International Journal of Foundations of Computer Science
Vol. 29, No. 5 (2018) 911–933
© World Scientific Publishing Company
DOI: 10.1142/S012905411842011X

On the Strength of Unambiguous Tree Automata

Henryk Michalewski
Institute of Mathematics, University of Warsaw, Banacha 2
Warsaw 02-097, Poland
h.michalewski@mimuw.edu.pl

Michał Skrzypczak*
Institute of Informatics, University of Warsaw, Banacha 2
Warsaw 02-097, Poland
mskrzypczak@mimuw.edu.pl

Received 10 February 2017
Accepted 30 March 2018
Communicated by Srečko Brlek

This work is a study of the class of non-deterministic automata on infinite trees that are *unambiguous* i.e. have at most one accepting run on every tree. The motivating question asks if the fact that an automaton is unambiguous implies some drop in the descriptive complexity of the language recognised by the automaton. As it turns out, such a drop occurs for the parity index and does not occur for the weak parity index.

More precisely, given an unambiguous parity automaton $\mathcal{A}$ of index $(i, 2j)$, we show how to construct an alternating automaton $\textsc{Transformation}(\mathcal{A})$ which accepts the same language, but is simpler in terms of the acceptance condition. In particular, if $\mathcal{A}$ is a Büchi automaton $(i = 0, j = 1)$ then $\textsc{Transformation}(\mathcal{A})$ is a weak alternating automaton. In general, $\textsc{Transformation}(\mathcal{A})$ belongs to the class $\mathrm{Comp}(i+1, 2j)$, what implies that it is simultaneously of alternating index $(i, 2j)$ and of the dual index $(i+1, 2j+1)$. The transformation algorithm is based on a separation procedure of Arnold and Santocanale (2005).

In the case of non-deterministic automata with the weak parity condition, we provide a separation procedure analogous to the one used above. However, as illustrated by examples, this separation procedure cannot be used to prove a complexity drop in the weak case, as there is no such drop.

Keywords: Infinite trees; unambiguity; Rabin–Mostowski index.

1. Introduction

Non-determinism is a very powerful concept that allows a given machine to *guess* some choices or witnesses during a computation. The cost of this ability is that

*Corresponding author.

in many cases non-deterministic machines are much less tractable than deterministic ones. Thus, people seek for ways of limiting the power of non-deterministic machines while still preserving some of their nice properties. One of the most important among such restricted forms of non-determinism is the notion of *unambiguity*: a non-deterministic machine is *unambiguous* if it admits at most one successful computation on every input.

In the case of regular languages of finite and infinite words, each automaton can be determinised (and thus made unambiguous) at an exponential cost. However, in the case of infinite trees there are automata for which one cannot find an equivalent unambiguous one [16] (see also [5]). Thus, the assumption of unambiguity limits the expressive power of automata over infinite trees.

In this work we try to understand whether this limitations in the expressive power are also translated into the descriptive complexity of the languages. To determine descriptive complexity of languages we use the standard measure i.e. the *parity index* hierarchy. A parity automaton $\mathcal{A}$ has index (i,j) if the priorities of the states of the automaton belong to the set $\{i, i+1, \ldots, j\}$. In particular, the Büchi acceptance condition can be translated to the index $(1,2)$. $\mathrm{Comp}(i,j)$ stands for the class of alternating automata where each strongly-connected component is of index (i,j) or $(i+1, j+1)$, see page 915. The hierarchy induced by parity indices is strict [1, 4] in the sense that there exist languages requiring big indices: for every pair (i,j) there exists a regular language of infinite trees that is of index (i,j) and cannot be recognised by any alternating nor non-deterministic automaton of a lower index.

The main result of this work states that the fact that a given automaton $\mathcal{A}$ is unambiguous allows to effectively find another equivalent automaton with a simpler acceptance condition. This implies that $\mathcal{L}(\mathcal{A})$ is descriptively simpler than generic languages recognised by non-deterministic automata of the same index as $\mathcal{A}$. More precisely, in Sec. 4 we propose an algorithm TRANSFORMATION, based on a separation procedure of Arnold and Santocanale [3]. The properties of the algorithm TRANSFORMATION are expressed by the following theorem.

Theorem 1. *If $\mathcal{A}$ is an unambiguous automaton of index $(i,2j)$ then* TRANSFORMATION$(\mathcal{A})$ *accepts the same language as $\mathcal{A}$ and belongs to the class* $\mathrm{Comp}(i+1,2j)$*, in particular it is simultaneously of alternating index $(i,2j)$ and of the dual index $(i+1,2j+1)$. In the specific case of an unambiguous Büchi automaton $\mathcal{A}$,* TRANSFORMATION$(\mathcal{A})$ *is weak.*

Additionally, the number of states of TRANSFORMATION$(\mathcal{A})$ *is polynomial in the number of states of $\mathcal{A}$.*

The above theorem is the main result presented in the conference paper [13].

To make the picture more complete, the present work additionally provides a study of the descriptive complexity drop for unambiguous automata with the weak parity acceptance condition (i.e. parity automata where each transition does not decrease the priorities of the states). Contrary to the case of the general parity acceptance condition; in the weak case there is no drop in the descriptive complexity of the recognised language, as expressed by the following theorem.

Theorem 2. *For every $n \geq 0$ there exist weak unambiguous automata $\mathcal{A}_n$ and $\mathcal{B}_n$ such that:*

- *$\mathcal{A}_n$ has index $(0, n+1)$ and $\mathcal{B}_n$ has index $(1, n+2)$,*
- *$\mathcal{L}(\mathcal{A}_n) = \mathcal{L}(\mathcal{B}_n)^{\mathrm{c}}$, i.e. they recognise complementary languages,*
- *$\mathcal{L}(\mathcal{A}_n)$ cannot be recognised by any weak alternating automaton of index $(1, n+2)$, dually for $\mathcal{B}_n$.*

This is a bit surprising since the separation result still holds in the weak case, as shown in Theorem 2.

1.1. *Related work*

This paper is an extended journal version of [13]. The new results, that were not presented before, are: a simple proof of Theorem 1 for the case of Büchi automata that is based on the results of [19] and [9]; a procedure for separation of weak non-deterministic automata; and a proof that weak unambiguous automata admit no complexity collapse. The initial part of this paper (i.e. Secs. 2, 3, 4, and 5) is based on the content of [13], in particular Figs. 1 and 2 come from that paper.

Although unambiguous automata on infinite trees are still not well-understood, there is already a number of results estimating their expressive power.

The first and most fundamental of these results is the observation of Niwiński and Walukiewicz [16] (see also [5]) that not all regular tree languages can be recognised by unambiguous automata. The example of a regular language that cannot be recognised by any unambiguous automaton provided in [16] is the language of trees labelled $\{a, b\}$ such that the letter a appears at least once in the tree. The extreme simplicity of that language suggests that unambiguous automata over infinite trees are not very expressive.

As it turned out, this impression was not entirely correct. Firstly, Hummel in [10] proved that unambiguous languages are topologically harder than deterministic ones. Further improvements from [7] provided examples of unambiguous tree languages reaching high into the second level of the index hierarchy. This line of research culminates in a recent unpublished work [18] which shows that there are unambiguous languages lying arbitrarily high in the index hierarchy. The construction given in [18] provides an unambiguous automaton of index $(0, 2j{+}2)$ to recognise a language hard for the level $(0, 2j)$ of the index hierarchy. Thus, the drop in the descriptive complexity from Theorem 1 does not affect these examples.

The above mentioned constructions show that unambiguous automata can recognise very complex languages. However, they do not explain whether the condition of unambiguity comes at some cost (e.g. increase in the index of the considered automaton comparing to the language recognised by it). The first result of that form is a result of Finkel and Simonnet [9] who provided a simple proof based on the Lusin-Souslin Theorem [11, Theorem 15.1], that any language recognised by an unambiguous Büchi automaton must be Borel. Thus, although both Büchi automata and unambiguous automata can recognise non-Borel sets; combining these two assumptions implies that the language is topologically simple.

The construction of this paper can be seen as an extension of the result of Finkel and Simmonet by providing not only a set-theoretical argument but also an automata construction encapsulated by the algorithm Transformation. Another advantage of this approach is the fact that the algorithm works for arbitrarily high levels of the index hierarchy, and not only for the Büchi case.

The combination of all the above mentioned results provides a quite complete picture, with unambiguous automata being able to recognise arbitrarily complex languages; and this complexity coming at the cost of having higher index than an alternating automaton for the same language.

1.2. *Outline of the paper*

Section 2 provides basic notions used throughout the paper. In Sec. 3 we show that in a certain sense transitions of an unambiguous automaton must induce disjoint languages. Based on that observation and the separation procedure of Arnold and Santocanale, an algorithm Partition is designed. This algorithm is used in Sec. 4 to construct the automaton Transformation($\mathcal{A}$). Section 5 concludes the proof of Theorem 1 by showing correctness of Transformation.

Section 6 derives Theorem 1 in the case of Büchi automata from the results of [19] and [9]. This can be seen as an alternative proof of this specific case, with worse complexity of the resulting automaton.

In Sec. 7 we prove a modification of the separation result of Arnold and Santocanale for the case of weak non-deterministic automata. However, it turns out that this separation result cannot be used to build a variant of Transformation algorithm to prove a complexity collapse for weak unambiguous automata. The reason is that weak unambiguous automata admit no such collapse — they can recognise languages that are as hard as possible in the respective weak alternating index classes, see Theorem 2 proved in Subsec. 7.2.

2. Basic Notions

In this section we introduce basic notions used in the rest of the paper. A good survey of the relations between deterministic, unambiguous, and non-deterministic automata is [6]. A general background on automata and logic over infinite trees can be found in [21].

Our models are infinite, labelled, full binary trees. The labels come from a non-empty finite set A called *alphabet*. A tree t is a function $t\colon \{\mathrm{L},\mathrm{R}\}^* \to A$. The set of all such trees is Tr_A. Vertices of a tree are denoted $u, v, w \in \{\mathrm{L},\mathrm{R}\}^*$. The prefix-order on vertices is $\preceq$, the minimal element of this order is the *root* $\epsilon \in \{\mathrm{L},\mathrm{R}\}^*$.

The label of a tree $t \in Tr_A$ in a vertex $u \in \{\mathtt{L},\mathtt{R}\}^*$ is $t(u) \in A$. $t{\upharpoonright}_u$. stands for the subtree of a tree t rooted in a vertex u. Infinite branches of a tree are denoted as $\alpha, \beta \in \{\mathtt{L},\mathtt{R}\}^\omega$. We extend the prefix order to them, thus $u \prec \alpha$ if u is a prefix of α. For an infinite branch $\alpha \in \{\mathtt{L},\mathtt{R}\}^\omega$ and $k \in \omega$ by $\alpha{\upharpoonright}_k$ we denote the prefix of α of length k (e.g. $\alpha{\upharpoonright}_0 = \epsilon$).

A *non-deterministic tree automaton* $\mathcal{A}$ is a tuple $\langle Q, A, q_0, \Delta, \Omega\rangle$ where: Q is a finite set of *states*; A is an alphabet; $q_\mathrm{I} \in Q$ is an *initial state*; $\Delta \subseteq Q \times A \times Q \times Q$ is a *transition relation*; $\Omega\colon Q \to \mathbb{N}$ is a *priority function*.

If the automaton $\mathcal{A}$ is not known from the context we explicitly put it in the superscript, i.e. $Q^{\mathcal{A}}$ is the set of states of $\mathcal{A}$.

A *run* of an automaton $\mathcal{A}$ on a tree t is a tree $\rho \in Tr_Q$ such that for every vertex u we have $\big(\rho(u), t(u), \rho(u\mathtt{L}), \rho(u\mathtt{R})\big) \in \Delta$. A run ρ is *parity-accepting* if on every branch α of the tree we have

$$\limsup_{n\to\infty}\ \Omega\big(\rho(\alpha{\upharpoonright}_n)\big) \equiv 0 \mod 2. \qquad (\triangle)$$

We say that a run ρ *starts* from the state $\rho(\epsilon)$. A run ρ is *accepting* if it is parity-accepting and starts from q_I. The *language recognised* by $\mathcal{A}$ (denoted $\mathcal{L}(\mathcal{A})$) is the set of all trees t such that there is an accepting run ρ of $\mathcal{A}$ on t.

A non-deterministic automaton $\mathcal{A}$ is *unambiguous* if for every tree t there is at most one accepting run of $\mathcal{A}$ on t.

An *alternating tree automaton* $\mathcal{C}$ is a tuple $\langle Q, A, Q_\exists, Q_\forall, q_0, \Delta, \Omega\rangle$ where: Q is a finite set of *states*; A is an alphabet; $Q_\exists \sqcup Q_\forall$ is a partition of Q into sets of positions of the players $\exists$ and $\forall$; $q_\mathrm{I} \in Q$ is an *initial state*; $\Delta \subseteq Q \times A \times \{\epsilon,\mathtt{L},\mathtt{R}\} \times Q$ is a *transition relation*; $\Omega\colon Q \to \mathbb{N}$ is a *priority function*. For technical reasons we assume that for every $q \in Q$ and $a \in A$ there is at least one transition $(q, a, d, q') \in \Delta$ for some $q' \in Q$ and $d \in \{\epsilon,\mathtt{L},\mathtt{R}\}$.

An alternating tree automaton $\mathcal{C}$ induces, for every tree $t \in Tr_A$, a parity game $\mathcal{G}(\mathcal{C}, t)$. The positions of this game are of the form $(u, q) \in \{\mathtt{L},\mathtt{R}\}^* \times Q$. The initial position is (ϵ, q_I). A position (u, q) belongs to the player $\exists$ if $q \in Q_\exists$, otherwise (u, q) belongs to $\forall$. The priority of a position (u, q) is $\Omega(q)$. There is an edge between (u, q) and (ud, q') whenever $(q, t(u), d, q') \in \delta$. An infinite play π in $\mathcal{G}(\mathcal{C}, t)$ is winning for $\exists$ if the highest priority occurring infinitely often on π is even, as in condition $(\triangle)$.

We say that an alternating tree automaton $\mathcal{C}$ *accepts* a tree t if the player $\exists$ has a winning strategy in $\mathcal{G}(\mathcal{C}, t)$. The language of trees accepted by $\mathcal{C}$ is denoted by $\mathcal{L}(\mathcal{C})$.

A non-deterministic or alternating automaton $\mathcal{A}$ *has index* (i, j) if the priorities of $\mathcal{A}$ are among $\{i, i+1, \ldots, j\}$. Such an automaton is *weak* if the priorities of states are non-decreasing along transitions, see [12]. An automaton of index $(1, 2)$ is called a *Büchi automaton*. Every alternating tree automaton can be naturally seen as a graph — the set of nodes is Q and there is an edge (q, q') if $(q, a, d, q') \in \Delta$ for some $a \in A$ and $d \in \{\epsilon,\mathtt{L},\mathtt{R}\}$. We say that an alternating tree automaton $\mathcal{D}$ is a $\mathrm{Comp}(i, j)$ *automaton* if every strongly-connected component of the graph of $\mathcal{D}$ is of index (i, j) or $(i+1, j+1)$, see [3].

Note that an alternating automaton $\mathcal{C}$ is $\mathrm{Comp}(0,0)$ if and only if $\mathcal{C}$ is a weak. The following fact gives a connection between these automata and weak MSO (the variant of monadic second-order logic where set quantifiers are restricted to finite sets).

Theorem 3 (Rabin [17], also Kupferman Vardi [12]). *If $\mathcal{C}$ is an alternating* $\mathrm{Comp}(0,0)$ *automaton then $\mathcal{L}(\mathcal{C})$ is definable in weak* MSO. *Similarly, if $L \subseteq Tr_A$ is definable in weak* MSO *then there exists a* $\mathrm{Comp}(0,0)$ *automaton recognising L.*

The crucial technical tool in our proof is the SEPARATION algorithm by Arnold and Santocanale [3]. A particular case of this algorithm for $i = j = 1$ is the classical Rabin separation construction (see [17]): if L_1, L_2 are two disjoint languages recognisable by Büchi alternating tree automata then one can effectively construct a weak MSO-definable language L_S that separates them.

Algorithm 1: SEPARATION

```
Input: Two non-deterministic automata $\mathcal{A}$, $\mathcal{B}$ of index $(i, 2j)$ such that
       $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}) = \emptyset$.
Output: An alternating $\mathrm{Comp}(i{+}1, 2j)$ automaton $\mathcal{S}$ such that
       $$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{S}) \quad \text{and} \quad \mathcal{L}(\mathcal{B}) \cap \mathcal{L}(\mathcal{S}) = \emptyset.$$
```

3. Partition Property

In this section we will prove Lemma 4 stating that if an automaton $\mathcal{A}$ is unambiguous then the transitions of $\mathcal{A}$ need to induce disjoint languages. This will be important in the algorithm PARTITION which for a given unambiguous automaton of index $(i, 2j)$, constructs a family of $\mathrm{Comp}(i+1, 2j)$ automata that split the set of all trees into disjoint sets corresponding to the respective transitions of $\mathcal{A}$. PARTITION will be used in TRANSFORMATION.

Let us fix an unambiguous automaton $\mathcal{A}$ of index $(i, 2j)$. Let Q be the set of states of $\mathcal{A}$ and A be its working alphabet. We say that a transition $\delta = (q, a, q_{\mathrm{L}}, q_{\mathrm{R}})$ of $\mathcal{A}$ *starts* from (q, a); let $\Delta_{q,a}$ be the set of such transitions.

A pair $(q, a) \in Q \times A$ is *productive* if it appears in some accepting run: there exists a tree $t \in Tr_A$ and an accepting run ρ of $\mathcal{A}$ on t such that for some vertex u we have $\rho(u) = q$ and $t(u) = a$. This definition combines two requirements: that there exists an accepting run that leads to the pair (q, a) and that some tree can be parity-accepted starting from (q, a). Note that if (q, a) is productive then there exists at least one transition starting from (q, a). Without changing the language $\mathcal{L}(\mathcal{A})$ we can assume that if a pair is not productive then there is no transition starting from this pair.

For every transition $\delta = (q, a, q_{\mathrm{L}}, q_{\mathrm{R}})$ of $\mathcal{A}$ we define L_δ as the language of trees such that there exists a run ρ of $\mathcal{A}$ on t that is parity-accepting and *uses δ in the* *root of* t $\rho(\epsilon) = q$, $t(\epsilon) = a$, $\rho(\mathtt{L}) = q_\mathtt{L}$, and $\rho(\mathtt{R}) = q_\mathtt{R}$. Clearly the language L_δ can be recognised by an unambiguous automaton of index $(i, 2j)$. If (q, a) is not productive then $L_{(q,a,q_\mathtt{L},q_\mathtt{R})} = \emptyset$. The following lemma is a simple consequence of unambiguity of the given automaton $\mathcal{A}$.

Lemma 4. *If $\delta_1 \neq \delta_2$ are two transitions starting from the same pair (q, a) then the languages L_{δ_1}, L_{δ_2} are disjoint.*

Proof. First, if (q, a) is not productive then by our assumption $L_{\delta_1} = L_{\delta_2} = \emptyset$. Assume contrary that (q, a) is productive and there exists a tree $r \in L_{\delta_1} \cap L_{\delta_2}$ with two respective parity-accepting runs ρ_1, ρ_2. Since (q, a) is productive so there exists a tree t and an accepting run ρ on t such that $\rho(u) = q$ and $t(u) = a$ for some vertex u. Consider the tree $t' = t[u \leftarrow r]$ — the tree obtained from t by substituting r as the subtree under u. Since $\rho(u) = q$ and both ρ_1, ρ_2 start from (q, a), we can construct two accepting runs $\rho[u \leftarrow \rho_1]$ and $\rho[u \leftarrow \rho_2]$ on t'. Since these runs differ on the transition used in u, we obtain a contradiction to the fact that $\mathcal{A}$ is unambiguous. □

The above lemma will be important in the algorithm Partition, because it uses the Seperation algorithm which in turn requires disjointness of the languages.

Algorithm 2: Partition

```
    Input: An unambiguous automaton $\mathcal{A}$ of index $(i, 2j)$
    Output: for every $\delta \in \Delta$ an automaton $\mathcal{C}_\delta$
 1  foreach $(q, a) \in Q \times A$, productive do
 2      foreach $\delta \in \Delta_{q,a}$ do
 3          $\mathcal{E}_\delta \leftarrow$ non-det. $(i, 2j)$ automaton recognising $L_\delta$
 4          $\mathcal{F}_\delta \leftarrow$ non-det. $(i, 2j)$ automaton recognising $\bigcup_{\eta \in \Delta_{q,a}, \eta \neq \delta} L_\eta$
 5      foreach $\delta \in \Delta_{q,a}$ do
 6          $\mathcal{D}_\delta \leftarrow$ Separation$(E_\delta, F_\delta)$
 7      foreach $\delta \in \Delta_{q,a}$ do
 8          $\mathcal{C}_\delta \leftarrow$ Comp$(i+1, 2j)$ automaton recognising $\mathcal{L}(\mathcal{D}_\delta) \setminus \bigcup_{\eta \neq \delta} \mathcal{L}(\mathcal{D}_\eta)$.
 9  foreach $\delta = (q, a, q_\mathtt{L}, q_\mathtt{R}) \in \Delta_{q,a}$ with $(q, a)$ non-productive do
10      $\mathcal{C}_\delta \leftarrow$ Comp$(0, 0)$ automaton recognising the empty language.
```

The following lemma summarizes properties of the algorithm Partition.

Lemma 5. *Assume that $\mathcal{A}$ is an unambiguous automaton of index $(i, 2j)$ and let $(q, a) \in Q \times A$. Take the automata $(\mathcal{C}_\delta)_{\delta \in \Delta_{q,a}}$ constructed by* Parition$(\mathcal{A})$. *Then the languages $\mathcal{L}(\mathcal{C}_\delta)$ for $\delta \in \Delta_{q,a}$ are pairwise disjoint and $L_\delta \subseteq \mathcal{L}(\mathcal{C}_\delta)$.*

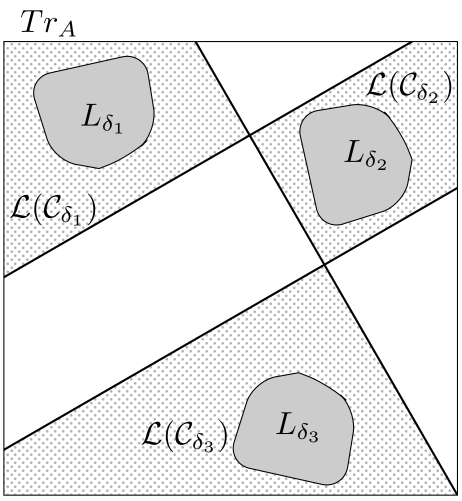

Tr_A
L_{δ_1}
$\mathcal{L}(\mathcal{C}_{\delta_2})$
L_{δ_2}
$\mathcal{L}(\mathcal{C}_{\delta_1})$
$\mathcal{L}(\mathcal{C}_{\delta_3})$
L_{δ_3}

Fig. 1. An illustration of the output of the algorithm Partition. The three circles are the languages L_{δ_i} for the transitions starting in a fixed pair (q, a). Each straight line represents the language $\mathcal{L}(\mathcal{D}_{\delta_i})$ that separates the respective language L_{δ_i} from the others. Our construction provides the automata $\mathcal{C}_{\delta_i}$ recognising the dotted regions.

A proof of this lemma follows directly from the definition of the respective automata, see Fig. 1 for an illustration of this construction.

4. Construction of the Automaton

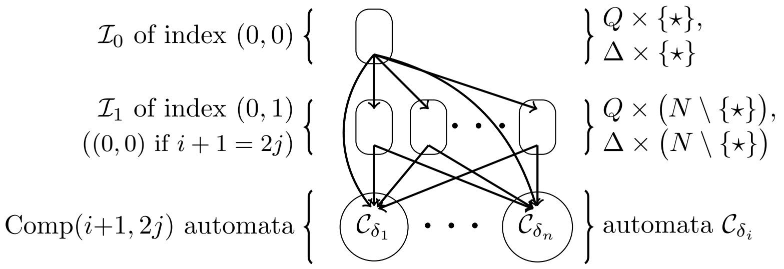
In this and the following section we will describe the algorithm Transformation and prove Theorem 1 which states correctness and properties of this algorithm. Given an automaton $\mathcal{A}$ of index $(i, 2j)$, the algorithm Transformation constructs an alternating $\mathrm{Comp}(i+1, 2j)$ automaton $\mathcal{R}$ recognising $\mathcal{L}(\mathcal{A})$. It will consist of two sub-automata running in parallel:

(1) In the first sub-automaton the role of $\exists$ will be to propose a partial run $\rho\colon \{\mathtt{L}, \mathtt{R}\}^* \rightharpoonup Q$ on a given tree t. She will be forced to propose certain unique run ρ_t that depends only on the tree t, see Definition 7. At any moment $\forall$ can *challenge* the currently proposed transition and check if it agrees with the definition of ρ_t (namely Condition ($\diamond$)).
(2) In the second sub-automaton the role of $\forall$ will be to prove that the partial run ρ_t is not parity-accepting. That is, he will find a leaf in ρ_t or an infinite branch of ρ_t that does not satisfy the parity condition. Since the run ρ_t is unique, $\forall$ can declare in advance what will be the odd priority n that is the limes superior (i.e. $\limsup$) of priorities of ρ_t on the selected branch.

The automaton $\mathcal{R}$ consists of an *initial component* $\mathcal{I}$ and of the union of the automata $\mathcal{C}_\delta$ constructed by the procedure Partition.

Algorithm 3: Transformation

```
Input: An unambiguous automaton $\mathcal{A}$ of index $(i, 2j)$
Output: An automaton $\mathcal{R}$
$N \leftarrow \{\star\} \cup \big\{n \in \{i, \ldots, 2j\} \mid n \text{ is odd}\big\}$
$Q_{\mathcal{I},\exists} \leftarrow Q^{\mathcal{A}} \times N \sqcup \{\bot, \top\}$
$Q_{\mathcal{I},\forall} \leftarrow \Delta^{\mathcal{A}} \times N$
$\Delta_{\mathcal{I}} \leftarrow \big\{(\bot, a, \epsilon, \bot), (\top, a, \epsilon, \top) \mid a \in A^{\mathcal{A}}\big\}$
$q_{\mathrm{I}}^{\mathcal{R}} \leftarrow (q_{\mathrm{I}}^{\mathcal{A}}, \star)$
$\big(\mathcal{C}_\delta\big)_{\delta \in \Delta} \leftarrow \textsc{Partition}(\mathcal{A})$
foreach $a \in A$, $q \in Q^{\mathcal{A}}$, $n \in N$ do
    if $n \neq \star$ and $\Omega^{\mathcal{A}}(q) > n$ then
        $\Delta_{\mathcal{I}} \leftarrow \Delta_{\mathcal{I}} \cup \{((q, n), a, \epsilon, \top)\}$
    else
        $\Delta_{\mathcal{I}} \leftarrow \Delta_{\mathcal{I}} \cup \Big\{\big((q, n), a, \epsilon, (\delta, n)\big) \mid \delta \in \Delta_{q,a}^{\mathcal{A}}\Big\}$
foreach $a \in A$, $\delta = (q, a, q_{\mathtt{L}}, q_{\mathtt{R}}) \in \Delta^{\mathcal{A}}$, $n \in N$ do
    $\Delta_{\mathcal{I}} \leftarrow \Delta_{\mathcal{I}} \cup \big\{(\delta, a, \epsilon, q_{\mathrm{I}}^{\mathcal{C}_\delta})\big\}$   /* such a transition is a *challenge* */
    if $n \neq \star$ then
        $\Delta_{\mathcal{I}} \leftarrow \Delta_{\mathcal{I}} \cup \big\{(\delta, a, d, (q_d, n)) \mid d \in \{\mathtt{L}, \mathtt{R}\}\big\}$
    else
        $\Delta_{\mathcal{I}} \leftarrow \Delta_{\mathcal{I}} \cup \big\{(\delta, a, d, (q_d, n')) \mid d \in \{\mathtt{L}, \mathtt{R}\}, n' \in N\big\}$
$Q_{\exists}^{\mathcal{R}} \leftarrow Q_{\mathcal{I},\exists} \sqcup \bigsqcup_{\delta \in \Delta^{\mathcal{A}}} Q_{\exists}^{\mathcal{C}_\delta}$
$Q_{\forall}^{\mathcal{R}} \leftarrow Q_{\mathcal{I},\forall} \sqcup \bigsqcup_{\delta \in \Delta^{\mathcal{A}}} Q_{\forall}^{\mathcal{C}_\delta}$
$\Delta^{\mathcal{R}} \leftarrow \Delta_{\mathcal{I}} \sqcup \bigsqcup_{\delta \in \Delta^{\mathcal{A}}} \Delta^{\mathcal{C}_\delta}$
foreach $q \in Q^{\mathcal{A}}$ do
    $\Omega^{\mathcal{R}}(q, \star) = 0$
    foreach $n \in N \setminus \{\star\}$ do
        if $\Omega^{\mathcal{A}}(q) \geq n$ then
            $\Omega^{\mathcal{R}}(q, n) = 1$
        else
            $\Omega^{\mathcal{R}}(q, n) = 0$
foreach $\delta = (q, a, q_{\mathtt{L}}, q_{\mathtt{R}}) \in Q^{\mathcal{A}}$ do
    $\Omega^{\mathcal{R}}(\delta, \star) = 0$
    foreach $n \in N \setminus \{\star\}$ do
        if $\Omega^{\mathcal{A}}(q) \geq n$ then
            $\Omega^{\mathcal{R}}(\delta, n) = 1$
        else
            $\Omega^{\mathcal{R}}(\delta, n) = 0$
```



$\mathcal{I}_0$ of index $(0,0)$
$\mathcal{I}_1$ of index $(0,1)$
$((0,0)$ if $i+1=2j)$
Comp$(i{+}1,2j)$ automata
$\mathcal{C}_{\delta_1}$
$\mathcal{C}_{\delta_n}$
$Q\times\{\star\}$,
$\Delta\times\{\star\}$
$Q\times(N\setminus\{\star\})$,
$\Delta\times(N\setminus\{\star\})$
automata $\mathcal{C}_{\delta_i}$

Fig. 2. The structure of the automaton $\mathcal{R}$.

The idea of the automaton $\mathcal{R}$ is to simulate the following behaviour. Assume that the label of the current vertex is a and the current state is $(q,n)\in Q_{\mathcal{I},\exists}$:

- if $n\neq\star$ and $\Omega^{\mathcal{A}}(q)>n$ then $\forall$ loses, see line 9;
- $\exists$ declares a transition $\delta=(q,a,q_{\mathtt{L}},q_{\mathtt{R}})$ of $\mathcal{A}$, see line 11;
- $\forall$ can decide to *challenge* this transition, see line 13;
- if $n\neq\star$ then $\forall$ chooses a direction and the game proceeds, see line 15;
- if $n=\star$ then $\forall$ chooses a direction and a new value $n'\in N$, see line 17.

Figure 2 depicts the structure of the automaton $\mathcal{R}$. The initial component $\mathcal{I}$ is split into two parts: $\mathcal{I}_0$ where $n=\star$ and $\mathcal{I}_1$ where $n\neq\star$.

We will now proceed with proving properties of the procedure Transformation.

Lemma 6. *If $\mathcal{A}$ is unambiguous and of index $(i,2j)$ then $\mathcal{R}$ is in* Comp$(i{+}1,2j)$.

Proof. We first argue that if $i{+}1<2j$ then $\mathcal{R}$ is a Comp$(i{+}1,2j)$ automaton. Note every strongly-connected component in the graph of $\mathcal{R}$ is either a component of $\mathcal{I}_0$, $\mathcal{I}_1$, or of $\mathcal{C}_\delta$ for $\delta\in\Delta^{\mathcal{A}}$. Recall that all the components $\mathcal{A}_\delta$ are by the construction Comp$(i+1,2j)$-automata. By the definition, $\mathcal{I}_0$ and $\mathcal{I}_1$ are Comp$(1,2)$-automata, so the whole automaton $\mathcal{R}$ is also Comp$(i+1,2n)$.

Consider the opposite case: $i+1=2j$. By shifting all the priorities we can assume that $i=j=1$ (i.e. $\mathcal{A}$ is Büchi). Observe that the only possible odd value n between i and $2j$ is $n=1$. It means that if $\forall$ declares a value $n\neq\star$ then always $\Omega(q)\geq n$ holds. It means that there are no states in $\mathcal{I}_1$ with priority 1. Therefore, both $\mathcal{I}_0$ and $\mathcal{I}_1$ are Comp$(0,0)$ automata and $\mathcal{R}$ is a Comp$(0,0)$ automaton. □

5. Correctness of the Construction

In this section we prove that the automaton $\mathcal{R}$ constructed by the algorithm Transformation recognises the same language as the given unambiguous automaton $\mathcal{A}$.

This way we conclude the proof of Theorem 1. Let $\mathcal{A}$ be an unambiguous automaton of index $(i, 2j)$.

Definition 7. Let $t \in Tr_A$ be a tree. We define ρ_t as the unique maximal partial run ρ_t of $\mathcal{A}$ on t, i.e. a partial function $\rho_t \colon \{\mathtt{L}, \mathtt{R}\}^* \rightharpoonup Q^{\mathcal{A}}$ such that:

- $\rho_t(\epsilon) = q_{\mathrm{I}}^{\mathcal{A}}$;
- if $u \in dom(\rho_t)$ and $t\restriction_u \in \mathcal{L}(\mathcal{C}_\delta)$ for some $\delta \in \Delta^{\mathcal{A}}$ then[a]
$$\delta = \big(\rho_t(u), t(u), \rho_t(u\mathtt{L}), \rho_t(u\mathtt{R})\big); \qquad (\diamond)$$
- if $u \in dom(\rho_t)$ and $t\restriction_u \notin \mathcal{L}(\mathcal{C}_\delta)$ for any $\delta \in \Delta^{\mathcal{A}}$ then $u\mathtt{L}, u\mathtt{R} \notin dom(\rho_t)$.

Lemma 8. $t \in \mathcal{L}(\mathcal{A})$ *if and only if* ρ_t *is total and accepting.*

Proof. If ρ_t is accepting then it is a witness that $t \in \mathcal{L}(\mathcal{A})$. Let ρ be an accepting run of $\mathcal{A}$ on t. We inductively prove that $\rho = \rho_t$. Take a node u of t and define $q = \rho(u)$, $a = t(u)$, $q_{\mathtt{L}} = \rho_t(u\mathtt{L})$, and $q_{\mathtt{R}} = \rho_t(u\mathtt{R})$. Observe that ρ is a witness that (q, a) is productive and for $\delta = (q, a, q_{\mathtt{L}}, q_{\mathtt{R}})$ we have
$$t \in L_\delta \subseteq \mathcal{L}(\mathcal{C}_\delta).$$
Therefore, $\rho_t(u\mathtt{L}) = \rho(u\mathtt{L})$ and $\rho_t(u\mathtt{R}) = \rho(u\mathtt{R})$. □

5.1. $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{R})$

Lemma 9. *If* $t \in \mathcal{L}(\mathcal{A})$ *then* $t \in \mathcal{L}(\mathcal{R})$.

Proof. Assume that $t \in \mathcal{L}(\mathcal{A})$. By Lemma 8 we know that ρ_t is the unique accepting run of $\mathcal{A}$ on t. Consider the following strategy $\sigma_\exists$ for $\exists$ in the initial component $\mathcal{I}$ of the automaton $\mathcal{R}$: always declare δ consistent with ρ_t. Extend it to the winning strategies in $\mathcal{C}_\delta$ whenever they exist. That is, if the current vertex is u and the state of $\mathcal{R}$ is of the form $(q, n) \in \mathcal{I}$ then move to the state (δ, n) for $\delta = (\rho_t(u), t(u), \rho_t(u\mathtt{L}), \rho_t(u\mathtt{R}))$. Whenever the game moves from the initial component $\mathcal{I}$ into one of the automata $\mathcal{C}_\delta$ in a vertex u, fix some winning strategy in $\mathcal{G}(\mathcal{C}_\delta, t\restriction_u)$ (if exists) and play according to this strategy; if there is no such strategy, play using any strategy. Take a play consistent with $\sigma_\exists$ in $\mathcal{G}(\mathcal{R}, t)$. There are the following cases:

- $\forall$ loses in a finite time according to the transition from line 9 in the algorithm Transformation.
- $\forall$ stays forever in the initial component $\mathcal{I}$ never changing the value of $n = \star$ and loses by the parity criterion.
- In some vertex u of the tree $\forall$ *challenges* the transition δ given by $\exists$ and the game proceeds to the state $q_{\mathrm{I}}^{\mathcal{C}_\delta}$. In that case $t\restriction_u \in L_\delta$ by the definition of L_δ (the

[a]By Lemma 5 there is at most one such δ.

run $\rho_t\restriction_u$ is a witness) and therefore $t\restriction_u \in \mathcal{L}(\mathcal{C}_\delta)$. So $\exists$ has a winning strategy in $\mathcal{G}(\mathcal{C}_\delta, t\restriction_u)$ and $\exists$ wins the rest of the game.

- $\forall$ declares a value $n \neq \star$ at some point and then never *challenges* $\exists$. In that case the game follows an infinite branch α of t. Since ρ_t is accepting so we know that $k \overset{\text{def}}{=} \limsup_{i\to\infty} \Omega^{\mathcal{A}}(\rho_t(\alpha\restriction_i))$ is even. If $k > n$ then $\forall$ loses at some point according to the transition from line 9. Otherwise $k < n$ and from some point on all the states of $\mathcal{R}$ visited during the game have priority 0, thus $\forall$ loses by the parity criterion in $\mathcal{I}_1$. □

Lemma 10. *If $t \notin \mathcal{L}(\mathcal{A})$ then $t \notin \mathcal{L}(\mathcal{R})$.*

Proof. We assume that $t \notin \mathcal{L}(\mathcal{A})$ and define a winning strategy for $\forall$ in the game $\mathcal{G}(\mathcal{R}, t)$. Let us fix the run ρ_t as in Definition 7.

Note that either ρ_t is a partial run: there is a vertex u such that $\rho_t(u) = q$ and $(q, t(u))$ is not productive; or ρ_t is a total run. Since $t \notin \mathcal{L}(\mathcal{A})$, ρ_t cannot be a total accepting run. Let α be a finite or infinite branch: either $\alpha \in \{\text{L}, \text{R}\}^*$ and α is a leaf of ρ_t or α is an infinite branch such that $k \overset{\text{def}}{=} \limsup_{i\to\infty} \Omega^{\mathcal{A}}(\rho_t(\alpha\restriction_i))$ is odd. If α is finite let us put any odd value between i and $2j$ as k. Consider the following strategy for $\forall$:

- $\forall$ keeps $n = \star$ until there are no more states of priority greater than k along α in ρ_t. Then he declares $n' = k$.
- $\forall$ *challenges* a transition δ given by $\exists$ in a vertex u if and only if $t\restriction_u \notin \mathcal{C}_\delta$.
- $\forall$ always follows α: in a vertex $u \in \{\text{L}, \text{R}\}^*$ he chooses the direction d in such a way that $ud \preceq \alpha$.

As in the proof of Lemma 9, we extend this strategy to strategies in the components $\mathcal{C}_\delta$ whenever such strategies exist: if the game moves from the component $\mathcal{I}$ into one of the component $\mathcal{C}_\delta$ in a vertex u then $\forall$ uses some winning strategy in the game $\mathcal{G}(\mathcal{C}_\delta, t\restriction_u)$ (if it exists); if there is no such strategy, $\forall$ plays using any strategy.

Consider any play π consistent with $\sigma_\forall$. Note that if α is a finite word and the play π reaches the vertex α in a state (δ, n) in $\mathcal{I}$ then by the definition of ρ_t we know that $t\restriction_u \notin \mathcal{C}_\delta$ and thus $\forall$ *challenges* this transition and wins in the game $\mathcal{G}(\mathcal{C}_\delta, t\restriction_u)$. By the definition of the strategy $\sigma_\forall$, $\forall$ never loses according to the transition from line 9 in the algorithm Transformation — if $\forall$ declared $n \neq \star$ then the play will never reach a state of priority greater than n.

Let us consider the remaining cases. First assume that at some vertex u player $\forall$ *challenged* a transition δ declared by $\exists$. It means that $t\restriction_u \notin \mathcal{L}(\mathcal{C}_\delta)$ and $\forall$ has a winning strategy in $\mathcal{G}(\mathcal{C}_\delta, t\restriction_u)$ and wins in that case.

The last case is that $\forall$ did not *challenge* any transition declared by $\exists$ and the play followed the branch α. Then, for every $i \in \mathbb{N}$ the game reached the vertex $\alpha\restriction_i$ in a state (q, n) satisfying $q = \rho_t(\alpha\restriction_i)$. In that case there is some vertex u along α

where $\forall$ declared $n = k$. Therefore, infinitely many times $\Omega(q) = n$ in π so $\forall$ wins that play by the parity criterion. $\square$

The two above lemmas imply that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{R}) = \textsc{Transformation}(\mathcal{A})$, what concludes the proof of Theorem 1.

6. The Special Case of Büchi Automata

In this section we show how to derive the following special case of Theorem 1 from the results of [19] and [9]. For that we need to recall the following two results of the cited papers.

Corollary 11 ([9, Corollary 4.14]). *If $\mathcal{A}$ is an unambiguous Büchi automaton then $\mathcal{L}(\mathcal{A})$ is Borel.*

Theorem 12 ([19, Theorem 1]). *If $\mathcal{A}$ is a non-deterministic Büchi automaton over infinite trees then the language $\mathcal{L}(\mathcal{A})$ is Borel if and only if it can be recognised by a* $\mathrm{Comp}(0,0)$ *automaton. If the language is not Borel then it is $\mathbf{\Sigma}^1_1$-complete. Moreover, this dichotomy is effective.*

Using these two results, we obtain immediately the following theorem.

Theorem 13. *If $\mathcal{A}$ is an unambiguous Büchi automaton then $\mathcal{L}(\mathcal{A})$ can be recognised by a weak alternating automaton (i.e.* $\mathrm{Comp}(0,0)$ *automaton).*

Proof. By Corollary 11, the language recognised by an unambiguous Büchi automaton is Borel. Theorem 12 implies that if a Büchi automaton $\mathcal{A}$ recognises a Borel language then $\mathcal{L}(\mathcal{A})$ can (effectively) be recognised by a $\mathrm{Comp}(0,0)$ automaton. $\square$

Comparing to the construction presented in this paper, the above proof provides much worse complexity — the construction from [19] of the weak alternating automaton is exponential in the size of the original Büchi automaton.

7. The Case of Weak Unambiguous Automata

In this section we discuss the complexity of unambiguous automata with weak parity conditions. First, we observe that weak non-deterministic automata admit a separation result similar to the algorithm $\textsc{Separation}$. This is formalised as the algorithm $\textsc{WeakSeparation}$, see Subsec. 7.1. However, this algorithm cannot be used in a variant of the algorithm $\textsc{Transformation}$, as illustrated by Theorem 2.

The languages $\mathcal{L}(\mathcal{A}_n)$ are the ones used by Skurczyński [20] to prove that there are regular tree languages arbitrarily high in the finite levels of the Borel hierarchy. This theorem is proved in Subsec. 7.2.

Throughout this section we use the basic notions of descriptive set theory (i.e. completeness for finite levels of the Borel hierarchy). For a simple introduction to this subject, see [22] or [15].

Before we move to the technical part of the section, we want to recall that from the point of view of single branches, weak non-deterministic automata are weaker than the alternating ones.

Remark 14. The language of infinite trees that contain infinitely many letters a on the leftmost branch cannot be recognised by a weak non-deterministic automaton.

Proof. If $\mathcal{A}$ is a weak non-deterministic automaton over ω-words then $\mathcal{A}$ can be turned into an equivalent non-deterministic co-Büchi automaton $\mathcal{B}$. The construction of [14] allows us to turn $\mathcal{B}$ into an equivalent deterministic co-Büchi automaton $\mathcal{B}'$. Therefore, $\mathcal{L}(\mathcal{A}) \in \mathbf{\Sigma}^0_2$ what contradicts the fact that the set of ω-words containing infinitely many letters a is known to be $\mathbf{\Pi}^0_2$-complete. □

7.1. *Separation for weak automata*

For the sake of completeness, we state in this section an analogue of the algorithm Separation working for weak non-deterministic automata.

Algorithm 4: WeakSeparation

Input: Two weak non-deterministic automata $\mathcal{A}$, $\mathcal{B}$ of index $(0, j)$ such that $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}) = \emptyset$.

Output: Two weak alternating automata $\mathcal{R}$, $\mathcal{S}$ of index $(0, j)$ such that

$$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{R}) \quad \text{and} \quad \mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{S})^{\mathrm{c}} \quad \text{and} \quad \mathcal{L}(\mathcal{S}) \supseteq \mathcal{L}(\mathcal{B}).$$

Notice that in general we cannot take $\mathcal{R} = \mathcal{A}$ and $\mathcal{S}$ recognising the complement of $\mathcal{L}(\mathcal{A})$ because the complement of the language $\mathcal{L}(\mathcal{A})$ may itself require weak index $(1, j+1)$.

Outline of the construction. The construction follows the same lines as the one from Sec. 2.2 of [3]: we start from constructing a pathfinder out of the game for deciding non-emptiness; then we construct a product automaton of $\mathcal{A}$, $\mathcal{B}$, and the pathfinder. Finally, we need to define the acceptance condition for the new automaton — instead of constructing a $\mathrm{Comp}(i, 2j)$ automaton, we aim at constructing a pair of weak alternating automata that both have index $(0, j)$. This is achieved by following the construction of Theorem 9 from [2].

Details of the construction. We start by formalising the construction of a *pathfinder*, described in Sec. 2.2.1 of [3]. We call this algorithm Pathfinder.

```
Algorithm 5: PATHFINDER
  Input: Two non-deterministic automata $\mathcal{A}$, $\mathcal{B}$ such that $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}) = \emptyset$.
  Output: A finite set $M$, an element $m_{\mathrm{I}} \in M$, and a function
          $\sigma\colon M \times A \times \Delta^{\mathcal{A}} \times \Delta^{\mathcal{B}} \to M \times \{\mathtt{L}, \mathtt{R}\}$. Elements of $M$ are called
          memory states, $m_{\mathrm{I}}$ is the initial memory state, and $\sigma$ is a memory
          update function.
```

The crucial feature of the algorithm Pathfinder is expressed by Fact 15. Intuitively it says, that given two runs of the automata $\mathcal{A}$ and $\mathcal{B}$ respectively, the pathfinder is able to find an infinite branch on which at least one of the runs is rejecting (see Lemma 2.9 in [3]).

Fact 15. Fix two non-deterministic automata $\mathcal{A}$, $\mathcal{B}$ that recognise disjoint languages. Let $(M, m_{\mathrm{I}}, \sigma) = \text{Pathfinder}(\mathcal{A}, \mathcal{B})$. Consider arbitrary four sequences:

- $\delta_0, \delta_1, \ldots \in \Delta^{\mathcal{A}}$ of transitions of $\mathcal{A}$,
- $\delta'_0, \delta'_1, \ldots \in \Delta^{\mathcal{B}}$ of transitions of $\mathcal{B}$,
- $a_0, a_1, \ldots \in A$ of letters,
- $m_0, m_1, \ldots \in M$ of *memory states.*

such that they form a *run of* σ, that is:

- $\delta_0 = (q^{\mathcal{A}}_{\mathrm{I}}, a_0, q_{0,\mathtt{L}}, q_{0,\mathtt{R}})$; $\delta'_0 = (q^{\mathcal{B}}_{\mathrm{I}}, a_0, q'_{0,\mathtt{L}}, q'_{0,\mathtt{R}})$; and $m_0 = m_{\mathrm{I}}$;
- for every n the transitions δ_n and δ'_n are of the form $\delta_n = (q_n, a_n, q_{n,\mathtt{L}}, q_{n,\mathtt{R}})$ and $\delta'_n = (q'_n, a_n, q'_{n,\mathtt{L}}, q'_{n,\mathtt{R}})$ with the same letter a_n;
- for every n, we have $\sigma(m_n, a_n, \delta_n, \delta'_n) = (m_{n+1}, d_n)$ with $d_n \in \{\mathtt{L}, \mathtt{R}\}$ such that $q_{n+1} = q_{n,d_n}$ and $q'_{n+1} = q'_{n,d_n}$.

Then at most one of the sequences $\big(\Omega^{\mathcal{A}}(q_n)\big)_{n\in\mathbb{N}}$, $\big(\Omega^{\mathcal{B}}(q'_n)\big)_{n\in\mathbb{N}}$ satisfies the parity condition.

Let $\mathcal{A}$ and $\mathcal{B}$ be two weak non-deterministic automata recognising separate languages. Let $(M, m_{\mathrm{I}}, \sigma) = \text{Pathfinder}(\mathcal{A}, \mathcal{B})$ be the pathfinder constructed for them. Our aim is to combine the automata $\mathcal{A}$ and $\mathcal{B}$ together with the pathfinder $(M, m_{\mathrm{I}}, \sigma)$ to construct a pair $\mathcal{R}$, $\mathcal{S}$ of weak alternating automata such that the language $\mathcal{L}(\mathcal{R})$ is the complement of $\mathcal{L}(\mathcal{S})$ and they separate $\mathcal{L}(\mathcal{A})$ from $\mathcal{L}(\mathcal{B})$. The automata $\mathcal{R}$ and $\mathcal{S}$ will have almost the same structure, the only difference will be in the transitions and the acceptance condition. The set of states of $\mathcal{R}$ and $\mathcal{S}$ will essentially[b] be $Q^{\mathcal{A}} \times Q^{\mathcal{B}} \times M$. The initial state of both automata will be $(q^{\mathcal{A}}_{\mathrm{I}}, q^{\mathcal{B}}_{\mathrm{I}}, m_{\mathrm{I}})$. In the automaton $\mathcal{R}$ the role of $\exists$ (resp. $\forall$) will be to propose a run of $\mathcal{A}$ (resp. $\mathcal{B}$) over the given tree. The roles in the automaton $\mathcal{S}$ will be swapped. The directions in which the automata proceed will be chosen according to the pathfinder $(M, m_{\mathrm{I}}, \sigma)$.

[b]We will also use an additional memory structure $\mathcal{T}$ to define priorities of $\mathcal{R}$ and $\mathcal{S}$. This structure will be discussed later.

We present the transitions of $\mathcal{R}$ and $\mathcal{S}$ as Boolean combinations of successive directions and states following the presentation of [3], formally we should present them using intermediate states that allow players resolve the Boolean operators: $\vee$ corresponds to choices of $\exists$ and $\wedge$ to choices of $\forall$.

$$\delta^{\mathcal{R}}\big((q,q',m),a\big) \overset{\text{def}}{=} \bigvee_{\delta=\left(q,a,q_{\mathrm{L}},q_{\mathrm{R}}\right)\in\Delta^{\mathcal{A}}} \; \bigwedge_{\delta'=\left(q',a,q'_{\mathrm{L}},q'_{\mathrm{R}}\right)\in\Delta^{\mathcal{B}}} \big(d,(q_d,q'_d,m')\big),$$
$$\text{where } \sigma(m,a,\delta,\delta')=(m',d).$$
$$\delta^{\mathcal{S}}\big((q,q',m),a\big) \overset{\text{def}}{=} \bigwedge_{\delta=\left(q,a,q_{\mathrm{L}},q_{\mathrm{R}}\right)\in\Delta^{\mathcal{A}}} \; \bigvee_{\delta'=\left(q',a,q'_{\mathrm{L}},q'_{\mathrm{R}}\right)\in\Delta^{\mathcal{B}}} \big(d,(q_d,q'_d,m')\big),$$
$$\text{where } \sigma(m,a,\delta,\delta')=(m',d).$$

The priorities of the states of these automata will be computed by a memory structure $\mathcal{T}$ in such a way to guarantee the following invariants. Consider an infinite play π of the acceptance game for the constructed automata in which the sequence of states of $\mathcal{A}$ was $q_0, q_1, \ldots$ and the sequence of states of $\mathcal{B}$ was $q'_0, q'_1, \ldots$. The invariants say that:

$$\Big((q_n)_{n\in\mathbb{N}} \text{ is accepting in } \mathcal{A}\Big) \Longrightarrow \Big(\pi \text{ is accepting in } \mathcal{R}\Big) \tag{7.1}$$
$$\Big((q'_n)_{n\in\mathbb{N}} \text{ is accepting in } \mathcal{B}\Big) \Longrightarrow \Big(\pi \text{ is accepting in } \mathcal{S}\Big) \tag{7.2}$$
$$\Big(\pi \text{ is accepting in } \mathcal{R}\Big) \Longleftrightarrow \Big(\pi \text{ is rejecting in } \mathcal{S}\Big) \tag{7.3}$$

If both sequences $(q_n)_{n\in\mathbb{N}}$ and $(q'_n)_{n\in\mathbb{N}}$ are rejecting in $\mathcal{A}$ and $\mathcal{B}$ respectively, then the play π can be either accepting or rejecting in $\mathcal{R}$, but Invariant (7.3) still needs to hold. Fact 15 implies that it is never the case that both sequences $(q_n)_{n\in\mathbb{N}}$ and $(q'_n)_{n\in\mathbb{N}}$ are accepting in $\mathcal{A}$ and $\mathcal{B}$ respectively.

Lemma 16. *If Invariants* (7.1), (7.2), *and* (7.3) *are satisfied by automata $\mathcal{R}$ and $\mathcal{S}$ then the following holds*

$$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{R}) \quad \textit{and} \quad \mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{S})^{\mathrm{c}} \quad \textit{and} \quad \mathcal{L}(\mathcal{S}) \supseteq \mathcal{L}(\mathcal{B}).$$

Proof. The proof is the same as in [3]. Consider a tree $t \in Tr_{\Sigma}$.

If $t \in \mathcal{L}(\mathcal{A})$ then $\exists$ has a strategy in the game $\mathcal{G}(\mathcal{R}, t)$ in which she plays a fixed accepting run of $\mathcal{A}$ over t. This strategy is winning by Invariant 7.1 and therefore $t \in \mathcal{L}(\mathcal{R})$.

If $t \in \mathcal{L}(\mathcal{B})$ then $\exists$ has a strategy in the game $\mathcal{G}(\mathcal{S}, t)$ in which she plays a fixed accepting run of $\mathcal{B}$ over t. This strategy is winning by Invariant 7.2 and therefore $t \in \mathcal{L}(\mathcal{S})$.

Since the sets of accepting plays of $\mathcal{R}$ and $\mathcal{S}$ are complementary (Invariant 7.3) and the players in charge of the transitions are swapped, $\exists$ wins $\mathcal{G}(\mathcal{R}, t)$ if and only if she loses $\mathcal{G}(\mathcal{S}, t)$. Therefore, the language $\mathcal{L}(\mathcal{R})$ is the complement of $\mathcal{L}(\mathcal{S})$. $\square$

What remains is to define priorities in $\mathcal{R}$ and $\mathcal{S}$ in a way which guarantees the above invariants. This is achieved by a variant of construction in Lemma 8 of [2]. We define a memory structure (i.e. transducer) $\mathcal{T}$ that reads two non-decreasing sequences of priorities $o_0, o_1, \ldots \in \{0, 1, \ldots, j\}$ and $p_0, p_1, \ldots \in \{0, 1, \ldots, j\}$. $\mathcal{T}$ will construct two new non-decreasing sequences of priorities that will be used by the automata $\mathcal{R}$ and $\mathcal{S}$ respectively. Formally, we will have $o_n \stackrel{\text{def}}{=} \Omega^{\mathcal{A}}(q_n)$ and $p_n \stackrel{\text{def}}{=} \Omega^{\mathcal{B}}(q'_n)$ for $n = 0, 1, \ldots$.

To simplify the presentation, we will encode the given non-decreasing sequences $(o_n)_{n\in\mathbb{N}}$ and $(p_n)_{n\in\mathbb{N}}$ by one sequence of *increases* $(I_n)_{n\in\mathbb{N}}$ over the alphabet $\{0, I_A, I_B\}$: without loss of generality we assume that the initial pair (o_0, p_0) is $(0, 0)$; and encode each successive pair (o_{n+1}, p_{n+1}) as the non-empty word

$$w_n \stackrel{\text{def}}{=} 0\ I_A^{o_{n+1}-o_n}\ I_B^{p_{n+1}-p_n} \quad \in \{0, I_A, I_B\}^*.$$

Then define the infinite sequence $(I_n)_{n\in\mathbb{N}}$ that encodes $\langle (o_n)_{n\in\mathbb{N}}, (p_n)_{n\in\mathbb{N}}\rangle$ as $(I_n)_{n\in\mathbb{N}} \stackrel{\text{def}}{=} w_0 \cdot w_1 \cdot w_2 \cdots$. We call a sequence over the alphabet $\{0, I_A, I_B\}$ *correct* if it contains at most j occurrences of I_A and at most j occurrences of I_B. Notice that the sequence $(I_n)_{n\in\mathbb{N}}$ is correct because each $o_n, p_n \leq j$. We restrict our attention to correct sequences $(I_n)_{n\in\mathbb{N}}$. We call such a sequence I_A*-accepting* if the number of I_A is even; I_B*-accepting* if the number of I_B is even; and I_AI_B*-accepting* if both the numbers are even. Fact 15 implies that we will never encounter an I_AI_B-accepting sequence in our construction.

The set of states of $\mathcal{T}$, to which we refer as the memory structure, is $H = \{0, 1, \ldots, j\}^4$ with the initial memory state $(0, 0, 0, 0)$. Consider a memory state $(o, p, r, s) \in H$. The coordinates o and p count the number of letters I_A and I_B that were seen so far (i.e. they decode the sequences o_n and p_n on the fly). The coordinates r and s will indicate the expected priorities of the automata $\mathcal{R}$ and $\mathcal{S}$. Consider the pseudo-code Transition presented in Algorithm 6 for updating the memory state $\delta^{\mathcal{T}}\big((o, p, r, s), a\big) \stackrel{\text{def}}{=} \textsc{Transition}(o, p, r, s, a)$.

Now automata $\mathcal{R}$ and $\mathcal{S}$ are defined as the product of $\mathcal{A}$, $\mathcal{B}$, M, and $\mathcal{T}$, where the priorities of $\mathcal{R}$ are defined as the r component of the current memory state of $\mathcal{T}$ and the priorities of $\mathcal{S}$ are defined as the s component of the current memory state of $\mathcal{T}$. What remains is to argue that $\delta^{\mathcal{T}}$ actually produces values inside $\{0, 1, \ldots, j\}^4$ and Invariants (7.1), (7.2), and (7.3) are satisfied. This aim is explained on Fig. 3.

Lemma 17. *The transitions of $\mathcal{T}$ are well defined, i.e. if $(I_n)_{n\in\mathbb{N}}$ is a correct sequence then $\mathcal{T}\big((I_n)_{n\in\mathbb{N}}\big)$ never reaches a value greater than j.*

Proof. This lemma follows inductively from an invariant stating that

$$r, s \leq \frac{o + p + 1}{2}. \qquad \square$$

The next two lemmas imply Invariants (7.1), (7.2), and (7.3). Consider a correct sequence $(I_n)_{n\in\mathbb{N}} \in \{0, I_A, I_B\}^\omega$ and assume that it is not I_AI_B-accepting. Let

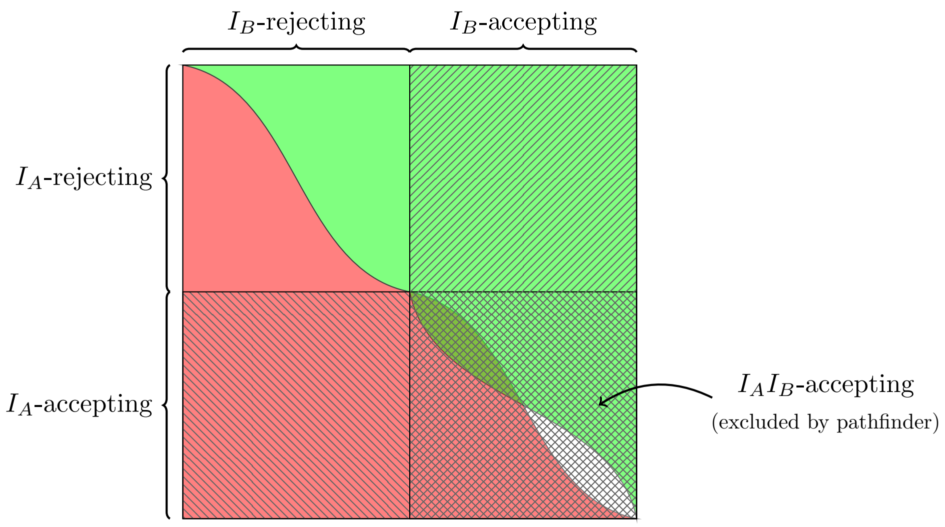


Fig. 3. The big square represents all the correct sequences $(I_n)_{n\in\mathbb{N}}$. The I_AI_B-accepting sequences (right-bottom square) are excluded by the properties of the pathfinder. Our aim is to separate the left-bottom square from the right-top one, using two sets: the red set corresponds to the acceptance of $\mathcal{R}$; the green set corresponds to the acceptance by $\mathcal{S}$. The constructed sets need to complement each other in the left-upper square. Their behaviour in the right-bottom square does not matter.

$\big((o_n, p_n, r_n, s_n)\big)_{n\in\mathbb{N}} = \mathcal{T}\big((I_n)_{n\in\mathbb{N}}\big)$ be the sequence of memory states of $\mathcal{T}$ when reading $(I_n)_{n\in\mathbb{N}}$. Assume that $r_{\max}$, and $s_{\max}$ are the maximal values of the respective coordinates in the sequence.

Lemma 18. *If $(I_n)_{n\in\mathbb{N}}$ is I_A-accepting then $r_{\max}$ is even. If $(I_n)_{n\in\mathbb{N}}$ is I_B-accepting then $s_{\max}$ is even.*

Proof. Assume that $(I_n)_{n\in\mathbb{N}}$ is I_A-accepting. By the assumption we know that $(I_n)_{n\in\mathbb{N}}$ is not I_B-accepting. It means that from some point on we have o_n even and p_n odd. Therefore, we define r_{n+1} to be even as well (lines 13 to 16 of the algorithm). The dual case is symmetric. □

The next lemma implies Invariant (7.3) and concludes the proof of correctness of the presented construction.

Lemma 19. *Exactly one of $r_{\max}$ and $s_{\max}$ is even.*

Proof. Let N be the greatest number such that in the execution of TRANSITION(o_N, p_N, r_N, s_N) the line 13 was reached (i.e. the algorithm did not return in lines 2 nor 12). Such N must exist because the sequence $(I_n)_{n\in\mathbb{N}}$ contains at least one symbol I_A or I_B and is ultimately constant 0.

Therefore, $o_{N+1} + p_{N+1} = 1 \mod 2$; $r_{N+1} + o_{N+1} = 0 \mod 2$; and $s_{N+1} + p_{N+1} = 0 \mod 2$, what implies that $r_{N+1} + s_{N+1} = 1 \mod 2$. In the executions

Algorithm 6: Transition

```
Input: A state $(o, p, r, s) \in H$ and a letter $a \in \{0, I_A, I_B\}$
Output: A state $(o', p', r', s') \in H$
1  if $a = 0$ then
2      return $(o, p, r, s)$
3  if $a = I_A$ then
4      $o' \leftarrow o' + 1$
5  else
6      $o' \leftarrow o'$
7  if $a = I_B$ then
8      $p' \leftarrow p' + 1$
9  else
10     $p' \leftarrow p'$
11 if $o' + p' = 0 \mod 2$ then
12     return $(o', p', r, s)$
   /* We know at this point that $o' + p' = 1 \mod 2$. In other words
      exactly one of the automata $\mathcal{A}$ and $\mathcal{B}$ seems to accept.      */
13 if $r + o' = 1 \mod 2$ then
14     $r' \leftarrow r + 1$
15 else
16     $r' \leftarrow r$
17 if $s + p' = 1 \mod 2$ then
18     $s' \leftarrow s + 1$
19 else
20     $s' \leftarrow s$
   /* We know at this point that $r' + o' = 0 \mod 2$ and $s' + p' = 0$
      $\mod 2$.                                                           */
21 return $(o', p', r', s')$
```

of Transition(o_n, p_n, r_n, s_n) for $n > N$ the values of r and s are not modified because line 13 is not reached there. It means that $r_{\max} = r_{N+1}$ and $s_{\max} = s_{N+1}$ so $r_{\max} + s_{\max} = 1 \mod 2$. Thus, exactly one of these numbers is even. □

7.2. *Hardness for weak unambiguous automata*

In this section we prove Theorem 2. We take $A = \{a, b\}$ as the alphabet. The construction will be inductive. We start from a pair of automata $\mathcal{A}_0$ and $\mathcal{B}_0$ defined by the following procedure.

Algorithm 7: BaseCase

```
Output: A pair of weak non-det. automata $\mathcal{R}$ and $\mathcal{S}$
1  $Q^{\mathcal{R}} \leftarrow \{\star, \top, \bot\}$
2  $Q^{\mathcal{S}} \leftarrow \{\star, \top\}$
3  $\Delta^{\mathcal{R}} \leftarrow \{(\star, a, \star, \top), (\star, b, \bot, \bot), (\top, \_, \top, \top), (\bot, \_, \bot, \bot)\}$
4  $\Delta^{\mathcal{S}} \leftarrow \{(\star, a, \star, \top), (\star, b, \top, \top), (\top, \_, \top, \top)\}$
5  $q_{\mathrm{I}}^{\mathcal{R}} \leftarrow \star$
6  $q_{\mathrm{I}}^{\mathcal{S}} \leftarrow \star$
7  $\Omega^{\mathcal{R}}(\star) \leftarrow 0$
8  $\Omega^{\mathcal{R}}(\top) \leftarrow 0$
9  $\Omega^{\mathcal{R}}(\bot) \leftarrow 1$
10 $\Omega^{\mathcal{S}}(\star) \leftarrow 1$
11 $\Omega^{\mathcal{S}}(\top) \leftarrow 2$
```

Fact 20. Let $(\mathcal{A}_0, \mathcal{B}_0) = $ BaseCase(). Then both automata $\mathcal{A}_0$ and $\mathcal{B}_0$ are weak and unambiguous, they recognise complementary languages, the index of $\mathcal{A}_0$ is $(0,1)$ and the index of $\mathcal{B}_0$ is $(1,2)$.

Now, starting from this base case we construct inductively our sequence of automata using the following algorithm.

Algorithm 8: Increase

```
Input: A pair of weak non-det. automata: $\mathcal{A}$ of index $(0, j)$ and $\mathcal{B}$ of index
       $(1, j+1)$
Output: A pair of weak non-det. automata $\mathcal{R}$ and $\mathcal{S}$
1  $Q^{\mathcal{R}} \leftarrow Q^{\mathcal{B}} \sqcup \{\star\}$ /* the union is disjoint: we add a fresh state
       $\star \notin Q^{\mathcal{B}}$                                              */
2  $Q^{\mathcal{S}} \leftarrow Q^{\mathcal{A}} \sqcup Q^{\mathcal{B}} \sqcup \{\star, \top\}$/* the same holds here for $\star$ and $\top$        */
3  $\Delta^{\mathcal{R}} \leftarrow \Delta^{\mathcal{B}} \sqcup \{(\star, \_, \star, q_{\mathrm{I}}^{\mathcal{B}})\}$
4  $\Delta^{\mathcal{S}} \leftarrow \Delta^{\mathcal{A}} \sqcup \Delta^{\mathcal{B}} \cup \{(\star, \_, \star, q_{\mathrm{I}}^{\mathcal{B}}), (\star, \_, \top, q_{\mathrm{I}}^{\mathcal{A}}), (\top, \_, \top, \top)\}$
5  $q_{\mathrm{I}}^{\mathcal{R}} \leftarrow \star$
6  $q_{\mathrm{I}}^{\mathcal{S}} \leftarrow \star$
7  $\Omega^{\mathcal{R}}(\star) \leftarrow 0$
8  $\Omega^{\mathcal{S}}(\star) \leftarrow 1$
9  $\Omega^{\mathcal{S}}(\top) \leftarrow 2$
10 foreach $q \in Q^{\mathcal{B}}$ do
11     $\Omega^{\mathcal{R}}(q) \leftarrow \Omega^{\mathcal{B}}(q)$
12     $\Omega^{\mathcal{S}}(q) \leftarrow \Omega^{\mathcal{B}}(q)$
13 foreach $q \in Q^{\mathcal{A}}$ do
14     $\Omega^{\mathcal{S}}(q) \leftarrow \Omega^{\mathcal{A}}(q) + 2$
```

Lemma 21. *Let $\mathcal{A}$, $\mathcal{B}$ be two weak non-deterministic automata over the same alphabet A. Assume that $\mathcal{A}$ has index $(0, j)$ and $\mathcal{B}$ has index $(1, j+1)$; and additionally $\mathcal{L}(\mathcal{A})$ is the complement of $\mathcal{L}(\mathcal{B})$. Let $(\mathcal{R}, \mathcal{S}) = \textsc{Increase}(\mathcal{A}, \mathcal{B})$. Then:*

$$\mathcal{L}(\mathcal{R}) = \big\{t \in Tr_A \mid \forall_{n\in\mathbb{N}}\, t{\restriction}_{\mathtt{L}^n\mathtt{R}} \in \mathcal{L}(\mathcal{B})\big\},$$
$$\mathcal{L}(\mathcal{S}) = \big\{t \in Tr_A \mid \exists_{n\in\mathbb{N}}\, t{\restriction}_{\mathtt{L}^n\mathtt{R}} \in \mathcal{L}(\mathcal{A})\big\}.$$

In particular, $\mathcal{L}(\mathcal{R})$ is the complement of $\mathcal{L}(\mathcal{S})$. Additionally, if $\mathcal{A}$ and $\mathcal{B}$ unambiguous then so are $\mathcal{R}$ and $\mathcal{S}$.

The automaton $\mathcal{R}$ has index $(0, j+1)$ and the automaton $\mathcal{S}$ has index $(1, j+2)$.

Proof. The only non-trivial statement is the one about unambiguity of $\mathcal{S}$. Consider a tree $t \in Tr_A$. There is at most one N such that:

- $t{\restriction}_{\mathtt{L}^N\mathtt{R}} \in \mathcal{L}(\mathcal{A})$,
- for all $n < N$ we have $t{\restriction}_{\mathtt{L}^n\mathtt{R}} \notin \mathcal{L}(\mathcal{A})$.

Since the language $\mathcal{L}(\mathcal{A})$ is the complement of $\mathcal{L}(\mathcal{B})$, the second item above implies that for all $n < N$ we have $t{\restriction}_{\mathtt{L}^n\mathtt{R}} \in \mathcal{L}(\mathcal{B})$.

The initial state of $\mathcal{S}$ is $\star$ and the only transitions from $\star$ are $(\star, \_, \star, q_{\mathrm{I}}^{\mathcal{B}})$ and $(\star, \_, \top, q_{\mathrm{I}}^{\mathcal{A}})$. Thus, the state $\star$ appears only in the nodes of the form $\mathtt{L}^n$ for $n \in \mathbb{N}$. For each such node, the states $q_{\mathrm{I}}^{\mathcal{A}}$ and $q_{\mathrm{I}}^{\mathcal{B}}$ verify if the subtree $t{\restriction}_{\mathtt{L}^n\mathtt{R}}$ for $n \in \mathbb{N}$ belongs to $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{B})$ respectively. Since $\mathcal{A}$ and $\mathcal{B}$ are unambiguous, there is at most one accepting run of the respective automaton over the considered subtree $t{\restriction}_{\mathtt{L}^n\mathtt{R}}$. Therefore, there is at most one accepting run of $\mathcal{S}$ over t. In this run the state $\star$ appears exactly in the nodes $\mathtt{L}^n$ for $n \leq N$. □

Definition 22. Let

$$(\mathcal{A}_0, \mathcal{B}_0) \stackrel{\text{def}}{=} \textsc{BaseCase}()$$
$$(\mathcal{A}_{n+1}, \mathcal{B}_{n+1}) \stackrel{\text{def}}{=} \textsc{Increase}(\mathcal{A}_n, \mathcal{B}_n) \qquad \text{for } n = 0, 1, 2, \ldots$$

Lemma 23. *For every $n = 0, \ldots$ the automata $\mathcal{A}_n$ and $\mathcal{B}_n$ are weak, unambiguous, recognise complementary languages, and their indices are respectively $(0, n+1)$ and $(1, n+2)$.*

Proof. Easy induction using Lemma 21. □

To prove that the languages of the automata $\mathcal{A}_n$ and $\mathcal{B}_n$ do not admit any complexity collapse, we will use topological methods, see for instance [8] or [15].

Theorem 24 ([8]). *If L is recognizable by a weak alternating automaton of index $(0, j)$ then $L \in \mathbf{\Pi}^0_j$. Dually, for index $(1, j+1)$, we have $L \in \mathbf{\Sigma}^0_j$.*

Therefore, it is enough to observe the following lemma. The reasoning is very similar to Lemma 3.2 in [20].

Lemma 25. *For every $n = 0, \ldots$ the language $\mathcal{L}(\mathcal{A}_n)$ is $\mathbf{\Pi}^0_{n+1}$-complete and the language $\mathcal{L}(\mathcal{B}_n)$ is $\mathbf{\Sigma}^0_{n+1}$-complete. In particular, $\mathcal{L}(\mathcal{A}_n) \notin \mathbf{\Sigma}^0_{n+1}$ and $\mathcal{L}(\mathcal{B}_n) \notin \mathbf{\Pi}^0_{n+1}$.*

Proof. Induction on n using Exercise 23.3 in [11]. $\square$

This concludes the proof of Theorem 2 — the automata $\mathcal{A}_n$ and $\mathcal{B}_n$ are weak unambiguous of appropriate indices. Additionally, if a weak alternating automaton $\mathcal{C}$ recognises the same language as $\mathcal{A}_n$ (resp. $\mathcal{B}_n$) then the index of $\mathcal{C}$ is at least as high as the index of $\mathcal{A}_n$ (resp. $\mathcal{B}_n$).

8. Conclusion

This work studies the question how to use the fact that a given automaton $\mathcal{A}$ is unambiguous to provide an upper bound for the complexity of the language recognised by $\mathcal{A}$. The answer to that question depends on the acceptance condition of the automaton. In the case of general parity automata, the paper provides an algorithm TRANSFORMATION which, for a given unambiguous automaton $\mathcal{A}$ of index $(i, 2j)$, outputs an automaton TRANSFORMATION$(\mathcal{A})$ accepting the same language and belonging to the class $\mathrm{Comp}(i+1, 2j)$. The core of this algorithm is a separation procedure by Arnold and Santocanale [3]. The algorithm TRANSFORMATION provides an effective drop in the descriptive complexity of the language using its unambiguity. The construction can be considered as an automata-theoretic counterpart of the Lusin-Souslin Theorem [11, Theorem 15.1].

The situation is different in the case of automata with the weak parity acceptance condition. In that case, a similar separation result holds, see Algorithm 4. However, as Theorem 2 shows, unambiguous automata with the weak parity acceptance condition admit no drop in the descriptive complexity of their languages.

Acknowledgments

This work has been supported by Polish National Science Centre grant no. 2014-13/B/ST6/03595.

References

[1] A. Arnold, The μ-calculus alternation-depth hierarchy is strict on binary trees, *ITA* **33**(4/5) (1999) 329–340.

[2] A. Arnold, H. Michalewski and D. Niwiński, On the separation question for tree languages, in *STACS* (Leibniz, 2012), pp. 396–407.

[3] A. Arnold and L. Santocanale, Ambiguous classes in μ-calculi hierarchies, *TCS* **333**(1–2) (2005) 265–296.

[4] J. Bradfield, Simplifying the modal μ-calculus alternation hierarchy, in *STACS* (Springer, 1998), pp. 39–49.

[5] A. Carayol, C. Löding, D. Niwiński and I. Walukiewicz, Choice functions and well-orderings over the infinite binary tree, *Cent. Europ. J. of Math.* **8** (2010) 662–682.

[6] T. Colcombet, Forms of determinism for automata (invited talk), in *STACS* (Leibniz, 2012), pp. 1–23.

[7] J. Duparc, K. Fournier and S. Hummel, On unambiguous regular tree languages of index (0, 2), in *CSL* (Leibniz, 2015), pp. 534–548.

[8] J. Duparc and F. Murlak, On the topological complexity of weakly recognizable tree languages, in *FCT* (Springer, 2007), pp. 261–273.

[9] O. Finkel and P. Simonnet, On recognizable tree languages beyond the Borel hierarchy, *Fundam. Inform.* **95**(2–3) (2009) 287–303.

[10] S. Hummel, Unambiguous tree languages are topologically harder than deterministic ones, in *GandALF* (2012), pp. 247–260; arXiv: 1210.2028.

[11] A. Kechris, *Classical Descriptive Set Theory* (Springer-Verlag, New York, 1995).

[12] O. Kupferman and M. Y. Vardi, The weakness of self-complementation, in *STACS* (Springer, 1999), pp. 455–466.

[13] H. Michalewski and M. Skrzypczak, Unambiguous büchi is weak, in *DLT* (2016), pp. 319–331, abs/1401.4025.

[14] S. Miyano and T. Hayashi, Alternating finite automata on omega-words, *Theor. Comput. Sci.* **32** (1984) 321–330.

[15] F. Murlak, *Effective Topological Hierarchies of Recognizable Tree Languages*, PhD thesis, University of Warsaw (2008).

[16] D. Niwiński and I. Walukiewicz, Ambiguity problem for automata on infinite trees, unpublished (1996).

[17] M. O. Rabin, Weakly definable relations and special automata, in *Proceedings of the Symposium on Mathematical Logic and Foundations of Set Theory* (North-Holland, 1970), pp. 1–23.

[18] M. Skrzypczak, Unambiguous languages exhaust the index hierarchy, *CoRR* (2018), abs/1803.06163.

[19] M. Skrzypczak and I. Walukiewicz, Deciding the topological complexity of Büchi languages, in *ICALP (2)* (Leibniz, 2016), pp. 99:1–99:13.

[20] J. Skurczyński, The Borel hierarchy is infinite in the class of regular sets of trees, *Theoretical Computer Science* **112**(2) (1993) 413–418.

[21] W. Thomas, Languages, automata, and logic, in *Handbook of Formal Languages* (Springer, 1996), pp. 389–455.

[22] W. Thomas and H. Lescow, Logical specifications of infinite computations, in *REX School/Symposium* (Springer, 1993), pp. 583–621.