Leaf Powers and Their Properties: Using the Trees [⋆]

Michael Fellows[1], Daniel Meister[2], Frances Rosamond[1], R. Sritharan[3], and Jan Arne Telle[2]

[1] University of Newcastle, Newcastle, Australia.
michael.fellows@newcastle.edu.au, frances.rosamond@newcastle.edu.au
[2] Department of Informatics, University of Bergen, Norway.
daniel.meister@ii.uib.no, jan.arne.telle@ii.uib.no
[3] Department of Computer Science, The University of Dayton, Dayton, USA.
srithara@notes.udayton.edu

Abstract. A graph G on n vertices is a *k-leaf power* ($G \in \mathcal{G}_k$) if it is isomorphic to a graph that can be "generated" from a tree T that has n leaves, by taking the leaves to represent vertices of G, and making two vertices adjacent if and only if they are at distance at most k in T. We address two questions in this paper:
(1) As k increases, do we always have $\mathcal{G}_k \subseteq \mathcal{G}_{k+1}$?
Answering an open question of Andreas Brandstädt and Van Bang Le [2, 3, 1], we show that the answer, perhaps surprisingly, is "no."
(2) How should one design algorithms to determine, for k-leaf powers, if they have some property?
One way this can be done is to use the fact that k-leaf powers have bounded cliquewidth. This fact, plus the FPT cliquewidth approximation algorithm of Oum and Seymour [14], combined with the results of Courcelle, Makowsky and Rotics [7], allows us to conclude that properties expressible in a general logic formalism, can be decided in FPT time for k-leaf powers, parameterizing by k. This is wildly inefficient. We explore a different approach, under the assumption that a generating tree is given with the graph. We show that one can use the tree *directly* to decide the property, by means of a finite-state tree automaton. (A more general theorem has been independently obtained by Blumensath and Courcelle [5].)
We place our results in a general context of "tree-definable" graph classes, of which k-leaf powers are one particular example.

1 Introduction

A graph G is a tree power if there is an integer k and a tree T on the same vertex set as G such that two vertices are adjacent in G if and only if they are at distance at most k in T [11]. A leaf-power graph is defined similarly except that the vertex set of G now corresponds to the leaves of T. Leaf powers, and k-leaf

[⋆] This work is supported by the Research Council of Norway.

powers where the power k is specified, were introduced by Nishimura, Ragde, Thilikos [13] in 2002 and have attracted a considerable amount of attention. These graphs have applications in the reconstruction of the evolutionary history of species and admit efficient algorithms for generaly difficult problems, that mainly exploit the tree structure of the underlying tree. Note that k-leaf powers can have arbitrarily large cliques, which means that the class of k-leaf powers is of unbounded treewidth. It was recently shown that a k-leaf power has cliquewidth at most $3k/2$ [10]. It was known that $(k-1)$-leaf powers are k-leaf powers for $k \in \{1, 2, 3, 4\}$ [2], but it was an open question whether this containment relationship continued. In this paper we show that this is not the case.

Leaf-power graphs are only one example of graphs with nice algorithmic properties that rely on an underlying tree structure. In this paper, we generalise the main concepts. The definition of leaf-power graphs consists of three elements: a tree, a selection of tree vertices and an adjacency condition. For a fixed k, the selected vertices are the leaves of the tree and the adjacency condition is a distance condition. "Being a leaf, i.e., a vertex of certain degree" and "being a pair of vertices at bounded distance" are predicates that can be expressed already in limited logic. Based on these observations, we introduce the notion of tree-definable graph classes. A tree will then define a graph where the vertex set is selected by a unary graph predicate and adjacency is defined by a binary graph predicate. We show that every problem that is definable in monadic second-order logic allowing quantification over sets of vertices can be solved in linear time on a tree-definable graph class, when the representing tree is given. When allowing quantification over sets of edges the results are not so positive, and we apply a Myhill-Nerode argument to show that the set of trees generating Hamiltonian 3-leaf powers does not have finite index. The logical expressions used to generate 4-leaf powers and 5-leaf powers are very similar and yet we show that there exists a 4-leaf power which is not a 5-leaf power, thereby answering an open question of Andreas Brandstädt and Van Bang Le [2, 3, 1].

The paper is organised as follows. In the next section, we define the notion of *tree-definable graph class*. In Section 3, we show that well-known graph classes such as cographs and fixed leaf-power classes are tree-definable. We also show that not every 4-leaf power is a 5-leaf power. Algorithmic consequences for tree-definable graph classes are discussed in Section 4. In the final Section 5, we present open problems.

2 Tree-definable graph classes

We give a very general method for defining graphs. It has three ingredients: a tree, a logical expression extracting the vertex set of the graph from the tree, and another logical expression extracting the adjacencies in the graph from the tree. In the next section, we will show for some well-known graph classes that they can be defined in this uniform way. (We have recently learned that some of our results were obtained earlier, in a very general setting, by Blumensath and Courcelle [5], where they study *rational transductions* of relational systems.)

To begin, we specify the logic that we use in this paper. An M2O *graph expression* has four types of variables, for vertices, edges, sets of vertices, sets of edges, and is formed from the following elements:

– equality for each of the four variable types
– the two predicates $x \in X$ and $e \in Y$ for x a vertex variable, X a vertex set variable, e an edge variable and Y an edge set variable and the two predicates $\mathrm{adj}(u,v)$ and $\mathrm{inc}(u,e)$ for u and v vertex variables and e an edge variable
– the connectives $\wedge, \vee, \neg$ and the quantifiers $\forall, \exists$.

Every variable type allows quantification. Clearly, $x \in X$ is true if x is interpreted by a vertex contained in X; similarly for $e \in Y$. And $\mathrm{adj}(u,v)$ is true if u and v are interpreted by adjacent, in particular different, vertices, and $\mathrm{inc}(u,e)$ is true if u is an endpoint of e, i.e., e is incident to u. A graph expression without free variables is called *graph sentence*. For an M2O graph sentence Φ we say that a graph G is a *model* for Φ, denoted as $G \models \Phi$, if Φ is true on G.

Definition 1. *Let $\Phi_1 = \Phi_1(x)$ and $\Phi_2 = \Phi_2(x_1, x_2)$ be* M2O *graph expressions with x, x_1 and x_2 vertex variables. For a tree T, the graph G represented by T with respect to (Φ_1, Φ_2), denoted as $G_{\Phi_1,\Phi_2}(T)$, is defined as follows:*

– $V(G) =_{\mathrm{def}} \{u \in V(T) : T \models \Phi_1(u)\}$
– $E(G) =_{\mathrm{def}} \{uv : u, v \in V(G) \text{ and } u \neq v \text{ and } T \models \Phi_2(u,v)\}$.

*We denote by $\mathcal{G}(\Phi_1, \Phi_2)$ the class of graphs defined by Φ_1, Φ_2, i.e., $\mathcal{G}(\Phi_1, \Phi_2) = \{G_{\Phi_1,\Phi_2}(T) : T \text{ a tree}\}$. We say that $\mathcal{G}(\Phi_1, \Phi_2)$ is a **tree-definable** graph class.*

Note that every tree represents a graph with respect to every pair (Φ_1, Φ_2) of graph expressions, but the represented graph may be the empty graph or may have no edges. The definition of tree-definable graph classes raises immediate questions of three types, that we will address in this paper. Let (Φ_1, Φ_2) and (Φ_1', Φ_2') be pairs of graph expressions.

1) Given a tree T, how difficult is it to compute the graph that is represented by T with respect to (Φ_1, Φ_2), i.e., the graph $G_{\Phi_1,\Phi_2}(T)$?
2) Given a graph G, how difficult is it to compute a tree T such that T represents G with respect to (Φ_1, Φ_2), i.e., such that $G = G_{\Phi_1,\Phi_2}(T)$?
3) How difficult is it to decide whether $\mathcal{G}(\Phi_1, \Phi_2)$ is contained in $\mathcal{G}(\Phi_1', \Phi_2')$, i.e., whether $\mathcal{G}(\Phi_1, \Phi_2) \subseteq \mathcal{G}(\Phi_1', \Phi_2')$?

In view of question 2 we remark that we restrict to named graphs. If graphs have unnamed vertices, question 2 should be rephrased as an isomorphism problem. Questions 1 and 2 are very natural, as they state the verification problem and a restricted version of the graph class recognition problem. Question 3 is partially motivated by a later result about the existence of efficient algorithms for certain problems. We address the three questions, among others, in different parts of this paper.

Another, in some sense more concrete question asks about graph classes that are captured by our model. Are well-known graph classes tree-definable? In the next section, we answer this question in the affirmative and give two examples. This also supports our opinion that the definition of tree-definable graph classes is indeed a natural approach to unify graph classes.

3 Examples of tree-definable graph classes

As our first example we show that all (fixed) leaf-power classes are tree-definable. For a number $k \geq 1$, a graph G is a *k-leaf power* if there is a tree T such that the vertices of G are in 1-to-1 correspondence with the leaves of T and two vertices of G are adjacent if and only if the distance of the corresponding leaves is at most k in T. Hence, for the definition of the two predicates, the vertex selection predicate should be true only for leaves, and adjacency is a simple distance condition. Let

- $L(x) =_{\text{def}} \exists u \forall v(\text{adj}(x,v) \rightarrow u = v)$
- $P_1(x_1,x_2) =_{\text{def}} (x_1 = x_2 \vee \text{adj}(x_1,x_2))$
  $P_{i+1}(x_1,x_2) =_{\text{def}} \exists u(P_i(x_1,u) \wedge P_1(u,x_2))$ for all $i \geq 1$.

For a graph $G = (V,E)$, two (not necessarily different) vertices u, v of G and a number $k \geq 1$, the following holds:

- $G \models L(u)$ if and only if $d_G(u) \leq 1$
- $G \models P_k(u,v)$ if and only if there is a u,v-path of length at most k in G.

Hence, $\mathcal{G}(L,P_k)$ is the class of k-leaf powers. Using the example of k-leaf powers, we show that the inclusion relation of tree-definable graph classes is not easy to determine from the logical expressions directly, thereby settling an open problem. It is known that k-leaf powers are $(k+1)$-leaf powers for $k \in \{1,2,3\}$, i.e., $\mathcal{G}(L,P_1) \subseteq \mathcal{G}(L,P_2) \subseteq \mathcal{G}(L,P_3) \subseteq \mathcal{G}(L,P_4)$ [2]. It was an open question whether this containment relationship continues. The following result shows that this is not the case.

Lemma 1. *There is a 4-leaf power that is not a 5-leaf power.*

Proof. Consider the tree T in Figure 1. Let G be the square of T, i.e., the graph on the vertices of T where two vertices are adjacent if and only if they are at distance at most 2 in T. It is clear that G is a 4-leaf power. We show that G is not a 5-leaf power. For a contradiction, assume that G is a 5-leaf power. Let G be the 5-leaf power of tree S'. Since G has no true twins, no vertex of S' is adjacent to two leaves. Let S be the subgraph of S' induced by all non-leaf vertices. It holds that G is isomorphic to an induced subgraph of the third power of S. For ease of notation, let vertices of S have the name of their corresponding vertex as in Figure 1.

Since vertices 2, 3 and 4 are pairwise adjacent in G, it holds for the distances between the three vertices: $\text{d}_S(2,3), \text{d}_S(2,4), \text{d}_S(3,4) \leq 3$. If $\text{d}_S(2,3) = 3$ then

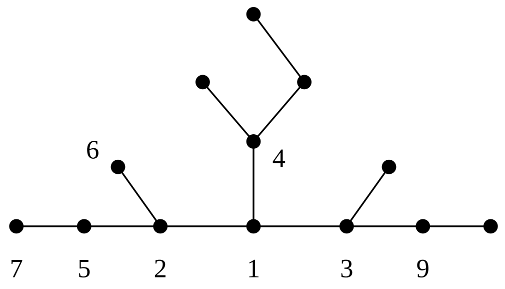

Fig. 1. The square of the depicted tree, i.e., the graph on the vertex set of the tree with vertices adjacent if they are adjacent in the tree or have a common neighbour, is a 4-leaf power but not a 5-leaf power.

$\mathrm{d}_S(2,4) \neq \mathrm{d}_S(3,4)$. Hence, one of the three distances is at most 2. Without loss of generality, $\mathrm{d}_S(2,3) \leq 2$. Consider the path P between 5 and 9 in S: since 5 and 9 are adjacent to 1 in G, it holds $\mathrm{d}_S(5,9) \leq \mathrm{d}_S(5,1) + \mathrm{d}_S(9,1) \leq 6$. We consider different cases. Let $5 \leq \mathrm{d}_S(5,9) \leq 6$. Then, 1 lies on P, at distance 3 to 5 or 9; by symmetry we can assume $\mathrm{d}_S(1,5) = 3$. If 3 lies on the $1,5$-path in S then 5 is adjacent also to 3. Hence, 3 does not lie on the $1,5$-path in S. If the $3,9$-path in S contains two vertices of the $1,5$-path then $\mathrm{d}_S(3,9) \geq 4$. Hence, 2 and 3 are contained in different connected components of $S-1$. The distance condition $\mathrm{d}_S(2,3) \leq 2$ then shows that 2 and 3 are neighbours of 1 in S. In particular, 2 is vertex on the $1,5$-path in S. This also implies that $\mathrm{d}_S(1,9) = 3$, since $4 \leq \mathrm{d}_S(2,9) \leq 1 + \mathrm{d}_S(1,9)$. Then, 3 lies on the $1,9$-path in S. Summarising, we obtain that the $5,9$-path in S is the following: $(5,x,2,1,3,y,9)$ for some vertices x and y. Now, consider vertex 6. It is adjacent to 1, 2, 5 and non-adjacent to 3. Hence, 6 is a neighbour of x in S. Now, consider vertex 7. The neighbours of 7 are 2 and 5, but there is no tree that allows assignment of 7. Hence, $\mathrm{d}_S(5,9) = 4$. Note that every vertex of G that corresponds to a vertex on P besides 5 and 9 is a common neighbour of 5 and 9. Since 1 is the only common neighbour of 5 and 9 in G, 2 and 3 are not contained in P. This, however, is only possible for $\mathrm{d}_S(2,3) \geq 4$. We conclude that G cannot be a 5-leaf power.

Hence, Lemma 1 shows $\mathcal{G}(L, P_4) \not\subseteq \mathcal{G}(L, P_5)$. This example also shows that the inclusion question, given M2O predicates $\Phi_1, \Phi_2, \Phi'_1, \Phi'_2$ to decide whether the inclusion $\mathcal{G}(\Phi_1, \Phi_2) \subseteq \mathcal{G}(\Phi'_1, \Phi'_2)$ holds, cannot be determined easily from just looking at the expressions.

As a second example, we show that cographs are tree-definable. Cographs are the P_4-free graphs, i.e., the graphs that do not contain the induced path on four vertices as induced subgraph. Our result is based on the cotree characterisation of cographs. A rooted tree whose non-leaf vertices are labelled 0 or 1 is called a *cotree*. It holds that a graph G is a cograph if and only if there is a cotree T such that the vertices of G correspond to the leaves of T and two vertices u, v of G are adjacent if and only if the least common ancestor of u and v is labelled 1.

Proposition 1. *The class of cographs is a tree-definable graph class.*

Proof. We prove the proposition in two steps. We first extend the definition of tree-definable graph classes to coloured trees, and then we show that coloured trees are captured by our model.

We need five vertex colours with the following meanings: leaf, root vertex with label 0, root vertex with label 1, non-root vertex with label 0, non-root vertex with label 1. First, we give a predicate that checks whether a given coloured tree represents a cotree correctly. This means that every vertex has a colour, there is exactly one root vertex and the leaves are exactly the vertices with the leaf colour. The colour class predicates are denoted as C_1, C_2, C_3, C_4, C_5.

- $L(x) =_{\text{def}} \exists u \forall v(\text{adj}(x,v) \to u = v)$
- $Q =_{\text{def}} \forall u((L(u) \to C_1(u)) \land (C_1(u) \to L(u)))$
- $R =_{\text{def}} \exists u((C_2(u) \lor C_3(u)) \land \forall v((C_2(v) \lor C_3(v)) \to u = v))$
- $P =_{\text{def}} \forall u(C_1(u) \lor C_2(u) \lor C_3(u) \lor C_4(u) \lor C_5(u)) \land Q \land R$

It holds for a 5-coloured tree T that $T \models P$ if and only if T correctly represents a cotree, in particular, has a unique root vertex and all leaves are coloured with the unique leaf colour. It remains to define adjacency. To find the first common vertex of two leaf-root paths, we need predicates for 'path' and 'first common vertex'. The following expressions describe 'path' in the following way, precisely, predicate X is true if the vertices in Z describe a path between a and b.

- $N_{\geq 2}(x,a,b,Z) =_{\text{def}} \exists v_1 \exists v_2 (v_1, v_2 \in Z \cup \{a,b\} \land v_1 \neq v_2 \land \text{adj}(x,v_1) \land \text{adj}(x,v_2))$
- $N_{\leq 2}(x,a,b,Z) =_{\text{def}} \exists v_1 \exists v_2 \forall u((u \in Z \cup \{a,b\} \land \text{adj}(x,u)) \to (u = v_1 \lor u = v_2))$
- $X(a,b,Z) =_{\text{def}} \forall u(u \in Z \to (N_{\geq 2}(u,a,b,Z) \land N_{\leq 2}(u,a,b,Z))) \land \exists u(u \in Z \land \text{adj}(a,u)) \land \exists u(u \in Z \land \text{adj}(b,u))$.

To determine the first common vertex of two root paths, we apply the same ideas but omit this here.

For the second step, we encode vertex colours into an uncoloured tree. The idea is to represent colours by the number of leaves adjacent to a non-leaf vertex. Predicate L verifies whether a vertex is a leaf or a non-leaf, and similar to predicates $N_{\leq 2}$ and $N_{\geq 2}$ we can formulate predicates that check whether a non-leaf vertex has exactly 1, 2, 3 or 4 or at least five adjacent leaves. The final expressions then select the non-leaf vertices with exactly one adjacent leaf to represent the vertices of the graph and two vertices are adjacent if and only if the tree is syntactically correct and the two paths to the root meet first on a 1 labelled vertex, i.e., a vertex with exactly three adjacent leaves. If the tree is not syntactically correct, the represented graph has no edges. This is also a cograph. It follows with the cograph characterisation via cotrees that cographs are tree-representable.

It is a well-known fact that the cographs are exactly the graphs of clique-width at most 2.

4 Efficiently solvable problems

In this section, we mainly show that the computational bottleneck for many problems on tree-definable graphs is the computation of a representing tree. We show that all problems that can be formulated in a restricted version of M2O have efficient algorithms on input a representing tree. Our result is based on Courcelle's celebrated theorem about solvability of problems on graphs of bounded treewidth [6].

We show that problems that are expressible in a restricted version of M2O logic can be solved efficiently on the representing trees as input. This restricted logic does not allow edge set variables in expressions, and we call such expressions M1O *graph expressions.*

Lemma 2. *Let Φ_1 be a unary* M2O *graph expression and let Φ_2 be a binary* M2O *graph expression. Let Ψ be an* M1O *graph sentence. There is an* M2O *graph sentence Ψ' such that for all trees T: $G_{\Phi_1,\Phi_2}(T) \models \Psi$ if and only if $T \models \Psi'$.*

Proof. We modify Ψ to obtain Ψ'. Consider the following predicate P:

$$P(X) =_{\text{def}} \exists a \exists b \forall c (a \in X \wedge b \in X \wedge \neg(a = b) \wedge (c \in X \rightarrow (c = a \vee c = b)) \wedge \Phi_1(a) \wedge \Phi_1(b) \wedge \Phi_2(a, b))$$

where a, b, c are vertex variables and X is a vertex set variable. Let G be a graph and let U be a set of vertices of G. Then, the following holds:

- $G \models P(U)$ if and only if $U = \{u, v\}$ for some vertices u, v of G, $u \neq v$, and $G \models \Phi_1(u)$ and $G \models \Phi_1(v)$ and $G \models \Phi_2(u, v)$.

We show the claim of the lemma by induction over the definition of our graph expressions. Note that, by assumption, Ψ does not contain $e \in Y$ for e an edge variable and Y an edge set variable as subexpression. For the proof, we have to consider arbitrary graph expressions without edge set variable occurrences. For an M2O graph expression F without edge set variable occurrences we write $F = F(\alpha, \beta, \gamma)$ where α is a list of the vertex variables that occur in F, β is a list of the vertex set variables that occur in F and γ is a list of the edge variables that occur in F. Expression H denotes the equivalent expression after modification. We begin with atomic expressions.

- $F(\langle x_1, x_2 \rangle, \langle\rangle, \langle\rangle) = (x_1 = x_2)$ for x_1, x_2 vertex variables
  $H(\langle x_1, x_2 \rangle, \langle\rangle) =_{\text{def}} (x_1 = x_2)$
- $F(\langle\rangle, \langle X_1, X_2 \rangle, \langle\rangle) = (X_1 = X_2)$ for X_1, X_2 vertex set variables
  $H(\langle\rangle, \langle X_1, X_2 \rangle) =_{\text{def}} (X_1 = X_2)$
- $F(\langle\rangle, \langle\rangle, \langle y_1, y_2 \rangle) = (y_1 = y_2)$ for y_1, y_2 edge variables
  $H(\langle\rangle, \langle Y_1, Y_2 \rangle) =_{\text{def}} (Y_1 = Y_2)$
- $F(\langle x \rangle, \langle X \rangle, \langle\rangle) = (x \in X)$ for x a vertex variable and X a vertex set variable
  $H(\langle x \rangle, \langle X \rangle) =_{\text{def}} (x \in X)$
- $F(\langle x_1, x_2 \rangle, \langle\rangle, \langle\rangle) = (\text{adj}(x_1, x_2))$ for x_1, x_2 vertex variables
  $H(\langle x_1, x_2 \rangle, \langle\rangle) =_{\text{def}} \Phi_2(x_1, x_2)$

– $F(\langle x\rangle, \langle\rangle, \langle y\rangle) = (\mathrm{inc}(x,y))$ for x a vertex variable and y an edge variable
$H(\langle x\rangle, \langle Y\rangle) =_{\mathrm{def}} (x \in Y)$

We continue with the non-atomic expressions. Clearly, it suffices to consider only $\exists$-quantified variables. By H', we denote the expression inductively obtained for F'.

– $F(\alpha, \beta, \gamma) = \exists x F'(\langle x\rangle \circ \alpha, \beta, \gamma)$ for x a vertex variable
$H(\alpha, \beta \circ \gamma') =_{\mathrm{def}} \exists x(\Phi_1(x) \wedge H'(\langle x\rangle \circ \alpha, \beta \circ \gamma'))$
– $F(\alpha, \beta, \gamma) = \exists X F'(\alpha, \langle X\rangle \circ \beta, \gamma)$ for X a vertex set variable
$H(\alpha, \beta \circ \gamma') =_{\mathrm{def}} \exists X(\forall a(a \in X \rightarrow \Phi_1(a)) \wedge H'(\alpha, \langle X\rangle \circ \beta \circ \gamma'))$
– $F(\alpha, \beta, \gamma) = \exists y F'(\alpha, \beta, \langle y\rangle \circ \gamma)$ for y an edge variable
$H(\alpha, \beta \circ \gamma') =_{\mathrm{def}} \exists Y(P(Y) \wedge H'(\alpha, \beta \circ \langle Y\rangle \circ \gamma'))$
– the cases for $\wedge$, $\vee$, $\neg$ are obvious.

This completes the proof.

Theorem 1 ([6]). *Let Ψ be an* M2O *graph sentence. There is a linear-time algorithm that decides on input a tree T whether $T \models \Psi$.*

Corollary 1. *Let $\mathcal{G}$ be a tree-definable graph class, defined by the two* M2O *graph expressions Φ_1 and Φ_2. Let there be an algorithm that on input a graph G from $\mathcal{G}$ computes a tree T such that $G_{\Phi_1,\Phi_2}(T) = G$ in time $\mathcal{O}(f(n,m))$. Then, for every problem Ψ definable as an* M1O *graph sentence, there is an algorithm that decides Ψ on input graphs from $\mathcal{G}$ in time $\mathcal{O}(f(n,m))$.*

Proof. The algorithm is simple: on input a graph G from $\mathcal{G}$, compute a representing tree for G in time $\mathcal{O}(f(n,m))$. Note that T has at most $\mathcal{O}(f(n,m))$ vertices. Lemma 2 shows that there is an M2O graph sentence Ψ' such that $T \models \Psi'$ if and only if $G \models \Psi$. It is important to note that Ψ' does not depend on T. By Theorem 1 there exists an algorithm for deciding $T \models \Psi'$ in time linear in the size of T. Thus, there is an algorithm for deciding whether $G \models \Psi$ in time $\mathcal{O}(f(n,m))$.

With Corollary 1, the main algorithmic challenge for graph problems expressible in M1O logic is to design efficient algorithms for computing a representing tree. An important parameter determining the running time of such an algorithm is the size of the output. In general, a graph can have a large representing tree. As an example, our formulas for cographs in Proposition 1 allow trees that can be obtained from cotrees by subdividing arbitrary edges with vertices of label 0. Therefore, it is an interesting question to determine bounds on the size of representing trees. Lower bounds additionally provide lower bounds on the running times of algorithms.

Besides the questions about running time, the main question is which problems can be solved by our approach. We consider a famous example. A *Hamiltonian cycle* in a graph G is a cycle that visits every vertex of G exactly once. It is known that Hamiltonicity, the problem asking whether a graph contains a Hamiltonian cycle, is not expressible in M1O logic [12]. The essence of our approach is to *use the trees* that generate the graphs in the family (e.g., k-leaf powers). We may be interested in a property (such as Hamiltonicity) where Corollary 1 does not apply, but it still might be the case that the property (of the graphs generated by the trees) might be recognizable by a finite-state tree automaton acting on the generating trees. However, for *Hamiltonicity*, we next show that this cannot be done for k-leaf powers.

Our argument uses the "Myhill-Nerode" point of view, such as exposited in [8].

Our non-expressibility result is obtained in two steps. We first show that the problem is not finite-state for a special congruence relation. For rooted trees T_1 and T_2 with roots R_1 and R_2, respectively, the result of the operation $\oplus$ on T_1 and T_2, denoted as $T_1 \oplus T_2$, is the composition of T_1 and T_2 by gluing the two trees together on R_1 and R_2. The compound graph, that is a tree, has root the glue vertex (R_1, R_2). The relation $\sim_{\mathcal{F}}$ over a set $\mathcal{F}$ of rooted trees is defined as follows: for two rooted trees T_1 and T_2, $T_1 \sim_{\mathcal{F}} T_2$ if and only if $T_1 \oplus T \in \mathcal{F} \Leftrightarrow T_2 \oplus T \in \mathcal{F}$ for all rooted trees T. Informally, two rooted trees are considered equivalent with respect to $\mathcal{F}$ if they behave equally after compounding with the same tree.

Lemma 3. *Let $\mathcal{F}$ be the set of rooted trees that represent a 3-leaf power with a Hamiltonian cycle. Then, $\sim_{\mathcal{F}}$ does not have finite index.*

Proof. Let G be a graph whose vertex set can be partitioned into a clique A and an independent set B. Furthermore, let every vertex in A be adjacent to every vertex in B. Such graphs are called *complete split graphs*. It obviously holds that G has a Hamiltonian cycle if and only if $|A| \geq |B|$, since two vertices of B cannot appear consecutively on the cycle.

First we show that complete split graphs are 3-leaf powers. Let G be a complete split graph with vertex partition (A, B) such that A is a clique and B is an independent set. A *star* is a tree with a universal vertex. We obtain a tree for G from the star on $|B| + 1$ vertices by attaching a new leaf to every leaf of the star and $|A|$ new leaves to the universal vertex of the star. Let the new tree be rooted at the universal vertex of the star. For $r \geq 1$, denote by G_r the star on $r + 1$ vertices, which is a complete split graph, and denote by T_r the corresponding rooted tree due to the above construction. Note that, for arbitrary i and l, $T_i \oplus G_l$ is a tree for a complete split graph with independent set of size i and clique of size $l + 1$. We assume that G_l has root a universal vertex.

Second we show that all T_r are in different equivalence classes of $\sim_{\mathcal{F}}$, thus showing that $\sim_{\mathcal{F}}$ does not have finite index. Let $1 \leq i < j$. By the discussions about, $T_i \oplus G_{j-i}$ has no Hamiltonian cycle and $T_j \oplus G_{j-i}$ has a Hamiltonian cycle. Hence, $T_i \not\sim_{\mathcal{F}} T_j$.

In view of Corollary 1, the above provides an alternate proof that Hamiltonicity cannot be expressed in M1O logic. It also shows that an attempt to mimic the form of Lemma 2 by having the "extractor" graph expressions in M1O logic, does not support a conclusion that properties expressible in M2O logic are finite-state recognizable by automata acting on the generating trees. The proof of Lemma 3 is easily modified to show a similar result for any fixed

$k \geq 3$. We contrast the above with a limited positive result for a problem that cannot be formulated in M1O.

Lemma 4. *Let $\mathcal{F}$ be the set of rooted trees that represent a 2-leaf power with a Hamiltonian cycle. Then, $\sim_{\mathcal{F}}$ has finite index.*

Proof. Note that 2-leaf powers are exactly the disjoint unions of complete graphs. So, it holds that a 2-leaf power contains a Hamiltonian cycle if and only if it is connected and has at least three vertices. Hence, a 2-leaf power has a Hamiltonian cycle if and only if it is represented by a star with at least three leaves. We show that $\sim_{\mathcal{F}}$ defines three equivalence classes on the set of rooted trees on at least four vertices: stars with root the centre vertex, stars with root a leaf, rooted trees that are not stars.

Let T and T' be rooted trees. First, let T not be a star. This means that there is a path of length at least 3 between two leaves in T. Then, there is a path of length at least 3 between two leaves also in $T \oplus T'$, which means that $T \oplus T'$ represents a disconnected 2-leaf power, and $T \oplus T'$ is not in $\mathcal{F}$. Second, let T be a star with root vertex a leaf. We distinguish two cases for T'. If T' has exactly one vertex then $T \oplus T'$ represents the same 2-leaf power as T, if T' has at least two vertices then $T \oplus T'$ contains two leaves at distance at least 3. Hence, all trees in the second class behave equally. Third, let T be a star with root vertex the centre. Again, we distinguish two cases. If T' is a star with root a universal vertex then $T \oplus T'$ is a star with root a universal vertex, if T' is a star with root not a universal vertex or T' is not a star then $T \oplus T'$ contains a pair of leaves at distance at least 3. Hence, the trees in this class behave equally.

We have seen that $\sim_{\mathcal{F}}$ partitions the set of rooted trees on at least four vertices into three classes. Since there is only a finite number of rooted trees on at most three vertices, $\sim_{\mathcal{F}}$ partitions the set of rooted trees into a finite number of classes, which shows the claim of the lemma.

5 Conclusions and Open Problems

We have shown that *graphs of bounded leaf-power* is not a well-behaved monotonic parameter, answering a noted open problem: there are graphs that are 4-leaf powers but not 5-leaf powers. Recently, we learned that Brandstädt and Wagner have generalized our construction of a 4-leaf power that is not a 5-leaf power, based on an early draft of the present paper [9], to construct for any $k \geq 4$ a k-leaf power that is not a k+1-leaf power [4]. But maybe it can be shown that in some sense the notion is "almost monotonic" for some suitable interpretation of *almost*.

We have also explored how the generating trees for k-leaf powers can be used directly to decide properties of graphs in these families, and have shown if a property is expressible in M1O logic, then this approach can be carried out. Such expressibility is sufficient, but not necessary for the approach to succeed. We have shown that Hamiltonicity, a property that is not expressible in M1O, is finite-state for 2-leaf powers, but not for k-leaf powers for $k \geq 3$.

The issue has been explored in the general setup that the graphs are *extracted* from the trees in some manner, and then we want to know if the trees might be exploited for finite-state recognition of properties. Our general result, Lemma 2, shows that if the extraction is by means of M2O expressions, and the properties are expressible in M1O, then the program succeeds. (We have recently learned that our general lemma seems to be a special case of results in a very general setting independently and recently published by Blumensath and Courcelle [5].)

However, this area merits much more investigation. Our main motivation is to try to find general FPT results for classes of problems, for graphs in graph families extracted from trees, much as Courcelle's Theorem does for graphs of bounded treewidth. But not just general FPT results, we also want to explore how such results can be obtained *in an efficient manner*.

Another possibility for handling the extraction of the graphs from the trees would be by means of finite-state automata. For example, consider the (somewhat artificial) parameterized family of graphs $\mathcal{F}_k$ defined where two leaves l and l' of the generating tree T are adjacent in the extracted graph $G(T)$ if there exists a cutwidth at most k layout of T with u and v consecutive. Using the approach of [8] (Theorem 6.82) one can construct a fairly efficient tree-automaton for this extraction. Is there an efficient and general way to decide properties of such graphs? Can we get good quantitative bounds on the sizes of the automata involved in these kinds of approaches?

References

1. A. Brandstädt. Talk at GROW workshop in Eugene, USA, October 2007.
2. A. Brandstädt, V.B. Le. Structure and linear time recognition of 3-leaf powers. *Information Processing Letters* 98: 133–138, 2006.
3. A. Brandstädt, P. Wagner. On (k, l)-leaf powers. Proceedings of the 32nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2007, Lecture Notes in Computer Science 4708: 525–535, 2007.
4. A. Brandstädt, P. Wagner. On k- versus $(k + 1)$-Leaf Powers. Manuscipt 2008.
5. A. Blumensath and B. Courcelle. Recognizability, hypergraph operations, and logical types. *Information and Computation* 204(6): 853–919, 2006.
6. B. Courcelle. The monadic second order theory of graphs I: recognizable sets of finite graphs. *Information and Computation* 85: 12–75, 1990.
7. B. Courcelle, J. Makowsky and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems* 33(2): 125–150, 2000.
8. R. Downey and M. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
9. M. Fellows, D. Meister, F. Rosamond, R. Sritharan, J.A. Telle. On graphs that are k-leaf powers. Manuscript 2007.
10. F. Gurski, E. Wanke. The Clique-Width of Tree-Power and Leaf-Power Graphs. Proceedings of WG '07, Lecture Notes in Computer Science 4769:76-85, Springer Verlag, 2007.
11. P. Kearney, D. Corneil. Tree Powers. *Journal of Algorithms* 29(1): 111-131, 1998.
12. J.A. Makowsky. Model theory and computer science: an appetizer. In: Handbook of Logic in Computer Science, Vol. 1: 763–814, Oxford University Press, New York, 1992.

13. N. Nishimura, P. Ragde, D.M. Thilikos. On graph powers of leaf-labeled trees. *Journal of Algorithms* 42: 69–108, 2002.
14. S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B* 96(4): 514–528, 2006.