Deciding Nondeterministic Hierarchy of Deterministic Tree Automata

Damian Niwinski, Igor Walukiewicz

Deciding Nondeterministic Hierarchy of Deterministic Tree Automata

Damian Niwiński [1,2]

Institute of Informatics
Warsaw University
Warsaw, Poland

Igor Walukiewicz [3]

LaBRI
University of Bordeaux I
Bordeaux, France

Abstract

We show an algorithm which, for a given deterministic parity automaton on infinite trees, computes the minimal Mostowski (or Rabin) index of a *nondeterministic* automaton recognizing the same language. This extends a previous result of Urbański on deciding if a given deterministic Rabin automaton is equivalent to a nondeterministic Büchi automaton. The algorithm runs in the time of verifying the non-emptiness of nondeterministic parity automata.

Key words: Parity tree automata, Mostowski index, decidability.

1 Introduction

Finite–state automata running in infinite time constitute an automata-theoretic counterpart of many logics relevant to verification, such as μ-calculi, temporal logics, and the monadic second-order logic. For logics referring to branching time, automata on infinite trees seem to be optimal choice. A well-known paradigm translates a formula into an automaton recognizing its tree models, thus reducing model-checking to the non-emptiness problem for tree automata.

The semantical complexity of temporal formulas is reflected by the structure of automata, in particular by the acceptance condition. Today the most

[1] Supported by the European Research Training Network GAMES. The first author was additionally supported by the Polish KBN grant No. 4 T11C 042 25.

[2] Email: niwinski@mimuw.edu.pl

[3] Email: igw@labri.fr

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

common variant is the *parity* condition, which reveals subtle correspondences between automata, the μ-calculus, and games [3]. Parity automata can be organized into a hierarchy according to their *Mostowski indices*[4] (see Figure 1 below). Understanding the structure of this hierarchy helps us to understand the trade–off between expressiveness and efficiency in the model-checking method.

It is known that the hierarchy of Mostowski indices is strict for all kinds of tree automata: deterministic [18], nondeterministic [10], alternating [1] (building on [2,7]), as well as the so-called weak alternating automata [9]. However, very little is known about the *effectiveness* of these hierarchies, that is, whether we can *compute* the minimal Mostowski index of a tree language, starting from any given automaton.

The problem appears somewhat easier if the *input* automaton is deterministic. Deterministic tree languages form a proper, but effective, subclass of all recognizable tree languages (we can determinize an automaton in EXPTIME [13], whenever possible). Computing the level in the deterministic hierarchy can be accomplished by reduction to an analogous problem for word automata [12], see Remark 2.4 below. Note however that the level of a deterministic language in a nondeterministic hierarchy can be arbitrarily smaller than in the deterministic one[5].

Concerning nondeterministic hierarchy, Urbański [17] showed how to decide if a given deterministic Rabin automaton is equivalent to a Büchi automaton (possibly nondeterministic). In the present paper we extend this result by showing an algorithm which, for a given deterministic parity automaton, computes its exact Mostowski index in the nondeterministic hierarchy. To complete the picture, note that the relation of deterministic languages to alternating hierarchy is effective for easy reasons, because they are all co-Büchi, hence on the level $(0, 1)$ of the alternating hierarchy.

To show our result, we refine the technique introduced in [12], where we solved the problem for tree languages $\forall L$, where $L \subseteq \Sigma^\omega$ and '$\forall$' is understoodd in the CTL manner (that is, $t \in \forall L$ if the ω-words read along all paths of t are in L). There, computing the nondeterministic index of the tree language $\forall L$ reduced to detecting some special patterns in a deterministic (word) automaton for L, which we called flowers.

An arbitrary deterministic tree language can be characterized quite similarly if we take into consideration both labels and directions of paths (e.g., for binary trees, the alphabet of paths becomes $\Sigma \times \{l, r\}$). It turns out that the nondeterministic index of the tree language depends again on the presence of some flower-like patterns in the deterministic automaton for the path language.

[4] Here we credit A. W. Mostowski, who first considered [8] tree automata with such accepting condition. The Mostowski indices refine the *Rabin indices* [14].See [16] for relations between various kinds of automata.

[5] It follows easily from the fact that all recognizable *word* languages can be recognized by Büchi automata, while the deterministic hierarchy is infinite [18].

Searching for flowers in a deterministic word automaton can be carried on in polynomial time, however the construction also requires detection of unproductive states of the input tree automaton. This amounts to solving the non-emptiness problem, the question whose exact complexity is currently unknown (estimated by UP $\cap$ co-UP [5]).

2 Basic notions

Automata on infinite words.

A finite nondeterministic parity automaton on infinite words is presented by $\mathcal{A} = \langle \Sigma, Q, q_I, Tr, rank \rangle$, where Σ is a finite alphabet, Q is a finite set of *states* with an *initial state* q_I, $Tr \subseteq Q \times \Sigma \times Q$ is a set of *transitions*, and $rank : Q \to \omega$ is the *ranking* function. A transition (q, a, p) is usually written $q \stackrel{a}{\to} p$.

A *run* of an automaton A on an infinite word $u \in \Sigma^\omega$ can be presented as an infinite word $\rho \in Q^\omega$ such that $\rho(0) = q_I$, and $\rho(m) \stackrel{a}{\to} \rho(m+1)$, whenever $u(m) = a$, for every $m < \omega$. As usual, the run ρ is *accepting* if $\limsup_{n\to\infty} rank(\rho(n))$ is *even*; in other words, the highest rank repeating infinitely often is even. The language $L(A)$ recognized by A consists of those words in Σ^ω for which there exists an accepting run.

Automata on infinite trees.

A full binary tree valued (labeled) in a finite alphabet Σ is a mappings $t : \{l, r\}^* \to \Sigma$, we denote the set of all such trees by T_Σ.

A *nondeterministic parity tree automaton* $\mathcal{A} = \langle \Sigma, Q, q_I, Tr, rank \rangle$ is like an automaton on words except for that $Tr \subseteq Q \times \Sigma \times Q \times Q$. A *run* of $\mathcal{A}$ on a tree $t \in T_\Sigma$ is itself a Q–valued tree $\rho : \{l, r\}^* \to Q$ such that $\rho(\mathrm{e}) = q_I$, and, for each $w \in \mathrm{dom}(\rho)$, $\langle \rho(w), a, \rho(wl), \rho(wr) \rangle \in Tr$, whenever $t(w) = a$. A *path* in ρ is *accepting* if the highest rank occurring infinitely often along it is even. More formally, for a path $P = p_0 p_1 \ldots \in \{l, r\}^\omega$, this means that $\limsup_{n\to\infty} rank(\rho(p_0 p_1 \ldots p_n))$ is even. A *run is accepting* if so are all its paths. The tree language $T(\mathcal{A})$ *recognized* by $\mathcal{A}$ consists of those trees in T_Σ which admit an accepting run.

Deterministic automata.

An automaton on words, or on trees, is *deterministic* if Tr is a partial function from $Q \times \Sigma$ to Q, or to $Q \times Q$, respectively. It is well-known that a parity word automaton can be always converted into a deterministic one but a tree automaton in general cannot. We call a tree language *deterministic* if it is recognized by a deterministic parity automaton.

It will be profitable to identify a deterministic tree automaton A as above with a (deterministic) automaton on infinite words $\mathcal{A}_w = \langle \Sigma \times \{l, r\}, Q, q_I, Tr_w, rank \rangle$,

where

$$Tr_w = \{q \xrightarrow{a,l} q_1,\ q \xrightarrow{a,r} q_2 : (q, a, q_1, q_2) \in Tr\}$$

A *labeled path* in a tree $t : \{l, r\}^* \to \Sigma$ is an infinite sequence $(\sigma_0 p_0), (\sigma_1 p_1), (\sigma_2 p_2) \ldots$, such that $\sigma_i \in \Sigma$, $p_i \in \{l, r\}$, and $t(p_0 \ldots p_{i-1}) = \sigma_i$ (so in particular $t(\varepsilon) = \sigma_0$). It should be clear that $\mathcal{A}$ recognizes a tree t if A_w recognizes all labeled paths of t. Conversely, any *deterministic* word automaton over $\Sigma \times \{l, r\}$ induces a (deterministic) tree automaton over Σ in the obvious manner. In the sequel we will usually not distinguish notationally between $\mathcal{A}$ and $\mathcal{A}_w$, but it will be clear from the context if we view it as an automaton on words or on trees.

Hierarchy of Mostowski indices

The *Mostowski index* of a parity automaton $\mathcal{A}$ is the pair (min (*rank*(Q)), max (*rank*(Q))). We let $(\iota, \kappa) \sqsubseteq (\iota', \kappa')$ if either $\iota' \leq \iota$ and $\kappa \leq \kappa'$ or $\iota = 0$, $\iota' = 1$, and $\kappa + 2 \leq \kappa'$. It is easy to see that, if $(\iota, \kappa) \sqsubseteq (\iota', \kappa')$ then any automaton of index (ι, κ) can be transformed into an equivalent automaton of index (ι', κ') by modification of ranks. Therefore, for any type of automata, the Mostowski indices induce a hierarchy of (tree) languages depicted on the Figure 1. (Without loss of generality we may assume that min(*rank*(Q)) $\in \{0, 1\}$; otherwise scale down the rank by *rank*$(q) :=$ *rank*$(q) - 2$.)

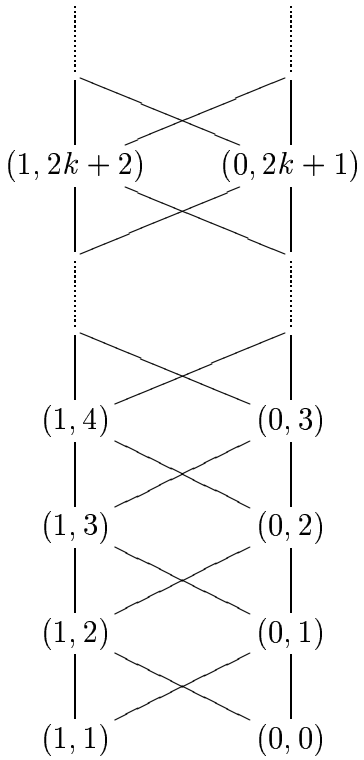


Fig. 1. Hierarchy of Mostowski indices.

It is known that the hierarchy of Figure 1 is *strict* for deterministic automata on words and trees [6] [18], and for nondeterministic automata on trees [10] (also for alternating automata which we do not consider here). We recall the examples from [10,11], because they are related to our proof.

For $n \in \mathbb{N}$, let M_n be the set of trees t over alphabet $\{0, 1, \ldots, n\}$, such that for any path $u \in \{l, r\}^\omega$ of t, $\limsup_{i\to\infty} t(u_i)$ is *even*. Let N_n be defined similarly, with 'even' replaced by 'odd'. We call a tree language L, *ι-n-feasible* if there is a nondeterministic parity automaton of index (ι, n) recognizing L. Otherwise L is *ι-n-unfeasible*.

Theorem 2.1 ([10,11]) *For $n \in \mathbb{N}$: (i) M_n is 0-n-feasible but 1-$(n+1)$-unfeasible; (ii) N_n is 1-$(n+1)$-feasible but 0-n-unfeasible.*

Flowers

For an integer k, a *k-loop* in a deterministic word automaton $\mathcal{A}$ is a path $v_1, \ldots, v_j = v_1$ in the automaton graph (with $j > 1$), such that $\max\{rank(v_i) : i = 1, \ldots, j\} = k$. Given integers $m \leq n$, a state $q \in Q$ is a *m-n-flower* in $\mathcal{A}$ if for every $k = m, \ldots, n$, there is, in the graph of $\mathcal{A}$, a k-loop containing q. We have introduced this concept in [12], together with a rank lifting operation on automata, $\uparrow^i$ (for $i \in \mathbb{N}$), which does not change states and transitions of an automaton, but may, for some states, shift ranks smaller that i (maximally to $i+1$). We need not the details of this operation here, so we only summarize the results to be used.

Lemma 2.2 ([12]) *For a deterministic word automaton $\mathcal{A}$, let $\mathcal{B} = \mathcal{A} \uparrow^0 \uparrow^1 \ldots \uparrow^i$. Then $L(\mathcal{B}) = L(\mathcal{A})$ and moreover if a state q has the priority $m \leq i$ in $\mathcal{B}$ then q is a m-i-flower in $\mathcal{B}$.*

We will use the following consequence of this lemma.

Lemma 2.3 *If n is greater than all ranks of the states of $\mathcal{A}$ then the maximal rank in any strongly connected component (SCC) of $\mathcal{A} \uparrow^0 \uparrow^1 \ldots \uparrow^n$ is n or $n+1$.*

Proof. By the property of $\uparrow^i$, it can be maximally $n+1$. Now if a state q has $rank(q) = i \leq n$ in $\mathcal{A} \uparrow^0 \uparrow^1 \ldots \uparrow^n$ then by the previous lemma it lies on some n-loop, which is of course contained in the SCC. □

Remark 2.4 In [12] we have also showed how to determine the deterministic Mostowski index of a word automaton, by analyzing the flowers in the $\uparrow$-modified automaton. Together with the aforementioned correspondence between deterministic tree automata and word automata for labeled paths, this implies a procedure to determine the level of a deterministic tree language in the deterministic hierarchy. As we have also showed [13] how to transform a

[6] Strictly speaking, Wagner [18] did not considered trees, but the result follows easily from the word case; it also follows from [10] because the examples there are deterministic.

nondeterministic tree automaton into a deterministic one whenever it is possible (within the EXPTIME bound), the case of deterministic hierarchy can be considered settled.

3 Forbidden flower patterns

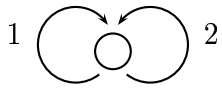
Now for each Mostowski index (ι, n), we will define a flower–like pattern $P(\iota, n)$, that is a family of subgraphs, which may occur in a deterministic word automaton over $\Sigma \times \{l, r\}$. Recall that, by the previous section, such an automaton corresponds to a tree automaton over Σ. Considering the indices $(1, n)$ and $(0, n-1)$ as dual, the idea is to show that if $\mathcal{A}$ contains a $P(\iota, n)$ pattern then $T(A)$ cannot be recognized by a nondeterministic tree automaton with the index dual to (ι, n). The patterns will be constructed in regular way starting from $P(0, 2)$ and $P(1, 3)$, but the basic cases are somewhat different.

We will use letters $a, b, c, \ldots$ for states. Let $a \rightsquigarrow b$ be a short notation for a path $a = v_1, \ldots, v_j = b$ in the automaton graph. (We always assume that $j > 1$, i.e., the path goes through at least one edge.) We will write $a \overset{k}{\rightsquigarrow} b$ if moreover this is a *k-path*, i.e., $\max\{rank(v_i) : i = 1, \ldots, j\} = k$. So a path $a \overset{k}{\rightsquigarrow} a$ is a k-loop.

We say that two paths $a = v_1, \ldots, v_j = b$ and $a = w_1, \ldots, w_\ell$ *split at* a if there exist two transitions $a \overset{\sigma,p}{\to} v_2$ and $a \overset{\sigma',p'}{\to} w_2$, such that $\sigma = \sigma'$, but $p \neq p'$.

3.1 The $(1, 2)$ *case*

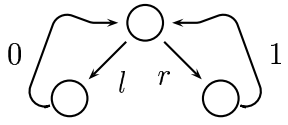
A $P(1, 2)$ pattern consists of a point a and two loops $a \overset{1+2\alpha}{\rightsquigarrow} a$ and $a \overset{2+2\alpha}{\rightsquigarrow} a$ (they need not split):



Note that, at the figures, we present patterns with the smallest possible ranks, keeping in mind that shifting them by the same even number produces a pattern of the same class.

3.2 The $(0, 1)$ *case*

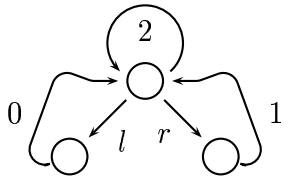
A $P(0, 1)$ pattern consists of two loops $a \overset{0+2\alpha}{\rightsquigarrow} a$ and $a \overset{1+2\alpha}{\rightsquigarrow} a$ which split at a:



(Of course the picture represents only one of the two symmetric cases.)

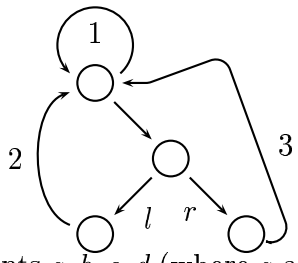
3.3 The $(0,2)$ *case*

A $P(0,2)$ pattern consist of three loops $a \overset{0+2\alpha}{\rightsquigarrow} a$, $a \overset{1+2\alpha}{\rightsquigarrow} a$, and $a \overset{2+2\alpha}{\rightsquigarrow} a$, where the first two split at a (notice that the third one need not split with any of them).



3.4 The $(1,2,3)$ *case*

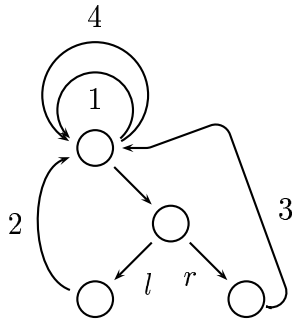
A $P(1,3)$ pattern is a bit more complicated:



It can be presented by points a, b, c, d (where a and b need not be different), together with a loop $a \overset{1+2\alpha}{\rightsquigarrow} a$ and the paths $a \rightsquigarrow b$, $b \rightsquigarrow c$, $b \rightsquigarrow d$, $c \rightsquigarrow a$, and $d \rightsquigarrow a$, such that the composition $b \rightsquigarrow c \rightsquigarrow a \rightsquigarrow b$ forms a $2+2\alpha$-loop, the composition $b \rightsquigarrow d \rightsquigarrow a \rightsquigarrow b$ forms a $3+2\alpha$-loop, and these two loops split at b.

3.5 The $(1,n)$ *case,* $n \geq 4$

A $P(1,4)$ pattern is obtained from a $P(1,3)$ pattern as above, by adding a $4+2\alpha$ loop in a:

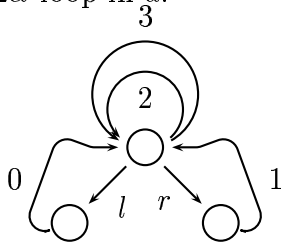


More generally, for $n \geq 4$, a $P(1,n)$ pattern is obtained from a $P(1,n-1)$

pattern (with a shifting parameter 2α) by adding a loop $a \overset{n+2\alpha}{\rightsquigarrow} a$.

3.6 The $(0, n)$ case, $n \geq 3$

Similarly to the previous case, a $P(0,3)$ pattern is obtained from a $P(0,2)$ pattern by adding a $3+2\alpha$ loop in a:



More generally, for $n \geq 3$, a $P(0,n)$ pattern is obtained from a $P(0,n-1)$ pattern (with a shifting parameter 2α) by adding a loop $a \overset{n+2\alpha}{\rightsquigarrow} a$.

A state q of automaton $\mathcal{A}$ is *productive* if $\mathcal{A}$ accepts some tree from q, that is $T(\mathcal{A}_q) \neq \emptyset$, where $\mathcal{A}_q$ is $\mathcal{A}$ with the initial state replaced by q. A pattern is productive if so are *all* states occurring in it (that is, the states distinguished by the construction, as well as the states on the paths). Let $\overline{(\iota, n)}$ denote the index dual to (ι, n).

We are ready to state the following.

Theorem 3.1 *If a deterministic tree automaton $\mathcal{A}$ contains a productive $\overline{(\iota, n)}$ pattern ($\iota \in \{0,1\}$) then $T(\mathcal{A})$ cannot be recognized by a nondeterministic tree automaton of index (ι, n).*

Proof (Idea) We follow a general method of the proofs of hierarchy results previously explored in [10] and in [12] which in turn followed the original idea of Rabin [15], who first showed that (in our notation) M_1 cannot accepted by an automaton of index $(1,2)$.

Given a hypothetical automaton of m states, one develops the forbidden pattern into a tree in order to "fool" the automaton. Productiveness is used to complete the non existing subtrees. The argument is recursive starting from the levels $(0,3)$ and $(1,4)$, but the basic levels require some special constructions. □

4 On the positive side

A more difficult direction is to show that if an automaton $\mathcal{A}$ does not contain a forbidden pattern then $T(\mathcal{A})$ can indeed be recognized by a nondeterministic automaton of the required index (which is in general smaller than the index of $\mathcal{A}$).

Theorem 4.1 *If a deterministic tree automaton $\mathcal{A}$ does not contain any productive $\overline{(\iota, n)}$ pattern ($\iota \in \{0, 1\}$) then $T(\mathcal{A})$ is recognized by a nondeterministic tree automaton of index (ι, n).*

The proof splits into several cases depending on (ι, n). A typical argument will consist in decomposing an automaton $\mathcal{A}$ (viewed as automaton on words) into strongly connected components, and applying inductive arguments to the sub-automata induced this way.

In what follows we make a proviso that the automaton $\mathcal{A}$ has only productive states; therefore all patterns in consideration are also productive. Recall that we call a tree language (ι, n)-feasible if it can be recognized by a nondeterministic automaton of index (ι, n).

4.1 The $(0, 1)$ case

Lemma 4.2 *If there is no $P(1, 2)$ pattern in $\mathcal{A}$ then $T(\mathcal{A})$ is $(0, 1)$-feasible.*

Proof. It follows from the Flower Lemma of [12] that $\mathcal{A}$ is a (deterministic) $(0, 1)$-automaton, hence $\mathcal{A}$ itself suffices. □

4.2 The $(1, 2)$ case

Lemma 4.3 *If $\mathcal{A}$ does not have $P(0, 1)$ pattern then $T(\mathcal{A})$ is $(1, 2)$-feasible.*

Although this case was already settled in [13], we sketch another proof here, which will serve as the basis of inductive argument.

Proof (Sketch) Let $\widehat{\mathcal{A}} = \mathcal{A} \uparrow^0 \uparrow^1 \ldots \uparrow^n$ where n is an odd number greater than the biggest rank in $\mathcal{A}$.

Given a tree t in $T(\mathcal{A})$, there is a unique run of $\widehat{\mathcal{A}}$ on t. This defines parts of the tree accepted by different strongly connected components (SCCs) of $\widehat{\mathcal{A}}$. We can have an automaton without acceptance conditions that calculates in each node the state of the unique run of $\mathcal{A}$ on t. The automaton we want to construct will be a product of this automaton and $(1, 2)$ automata, one for each SCC of $\widehat{\mathcal{A}}$. The role of the latter automata will be to check if all paths of the run of $\widehat{\mathcal{A}}$ that stay forever in a given SCC are accepting. The composition of these automata will give us $(1, 2)$ automaton recognizing $T(\mathcal{A})$.

Consider first an SCC, C say, with maximal rank n. We know that n is odd and that in C there is no $P(0, 1)$ pattern. This means that there is no state in C of rank $< n$ with arrows on l and r directions leading to C. If there were such a state then in one direction we would have a loop on $n - 1$ and on the other a loop on n.

Hence, the part of the run of $\mathcal{A}$ on t staying in C is a tree were the only splits (states with l and r arrows pointing to elements of C) are in nodes with states of rank n. If the run is accepting then in this part there can be only finitely many occurrences of states of rank n as n is odd. A $(1, 2)$ automaton

can recognize whether in such part all the paths are accepting. It can wait till there are no more splits and then use $(1,2)$ condition to recognize that the remaining path is accepting. (Recall that any automaton on words can be simulated by a nondeterministic Büchi, i.e., $(1,2)$-automaton [16].)

The other case is when a maximal rank in a SCC component C of $\widehat{\mathcal{A}}$ is $n+1$. It is even as n is odd. Consider the SCCs of the graph $C - \{q : \widehat{rank}(q) = n+1\}$. By the above argument each such SCC can be simulated by a $(1,2)$ automaton [7]. Hence to recognize whether all the paths staying in C are accepting we take all these automata, put them together in the same way as SCCs of C are put together and change the rank of $n+1$ to 2.

□

4.3 The $(1,3)$ case

Lemma 4.4 *If $\mathcal{A}$ does not have a $P(0,2)$ pattern then $T(\mathcal{A})$ is a $(1,3)$-feasible.*

Proof. Again, let $\widehat{\mathcal{A}} = \mathcal{A}\uparrow^0\uparrow^1 \ldots \uparrow^n$ where n is an even number greater than the biggest priority in $\mathcal{A}$. As in the previous proof, for each SCC C of $\mathcal{A}$ we construct a $(1,3)$-automaton checking whether every path of the run of $\mathcal{A}$ staying in C is accepting.

Take an SCC C with maximal n. As n is even we know that in $C \setminus \{q : \widehat{rank}(q) = n\}$ there is no $P(0,1)$ pattern. Hence, by the result of Lemma 4.3, for each SCC of $C \setminus \{q : \widehat{rank}(q) = n\}$ we have an $(1,2)$ automaton verifying a part of the run staying in this SCC. Then for the whole C we compose these automata exactly in the same way as SCCs of C are composed and then change all ranks n to 2.

Take an SCC C with maximal $n+1$. The part $C \setminus \{q : \widehat{rank}(q) = n+1\}$ is equivalent to a $(1,2)$ automaton by the above paragraph. Hence C is equivalent to $(1,2,3)$ automaton when we change rank $n+1$ to 3. □

4.4 The $(0,2)$ case

Lemma 4.5 *If there is no $P(1,3)$ pattern in $\mathcal{A}$ then $T(\mathcal{A})$ is $(0,2)$-feasible.*

Proof. Let $\widehat{\mathcal{A}} = \mathcal{A}\uparrow^0\uparrow^1 \ldots \uparrow^n$ where n is an odd number greater than the biggest priority in $\mathcal{A}$. As before it is enough to show for each SCC of $\widehat{\mathcal{A}}$ how to recognize its language by a $(0,1,2)$ automaton.

Consider an SCC C with maximal rank n. Recall that n is odd. The first step is to consider $C - \{q : \widehat{rank}(q) = n\}$ and the SCCs in it. Suppose that in one such SCC D there is a vertex x with a split, i.e, the arrows on both directions l and r. As $rank(x) \leq n-1$ we have a $(n-1)$-loop through x (there must be vertex of rank $(n-1)$ in D) in one of these directions and an n-loop

[7] Here and further we freely consider SCCs as tree automata. Strictly speaking, they require completion by some dummy states.

in C in the other (as all the nodes are from C). If in D there is a node of y with $\widehat{rank}(y) < n-1$ then we have a $rank(y)$-loop ($rank(y)+1$-loop if $rank(y)$ is even) through y and then a path from y to x and paths from x to y with priorities $n-1$ and n. In short, we get a $P(1,3)$ pattern.

Hence, in every SCC D of $C - \{q : \widehat{rank}(q) = n\}$ either there is no vertex with arrows into both directions staying in D or all the vertices in D have rank $(n-1)$.

The $(0,2)$ automaton recognizing paths staying in C works as follows. It uses $1,2$ for the part where the computation of $\mathcal{A}$ enters forever in a component D with no split (hence it never sees n from C again). For the rest of C it uses 1 for n and 0 for $n-1$ to follow the computation between n and components D with only $n-1$. Such a computation can traverse also finite intervals of SCCs with no split and we use 1 there too. Observe that these intervals begin and finish in a node of rank n.

For an SCC C with maximal $n+1$ we have by the above that each SCC of $C - \{q : \widehat{rank}(q) = n\}$ can be handled by a $(0,2)$ automaton. Hence, we combine these automata in the same way as in C and change ranks $(n+1)$, which is even, to 2. □

4.5 The $(0,i)$ case, $i > 2$

Lemma 4.6 *Let $i \geq 2$. If $\mathcal{A}$ does not have $P(1,i+1)$ pattern then $T(\mathcal{A})$ is $(0,i)$-feasible.*

Proof (Sketch). The case of $i = 2$ is settled in Lemma 4.5. We consider inductive step for i odd; the other case is similar. Let $\widehat{\mathcal{A}} = \mathcal{A} \uparrow^0 \uparrow^1 \ldots \uparrow^n$ where n is an even number greater than the biggest priority in $\mathcal{A}$.

Take an SCC C with maximal n. As n is even we know that in $C \setminus \{q : \widehat{rank}(q) = n\}$ there is no $P(1,i)$ pattern. Hence, by the induction hypothesis, this part is equivalent to a $(0,i-1)$ automaton. Then C is also equivalent to a $(0,i-1)$ automaton, as we can just change change n to 2.

Take an SCC C with maximal rank $n+1$. The part $C \setminus \{q : \widehat{rank}(q) = n+1\}$ is equivalent to a $(0,i-1)$ automaton by the above paragraph. Hence C is equivalent to $(0,i)$ automaton when we change rank $n+1$ to i. □

4.6 The $(1,i)$ case, $i > 2$

Lemma 4.7 *Let $i \geq 2$. If $\mathcal{A}$ does not have $P(0,i)$ pattern then $T(\mathcal{A})$ is $(1,i+1)$-feasible.*

Proof (Sketch). The case of $i = 2$ is settled in Lemma 4.4. We consider inductive step for i odd, the other case is similar. Let $\widehat{\mathcal{A}} = \mathcal{A} \uparrow^0 \uparrow^1 \ldots \uparrow^n$ where n is an odd number greater than the biggest priority in $\mathcal{A}$.

As before consider the SCCs of $\widehat{\mathcal{A}}$ one by one.

Take an SCC, call it $\mathcal{C}$, with the biggest rank n. When we remove states of rank n from $\mathcal{C}$ then in the rest we cannot have $P(0, i-1)$ pattern. The induction hypothesis implies that this part can be simulated by a $(1, i)$ automaton. Hence the whole $\mathcal{C}$ is recognizable by a $(1, i)$ automaton when we use i for vertices originally with rank n.

Now let $\mathcal{C}$ be a SCC with the greatest rank $n+1$. When we consider $\mathcal{C}$ without states of rank $n+1$, we get, by the preceding paragraph, that each of SCCs of this graph is equivalent to a $(1, i)$ automaton. Hence we can use $i+1$ in place of $n+1$ and obtain an $(1, i+1)$ automaton equivalent to $\mathcal{C}$. □

5 Decision procedure

We now estimate complexity of the procedure which, given a deterministic parity tree automaton $\mathcal{A}$, computes the level of $T(\mathcal{A})$ in the nondeterministic hierarchy.

We first need to reduce the graph of $\mathcal{A}$ to productive states only. Assuming that $\mathcal{A}$ has no more unproductive states, we compute $\widehat{\mathcal{A}} = \mathcal{A} \uparrow^0 \uparrow^1 \ldots \uparrow^n \uparrow^{n+1}$, where n is maximal rank of $\mathcal{A}$, in time polynomial on $|\mathcal{A}|$ ([12]). Searching for $P(\iota, k)$-patterns in $\widehat{\mathcal{A}}$, for $k \leq n$, can of course be carried on in polynomial time.

Hence, the most costly part of the procedure consists in computing the productive states of $\mathcal{A}$ (viewed as tree automaton), which amounts to solution of the non-emptiness problem for parity tree automaton. The fact that $\mathcal{A}$ is deterministic does not help (any automaton can be transformed into a deterministic one with the same nonemptiness status, namely an automaton reading the runs of the original automaton). This problem is equivalent to the model-checking problem for the modal μ-calculus, and to solving parity games [4]. The best deterministic algorithms known so far run in time $\mathcal{O}(n^{\frac{k}{2}})$ and space $\mathcal{O}(n)$ [6], where $n = |\mathcal{A}|$ and (ι, k) is the index of the automaton.

The best nondeterministic estimation is UP ∩ co-UP [5] (improving NP ∩ co-NP upper bound of [4]), which places our problem in $P^{UP \cap co-UP}$.

References

[1] Arnold, A., *The μ-calculus alternation-depth hierarchy is strict on binary trees*, RAIRO-Theoretical Informatics and Applications **33** (1999), 329–339.

[2] Bradfield, J.C., *The modal mu-calculus alternation hierarchy is strict*, Theoret. Comput. Sci. **195** (1997), 133–153.

[3] Emerson, E.A., and C. S. Jutla, "Tree automata, mu-calculus and determinacy", in: *Proceedings 32th Annual IEEE Symp. on Foundations of Comput. Sci.* (1991), 368–377.

[4] Emerson, E.A., C. S. Jutla, and A. P. Sistla, *On model-checking for fragments of the μ-calculus*, CAV'93, Lect. Notes Comput. Sci. **697** (1993), 385–396.

[5] Jurdziński, M., *Deciding the winner in parity games is UP ∩ co-UP*, Information Processing Letters **68** no. 3 (1998), 119–124.

[6] Jurdziński, M., *Small progress measures for solving parity games*, in: STACS 2000, Lect. Notes Comput. Sci. **1770** (2000), 290–301.

[7] Lenzi, G., *A hierarchy theorem for the mu-calculus*, in: ICALP '96, Lect. Notes Comput. Sci. **1099** (1996), 87–109.

[8] Mostowski, A.W., *Regular expressions for infinite trees and a standard form of automata*, in: Computation theory, Lect. Notes Comput. Sci. **208** (1985), 169–176.

[9] Mostowski, A.W., *Hierarchies of weak automata and weak monadic formulas*, Theoretical Comput. Sci. **83** (1991), 323–335.

[10] Niwiński, D., *On fixed point clones*, ICALP'86, Lect. Notes Comput. Sci. **226** (1986), 464–473.

[11] Niwiński, D., *Fixed points characterization of infinite behaviour of finite state systems*, Theoretical Computer Science **189** (1997), 1–69.

[12] Niwiński, D. and I. Walukiewicz, *Relating hierarchies of word and tree automata*, STACS'98, Lect. Notes Comput. Sci. **1373** (1998), 320–331.

[13] Niwiński, D. and I. Walukiewicz, *A gap property of deterministic tree languages*, Theoretical Comput. Sci. **303** (2003), 215–231.

[14] Rabin, M.O., *Decidability of second-order theories and automata on infinite trees*, Trans. Amer. Soc. **141** (1969), 1–35.

[15] Rabin, M.O., "Weakly definable relations and special automata", in: *Mathematical Logic and Foundation of Set Theory*, North–Holland, Amsterdam (1970), 1–23.

[16] Thomas, W., "Languages, automata, and logic", in: *Handbook of Formal Languages* (1997), volume 3, 389–455.

[17] Urbański, T. F., *On deciding if deterministic Rabin language is in Büchi class*, ICALP 2000, Lect. Notes Comput. Sci. **1853** (2000), 663–674.

[18] Wagner, K., *Eine topologische Charakterisierung einiger Klassen regulärer Folgenmengen*, J. Inf. Process. Cybern. EIK **13** (1977), 473–487.