The Nesting-Depth of Disjunctive μ-Calculus for Tree Languages and the Limitedness Problem

Thomas Colcombet[1] and Christof Löding[2]

[1] Liafa, Cnrs and Université Paris Diderot, Case 7014, 75205 Paris Cedex 13, France
thomas.colcombet@liafa.jussieu.fr
[2] RWTH Aachen, Informatik 7, 52056 Aachen, Germany
loeding@cs.rwth-aachen.de

Abstract. In this paper we lift the result of Hashiguchi of decidability of the restricted star-height problem for words to the level of finite trees. Formally, we show that it is decidable, given a regular tree language L and a natural number k whether L can be described by a disjunctive μ-calculus formula with at most k nesting of fixpoints. We show the same result for disjunctive μ-formulas allowing substitution. The latter result is equivalent to deciding if the language is definable by a regular expression with nesting depth at most k of Kleene-stars.

The proof, following the approach of Kirsten in the word case, goes by reduction to the decidability of the limitedness problem for nondeterministic nested distance desert automata over trees. We solve this problem in the more general framework of alternating tree automata.

1 Introduction

For regular languages of finite words, the star-height problem is to determine for a given language the minimal number of nestings of Kleene-stars required in a regular expression describing this language. The star-height can be seen as a measure for the complexity of a regular language. This problem was raised by Eggan in [6] who showed that the star-height of languages induces a strict hierarchy.

The problem was solved 25 years later by Hashiguchi [7] with a very complex proof. Recently, Kirsten has given a new proof [8] using the notion of nested distance desert automata. These automata compute a value (a natural number) for each accepted input. Kirsten showed that the star-height problem can be reduced to the limitedness problem for nested distance desert automata. A nested distance desert automaton is called limited if the mapping it computes is bounded. This problem is solved in [8] using an algebraic approach.

In this paper we lift Kirsten's result from words to trees. The theory of regular languages of finite trees has been initiated in [9,5] and since then it has turned out that a lot of concepts can be generalized from words to the more general setting of trees (see [4] for an overview). Similar to the case of words, regular languages of finite trees can be described by several formalisms, including finite automata and regular expressions. The definition of regular expressions for trees

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 416–430, 2008.

is very similar to the one for words using the operations of union, concatenation, and iteration (Kleene-star). So the star-height problem for regular tree languages is posed in the same way as for words. However it is more convenient to refer to the nesting-depth problem for disjunctive μ-calculus. Disjunctive μ-calculus is another way of defining regular languages of trees. This formalism uses a fix point operator μ in place of the Kleene-star, and comes in two variants: with substitution and without substitution. In both cases, the number of nestings of μ-fix-points induces a strict hierarchy of languages. In the case with substitution, this hierarchy coincides with the star-height hierarchy.

As mentioned above, the proof of Kirsten consists of two main steps: a reduction of the star-height problem to the limitedness problem for nested distance desert automata, and the decidability proof for the limitedness problem. Our proof is along the same lines. The first step is very similar to the one presented in [8]. We define a natural extension of distance desert automata from words to trees (but we prefer to call them cost automata) and reduce the nesting-depth problem (both with and without substitution) for tree languages to the limitedness problem for cost automata. One should note here that in [8] a specific construction is used to obtain an automaton for the given language that has some good properties making the reduction work. We introduce here the notion of subset automata as an abstract notion for capturing the properties of an automaton required for the reduction. Then we show that every regular language of trees can be accepted by such a subset automaton.

The main obstacle for the second step of the proof is that the algebraic objects required for dealing with languages of trees are more complex than the ones that can be used when dealing with words. To describe the behavior of a tree automaton one has to capture the fact that it runs in parallel over many branches of the tree. We solve this problem by reducing the limitedness problem for cost automata to the same problem for the much simpler class of purely non-deterministic automata. These automata, started at the root of the tree, only move to one of the successors of the current node. In this way each of their computations corresponds to a single path through the tree. This reduction uses game-theoretic arguments and even works if we start from an alternating cost automaton instead of a non-deterministic one. Then, proving the decidability of the limitedness problem for purely non-deterministic automata is a technical work that relies on algebraic arguments similar to the proof of Kirsten. New ideas inspired from [2] are required for making the proof compatible with the branching nature of trees.

To sum up, the contributions of this paper are the following:

1. We show the reduction of the nesting-depth problem for tree languages to the limitedness problem for cost automata (Lemma 5). This reduction is done twice, in the case of disjunctive μ-calculus with substitution, as well as without substitution. The new notion of subset automata – that we believe to be of independent interest – is used.
2. We prove the decidability of the limitedness problem in the more general framework of alternating cost automata (Theorem 2). This requires new

game-theoretic arguments for a first reduction to the case of purely nondeterministic automata, as well as an involved variant of the proof of Kirsten.

Combining these results we obtain our main theorem:

Theorem 1. *Given a regular language L of trees the following values and corresponding formulas or expressions can be computed.*

1. *The minimal nesting-depth of a disjunctive μ-calculus formula for L.*
2. *The minimal nesting-depth of a disjunctive μ-calculus formula with substitution for L.*
3. *The minimal star-height of a regular expression for L.*

The paper is organized as follows. In Section 2 we give the main definitions concerning tree automata, disjunctive μ-calculus, and cost automata. Section 3 presents the reduction of the nesting-depth problem to the limitedness problem for cost automata on trees. In Section 4 we show that the limitedness problem for alternating cost automata on trees is decidable.

2 Definitions

In this section we introduce the basics on trees, tree automata, μ-calculus, and regular expressions. The reader not familiar with the subject of regular tree languages is referred to [4] for an introduction.

2.1 Trees and Patterns

We fix from now a *ranked alphabet* A, i.e., a finite set of symbols, together with an arity $|a|$ for every $a \in A$. The set of (finite) *trees* $\mathcal{T}$ is the least set such that for all $a \in A$ and all $t_1, \ldots, t_{|a|} \in \mathcal{T}$, $a(t_1, \ldots, t_{|a|}) \in \mathcal{T}$. In case of $|a| = 0$ we simply write a instead of $a()$. If we want to make the alphabet of labels explicit we also refer to a tree as an A-tree. Given a finite set X of *variables* disjoint from A, the set of X-*patterns* $\mathcal{T}[X]$ is the least set containing X and such that for all $a \in A$ and all $t_1, \ldots, t_{|a|} \in \mathcal{T}[X]$, $a(t_1, \ldots, t_{|a|}) \in \mathcal{T}[X]$. Hence trees are just $\emptyset$-patterns. Sets of trees and sets of patterns are called *languages*. For languages $L_1, \ldots, L_{|a|}$ we denote by $a(L_1, \ldots, L_a)$ the language $\{a(t_1, \ldots, t_{|a|}) \ : \ t_1 \in L_1, \ldots, t_{|a|} \in L_{|a|}\}$.

As usual, the *domain* $dom(t) \subseteq \mathbb{N}^*$ of a tree or pattern $t = a(t_1, \ldots, t_{|a|})$ is defined inductively as $dom(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{|a|} i \cdot dom(t_i)$, and we view t as a mapping from its domain to the alphabet A (and X in the case of patterns). The elements of the domain are called *nodes*. We refer to nodes that are labeled by some $x \in X$ as *variables nodes*. Nodes that are labeled with symbols of arity 0 or variables are called *leaves*. The other ones are called *inner nodes*.

Given two sets X, Y disjoint from A, a mapping v from X to languages of Y-patterns, and an X-pattern t, we denote by $t[v]$ the language of Y-patterns obtained by replacing every $x \in X$ appearing in t by some $t' \in v(x)$. We lift this notation to languages of X-patterns, $K[v] = \bigcup_{t \in K} t[v]$. By $K[x := K']$ we denote the set of patterns that is obtained by taking a pattern from K and replacing each x with some pattern from K'.

2.2 Automata

A *non-deterministic tree automaton* is of the form $\mathcal{A} = (Q, A, In, \Delta)$, where Q is a finite set of *states* disjoint from A, A is the ranked alphabet, $In \subseteq Q$ is the set of *initial states*, $\Delta \subseteq \cup_{a \in A} Q \times \{a\} \times Q^{|a|} \cup Q \times \{\epsilon\} \times Q$ is the *transition relation*. Transitions of the form (q, ϵ, r) are called *ϵ-transitions*, and transitions for symbols of arity 0 are written as (q, a). Given a state q of $\mathcal{A}$ we denote by $\mathcal{A}_q$ the automaton $\mathcal{A}$ with q as only initial state.

To define acceptance of an automaton we use the notion of *run*. If the automaton does not have ϵ-transitions, then a run ρ on a tree t is a Q-tree (in which states can have any arity) with the same domain as t that starts in an initial state and respects the transitions. But as we are working with automata with ϵ-transitions, the domain of a run can be different from the domain of the input tree. Therefore we adopt an inductive definition of runs. Let t be an A-tree.

- If $t = a$ and $(q, a) \in \Delta$, then q is a run of $\mathcal{A}$ on t.
- If $t = a(t_1, \ldots, t_{|a|})$ and $(q, a, q_1, \ldots, q_{|a|}) \in \Delta$, then $q(\rho_1, \ldots, \rho_{|a|})$ is a run of $\mathcal{A}$ on t, where each ρ_i is a run of $\mathcal{A}$ on t_i with state q_i at the root.
- If $(p, \epsilon, q) \in \Delta$ and ρ' is a run of $\mathcal{A}$ on t with state q at the root, then $p(\rho')$ is also a run of $\mathcal{A}$ on t.

A tree t is accepted by $\mathcal{A}$ if there is a run of $\mathcal{A}$ on t that starts in an initial state. The language $L(\mathcal{A})$ is the set of all trees that are accepted by $\mathcal{A}$. A regular language is a language that is accepted by some automaton.

If we want an automaton $\mathcal{A}$ to read X-patterns instead of solely trees, then we explicitly specify the transitions that the automaton can use at leaves labeled with a variable. This is done by giving a relation between states and variables. If R is such a relation, then $\mathcal{A}[R]$ denotes the automaton $\mathcal{A}$ with the additional transitions (q, x) for $(q, x) \in R$.

2.3 Disjunctive μ-Calculus

In this section, we introduce two other formalisms for describing regular languages of trees, namely the disjunctive μ-calculus and regular expressions.

A *disjunctive μ-calculus formula with substitution* (simply *μ-formula with substitution* from now) has the following syntax:

$$\phi ::= \quad \bot \quad | \quad a(\underbrace{\phi, \ldots, \phi}_{|a|}) \quad | \quad \phi + \phi \quad | \quad x \quad | \quad \mu x.\phi \quad | \quad \phi[x := \phi] \ ,$$

in which $a \in A$, and x is a *variable*. If $|a| = 0$ we just write a instead of $a()$. If a μ-formula with substitution does not use the rule $\phi[x := \phi]$, it is simply called a *μ-formula*. One defines the *free variables* of a μ-formula as usual. A μ-formula with no free variables is *closed*.

The semantics $[\![\phi]\!]$ of a μ-formula is the language of patterns defined by:

- $[\![\bot]\!] = \emptyset$, $[\![x]\!] = \{x\}$,
- $[\![a(\psi_1, \ldots, \psi_{|a|})]\!] = a([\![\psi_1]\!], \ldots, [\![\psi_{|a|}]\!])$, $[\![\psi + \psi']\!] = [\![\psi]\!] \cup [\![\psi']\!]$,

– $[\![\phi[x := \psi]]\!] = [\![\phi]\!][x := [\![\psi]\!]]$,
– $[\![\mu x.\psi]\!] = \bigcup_{n\in\mathbb{N}} L_n$, in which $L_0 = \emptyset$ and $L_{n+1} = L_n \cup [\![\psi]\!][x := L_n]$.

A closed μ-formula with substitution defines a regular language of trees. Reciprocally, every regular language of trees is the semantics of a μ-formula.

The *nesting-depth* of a μ-formula ϕ (with or without substitution) is the maximal number of nestings of fixpoint operators:

– $nd(\bot) = nd(x) = 0$,
– $nd(\phi + \phi') = nd(\phi[x := \phi']) = \max(nd(\phi), nd(\phi'))$,
– $nd(a(\phi_1, \ldots, \phi_{|a|})) = \max(nd(\phi_1), \ldots, nd(\phi_{|a|}))$,
– $nd(\mu x.\phi) = 1 + nd(\phi)$.

Remark 1. Regular expressions over words have be extended to trees, see e.g., [4], Chapter 2. Star-height can be defined in this framework as for word languages. This parameter is linked to the nesting-depth as follows: Each regular tree language can be defined by a μ-formula with substitution of nesting-depth k iff it can be defined by a regular expression of star-height k. Hence solving the nesting-depth problem also solves the star-height problem.

2.4 Cost Automata

We now extend the model of tree automata such that trees are not just accepted or rejected but furthermore a cost is computed. For this purpose, we add a function that assigns to each state a priority from a set D. This set D is totally ordered and partitioned into *increments* and *resets*. We can view such an automaton as having as many counters as there are increments. Whenever a state is visited that is assigned an increment, then the corresponding counter is incremented, and all counters for smaller increments (recall that D is ordered) are reset. If the automaton visits a state that is assigned a reset, then all counters for increments that are smaller than this reset are set to 0. If we consider a run of such an automaton, then the cost along a path through the run corresponds to the maximal value of one of the counters. The cost of a run is the maximal cost of all the paths. The cost of a tree is the minimal cost over all runs for this tree.

Formally, a cost tree automaton is of the form $\mathcal{A} = (Q, A, In, \Delta, pri)$, where the first four components are as before, and $pri : Q \to D$ is a priority function. The set D of priorities is totally ordered, and the elements of D are referred to as increments and resets. Usually, it is of the form $D = \{I_1, R_1, \ldots, I_k, R_k\}$ where the I_i are increments, the R_i are resets, and the order is $I_1 < R_1 < \cdots < I_k < R_k$. The same notation is used in [1] for hierarchical B-automata, which work in the same way as our cost automata but on words and not on trees. In [8] the increments are called $\angle_i$ (péage) and the resets $\curlyvee_i$ (source).

A run ρ on a tree t is defined as for standard tree automata. The language $L(\mathcal{A})$ is also defined as if no counters where involved. The difference is that a cost is associated to each run, each tree, and each language by the automaton. We start by defining the cost of a sequence of states. Let $\sigma = p_0 p_1 \cdots p_n \in Q^*$. If all

priorities in σ are at most I_i, then we write $|\sigma|_{I_i}$ for the number of states with priority I_i in σ:

$$|\sigma|_{I_i} = \begin{cases} 0 \text{ if } pri(p_j) > I_i \text{ for some } j, \\ |\{j \ : \ pri(p_j) = I_i\}| \text{ otherwise.} \end{cases}$$

Intuitively, this corresponds to the number of increments to the counter for I_i. The condition that no priority higher that I_i occurs means that the counter is not reset. The cost of σ is defined as

$$val(\sigma) = \max\{|\sigma'|_{I_i} \ : \ i \in [k] \text{ and } \sigma' \text{ is a factor of } \sigma\},$$

where σ' is a factor of σ if $\sigma = \sigma_1\sigma'\sigma_2$ for some σ_1, σ_2. The cost $val(\rho)$ of a run ρ is defined as the maximal cost of all the state sequences along paths through ρ. The cost assigned to a tree t by the automaton $\mathcal{A}$ is:

$$\mathcal{A}(t) = \min\{val(\rho) \ : \ \rho \text{ is a run of } \mathcal{A} \text{ on } t\} \qquad (\text{and } \omega \text{ if } t \notin L(\mathcal{A})).$$

Given a language of trees K, we define its cost for the automaton $\mathcal{A}$ as:

$$\mathcal{A}(K) = \sup\{\mathcal{A}(t) \ : \ t \in K\} \qquad (\text{and } 0 \text{ if } K = \emptyset).$$

This value $\mathcal{A}(K)$ can be ω, and this for two reasons: either if $K \not\subseteq L(\mathcal{A})$, or $K \subseteq L(\mathcal{A})$ but K contains trees of arbitrary high costs.

A cost automaton $\mathcal{A}$ is *limited* if $\mathcal{A}(L(\mathcal{A})) < \omega$. It is *uniformly universal* if $\mathcal{A}(\mathcal{T}) < \omega$. Those two notions are tightly related as follows:

Remark 2. A cost automaton is uniformly universal iff it is both universal (as a standard tree automaton) and limited. Conversely, given a tree automaton $\mathcal{C}$ accepting the language complement of $L(\mathcal{A})$, one can see it as a cost tree automaton of single priority R_1. Then $\mathcal{A}$ is limited iff $\mathcal{A}+\mathcal{C}$ is uniformly universal, in which $\mathcal{A}+\mathcal{C}$ is the disjoint union of the automata $\mathcal{A}$ and $\mathcal{C}$ (as the standard construction for the union of languages).

When we reduce the problems of determining the star-height or the nesting-depth of a language to the uniform universality problem for cost automata, then we use automata with ϵ-transitions. The solution of the uniform universality problem is presented for automata without ϵ-transitions. To justify this we now present a result that allows to remove ϵ-transitions while preserving uniform universality. The proof uses a construction that replaces sequences of ϵ-transitions by a single state whose priority is the maximal priority occurring in this sequence.

Lemma 1. *For each cost automaton $\mathcal{A}$ there exists a cost automaton $\mathcal{B}$ without ϵ-transitions such that for each language K of trees we have $\mathcal{A}(K) < \omega$ iff $\mathcal{B}(K) < \omega$.*

3 From Nesting-Depth to Limitedness

In this section we describe how it is possible to reduce the nesting-depth problem for regular tree languages to the the limitedness problem for non-deterministic cost automata. We present this reduction for the case of μ-formulas with substitution. The case without substitution follows the same lines.

The reduction consists of two parts. In the first one we present subset automata, which are non-deterministic tree automata that have special properties with respect to the subset ordering of languages. In the second part we construct a cost tree automaton and present our main Lemma 6 relating the nesting-depth to the limitedness problem for this automaton.

3.1 Subset Automata

We define in this section the notion of a subset-automaton. Though we do not develop this aspect in the present abstract, we point out that this notion is purely driven by algebraic considerations.

Given a tree automaton $\mathcal{A} = (Q_\mathcal{A}, A, In_\mathcal{A}, \Delta_\mathcal{A})$, the transitions in $\Delta_\mathcal{A}$ are partitioned into ϵ-transitions $\Delta^\epsilon_\mathcal{A}$ and non-ϵ-transitions $\Delta^{\neg\epsilon}_\mathcal{A}$. The automaton is a *subset-automaton* if it satisfies the following items:

1. $\mathcal{A}^{\neg\epsilon} = (Q_\mathcal{A}, A, In_\mathcal{A}, \Delta^{\neg\epsilon}_\mathcal{A})$ is (bottom-up) deterministic and complete, i.e., for all $a \in A$ and $p_1, \dots, p_{|a|} \in Q_\mathcal{A}$, the set $\{p \ : \ (p, a, p_1, \dots, p_{|a|}) \in \Delta^{\neg\epsilon}_\mathcal{A}\}$ is a singleton; we denote by $a_\mathcal{A}(p_1, \dots, p_{|A|})$ its sole element.
2. The relation $p \leq q$ if $(q, \epsilon, p) \in \Delta^\epsilon_\mathcal{A}$ equips $Q_\mathcal{A}$ with a complete sup-semilattice structure, i.e, $\leq$ is an order, and every subset P of $Q_\mathcal{A}$ has a least upper bound $\vee P$ for the order $\leq$ (in particular, there exists a minimum element $\bot_\mathcal{A} = \vee\emptyset$ and a maximal element $\top_\mathcal{A} = \vee Q_\mathcal{A}$).
3. For all $a \in A$, the mapping $a_\mathcal{A}$ is continuous with respect to the sup-semilattice $(Q_\mathcal{A}, \leq)$, i.e., for every $P_1, \dots, P_{|a|} \subseteq Q_\mathcal{A}$,

$$a_\mathcal{A}(\vee P_1, \dots, \vee P_{|a|}) = \vee\{a_\mathcal{A}(p_1, \dots, p_{|a|}) \ : \ p_1 \in P_1, \dots, p_{|a|} \in P_{|a|}\} \ .$$

4. $In_\mathcal{A} = \{q \in Q_A \ : \ q \leq q_0\}$ for some q_0 in $Q_\mathcal{A}$.

Since $\mathcal{A}^{\neg\epsilon}$ is deterministic and complete, given a tree t, there exists a unique state $f(t)$ such that $\mathcal{A}^{\neg\epsilon}_{f(t)}$ has a run over t. Remark that in an algebraic framework, Item 1 means that the mappings $a_\mathcal{A}$ equip $Q_\mathcal{A}$ with a tree algebra structure. The mapping f is nothing but the unique tree algebra morphism from the free algebra of trees to this algebra. As f depends on the automaton $\mathcal{A}$, it should rather be called $f_\mathcal{A}$. However, we drop the subscript to simplify the notation.

Consider now a tree language K, define:

$$f(K) = \bigvee_{t \in K} f(t) \ , \qquad \text{and} \quad F(K) = L(\mathcal{A}_{f(K)}) \ .$$

This extended mapping f is nothing but the extension of the previous morphism to the setting of tree algebras equipped with a complete sup-semi-lattice structure. The corresponding free algebra is the set of tree languages equipped with the inclusion ordering. With this equivalence in mind the following lemmas are natural. Our first lemma makes the morphism properties of f explicit.

Lemma 2. *For all sets of tree languages $Z \subseteq 2^{\mathcal{T}}$, $f\left(\bigcup_{Y\in Z} Y\right) = \bigvee_{Y\in Z} f(Y)$. For all $a \in A$ and tree languages $K_1, \ldots, K_{|a|}$,*

$$f(a(K_1, \ldots, K_{|a|})) = a_{\mathcal{A}}(f(K_1), \ldots, f(K_{|a|})) \ .$$

The second lemma shows how f is related to the semantics of the automaton $\mathcal{A}$.

Lemma 3. *For all states $q \in Q_{\mathcal{A}}$, all trees t, and all tree language K:*

- $t \in L(\mathcal{A}_q)$ *iff* $f(t) \leq q$,
- $K \subseteq L(\mathcal{A}_q)$ *iff* $F(K) \subseteq L(\mathcal{A}_q)$ *iff* $f(K) \leq q$.

The first item of the lemma characterizes the language accepted from state q. The second statement shows that this equivalence can be raised to the level of languages. More precisely, there is a very simple way to check if a language K is included in some $L(\mathcal{A}_q)$ (this is not symmetric and does not work for the superset relation), hence the name of a subset-automaton.

Finally, we need the following:

Lemma 4. *Every regular language is accepted by a subset-automaton.*

There are several ways for proving this lemma, each one of different interest. Here we sketch two possibilities. Let L the tree language for which we want to find a subset automaton.

An adapted version of the construction used by Kirsten [8] starts from an automaton $\mathcal{C}$ (with state set $Q_{\mathcal{C}}$) accepting the *complement* language of L. Then one constructs a non-deterministic automaton $\mathcal{A}$ with state set $2^{Q_{\mathcal{C}}}$ in such a way that for all $P \subseteq Q_{\mathcal{C}}$ and for all trees t: $t \in L(\mathcal{A}_P)$ iff $\forall q \in P,\ t \notin L(\mathcal{C}_q)$. Such an automaton accepts L with initial states $\{P \subseteq Q_{\mathcal{C}} \ : \ In_{\mathcal{C}} \subseteq P\}$ (in which $In_{\mathcal{C}}$ is the set of initial states of $\mathcal{C}$). The states are equipped with a complete sup-semi-lattice structure by $P \leq R$ iff $R \subseteq P$. The automaton obtained by adding the corresponding transitions (R, ϵ, P) to $\mathcal{A}$ yields a subset automaton accepting L. This construction yields an exponential upper bound in the size of an automaton that accepts the complement of the language.

Second, the language theoretic construction consists in considering the set of residuals of L (a language is a residual if it is of the form $\{t \ : \ s[x := t] \in L\}$ in which s is an $\{x\}$-pattern with a single unique occurrence of x). It is classical that every regular tree language L has finitely many residuals. One constructs an automaton that has intersections of residuals as states. Those intersections induce a complete sup-semi-lattice structure for the inclusion. The property of residuals makes the remaining of the construction unique from this point. Once more this construction yields a subset automaton; more precisely, the minimal one.

3.2 Reduction of the Nesting-Depth Problem to Limitedness

The reduction is stated in the following lemma.

Lemma 5. *Given a regular tree language L and a natural number k, there exists effectively a cost tree automaton that is limited iff L can be defined by a μ-formula of nesting-depth at most k. The same statement holds for μ-formulas with substitution.*

We sketch the proof here for the case with substitution. Consider a regular tree language L, a subset automaton $\mathcal{A}$ for the language L, and the corresponding mapping f. We construct for every $k \in \mathbb{N}$ a cost automaton $\mathcal{B}^k = (Q_k, A, In_k, \Delta_k, pri_k)$ and a mapping π from Q_k to $Q_\mathcal{A}$ as follows:

- Q_k is the set of nonempty words over $Q_\mathcal{A}$ of length up to $k+1$, we set $\pi(u)$ to be the last letter of u.
- In_k is $In_\mathcal{A}$, i.e., words consisting of a single initial state of $\mathcal{A}$.
- Δ_k contains all transitions of the form:
  1. $(up, a, up_1, \ldots, up_{|a|})$ whenever $(p, a, p_1, \ldots, p_{|a|}) \in \Delta_A$, (up, ϵ, ur) whenever $(p, \epsilon, r) \in \Delta_\mathcal{A}$,
  2. (up, ϵ, upp),
  3. (uqp, ϵ, up).
- $pri_k(u) = \begin{cases} R_{k+2-|u|} & \text{if } u = vpp \text{ for some } p \in Q_\mathcal{A} \\ I_{k+2-|u|} & \text{else.} \end{cases}$

It should be rather clear that $L(\mathcal{B}^k) = L(\mathcal{A}) = L$. Indeed, every run of $\mathcal{A}$ can be seen as a run of $\mathcal{B}^k$, and conversely every run of $\mathcal{B}^k$ is mapped by π to a run of $\mathcal{A}$ with the same initial state. The key lemma is the following:

Lemma 6. *The cost automaton $\mathcal{B}^k$ is limited iff L is the evaluation of a μ-formula with substitution of nesting-depth at most k. In this case, the μ-formula with substitution can be effectively given.*

Let us give some ideas about the proof. From left to right: this part does not require $\mathcal{A}$ to be a subset automaton. Assuming that $\mathcal{B}^k$ is limited, one obtains a value $N = \mathcal{B}^k(L) < \omega$. The principle is to construct a μ-formula with substitution that is able to 'simulate' the behavior of the automaton $\mathcal{B}^k$ up to the value N of counters.

From right to left. The idea is to prove that for all μ-formulas with substitution ϕ of nesting-depth at most k, every tree in $[\![\phi]\!]$ is accepted by a run of $\mathcal{B}^k$ of cost at most $|\phi|$ (i.e., the size of ϕ) from state $f([\![\phi]\!])$. In practice, this is done via an induction on the structure of ϕ. This means that one has to deal with non-closed formulas and free variables. Hence, our induction hypothesis is more technical: given a μ-formula with substitution ϕ of nesting-depth at most k and free variables X, given a mapping v from X to tree languages,

$$\mathcal{B}^k_{f([\![\phi]\!][v])}[\{(x, f(v(x))) \ : \ x \in X\}]([\![\phi]\!]) \leq |\phi|.$$

There is no special difficulty in the proof itself. It of course relies heavily on the properties of $\mathcal{A}$ and f that we have presented above.

If ϕ has no free variables and evaluates to L, we get that $\mathcal{B}^k_{f(L)}(L) \leq |\phi| < \omega$. Recall that $\mathcal{B}^k$ accepts the language L. Since all trees in L are accepted by $\mathcal{B}^k$, this means by Lemma 3 that $f(L) \in In_k$. Hence $\mathcal{B}^k(L(\mathcal{B}^k)) < \omega$: the cost automaton $\mathcal{B}^k$ is limited.

4 Decidability of the Limitedness Problem

The aim of this section is to show the following theorem.

Theorem 2. *The limitedness for alternating cost tree automata is decidable.*

In our proof we work with the uniform universality problem according to Remark 2. The proof goes in two steps: first we show how to reduce the uniform universality problem of alternating cost tree automata to the one of purely non-deterministic automata, a very weak form of non-deterministic automata (Lemma 10). We then show the decidability of the latter problem (Lemma 11). Among those two parts, we emphasize on the first one which is completely new, while the first one, though more involved, roughly follows the algebraic arguments of [8] in combination with ideas from [2] for handling trees.

The rest of this section is divided as follows. We first introduce in Section 4.1 cost games (a game theoretic counterpart to cost automata) and establish a result on positional strategies for them (Lemma 7). We then use in Section 4.2 cost games in a proof for Lemma 10. In Section 4.3 we present Lemma 11, the decidability of uniform universality for purely non-deterministic automata.

4.1 Cost Games

The semantics of alternating cost automata, which we introduce below, is defined by means of a game. For this reason we first introduce the general terminology for games that we need later.

A *cost game* is of the form $\mathfrak{G} = (V_E, V_A, v_0, E, pri, F)$ with the following components:

- $V := V_E \cup V_A$ is the finite set of vertices, where V_E are the vertices of Eva and V_A are the vertices of Adam (the sets V_E and V_A are disjoint).
- v_0 is the initial vertex.
- $E \subseteq V \times V$ is the set of edges. We require that the graph (V, E) is acyclic.
- $pri : V \to D$ is a priority function where D is as for cost tree automata.
- F is a subset of states that Eva should avoid.

A play σ is a finite sequence of vertices such that successive vertices in the sequence are connected by an edge (note that we consider finite and acyclic games and therefore only finite plays are possible). The cost $val(\sigma)$ of a play σ is defined as for automata with the only difference that the cost is ω if the play contains a vertex from F:

$$val(\sigma) = \begin{cases} \omega \text{ if } \sigma \text{ contains a vertex from } F, \\ \max\{|\sigma'|_{I_i} \ : \ i \in [k] \text{ and } \sigma' \text{ is a factor of } \sigma\} \text{ otherwise.} \end{cases}$$

The notion of strategy for Eva or Adam is defined as usual, it is a function that takes a play ending in a node of the respective player and maps it to one of the possible moves (if such a move exists, otherwise it is undefined). Given two strategies f_E for Eva and f_A for Adam, they define a unique play $\sigma(f_E, f_A)$ from the initial vertex v_0.

The goal of Eva is to minimize the cost of the play while Adam tries to maximize it. If we fix a threshold for the cost, then we can talk about winning strategies: We call a strategy f_E for Eva a winning strategy in $(\mathfrak{G}, val, N)$ if $\max_{f_A} val(\sigma(f_E, f_A)) \leq N$ (where f_A ranges over strategies for Adam). Similarly, a strategy f_A for Adam is called a winning strategy in $(\mathfrak{G}, val, N)$ if $\min_{f_E} val(\sigma(f_E, f_A)) > N$. As the plays of $\mathfrak{G}$ are of finite duration, for each N the game $(\mathfrak{G}, val, N)$ is determined.

Proposition 1. *For each N, either Adam or Eva has a winning strategy in $(\mathfrak{G}, val, N)$.*

From this proposition one can easily deduce that the following equality holds:

$$\min_{f_E} \max_{f_A} val(\sigma(f_E, f_A)) = \max_{f_A} \min_{f_E} val(\sigma(f_E, f_A))$$

where f_E and f_A range over strategies for Eva and Adam. We call the corresponding value the value of the game.

In the reduction from the uniform universality problem for alternating automata to the one for purely non-deterministic ones we want to annotate input trees with strategies of Adam. For this purpose we need positional strategies, i.e., strategies that make their choice only depending on the current vertex and not on the whole history of the play.

Unfortunately, positional strategies are not sufficient for Adam. But for our reduction it is enough if we can guarantee a positional winning strategy for a smaller value. In the following we show that this is indeed possible.

Formally, a *positional strategy* for Adam is a function $f_A : V_A \to V$ such that $(v, f_A(v)) \in E$ for all $v \in V_A$ that have an E-successor, and $f_A(v)$ is undefined otherwise.

Lemma 7. *Let $\mathfrak{G}$ be a cost game with k increments, and let $N \geq 1$. If Adam has a winning strategy in $(\mathfrak{G}, val, N^k - 1)$, then Adam has a positional winning strategy in $(\mathfrak{G}, val, N - 1)$.*

To prove the lemma we want to compute optimal values for Adam at each vertex of the game, and at the same time construct positional strategies in a bottom-up fashion, starting at the nodes without successor. The problem is that the way the value of a play is defined, the optimal choice for Adam at a position might depend on how the play arrived at this position. To avoid this problem we first define a new valuation *pval* ("p" for positional), show that *val* and *pval* are related as indicated in Lemma 7, and prove that this new valuation allows optimal positional strategies for Adam.

The formal definition of *pval* is parameterized by N, i.e., the function should be called $pval_N$ to be more precise. To avoid the subscript we fix some value N for the remainder of this section.

The idea for *pval* is the following. To allow a backward construction (starting from the leaves of the game) we evaluate the plays by reading them right to left. As before, when reading an increment I_i, the corresponding counter is increased and all the smaller ones are reset. But now we view the sequence of the counters as the digits of a single number encoded in base N. In particular, if a counter reaches value N, then it is set back to 0 and the next higher digit (counter) is increased by 1. The goal of Adam is to reach the value N^k, i.e., to exceed the highest value that can be represented with k digits in base N.

Formally, *pval* is a function $pval : V^* \to \{0, \ldots, N-1\}^k \cup \{\omega\}$. The set $\{0, \ldots, N-1\}^k \cup \{\omega\}$ is denoted as N^k_ω. We define *pval* by induction on the length of a play using the operator $\oplus : D \times N^k_\omega \to N^k_\omega$ defined as follows (using infix notation) according to the informal description above:

$$c \oplus (n_k, \ldots, n_1) = \begin{cases} (n_k, \ldots, n_{i+1}, 0, \ldots, 0) \text{ if } c = R_i, \\ \omega \text{ if } c = I_i \text{ and } n_j = N-1 \text{ for all } j \geq i, \\ (n_k, \ldots, n_{j+1}, n_j + 1, 0, \ldots, 0) \text{ for } c = I_i \text{ and the} \\ \qquad\qquad \text{smallest } j \geq i \text{ with } n_j < N-1. \end{cases}$$

and $c \oplus \omega = \omega$. Now we can define *pval* inductively by $pval(\varepsilon) = (0, \ldots, 0)$ and

$$pval(v\sigma) = \begin{cases} \omega & \text{if } v \in F, \\ pri(v) \oplus pval(\sigma) & \text{otherwise.} \end{cases}$$

Our first lemma relates the values computed by *val* and *pval*. The proof is based on a simple analysis of the definitions of the the two measures.

Lemma 8. *Let σ be a play.*

(a) If $val(\sigma) \geq N^k$, then $pval(\sigma) = \omega$.
(b) If $pval(\sigma) = \omega$, then $val(\sigma) \geq N$.

Having established this relation we now look at strategies for Adam when he tries to reach the *pval*-value of ω. We say that a strategy f_A for Adam in $\mathfrak{G}$ is a winning strategy in $(\mathfrak{G}, pval)$ if $\min_{f_E} pval(\sigma(f_E, f_A)) = \omega$. Note that according to Lemma 8 such a winning strategy is also a winning strategy in $(\mathfrak{G}, val, N-1)$.

For $(\mathfrak{G}, pval)$ a positional strategy can be constructed inductively starting at the leaves by always picking the optimal successor.

Lemma 9. *If Adam has a winning strategy in $(\mathfrak{G}, pval)$, then Adam has a positional winning strategy for $(\mathfrak{G}, pval)$.*

Lemma 7 is then a direct consequence of Lemmas 8 and 9.

4.2 Alternating and Purely Non-deterministic Automata

An alternating automaton in general does not send exactly one state to each successor of a node in a tree but it can send several states into the same direction and also no state at all in certain directions. For the formal definition we let

$\Gamma = \{1, \ldots, r\}$, where r is the maximal rank of a letter in the alphabet A. An *alternating cost automaton* is of the form $\mathcal{A} = (Q, A, In, \delta, pri)$, where

- Q is a finite set of *states*,
- A is the alphabet,
- $In \subseteq Q$ is the set of initial states,
- $\delta : Q \times A \to B^+(\Gamma \times Q)$ is the transition function, mapping $Q \times A$ to positive boolean combinations over $\Gamma \times Q$, such that $\delta(q, a)$ is an element of $B^+(\{1, \ldots, |a|\} \times Q)$. If $|a| = 0$, then this set contains only the formulas *true* and *false*.
- $pri : Q \to D$ is a *priority function*.

A cost automaton is called *purely non-deterministic* if the formulas in the transition function only use disjunctions for symbols of arity at least 1.

Often it is convenient to view the transition function of an alternating automaton as a mapping $\delta : Q \times A \to 2^{2^{\Gamma \times Q}}$, where $\delta(q, a) = \{P_1, \ldots, P_n\}$ corresponds to the following formula in DNF: $\bigvee_{i=1}^{n} \bigwedge_{(h,p) \in P_i} (h, p)$. Whenever we write $P \in \delta(q, a)$, then we refer to this representation of automata.

The semantics of a cost automaton is defined by means of a cost game $\mathfrak{G}_{\mathcal{A},t}$ that we describe in the following.

- The vertices or positions of the game are $V = V_E \cup V_A$ with

  $V_E = (Q \times dom(t)) \cup \{v_0\}$ and
  $V_A = \{(u, P) \ : \ u$ is an inner node of t and $P \in \delta(q, t(u))$ for some $q \in Q\}$.

- The initial position is v_0.
- The edges are defined as follows:
  - From v_0 there are edges to (q, ε) for all $q \in In$.
  - From a position (q, u) where u is not a leaf, Eva can move to all positions (u, P) with $P \in \delta(q, t(u))$, i.e., she chooses one of the sets specified by the transition function for state q at node u. (At positions (q, u) where u is a leaf the game stops.)
  - From position (u, P) Adam can move to all positions (p, uh) with $(h, p) \in P$, i.e., Adam chooses one pair of direction and state from P and then moves correspondingly in the tree.
- The priority function is defined by extending pri of $\mathcal{A}$ to the vertices of the game by setting $pri(q, u) = pri(q)$ and $pri(u, P)$ to be some reset smaller than all other elements of D (the vertices of the form (u, P) do not have any influence on the cost of the play).
- The set F contains all vertices (q, u) such that u is a leaf and $\delta(q, t(u)) = false$.

The cost of t is defined as the value of the game:

$$\mathcal{A}(t) = \min_{f_E} \max_{f_A} val(\sigma(f_E, f_A)) \left[= \max_{f_A} \min_{f_E} val(\sigma(f_E, f_A)) \right]$$

where f_E and f_A range over strategies for Eva and Adam. One can note that for the case of non-deterministic automata this value is the same as the one defined using runs. Strategies of Eva correspond to runs and strategies of Adam select a path through the run. As before we extend the definition to languages of trees: $\mathcal{A}(K) = \sup\{\mathcal{A}(t) \ : \ t \in K\}$.

We now come to the reduction from alternating to purely non-deterministic automata, which is based on the following idea: An alternating automaton $\mathcal{A}$ is uniformly universal if there exists an N such that for each t Eva has a winning strategy in $(\mathfrak{G}_{\mathcal{A},t}, val, N)$. If the game is won by Eva this means that Adam does not have a winning strategy. The purely non-deterministic automaton $\mathcal{B}$ that we construct works over trees that are annotated with strategies for Adam. The aim is to check that the strategy of Adam fails, which can be done in a purely non-deterministic way. If $\mathcal{A}$ is uniformly universal, then all strategies of Adam fail and hence $\mathcal{B}$ is also uniformly universal.

Lemma 10. *For each alternating cost automaton $\mathcal{A}$ one can construct a purely non-deterministic cost automaton $\mathcal{B}$ such that $\mathcal{A}$ is uniformly universal iff $\mathcal{B}$ is uniformly universal.*

4.3 Uniform Universality of Purely Non-deterministic Tree Automata

What remains to be shown is the following lemma:

Lemma 11. *It is possible, given a purely non-deterministic cost tree automaton, to decide whether it is uniformly universal or not.*

The reduction done so far, to purely non-deterministic automata, has led us to an almost word-theoretic problem. The proof of Lemma 11 relies on word-related considerations. In particular the proof heavily relies on the theory of semigroups, in a way similar to the proof of Kirsten [8]. The principal difficulty is to make the proof of Kirsten compatible with the tree nature of the problem. This is done using ideas originating from [2]. The proof itself is long and technical.

5 Conclusion

We have shown that the problems of nesting-depth of the disjunctive μ-calculus (with and without substitution) for regular tree languages are decidable. The proof uses cost tree automata, a tree version of the model of nested distance desert automata used by Kirsten in [8] for deciding the star-height of regular word languages. The main new contributions are the notion of subset automata, an abstract description of tree automata that allow the reduction of the star-height or nesting-depth problem to the limitedness of cost automata, the reduction of the uniform universality problem for alternating cost automata to the same problem for purely non-deterministic automata, and the adaption of the algebraic methods from [8] to the tree setting.

Possible future work includes the study of other complexity measures for tree languages. The most natural one is the nesting-depth for μ-calculus (that is for formulas allowing furthermore the intersection). This problem is open in the word case, and is referred to as the semi-restricted star-height in the framework of word languages (the restricted star-height being the one posed by Eggan and studied by Hashigushi, Kirsten, and in this work, and the generalized star-height corresponding to regular expressions with complementation, for which we do not even know if the hierarchy is strict). Another complexity measure concerns the number of distinct variables used in μ-formulas or regular expressions, and more precisely the number of variables used in fix-points (as opposed to variables used for substitutions). For those different problems, reduction to limitedness questions seems the natural path to follow.

Furthermore, we hope to be able to adapt the game-theoretic framework presented in Section 4 also to the setting of infinite trees (the remaining of the proof of limitedness being easy to adapt to this framework). This would be a major step for solving the problem of parity rank for regular languages of infinite trees [3], i.e., the problem of finding the minimal number of priorities required for a parity automaton that accepts a given regular language of infinite trees (a parameter also known as Mostowski index and tightly connected to the Rabin index).

References

1. Bojanczyk, M., Colcombet, T.: Bounds in ω-regularity. In: Proceedings of LICS 2006, pp. 285–296. IEEE Computer Society Press, Los Alamitos (2006)
2. Colcombet, T.: A combinatorial theorem for trees. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 901–912. Springer, Heidelberg (2007)
3. Colcombet, T., Löding, C.: The non-deterministic Mostowski hierarchy and distance-parity automata. In: Proceedings of ICALP 2008. LNCS, vol. 5126, pp. 398–409. Springer, Heidelberg (2008)
4. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Löding, C., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007), `http://tata.gforge.inria.fr`
5. Doner, J.: Tree acceptors and some of their applications. Journal of Computer and System Sciences 4, 406–451 (1970)
6. Eggan, L.C.: Transition graphs and the star-height of regular events. Michigan Math. J. 10(4), 385–397 (1963)
7. Hashiguchi, K.: Algorithms for determining relative star height and star height. Inf. Comput. 78(2), 124–169 (1988)
8. Kirsten, D.: Distance desert automata and the star height problem. RAIRO 3(39), 455–509 (2005)
9. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. Mathematical Systems Theory 2(1), 57–81 (1968)