Unambiguous Büchi is weak

Henryk Michalewski and Michał Skrzypczak

University of Warsaw
Banacha 2 Warsaw, Poland
{h.michalewski,m.skrzypczak}@mimuw.edu.pl

Abstract. A non-deterministic automaton on infinite trees is *unambiguous* if it has at most one accepting run on every tree. For a given unambiguous parity automaton $\mathcal{A}$ of index $(i, 2j)$ we construct an alternating automaton $\textsc{Transformation}(\mathcal{A})$ which accepts the same language, but is simpler in terms of alternating hierarchy of automata. 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 $\mathrm{Comp}(i + 1, 2j)$, in particular it is simultaneously of alternating index $(i, 2j)$ and of the dual index $(i + 1, 2j + 1)$. The main theorem of this paper is a correctness proof of the algorithm $\textsc{Transformation}$. The transformation algorithm is based on a separation algorithm of Arnold and Santocanale [2] and extends results of Finkel and Simonnet [7].

1 Introduction

Determinising a given computation typically leads to an additional cost. Presence of such cost inspires investigation of intermediate models of computations. Here we focus on *unambiguity*, that is the requirement that there are no two distinct accepting computations on the same input. In the case of finite and infinite words a given automaton can be determinised at an exponential cost, but in the case of infinite trees there are automata which cannot be determinised at all. Moreover, there are automata for which one cannot find an equivalent unambiguous automaton [12]. Also, there exist unambiguous automata which cannot be simulated by deterministic ones [8] (see Figure 1).

Most questions about automata on finite or infinite words are decidable. However, in the case of automata on infinite trees many fundamental decidability problems are open, unless we limit attention to deterministic automata. Then it is decidable whether a given language is recognisable by a deterministic automaton [15], the non-deterministic index problem is decidable [13,14], as well as it is possible to locate the language in the Wadge hierarchy [11]. Moving beyond deterministic automata is a topic of an on-going research [5,6] and the study of unambiguous automata is a part of this effort. Admittedly, problems for this class seem to be much harder than for deterministic automata, in particular one can decide if a given automaton is unambiguous, but it is an open problem, whether a given regular language is unambiguous. Additionally, there are no upper bounds on the descriptive complexity (e.g. the parity index) or topological complexity of unambiguous languages among all regular tree languages.

In this work we focus on descriptive complexity and a fortiori also on topological complexity of languages defined by unambigous automata. The most canonical measure of descriptive complexity of regular tree languages is the *parity index*. 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 corresponds to the index $(1, 2)$. It was shown in [1,3] that some languages require 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. It means that the non-deterministic and alternating index hierarchies are *strict*.

We will show that the fact that a given automaton is unambiguous allows to effectively find another equivalent automaton with a simpler acceptance condition. More precisely, in Section 4 we propose an algorithm Transformation with the following properties:

Theorem 1. *For an unambiguous Büchi automaton $\mathcal{A}$,* Transformation$(\mathcal{A})$ *is a weak alternating automaton recognising the same language. More generally, 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 $(i+1, 2j+1)$.*

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

This theorem implies in particular that there is no unambiguous Büchi automaton which is strictly of index $(1, 2)$. Since a language accepted by an unambiguous Büchi automaton is also accepted by a weak alternating automaton, topologically such languages must be located at a finite level of the Borel hierarchy. One should note that in the above theorem and in the algorithm Transformation, the automaton must be simultaneously unambiguous and of appropriate index. It is still possible for a regular tree language to be both: recognised by some unambiguous automaton and by some other Büchi automaton. An example of such a language is the H-language proposed in [8]: „there exists a branch containing only a's and turning infinitely many times right", see Figure 1.

1.1 Related work

There exist two estimates on descriptive complexity of unambiguous languages. Firstly, a result of Hummel [8] shows that unambiguous languages are topologically harder than deterministic ones, see Figure 1. Secondly, Finkel and Simonnet [7] proved using the Lusin-Souslin Theorem [9, Theorem 15.1] from descriptive set theory, that any language recognised by an unambiguous Büchi automaton must be Borel.

Our theorem involves not only a set-theoretical argument but also an automata construction encapsulated by the algorithm Transformation. Our result also gives a stronger information about the descriptive complexity, since (1) it is an

open problem whether for a given regular Borel language of infinite trees does exist a weak alternating automaton accepting this language, (2) our TRANSFORMATION algorithm works for arbitrary parities and it is not clear how to generalize the set-theoretical method of Finkel and Simonnet [7] beyond Büchi automata.

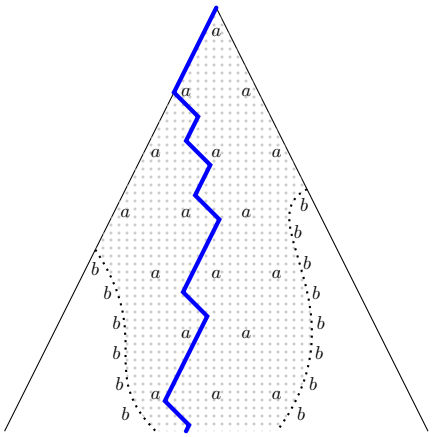

Fig. 1. A tree from the language H—the tree is labelled by letters a and b, the dotted region contains vertices reachable from the root by a-vertices. The blue thick branch is a branch consisting of a-vertices that turns R infinitely many times.

The Lusin-Souslin Theorem used in [7] says that if $f\colon X \to Y$ is injective and Borel then the image $f[X]$ is Borel in Y. The proof of this theorem is based on the Lusin Separation Theorem [9, Theorem 14.7]. These theorems are set-theoretical in nature and the result in this work can be considered as an automata-theoretic counterpart of the Lusin-Suslin theorem. As a sub-procedure in the algorithm TRANSFORMATION we use an algorithm SEPARATION from [2], which itself is an automata-theoretic counterpart of the Lusin Separation Theorem.

To the authors' best knowledge this is the first work where it is shown how to use the fact that a given automaton is unambiguous to derive upper bounds on the parity index of the recognised language. Therefore, this work should be treated as a first step towards descriptive complexity bounds for unambiguous languages, and generally better understanding of this class of automata.

1.2 Outline of the paper

We first prove Lemma 1 which states that if an automaton is unambiguous then the transitions of the automaton correspond to disjoint languages. In the algorithm PARTITION we use an algorithm of Arnold and Santocanale and show that these disjoint languages can be separated by $\mathrm{Comp}(i+1, 2j)$ languages.

In Section 4 we provide a construction of the automaton TRANSFORMATION$(\mathcal{A})$ and in Section 5.1 we conclude the proof of Theorem 1 by proving correctness of this construction.

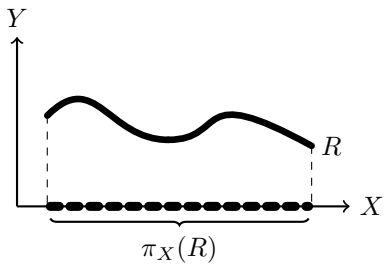

Fig. 2. An illustration of Lusin-Souslin Theorem. A relation $R \subseteq X \times Y$ is Borel and *uniformised*. The theorem implies that $\pi_X(R) \subseteq X$ is Borel as well.

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 [4]. A general background on automata and logic over infinite trees can be found in [17].

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 $t\colon \{\mathtt{L},\mathtt{R}\}^* \to A$. The set of all such trees is Tr_A. Vertices of a tree are denoted $u, v, w \in \{\mathtt{L},\mathtt{R}\}^*$. The prefix-order on vertices is $\preceq$, the minimal element of this order is the *root* $\epsilon \in \{\mathtt{L},\mathtt{R}\}^*$. The label of a tree $t \in Tr_A$ in a vertex $u \in \{\mathtt{L},\mathtt{R}\}^*$ is $t(u) \in A$. $t{\restriction_u}$. 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 $\alpha{\restriction_k}$ we denote the prefix of α of length k (e.g. $\alpha{\restriction_0} = \epsilon$).

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

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

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 $(\epsilon, q_{\mathrm{I}})$. 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 $\{i, i+1, \ldots, j\}$. 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 [2].

Note that an alternating automaton $\mathcal{C}$ is $\mathrm{Comp}(0, 0)$ if and only if $\mathcal{C}$ is a weak alternating automaton in the meaning of [10]. 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 2 (Rabin [16], also Kupferman Vardi [10]). *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 [2]. A particular case of this algorithm for $i = j = 1$ is the classical Rabin separation construction (see [16]): 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 $\mathcal{A}_1$, $\mathcal{A}_2$ of index $(i, 2j)$ such that $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) = \emptyset$.

Output: An alternating $\mathrm{Comp}(i{+}1, 2j)$ automaton $\mathcal{S}$ such that

$$\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{S}) \quad \text{and} \quad \mathcal{L}(\mathcal{A}_2) \cap \mathcal{L}(\mathcal{S}) = \emptyset.$$

3 Partition property

In this section we will prove Lemma 1 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 $\delta = (q, a, q_{\mathtt{L}}, q_{\mathtt{R}})$ 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 $\delta = (q, a, q_{\mathtt{L}}, q_{\mathtt{R}})$ 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 1. *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 $\mathcal{C}_\delta$

foreach $(q, a) \in Q \times A$, *productive* **do**
  foreach $\delta \in \Delta_{q,a}$ **do**
    $\mathcal{E}_\delta \leftarrow$ *non-det.* $(i, 2j)$ *automaton recognising* L_δ
    $\mathcal{F}_\delta \leftarrow$ *non-det.* $(i, 2j)$ *automaton recognising* $\bigcup_{\eta \in \Delta_{q,a}, \eta \neq \delta} L_\eta$
  foreach $\delta \in \Delta_{q,a}$ **do**
    $\mathcal{D}_\delta \leftarrow \textsc{Separation}(E_\delta, F_\delta)$
  foreach $\delta \in \Delta_{q,a}$ **do**
    $\mathcal{C}_\delta \leftarrow \mathrm{Comp}(i{+}1, 2j)$ *automaton recognising* $\mathcal{L}(\mathcal{D}_\delta) \setminus \bigcup_{\eta \neq \delta} \mathcal{L}(\mathcal{D}_\eta)$.
  $\mathcal{B}_{q,a} \leftarrow \mathrm{Comp}(i{+}1, 2j)$ *automaton recognising* $Tr_A \setminus \bigcup_{\delta \in \Delta_{q,a}} \mathcal{L}(\mathcal{D}_\delta)$.
foreach $\delta = (q, a, q_{\mathtt{L}}, q_{\mathtt{R}}) \in \Delta_{q,a}$ *with* (q, a) *non-productive* **do**
  $\mathcal{C}_\delta \leftarrow \mathrm{Comp}(0, 0)$ *automaton recognising the empty language.*

The following lemma summarizes properties of the algorithm Partition.

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

A proof of this lemma follows directly from the definition of the respective automata, see Figure 3 for an illustration of this construction.

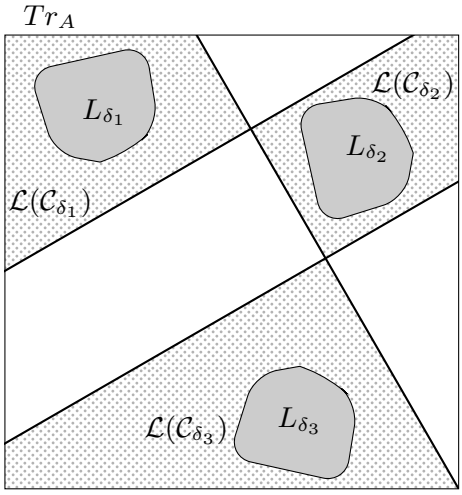

Fig. 3. An illustration of the output of the algorithm Partition. The three circles are the languages L_{δ_i} for the transitions starting in a fixed pair (q, a). Each straight line represents the language $\mathcal{L}(\mathcal{D}_{\delta_i})$ that separates the respective language L_{δ_i} from the others. Our construction provides the automata $\mathcal{C}_{\delta_i}$ recognising the dotted regions.

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 1. 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. lim sup) 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.

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;

Algorithm 3: TRANSFORMATION

Input: An unambiguous automaton $\mathcal{A}$ of index $(i, 2j)$
Output: An automaton $\mathcal{R}$

$N \leftarrow \{\star\} \cup \{n \in \{i, \dots, 2j\} \mid n \text{ is odd}\}$
$Q_{\mathcal{I},\exists} \leftarrow Q^{\mathcal{A}} \times N \sqcup \{\bot, \top\}$
$Q_{\mathcal{I},\forall} \leftarrow \Delta^{\mathcal{A}} \times N$
$\Delta_{\mathcal{I}} \leftarrow \{(\bot, a, \epsilon, \bot), (\top, a, \epsilon, \top) \mid a \in A^{\mathcal{A}}\}$
$q_{\mathrm{I}}^{\mathcal{R}} \leftarrow (q_{\mathrm{I}}^{\mathcal{A}}, \star)$
$(\mathcal{D}_\delta)_{\delta \in \Delta} \leftarrow$ 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
    $\Delta_{\mathcal{I}} \leftarrow \Delta_{\mathcal{I}} \cup \left\{\left((q, n), a, \epsilon, (\delta, n)\right) \mid \delta \in \Delta^{\mathcal{A}}_{q,a}\right\}$
foreach $a \in A$, $\delta = (q, a, q_{\mathtt{L}}, q_{\mathtt{R}}) \in \Delta^{\mathcal{A}}$, $n \in N$ **do**
  $\Delta_{\mathcal{I}} \leftarrow \Delta_{\mathcal{I}} \cup \{(\delta, a, \epsilon, q_{\mathrm{I}}^{\mathcal{C}_\delta})\}$ /* such a transition is a *challenge* */
  if $n \neq \star$ **then**
    $\Delta_{\mathcal{I}} \leftarrow \Delta_{\mathcal{I}} \cup \{(\delta, a, d, (q_d, n)) \mid d \in \{\mathtt{L}, \mathtt{R}\}\}$
  else
    $\Delta_{\mathcal{I}} \leftarrow \Delta_{\mathcal{I}} \cup \{(\delta, a, d, (q_d, n')) \mid d \in \{\mathtt{L}, \mathtt{R}\}, n' \in N\}$
$Q_{\exists}^{\mathcal{R}} \leftarrow Q_{\mathcal{I},\exists} \sqcup \bigsqcup_{\delta \in \Delta^{\mathcal{A}}} Q_{\exists}^{\mathcal{C}_\delta}$
$Q_{\forall}^{\mathcal{R}} \leftarrow Q_{\mathcal{I},\forall} \sqcup \bigsqcup_{\delta \in \Delta^{\mathcal{A}}} Q_{\forall}^{\mathcal{C}_\delta}$
$\Delta^{\mathcal{R}} \leftarrow \Delta_{\mathcal{I}} \sqcup \bigsqcup_{\delta \in \Delta^{\mathcal{A}}} \Delta^{\mathcal{C}_\delta}$
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
      $\Omega^{\mathcal{R}}(\delta, n) = 0$

– $\exists$ 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 4 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.

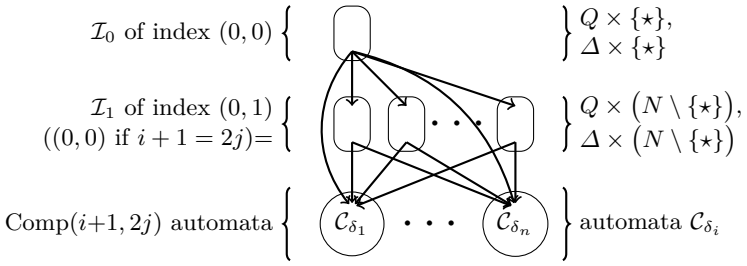

Fig. 4. The structure of the automaton $\mathcal{R}$.

Lemma 3. *If $\mathcal{A}$ is an unambiguous automaton of index $(i, 2j)$ then $\mathcal{R}$ is a* Comp$(i{+}1, 2j)$ *automaton.*

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 $\mathcal{A}$. Let $\mathcal{A}$ be an unambiguous automaton of index $(i, 2j)$.

Definition 1. *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*[1]

$$\delta = \big(\rho_t(u), t(u), \rho_t(u\mathtt{L}), \rho_t(u\mathtt{R})\big); \qquad (\diamond)$$

[1] By Lemma 2 there is at most one such δ.

– *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 4. $t \in \mathcal{L}(\mathcal{A})$ *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 5. *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 4 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 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 $k \stackrel{\text{def}}{=} \limsup_{i\to\infty} \Omega^{\mathcal{A}}(\rho_t(\alpha{\restriction_i}))$ 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 6. *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 1.

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 $\alpha \in \{\mathtt{L},\mathtt{R}\}^*$ and α is a leaf of ρ_t or α is an infinite branch such that $k \stackrel{\text{def}}{=} \limsup_{i\to\infty} \Omega^{\mathcal{A}}(\rho_t(\alpha\restriction_i))$ 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 $u \in \{\mathtt{L},\mathtt{R}\}^*$ he chooses the direction d in such a way that $ud \preceq \alpha$.

As in the proof of Lemma 5, 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. □

6 Conclusion

We presented a new algorithm Transformation which for a given unambiguous automaton $\mathcal{A}$ of index $(i, 2j)$ outputs an automaton Transformation($\mathcal{A}$) which accepts the same language and belongs to the class $\mathrm{Comp}(i+1, 2j)$. In particular, if $\mathcal{A}$ is an unambiguous Büchi automaton, then Transformation($\mathcal{A}$) is a weak alternating automaton. This can be considered an automata-theoretic counterpart of the Lusin-Souslin Theorem [9, Theorem 15.1].

6.1 Further work

This paper is a part of a broader project intended to understand better the descriptive complexity of unambiguous languages of infinite trees. In our view the crucial question is whether unambiguous automata can reach arbitrarily high levels in the alternating index hierarchy.

Conjecture. There exists a pair (i, j) such that if $\mathcal{A}$ is an unambiguous automaton on infinite trees then the language recognised by $\mathcal{A}$ can be recognised by an alternating automaton of index (i, j).

References

1. André Arnold. The mu-calculus alternation-depth hierarchy is strict on binary trees. *ITA*, 33(4/5):329–340, 1999.
2. André Arnold and Luigi Santocanale. Ambiguous classes in μ-calculi hierarchies. *TCS*, 333(1–2):265–296, 2005.
3. Julian Bradfield. Simplifying the modal mu-calculus alternation hierarchy. In *STACS*, pages 39–49, 1998.
4. Thomas Colcombet. Forms of determinism for automata (invited talk). In *STACS*, pages 1–23, 2012.
5. Thomas Colcombet, Denis Kuperberg, Christof Löding, and Michael Vanden Boom. Deciding the weak definability of Büchi definable tree languages. In *CSL*, pages 215–230, 2013.
6. Alessandro Facchini, Filip Murlak, and Michał Skrzypczak. Rabin-Mostowski index problem: A step beyond deterministic automata. In *LICS*, pages 499–508, 2013.
7. Olivier Finkel and Pierre Simonnet. On recognizable tree languages beyond the Borel hierarchy. *Fundam. Inform.*, 95(2–3):287–303, 2009.
8. Szczepan Hummel. Unambiguous tree languages are topologically harder than deterministic ones. In *GandALF*, pages 247–260, 2012.
9. Alexander Kechris. *Classical descriptive set theory.* Springer-Verlag, New York, 1995.
10. Orna Kupferman and Moshe Y. Vardi. The weakness of self-complementation. In *STACS*, pages 455–466, 1999.
11. Filip Murlak. The Wadge hierarchy of deterministic tree languages. *Logical Methods in Computer Science*, 4(4), 2008.
12. Damian Niwiński and Igor Walukiewicz. Ambiguity problem for automata on infinite trees. unpublished, 1996.
13. Damian Niwiński and Igor Walukiewicz. Relating hierarchies of word and tree automata. In *STACS*, pages 320–331, 1998.
14. Damian Niwiński and Igor Walukiewicz. A gap property of deterministic tree languages. *Theor. Comput. Sci.*, 1(303):215–231, 2003.
15. Damian Niwiński and Igor Walukiewicz. Deciding nondeterministic hierarchy of deterministic tree automata. *Electr. Notes Theor. Comput. Sci.*, 123:195–208, 2005.
16. Michael Oser Rabin. Weakly definable relations and special automata. In *Proceedings of the Symposium on Mathematical Logic and Foundations of Set Theory*, pages 1–23. North-Holland, 1970.
17. Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455. Springer, 1996.