Electronic Notes in Theoretical Computer Science 123 (2005) 195–208 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Deciding Nondeterministic Hierarchy of Deterministic Tree Automata

# Damian Niwin´ski[1](#_bookmark0) ,[2](#_bookmark0)

*Institute of Informatics Warsaw University Warsaw, Poland*

# Igor Walukiewicz[3](#_bookmark0)

*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 Urban´ski on deciding if a given deterministic Rabin automaton is equivalent to a nondeterministic Bu¨chi automaton. The algorithm runs in the time of verifying the non-emptiness of nondeterministic parity automata.

*Keywords:* Parity tree automata, Mostowski index, decidability.

# 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

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](mailto:myuserid@mydept.myinst.myedu)

3 Email: [igw@labri.fr](mailto:couserid@codept.coinst.coedu)

1571-0661 © 2005 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2004.05.015

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 struc- ture of automata, in particular by the acceptance condition. Today the most common variant is the *parity* condition, which reveals subtle correspondences between automata, the *µ*-calculus, and games [[3](#_bookmark11)]. Parity automata can be or- ganized into a hierarchy according to their *Mostowski indices* [4](#_bookmark1) (see Figure [1](#_bookmark4) be- low). 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](#_bookmark27)], nondeterministic [[10](#_bookmark19)], alternating [[1](#_bookmark10)] (building on [[2](#_bookmark12),[7](#_bookmark13)]), as well as the so-called weak alternating automata [[9](#_bookmark18)]. 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 deter- ministic. Deterministic tree languages form a proper, but effective, subclass of all recognizable tree languages (we can determinize an automaton in EX- PTIME [[13](#_bookmark22)], whenever possible). Computing the level in the deterministic hierarchy can be accomplished by reduction to an analogous problem for word automata [[12](#_bookmark21)], see Remark [2.4](#_bookmark5) below. Note however that the level of a deter- ministic language in a nondeterministic hierarchy can be arbitrarily smaller than in the deterministic one [5](#_bookmark2) .

Concerning nondeterministic hierarchy, Urban´ski [[17](#_bookmark26)] showed how to de- cide if a given deterministic Rabin automaton is equivalent to a Bu¨chi automa- ton (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 al- ternating hierarchy is effective for easy reasons, because they are all co-Bu¨chi, hence on the level (0*,* 1) of the alternating hierarchy.

To show our result, we refine the technique introduced in [[12](#_bookmark21)], where we solved the problem for tree languages ∀*L*, where *L* ⊆ Σ*ω* and ‘∀’ is under- stoodd in the CTL manner (that is, *t* ∈ ∀*L* if the *ω*-words read along all paths

4 Here we credit A. W. Mostowski, who first considered [[8](#_bookmark17)] tree automata with such ac- cepting condition. The Mostowski indices refine the *Rabin indices* [[14](#_bookmark23)].See [[16](#_bookmark25)] for relations between various kinds of automata.

5 It follows easily from the fact that all recognizable *word* languages can be recognized by

Bu¨chi automata, while the deterministic hierarchy is infinite [[18](#_bookmark27)].

of *t* are in *L*). There, computing the nondeterministic index of the tree lan- guage ∀*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 sim- ilarly if we take into consideration both labels and directions of paths (e.g., for binary trees, the alphabet of paths becomes Σ × {*l, r*}). It turns out that the nondeterministic index of the tree language depends again on the pres- ence of some flower-like patterns in the deterministic automaton for the path language.

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 ∩ co-UP [[5](#_bookmark14)]).

# Basic notions

Automata on infinite words.

A finite nondeterministic parity automaton on infinite words is presented by A = ⟨Σ*, Q, qI, Tr, rank*⟩, where Σ is a finite alphabet, *Q* is a finite set of *states* with an *initial state qI*, *Tr* ⊆ *Q* × Σ × *Q* is a set of *transitions*, and *rank* : *Q* → *ω* is the *ranking* function. A transition (*q, a, p*) is usually written

*q* →*a p*.

A *run* of an automaton *A* on an infinite word *u* ∈ Σ*ω* can be presented

*a*

as an infinite word *ρ* ∈ *Qω* such that *ρ*(0) = *qI*, and *ρ*(*m*) → *ρ*(*m* + 1),

whenever *u*(*m*) = *a*, for every *m < ω*. As usual, the run *ρ* is *accepting* if lim sup*n*→∞ *rank*(*ρ*(*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*}∗ → Σ, we denote the set of all such trees by *T*Σ.

A *nondeterministic parity tree automaton* A = ⟨Σ*, Q, qI, Tr , rank*⟩ is like an automaton on words except for that *Tr* ⊆ *Q* × Σ × *Q* × *Q*. A *run* of A on a tree *t* ∈ *T*Σ is itself a *Q*–valued tree *ρ* : {*l, r*}∗ → *Q* such that *ρ*(e)= *qI*, and, for each *w* ∈ dom(*ρ*), ⟨*ρ*(*w*)*, a, ρ*(*wl*)*, ρ*(*wr*)⟩ ∈ *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 *...* ∈ {*l, r*}*ω* , this means that lim sup*n*→∞ *rank*(*ρ*(*p*0*p*1 *... pn*)) is even. A *run is accepting* if so are all its

paths. The tree language *T* (A) *recognized* by 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* × Σ to *Q*, or to *Q* × *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 A*w* = ⟨Σ×{*l, r*}*, Q, qI, Tr w, rank*⟩, where

*a,l a,r*

*Tr w* = {*q* → *q*1*, q* → *q*2 : (*q, a, q*1*, q*2) ∈ *Tr* }

A *labeled path* in a tree *t* : {*l, r*}∗ → Σ is an infinite sequence (*σ*0*p*0)*,* (*σ*1*p*1)*,* (*σ*2 *p*2) *.. .*, such that *σi* ∈ Σ, *pi* ∈ {*l, r*}, and *t*(*p*0 *... pi*−1) = *σi* (so in particular *t*(*ε*) = *σ*0). It should be clear that A recognizes a tree *t* if *Aw* recognizes all labeled paths of *t*. Conversely, any *deterministic* word automaton over Σ × {*l, r*} induces a (deterministic) tree automaton over Σ in the obvious manner. In the sequel we will usually not distinguish notationally between A and 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 A is the pair (min (*rank*(*Q*)), max (*rank*(*Q*))). We let (*ι, κ*) ± (*ι*'*, κ*') if either *ι*' ≤ *ι* and *κ* ≤ *κ*' or *ι* = 0, *ι*' = 1, and *κ* +2 ≤ *κ*'. It is easy to see that, if (*ι, κ*) ± (*ι*'*, κ*') 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](#_bookmark4). (Without loss of generality we may assume that min(*rank*(*Q*)) ∈

{0*,* 1}; otherwise scale down the rank by *rank*(*q*) := *rank*(*q*) − 2.)

It is known that the hierarchy of Figure [1](#_bookmark4) is *strict* for deterministic au- tomata on words and trees [6](#_bookmark3) [[18](#_bookmark27)], and for nondeterministic automata on trees [[10](#_bookmark19)] (also for alternating automata which we do not consider here). We recall the examples from [[10](#_bookmark19),[11](#_bookmark20)], because they are related to our proof.

For *n* ∈ N, let *Mn* be the set of trees *t* over alphabet {0*,* 1*,..., n*}, such that for any path *u* ∈ {*l, r*}*ω* of *t*, lim sup*i*→∞ *t*(*ui*) is *even*. Let *Nn* be defined

6 Strictly speaking, Wagner [[18](#_bookmark27)] did not considered trees, but the result follows easily from the word case; it also follows from [[10](#_bookmark19)] because the examples there are deterministic.

¸¸¸¸¸¸

,,,,,

¸¸,¸,¸,,

,,,, ¸¸¸

(1*,* 2*k* + 2¸) (0*,* 2*k* + 1)

¸¸¸¸,¸,,,,

,,,,

,,

¸

,,

¸¸¸¸¸¸

¸¸¸¸¸¸

,,,,,

¸¸,¸,¸,,

,,,,,

(1*,* 4) ¸¸¸

¸¸¸¸

(0*,* 3)

,,

¸¸¸,¸,,,

,,,,, ¸¸¸¸¸

,

(1*,* 3) ¸¸¸

(0*,* 2)

,,

¸¸¸,¸,,,

,,,,, ¸¸¸¸¸

,

(1*,* 2) ¸¸¸

(0*,* 1)

,,

(1*,* 1)

¸¸¸,¸,,,

,,,,, ¸¸¸¸¸

,

(0*,* 0)

Fig. 1. Hierarchy of Mostowski indices.

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](#_bookmark19),[11](#_bookmark20)]) *For n* ∈ N*: (i) Mn is* 0*-n-feasible but* 1*-*(*n* + 1)*- unfeasible; (ii) Nn is* 1*-*(*n* + 1)*-feasible but* 0*-n-unfeasible.*

Flowers

For an integer *k*, a *k-loop* in a deterministic word automaton A is a path *v*1*,... , vj* = *v*1 in the automaton graph (with *j >* 1), such that max {*rank*(*vi*): *i* = 1*,... , j*} = *k*. Given integers *m* ≤ *n*,a state *q* ∈ *Q* is a *m-n-flower* in A if for every *k* = *m,... , n*, there is, in the graph of A,a *k*-loop containing *q*. We have introduced this concept in [[12](#_bookmark21)], together with a rank lifting operation on automata, ↑*i* (for *i* ∈ 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](#_bookmark21)]) *For a deterministic word automaton* A*, let* B = A ↑0↑1

*...* ↑*i. Then L*(B)= *L*(A) *and moreover if a state q has the priority m* ≤ *i in*

B *then q is a m-i-flower in* B*.*

We will use the following consequence of this lemma.

Lemma 2.3 *If n is greater than all ranks of the states of* A *then the maximal rank in any strongly connected component (SCC) of* A ↑0↑1 *...* ↑*n is n or n* + 1*.*

Proof. By the property of ↑*i*, it can be maximally *n* + 1. Now if a state *q* has *rank*(*q*)= *i* ≤ *n* in A ↑0↑1 *...* ↑*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](#_bookmark21)] we have also showed how to determine the determin- istic Mostowski index of a word automaton, by analyzing the flowers in the

↑-modified automaton. Together with the aforementioned correspondence be- tween 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](#_bookmark22)] how to transform a nondeterministic tree automaton into a deterministic one whenever it is pos- sible (within the EXPTIME bound), the case of deterministic hierarchy can be considered settled.

# Forbidden flower patterns

Now for each Mostowski index (*ι, n*), we will define a flower–like pattern *P* (*ι, n*), that is a family of subgraphs, which may occur in a deterministic word automaton over Σ × {*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 A contains a *P* (*ι, 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,...* for states. Let *a*~*b* be a short notation for a path *a* = *v*1*,... , vj* = *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* ~*k*

*b* if

moreover this is a *k-path*, i.e., max {*rank*(*vi*) : *i* = 1*,..., j*} = *k*. So a path

*a* ~*k*

*a* is a *k*-loop.

We say that two paths *a* = *v*1*,... , vj* = *b* and *a* = *w*1*,... , wl split at a* if

*σ,p*

there exist two transitions *a v*

→

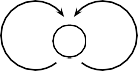
and *a σ*' *,p*' *w* , such that *σ* = *σ*', but *p* /= *p*'.

2 → 2

* 1. *The* (1*,* 2) *case*

A *P* (1*,* 2) pattern consists of a point *a* and two loops *a* 1+~2*α a* and *a* 2+~2*α a*

(they need not split):

1  2

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.

* 1. *The* (0*,* 1) *case*

A *P* (0*,* 1) pattern consists of two loops *a* 0+~2*α a* and *a* 1+~2*α a* which split at *a*:

0 1

*l*

*r*

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

* 1. *The* (0*,* 2) *case*

A *P* (0*,* 2) pattern consist of three loops *a* 0+~2*α a*, *a* 1+~2*α a*, and *a* 2+~2*α a*, where the first two split at *a* (notice that the third one need not split with any of them).

0 1

2

*l*

*r*

* 1. *The* (1*,* 2*,* 3) *case*

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



1

2

3

*l*

*r*

It can be presented by points *a, b, c, d* (where *a* and *b* need not be different), together with a loop *a* 1+~2*α a* and the paths *a*~*b*, *b* ~ *c*, *b* ~ *d*, *c* ~ *a*, and *d* ~ *a*, such that the composition *b* ~ *c* ~ *a*~*b* forms a 2 + 2*α*-loop, the composition *b* ~ *d* ~ *a*~*b* forms a 3 + 2*α*-loop, and these two loops split at *b*.

* 1. *The* (1*, n*) *case, n* ≥ 4

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

4



1

2

3

*l*

*r*

More generally, for *n* ≥ 4, a *P* (1*, n*) pattern is obtained from a *P* (1*,n* − 1) pattern (with a shifting parameter 2*α*) by adding a loop *a n*~+2*α a*.

* 1. *The* (0*, n*) *case, n* ≥ 3

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

*D. Niwin´ski, I. Walukiewicz / Electronic Notes*3*in Theoretical Computer Science 123 (2005) 195–208* 203

0 1

2

*l*

*r*

More generally, for *n* ≥ 3, a *P* (0*, n*) pattern is obtained from a *P* (0*,n* − 1) pattern (with a shifting parameter 2*α*) by adding a loop *a n*~+2*α a*.

A state *q* of automaton A is *productive* if A accepts some tree from *q*, that is *T* (A*q*) /= ∅, where A*q* is 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 (*ι, n*) denote the index dual to (*ι, n*).

We are ready to state the following.

Theorem 3.1 *If a deterministic tree automaton* A *contains a productive* (*ι, n*) *pattern (ι* ∈ {0*,* 1}*) then T* (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](#_bookmark19)] and in [[12](#_bookmark21)] which in turn followed the original idea of Rabin [[15](#_bookmark24)], 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 construc- tions.

# On the positive side

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

Theorem 4.1 *If a deterministic tree automaton* A *does not contain any pro- ductive* (*ι, n*) *pattern (ι* ∈ {0*,* 1}*) then T* (A) *is recognized by a nondetermin- istic 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 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 A has only produc- tive 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 nonde- terministic automaton of index (*ι, n*).

* 1. *The* (0*,* 1) *case*

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

Proof. It follows from the Flower Lemma of [[12](#_bookmark21)] that A is a (deterministic) (0*,* 1)-automaton, hence A itself suffices.

* 1. *The* (1*,* 2) *case*

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

Although this case was already settled in [[13](#_bookmark22)], we sketch another proof here, which will serve as the basis of inductive argument.

Proof (Sketch) Let A^ = A ↑0↑1 *...* ↑*n* where *n* is an odd number greater than the biggest rank in A.

Given a tree *t* in *T* (A), there is a unique run of A^ on *t*. This defines parts of the tree accepted by different strongly connected components (SCCs) of A^. We can have an automaton without acceptance conditions that calculates in each node the state of the unique run of 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 A^. The role of the latter automata will be to check if all paths of the run of A^ that stay forever in a given SCC are accepting. The composition of these automata will give us (1*,* 2) automaton recognizing *T* (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 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 Bu¨chi, i.e., (1*,* 2)-automaton [[16](#_bookmark25)].)

The other case is when a maximal rank in a SCC component *C* of A^ is *n* + 1. It is even as *n* is odd. Consider the SCCs of the graph *C* − {*q* : *r*ˆ*ank*(*q*) = *n* + 1}. By the above argument each such SCC can be simulated by a (1*,* 2) automaton [7](#_bookmark9) . 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.

* 1. *The* (1*,* 3) *case*

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

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

Take an SCC *C* with maximal *n*. As *n* is even we know that in *C* \ {*q* : *r*ˆ*ank*(*q*)= *n*} there is no *P* (0*,* 1) pattern. Hence, by the result of Lemma [4.3](#_bookmark6), for each SCC of *C* \ {*q* : *r*ˆ*ank*(*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* \ {*q* : *r*ˆ*ank*(*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.

* 1. *The* (0*,* 2) *case*

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

Proof. Let A^ = A ↑0↑1 *...* ↑*n* where *n* is an odd number greater than the biggest priority in A. As before it is enough to show for each SCC of 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* : *r*ˆ*ank*(*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

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

directions *l* and *r*. As *rank*(*x*) ≤ *n* − 1 we havea (*n* − 1)-loop through *x* (there must be vertex of rank (*n* − 1) in *D*) in one of these directions and an *n*-loop in *C* in the other (as all the nodes are from *C*). If in D there is a node of *y* with *r*ˆ*ank*(*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* : *r*ˆ*ank*(*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 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* : *r*ˆ*ank*(*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.

* 1. *The* (0*, i*) *case, i >* 2

Lemma 4.6 *Let i* ≥ 2*. If* A *does not have P* (1*,i* + 1) *pattern then T* (A) *is*

(0*, i*)*-feasible.*

Proof (Sketch). The case of *i* = 2 is settled in Lemma [4.5](#_bookmark8). We consider inductive step for *i* odd; the other case is similar. Let A^ = A ↑0↑1 *...* ↑*n* where *n* is an even number greater than the biggest priority in A.

Take an SCC *C* with maximal *n*. As *n* is even we know that in *C* \ {*q* : *r*ˆ*ank*(*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* \{*q* : *r*ˆ*ank*(*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*.

* 1. *The* (1*, i*) *case, i >* 2

Lemma 4.7 *Let i* ≥ 2*. If* A *does not have P* (0*, i*) *pattern then T* (A) *is*

(1*,i* + 1)*-feasible.*

Proof (Sketch). The case of *i* = 2 is settled in Lemma [4.4](#_bookmark7). We consider inductive step for *i* odd, the other case is similar. Let A^ = A ↑0↑1 *...* ↑*n* where *n* is an odd number greater than the biggest priority in A.

As before consider the SCCs of A^ one by one.

Take an SCC, call it C, with the biggest rank *n*. When we remove states of rank *n* from 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*) au- tomaton. Hence the whole C is recognizable by a (1*, i*) automaton when we use *i* for vertices originally with rank *n*.

Now let C be a SCC with the greatest rank *n* + 1. When we consider 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 C.

# Decision procedure

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

We first need to reduce the graph of A to productive states only. Assuming that A has no more unproductive states, we compute A^ = A ↑0↑1 *...* ↑*n*↑*n*+1, where *n* is maximal rank of A, in time polynomial on |A| ([[12](#_bookmark21)]). Searching for *P* (*ι, k*)-patterns in A^, for *k* ≤ *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 A (viewed as tree automaton), which amounts to solu- tion of the non-emptiness problem for parity tree automaton. The fact that 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](#_bookmark15)]. The best deterministic algorithms known so far run in time O(*n* 2 ) and space O(*n*) [[6](#_bookmark16)], where *n* = |A| and (*ι, k*) is the index of the automaton.

*k*

The best nondeterministic estimation is UP ∩ co-UP [[5](#_bookmark14)] (improving NP ∩

co-NP upper bound of [[4](#_bookmark15)]), which places our problem in *P UP* ∩*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. Jurdzin´ski, M., *Deciding the winner in parity games is UP* ∩ *co-UP*, Information Processing Letters 68 no. 3 (1998), 119–124.
6. Jurdzin´ski, 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. Niwin´ski, D., *On ﬁxed point clones*, ICALP’86, Lect. Notes Comput. Sci. 226 (1986), 464–473.
11. Niwin´ski, D., *Fixed points characterization of inﬁnite behaviour of ﬁnite state systems*, Theoretical Computer Science 189 (1997), 1–69.
12. Niwin´ski, D. and I. Walukiewicz, *Relating hierarchies of word and tree automata*, STACS’98, Lect. Notes Comput. Sci. 1373 (1998), 320–331.
13. Niwin´ski, 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. Urban´ski, T. F., *On deciding if deterministic Rabin language is in Bu¨chi class*, ICALP 2000, Lect. Notes Comput. Sci. 1853 (2000), 663–674.
18. Wagner, K., *Eine topologische Charakterisierung einiger Klassen regula¨rer Folgenmengen*,

J. Inf. Process. Cybern. EIK 13 (1977), 473–487.