AMBIGUITY HIERARCHY OF REGULAR INFINITE TREE LANGUAGES

ALEXANDER RABINOVICH AND DORON TIFERET

Tel Aviv University, Israel
e-mail address: rabinoa@tauex.tau.ac.il, sdoron5.t2@gmail.com
URL: https://www.cs.tau.ac.il/~rabinoa

ABSTRACT. An automaton is unambiguous if for every input it has at most one accepting computation. An automaton is $k$-ambiguous (for $k > 0$) if for every input it has at most $k$ accepting computations. An automaton is boundedly ambiguous if it is $k$-ambiguous for some $k \in \mathbb{N}$. An automaton is finitely (respectively, countably) ambiguous if for every input it has at most finitely (respectively, countably) many accepting computations.

The degree of ambiguity of a regular language is defined in a natural way. A language is $k$-ambiguous (respectively, boundedly, finitely, countably ambiguous) if it is accepted by a $k$-ambiguous (respectively, boundedly, finitely, countably ambiguous) automaton. Over finite words every regular language is accepted by a deterministic automaton. Over finite trees every regular language is accepted by an unambiguous automaton. Over $\omega$-words every regular language is accepted by an unambiguous Büchi automaton and by a deterministic parity automaton. Over infinite trees Carayol et al. showed that there are ambiguous languages.

We show that over infinite trees there is a hierarchy of degrees of ambiguity: For every $k > 1$ there are $k$-ambiguous languages that are not $k-1$ ambiguous; and there are finitely (respectively countably, uncountably) ambiguous languages that are not boundedly (respectively finitely, countably) ambiguous.

1. INTRODUCTION

Degrees of Ambiguity. The relationship between deterministic and nondeterministic machines plays a central role in computer science. An important topic is a comparison of expressiveness, succinctness and complexity of deterministic and nondeterministic models. Various restricted forms of nondeterminism were suggested and investigated (see [Col15, HSS17] for recent surveys).

Probably, the oldest restricted form of nondeterminism is unambiguity. An automaton is unambiguous if for every input there is at most one accepting run. For automata over finite words there is a rich and well-developed theory on the relationship between deterministic, unambiguous and nondeterministic automata [HSS17]. All three models have the same expressive power. Unambiguous automata are exponentially more succinct than deterministic
ones, and nondeterministic automata are exponentially more succinct than unambiguous ones [Lei81, Leu05].

Some problems are easier for unambiguous than for nondeterministic automata. As shown by Stearns and Hunt [SHI85], the equivalence and inclusion problems for unambiguous automata are in polynomial time, while these problems are PSPACE-complete for nondeterministic automata.

The complexity of basic regular operations on languages represented by unambiguous finite automata was investigated in [JJŠ16], and tight upper bounds on state complexity of intersection, concatenation and many other operations on languages represented by unambiguous automata were established.

It is well-known that the tight bound on the state complexity of the complementation of nondeterministic automata is $2^n$. In [JJŠ16], it was shown that the complement of the language accepted by an $n$-state unambiguous automaton is accepted by an unambiguous automaton with $2^{0.79n+\log n}$ states.

Many other notions of ambiguity were suggested and investigated. A recent paper [HSS17] surveys works on the degree of ambiguity and on various nondeterminism measures for finite automata on words.

An automaton is $k$-ambiguous if on every input it has at most $k$ accepting runs; it is boundedly ambiguous if it is $k$-ambiguous for some $k$; it is finitely ambiguous if on every input it has finitely many accepting runs.

It is clear that an unambiguous automaton is $k$-ambiguous for every $k > 0$, and a $k$-ambiguous automaton is finitely ambiguous. The reverse implications fail. For $\epsilon$-free automata over words (and over finite trees), on every input there are at most finitely many accepting runs. Hence, every $\epsilon$-free automaton on finite words and on finite trees is finitely ambiguous. However, over $\omega$-words there are nondeterministic automata with uncountably many accepting runs. Over $\omega$-words and over infinite trees, finitely ambiguous automata are a proper subclass of the class of countably ambiguous automata, which is a proper subclass of nondeterministic automata.

The cardinality of the set of accepting computations of an automaton over an infinite tree $t$ is bounded by the cardinality of the set of functions from the nodes of $t$ to the state of the automaton, and therefore, it is at most continuum $2^{\aleph_0}$. The set of accepting computations on $t$ is definable in Monadic Second-Order Logic (MSO). In Bárány et al. in [BKR10] it was shown that the continuum hypothesis holds for MSO-definable families of sets. Therefore, if the set of accepting computations of an automaton on a tree $t$ is uncountable, then its cardinality is $2^{\aleph_0}$. Hence, there are exactly two infinite degrees of ambiguity.

The degree of ambiguity of a regular language is defined in a natural way. A language is $k$-ambiguous (respectively, boundedly, finitely, countably ambiguous) if it is accepted by a $k$-ambiguous (respectively, boundedly, finitely, countably ambiguous) automaton.

Over finite words, every regular language is accepted by a deterministic automaton. Over finite trees, every regular language is accepted by a deterministic bottom-up tree automaton and by an unambiguous top-down tree automaton. Over $\omega$-words every regular language is accepted by an unambiguous Büchi automaton [Arn83] and by a deterministic parity automaton.

Hence, the regular languages over finite words, over finite trees and over $\omega$-words are unambiguous.

In [CLNW10] it was shown that the aforementioned situation is different for infinite trees. Carayol et al. [CLNW10] proved that the language $L_{\exists a}$ of infinite full-binary trees over
the alphabet \{a, c\}, defined as \( L_{3a} := \{ t \mid t \text{ has at least one node labeled by } a \} \) is ambiguous. The proof is based on the undefinability of a choice function in Monadic Second-Order logic (MSO) [GS83, CL07].

Our results imply that the complement of every countable regular language is not finitely ambiguous. Since \( L_{3a} \) is the complement (with respect to the alphabet \{a, c\}) of the language that consists of a single tree (i.e. the tree with all nodes labeled by c), we conclude that \( L_{3a} \) is not finitely ambiguous (this strengthens the above mentioned result of [CLNW10]). Our main result states that over infinite trees there is a hierarchy of degrees of ambiguity:

**Theorem 1.1** (Hierarchy).

1. For every \( k > 1 \) there are \( k \)-ambiguous languages that are not \((k-1)\)-ambiguous.
2. There are finitely ambiguous languages that are not boundedly ambiguous.
3. There are countably ambiguous languages that are not finitely ambiguous.
4. There are uncountably ambiguous languages that are not countably ambiguous.

Some natural tree languages that witness items (1), (3) and (4) of Theorem 1.1 are described in the examples below. We have not found a “natural” finitely ambiguous language which is not boundedly ambiguous (Theorem 1.1(2)).

**Examples 1.2.** Let \( T^\omega_\Sigma \) be the set of all infinite full-binary trees over an alphabet \( \Sigma \). Let \( \Sigma_k = \{c, a_1, a_2, ..., a_k\} \), and let \( L_{\neg a_i} := \{ t \in T^\omega_\Sigma_k \mid \text{no node in } t \text{ is labeled by } a_i \} \) for \( 1 \leq i \leq n \).

Define:

1. \( L_{\neg a_1 \vee \cdots \vee \neg a_k} := L_{\neg a_1} \cup \cdots \cup L_{\neg a_k} \). We show that this language is \( k \)-ambiguous, but is not \((k-1)\)-ambiguous (see Section 5). In [BS13] it was shown that \( L_{\neg a_1 \vee \neg a_2} \) is two ambiguous.
2. \( L_{3a_1} := \{ t \in T^\omega_\Sigma_1 \mid \text{there exists an } a_1\text{-labeled node in } t \} \). This is a countably ambiguous language that is not finitely ambiguous (see Section 4).
3. \( L_{\no\text{-max} \neg a_1} := \{ t \in T^\omega_\Sigma_1 \mid \text{above every } a_1\text{-labeled node in } t \text{ there is an } a_1\text{-labeled node} \} \). This is an uncountably ambiguous language that is not countably ambiguous (see Section 7).

**Organization of the Paper:** In Section 2 we recall notations and basic results about automata and monadic second-order logic. In Section 3 simple properties of languages are proved. Section 4 gives a sufficient condition for a language to be not finitely ambiguous. The proof techniques used in Section 4 refine the proof techniques of [CLNW10] – we rely on the fact that a choice function is not MSO-definable to obtain a lower bound for degree of ambiguity that is higher than the bound which was presented in [CLNW10]. Section 5 deals with \( k \)-ambiguous languages – for every \( k \in \mathbb{N} \), we describe a \( k \)-ambiguous language that is not \((k-1)\)-ambiguous. Section 6 provides an example of a finitely ambiguous language which is not boundedly ambiguous. Section 7 introduces a scheme for obtaining uncountably ambiguous languages from languages that are not boundedly ambiguous, and presents some natural examples of uncountably ambiguous languages. In Section 8, relying on the characterization of countable regular languages given by Niwiński [Niw91], we prove that every countable tree language is unambiguous. The conclusion is given in Section 9.

An extended abstract of this paper was published in [RT20]. Here we added the proofs that were sketched or missing in [RT20], presented natural examples of uncountably ambiguous languages (in Section 7), and added Section 8 in which we prove that countable languages are unambiguous using Niwiński’s Representation for Countable Languages.
2. Preliminary

We recall here standard terminology and notations about trees, automata and logic [PP04, Rab69]. In Subsection 2.3 we also recall Gurevich-Shelah’s theorem about undefinability of choice function and derive Lemma 2.5 which plays an important role in our proofs.

2.1. Trees. We view the set \( \{l, r\}^* \) of finite words over alphabet \( \{l, r\} \) as the domain of a full-binary tree, where the empty word \( \epsilon \) is the root of the tree, and for each node \( v \in \{l, r\}^* \), we call \( v \cdot l \) the left child of \( v \), and \( v \cdot r \) the right child of \( v \).

We define a tree order "\( \leq \)" as a partial order such that \( \forall u, v \in \{l, r\}^* : u \leq v \) if and only if \( u \) is a prefix of \( v \). Nodes \( u \) and \( v \) are incomparable — denoted by \( u \perp v \) — if neither \( u \leq v \) nor \( v \leq u \); a set \( U \) of nodes is an antichain, if its elements are incomparable with each other.

We say that an infinite sequence \( \pi = v_0, v_1, \ldots \) is a tree branch if \( v_0 = \epsilon \) and \( \forall i \in \mathbb{N} : v_{i+1} = v_i \cdot l \) or \( v_{i+1} = v_i \cdot r \).

If \( \Sigma \) is a finite alphabet, then a \( \Sigma \)-labeled full-binary tree \( t \) is a labeling function \( t : \{l, r\}^* \to \Sigma \). We denote by \( T_\Sigma^\omega \) the set of all \( \Sigma \)-labeled full-binary trees. We often use “tree” for “labeled full-binary tree.”

Given a \( \Sigma \)-labeled tree \( t \) and a node \( v \in \{l, r\}^* \), the tree \( t_{\geq v} \) (called the subtree of \( t \), rooted at \( v \)) is defined by \( t_{\geq v}(u) := t(v \cdot u) \) for each \( u \in \{l, r\}^* \).

Grafting. Given two labeled trees \( t_1 \) and \( t_2 \) and a node \( v \in \{l, r\}^* \), the grafting of \( t_2 \) on \( v \) in \( t_1 \), denoted by \( t_1[t_2/v] \), is the tree \( t \) that is obtained from \( t_1 \) by replacing the subtree of \( t_1 \) rooted at \( v \) by \( t_2 \). Formally, \( t(u) := \begin{cases} t_2(w) & \exists w \in \{l, r\}^* : u = v \cdot w \\ t_1(u) & \text{otherwise} \end{cases} \)

More generally, given a tree \( t_1 \), an antichain \( Y \subseteq \{l, r\}^* \) and a tree \( t_2 \), the grafting of \( t_2 \) on \( Y \) in \( t_1 \), denoted by \( t_1[t_2/Y] \), is obtained by replacing each subtree of \( t_1 \) rooted at a node \( y \in Y \) by the tree \( t_2 \).

Tree Language. A language \( L \) over an alphabet \( \Sigma \) is a set of \( \Sigma \)-labeled trees. We denote by \( \overline{L} := T_\Sigma^\omega \setminus L \) the complement of \( L \).

2.2. Automata. We define the following notions of automata:

Parity \( \omega \)-word Automata (PWA). A PWA is a tuple \( (Q_A, \Sigma, Q_I, \delta, \mathbb{C}) \) where \( \Sigma \) is a finite alphabet, \( Q \) is a finite set of states, \( Q_I \subseteq Q \) is a set of initial states, \( \delta \subseteq Q \times \Sigma \times Q \) is a transition relation, and \( \mathbb{C} : Q \to \mathbb{N} \) is a coloring function. A run of \( \mathcal{A} \) on an \( \omega \)-word \( y = a_0a_1 \ldots \) is an infinite sequence \( \rho = q_0q_1 \ldots \) such that \( q_0 \in Q_I \), and \( (q_i, a_i, q_{i+1}) \in \delta \) for all \( i \in \mathbb{N} \). We say that \( \rho \) is accepting if the maximal number that occurs infinitely often in \( \mathbb{C}(q_0)\mathbb{C}(q_1) \ldots \) is even.

Language. We denote the set of all accepting runs of \( \mathcal{A} \) on \( y \) by \( \text{ACC}(\mathcal{A}, y) \). The language of \( \mathcal{A} \) is defined as \( L(\mathcal{A}) := \{ y \in \Sigma^\omega \mid \text{ACC}(\mathcal{A}, y) \neq \emptyset \} \).
Parity Tree Automata (PTA) on Infinite Trees. A PTA is a tuple \((Q_A, \Sigma, Q_I, \delta, C)\) where \(\delta \subseteq Q \times \Sigma \times Q \times Q\) and \(\Sigma, Q, Q_I, F\) are defined as in PWA. A computation of \(A\) on a tree \(t\) is a function \(\phi : \{l, r\}^* \rightarrow Q\) such that \(\phi(\epsilon) \in Q_I\), and \(\forall v \in \{l, r\}^* : (\phi(v), t(v), \phi(v \cdot l), \phi(v \cdot r)) \in \delta\). We say that \(\phi\) is accepting if for each tree branch \(\pi = v_0v_1\ldots\), the maximal number that occurs infinitely often in \(\mathbb{C}(\phi(v_0))\mathbb{C}(\phi(v_1))\ldots\) is even.

Given a PTA \(A = (Q_A, \Sigma, Q_I, \delta_A, C_A)\) and a set \(Q' \subseteq Q_A\), we define \(A_{Q'} := (Q_A, \Sigma, Q', \delta_A, C_A)\) as the automaton obtained from \(A\) by replacing the set of initial states \(Q_I\) with \(Q'\). For a singleton \(Q' = \{q\}\), we simplify this notation by \(A_q := A_{Q'}\).

Language. We denote the set of all accepting computations of \(A\) on \(t\) by \(ACC(A, t)\). The language of \(A\) is defined as \(L(A) := \{t \in T_\Sigma^\omega \mid ACC(A, t) \neq \emptyset\}\). A tree language is said to be regular if it is accepted by a PTA.

A state \(q \in Q\) of a PTA \(A\) is called useful if there is a tree \(t \in L(A)\), a computation \(\phi \in ACC(A, t)\) and a node \(v \in \{l, r\}^*\) such that \(\phi(v) = q\). Throughout the paper we will assume that all states of PTA are useful.

Degree of Ambiguity of an Automaton. We denote by \(|X|\) the cardinality of a set \(X\). An automaton \(A\) is \(k\)-ambiguous if \(|ACC(A, t)| \leq k\) for all \(t \in L(A)\); \(A\) is unambiguous if it is 1-ambiguous; \(A\) is boundedly ambiguous if there is a \(k \in \mathbb{N}\) such that \(A\) is \(k\)-ambiguous; \(A\) is finitely ambiguous if \(ACC(A, t)\) is finite for all \(t\); \(A\) is countably ambiguous if \(ACC(A, t)\) is countable for all \(t\).

The degree of ambiguity of \(A\) (notation \(da(A)\)) is defined by \(da(A) := k\) if \(A\) is \(k\)-ambiguous and either \(k = 1\) or \(A\) is not \((k-1)\)-ambiguous, \(da(A) := finite\) if \(A\) is finitely ambiguous and not boundedly ambiguous, \(da(A) := \aleph_0\) if \(A\) is countably ambiguous and not finitely ambiguous, and \(da(A) := 2^\aleph_0\) if \(A\) is not countably ambiguous.

We order the degrees of ambiguity in a natural way: \(i < j < finite < \aleph_0 < 2^\aleph_0\), for \(i < j \in \mathbb{N}\).

Degree of Ambiguity of a Language. We say that a regular tree language \(L\) is unambiguous (respectively, \(k\)-ambiguous, finitely ambiguous, countably ambiguous) if it is accepted by an unambiguous (respectively, \(k\)-ambiguous, finitely ambiguous, countably ambiguous) automaton. We define \(da(L) := min_A(da(A) \mid L(A) = L)\).

2.3. Monadic Second-Order Logic. We use standard notations and terminology about monadic second-order logic (MSO) [Rab69, TB73, Tho90].

Let \(\tau\) be a relational signature. A structure (for \(\tau\)) is a tuple \(M = (D, \{R^M \mid R \in \tau\})\) where \(D\) is a domain, and each symbol \(R \in \tau\) is interpreted as a relation \(R^M\) on \(D\).

MSO-formulas use first-order variables, which are interpreted by elements of the structure, and monadic second-order variables, which are interpreted as sets of elements. Atomic MSO-formulas are of the following form:

- \(R(x_1, \ldots, x_n)\) for an \(n\)-ary relational symbol \(R\) and first order variables \(x_1, \ldots, x_n\)
- \(x = y\) for two first-order variables \(x\) and \(y\)
- \(x \in X\) for a first-order variable \(x\) and a second-order variable \(X\)

MSO-formulas are constructed from the atomic formulas, using boolean connectives, the first-order quantifiers, and the second-order quantifiers.

We write \(\psi(X_1, \ldots, X_n, x_1, \ldots, x_m)\) to indicate that the free variables of the formula \(\psi\) are \(X_1, \ldots, X_n\) (second order variables) and \(x_1, \ldots, x_m\) (first order variables). We write
M |= ψ(A₁, ..., Aₙ, a₁, ..., aₘ) if ψ holds in M when subsets Aᵢ are assigned to Xᵢ for i = 1, ..., n and elements aᵢ are assigned to variables x₁, ..., xₘ for i = 1, ..., m.

**Coding.** Let Δ be a finite set. We can code a function from a set D to Δ by a tuple of unary predicates on D. This type of coding is standard, and we shall use explicit variables that range over such mappings and expressions of the form “F(u) = d” (for d ∈ Δ) in MSO-formulas, rather than their codings.

Formally, for each finite set Δ we have second-order variables X₁^Δ, X₂^Δ, ... that range over the functions from D to Δ, and atomic formulas Xᵢ^Δ(u) = d for d ∈ Δ and u a first order variables [TB73]. Often the type of the second order variables will be clear from the context and we drop the superscript Δ.

**Definable Relations.** The powerset of D is denoted by ℙ(D). We say that a relation R ⊆ ℙ(D)^n × D^m is MSO-definable in a structure S with universe D if there is an MSO-formula ψ(X₁, ..., Xₙ, x₁, ..., xₘ) such that R = {(D₁, ..., Dₙ, u₁, ..., uₘ) ∈ ℙ(D)^n × D^m | S |= ψ(D₁, ..., Dₙ, u₁, ..., uₘ)}.

An element d ∈ D is MSO-definable in a structure S if there is a formula ψ(x) such that S |= ϕ(u) iff u = d. A set U ⊆ D is MSO-definable if there is a formula ϕ(X) such that S |= ϕ(V) iff V = U. A function is MSO-definable if its graph is.

The unlabeled binary tree is the structure ((l, r)^*, {E_l, E_r}) where E_l and E_r are binary symbols, respectively interpreted as {{v, v · l) | v ∈ {l, r}^*}} and {{v, v · r) | v ∈ {l, r}^*}}.

It is easy to verify the correctness of the following lemma:

**Lemma 2.1.** The following relations are MSO-definable in the unlabeled full-binary tree.
- The ancestor relation ≤.
- “A set of nodes is a branch,” “A set of nodes is an antichain.”
- Let A = (Q, Σ, Q_I, δ, C) be a PTA. We use ϕ for a function {l, r}^* → Q and σ for a function {l, r}^* → Σ.
  - “ϕ is a computation of A on the tree σ.”
  - “ϕ is an accepting computation of A on the tree σ.”

The following two fundamental theorems were proved by Rabin in his famous 1969 paper [Rab69].

**Theorem 2.2** (Rabin [Rab69]). A tree language is regular iff it is MSO-definable in the unlabeled binary tree structure.

A labeled tree is regular iff it has finitely many different subtrees. An equivalent definition is: a tree is regular iff its labeling is MSO-definable [Rab69]. Hence, for every regular Σ-labeled tree t₀ there is an MSO-formula ψ_{t₀}(σ^Σ), where σ^Σ is the coding of {l, r}^* → Σ, that is satisfied by t : {l, r}^* → Σ iff t = t₀.

**Theorem 2.3** (Rabin’s basis theorem [Rab69]). Any non-empty regular tree language contains a regular tree.

**Choice Function.** A choice function is a mapping that assigns to each non-empty set of nodes one element from the set.

**Theorem 2.4** (Gurevich and Shelah [GS83]). There is no MSO-definable choice function on the full-binary tree.
The following lemma follows from Theorem 2.4. It plays a key role in our proofs in Section 4, where sufficient conditions are provided for a language to be not finitely ambiguous.

**Lemma 2.5.** There is no MSO-definable function that assigns to every non-empty antichain \( Y \) a finite non-empty subset \( X \subseteq Y \).

**Proof.** Assume, for the sake of contradiction, that a function that returns a finite non-empty subset for each non-empty antichain is MSO-definable in the unlabeled full-binary tree, by an MSO-formula \( \text{Finite.AntichainSubset}(X,Y) \).

**Claim 2.5.1** (Choice function over finite sets). There is an MSO-definable function that assigns to each non-empty finite set \( X \subseteq \{l,r\}^* \) an element \( x \in X \).

**Proof.** We first define a lexicographic order \( \preceq_{\text{lex}} \) on \( \{l,r\}^* \), by \( u \preceq_{\text{lex}} v \) if \( u \) is a prefix of \( v \) or \( u = w \cdot l \cdot u' \) and \( v = w \cdot r \cdot v' \) for some \( w, u', v' \in \{l,r\}^* \).

It is easy to verify that \( \preceq_{\text{lex}} \) is MSO-definable in the unlabeled full-binary tree. \( \preceq_{\text{lex}} \) is a linear order, and therefore each non-empty finite set has exactly one \( \preceq_{\text{lex}} \)-minimal element. We conclude that a finite set choice function is definable by \( \text{FiniteChoice}(X,x) := \text{"x is the } \preceq_{\text{lex}} \text{-minimal element in } X\)".

Let \( \text{FiniteChoice}(X,x) \) be an MSO-formula that defines a function as in Claim 2.5.1. We will use formulas \( \text{Finite.AntichainSubset}(X,Y) \) and \( \text{FiniteChoice}(X,x) \) to define a choice function by an MSO-formula \( \text{Choice}(X,x) \) which is the conjunction of the following conditions:

1. \( \exists Z : \text{"Z is the set of } \preceq \text{-minimal elements in } X\)"
2. \( \exists Y : \text{Finite.AntichainSubset}(Z,Y) \)
3. \( \text{FiniteChoice}(Y,x) \)

For each non-empty set \( X \) there is a unique subset \( Z \subseteq X \) of the \( \preceq \)-minimal elements in \( X \). This set is a non-empty antichain, and therefore \( \text{Finite.AntichainSubset}(Z,Y) \) returns a finite subset \( Y \subseteq Z \). Therefore, \( \text{FiniteChoice}(Y,x) \) returns an element in \( Y \). We conclude that \( \text{Choice}(X,x) \) returns an element \( x \in X \) and therefore defines a choice function in the unlabeled full-binary tree, in contradiction to Theorem 2.4. \( \square \)

### 3. Simple Properties of Automata and Languages

In this section some simple lemmas are collected.

**Lemma 3.1.** Let \( A_1 = (Q_1, \Sigma_1, Q_1^1, \delta_1, C_1) \) and \( A_2 = (Q_2, \Sigma_2, Q_2^1, \delta_2, C_2) \) be two PTA. Then:

1. There exists an automaton \( B \) such that \( L(B) = L(A_1) \cup L(A_2) \) and for each \( t \in L(A_1) \cup L(A_2) \), \( |ACC(B,t)| \leq |ACC(A_1,t)| + |ACC(A_2,t)| \).
2. There exists an automaton \( B \) such that \( L(B) = L(A_1) \cap L(A_2) \) and for each \( t \in L(A_1) \cap L(A_2) \), \( |ACC(B,t)| \leq |ACC(A_1,t)| \cdot |ACC(A_2,t)| \).

**Proof.** (1) Assume that \( Q_1 \) and \( Q_2 \) are disjoint, and let \( B := (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, Q_1^1 \cup Q_2^1, \delta_1 \cup \delta_2, C_1 \cup C_2) \). It is clear that \( L(B) = L(A_1) \cup L(A_2) \).

Let \( t \in L(B) \). By definition of \( B \), for each \( \phi \in ACC(B,t) \) we either have \( \phi \in ACC(A_1,t) \) or \( \phi \in ACC(A_2,t) \). Therefore, we obtain \( |ACC(B,t)| = |ACC(A_1,t)| + |ACC(A_2,t)| \).

(2) It is easy to verify that there is an MSO-formula over \( \omega \)-words that holds for \( w = (c_1,c_1'), \ldots, (c_i,c_i') \) \( \cdots \in (Image(C_1) \times Image(C_2))^\omega \) iff the maximal color that appears
Then, there exists a tree $t$ countably ambiguous.

Let $\phi$ be a tree automaton. We often use implicitly the following simple Lemma.

**Lemma 3.3 (Grafting).** Let $\mathcal{A}$ be an automaton, $t$, $t_1$, $v \in \{l, r\}^*$ and $\phi \in ACC(\mathcal{A}, t)$, and $\phi_1 \in ACC(\mathcal{A}_q, t_1)$. If $\phi(v) = q$, then $\phi[\phi_1/v]$ is an accepting computation of $\mathcal{A}$ on $t[t_1/v]$.

A similar lemma holds for general grafting. As an immediate consequence, we obtain the following lemma:

**Lemma 3.4.** $da(\mathcal{A}) \geq da(\mathcal{A}_q)$ for every useful state $q$ of $\mathcal{A}$.

**Corollary 3.5.** Let $\mathcal{A}$ be a boundedly (respectively, finitely, countably) ambiguous PTA with a set $Q$ of useful states, and let $Q' \subseteq Q$. Then $\mathcal{A}_{Q'}$ is boundedly (respectively, finitely, countably) ambiguous.

**Lemma 3.6.** Let $L_1$ and $L_2$ be two tree languages such that $da(L_1) \neq da(L_2)$ and $L_1 \subseteq L_2$. Then, there exists a tree $t \in L_2 \setminus L_1$.

**Proof.** The lemma follows immediately, since otherwise we have $L_1 = L_2$ and therefore $da(L_1) = da(L_2)$, in contradiction to $da(L_1) \neq da(L_2)$. \qed
Lemma 3.7. Let $A = (Q, \Sigma, Q_I, \delta, \emptyset, \emptyset)$ be a PTA. Then, there exists a PTA $B = (Q_B, \Sigma, \{q_I^B\}, \delta_B, \emptyset)$ with single initial state such that $L(B) = L(A)$, and $da(B) \leq da(A)$.

Proof. Let $Q_B := Q \cup \{q_I^B\}$ and $\delta_B := \delta_A \cup \{(q_I^B, a, q_I, q_r) \mid q_I \in Q_I$ and $(q_I, a, q_r, q_r) \in \delta\}$. It is easy to see that $L(B) = L(A)$.

Let $t \in L(A)$, and let $g_t$ be a function from $ACC(A, t)$ to $ACC(B, t)$ that maps each computation $\phi \in ACC(A, t)$ to a computation $\phi'$ that assigns $q_I^B$ to node $\epsilon$, and $\phi(v)$ to other nodes. It is easy to see that $\phi' \in ACC(B, t)$, and that $g_t$ is surjective, and therefore $\forall t : |ACC(A, t)| \geq |ACC(B, t)|$, as requested.

Definition 3.8 (Moore machine). A Moore machine is a tuple $M = (\Sigma, \Gamma, Q, q_1, \delta, out)$, where $\Sigma$ is a finite input alphabet, $Q$ is a finite set of states, $q_I \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q$ is a transition function, $\Gamma$ is an output alphabet, and $out : Q \to \Gamma$ is an output function.

Define $\hat{\delta} : \Sigma^* \to Q$ by $\hat{\delta}(\epsilon) := q_I$ and $\hat{\delta}(w) := \delta(\hat{\delta}(w'), a)$ for $w = w' \cdot a$ where $w' \in \Sigma^*$ and $a \in \Sigma$. We say that a function $F : \Sigma^* \to \Gamma$ is definable by a Moore machine if there is a Moore machine $M$ such that $F(w) = out(\hat{\delta}(w))$ for all $w \in \Sigma^*$.

Definition 3.9. Let $F : \Sigma_1^* \to \Sigma_2$ be a function definable by a Moore machine, and let $t_1 \in T_{\Sigma_1}$. We define $t_2 := \hat{F}(t_1)$ as a tree in $T_{\Sigma_2}$ such that $t_2(v) := F(t_1(v_1) \cdots t_1(v_k))$ where $v_1, v_2, \ldots, v_k$ is the path from the root to $v$.

For a tree language $L \subseteq T_{\Sigma_1}^\omega$, we define $\hat{F}(L) := \{\hat{F}(t) \mid t \in L\} \subseteq T_{\Sigma_2}^\omega$.

Lemma 3.10 (Reduction). Let $L_1$ and $L_2$ be regular tree languages over alphabets $\Sigma_1$ and $\Sigma_2$, respectively. Let $F : \Sigma_1^* \to \Sigma_2$ be a function definable by a Moore machine. Assume that for each $t \in T_{\Sigma_1}^\omega$, $t \in L_1$ iff $\hat{F}(t) \in L_2$. Then $da(L_1) \leq da(L_2)$.

Proof. Let $A_2 = (Q_2, \Sigma_2, Q_I^2, \delta_2, \emptyset)$ such that $A_2$ accepts $L_2$ and $da(A_2) = da(L_2)$.

Let $M = (\Sigma_1, \Sigma_2, Q_M, q_I^M, \delta_M, out_M)$ be a Moore machine defining $F$. We will use $A_2$ and $M$ to define an automaton $A_1 = (Q_1, \Sigma_1, Q_I^1, \delta_1, C_1)$ such that $t \in L(A_1)$ iff $\hat{F}(t) \in L(A_2)$, by:

- $Q_1 := Q_2 \times Q_M$
- $Q_I^1 := Q_I^2 \times \{q_I^M\}$
- $((q, p), a, (q_1, p_1), (q_2, p_2)) \in \delta_1$ iff $p_1 = p_2 = \delta_M(p, a)$ and $(q, out_M(p), q_1, q_2) \in \delta_2$
- $C_1(q, p) := C_2(q)$

First notice that $\forall t : t \in L(A_1) \iff \hat{F}(t) \in L(A_2) \iff \hat{F}(t) \in L_2 \iff t \in L_1$, and therefore $L(A_1) = L_1$ as needed.

Let $\phi \in ACC(A_1)$, and define a computation $\phi'$ by $\phi'(v) = q_I$ for $\phi(v) = (q_1, q_2) \in Q_2 \times Q_M$. It is easy to see that $\phi' \in ACC(A_2, \hat{F}(t))$ and since $M$ is deterministic, we conclude that $|ACC(A_1, t)| \leq |ACC(A_2, \hat{F}(t))|$, and therefore $da(A_1) \leq da(A_2)$.

We conclude that $da(L_1) \leq da(A_1) \leq da(A_2) = da(L_2)$, as requested.

Fact 3.11. A tree $t$ is regular iff its labeling $t : \{l, r\}^* \to \Sigma$ is definable by a Moore machine.
4. Not-Finitely Ambiguous Languages

We provide here sufficient conditions for a language to be not finitely ambiguous. These conditions will allow us to present some natural languages which are countably ambiguous and not finitely ambiguous, proving Theorem 1.1(3). In addition, these results are used in Sects. 5-7 where it is proved that for every $k > 1$ there is a language of ambiguity degree equal to $k$ and there are languages with finite and uncountable degrees of ambiguity.

First, we state our main technical result – Proposition 4.1. Then, we derive some consequences. Finally, a proof of Proposition 4.1 is given. Our proof relies on the fact that there is no MSO-definable function that assigns to every non-empty antichain $Y$ a finite non-empty subset $X \subseteq Y$ (Lemma 2.5), and our proof techniques refine the proof techniques of [CLNW10].

Recall that for trees $t$ and $t'$ and an antichain $Y$, we denote by $t[t'/Y]$ the tree obtained from $t$ by grafting $t'$ at every node in $Y$.

**Proposition 4.1.** Let $t_0$ and $t_1$ be regular trees and $L$ be a regular language such that $t_0 \notin L$ and $t_0[t_1/Y] \in L$ for every non-empty antichain $Y$. Then $L$ is not finitely ambiguous.

**Definition 4.2.** For a tree language $L$ over alphabet $\Sigma$, we denote by $\text{Subtree}(L)$ the tree language $\{t \in T_\Sigma^\omega \mid \exists t' \in L \exists v : t'_{\geq v} = t\}$.

**Corollary 4.3.** Let $L$ be a non-empty regular language over an alphabet $\Sigma$ such that $\text{Subtree}(L) \neq T_\Sigma^\omega$. Then, the complement of $L$ is not finitely ambiguous.

**Proof.** Let $L$ be as in Corollary 4.3. We claim that there are regular $\Sigma$-labeled trees $t_0 \in L$ and $t_1 \notin \text{Subtree}(L)$. Indeed, by Rabin’s basis theorem there is a regular $t_0 \in L$. Since $L$ is regular, there is an automaton $B = (Q, \Sigma, \{a_1\}, \delta, C)$ (with only useful states) that accepts $L$. It is clear that $B_Q$ accepts $\text{Subtree}(L)$, and therefore $\text{Subtree}(L)$ is regular. The complement of $\text{Subtree}(L)$ is regular (as the complement of a regular language) and non-empty (since $\text{Subtree}(L) \neq T_\Sigma^\omega$), and therefore contains a regular tree $t_1$ (by Rabin’s basis theorem). Note that $t_0[t_1/Y] \notin L$ for every non-empty antichain $Y$.

The complement of $L$ satisfies the assumption of Proposition 4.1. Therefore, it is not finitely ambiguous.

**Corollary 4.4** (Not finitely ambiguous languages). The following languages are not finitely ambiguous:

1. The complement of a non-empty regular countable tree language.
2. The complement of a regular language that contains a single tree.
3. The language $L_{3a_1} := \{t \in T_\Sigma^\omega \mid t \text{ has at least one node labeled by } a_1\}$ over alphabet $\Sigma = \{a_1, \ldots, a_m, c\}$.

**Proof.** (1) Let $L$ be a non-empty regular countable tree language. Every tree has countably many subtrees, and since $L$ is countable we conclude that $\text{Subtree}(L)$ is countable. Therefore, $\text{Subtree}(L)$ does not contain all trees. By Corollary 4.3, we conclude that $\overline{L}$ is not finitely ambiguous.

(2) Follows immediately from (1).

(3) By the definition of $L_{3a_1}$ we have $L_{3a_1} \cap T_{\{c,a_1\}}^\omega = T_{\{c,a_1\}}^\omega \setminus \{t_c\}$, and therefore by (2), $L_{3a_1} \cap T_{\{c,a_1\}}^\omega$ is not finitely ambiguous. It is easy to see that $T_{\{c,a_1\}}^\omega$ is unambiguous (since there is a deterministic automaton that accepts it). Therefore, by Corollary 3.2 we conclude that $L_{3a_1}$ is not finitely ambiguous.
It is easy to prove that the complement of every finite language (i.e. a language which consists of finitely many trees) is countably ambiguous. Therefore, we obtain:

**Corollary 4.5.** If $L$ is regular and its complement is finite and non-empty, then $da(L) = \aleph_0$.

**Proof of Corollary 4.5.** We first prove the following claim:

**Claim 4.5.1.** Let $L$ be a regular tree language containing a single tree. Then $L$ is countably ambiguous.

**Proof.** Assume that $L = \{t\}$. $L$ is a regular language, and therefore $t$ is regular. By Fact 3.11 there is a Moore machine $M = (\{l, r\}, \Sigma, Q_M, q^M_0, \delta_M, out_M)$ such that for each $v \in \{l, r\}^*$, 

\[
\text{out}(\hat{\delta}(v)) = \sigma \iff t(v) = \sigma \quad \text{(that is, } M \text{ defines the function } t : \{l, r\}^* \to \Sigma).
\]

We will use $M$ to construct a countably ambiguous automaton $A$ that accepts $L$ by guessing a node $v \in \{l, r\}^*$ such that $t(v) \neq t'(v)$ for each tree $t' \in \overline{L}$.

Let $A := (Q_A, \Sigma, I, \delta, C)$ such that:

- $Q_A := \{q, q'\} \times Q_M$
- $Q_I := \{(q', q^M_i)\}$
- $\delta$ is defined by:
  - $((q, p), a, (q, p'), (q, p'')) \in \delta$ iff $\delta_M(p, l) = p', \delta_M(p, r) = p''$
  - $((q', p), a, (q', p'), (q', p'')) \in \delta$ iff $\delta_M(p, l) = p', \delta_M(p, r) = p''$ and $\text{out}(p) \neq a$
  - $((q', p), a, (q', p'), (q', p'')) \in \delta$ iff $\delta_M(p, l) = p', \delta_M(p, r) = p''$ and $\text{out}(p) = a.$
- $\forall p \in Q_M : C(q, p) := 0$ and $C(q', p) := 1$

By definition of $A$, it is clear that $t' \in L(A)$ iff there is a node $v$ such that $t'(v) \neq t(v)$, and therefore $t' \in L(A)$ iff $t' \neq t$.

For each computation $\phi$ of $A$ on $t'$, the $Q_M$ component is determined deterministically by $M$ and $t$. If $\phi$ is accepting, there are finitely many nodes $v$ such that the first component of $\phi(v)$ is $q'$ – otherwise, there would be a branch where the maximal color assigned infinitely often by $C$ is odd, in contradiction to $\phi$ being an accepting computation. Therefore, there are countably many accepting computations on each tree $t' \in L(A)$, and $A$ is countably ambiguous.

$L$ is finite and therefore there are $t_1, \ldots, t_k \in T_0^\omega$ such that $L = \{t_1, \ldots, t_k\}$. A finite tree language does not contain a non-regular tree, and therefore $t_1, \ldots, t_k$ are regular. By Claim 4.5.1, for each tree $t_i \in L$, there is a countably ambiguous automaton $A_i$ such that $t \in L(A_i)$ iff $t \neq t_i$. Notice that $L = L(A_1) \cap \ldots \cap L(A_k)$, and therefore by Lemma 3.1 we conclude that $L$ is countably ambiguous.

**On the Proof of Proposition 4.1.** In the rest of this section, Proposition 4.1 is proved. Let us sketch some ideas of the proof. For a language $L$, as in Proposition 4.1, and any non-empty antichain $Y$ we show that if $A$ does not accept $t_0$ and accepts $t := t_0[t_1/Y]$, then every $\phi \in ACC(A, t)$ chooses (in an MSO-definable way) an element from $Y$. Hence, the computations in $ACC(A, t)$ choose together a subset $X$ of $Y$ of cardinality $\leq |ACC(A, t)|$ (each computation chooses a single element). Therefore, if $A$ accepts $L$ and is finitely ambiguous, then $X$ is finite – a contradiction to Lemma 2.5. To implement this plan, in Subsection 4.1 we recall a game theoretical interpretation of “a tree is accepted by an automaton.” Then, in Subsection 4.2 we analyze which concepts related to these games are MSO-definable. Finally, in Subsection 4.3, the proof is completed.
4.1. **Membership Game.** Let $\mathcal{A} = (Q, \Sigma, \{q_I\}, \delta, \mathbb{C})$ be a PTA, and let $t$ be a $\Sigma$-labeled tree. A two-player game $G_{t,\mathcal{A}}$ (called a “membership game”) between Automaton and Pathfinder is defined as follows. The positions of Automaton are $\{l, r\}^* \times Q \times Q$, and the positions of Pathfinder are $\{l, r\}^* \times Q \times Q$. The initial position is $(e, q_I)$.

From a position $(v, q) \in \{l, r\}^* \times Q$ Automaton chooses a tuple $(q_l, q_r) \in Q \times Q$ such that $\exists a \in \Sigma : (q_l, a, q_r, q_r) \in \delta$, and moves to the position $(v, q_l, q_r)$. From a position $(v, q_l, q_r) \in \{l, r\}^* \times Q \times Q$ Pathfinder chooses a direction $d \in \{l, r\}$, and moves to the position $(v \cdot d, q_d)$.

We define a play $\pi := e_0, d_0, e_1, d_1, \ldots, e_i, d_i, \ldots \in (Q \times Q \times \{l, r\})^\omega$ as an infinite sequence of moves, corresponding to the choices of Automaton and Pathfinder from the initial position. We say that the move $e_i = (q_l, q_r)$ from position $(v, q)$ is invalid for Automaton if $(q, t(v), q_l, q_r) \notin \delta$.

A strategy for a player in $G_{t,\mathcal{A}}$ is a function that determines the next move of the player based on previous moves of both players.

A **positional strategy** for a player in $G_{t,\mathcal{A}}$ is a strategy that determines the next move of the player based only on the current position. A positional strategy for Automaton is a function $str : \{l, r\}^* \times Q \to Q \times Q$, and a positional strategy for Pathfinder is a function $STR : \{l, r\}^* \times Q \times Q \to \{l, r\}$.

Let $C_G$ be a coloring function that maps each position in $G_{t,\mathcal{A}}$ to a color in $\mathbb{N}$. We define $C_G(v, q) := C(q)$ for Automaton’s positions, and $C_G(v, q_l, q_r) := 0$ for Pathfinder’s positions.

For each play $\pi$ define $\pi_\pi$ as the infinite sequence of positions corresponding to the moves in $\pi$. A play $\pi$ is winning for Automaton iff $\pi$ does not contain an invalid move for Automaton, and the maximal color that $C_G$ assigns infinitely often to the positions in $\pi_\pi$ is even. Since all Pathfinder’s positions are colored by 0, it is sufficient to consider the coloring of Automaton’s positions in $\pi_\pi$.

We say that a play is consistent with a strategy of a player if all moves of the player are according to the strategy. A winning strategy for a player is a strategy such that each play that is consistent with the strategy is winning for the player.

Parity games are positionally determined [EJ91], i.e., for each parity game, one of the players has a positional winning strategy. Therefore, if a player has a winning strategy, then he has a positional winning strategy. Additionally, if a positional strategy of a player wins against all positional strategies of the other player, then it is a winning strategy.

We recall standard definitions and facts about the connections between games and tree automata [GH82, PP04].

Let $\phi : \{l, r\}^* \to Q$ be a function such that $\phi(e) = q_I$ and $\forall v \in \{l, r\} : \exists a \in \Sigma : (\phi(v), a, \phi(v \cdot l), \phi(v \cdot r)) \in \delta$. We define a positional strategy $str_\phi : \{l, r\}^* \times Q \to Q \times Q$ for Automaton, by $str_\phi(v, q) := (\phi(v \cdot l), \phi(v \cdot r))$. Conversely, for each positional strategy $str : \{l, r\}^* \times Q \to Q \times Q$ of Automaton we construct a function $\phi_str : \{l, r\}^* \to Q$ by $\phi(e) := q_I$ and for all $v \in \{l, r\}^*$ we set $\phi(v \cdot l) := q_l$, and $\phi(v \cdot r) := q_r$ where $str(v, \phi(v)) = (q_l, q_r)$.

**Claim 4.1.1.**

(1) Let $\pi$ be a play that is consistent with $str_\phi$, and let $(v_i, q_i)$ be the $i$-th position of Automaton in $\pi_\pi$. Then, $\phi(v_i) = q_i$.

(2) If $\phi \in ACC(\mathcal{A}, t)$, then $str_\phi$ is a positional winning strategy for Automaton.

(3) If $str$ is a positional winning strategy for Automaton, then $\phi_str \in ACC(\mathcal{A}, t)$. 
Proof. (1) We will prove by induction on \( i \). For \( i = 0 \) we have \((v_0, q_0) = (\epsilon, q_l)\) (by definition of \( G_{t,A} \)), and indeed \( \phi(v_0) = \phi(\epsilon) = q_l \). Assume the claim holds for \( i = k \) and we prove for \( i = k + 1 \).

Let \( d \in \{l, r\} \) be the \( i \)-th move of Pathfinder in \( \pi \). By definition of \( G_{t,A} \) we have \( v_{i+1} = v_i \cdot d \), and \( q_{i+1} = q_i \), where \( \text{str}_\phi(v_i, q_i) = (q_i, q_r) \).

By definition \( \text{str}_\phi \) we have \((q_i, q_r) = (\phi(v_i \cdot l), \phi(v_i \cdot l)) \), and therefore \( q_{i+1} = \phi(v_i \cdot d) = \phi(v_{i+1}) \), as requested.

(2) and (3) are well known results about membership games [PP04]. \( \blacksquare \)

The next claim describes what happens when Pathfinder plays his winning strategy in \( G_{t,A} \) against an Automaton’s winning strategy in \( G_{t',A} \) (for \( t' \neq t \)).

Claim 4.1.2. Assume \( t \notin L(A) \) and let \( \phi \) be an accepting computation of \( A \) on a tree \( t' \), and \( \text{STR} \) be a winning strategy of Pathfinder in \( G_{t,A} \). Let \( \pi := e_0, d_0, e_1, d_1, \ldots, e_i, d_i, \ldots \) be the play that is consistent with \( \text{str}_\phi \) and \( \text{STR} \). Then, there is \( i \in \mathbb{N} \) such that \( e_i \) is an invalid move for Automaton in \( G_{t,A} \). Moreover, if \( e_i \) is the first invalid move for Automaton in \( \pi \), then \( t(v) \neq t'(v) \) for \( v := d_0 \ldots d_{i-1} \).

Proof. Assume, for the sake of contradiction, that \( \pi \) does not contain an invalid move for Automaton, and let \((v_i, q_i)\) be the \( i \)-th position of Automaton in \( \pi \). By definition of \( G_{t,A} \) it is easy to see that \( \pi = v_0, \ldots, v_i, \ldots \) is a branch in the full-binary tree. Since \( \phi \) is an accepting computation of \( A \) on \( t' \), we conclude that the maximal color that \( C \) assigns infinitely often to states in \( \phi(\pi) \) is even. By Claim 4.1.1(1) we have \( \phi(v_i) = q_i \), and therefore \( \phi(\pi) = q_0 \ldots q_i \ldots \). By the definition of \( C_G \) we have \( C_G(v_i, q_i) = C(q_i) \) and we conclude that the maximal color that \( C \) assigns infinitely often in \( \pi \) is even, and therefore the play is winning for Automaton – a contradiction to \( \text{STR} \) being a winning strategy of Pathfinder.

Therefore, Automaton makes an invalid move in \( \pi \). Let \( e_i = (q_i, q_r) \) be the first invalid move of Automaton in \( \pi \). Since \( e_i \) is invalid we have \((q_i, t(v_i)), q_i, q_r) \notin \delta \), and by definition of \( \text{str}_\phi \) we obtain \((q_i, q_r) = (\phi(v_i \cdot l), \phi(v_i \cdot r)) \). Since \( \phi(v_i) = q_i \) we have \((\phi(v_i), t(v_i), \phi(v_i \cdot l), \phi(v_i \cdot r)) \notin \delta \). \( \phi \) is a computation of \( A \) on \( t' \) and therefore \((\phi(v_i), t'(v_i), \phi(v_i \cdot l), \phi(v_i \cdot r)) \in \delta \), and we conclude that \( t(v_i) \neq t'(v_i) \). Notice that by the definition of \( G_{t,A} \) we have \( v_i = d_0 \ldots d_{i-1} \), and the claim follows. \( \blacksquare \)

4.2. MSO-definability. Throughout this section we will use the following conventions and terminology.

Positional Pathfinder strategies as labeled trees: We will define a positional strategy \( \text{STR} \) for Pathfinder as a function in \( \{l, r\}^{*} \times Q \times Q \to \{l, r\} \). Hence, it can be considered as a \( Q \times Q \to \{l, r\} \) labeled tree. Below we will not distinguish between a positional Pathfinder’s strategy and the corresponding \( Q \times Q \to \{l, r\} \) labeled full-binary tree.

In particular, we call such a strategy regular, if the corresponding tree is regular.

MSO-definability: We will use “MSO-definable” for “MSO-definable in the unlabeled full-binary tree.”

The rest of the proof deals with MSO-definability. By Claim 4.1.2, there is a function \( \text{Invalid}_A(\phi, \text{STR}, t, v) \) that, for every accepting computation \( \phi \) of \( A \) on \( t' \), returns a node \( v \) such that \( t'(v) \neq t(v) \). This function depends on the strategy \( \text{STR} \) of Pathfinder. The restriction of \( \text{Invalid}_A \) to the Pathfinder positional winning strategies in \( G_{t,A} \) is MSO-definable (with parameters \( t \) and \( \text{STR} \)) by the following formula \( \text{Leads}_A(\phi, \text{STR}, t, v) \), that
describes in MSO the play of $\phi$ against $\text{STR}$ up to the first invalid move of Automaton (at the position $(v, \phi(v))$).

Define $\text{Leads}_A(\phi, \text{STR}, t, v)$ as the conjunction of:

1. $\phi(e) = q_t$ – the play starts from the initial position.
2. $\forall u < v : ((\phi(u), t(u), \phi(u \cdot l), \phi(u \cdot r)) \in \delta$ – all Automaton’s moves at the positions $(u, q)$, where $u$ is an ancestor of $v$ respect $\delta$. (By Claim 4.1.1(1), in any play consistent with $\phi$, Automaton can reach only the positions of the form $(u, \phi(u))$).
3. $(\phi(v), t(v), \phi(v \cdot 1), \phi(v \cdot r)) \notin \delta$ – the Automaton move at $(v, \phi(v))$ is invalid.
4. $\forall u < v : (\text{STR}(u, \phi(u \cdot l), \phi(u \cdot r)) = l \iff u \cdot l \leq v)$ – the Pathfinder moves $d_0 \ldots d_j$...

are consistent with $\text{STR}$ and are along the path from the root to $v$, i.e., $d_0 d_1 \ldots d_j \leq v$.

To sum up, we have the following claim:

**Claim 4.1.3.** $\text{Leads}_A(\phi, \text{STR}, t, v)$ defines a function that, for every tree $t \notin L(A)$, every Pathfinder’s positional (in $G_{t, A}$) winning strategy $\text{STR}$, and every $\phi \in \text{ACC}(A, t')$, returns a node $v$ such that $t(v) \neq t'(v)$.

Claim 4.1.3 plays a crucial role in our proof. It is instructive to compare it with Theorem 2.4 which implies that there is no MSO-definable function $F(t, D, v)$ that for a tree $t \neq t'$ and $D := \{u \mid t(u) \neq t'(u)\}$ returns a node $v$ such that $t(v) \neq t'(v)$.

The following claim is folklore. Due to the lack of references, it is proved in the Appendix.

**Claim 4.1.4.** Let $t_0$ be a regular tree such that $t_0 \notin L(A)$. Then, Pathfinder has a regular positional winning strategy in $G_{t_0, A}$.

Let $t_0$ be a regular tree such that $t_0 \notin L(A)$. By Claim 4.1.4 there is a regular positional winning strategy $\widehat{\text{STR}}$ of Pathfinder in $G_{t_0, A}$. Now, we can substitute $\widehat{\text{STR}}$ and $t_0$ for arguments $\text{STR}$ and $t$ of $\text{Leads}_A$ and obtain the following Proposition:

**Proposition 4.6.** For every regular tree $t_0 \notin L(A)$ and a regular positional winning strategy $\widehat{\text{STR}}$ for Pathfinder in $G_{t_0, A}$, there is an MSO-definable function that, for each accepting computation $\phi$ of $A$ on $t'$, returns a node $v$ such that $t_0(v) \neq t'(v)$.

Proof. Let $\psi_{t_0}(\sigma)$ and $\psi_{\widehat{\text{STR}}}(\text{STR})$ be MSO-formulas that define $t_0$ and $\widehat{\text{STR}}$. Then, by Claim 4.1.3, $\exists \sigma \exists \text{STR} : \psi_{t_0}(\sigma) \land \psi_{\widehat{\text{STR}}}(\text{STR}) \land \text{Leads}_A(\phi, \text{STR}, \sigma, v)$ defines such a function.

Let us continue with the proof of Proposition 4.1. Recall that for trees $t$ and $t'$ and an antichain $Y$, we denote by $t[t'/Y]$ the tree obtained from $t$ by grafting $t'$ at every node in $Y$.

**Claim 4.1.5.** Let $t_0$ and $t_1$ be regular trees. Then, there is an MSO-formula $\text{graft}_{t_0, t_1}(Y, \sigma)$ defining a function that for every antichain $Y$ returns the tree $t_0[t_1'/Y]$.

Proof of Claim 4.1.5. $t_0$ and $t_1$ are regular, and therefore there are MSO-formulas $\psi_{t_0}(\sigma)$ and $\psi_{t_1}(\sigma)$ that defines $t_0$ and $t_1$.

Let $\psi_{t_1}(y, \sigma)$ be a formula that is obtained from $\psi_{t_1}(\sigma)$ by relativizing the first-order quantifiers to $\geq y$, i.e., by replacing subformulas of the form $\exists x(\ldots)$ and $\forall x(\ldots)$ by $\exists x(x \geq y) \land (\ldots)$ and $\forall x(x \geq y) \rightarrow (\ldots)$. Then, $v, t \models \psi_{t_1}(y, \sigma)$ iff $t_{\geq v} = t_1$. Hence, $\text{graft}_{t_0, t_1}(Y, \sigma)$ can be defined as the conjunction of:

1. $\exists \sigma_0 \psi_{t_0}(\sigma_0) \land \forall v - "\text{if no } Y \text{ node is an ancestor of } v \text{ then } \sigma(v) = \sigma_0(v),"$ and
2. $\forall y(y \in Y) \rightarrow \psi_{t_1}(y, \sigma) - "\text{at every node in } Y \text{ a tree } t_1 \text{ is grafted}."

\[\]
4.3. Finishing the Proof of Proposition 4.1. Now, we have all the ingredients ready for the proof of Proposition 4.1.

Let $A$ be such that $L(A) = L$, and let $\alpha_{t_0, A, \text{STR}}(\phi, v)$ be a formula that defines the function from Proposition 4.6 ($t_0[t_1/Y]$ now takes the role of $t$).

Define a formula: \[
\text{Choice}_{A, t_0, t_1, \text{STR}}(Y, \phi, y) := y \in Y \land \exists v(\alpha_{t_0, A, \text{STR}}(\phi, v) \land v \geq y).
\]

**Claim 4.1.6.** Choice $\text{Choice}_{A, t_0, t_1, \text{STR}}(Y, \phi, y)$ defines a function that for every non-empty antichain $Y$ and an accepting computation $\phi$ of $A$ on $t_0[t_1/Y]$, returns a node $y \in Y$.

**Proof.** By Proposition 4.6, $\alpha_{t_0, A, \text{STR}}(\phi, v)$ returns a node $v$ such that $t_0(v) \neq (t_0[t_1/Y])(v)$. By definition of $t_0[t_1/Y]$, there is a unique node $y \in Y$ such that $v \geq y$. ■

Define $\text{ChooseSubset}_{A, t_0, t_1, \text{STR}}(Y, X) := \forall x : x \in X$ iff the following conditions hold:

1. $x \in Y$ and
2. $\exists$ such that
   (a) $\text{graft}_{t_0, t_1}(Y, \sigma) - \text{"} \sigma = t_0[t_1/Y]"$ and
   (b) $\exists \phi \text{AcceptingRun}_{A}(\sigma, \phi) \land \text{Choice}_{A, t_0, t_1, \text{STR}}(Y, \phi, x)$, where $\text{AcceptingRun}_{A}(\sigma, \phi)$ defines “$\phi$ is an accepting computation of $A$ on the tree $\sigma$."

**Claim 4.1.7.** $\text{ChooseSubset}_{A, t_0, t_1, \text{STR}}(Y, X)$ defines a function that maps every non-empty antichain $Y$ to a non-empty subset $X \subseteq Y$. Moreover, $|X| \leq |\text{ACC}(A, t_0[t_1/Y])|$. 

**Proof.** If $Y$ is non-empty, then $t_0[t_1/Y] \in L$. Hence, $A$ has at least one accepting computation on $t_0[t_1/Y]$. Therefore, $X$ is non-empty, by Claim 4.1.6. The “Moreover” part immediately follows from Claim 4.1.6. ■

Let $A$ be such that $L(A) = L$ and assume, for the sake of contradiction, that $A$ is finitely ambiguous. In particular, there are finitely many accepting computations of $A$ on $t_0[t_1/Y]$, and therefore by Claim 4.1.7, we conclude that $\text{ChooseSubset}_{A, t_0, t_1, \text{STR}}(Y, X)$ assigns to every non-empty antichain $Y$ a finite non-empty $X \subseteq Y$ – a contradiction to Lemma 2.5.

5. $k$-Ambiguous Languages

In this section we prove that for every $0 < k \in \mathbb{N}$, there is a tree language with the degree of ambiguity equal to $k$. First, we introduce some notations. For a letter $\sigma$, we denote by $t_\sigma$, the full-binary tree with all nodes labeled by $\sigma$. Let $L_{-a_1 \lor \cdots \lor -a_k} := L_{-a_1} \cup \cdots \cup L_{-a_k}$ be a tree language over alphabet $\Sigma_n = \{c, a_1, a_2, \ldots, a_n\}$, where $L_{-a_i} := \{t \in T_{\Sigma_n}^{\omega} \mid \text{no node in } t \text{ is labeled by } a_i\}$. 

**Proposition 5.1.** The degree of ambiguity of $L_{-a_1 \lor \cdots \lor -a_k}$ for $k \leq n$ is $k$.

It is easy to see that $L_{-a_i}$ are accepted by deterministic PTA. Therefore, by Lemma 3.1, we obtain that $L_{-a_1 \lor \cdots \lor -a_k}$ is $k$-ambiguous. In the rest of this section we will show that $L_{-a_1 \lor \cdots \lor -a_k}$ is not $(k - 1)$-ambiguous. It was shown in [BS13] that $L_{-a_1 \lor -a_2}$ is ambiguous.

**Lemma 5.2.** Let $L_{\exists a_1 \land \cdots \land \exists a_m} := \{t \in T_{\Sigma_n}^{\omega} \mid \text{for every } i \leq m \text{ there is a node in } t \text{ labeled by } a_i\}$, and let $L$ be a tree language such that $t_c \notin L$ and $L_{\exists a_1 \land \cdots \land \exists a_m} \cap T_{\{c,a_1,\ldots,a_m\}}^{\omega} \subseteq L$. Then, $L$ is not finitely ambiguous.
Proof. Define a function \( F : \Sigma^* \to \Sigma \) such that \( F(\sigma_1 \ldots \sigma_k) := a_{k-i+1} \) if there is \( i \) such that \( \sigma_i = a_1 \), for all \( j < i : \sigma_j \not= a_1 \) and \( k - i + 1 \leq m \). Otherwise, \( F(\sigma_1 \ldots \sigma_k) := c \).

It is easy to see that \( F \) is definable by a Moore machine, and \( \forall t \in T^\omega_{\Sigma} : t \in L_{\geq a_1} \) iff \( \widehat{F}(t) \in L \). Therefore, by Lemma 3.10 we conclude that \( da(L) \geq da(L_{\geq a_1}) \). Since \( L_{\geq a_1} \) is not finitely ambiguous (by Corollary 4.4 (3)), we conclude that \( L \) is not finitely ambiguous. \( \square \)

**Notations.** Let \( a \in \Sigma \), \( t_1 \in T^\omega_{\Sigma} \) and \( t_2 \in T^\omega_{\Sigma} \). We define \( T_{e}(a,t_1,t_2) \in T^\omega_{\Sigma} \) as a tree \( t \) where \( t(e) = a \), \( t_{\geq 1} = t_1 \) and \( t_{\geq r} = t_2 \).

**Lemma 5.3.** Let \( A \) be a finitely ambiguous automaton over alphabet \( \Sigma \) such that \( L(A) = L_{\leq a_1 \lor \ldots \lor \leq a_{m}} \) for \( k \leq n \). Then \( |ACC(A,t_c)| \geq k \).

Proof. We will prove by induction on \( k \). For \( k = 1 \) the claim holds trivially, since \( t_c \in L(A) \) implies that \( |ACC(A,t_c)| \geq 1 \).

Assume the claim holds for all \( k < m \leq n \) and prove for \( k = m \).

Let \( A = (Q, \Sigma, Q_I, \delta, \emptyset, C) \) be a finitely ambiguous automaton that accepts \( L_{\leq a_1 \lor \ldots \lor \leq a_{m}} \).
Define \( R := \{(q_1,q_2) \in Q \times Q | \exists q_i \in Q_I : (q_i,c,q_i,q_2) \in \delta \} \), and let \( R[1] \) and \( R[2] \) be the projections of the first and second coordinate of \( R \) on \( Q \), respectively.

Define \( Q_{\geq m} := \{ q \in R[1] | L(A) \cap L_{\geq a_{m}} \neq \emptyset \} \), and let \( Q_{\leq m \land \lnot t} := \{ q \in Q_{\geq m} | t_c \in L(A_q) \} \) and \( Q_{\leq m \land \lnot t} := Q_{\leq m} \setminus Q_{\geq m} \).

By definition of \( Q_{\leq m \land \lnot t_c} \) we have \( t_c \notin L(A_{Q_{\leq m \land \lnot t_c}}) \) and therefore \( L(A_{Q_{\leq m \land \lnot t_c}}) \cap T^\omega_{\{c,a_m\}} \setminus \{ t_c \} \) is not finitely ambiguous by Corollary 4.4 (2). \( L(A_{Q_{\geq m \land \lnot t_c}}) \) is finitely ambiguous (by Corollary 3.5) and since \( T^\omega_{\{c,a_m\}} \) is unambiguous we conclude that \( L(A_{Q_{\geq m \land \lnot t_c}}) \cap T^\omega_{\{c,a_m\}} \setminus \{ t_c \} \) is not finitely ambiguous by Corollary 3.2. Therefore, by Lemma 3.6, there is a tree \( t' \in T^\omega_{\{c,a_m\}} \setminus \{ t_c \} = L_{\geq a_m} \cap T^\omega_{\{c,a_m\}} \) such that \( t' \notin L(A_{Q_{\geq m \land \lnot t_c}}) \), and since \( L_{\geq a_m} \cap T^\omega_{\{c,a_m\}} \subseteq L(A_{Q_{\leq m \land t_c}}) = L(A_{Q_{\leq m \land t_c}}) \cup L(A_{Q_{\leq m \land \lnot t_c}}) \) we conclude that \( t' \notin L(A_{Q_{\leq m \land \lnot t_c}}) \).

Define \( Q' := \{ q \in R[1] | t' \in L(A_q) \} \) and \( R' := \{(q_1,q_2) \in R | q_1 \in Q' \} \). Since \( t' \in L_{\geq a_m} \cap T^\omega_{\{c,a_m\}} \), we conclude that \( \{ t \in T^\omega_{\Sigma} \setminus T_{e}(c,t',t) \in L_{\leq a_1 \lor \ldots \lor \leq a_{m-1}} \} = L_{\leq a_1 \lor \ldots \lor \leq a_{m-1}} \). Therefore, \( L(A_{R'[2]}) = L_{\leq a_1 \lor \ldots \lor \leq a_{m-1}} \), and by induction assumption we now obtain that: \( |ACC(A_{R'[2]},t_c)| \geq m - 1 \).

For each computation \( \phi \in ACC(A_{R'[2]},t_c) \) we will construct a computation \( g(\phi) \in ACC(A,t_c) \), as following. Let \( q_2 := \phi(\epsilon) \). By the definition of \( R' \), there is \( (q_1,q_2) \in R' \) such that \( t' \in L(A_{q_1}) \). Since \( t' \in L(A_{Q_{\leq m \land t_c}}) \) we have \( t_c \in L(A_{q_1}) \), and therefore there is a computation \( \phi_c \in ACC(A_{q_1},t_c) \). Let \( q_i \in Q_I \) such that \( (q_i,c,q_i,q_2) \in \delta \). By defining \( g(\phi) := T_{e}(c,\phi_c,\phi) \) we obtain that \( g(\phi) \in ACC(A,t_c) \), as requested.

Let \( \Phi := \{ (g(\phi)) | \phi \in ACC(A_{R'[2]},t_c) \} \), \( g(\phi) \geq \epsilon = \phi \) and therefore \( g \) is injective, and we conclude that \( |\Phi| = |ACC(A_{R'[2]},t_c)| \geq m - 1 \).

We now need to find an additional computation \( \phi \in ACC(A,t_c) \) such that \( \phi \not\in \Phi \), resulting \( |ACC(A,t_c)| \geq m \).

Let \( Q_{\geq m} := \{ q \in R[2] | L(A_q) \cap L_{\geq a_{m-1}} \not= \emptyset \} \) and let \( Q_{\leq m \land a_1 \lor \ldots \lor a_{m-1}} := \{ q \in Q_{\geq m} \cap L_{\geq a_{m-1}} \} \) and \( Q_{\leq m \land a_1 \lor \ldots \lor a_{m-1}} := Q_{a_1 \lor \ldots \lor a_{m-1}} \setminus Q_{\leq m \land a_1 \lor \ldots \lor a_{m-1}} \).

**Claim 5.3.1.** There is a full-binary tree \( t'' \in L_{\geq a_1 \lor \ldots \lor \leq a_{m-1}} \cap T^\omega_{\{c,a_1,\ldots,a_{m-1}\}} \) such that \( t'' \in L(A_{Q_{\leq m \land a_1 \lor \ldots \lor a_{m-1}}} \) and \( t'' \notin L(A_{Q_{\leq m \land a_1 \lor \ldots \lor a_{m-1}}}) \).
We have

Let \( \Sigma = \text{Tree} \) nodes, and therefore \( t \) by grafting computations \( \phi \) from and therefore \( t \) \( \text{ACC} \) and therefore by Lemma 5.2 we conclude that \( L(Q_t) \) being finitely ambiguous.

\[ \text{Proof.} \]

By the definition of \( R[2] \) we have \( L_{a_1 \ldots \ldots a_m} \cap T_{\{c,a_1,\ldots,a_m-1\}}^w \subseteq L(A_{R[2]}) \) and therefore by the definition of \( Q_{t,c} \) and \( Q_{\text{ACC}} \) we conclude that \( L(A_{Q_{t,c}}) \) is not finitely ambiguous – a contradiction to \( A \) being finitely ambiguous.

Let \( t'' \) be a tree as in Claim 5.3.1. We have \( t'' \in L_{a_1 \ldots \ldots a_m} \cap T_{\{c,a_1,\ldots,a_m-1\}}^w \), and therefore \( \text{Tree}(c,t_c,t'') \in L_{a_1 \vee \ldots \vee a_m} = L(A) \), and there is a computation \( \phi \in \text{ACC}(A, \text{Tree}(c,t_c,t'')) \). Let \( q := \phi(r) \). By definition of \( t'' \), we have \( q \in Q_{t_c} \) and \( t_c \in L(A_q) \). Let \( \phi_c \in \text{ACC}(A_q,t_c) \) and let \( \phi' \) be the computation obtained from \( \phi \) by grafting \( \phi_c \) on \( r \). We conclude that \( \phi' \in \text{ACC}(A,t_c) \).

Assume, for the sake of contradiction, that \( \phi' \notin \Phi \), and let \( q_1 := \phi'(l) \) and \( q_2 := \phi'(r) \). We have \( t' \in L(A_{q_1}) \) (by definition of \( \Phi \)) and \( t'' \in L(A_{q_2}) \) (by definition of \( \phi' \)). Therefore, by grafting computations \( \phi_{t'} \in \text{ACC}(A_{q_1}, t') \) and \( \phi_{t''} \in \text{ACC}(A_{q_2}, t'') \) to the left and right children of the root of \( t_c \), respectively, we obtain \( \text{Tree}(c,t',t'') \in L(A) \). That is a contradiction, since \( t' \) contains an \( a_m \) labeled node, and \( t'' \) contains \( a_1, \ldots, a_{m-1} \) labeled nodes, and therefore \( \text{Tree}(c,t',t'') \notin L_{a_1 \vee \ldots \vee a_m} \).

We conclude that \( \phi' \notin \Phi \), and therefore \( |\text{ACC}(A,t_c)| \geq 1 + |\Phi| = 1 + (m - 1) = m \).

6. FINITELY AMBIGUOUS LANGUAGES

Definition 6.1. Let \( \Sigma = \{a_1, a_2, c\} \). We define the following languages over \( \Sigma \):

- For \( k, m \in \mathbb{N} \) such that \( k < m \), we define \( L_{k,m} \) as the set of trees that are obtained from \( t_c \) by grafting a tree \( t' \in L_{a_1 \vee \ldots \vee a_2} \) on node \( l^k r \), and grafting \( t_{a_1} \) on node \( t^m \).
- For \( m \in \mathbb{N} \) we define \( L_m := \cup_{k<m} L_{k,m} \).
- \( L^a := \cup_{m \in \mathbb{N}} L_m \).

In this section we prove the following proposition:

Proposition 6.2. The degree of ambiguity of \( L^a \) is finite.

The proposition follows from Lemma 6.3 and Lemma 6.6 proved below.

Lemma 6.3. There is a finitely ambiguous automaton that accepts \( L^a \)

Proof. On a tree \( t \in L_m \) the automaton “guesses” a position \( i < m \), checks that \( t_{\geq i} \in L_{a_1 \vee \ldots \vee a_2} \) (using a 2-ambiguous automaton), checks that \( t_{\geq i} = t_c \) for all \( j \neq i \) and \( j < m \), and checks that \( t_{\geq i} = t_{a_1} \) (using deterministic automata). Below, a more detailed proof is given.

First, notice that there are deterministic PTA \( A_c, A_{a_1}, A_{a_2} \) and \( A_{-a_2} \) that accepts languages \( \{t_c\}, \{t_{a_1}\}, L_{a_1} \) and \( L_{-a_2} \), respectively.

By Lemma 3.1, there is a 2-ambiguous automaton \( A_{-a_1 \vee -a_2} \) that accepts the language \( L_{-a_1 \vee -a_2} := L_{-a_1} \cup L_{-a_2} \).

We will construct an automaton \( B := (Q_B, \Sigma_B, Q_{t_B}, \delta_B, \zeta_B) \) that accepts \( L^a \).

- \( Q_B \) is defined as the union of states of \( A_{a_1}, A_c \) and \( A_{-a_1 \vee -a_2} \), along with additional states \( q_1, q_2 \).
- \( \Sigma_B := \{a_1, a_2, c\} \)
• \( Q_{t_B} := \{ q_1 \} \)
• \( \delta_B \) will consists of the transitions of \( A_{a_1}, A_c \) and \( A_{a_1 \lor \neg a_2} \), along with additional transitions:
  - \( (q_1, c, q_1, p) \in \delta_B \) for \( p \) an initial state in \( A_c \)
  - \( (q_1, c, q_2, p) \in \delta_B \) for \( p \) an initial state in \( A_{a_1 \lor \neg a_2} \)
  - \( (q_2, c, q_2, p) \in \delta_B \) for \( p \) an initial state in \( A_c \)
  - \( (q_2, a_1, p, p) \in \delta_B \) for \( p \) an initial state in \( A_{a_1} \)
• \( C_B(q_1) := 1 \), \( C_B(q_2) := 1 \), and for other states, the assigned color would be the same as in the automaton the state has originated from \((A_{a_1}, A_c) \) or \( (A_{a_1 \lor \neg a_2}) \)

It is easy to see that \( L(B) = L^{f_a} \).

Let \( t \in L(B) \). By definition of \( L^{f_a} \), there is \( m \in \mathbb{N} \) such that \( t \in L_m \). If \( \phi \) is an accepting computation on \( t \), then \( \phi \) assigns to the first \( m + 2 \) nodes on the leftmost branch the sequence \( q_1, q_2, \ldots, q_1, q_2, \ldots, q_m \) for some \( i \in \{1, \ldots, m\} \), where \( q_{a_1} \) is the initial state of \( A_{a_1} \) \( i \) times + \( m - i + 1 \) times (total \( m \) possibilities). \( \phi \) assigns to \( l^i \cdot r \) the initial state of \( A_c \) if \( j < i - 1 \) or \( i - 1 < j < m \); the initial state of \( A_{a_1 \lor \neg a_2} \) if \( j = i - 1 \); and the initial state of \( A_{a_1} \) if \( j \geq m \). Since \( A_c \) and \( A_{a_1} \) are deterministic and \( A_{a_1 \lor \neg a_2} \) is 2-ambiguous, the number of accepting computations on \( t \) is at most \( 2m \), hence, finite. \( \square \)

**Lemma 6.4.** Let \( L \) be a tree language such that \( L_m \subseteq L \subseteq L^{f_a} \). Then, \( L \) is not \( m - 1 \) ambiguous.

**Proof.** Let \( \mathcal{A} \) be an automaton with states \( Q \) that accepts \( L \), and assume \( \mathcal{A} \) is finitely ambiguous. Define a set \( Q' \subseteq Q \) by \( Q' := \{ \phi(l^i r) \mid i < m \land \exists t \in L : \phi \in \text{ACC}(\mathcal{A}, t) \} \) and \( Q_{a_1} := \{ q \in Q' \mid L_{a_1} \cap L(\mathcal{A}_q) = \emptyset \} \), and let \( Q_{t_c \land \exists a_1} := \{ q \in Q_{a_1} \mid t_c \in L(\mathcal{A}_q) \} \) and \( Q_{\neg t_c \lor \exists a_1} := Q_{a_1} \setminus Q_{t_c \land \exists a_1} \).

Relying on the fact that \( T_{\{c,a_1\}} \setminus \{ t_c \} \) is not finitely ambiguous (by Corollary 4.4 (2)), we derive the following claim:

**Claim 6.4.1.** There is a tree \( t_{a_1} \in (T_{\{c,a_1\}} \setminus \{ t_c \}) \cap (L(\mathcal{A}_{Q_{t_c \land \exists a_1}}) \setminus L(\mathcal{A}_{Q_{\neg t_c \lor \exists a_1}})) \). \( \square \)

Recall that \( t^m \) is the tree that is obtained from \( t_c \) by grafting \( t_{a_1} \) on node \( t^m \). For each \( i < m \), define \( t_i \) as the tree that is obtained from \( t^m \) by grafting \( t_{a_1} \) on node \( l^i r \). It is clear that \( t_i \in L(\mathcal{A}) \), and therefore there is an accepting computation \( \phi_i \) of \( \mathcal{A} \) on \( t_i \).

\[ t_{a_1} \in L(\mathcal{A}_{Q_{t_c \land \exists a_1}}) \setminus L(\mathcal{A}_{Q_{\neg t_c \lor \exists a_1}}) \] and since \( t_{a_1} \in \mathcal{A}_{\phi_i(l^i r)} \) we conclude that \( \phi_i(l^i r) \in Q_{t_c \land \exists a_1} \) and therefore \( t_c \in L(\mathcal{A}_{\phi_i(l^i r)}) \). Let \( \phi_i^c \in \text{ACC}(\mathcal{A}_{\phi_i(l^i r)}, t_c) \), and construct a computation \( \phi'_i \) from \( \phi_i \) by grafting \( \phi_i^c \) on \( l^i r \). This tree that is obtained from \( t_i \) by grafting \( t_c \) on \( l^i r \) is the tree \( t^m \) and therefore \( \phi'_i \in \text{ACC}(\mathcal{A}, t^m) \).

We are going to show that for all \( i < j < m \), the computations \( \phi'_i, \phi'_j \in \text{ACC}(\mathcal{A}, t^m) \) are different. Assume towards a contradiction \( \phi'_i = \phi'_j \) and let \( \tilde{\phi} := \phi'_i \). Define \( p_i := \tilde{\phi}(l^i r) \), \( p_j := \tilde{\phi}(l^j r) \), and let \( \phi_{p_i} \in \text{ACC}(\mathcal{A}_{p_i}, t_{a_1}) \) and \( \phi_{p_j} \in \text{ACC}(\mathcal{A}_{p_j}, t_{a_1}) \). Construct \( t' \) from \( t^m \) by grafting \( t_{a_1} \) on nodes \( l^i r \) and \( l^j r \), and construct \( \phi' \) from \( \tilde{\phi} \) by grafting \( \phi_{p_i} \) on \( l^i r \) and \( \phi_{p_j} \) on \( l^j r \). It follows that \( \phi' \) is an accepting computation of \( \mathcal{A} \) on \( t' \). That is a contradiction, since \( t' \notin L^{f_a} \) (since \( t_{a_1} \geq l^i r = l^j r = t_{a_1} \neq t_c \) and therefore \( t' \notin L \) (since \( L \subseteq L^{f_a} \)). We conclude that there are at least \( m \) different accepting computations of \( \mathcal{A} \) on \( t^m \). \( \square \)

**Remark 6.5.** The language \( L_m \) is \( 2m \) ambiguous but not \( m - 1 \) ambiguous. This implies that the hierarchy of ambiguous languages is infinite. The point of the more complex construction in Section 5 is to show that this hierarchy is populated at every level.
Lemma 6.6. \(L^{fa}\) is not boundedly ambiguous

Proof. \(\forall m \in \mathbb{N} : L_m \subseteq L^{fa}\), and therefore from Lemma 6.4 it follows that \(L^{fa}\) is not \((m - 1)\)-ambiguous. That is, \(L^{fa}\) is not boundedly ambiguous.

\]

7. Uncountably Ambiguous Languages

In this section we introduce a scheme for obtaining uncountably ambiguous languages from languages that are not boundedly ambiguous. We then use this scheme to obtain natural examples of uncountably ambiguous tree languages.

Definition 7.1. Let \(L\) be an arbitrary regular tree language over alphabet \(\Sigma\) that is not boundedly ambiguous, and let \(L_0\) be an arbitrary regular tree language over alphabet \(\Sigma\) such that \(L_0 \cap L = \emptyset\). Let \(c \in \Sigma\) and define a language \(L[L_0, \phi/c]\) over alphabet \(\Sigma\): \(t \in L[L_0, \phi/c]\) iff the following conditions hold:

- \(\forall v \in l^* : t(v) = c\)
- There is an infinite set \(I \subseteq \mathbb{N}\) such that \(\forall i \in I : t_{\geq l^*} \in L^{\phi/c} \) and \(\forall i \notin I : t_{\geq l^*} \in L_0\).

Proposition 7.2. The degree of ambiguity of \(L[L_0, L^{\phi/c}]\) is \(2^{2^{\aleph_0}}\).

Proof. Let \(A = (Q, \Sigma, \Sigma I, \delta, \epsilon)\) be a PTA that accepts \(L[L_0, L^{\phi/c}]\). We will show that \(da(A) = 2^{2^{\aleph_0}}\).

Let \(Q' := \{\phi(u) | u \in l^* \cdot r \) and \(\exists t : \phi \in ACC(A, t)\}, and define \(Q_{unamb \land L_0} := \{q \in Q' | A_q \) is unambiguous and \(L(A_q) \cap L_0 = \emptyset\}\).

Claim 7.2.1. \(L(A_{Q_{unamb \land L_0}}) \subseteq L^{\phi/c}\).

Proof. Assume, for the sake of contradiction, that there is a tree \(t \in L(A_{Q_{unamb \land L_0}})\) such that \(t \notin L^{\phi/c}\). By definition of \(Q_{unamb \land L_0}\) we conclude that \(t \notin L_0\).

Let \(q \in Q_{unamb \land L_0}\) such that \(t \in L(A_q)\) and let \(\phi \in ACC(A_q, t)\). Recall that \(q \in Q'\) (since \(Q_{unamb \land L_0} \subseteq Q'\)) and there is a tree \(t' \in L(A)\), a computation \(\phi' \in ACC(A, t')\) and a node \(u \in l^* \cdot r\) such that \(\phi'(u) = q\). By the grafting lemma we conclude that \(\phi'[\phi/u]\) is an accepting computation of \(A\) on \(t'[t/u]\). Therefore, \(t'[t/u] \in L(A)\) for \(t \notin L^{\phi/c} \cup L_0\) — a contradiction to the definition of \(A\).

Notice that \(L(A_{Q_{unamb \land L_0}})\) is boundedly ambiguous by Corollary 3.2 (as a finite union of unambiguous languages), and since \(L^{\phi/c}\) is not boundedly ambiguous we conclude that \(da(L(A_{Q_{unamb \land L_0}})) = da(L^{\phi/c})\). By Claim 7.2.1 we obtain \(L(A_{Q_{unamb \land L_0}}) \subseteq L^{\phi/c}\), and applying Lemma 3.6 we conclude that there is a tree \(t_{\phi/c} \in L^{\phi/c}\) such that \(t_{\phi/c} \notin L(A_{Q_{unamb \land L_0}})\).

Let \(c \in \Sigma\) be as in the definition of \(L[L_0, L^{\phi/c}]\), and let \(t_c\) be a tree where all nodes are labeled by \(c\). Let \(A := l^* \cdot r\) be an antichain, and define \(t'' := t_c [t_{\phi/c} / A]\). By the definition of \(A\) it is clear that \(t'' \in L(A)\). Let \(\phi'' \in ACC(A, t'')\), and let \(B := \{u \in A | L(A_{\phi''(u)}) \cap L_0 \neq \emptyset\}\).

For each \(u \in B\) there is a tree \(t_u \in L_0\) and a computation \(\phi_u \in ACC(A_{\phi''(u)}, t_u)\). Therefore, by the grafting lemma, we conclude that the tree \(t'''\) that is obtained from \(t''\) by grafting \(t_u\) on each node \(u \in B\) is in \(L(A)\).

Assume, for the sake of contradiction, that \(A \setminus B\) is finite. By definition of \(t'''\), for each \(i \in \mathbb{N}\) such that \(u := l^* \cdot r \in B\) we have \(t_{\geq l^*} = t_u \in L_0\). Therefore, \(|\{i \in \mathbb{N} | t_{\geq l^*} \in L^{\phi/c}\}| = \)
since the following conditions hold:

\[ L \backslash B \text{ is infinite, and therefore there is a state } q \text{ and an infinite set } \tilde{A} \subseteq A \backslash B \text{ such that } \phi''(u) = q \text{ for all } u \in \tilde{A}. \]

Recall that \( \forall u \in \tilde{A} : t''_u = t'_{-ba} \). Notice that for each \( u \in \tilde{A} \) we have \( u \notin B \), and by definition of \( B \) we obtain \( L(\tilde{A}_{q''(u)}) \cap L_0 = L(q) \cap L_0 = \emptyset \). Since \( t_{-ba} \notin L(A_{\text{amb}^\sim \sim -L_0}) \) we conclude that \( q \notin Q_{\text{amb}^\sim \sim -L_0} \) – hence, \( A_q \) is ambiguous.

Let \( t_{amb} \in L(A_q) \) be a tree having at least two accepting computations \( \phi_1, \phi_2 \in ACC(A_q, t_{amb}) \). Let \( \hat{t} := t''[t_{amb} / \tilde{A}] \), and \( \hat{\phi} := \phi[\phi_1 / \tilde{A}] \). By the grafting lemma we obtain \( \hat{\phi} \in ACC(A, \hat{t}) \). For each \( A' \subseteq \tilde{A} \), define a computation \( \phi_{A'} := \hat{\phi}[\phi_2 / A'] \). Notice that \( \phi_{A'} \in ACC(A, \hat{t}) \) (by the grafting lemma) and that \( \forall A_1, A_2 \subseteq \tilde{A} : A_1 \neq A_2 \rightarrow \phi_{A_1} \neq \phi_{A_2} \) (since \( \phi_1 \neq \phi_2 \)). Therefore, \( |ACC(A, \hat{t})| \geq |\{A' | A' \subseteq \tilde{A}\}| = 2^{\aleph_0} \), and \( da(A) = 2^{\aleph_0} \), as requested.

We will now introduce a couple of definitions, and present three natural examples of infinite tree languages that are not countably ambiguous.

**Definition 7.3** (Characteristic tree). The characteristic tree of \( U_1, \ldots, U_n \subseteq \{l,r\}^* \) is a \( \{0,1\}^\ast \)-labeled tree \( t[U_1, \ldots, U_n] \) such that \( t[U_1, \ldots, U_n](u) := (b_1, \ldots, b_n) \) when \( u \in U_i \) for each \( 1 \leq i \leq n \).

**Definition 7.4.** For a set \( U \subseteq \{l,r\}^* \) we define \( U \downarrow \) as the downward closure of \( U \).

**Definition 7.5.** A set \( X \subseteq \{l,r\}^* \) is called perfect if \( X \neq \emptyset \) and \( \forall u \in X : \exists v_1, v_2 \in X \) such that \( v_1 \neq v_2 \) and \( v_1 \) and \( v_2 \) are incomparable and greater than \( u \).

**Proposition 7.6.** The following regular languages are not countably ambiguous:

1. \( L_{X \subseteq Y \downarrow} := \{t[X, Y] | X \subseteq Y \downarrow \} \) – “for each node in \( X \) there is a greater or equal node in \( Y \).”

2. \( L_{\text{no-max}} := \{t[X] | X \text{ has no maximal element} \} \) – “for each node in \( X \) there is a greater node in \( X \).”

3. \( L_{\text{perf}} := \{t[X] | X \text{ is perfect} \} \) – “for each node in \( X \) there are at least two greater incomparable nodes in \( X \).”

In the rest of this section we will prove Proposition 7.6.

**Proof of Proposition 7.6(1).** Let \( L_{\text{left}} := \{t[X, Y] | X = l^* \text{ and } Y \cap l^* = \emptyset \} \). It is easy to see that \( L_{\text{left}} \) can be accepted by a deterministic PTA, and therefore \( da(L_{\text{left}}) = 1 \).

By Lemma 3.1 we conclude that \( da(L_{X \subseteq Y \downarrow} \cap L_{\text{left}}) \leq da(L_{X \subseteq Y \downarrow}) \cdot da(L_{\text{left}}) = da(L_{X \subseteq Y \downarrow}) \).

We will show that \( L_{X \subseteq Y \downarrow} \cap L_{\text{left}} \) is not countably ambiguous. By the above inequality, this implies that \( L_{X \subseteq Y \downarrow} \) is not countably ambiguous.

**Claim 7.6.1.** Let \( L_{X = \emptyset, Y \neq \emptyset} := \{t[X, Y] | X = \emptyset \text{ and } Y \neq \emptyset \} \). Then \( t' \in L_{X \subseteq Y \downarrow} \cap L_{\text{left}} \) iff the following conditions hold:

1. \( \forall u \in l^* : t'(u) = (1, 0) \)
2. There is an infinite set \( I \subseteq \mathbb{N} \) such that:
   a. If \( i \in I \) then \( t'_{l^* -r} \in L_{X = \emptyset, Y \neq \emptyset} \)
   b. If \( i \notin I \) then \( t'_{l^* -r} \in \{t[0, 0] \} \)

**Proof.** Let \( \exists t' \in L_{X \subseteq Y \downarrow} \cap L_{\text{left}} \). By definition of \( L_{\text{left}} \) it is clear that the condition (1) holds, and that for each \( i \in \mathbb{N} : t'_{l^* -r} \in L_{X = \emptyset, Y \neq \emptyset} \) or \( t'_{l^* -r} = t[0, 0] \). Assume, for the sake of
contradiction, that the set \( \{ i \in \mathbb{N} \mid t'_{\geq \omega} \in L_{X=\emptyset,Y\neq\emptyset} \} \) is finite. Therefore, by the second condition, there is an index \( k \in \mathbb{N} \) such that \( \forall i \geq k : t'_{\geq \omega} = t[0,\emptyset] \). Let \( u := t^k \). By the definition of \( L_{\text{left}} \) we have \( u \in X \), and for each \( v \geq u \) we have either \( t'(v) = (1,0) \) if \( v \in t^* \), or \( t'(v) = (0,0) \) otherwise. Hence, \( \forall v \geq u : v \notin Y \), in contradiction to \( t' \in L_{X \subseteq Y \downarrow} \).

\( \iff \) Assume that the conditions hold for \( t' \). It is easy to see that \( t' \in L_{\text{left}} \). We will show that \( t' \in L_{X \subseteq Y \downarrow} \). Assume, for the sake of contradiction, that there is a node \( u \in X \) such that \( v \notin Y \) for each node \( v \geq u \). Since all nodes in \( X \) are in \( t^* \) we conclude that there is \( i \in \mathbb{N} \) such that \( u = t^i \). Notice that the set \( J \subseteq \mathbb{N} \) is infinite, and therefore there is \( j > i \) such that \( t'_j \uparrow \in L_{X=\emptyset,Y\neq\emptyset} \). Therefore, there is a node \( v \geq t^i \cdot r > t^i = u \) such that \( v \in Y \) – a contradiction.

Observe that the language \( L_{X=\emptyset,Y\neq\emptyset} := \{ t[X,Y] \mid X = \emptyset \) and \( Y \neq \emptyset \} \) can be considered as a tree language over alphabet \( \{ 0 \} \times \{ 0,1 \} \), and that \( L_{X=\emptyset,Y\neq\emptyset} = T_{\{ 0 \} \times \{ 0,1 \}} \setminus \{ t[0,\emptyset] \} \). Therefore, by Corollary 4.4(2) we conclude that \( L_{X=\emptyset,Y\neq\emptyset} \) is not finitely ambiguous.

Notice that by Claim 7.6.1 we obtain \( L_{X \subseteq Y \downarrow} \cap L_{\text{left}} = \Sigma[L_0, L^{-\text{ba}}] \), for \( L_0 = \{ t[0,\emptyset] \} \) and \( L^{-\text{ba}} = L_{X=\emptyset,Y\neq\emptyset} \). Therefore, applying Proposition 7.2 we conclude that \( L_{X \subseteq Y \downarrow} \cap L_{\text{left}} \) is not countably ambiguous.

To prove Proposition 7.6(2), we will first prove the following lemma:

**Lemma 7.7.** \( L_{\text{nno-max}} \) is not finitely ambiguous.

**Proof.** Let \( A = (Q, \Sigma, Q_I, \delta, C) \) be a PTA that accepts \( L_{\text{nno-max}} \). Let \( Q' := \{ q \in Q \mid \exists q_i \in Q_I \exists q_j \in Q : (q_i, 1, q, q') \in \delta \) and \( t[0] \in L_A(Q') \} \).

**Claim 7.7.1.** Define \( L_{\emptyset} := T^2_\Sigma \setminus \{ t[0] \} \). Then:

1. \( L_{\text{nno-max}} \setminus \{ t[0] \} \subseteq L(A_{Q'}) \)
2. \( L(A_{Q'}) \subseteq L_{\emptyset} \)

**Proof.** (1) Let \( t' \in L_{\text{nno-max}} \setminus \{ t[0] \} \), and let \( t_{\infty} := t[\{ \} \} \) (that is, \( t_{\infty}(\epsilon) := 1 \), and \( \forall u \neq \epsilon : t_{\infty}(u) := 0 \). Let \( t'' := t_{\infty}[t'/t] \). By the definition of \( L_{\text{nno-max}} \) we obtain \( t'' \in L_{\text{nno-max}} \). Therefore, there is a computation \( \phi \in ACC(A, t'') \) such that \( \phi(1) \in Q' \) and \( t' \in L(A_{\phi(t')}) \), as requested.

(2) Assume, for the sake of contradiction, that \( t[0] \in L(A_{Q'}) \). Then there is a transition \( (q_i, 1, q_1, q_2) \in \delta \) from an initial state \( q_i \) such that \( t[0] \in L(A_{q_1}) \) and \( t[0] \in L(A_{q_2}) \). Therefore, we conclude that \( t_{\infty} := t[\{ \} \} \) is accepted by \( A \) – a contradiction to the definition of \( L_{\text{nno-max}} \).

Let \( \Sigma := \{ 0,1 \} \). Define a function \( F : \Sigma^* \to \Sigma \) such that

\[
F(\sigma_1, \ldots, \sigma_m) := \begin{cases} 1 & \exists 1 \leq i \leq m : \sigma_i = 1 \\ 0 & \text{otherwise} \end{cases}
\]

It is easy to see that \( F \) is definable by a Moore machine. We show that \( F \) reduces \( L_{\emptyset} \) to \( L(A_{Q'}) \).

Notice that \( \forall t' \in T^2_\Sigma : t' \in L_{\emptyset} \to \hat{F}(t') \in L_{\text{nno-max}} \setminus \{ t[0] \} \). Since \( L_{\text{nno-max}} \setminus \{ t[0] \} \subseteq L(A_{Q'}) \) (by Claim 7.7.1(1)) we conclude that \( \forall t' \in T^2_\Sigma : t' \in L_{\emptyset} \to \hat{F}(t') \in L(A_{Q'}) \). Conversely, \( \forall t' \in T^2_\Sigma : \hat{F}(t') \in L_{\emptyset} \to t' \in L_{\emptyset} \), and since \( L(A_{Q'}) \subseteq L_{\emptyset} \) (by Claim 7.7.1(2)) we obtain \( \forall t' \in T^2_\Sigma : \hat{F}(t') \in L(A_{Q'}) \to t' \in L_{\emptyset} \).
Therefore, by Lemma 3.10, we conclude that \( da(L(A_Q')) \geq da(L_{\emptyset}) \). Notice that \( L_{\emptyset} = T_{\Sigma}^\emptyset \setminus \{t[\emptyset]\} \) and by Corollary 4.4(2) we obtain \( da(L_{\emptyset}) \geq \aleph_0 \). Hence, \( A_Q' \) is not finitely ambiguous, and by Corollary 3.5 we conclude that \( da(A) \geq \aleph_0 \).

\[\Box\]

**Proof of Proposition 7.6(2).** Let \( L_{t^* \cap X = \emptyset} := \{t[X] \mid X \cap t^* = \emptyset\} \). It is easy to construct a deterministic PTA that accepts \( L_{t^* \cap X = \emptyset} \) and therefore \( da(L_{t^* \cap X = \emptyset}) = 1 \).

By Lemma 3.1 we conclude that \( da(L_{no\text{-}max} \cap L_{t^* \cap X = \emptyset}) \leq da(L_{no\text{-}max} \cdot da(L_{t^* \cap X = \emptyset}) = da(L_{no\text{-}max}) \). We will show that \( da(L_{no\text{-}max} \cap L_{t^* \cap X = \emptyset}) = 2^{\aleph_0} \), and the lemma will follow.

Notice that \( t' \in L_{no\text{-}max} \cap L_{t^* \cap X = \emptyset} \) iff the following hold:

- \( \forall u \in l^* : t(u) = 0 \)
- \( \forall u \in l^* \cdot r : t^*_{\geq u} \in L_{no\text{-}max} \)

It is easy to see that \( L_{no\text{-}max} \cap L_{t^* \cap X = \emptyset} = \Sigma[L_0, L_{ba}] \) for \( L_{ba} := L_{no\text{-}max} \) (which is not boundedly ambiguous, by Lemma 7.7) and \( L_0 := \emptyset \). Therefore, by Proposition 7.2 we conclude that \( da(L_{no\text{-}max} \cap L_{t^* \cap X = \emptyset}) = 2^{\aleph_0} \), as requested.

\[\Box\]

**Proof of Proposition 7.6(3).** Let \( \text{contains}_{-t^*} := \{t[X] \mid t^* \subseteq X\} \). It is easy to see that \( \text{contains}_{-t^*} \) can be accepted by a deterministic PTA, and therefore \( da(\text{contains}_{-t^*}) = 1 \).

Look at the language \( L_{perf} \cap \text{contains}_{-t^*} \). By Lemma 3.1 we obtain \( da(L_{perf} \cap \text{contains}_{-t^*}) \leq da(L_{perf}) \cdot da(\text{contains}_{-t^*}) = da(L_{perf}) \). We will show that \( L_{perf} \cap \text{contains}_{-t^*} \) is not countably ambiguous. By the above inequality, this implies that \( da(L_{perf}) = 2^{\aleph_0} \).

**Claim 7.7.2.** \( L_{perf} \) is not finitely ambiguous.

**Proof.** Define a function \( F : \Sigma^* \rightarrow \Sigma \) such that \( F(\sigma_1, \ldots, \sigma_m) := \begin{cases} 1 & \exists 1 \leq i \leq m : \sigma_i = 1 \\ 0 & \text{otherwise.} \end{cases} \) It is easy to see that \( F \) is definable by a Moore machine, and that \( \forall t' \in T_{\Sigma}^\emptyset : t' \in T_{\Sigma}^\emptyset \setminus \{t[\emptyset]\} \leftrightarrow \hat{F}(t') \in L_{perf} \). Notice that \( T_{\Sigma}^\emptyset \setminus \{t[\emptyset]\} \) is not finitely ambiguous (by Corollary 4.4(2)), and therefore by Lemma 3.10 we conclude that \( L_{perf} \) is not finitely ambiguous.

**Claim 7.7.3.** \( t' \in L_{perf} \cap \text{contains}_{-t^*} \) iff the following conditions hold:

1. \( \forall u \in l^* : t'(u) = 1 \)
2. There is an infinite set \( I \subseteq \mathbb{N} \) such that \( \forall i \in I : t'_{\geq \mu, r} \in L_{perf} \) and \( \forall i \notin I : t'_{\geq \mu, r} \in \{t[\emptyset]\} \).

**Proof.** \( \Rightarrow \): Let \( t' \in L_{perf} \cap \text{contains}_{-t^*} \). By definition of \( \text{contains}_{-t^*} \) it is clear that condition (1) holds for \( t' \). Notice that \( \forall i \in \mathbb{N} : t'_{\geq \mu, r} \in L_{perf} \) or \( t'_{\geq \mu, r} = t[\emptyset] \). Assume, for the sake of contradiction, that \( \{i \in \mathbb{N} \mid t'_{\geq \mu, r} \in L_{perf}\} \) is finite. Therefore, there is \( k \in \mathbb{N} \) such that \( \forall i \geq k : t'_{\geq \mu, r} = t[\emptyset] \). Let \( u := t^k \), and notice that \( t'(u) = 1 \), and \( \forall u > v : t'(v) = 1 \leftrightarrow v \in l^* \).

Hence, each pair of 1-labeled nodes that are greater than \( u \) are comparable – a contradiction to the definition of \( L_{perf} \).

\( \Leftarrow \): Let \( t' \) such that the conditions hold. By the first condition it is clear that \( t' \in L_{contains_{-t^*}} \). We will prove that \( t' \in L_{perf} \), and the claim will follow. First, notice that \( t'(\epsilon) = 1 \), and therefore \( t' \neq t[\emptyset] \). Let \( u \) be a node such that \( t'(u) = 1 \). If \( u \in l^* \) then by the second condition, there is a node \( v \in l^* \cdot r \) such that \( v > u \) and \( t_{\geq v} \in L_{perf} \). Therefore, there are two nodes \( w_1, w_2 > v > u \) such that \( w_1 \perp w_2 \) and \( t'(w_1) = t'(w_2) = 1 \). Otherwise (\( u \notin l^* \)), there is a node \( v \in l^* \cdot r \), such that \( u > v \) and \( t_{\geq v} \in L_{perf} \), and by definition of \( L_{perf} \) we conclude that there are two nodes \( w_1, w_2 > u \) such that \( w_1 \perp w_2 \) and \( t'(w_1) = t'(w_2) = 1 \) – hence, \( t' \in L_{perf} \).

\[\Box\]
It is easy to see that $L_{\text{perf}} \cap L_{\text{contains} - t^*} = 2 \left[ L_0, L^{-ba} \right]$ for $L^{-ba} := L_{\text{perf}}$ (which is not boundedly ambiguous, by Claim 7.7.2) and $L_0 := \{ t[\emptyset] \}$. Therefore, by Proposition 7.2 we conclude that $L_{\text{perf}} \cap L_{\text{contains} - t^*} = 2^{\aleph_0}$, as requested.

Observe that our proof shows that $L_{\text{perf} \wedge \text{min}} := \{ t[X] \mid X \text{ is perfect and has the } \leq \text{-minimal element} \}$ is also uncountably ambiguous. We conclude with an instructive example of an unambiguous language which is similar to $L_{\text{perf} \wedge \text{min}}$. Let $X \subseteq \{ l, r \}^*$ be a set of nodes. We say that $u \in X$ is a $X$-successor of $v$ if $u > v$ and there is no node $w \in X$ such that $v < w < u$. We call $X$ a full-binary subset-tree if there is a minimal node, and each node in $X$ has two $X$-successors.

Note that if $X$ is a full-binary subset tree then $X$ is perfect and has the $\leq$-minimal element. However the language $L_{\text{binary}} := \{ t[X] \mid X \text{ is a full-binary subset tree} \}$ is unambiguous.


text:

8. **Countable Languages are Unambiguous**

In this section we prove the following Proposition:

**Proposition 8.1.** Each regular countable tree language is unambiguous

This section is self-contained and lacks technical connections to the previous sections. It is instructive to compare the above Proposition with Corollary 4.4(1) which states that the complement of every countable tree language is not finitely ambiguous.

To prove Proposition 8.1 we first recall finite tree automata (Subsection 8.1). Then, we present Niwiński’s Representation for Countable Languages (Subsection 8.2). Finally, the proof of Proposition 8.1 is given (Subsection 8.3).

8.1. **Finite Trees and Finite Tree Automata.**

*Finite Trees.* A finite tree is a finite set $U \subseteq \{ l, r \}^*$ that is closed under prefix relation. $U$ is called a finite binary tree if $\forall u \in U : u \cdot l \in U \leftrightarrow u \cdot r \in U$.

*Finite $\Sigma$-labeled Binary Trees.* Let $\Sigma$ be partitioned into two sets: $\Sigma_2$ – labels of internal nodes, and $\Sigma_0$ – labels of leaves. A finite $\Sigma$-labeled binary tree is a function $t_U : U \to \Sigma$, where $U \subseteq \{ l, r \}^*$ is a finite binary tree, $t_U(v) \in \Sigma_0$ if $v$ is a leaf, and $t_U(v) \in \Sigma_2$ if $v$ has children.

When it is clear from the context, we will use “finite tree” or “labeled finite tree” for “$\Sigma$-labeled finite binary tree”.

*Finite Tree Automata (FTA).* An automaton over $\Sigma$-labeled finite trees is a tuple $B = (Q, \Sigma, Q_I, \delta)$, where $Q$ is a finite set of states, $\Sigma = \Sigma_0 \cup \Sigma_2$ is an alphabet, $Q_I$ is a set of initial states, and $\delta \subseteq (Q \times \Sigma_0) \cup (Q \times \Sigma_2 \times Q \times Q)$ is a set of transitions.

An accepting computation of $B$ on a finite tree $t_U$ is a function $\phi : U \to Q$, such that $\phi(\epsilon) \in Q_I$, and for each node $u \in U$, if $u$ is not a leaf then $(\phi(u), t_U(u), \phi(u \cdot l), \phi(u \cdot r)) \in \delta$, and otherwise $(\phi(u), t_U(u)) \in \delta$.

The language of a FTA $B$ is the set of finite trees $t$ such that $B$ has an accepting computation on $t$. A finite tree language is regular iff it is accepted by a FTA. It is well-known that every regular finite tree language is unambiguous (i.e., for every finite tree language there is an unambiguous automaton that accepts it).
8.2. Niwiński’s Representation for Countable Languages.

**Definition 8.2.** Define \( T_{\Sigma}^{fin}(x_1,\ldots,x_n) \) as the set of finite trees over alphabet \( \Sigma \cup \{x_1,\ldots,x_n\} \) where the internal nodes are \( \Sigma \)-labeled, and the leaves are \( \{x_1,\ldots,x_n\} \)-labeled.

Let \( \tau \in T_{\Sigma}^{fin}(x_1,\ldots,x_n) \) be a finite tree, and let \( t_1,\ldots,t_n \in T_{\Sigma}^{\omega} \) be infinite binary trees over alphabet \( \Sigma \). We define \( \tau[t_1/x_1,\ldots,t_n/x_n] \) as the infinite tree that is obtained from \( \tau \) by grafting \( t_i \) on leaves labeled by \( x_i \).

For a set \( M \subseteq T_{\Sigma}^{fin}(x_1,\ldots,x_n) \), we define \( M[t_1/x_1,\ldots,t_n/x_n] := \bigcup_{\tau \in M} \tau[t_1/x_1,\ldots,t_n/x_n] \).

**Theorem 8.3** (D. Niwiński [Niw91]). Let \( L \) be a countable regular tree language over alphabet \( \Sigma \). Then there is a finite set of trees \( \{t_1,\ldots,t_n\} \) such that the following hold:

1. For each tree \( t \in L \) and a tree branch \( \pi \), there is a node \( v \in \pi \) and a number \( 1 \leq i \leq n \) such that \( t_{\geq v} = t_i \).
2. There is a regular finite tree language \( M \subseteq T_{\Sigma}^{fin}(x_1,\ldots,x_n) \) s.t. \( L = M[t_1/x_1,\ldots,t_n/x_n] \).

The following lemma strengthen item (2) of Theorem 8.3 by adding another condition on \( M \), implying a unique representation of each tree in \( L \):

**Lemma 8.4.** Let \( L \) be a countable regular tree language over alphabet \( \Sigma \), and let \( \{t_1,\ldots,t_n\} \) be a finite set of trees as in Theorem 8.3. Then there is a regular finite tree language \( M \subseteq T_{\Sigma}^{fin}(x_1,\ldots,x_n) \) such that \( L = M[t_1/x_1,\ldots,t_n/x_n] \), and for each \( t \in L \) there is a unique finite tree \( \tau \in M \) such that \( t = \tau[t_1/x_1,\ldots,t_n/x_n] \).

**Proof.** For each tree \( t \in L \), let \( g(t) \) be the tree that is obtained from \( t \) by changing the label of each node \( v \in \{l,r\}^* \) where \( t_{\geq v} = t_i \) to \( x_i \), and removing all descendants of \( \{x_1,\ldots,x_n\} \)-labeled node.

**Claim 8.4.1.** \( g(t) \) is finite for all \( t \in L \).

**Proof.** Assume, for the sake of contradiction, that there is \( t \in L \) such that the set of nodes \( U \subseteq \{l,r\}^* \) of \( g(t) \) is infinite. The number of children of each node in \( U \) is bounded by 2, and therefore, by König’s Lemma, there is a tree branch \( \pi \) such that \( \forall v \in \pi : v \in U \). Therefore, by definition of \( g(t) \), we conclude that \( t_{\geq v} \neq t_i \) for each \( v \in \pi \) and \( 1 \leq i \leq n - 1 \) — a contradiction to item (1) of Theorem 8.3. \( \square \)

Notice that for each \( t \in L \) we obtain \( g(t)[t_1/x_1,\ldots,t_n/x_n] = t \), and therefore \( g \) is injective. Hence, \( L = M[t_1/x_1,\ldots,t_n/x_n] \) where \( M := \{g(t) \mid t \in L\} \). We will show that \( M \) is a regular language of finite trees.

It is easy to see that for each \( t \in L \) and finite tree \( \tau \in T_{\Sigma}^{fin}(x_1,\ldots,x_n) \), \( \tau = g(t) \) iff the following conditions hold:

- \( t = \tau[t_1/x_1,\ldots,t_n/x_n] \)
- \( t_{\geq v} \neq t_i \) for each node \( v \) in \( \tau \) that is not a leaf, and for each \( 1 \leq i \leq n \).

Since both conditions could be formulated in MSO, we conclude that \( M \) is MSO-definable, and therefore regular. \( \square \)
8.3. **Proof of Proposition 8.1.** Let $L$ be a countable regular tree language over alphabet $\Sigma$. We will show that $L$ can be accepted by an unambiguous PTA.

By Lemma 8.4, there is a regular finite tree language $M \subseteq T_{\Sigma}^{fin}(x_1, \ldots, x_n)$ and regular infinite trees $t_1, \ldots, t_n$ such that $L = M[t_1/x_1, \ldots, t_n/x_n]$. Additionally, for each $t \in L$ there is a unique $\tau \in M$ such that $t = \tau[t_1/x_1, \ldots, t_n/x_n]$.

Each infinite tree $t_i : \{l, r\}^* \rightarrow \Sigma$ is regular, and therefore, by Fact 3.11, is definable by a Moore machine $M_i = ((l, r), \Sigma, Q_i, q_1^i, \delta_i^M, out_i^M)$. Let $A_i := (Q_i, \Sigma, q_1^i, \delta_i, F_i)$ where $F_i := Q_i$, and $(q, a, q_1, q_2) \in \delta_i$ iff $q_1 = \delta(q, l)$, $q_2 = \delta(q, r)$ and $a = out_i^M(q)$. It is easy to verify that $A_i$ is unambiguous, and $L(A_i) = \{t_i\}$. $M$ is regular and therefore can be accepted by an unambiguous FTA $B = (Q_B, \Sigma \cup \{x_1, \ldots, x_n\}, q_1^B, \delta_B)$.

We use these automata to construct a PTA $A := (Q, \Sigma, Q_I, \delta, C)$, by:

- $Q := \cup_{1 \leq i \leq n} Q_i \cup Q_B$
- $q_1^i := \{q_1^B\} \cup \{q_i^1, (q_i^1, x_i) \in \delta_i\}$
- $\delta$ is the union of the following:
  - $\{(q, a, q_1, q_2) \in \delta_B \mid a \in \Sigma\}$ (all transitions of $B$ on inner nodes)
  - $\cup_{1 \leq i \leq n} \delta_i$
  - $\{(q, a, q_i^1, q_i^2) \mid \exists (q, a, q_1, q_2) \in \delta_B : (q_1, x_i) \in \delta_B \text{ and } (q_2, x_j) \in \delta_B\}$
  - $\{(q, a, q_i^1, q_i^2) \mid \exists (q, a, q_1, q_2) \in \delta_B : (q_2, x_j) \in \delta_B\}$
  - $\{(q, a, q_i^1, q_i^2) \mid \exists (q, a, q_1, q_2) \in \delta_B : (q_1, x_i) \in \delta_B\}$
- $C(q) := \begin{cases} C_i(q) \quad \text{if } q \in Q_i \\ 1 \quad \text{otherwise} \end{cases}$

It is easy to see that $L(A) = M[t_1/x_1, \ldots, t_n/x_n] = L$.

We will show that $A$ is unambiguous. For each accepting computation $\phi \in ACC(A, t)$, define a set of nodes $U_\phi := \{u \in \{l, r\}^* \mid \forall v < u : \phi(v) \in Q_B\}$. It is easy to see that $U_\phi$ is downward closed. Assume towards contradiction that $U_\phi$ is infinite – by König Lemma, $U_\phi$ contains an infinite tree branch $\pi$. By definition of $U_\phi$, all states in $\phi(\pi)$ are in $Q_B$, and therefore colored by 1. That is a contradiction to $\phi$ being an accepting computation.

Define a labeled finite tree $t_\phi : U_\phi \rightarrow \Sigma \cup \{x_1, \ldots, x_n\}$ by:

$$t_\phi := \begin{cases} x_i \quad \text{if } \phi(u) = q_i^1 \\ t(u) \quad \text{otherwise} \end{cases}$$

By definition of $t_\phi$ we obtain $t = t_\phi[t_1/x_1, \ldots, t_n/x_n]$, and by definition of $B$ we conclude that $t_\phi \in M$.

Assume, for the sake of contradiction, that $A$ is ambiguous. Therefore, there is a tree $t \in L$ and two distinct accepting computations $\phi_1, \phi_2 \in ACC(A, t)$. $A_i$ is deterministic for each $1 \leq i \leq n$, and therefore $\phi_1 \neq \phi_2$ iff $t_{\phi_1} \neq t_{\phi_2}$. We conclude that $t_{\phi_1}[t_1/x_1, \ldots, t_n/x_n] = t_{\phi_2}[t_1/x_1, \ldots, t_n/x_n]$ for $t_{\phi_1}, t_{\phi_2} \in M$ – a contradiction to the uniqueness property of $M$.

9. **Conclusion and Open Questions**

We proved that the ambiguity hierarchy is strict for regular languages over infinite trees.

For each level of the ambiguity hierarchy we provided a language which occupies this level. It is not difficult to see that all these languages are definable by MSO-formulas without the second-order quantifiers (formulas of the first-order fragment of MSO). Concerning the topological complexity, Olivier Finkel [Fin20] observed that these languages have low topological complexity: $L_{\neg a_1 \lor \cdots \lor \neg a_k}$ are closed languages; $L^{1a}$ and $L_{\exists a_3}$ are countable unions
of closed sets, i.e., a \( \Sigma_2^0 \)-sets; the uncountably ambiguous language \( L_{\text{no-max}} \) is \( \Pi_2^0 \). On the other hand, Skrzypczak [Skr18] proved that unambiguous languages climb up the whole index hierarchy and are topologically as complex as arbitrary regular tree languages.

A natural question is whether the ambiguity degree is decidable. However, this is not a trivial matter. In [BS13] some partial solutions for variants of the problem whether a given language is unambiguous are provided. We proved that countable regular languages are unambiguous. Since it is decidable whether a language is countable [Niw91], this provides a decidable sufficient condition for a language to be unambiguous.

A less ambitious task is to develop techniques for computing degrees of ambiguity and compute the degree of ambiguity of some natural languages. Let \( \Sigma_1 := \{c, a_1\} \) and \( L_{\exists a_1} := \{t \in T_{\Sigma_1}^\omega \mid \text{there are infinitely many } a_1\text{-labeled nodes in } t\} \). \( L_{\exists a_1} \) is \( \Pi_0^0 \). On the other hand, Skrzypczak [Skr18] proved that unambiguous languages climb up the whole index hierarchy and are topologically as complex as arbitrary regular tree languages.

A less ambitious task is to develop techniques for computing degrees of ambiguity and compute the degree of ambiguity of some natural languages. Let \( \Sigma_1 := \{c, a_1\} \) and \( L_{\exists a_1} := \{t \in T_{\Sigma_1}^\omega \mid \text{there are infinitely many } a_1\text{-labeled nodes in } t\} \). \( L_{\exists a_1} \) is \( \Pi_0^0 \). On the other hand, Skrzypczak [Skr18] proved that unambiguous languages climb up the whole index hierarchy and are topologically as complex as arbitrary regular tree languages.

We provided sufficient conditions for a language to be not finitely ambiguous and for a language to have uncountable degree of ambiguity. In particular, we proved that the degree of ambiguity of the complement of a countable regular language is \( \aleph_0 \) or \( 2^{\aleph_0} \), and provided natural examples of such languages with countable degree of ambiguity. We proved that the degree of ambiguity of the complement of a finite regular language is \( \aleph_0 \). Yet, it is open whether the degree of ambiguity of the complement of countable regular languages is \( \aleph_0 \).

ACKNOWLEDGMENTS

We would like to thank anonymous referees for their helpful suggestions.

REFERENCES

[Arn83] André Arnold. Rational omega-languages are non-ambiguous. Theor. Comput. Sci., 26:221–223, 09 1983.

[BKR10] Vince Bárány, Lukasz Kaiser, and Alex Rabinovich. Expressing cardinality quantifiers in monadic second-order logic over trees. Fundamenta Informaticae, 100(1-4):1–17, 2010.

[BS13] Marcin Bilkowski and Michal Skrzypczak. Unambiguity and uniformization problems on infinite trees. In Simona Ronchi Della Rocca, editor, Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy, volume 23 of LIPIcs, pages 81–100. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[CL07] Arnaud Carayol and Christof Löding. MSO on the infinite binary tree: Choice and order. In International Workshop on Computer Science Logic, pages 161–176. Springer, 2007.

[CLNW10] Arnaud Carayol, Christof Löding, Damian Niwinski, and Igor Walukiewicz. Choice functions and well-orderings over the infinite binary tree. Open Mathematics, 8(4):662–682, 2010.

[Col15] Thomas Colcombet. Unambiguity in automata theory. In International Workshop on Descriptional Complexity of Formal Systems, pages 3–18. Springer, 2015.

[EJ91] E Allen Emerson and Charanjit S Jutla. Tree automata, mu-calculus and determinacy. In FoCS, volume 91, pages 368–377. Citeseer, 1991.

[Fin20] Olivier Finkel. Personal communication, 12 2020.

[GH82] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In Proceedings of the fourteenth annual ACM symposium on Theory of computing, pages 60–65, 1982.

[GS83] Yuri Gurevich and Saharon Shelah. Rabin’s uniformization problem 1. The Journal of Symbolic Logic, 48(4):1105–1119, 1983.
Appendix A. Proof of Claim 4.1.4

Claim 4.1.4. Let $t_0$ be a regular tree such that $t_0 \notin L(A)$. Then, Pathfinder has a regular positional winning strategy in $G_{t_0,A}$.

Proof. $t_0$ is regular, and therefore there is a formula $\psi_{t_0}(\sigma)$ that defines $t_0$ in the unlabeled full-binary tree.

We will use $\psi_{t_0}(\sigma)$ to define the following formula $\text{PathfinderWins}_{A,t_0}(\phi,\text{STR})$, as the conjunction of the following conditions:

1. $\exists \pi$ such that:
   - (a) $\pi$ is a branch
   - (b) $\forall u \in \pi : (\text{STR}(u, \phi(u \cdot l), \phi(u \cdot r)) = l) \iff u \cdot l \in \pi$ – the Pathfinder moves $d_0 \ldots d_j \ldots$ are consistent with STR and are along the branch $\pi$.

2. $\exists \pi : \psi_{t_0}(\sigma)$ and at least one of the following holds:
   - (a) $\exists v \in \pi$ such that $(\phi(v), \sigma(v), \phi(v \cdot l), \phi(v \cdot r)) \notin \delta$ – the Automaton move at $(v, \phi(v))$ is invalid.
   - (b) The maximal color that $C$ assigns infinitely often to states in $\phi(\pi)$ is odd.
Claim A.1. \textit{PathfinderWins}_{A,t_0}(\phi, STR) holds for a positional strategy \textit{STR} of Pathfinder and a computation \phi of \textit{A} on a tree \textit{t'} iff the play \pi of \textit{STR} against \textit{str}_\phi in \textit{G}_{t_0,A} is winning for Pathfinder.

\textit{Proof.} By definition of \textit{G}_{t_0,A}, Pathfinder wins if either Automaton makes an invalid move (condition 2a) or the maximal color that is assigned infinitely often to the positions in \pi is odd. Since all Pathfinder positions have color 0, this is equivalent to the maximal color assigned infinitely often to Automaton positions being odd.

Let \pi = e_0, d_0, e_1, d_1, \ldots, e_i, d_i, \ldots. Notice that by condition 1, there is a unique branch \pi such that \pi = v_0, \ldots v_i, \ldots where \pi = d_0 \ldots d_{i-1}. By Claim 4.1.1, we have \phi(v_i) = q_i, where the i-th position of Automaton in \pi is (v_i, q_i). Since \textit{CG}(v_i, q_i) = \textit{C}(q_i), we conclude that the maximal color that \textit{C} assigns infinitely often to states in \phi(\pi) is odd iff the maximal color that \textit{CG} assigns infinitely often to positions in \pi is odd. This is assured by condition 2b. \hfill \blacksquare

Let \textit{WinningStrategy}_{t_0,A}(\textit{STR}) := \forall \phi \text{ such that the following holds:}
\begin{itemize}
  \item If there is t such that \phi is an accepting computation of \textit{A} on t, then:
    \begin{itemize}
      \item \textit{PathfinderWins}_{A,t_0}(\phi, \textit{STR}) holds
    \end{itemize}
\end{itemize}
Recalling that the set of all computation of \textit{A} is MSO-definable, we can conclude that \textit{WinningStrategy}_{t_0,A}(\textit{STR}) is MSO-definable in the unlabeled full-binary tree.

Claim A.2. \textit{WinningStrategy}_{t_0,A}(\textit{STR}) holds for a positional strategy \textit{STR} of Pathfinder iff \textit{STR} is a positional winning strategy of Pathfinder.

\textit{Proof.} \Rightarrow: By Claim A.1, \textit{STR} wins in \textit{G}_{t_0,A} against each positional strategy of Automaton. Assume, for the sake of contradiction, that is a non-positional strategy \textit{str'} of Automaton that wins against \textit{STR}. Then by positional determinacy of parity games, we conclude that there is a positional strategy \textit{str''} that wins against \textit{STR} – a contradiction.

\Leftarrow: Follows immediately from Claim A.1. \hfill \blacksquare

We have \textit{t_0} \notin \textit{L(A)} and therefore by Claim 4.1.1(3), Automaton does not have a positional winning strategy. From positional determinacy of parity games we conclude that Pathfinder has a positional winning strategy. Therefore, there is a strategy \textit{STR'} that satisfies \textit{WinningStrategy}_{t_0,A}(\textit{STR}) in the unlabeled full-binary tree.

Therefore, \textit{WinningStrategy}_{t_0,A}(\textit{STR}) defines a non-empty tree language over alphabet \textit{Q} \times \textit{Q} \rightarrow \{l, r\}. By Rabin’s basis Theorem, we conclude that there is a regular tree \textit{STR} in this language, and by Claim A.2 we conclude that \textit{STR} is a positional winning strategy for Pathfinder in \textit{G}_{t_0,A}. \hfill \blacksquare

Remark (Logic Free Proof of Claim 4.1.4). One can reduce a membership game for a regular tree \textit{t_0} to a game on a finite graph. By positional determinacy Theorem, Pathfinder will have a positional winning strategy in the reduced game. From this strategy a regular winning strategy in \textit{G}_{t_0,A} for Pathfinder is easily constructed.