its children nodes, also counts their occurrences modulo some constant q. We call them *modulo q counting automata*. A language that is defined by a modulo q counting automaton for some q, or equivalently definable in CMSO, is called *extended-regular*.

As in the unranked FO case, the difficulty is that the number of isomorphism types of trees of depth k is no longer finite, so we need to reason via approximation. We extend the notion of $\sim_n^k$ to $\sim_n^{k,q}$ in the obvious way by requiring that we count the $(n, k-1)$-types of the children of a node modulo q when above threshold n. We refer to those as (q, n, k)-types. The notion of similarity and guards are then extended as expected.

Again, the heart of the proof is the following intermediate result:

Theorem 13. *Let L be a q-periodic extended-regular tree language.*
Then L is definable in FO iff there exists n, k such that L is closed under (q, n, k)-guarded swaps.

Proof. (sketch) The proof is a combination of the ideas in the proof of Theorem 10 and Theorem 3. We only give an overview here. That the conditions are necessary is proved as in Theorem 10. We now fix q, n, k and an extended-regular tree language L which is q-periodic (with l for the number from the periodicity condition) and closed under (q, n, k)-swaps. We fix an automaton A recognizing L and m very big relative to $|A|$, in particular above the threshold for which A can count number of types exactly. Again we can assume n is above m.

The next result follows from the fact that (q, n, k)-types can be expressed in FO_{mod}:

Lemma 20. For all numbers d and all numbers n', there exists K_d such that $s \equiv_q^{K_d} t$ implies $s =_d^{q,n',k+1} t$ and s, t $(n', q, k+1)$-similar.

With the obvious intended meaning for the notation $s =_d^{q,n,k+1} t$, and using the same tools as in Lemma 6 we have:

Lemma 21. For each number d' and each $n' > n$ there exists a number d such that if $s =_d^{q,n',k+1} t$ then there exists t' such that $s \leq_{d'}^{q,n',k+1} t'$, and t, t' $(n', q, k+1)$-similar, and $t' \in L$ iff $t \in L$.

Proof. The proof follows the structure of Lemma 6. Safe (q, n, k)-types are those that occur at least d' times in t.

We again have two cases to consider. In the first case, every node in t has at small (below d_1, as defined in Lemma 6) number of children which have a descendant of type τ. In this case, we can construct a long τ-skeleton in t. Just as in that proof, by having sections of the skeleton large enough, we can guarantee a portion can be pumped without changing the run of the automaton, adding unsafe types, or changing the existing type structure. By pumping a multiple of q times, we will preserve the modulo q class of every type, so the resulting tree is $(n', q, k+1)$-similar to t.

In the second case, there is some node x in t that has a large number of children which have a descendant of type τ. Then can find a large (above m) number of children with a descendant of type τ which are safe and where the automaton A reaches the same state. It is now possible to duplicate any of the subtrees of these children without affecting membership in L, changing the type of x, or adding unsafe types. So in particular we can add a large multiple of q copies of some child, resulting in a tree that is $(n', q, k+1)$-similar to t. □

Pseudo-inclusion is defined as in the unranked case for first-order logic. We have the obvious extension of the pseudo-inclusion lemma:

Lemma 22. For all d', n' there exists n such that if $s \preceq_{d'}^{q,n,k+1} t$ and s, t $(n, q, k+1)$-similar then there exists t' such that s is $(q, n', k+1)$-pseudo-included in t', $t' =_{d'}^{q,n',k+1} t$, and $t' \in L$ iff $t \in L$.

Proof. The proof follows the argument in Lemma 7. In that proof we considered several cases. In most of these cases, no pumping is necessary and thus only swapping moves are used to get the desired result. As swapping does not affect the number of $(q, n, k+1)$-types at all (and hence does not effect their counts modulo any number), these cases also work here. When a node x of $(q, n, k+1)$-type τ needs to be expanded, we did so by adding a large number of copies of the subtree of a given child of x. Using the same argument in that proof, we see that a child exists that has many siblings for which the automaton reaches the same state and such that the subtree of the child has only safe types (that is all types have above d' occurences). We choose such a child and then add a large multiple of q copies of the child. □

For an abstract context U and tree t, $U <_{q,n,k+1} t$ means that $(q, n, k+1)$-types of U occur strictly more frequently in t. The notion of inclusion and thinness is extended in the obvious way to $(q, n, k+1)$-types.

Lemma 23. Let U be an abstract context. If $U <_{q,n,k+1} t$ and each node $x \in U$ is $(q, n, k+1)$-thin, then there exists t' such that U is $(q, n, k+1)$-included in t', $t' =_{\infty}^{q,n,k+1} t$ and, $t' \in L$ iff $t \in L$.

Proof. With the extension of the definitions in place, this follows from the same argument as in the unranked case for FO. Note that only swapping moves were applied in the proof of Lemma 8, so the exact number of each type (in particular, the number modulo-q of types) is preserved. □

Similarly, we have the version for forests, which again follows by the same set of swapping moves as in the FO case for unranked trees:

Lemma 24. Let U be a forest. If $U <_{q,n,k+1} t$ and each node $x \in U$ is $(n, k+1)$-thin, then there exists t' such that U is $(q, n, k+1)$-included in t', $t' =_{\infty}^{n,k+1} t$ and $t' \in L$ iff $t \in L$.

Proof of Theorem 14 for unranked trees (sketch). As usual we set all the numbers to be big enough in order to be able to apply all the previous lemmas. Starting with two trees s and t such that $s \equiv_q^K t$, we end up with two trees s and t'' such that s is $(q, n, k+1)$-pseudo-included in t'', $t'' \in L$ iff $t \in l$, and $t'' =_{d'}^{q,n,k+1} s$. We wish to show that $s \in L$ iff $t'' \in L$.

By construction t'' is $h(s)$ plus *loops* inserted between nodes of $h(s)$ and extra *branches* branching off the h-pseudo-tree. As before, each $(q, n, k+1)$-type which occurs outside of $h(s)$ must occur strictly more often within t'' than in s and therefore must appear at least d' times in s (and in t''). Furthermore for each $(q, n, k+1)$-type τ the total number of occurrences of τ outside of $h(s)$, including both the branches and the loops, is zero modulo q. Let Vl be the forest of loops occurring in t'' outside of $h(s)$ and Vb the forest of branches occurring in t'' outside of $h(s)$.

As in the FO unranked case, we reduce the size of Vl and Vb without affecting either membership in L or the cardinality modulo q of each type. We can further assume that all types are $(q, n, k+1)$-thin. Let $Vl_1 \cdots Vl_\alpha$ be the set of loops in $t'' \backslash h(s)$ and $Vb_1 \cdots Vb_\beta$ be the set of extra branches in $t'' \backslash h(s)$. From the remark above we have $\forall \tau \in \mathcal{T}_{q,n,k+1}$, $|V|_\tau = 0$ modulo q.

The main difference from the ranked case lies in the following lemma, replacing Lemma 19. Its proof will be obtained by induction on V using the same ideas as in Lemma 19.

Lemma 25. There exists α' abstract loops $Ul_1 \ldots Ul_{\alpha'}$ and β' branches $Ub_1 \ldots Ub_{\beta'}$ such that if Ul is the multiset of $(q, n, k+1)$-types occurring in all of the Ul_i and Ub is the same for the Ub_i, then we have $\forall \tau \in \mathcal{T}_{q,n,k+1}, q \cdot (|Ul|_\tau + |Ub|_\tau) = |Vb|_\tau$.

Proof. The functions α and α_B are defined as in the ranked case for FO$_{mod(P)}$. The function $\beta_\tau(\nu)$ is defined as the minimum number of nodes of type ν that τ requires – note that this is the same as the number of nodes of type ν that will occur in a thin realization of τ. γ is $\alpha - \beta$ as in the ranked case.

Again, let B be the multiset of all $(k+1)$-types occurring in Vl or Vb, and let B' be the multiset resulting from dividing all multiplicities in B by q. One significant difference is that now $\gamma_B(\nu) > 0$ for many $\nu \in \mathcal{T}_k$, since each branch will have the type of its root contributing to α more often then to β within that branch. Our goal is to construct abstract loops and branches such that the multiset formed with their $(k+1)$-types is exactly B'.

Let $C(\nu) = \gamma_{B'}(\nu)$ for each $\nu \in \mathcal{T}_k$. Assume we have already constructed n_1 abstract loops $Ul_1 \ldots Ul_{m_1}$ and n_1 branches $Ub_1 \ldots Ub_{n_1}$, along with an abstract j-context X (possibly empty) which we hope to extend into either an abstract loop or branch. Let $Comp$ be be the multiset of $(k+1)$-types assigned to non-port nodes of $\bigcup_{1 \leq i \leq m_1} Ul_i \cup \bigcup_{1 \leq i \leq n_1} Ub_i$, and let $Part$ be the multiset of $(k+1)$-types assigned to non-port nodes of X. Let B_1 be the union (as a multiset) of $Comp$ and $Part$, and $B_2 = B' - B_1$. We will assume inductively that B_1 is a sub-multiset of the types in B', and also that $\gamma_{Comp}(\nu)$ is at most $C(\nu)$ for each $\nu \in \mathcal{T}_k$. This second condition is equivalent to demanding that for every k-type ν, ν occurs as the root of some completed branch in $Ub_1 \ldots Ub_{n_1}$ no more often than $C(\nu)$. Note that this condition will ensure that $\gamma_{B_2 \cup Part}(\nu) \geq 0$ for each $\nu \in \mathcal{T}_k$.

If X is empty and $B_1 = B'$ then we are done. If X is an abstract loop and $B_1 \neq B'$, then we set $Ul_{m_1+1} = X$ and continue. If X is empty and $B_1 \neq B'$, then we proceed as in the ranked case: set X by choosing an arbitrary type occurring with positive multiplicity in B_2, and give it the ports that the type requires as children. Since we do not change $Comp$, we do not violate the second inductive invariant above.

Suppose X is not empty, is not an abstract loop, and has at least one port. If X has a single port whose k-type ν does not match the induced k-type of the root, then $\gamma_{Part}(\nu) < 0$. Since $\gamma_{B_2 \cup Part}(\nu) \geq 0$ by the inductive invariant, we know $\gamma_{B_2}(\nu) > 0$, which allows us to proceed as in the ranked case, choosing a type from B_2 that induces ν, and expanding X accordingly. The case where X has more than one port is handled similarly.

The last case is when X is not empty and has no ports. Let ν be the k-type induced by the $(k+1)$-type of the root of X. If $\gamma_{Comp}(\nu) < C(\nu)$, then we can add X as $Ub_{n_1} + 1$ and continue. If $\gamma_{Comp}(\nu) = C(\nu)$, then X has no ports to expand on, but cannot be added as a new completed branch without destroying the inductive invariant. In this case $\gamma_{Part \cup B_2}(\nu) = 0$, and $\gamma_{Part}(\nu) = 1$, so there is some $(k+1)$-type τ in B_2 that requires at least one child of type ν. Extend X by adding a new root of type τ, attaching the former X as one child and making the remaining required children ports. The multiplicity of τ in B_2 is reduced, and we can continue the induction. $\square$

We can now transform s by induction in order to insert q copies of Ul_i for each i, and likewise transform t to remove q copies of each Ub_j. This is done as in the proof of Theorem 10 for loops, and as in the unranked case of Theorem 2 for branches, working with groups of size q. The details are left to the reader. $\square$

From Theorem 13 we can now show:

Theorem 14. *Let L be an extended-regular tree language.*

Then L is definable in FO iff it is q-periodic and there exists k such that L is closed under k-guarded swaps.

Proof. Clearly, it is enough to show that closure under k-guarded swaps implies closure under (q, n, k)-guarded swaps for sufficiently large n. This is proved as in the unranked case without modulo quantifiers (Theorem 5). $\square$

A polynomial time algorithm follows for immediately for FO$_{mod}$ using the same techniques as in Theorem 8.