INSTITUT FÜR INFORMATIK

UND PRAKTISCHE MATHEMATIK

Deciding the First Level of the μ-calculus Alternation Hierarchy

Ralf Küsters and Thomas Wilke

Bericht Nr. 0209
July 2002

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik und Praktische Mathematik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Deciding the First Level of the μ-calculus Alternation Hierarchy

Ralf Küsters and Thomas Wilke

Bericht Nr. 0209
July 2002

e-mail: {kuesters,wilke}@ti.informatik.uni-kiel.de

Deciding the First Level of the μ-calculus Alternation Hierarchy

Ralf Küsters and Thomas Wilke
Institut für Informatik und Praktische Mathematik
CAU Kiel, Germany
{kuesters,wilke}@ti.informatik.uni-kiel.de

Abstract

We show that the following problem is decidable and complete for deterministic exponential time. Given a formula of modal μ-calculus, determine if the formula is equivalent to a formula without greatest fixed point operators. In other words, we show that the first level of the μ-calculus fixed point alternation hierarchy is decidable in deterministic exponential time.

1 Introduction

Fixed point operators are the salient feature of modal μ-calculus. Therefore, it is only natural that one is interested in characterizing how much these operators contribute to the expressive power of the logic. One way to address this issue is to investigate natural hierarchies induced by fixed point operators. The most important hierarchy, which has been studied for decades, is the fixed point alternation hierarchy: a property expressible in modal μ-calculus is classified according to the number of alternations between greatest and least fixed points required to express the property. In 1996, this hierarchy was shown to be strict, independently by Bradfield [3] and Lenzi [8] (see also [1]). That is, the more alternations are allowed, the more one can express. In the proof of the result, a sequence of specific properties is exhibited where the ith member can be expressed with $i+1$ alternations, but not with i alternations. That is, for the specific properties of that proof we know exactly

which level of the alternation hierarchy they belong to. In general, however, only very little is known about determining the number of alternations required to express an arbitrary property. In fact, Otto [10] showed in 1999 that it is decidable whether a given property can be expressed without fixed point operators, that is, he gave a decision procedure for level 0. Urbański [11] showed decidability for the second level under the assumption that the property is given as a *deterministic* tree automaton.

In this paper, we extend Otto's result to the first level of the hierarchy: we show that it is decidable whether a given property described by a μ-calculus formula can be expressed without greatest fixed points. (By duality, this also admits a decision procedure for determining whether a given property can be expressed without least fixed points.) More precisely, we show that the problem is complete for deterministic exponential time.

Our approach to the solution of the problem is an automata-theoretic one. We exploit the correspondence between modal μ-calculus and parity tree automata [9]. This correspondence respects the fixed point alternation hierarchy. In particular, a property is expressible with least fixed points only if and only if it is recognized by an alternating tree automaton with a particular acceptance condition, namely, where a run is accepting if and only if it has no infinite path. The decidability criterion we propose is similar to decidability criteria for related problems. We show that a property, say given as a parity tree automaton, is expressible with least fixed points only if and only if the automaton is equivalent to a so-called test automaton. Decidability criteria of this type were already proposed by Landweber for checking whether a given regular ω-language belongs to the first two levels of the Borel hierarchy [7].

A straightforward implementation of our decidability criterion would result in a doubly exponential time decision procedure. The key to a singly exponential time procedure is a specific way of complementing our bottom-up test automaton and a detailed analysis of standard emptiness tests for alternating tree automata.

An overview of the paper can be found at the end of the following section.

Related Work. In independent, unpublished work [12] , Igor Walukiewicz obtained essentially the same results as we do. He proved decidability in a restricted framework, where only properties of binary trees are studied, yet one can verify that his argument can be lifted to arbitrary transition systems.

2 Basic Notions and Main Result

In this section we briefly introduce modal μ-calculus and parity tree automata. We recall the definition of the fixed point alternation hierarchy and the correspondence to parity tree automata. For more detailed information we refer the reader to [13]. In Section 2.4, we state the main result of this paper.

As usual, mathematical objects like graphs, trees, automata, etc. will be defined as fixed length tuples with certain components, just as a graph is a pair (V, E). To refer to the individual components of a structure denoted $\mathbf{S}$, the subscript $_{\mathbf{S}}$ is used. For instance, the vertex set of a graph $\mathbf{G}$ is denoted by $V_{\mathbf{G}}$.

2.1 Modal μ-calculus

Modal μ-calculus is modal logic augmented by operators for least and greatest fixed points.

2.1.1 Kripke Structures and Kripke Trees

Modal μ-calculus—just as modal logic—is a logic to express properties of Kripke structures.

A Kripke structure is a directed graph together with an interpretation (or assignment) of propositional variables in each vertex of the graph. Once and for all, we fix a countably infinite supply $\mathcal{P}$ of propositional variables. Formally, a *Kripke structure* is a tuple

$$(W, R, \lambda)$$

where W is a nonempty set of *worlds,* $R \subseteq W \times W$ is an *accessibility relation,* and $\lambda\colon W \to 2^{\mathcal{P}}$ is a *labeling function* that assigns to each world the set of propositional variables that hold true in it. A *pointed Kripke structure* is a pair $(\mathbf{K}, w)$ where $\mathbf{K}$ is a Kripke structure and $w \in W_{\mathbf{K}}$ is a *distinguished world* of it.

Two pointed Kripke structures $(\mathbf{K}, w)$ and $(\mathbf{K}', w')$ are called *bisimilar* ($(\mathbf{K}, w) \cong (\mathbf{K}', w')$ for short) when there exists a relation $R \subseteq W_{\mathbf{K}} \times W_{\mathbf{K}'}$, called a *bisimulation relation*, such that $(w, w') \in R$ and for every $(v, v') \in R$ and $p \in \mathcal{P}$:

- $p \in \lambda_{\mathbf{K}}(v)$ iff $p \in \lambda_{\mathbf{K}}(v')$,
- whenever $(v, u) \in E_{\mathbf{K}}$ for some u, then there exists u' such that $(v', u') \in E_{\mathbf{K}'}$,
- whenever $(v', u') \in E_{\mathbf{K}'}$ for some u', then there exists u such that $(v, u) \in E_{\mathbf{K}}$.

A set S of pointed Kripke structures is *closed under bisimulation* if $(\mathbf{K}, w) \in S$ and $(\mathbf{K}', w') \cong (\mathbf{K}, w)$ implies $(\mathbf{K}', w') \in S$ for every pointed Kripke structures $(\mathbf{K}, w)$ and $(\mathbf{K}', w')$.

Most of the time we will only be interested in *Kripke trees,* which are pointed Kripke structures $(\mathbf{K}, w)$ where the underlying graph $(W_{\mathbf{K}}, R_{\mathbf{K}})$ is a directed tree rooted at w. For simplicity, we will often simply write $\mathbf{K}$, since the w is uniquely determined. In that case, we will also write $\rho_{\mathbf{K}}$ for w. Kripke trees will usually be denoted by t, t', ... Given a tree t and $w \in W_t$, the subtree of t rooted at w is denoted by $t \downarrow w$. We write $\mathrm{Suc}_t(w)$ for the set of *successors* of w in t, i.e., $\mathrm{Suc}_t(w) := \{w' \mid (w, w') \in R_t\}$. Note that this set might be infinite of any cardinality. A world w is a *leaf* if its set of successors is empty; otherwise w is called an *inner node.*

We write $\mathcal{T}$ for the set of all Kripke trees. For every finite set $P \subseteq \mathcal{P}$, the set $\mathcal{T}_P$ denotes the set of all trees $t \in \mathcal{T}$ such that $\lambda_t(w) \subseteq P$ holds for every world $w \in W_t$. A *tree set* over P is a subset of $\mathcal{T}_P$.

2.1.2 Syntax

As usual, the formulas of modal μ-calculus are built from the constant symbols $\bot$ and $\top$, the symbols from $\mathcal{P}$ and their negations, using disjunction and conjunction, the modalities $\Box$ and $\Diamond$, and operators for least and greatest fixed points, μ and ν, with some minor restriction on the use of the fixed point operators.[1] Formally, the set of all L_μ formulas is defined inductively as follows.

- The symbols $\bot$ and $\top$ are L_μ formulas.
- For every $p \in \mathcal{P}$, p and $\neg p$ are L_μ formulas.
- If ϕ and ψ are L_μ formulas, then $\phi \vee \psi$ and $\phi \wedge \psi$ are L_μ formulas.

[1] In this paper, no distinction is made between symbols for propositional constants and propositional variables.

- If ϕ is an L_μ formula, then $\Box\phi$ and $\Diamond\phi$ are L_μ formulas.
- If $p \in \mathcal{P}$ and ϕ is an L_μ formula where p occurs only positive (that is, $\neg p$ does not occur), then $\mu p\phi$ and $\nu p\phi$ are L_μ formulas.

The restriction on the application of the least and greatest fixed point operator expressed in the last rule above is imposed to justify the terminology: this restriction ensures that the argument of a fixed point operator can be viewed as a monotone function and that a fixed point actually exists (for details, see below).

2.1.3 Semantics

The formulas of modal μ-calculus are interpreted in Kripke structures. Inductively, it is defined to which set of worlds an arbitrary L_μ formula is evaluated in a given Kripke structure. More precisely, for every Kripke structure $\mathbf{K}$ and every L_μ formula ϕ a set $||\phi||_{\mathbf{K}} \subseteq W_{\mathbf{K}}$ is defined.

The semantics of the atomic formulas is determined by

$$||\bot||_{\mathbf{K}} = \emptyset \ , \qquad ||\top||_{\mathbf{K}} = W_{\mathbf{K}} \ ,$$
$$||p||_{\mathbf{K}} = \{w \in W_{\mathbf{K}} \mid p \in \lambda_{\mathbf{K}}(w)\} \ , \qquad ||\neg p||_{\mathbf{K}} = W_{\mathbf{K}} \setminus ||p||_{\mathbf{K}} \ .$$

Disjunction and conjunction are interpreted as union and intersection, respectively,

$$||\phi_0 \vee \phi_1||_{\mathbf{K}} = ||\phi_0||_{\mathbf{K}} \cup ||\phi_1||_{\mathbf{K}} \ ,$$
$$||\phi_0 \wedge \phi_1||_{\mathbf{K}} = ||\phi_0||_{\mathbf{K}} \cap ||\phi_1||_{\mathbf{K}} \ .$$

The modal operators are interpreted in the usual way:

$$||\Box\phi||_{\mathbf{K}} = \{w \in W_{\mathbf{K}} \mid \mathrm{Suc}_{\mathbf{K}}(w) \subseteq ||\phi||_{\mathbf{K}}\} \ ,$$
$$||\Diamond\phi||_{\mathbf{K}} = \{w \in W_{\mathbf{K}} \mid \mathrm{Suc}_{\mathbf{K}}(w) \cap ||\phi||_{\mathbf{K}} \neq \emptyset\} \ .$$

The definition of the semantics of the fixed point operators needs a little preparation. When $\mathbf{K}$ is a Kripke structure, p is a propositional variable, and $W \subseteq W_{\mathbf{K}}$, then $\mathbf{K}[W \mapsto p]$ denotes the Kripke structure defined by

$$\mathbf{K}[W \mapsto p] = (W_{\mathbf{K}}, R_{\mathbf{K}}, \lambda_{\mathbf{K}}[W \mapsto p])$$

where $\lambda_{\mathbf{K}}[W \mapsto p]$ itself is given by

$$\lambda_{\mathbf{K}}[W \mapsto p](w) = \begin{cases} \lambda_{\mathbf{K}}(w) \cup \{p\} & \text{if } w \in W, \\ \lambda_{\mathbf{K}}(w) \setminus \{p\} & \text{otherwise.} \end{cases}$$

The semantics of the fixed point operators is now defined by

$$||\mu p \phi||_{\mathbf{K}} = \bigcap \{W \subseteq W_{\mathbf{K}} \mid ||\phi||_{\mathbf{K}[\mathbf{W} \mapsto \mathbf{p}]} \subseteq W\} \ , \tag{1}$$

$$||\nu p \phi||_{\mathbf{K}} = \bigcup \{W \subseteq W_{\mathbf{K}} \mid ||\phi||_{\mathbf{K}[\mathbf{W} \mapsto \mathbf{p}]} \supseteq W\} \ . \tag{2}$$

Let ϕ be an arbitrary L_μ formula and $\mathbf{K}$ a Kripke structure. Then it is easy to see that $W \mapsto ||\phi||_{\mathbf{K}[\mathbf{p} \mapsto \mathbf{W}]}$ is a monotone function on $2^{W_{\mathbf{K}}}$. Therefore, this function has a least and a greatest fixed point (with respect to set inclusion). By the Knaster/Tarski Theorem, these fixed points are identical with the sets denoted by the right-hand sides of (1) and (2), respectively.

Given a pointed Kripke structure $\mathbf{K}$, we will write $(\mathbf{K}, w) \models \phi$ for $w \in ||\phi||_{\mathbf{K}}$. Two L_μ formulas ϕ and ψ are called *equivalent* iff $||\phi||_{\mathbf{K}} = ||\psi||_{\mathbf{K}}$ for every Kripke structure $\mathbf{K}$.

2.1.4 The Fixed Point Alternation Hierarchy

Intuitively, properties described by L_μ formulas are placed into the fixed point alternation hierarchy according to the number of alternations between least and greatest fixed point operators of the corresponding formula. There are several ways to define an appropriate concept of fixed point alternation. The simplest one is to count syntactic alternations between least and greatest fixed point operators. A more involved one, which was tailored specifically for the purposes of efficient model-checking, was introduced by Emerson and Lei [4]. It gives rise to a coarser hierarchy. The one we recall here defines an even coarser hierarchy and was introduced by Damian Niwiński [9].

We denote the relation "is proper subformula of" by $<$. Let ϕ be an L_μ formula. An *alternating μ-chain* in ϕ of length l is a sequence

$$\phi \geq \mu p_0 \psi_0 > \nu p_1 \psi_1 > \mu p_2 \psi_2 > \cdots > \mu / \nu p_{l-1} \psi_{l-1}$$

where, for every $i < l - 1$, the variable p_i occurs free in every formula ψ with $\psi_i \geq \psi \geq \psi_{i+1}$. The maximum length of an alternating μ-chain in ϕ is denoted by $m^\mu(\phi)$. Symmetrically, ν-chains and $m^\nu(\phi)$ are defined.

The *fixed point alternation hierarchy* is defined as follows:

$$\Sigma_0 := \Pi_0 := \{\phi \in L_\mu \mid m^\mu(\phi) = m^\nu(\phi) = 0\},$$

and for every $i \geq 0$,

$$\begin{aligned} \Sigma_{i+1} &:= \{\phi \in L_\mu \mid m^\mu(\phi) \leq i+1 \wedge m^\nu(\phi) \leq i\}, \\ \Pi_{i+1} &:= \{\phi \in L_\mu \mid m^\mu(\phi) \leq i \wedge m^\nu(\phi) \leq i+1\}. \end{aligned}$$

In what follows we call an L_μ formula Σ_i*-formula* if it belongs to Σ_i. Analogously, we refer to Π_i*-formulas*.

Example 1 Let $\phi_0 = \mu p_0(\Box p_0)$ and $\phi_1 = \mu p_1(p_0 \vee \Diamond p_1)$. Then $m^\mu(\phi_0) = m^\mu(\phi_1) = 1$ and $m^\nu(\phi_0) = m^\nu(\phi_1) = 0$. Thus, ϕ_0 and ϕ_1 are Σ_1-formulas. A more interesting example is the formula $\phi_2 = \mu p_1(\nu p_1(p_0 \wedge \Diamond p_1) \vee \Box p_1)$. This formula defines the property that on all paths eventually a world is reached where an infinite path starts on which p_0 holds generally. One verifies that $m^\mu(\phi_3) = m^\nu(\phi_3) = 1$. Thus, $\phi_3 \in \Sigma_2 \cap \Pi_2$.

2.2 Monadic Second Order Logic

In this section we define the monadic second order logic (MSOL). We will interpret MSOL formulas over Kripke trees. We fix a countably infinite set of unary predicate symbols $\mathcal{P}$, a countably infinite set $V_f = \{x, y, \dots\}$ of first order variables, and a countably infinite set $V_s = \{X, Y, \dots\}$ of second order variables.

The set of all MSOL formulas over $\mathcal{P}$, V_f, V_s, and the constant sr is defined inductively as follows:

$$p(x), s(x, y), X \subseteq Y, \mathsf{sr}$$

are MSOL formulas. They are closed under Boolean operators and first- and second-order quantification. A *sentence* is a formula without free variables.

The definition of the truth of a formulas for a given Kripke tree t and a valuation V, which assigns to every first-order variable an element of W_t and to every second-order variable a subset of W_t, is defined as follows: The constant sr is interpreted as $\rho_{\mathbf{t}}$. Further, $t, V \models p(x)$ iff $p \in \lambda_t(V(x))$; $t, V \models s(x, y)$ iff $(V(x), V(y)) \in E_t$; $t, V \models X \subseteq Y$ iff $V(X) \subseteq V(Y)$. The Boolean operators and the quantifiers are interpreted as usual [6].

2.3 Parity Tree Automata

2.3.1 Informal Description

Parity tree automata are finite-state devices designed to accept or reject pointed Kripke structures. The computation of a parity tree automaton on a pointed Kripke structure proceeds in rounds. At the beginning of every round there are several copies of the parity tree automaton in different worlds of the Kripke structure, each of them in its own state; some worlds might be occupied by many copies, others might not accommodate a single one. During a round, each copy splits up in several new copies, which are sent to neighbored worlds and change their states, all this done according to the transition function. Initially, there is only one copy of the parity tree automaton; it resides in the distinguished world of the pointed Kripke structure and starts in the initial state of the parity tree automaton. To determine acceptance or rejectance of a computation of a parity tree automaton on a pointed Kripke structure the entire computation tree is inspected; acceptance is then defined via path conditions for the infinite branches of the computation tree. Namely, every state will be assigned a priority and an infinite branch of the computation will be accepting if the maximum priority occurring infinitely often is even; a computation tree will be accepting if each of its infinite branches is accepting.

2.3.2 Formal Definition

Formally, a *parity tree automaton* is a tuple

$$\mathbf{A} = (Q, q_I, \delta, \Omega) \tag{3}$$

where

- Q is a finite set of *states,*
- $q_I \in Q$ is an *initial state,*
- δ is a *transition function* as specified below, and
- $\Omega \colon Q \to \omega$ is a *priority function,* which assigns a *priority* to each state.

The transition function δ maps every state to a transition condition over $\mathcal{P}$ and Q. A *transition condition over $\mathcal{P}$ and Q* is defined as follows:

- $\bot$ and $\top$ are transition conditions,
- p and $\neg p$ are transition conditions, for every $p \in Props$,
- q, $\Box q$, and $\Diamond q$ are transition conditions, for every $q \in Q$,
- $q \wedge q'$ and $q \vee q'$ are transition conditions, for every $q, q' \in Q$.

For $q \in Q$, we define $\mathbf{A}_q = (Q, q, \delta, \Delta)$ to be the parity tree automaton obtained from $\mathbf{A}$ by setting the initial state of $\mathbf{A}$ to q.

2.3.3 Runs

The computational behavior of parity tree automata is explained using the notion of a run. Assume $\mathbf{A}$ is a parity tree automaton and $(\mathbf{K}, w_I)$ a pointed Kripke structure. A *run* of $\mathbf{A}$ on $(\mathbf{K}, w_I)$ is a $(W \times Q)$-vertex-labelled tree

$$\mathbf{r} = (V_r, E_r, \lambda_r) \tag{4}$$

such that $\rho_{\mathbf{r}}$ is labelled (w_I, q_I) and for every vertex v with label (w, q) the following conditions are satisfied.

- $\delta(q) \neq \bot$.
- If $\delta(q) = p$, then $p \in \lambda_{\mathbf{K}}(w)$, and if $\delta(q) = \neg p$, then $p \notin \lambda_{\mathbf{K}}(w)$.
- If $\delta(q) = q'$, then there exists $v' \in \mathrm{Suc}_r(v)$ such that $\lambda_r(v') = (w, q')$.
- If $\delta(q) = \Diamond q'$, then there exists $v' \in \mathrm{Suc}_r(v)$ such that $\lambda_r(v') = (w', q')$ for some $w' \in \mathrm{Suc}_{\mathbf{K}}(w)$.
- If $\delta(q) = \Box q'$, then for every $w' \in \mathrm{Suc}_{\mathbf{K}}(w)$ there exists $v' \in \mathrm{Suc}_r(v)$ with $\lambda_r(v') = (w', q')$.
- If $\delta(q) = q' \vee q''$, then there exists $v' \in \mathrm{Suc}_r(v)$ such that $\lambda(v') = (w, q')$ or $\lambda(v') = (w, q'')$.
- If $\delta(q) = q' \wedge q''$, then there exist $v', v'' \in \mathrm{Suc}_r(v)$ such that $\lambda(v') = (w, q')$ and $\lambda(v'') = (w, q'')$.

The run is accepting if the state labeling of every infinite branch through $\mathbf{r}$ satisfies the parity acceptance condition determined by Ω. This is formalized as follows.

For a node v in $\mathbf{r}$ with $\lambda_r(v) = (w, q)$, define $\Omega(v) := \Omega_A(q)$. An infinite branch $\pi = v_0 v_1 v_2 \cdots$ of $\mathbf{r}$ is *accepting* if the maximum priority occurring infinitely often in the sequence $\Omega(v_0)\Omega(v_1)\Omega(v_2)\cdots$ is even. The run $\mathbf{r}$ is *accepting* if every infinite branch through $\mathbf{r}$ is accepting.

In other words, for every infinite branch π we consider the sequence of natural numbers that is obtained from π by extracting the state component of the labelings of the vertices and applying the priority function $\Omega_{\mathbf{A}}$; we require that the maximum natural number occurring infinitely often is even.

A pointed Kripke structure is *accepted* by $\mathbf{A}$ if there exists an accepting run of $\mathbf{A}$ on the Kripke structure. The set of all Kripke *trees* accepted by $\mathbf{A}$ is denoted by $T(\mathbf{A})$.

An L_μ formula ϕ is *equivalent* to a parity tree automaton $\mathbf{A}$ if every pointed Kripke structure that holds in ϕ is accepted by $\mathbf{A}$ and vice versa. The following correspondence between μ-calculus and parity tree automata is well-known:

Fact 1 *Every Σ_i-formula, $i \geq 1$, is equivalent (via linear time reduction) to a parity tree automaton with priorities in $\{1, \ldots, i\}$ if i is odd and $\{0, \ldots, i-1\}$ if i is even. Conversely, every parity tree automaton with priorities in one of these sets is equivalent to a Σ_i-formula. An analogous correspondence holds for Π_i-formulas and parity tree automata with priority sets $\{0, \ldots, i-1\}$ if i is odd and $\{1, \ldots, i\}$ if i is even.*

In what follows, the parity tree automata corresponding to Σ_1-formulas are called 1-*automata* and those corresponding to Π_1-formulas are 0-*automata*. Note that in 1-automata all states have priority 1 and in 0-automata all states have priority 0.

2.3.4 Complex Transition Conditions

In some situations more complex transitions than those defined above are convenient. The most general option is to allow any modal formula over $\mathcal{P}$ and S as a transition condition. (An adaption of the semantics is straightforward.) It is in fact easy to see that even such a rich syntax does not lead to a more powerful automaton model: every such automaton can be turned into

an automaton according to our definition by introducing additional states, namely a new state for every subformula of a complex transition condition, and adapting the transition condition appropriately.

So, henceforth, we will—without loss of generality—allow complex transition conditions.

2.4 Main Result

We consider the following decision problem:

Σ_1-DEFINABILITY. Given an L_μ formula, decide whether there exists a Σ_1-formula equivalent to it.

Analogously, the problem Π_1-DEFINABILITY is defined. The main purpose of this paper is to prove:

Theorem 1 Σ_1-DEFINABILITY *is an EXPTIME-complete problem.*

Since ϕ belongs to Σ_1 iff $\neg\phi$ belongs to Π_1, from the theorem we immediately obtain:

Corollary 1 Π_1-DEFINABILITY *is an EXPTIME-complete problem.*

Overview of the proof of Theorem 1. We first note that to prove the theorem it suffices to consider Kripke trees instead of arbitrary Kripke structures since an L_μ formula ϕ holds at a world w of a Kripke structure $\mathbf{K}$ iff ϕ holds at the root of the tree obtained by unraveling $\mathbf{K}$ at w.

The complexity lower bound claimed in Theorem 1 is obtained by reducing the problem L_μ-TAUTOLOGY, which is known to be EXPTIME-complete, to Σ_1-DEFINABILITY (see Section 7).

The most part of this paper is devoted to the proof of the complexity upper bound. The idea is as follows.

Due to Fact 1, we can assume that in the problem Σ_1-DEFINABILITY we are given a parity tree automaton $\mathbf{A}$ instead of an L_μ formula.

Given $\mathbf{A}$ over $P \subseteq \mathcal{P}$, we define a bottom-up tree automaton $\mathbf{B_A}$ accepting the so-called Σ_1-test language $\Sigma_1(\mathbf{A})$. A state of $\mathbf{B_A}$ is an equivalence class of a so-called characteristic equivalence relation of $\mathbf{A}$. The key to deciding Σ_1-DEFINABILITY is a theorem stating that $T(\mathbf{A})$ is Σ_1-definable iff

$T(\mathbf{A}) \subseteq \Sigma_1(\mathbf{A})$; the inclusion $\Sigma_1(\mathbf{A}) \subseteq T(\mathbf{A})$ holds in any case. This reduces Σ_1-Definability to the emptiness problem

$$T(\mathbf{A}) \cap (\mathcal{T}_P \setminus \Sigma_1(\mathbf{A})) \stackrel{?}{=} \emptyset \tag{5}$$

We show that the languages accepted by bottom-up automata are definable in monadic second order logic (MSOL). Consequently, the above emptiness problem, and thus, Σ_1-Definability is decidable. To obtain the claimed exponential upper bound, we prove that the complement of languages accepted by bottom-up tree automata are accepted by so-called top-down tree automata of the same size. A detailed analysis of standard emptiness tests for parity tree automata then shows that the above emptiness problem, where the complement of the Σ_1-test language is given as a top-down tree automaton, is decidable in exponential time.

In the following section, the characteristic equivalence relation is introduced. In Section 4, we define bottom-up tree automata and state important properties. The Σ_1-test language and the mentioned theorem can be found in Section 5. Section 6 studies the complement of bottom-up tree automata and introduces top-down tree automata. Finally, in Section 7 everything is put together to prove the complexity upper bound, as well as the complexity lower bound.

3 Characteristic Equivalence Relations

In this section, we introduce the characteristic equivalence relation of a parity tree automaton and study properties thereof.

If $\equiv$ is an equivalence relation on trees and t is a tree, then $[t]_\equiv := \{t' \mid t' \equiv t\}$ denotes the equivalence class of t w.r.t. $\equiv$. We say that $\equiv$ *saturates a tree set* T if T is a union of its classes.

Let $\mathbf{A}$ be a parity tree automaton recognizing T. We define its *characteristic equivalence relation,* denoted $\equiv_A$, as follows. It considers trees t and t' equivalent if for every state q of $\mathbf{A}$: $t \in T(\mathbf{A}_q)$ iff $t' \in T(\mathbf{A}_q)$. Let $C_\mathbf{A} := \{[t]_{\equiv_\mathbf{A}} \mid t \text{ is a tree } \}$ denote the set of equivalence classes w.r.t. $\equiv_\mathbf{A}$.

We summarize some simple properties.

Lemma 1 *Let* $\mathbf{A}$ *be an arbitrary tree automaton and* T *the tree set recognized by it.*

1. *The index of the characteristic equivalence relation* $\equiv_{\mathbf{A}}$ *can be bounded exponentially in the size of* $\mathbf{A}$.

2. *The characteristic equivalence relation of* $\mathbf{A}$ *saturates* T.

There is a more complicated property that the characteristic equivalence relation of an automaton enjoys: the equivalence class of the tree itself is determined by the labeling of the root and the equivalence classes of the subtrees rooted at the children of the root. Equivalence relations with this property are called strong equivalence relations.

An equivalence relation $\equiv$ on trees is a *strong equivalence relation* if it satisfies the following condition. Trees t and t' are equivalent w.r.t. $\equiv$ if

$$\lambda_t(\rho_{\mathbf{t}}) = \lambda_{t'}(\rho_{\mathbf{t}'}) \tag{6}$$

and

$$\{[t \downarrow w]_{\equiv} \mid w \in \mathrm{Suc}_t(\rho_{\mathbf{t}})\} = \{[t \downarrow w]_{\equiv} \mid w \in \mathrm{Suc}_{t'}(\rho_{\mathbf{t}'})\}. \tag{7}$$

Lemma 2 *The characteristic equivalence relation of any parity tree automaton is a strong equivalence relation. (This is even true for tree automata with infinite state spaces and infinitary transition conditions.)*

Proof. Let t and t' be trees satisfying (6) and (7). We want to show that $t \equiv_{\mathbf{A}} t'$, i.e., for every state q of $\mathbf{A}$: $t \in T(\mathbf{A}_q)$ iff $t' \in T(\mathbf{A}_q)$.

Assume $t \in T(\mathbf{A}_q)$. Let r be a run of $\mathbf{A}_q$ on t. We construct a run r' of $\mathbf{A}_q$ on t'. To this end, define $V_{\rho_{\mathbf{t}}} := \{v \in V_r \mid \lambda_r(v) = (\rho_{\mathbf{t}}, q) \text{ for some state } q\}$ and $V_{\mathrm{Suc}_t(\rho_{\mathbf{t}})} := \{v \in V_r \mid \lambda_r(v) = (w, q) \text{ for some state } q \text{ and } w \in \mathrm{Suc}_t(\rho_{\mathbf{t}})\}$.

Because of (7) there exists a mapping γ from $\{\rho_{\mathbf{t}}\} \cup \mathrm{Suc}_t(\rho_{\mathbf{t}})$ into $\{\rho_{\mathbf{t}'}\} \cup \mathrm{Suc}_{t'}(\rho_{\mathbf{t}'})$ such that $\gamma(\rho_{\mathbf{t}}) = \rho_{\mathbf{t}'}$, $\gamma(w) \in \mathrm{Suc}_{t'}(\rho_{\mathbf{t}'})$, and $[w]_{\equiv_{\mathbf{A}}} = [\gamma(w)]_{\equiv_{\mathbf{A}}}$ for every $w \in \mathrm{Suc}_t(\rho_{\mathbf{t}})$.

Define $\hat{r} := (V_{\rho_{\mathbf{t}}} \cup V_{\mathrm{Suc}_t(\rho_{\mathbf{t}})}, ((V_{\rho_{\mathbf{t}}} \times V_{\rho_{\mathbf{t}}}) \cup (V_{\rho_{\mathbf{t}}} \times V_{\mathrm{Suc}_t(\rho_{\mathbf{t}})})) \cap E_r, \lambda)$, where $\lambda(v) := (\rho_{\mathbf{t}'}, q')$ if $v \in V_{\rho_{\mathbf{t}}}$ and $\lambda_r(v) = (\rho_{\mathbf{t}}, q')$ for some q', and $\lambda(v) := (\gamma(w), q')$ if $v \in V_{\mathrm{Suc}_t(\rho_{\mathbf{t}})}$ and $\lambda_r(v) = (w, q')$ for some w and q'. We assume that all nodes of $\hat{r}$ not reachable from $\rho_{\hat{\mathbf{r}}}$ are removed.

We extend $\hat{r}$ to r' as follows. Every node v in $\hat{r}$ labelled $(\gamma(w), q')$ is replaced by a run of $\mathbf{A}_{q'}$ on $t' \downarrow \gamma(w)$. Such a run exists since $r \downarrow v$ is a run of $\mathbf{A}_{q'}$ on $t \downarrow w$ and $[w]_{\equiv_{\mathbf{A}}} = [\gamma(w)]_{\equiv_{\mathbf{A}}}$. It is easy to see that r' thus obtained is a run of $\mathbf{A}_q$ on t'. Hence, $t' \in T(\mathbf{A}_q)$.

By symmetry, we conclude that $t' \in T(\mathbf{A}_q)$ implies $t \in T(\mathbf{A}_q)$. □

This lemma is the key to the definition of the transition function of our bottom-up tree automaton accepting the Σ_1-test language, for it tells us the following. For every parity automaton $\mathbf{A}$ over P there exists a unique function $f_A \colon 2^{C_A} \times 2^P \to C_A$ such that the following holds.

Given a tree t and the sets

$$P' = \lambda_t(\rho_{\mathbf{t}}),$$
$$C = \{[t \downarrow w]_{\equiv_{\mathbf{A}}} \mid w \in \mathrm{Suc}_t(\rho_{\mathbf{t}})\},$$

we have

$$[t]_{\equiv_{\mathbf{A}}} = f_{\mathbf{A}}(C, P').$$

4 Bottom-up automata on infinite trees

In this section, we introduce bottom-up tree automaton (Section 4.1 and 4.2). In Section 4.3, we characterize the run of a bottom-up tree automaton as a fixed point of a certain mapping. This will later be used to perform proofs by transfinite induction. In Section 4.4 and 4.5 we investigate the Σ_1- and MSOL-definability of languages accepted by bottom-up tree automata.

4.1 Informal Description

Bottom-up tree automata are designed to accept or reject (infinite) trees. Given a tree, they first guess a frontier in the tree and assign to every node of this frontier a certain set of states, depending on the label of the node and on whether the node is a leaf or not. Then in a bottom-up powerset fashion the ancestors are labelled with sets of states according to the transition function. (All descendants of nodes of the frontier are labelled with the empty set.) A tree is accepted if the set of states the root is labelled with is non-empty and a subset of the set of final states of the automaton.

4.2 Definition

Formally, a *bottom-up tree automaton* $\mathbf{B}$ is a tuple (Q, P, F, δ), where

- Q is a finite set of states,
- P is a finite set of propositional variables,

- $F \subseteq Q$ is the set of final states, and
- $\delta\colon 2^Q \times 2^P \to Q$ is the transition function.

To define a run of $\mathbf{B}$ on a tree $t \in \mathcal{T}_P$, we need two additional functions, $\gamma\colon 2^P \to Q$ and $\Gamma\colon 2^P \to 2^Q$:

$$\begin{aligned} \gamma(P') &= \delta(\emptyset, P'), \\ \Gamma(P') &= \{\delta(Q', P') \mid Q' \subseteq Q \wedge Q' \neq \emptyset\}. \end{aligned}$$

The former function is used to determine the set of states assigned to leaves belonging to the frontier and the latter function determines the set of states assigned to inner nodes of the frontier. In other words, inner nodes are labeled with the set of all states that this node can possibly take according to the transition function δ.

We also define the function

$$\Delta\colon 2^{2^Q} \times 2^P \to 2^Q,$$

which is the "powerset variant" of δ: A state $q \in Q$ belongs to $\Delta(\mathcal{Q}, P')$ if there exists a set E and a selection function $s\colon \mathcal{Q} \to Q$ with $s(Q') \in Q'$, for every $Q' \in \mathcal{Q}$, $E = \{s(Q') \mid Q' \in \mathcal{Q}\}$, and $q = \delta(E, P')$.

A *frontier* $\mathcal{F}$ in t is a subset of W_t such that

- there does not exist a path between two different worlds in $\mathcal{F}$ ($\mathcal{F}$ is an anti-chain), and
- every maximum path in t starting from $\rho_{\mathbf{t}}$ contains a node in $\mathcal{F}$.

If $\mathcal{F}$ is a frontier, the set $\hat{\mathcal{F}}$ shall denote the set of ancestors of worlds in $\mathcal{F}$, including the worlds in $\mathcal{F}$.

Now, a *run* on t is a function $\beta\colon W_t \to 2^Q$ which has the following properties: There exists a frontier $\mathcal{F}$ in t such that

1. when $w \in \mathcal{F}$ is a leaf, then $\beta(w) = \{\gamma(\lambda_t(w))\}$,
2. when $w \in \mathcal{F}$ is an inner node, then $\beta(w) = \Gamma(\lambda_t(w))$,
3. when $w \in \hat{\mathcal{F}} \setminus \mathcal{F}$, then $\beta(w) = \Delta(\{\beta(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w))$,
4. when $w \notin \hat{\mathcal{F}}$, then $\beta(w) = \emptyset$.

A run β on t is called *accepting* if $\beta(\rho_{\mathbf{t}}) \neq \emptyset$ and $\beta(\rho_{\mathbf{t}}) \subseteq F$. The set of trees accepted by $\mathbf{B}$ is denoted $T(\mathbf{B}) := \{t \in \mathcal{T}_P \mid \text{there exists an accepting run of } \mathbf{B} \text{ on } t\}$.

Note that in the definition of a run, the frontier $\mathcal{F}$ is uniquely determined: Assume that there exist two distinct frontiers $\mathcal{F}$ and $\mathcal{F}'$ satisfying 1.–4. Then, w.l.o.g., there exists $w \in \mathcal{F} \setminus \mathcal{F}'$. Consider a maximum path from $\rho_{\mathbf{t}}$ that goes through w. If on the subpath from $\rho_{\mathbf{t}}$ to w, there exists a world in $\mathcal{F}'$, then this implies $\beta(w) = \emptyset$, in contradiction to $w \in \mathcal{F}$ and condition 1.–2. Conversely, if on the subpath starting from w there is a world $w' \in \mathcal{F}'$, then $\beta(w') = \emptyset$, in contradiction to $w' \in \mathcal{F}'$ and condition 1.–2. Thus, $\mathcal{F}$ is uniquely determined. In what follows, given β we refer to $\mathcal{F}$ by $\mathcal{F}_\beta$; $\hat{\mathcal{F}}_\beta$ shall denote the set of ancestors of $\mathcal{F}_\beta$ including the worlds in $\mathcal{F}_\beta$.

4.3 Alternative Definition of a Run

We show that a run of a bottom-up tree automaton can be described as a fixed point of a mapping $\mathcal{R}$.

Let $\mathbf{B} = (Q, P, F, \delta)$ be a bottom-up tree automaton, t be a tree, and β be a run of $\mathbf{B}$ on t.

Given a mapping $\alpha\colon W_t \to 2^Q$, the mapping $\mathcal{R}$ takes α to a mapping $\alpha'\colon W_t \to 2^Q$ such that, for every $w \in W_t$, $\alpha'(w) := \alpha(w)$ if w is a leaf and $\alpha'(w) := \alpha(w) \cup \Delta(\{\alpha(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w))$ otherwise. Obviously $\mathcal{R}$ is monotone, i.e., $\alpha(w) \subseteq \alpha'(w)$ for every $w \in W_t$. We define $\mathcal{R}^0(\alpha) := \alpha$, $\mathcal{R}^{\tau+1}(\alpha) := \mathcal{R}(\mathcal{R}^\tau(\alpha))$ for every ordinal τ, and $\mathcal{R}^\tau(\alpha)(w) := \bigcup_{\tau' < \tau} \mathcal{R}^{\tau'}(\alpha)(w)$ for every limit ordinal τ and $w \in W_t$. For notational convenience we write $\mathcal{R}^\tau_\alpha$ instead of $\mathcal{R}^\tau(\alpha)$.

Let α_β be the mapping defined as follows:

$$\alpha_\beta(w) := \beta(w) \text{ if } w \in \mathcal{F}_\beta \text{ and } \alpha_\beta(w) := \emptyset \text{ otherwise .}$$

Since $\mathcal{R}$ is monotone, there exists an ordinal ξ such that $\mathcal{R}^\xi_{\alpha_\beta} = \mathcal{R}^{\xi+1}_{\alpha_\beta}$, i.e., $\mathcal{R}^\xi_{\alpha_\beta}$ is a fixed point of $\mathcal{R}$. We will show:

Proposition 1 $\beta = \mathcal{R}^\xi_{\alpha_\beta}$.

This proposition follows immediately from the next two lemmas.

First, define by transfinite induction $D_0 := \mathcal{F}_\beta$, $D_{\tau+1} := D_\tau \cup \{w \mid \mathrm{Suc}_t(w) \subseteq D_\tau \wedge \mathrm{Suc}_t(w) \neq \emptyset\}$, and $D_\tau := \bigcup_{\tau' < \tau} D_{\tau'}$ for a limit ordinal τ. Let ξ' be the closure ordinal, i.e., $D_{\mathcal{F}_\beta} := D_{\xi'} = D_{\xi'+1}$.

Lemma 3 $\hat{\mathcal{F}}_\beta = D_{\mathcal{F}_\beta}$.

PROOF. By transfinite induction on τ, one easily shows that $D_\tau \subseteq \hat{\mathcal{F}}_\beta$ for every ordinal τ. Assume that there exists $w \in \hat{\mathcal{F}}_\beta \setminus D_{\mathcal{F}_\beta}$. We can conclude that neither w nor one of its ancestors belongs to $\mathcal{F}_\beta$. If w is a leaf, then $w \in \mathcal{F}_\beta = D_0$, since $\mathcal{F}_\beta$ is a frontier, and thus, every maximum path from the root must contain a world in $\mathcal{F}_\beta$. This contradicts the assumption that $w \notin D_{\mathcal{F}_\beta}$. Hence, w is an inner node. Because $w \notin D_{\mathcal{F}_\beta}$, we can conclude that there exists a $w' \in \mathrm{Suc}_t(w)$ with $w' \notin D_{\mathcal{F}_\beta}$, in particular, w' and all its ancestors do not belong to $\mathcal{F}_\beta$. Iterating this argument we find a maximum path in t starting from the root where no world belongs to $\mathcal{F}_\beta$, in contradiction to the assumption that $\mathcal{F}_\beta$ is a frontier. □

Lemma 4 *For every ordinal τ and $w \in W_t$, if $w \in D_\tau$, then $\mathcal{R}^\tau_{\alpha_\beta}(w) = \beta(w)$, otherwise if $w \notin D_\tau$, then $\mathcal{R}^\tau_{\alpha_\beta}(w) = \emptyset$.*

PROOF. The proof is by transfinite induction on τ. For $\tau = 0$ the claim holds by the definition of α_β.

Now, assume that $w \in D_{\tau+1}$. If w is a leaf, then $w \in D_0 = \mathcal{F}_\beta$, and thus, $\mathcal{R}^{\tau+1}_{\alpha_\beta}(w) = \alpha_\beta(w) = \beta(w)$. Next, assume that w is an inner node. By definition, $\mathcal{R}^{\tau+1}_{\alpha_\beta}(w) = \mathcal{R}^\tau_{\alpha_\beta}(w) \cup \Delta(\{\mathcal{R}^\tau_{\alpha_\beta}(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w))$. We know $w' \in D_\tau$ for every $w' \in \mathrm{Suc}_t(w)$. Induction yields, $\mathcal{R}^\tau_{\alpha_\beta}(w') = \beta(w')$ for every $w' \in \mathrm{Suc}_t(w)$. Hence, $\Delta(\{\mathcal{R}^\tau_{\alpha_\beta}(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w)) = \beta(w)$. If $w \notin D_\tau$, then, by induction, $\mathcal{R}^\tau_{\alpha_\beta}(w) = \emptyset$. Otherwise, $\mathcal{R}^\tau_{\alpha_\beta}(w) = \beta(w)$. Altogether, $\mathcal{R}^{\tau+1}_{\alpha_\beta}(w) = \beta(w)$.

Assume $w \notin D_{\tau+1}$. If w is a leaf, then $w \notin D_0 = \mathcal{F}_\beta$, and thus, $\mathcal{R}^{\tau+1}_{\alpha_\beta}(w) = \alpha_\beta(w) = \emptyset$. Otherwise, w is an inner node, and by definition, $\mathcal{R}^{\tau+1}_{\alpha_\beta}(w) = \mathcal{R}^\tau_{\alpha_\beta}(w) \cup \Delta(\{\mathcal{R}^\tau_{\alpha_\beta}(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w))$. Because $w \notin D_\tau$, induction yields $\mathcal{R}^\tau_{\alpha_\beta}(w) = \emptyset$. Moreover, there exists $w' \in \mathrm{Suc}_t(w)$ such that $w' \notin D_\tau$. Thus, $\mathcal{R}^\tau_{\alpha_\beta}(w') = \emptyset$. Consequently, $\Delta(\{\mathcal{R}^\tau_{\alpha_\beta}(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w)) = \emptyset$. Altogether, $\mathcal{R}^{\tau+1}_{\alpha_\beta}(w) = \emptyset$.

Next assume that τ is a limit ordinal and $w \in D_\tau$. Then, there exists a $\tau' < \tau$ such that $w \in D_{\tau'}$. Induction yields, $\mathcal{R}^{\tau'}_{\alpha_\beta}(w) = \beta(w)$. Moreover, by induction, $\mathcal{R}^{\tau''}_{\alpha_\beta}(w)$ is either $\emptyset$ or $\beta(w)$ for every $\tau'' < \tau$. Thus, $\mathcal{R}^\tau_{\alpha_\beta}(w) = \beta(w)$.

Finally, if $w \notin D_\tau$, for a limit ordinal τ, then $w \notin D_{\tau'}$, and thus, $\mathcal{R}^{\tau'}_{\alpha_\beta}(w) = \emptyset$ for every $\tau' < \tau$. Consequently, $\mathcal{R}^\tau_{\alpha_\beta}(w) = \emptyset$. □

We will now show how $\mathcal{R}$ can be used to construct a run on a tree t starting from a valuation of a frontier of t consistent with 1. and 2. in the definition of a run.

Let $\mathcal{F}$ be a frontier of t and let $\alpha_{\mathcal{F}} \colon W_t \to 2^Q$ be defined as follows: $\alpha_{\mathcal{F}}(w) := \{\gamma(\lambda_t(w))\}$ if $w \in \mathcal{F}$ is a leaf, $\alpha_{\mathcal{F}}(w) := \Gamma(\lambda_t(w))$ if $w \in \mathcal{F}$ is an inner node, and $\alpha_{\mathcal{F}}(w) := \emptyset$ if $w \notin \mathcal{F}$. We know that there exists ξ such that $\mathcal{R}^{\xi}_{\alpha_{\mathcal{F}}}$ is a fixed point of $\mathcal{R}$.

Proposition 2 *$\mathcal{R}^{\xi}_{\alpha_{\mathcal{F}}}$ is a run on t and $\mathcal{R}^{\xi}_{\alpha_{\mathcal{F}}}(\rho_{\mathbf{t}}) \neq \emptyset$.*

Proof. One easily shows by transfinite induction that $\mathcal{R}^{\tau}_{\alpha_{\mathcal{F}}}(w) = \emptyset$, for every ordinal τ and every $w \in W_t \setminus \hat{\mathcal{F}}$. Thus, condition 4. in the definition of runs is satisfied, with β replaced by $\mathcal{R}^{\xi}_{\alpha_{\mathcal{F}}}$. It is also easy to see that conditions 1. and 2. are met.

For condition 3., one first shows by transfinite induction that $\mathcal{R}^{\tau}_{\alpha_{\mathcal{F}}}(w) = \mathcal{R}^{\xi}_{\alpha_{\mathcal{F}}}(w)$, for every ordinal τ and every $w \in D_\tau$. With this, it easily follows by transfinite induction (*): $\mathcal{R}^{\tau}_{\alpha_{\mathcal{F}}}(w) = \Delta(\{\mathcal{R}^{\xi}_{\alpha_{\mathcal{F}}}(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w))$, for every ordinal $\tau > 0$ and every $w \in D_\tau$.

Finally, $\mathcal{R}^{\xi}_{\alpha_{\mathcal{F}}}(\rho_{\mathbf{t}}) \neq \emptyset$ follows by transfinite induction using (*) as well as $\mathcal{R}^{\tau}_{\alpha_{\mathcal{F}}}(w) \neq \emptyset$ for every $w \in D_0$. □

4.4 Σ_1-definability of Bottom-up Tree Automata

In general, the languages accepted by bottom-up tree automata are not closed under bisimulation, and thus, are not definable in L_μ; this is even true for bottom-up tree automata induced by parity tree automata. In Section 6.4 we show an example for top-down tree automata. Using Corollary 3 this carries over to bottom-up tree automata. Nevertheless, we show the following.

Proposition 3 *Whenever the language accepted by a bottom-up tree automaton is closed under bisimulation, then it is Σ_1-definable.*

Proof. Let $\mathbf{B} = (Q, P, F, \delta)$ be a bottom-up tree automaton and assume that $T(\mathbf{B})$ is closed under bisimulation. We define a 1-automaton $\mathbf{A} = (Q_{\mathbf{A}}, q_I, \delta_{\mathbf{A}}, \Omega_{\mathbf{A}})$ accepting $T(\mathbf{B})$, where q_I is a new state, $Q_{\mathbf{A}} := (2^Q \setminus \{\emptyset\}) \cup \{q_I\}$, $\Omega_{\mathbf{A}}$ assigns to every state priority 1, and $\delta_{\mathbf{A}}$ is defined as follows: For $P' \subseteq P$, let $\bigwedge P' := \bigwedge_{p \in P'} p \wedge \bigwedge_{p \notin P'} \neg p$. Now,

$$\delta_{\mathbf{A}}(q_I) := \bigvee_{\substack{q \subseteq 2^F \\ q \neq \emptyset}} q,$$

and for every $q \in Q_{\mathbf{A}} \setminus \{q_I\}$,

$$\begin{aligned}\delta_{\mathbf{A}}(q) \;:=\; & \bigvee_{\substack{P' \subseteq P \\ q = \{\gamma(P')\}}} (\Box\bot \wedge \bigwedge P') \vee \\ & \bigvee_{\substack{P' \subseteq P \\ q = \Gamma(P')}} (\Diamond\top \wedge \bigwedge P') \vee \\ & \bigvee_{\substack{P' \subseteq P \\ \mathcal{Q} \subseteq Q_{\mathbf{A}} \setminus \{q_I\}, \mathcal{Q} \neq \emptyset \\ q = \Delta(\mathcal{Q}, P')}} \bigwedge P' \wedge \Box \bigvee_{q' \in \mathcal{Q}} q' \wedge \bigwedge_{q' \in \mathcal{Q}} \Diamond q'.\end{aligned}$$

It remains to show that $T(\mathbf{B}) = T(\mathbf{A})$. We first show the inclusion $T(\mathbf{B}) \subseteq T(\mathbf{A})$, which holds independently of whether $T(\mathbf{B})$ is closed under bisimulation or not.

Let $t \in T(\mathbf{B})$ and let β be an accepting run of $\mathbf{B}$ on t. Let $\mathcal{F}_\beta$ be the frontier corresponding to β and $\hat{\mathcal{F}}_\beta$ be the set of ancestors of worlds in $\mathcal{F}_\beta$ including $\mathcal{F}_\beta$.

Define $r_\beta := (\hat{\mathcal{F}}_\beta \cup \{\rho\}, (E_t \cap \hat{\mathcal{F}}_\beta \times \hat{\mathcal{F}}_\beta) \cup \{(\rho, \rho_{\mathbf{t}})\}, \lambda)$, where $\lambda(\rho) := (\rho, q_I)$ and $\lambda(w) := (w, \beta(w))$ for every $w \in \hat{\mathcal{F}}_\beta$. It is easy to verify that r_β is a run of $\mathbf{A}$ on t. Thus, $t \in T((\mathbf{A})$.

Now we show $T(\mathbf{A}) \subseteq T(\mathbf{B})$. Assume $t \in T(\mathbf{A})$ and let $r = (V_r, E_r, \lambda_r)$ be an accepting run of $\mathbf{A}$ on t. We assume r to be minimal, i.e., no proper subtree of r containing $\rho_{\mathbf{r}}$ is an accepting run. We state important properties of r:

(I) There exists exactly one successor v of $\rho_{\mathbf{r}}$ and its label is $\lambda_r(v) = (\rho_{\mathbf{t}}, q)$, where $q \subseteq 2^F$ with $q \neq \emptyset$.

(II) For every leaf v of r with $\lambda_r(v) = (w, q)$, one of the following properties holds:

  (a) $t \downarrow w \models \Box\bot \wedge \bigwedge P'$ for some $P' \subseteq P$ and $q = \{\gamma(P')\}$.

  (b) $t \downarrow w \models \Diamond\top \wedge \bigwedge P'$ for some $P' \subseteq P$ and $q = \Gamma(P')$.

(III) For every inner node v, $v \neq \rho_{\mathbf{r}}$, with $\lambda_r(v) = (w, q)$ for some w and q, there exists $P' \subseteq P$ and $\mathcal{Q} \subseteq Q_{\mathbf{A}} \setminus \{q_I\}$ with $\mathcal{Q} \neq \emptyset$ such that

  (a) $\lambda_t(w) = P'$,

(b) for every $w' \in \mathrm{Suc}_t(w)$, there exists $v' \in \mathrm{Suc}_r(v)$ with $\lambda_r(v') = (w', q')$ for some $q' \in \mathcal{Q}$,

(c) for every $q' \in \mathcal{Q}$, there exists $v' \in \mathrm{Suc}_r(v)$ with $\lambda_r(v') = (w', q')$ for some $w' \in \mathrm{Suc}_t(w)$, and

(d) because of r's minimality, for every $v' \in \mathrm{Suc}_r(v)$ with $\lambda_r(v') = (w', q')$ for some w': $q' \in \mathcal{Q}$.

(IV) The minimality of r also implies: For every inner node v, $v \neq \rho_{\mathbf{r}}$, with $\lambda_r(v) = (w, q)$, for some w and q, and every $v' \in \mathrm{Suc}_r(v)$ with $\lambda_r(v) = (w', q')$, for some w' and q', it holds that $w' \in \mathrm{Suc}_t(w)$.

Let $t_r := (V_r \setminus \{\rho_{\mathbf{r}}\}, E_r \cap (V_r \setminus \{\rho_{\mathbf{r}}\} \times V_r \setminus \{\rho_{\mathbf{r}}\}), \lambda)$ with $\lambda(v) := \lambda_t(w)$ if $\lambda_r(v) = (w, q)$ for some w and q. Note that, due to (I), t_r is a tree, and every path in t_r is finite. Further, let t'_r be the tree obtained from t_r, by replacing every leaf v of t_r with $\lambda_r(v) = (w, q)$ for some w and q by a fresh copy of $t \downarrow w$. We claim that $t'_r \cong t$ and $t'_r \in T(\mathbf{B})$. Since $T(\mathbf{B})$ is bisimulation closed, this would imply that $t \in T(\mathbf{B})$. It remains to prove the claim.

First we define the set $\mathcal{F} := \{w \mid \text{there exists a leaf } v \text{ in } t_r \text{ with } \lambda_r(v) = (w, q) \text{ for some } q\}$. Let $\hat{\mathcal{F}}$ denote the set of ancestors in t of worlds in $\mathcal{F}$ including $\mathcal{F}$. Also, define $\mathcal{F}_r := \{v \mid v \text{ is a leaf of } t_r\}$ and let $\hat{\mathcal{F}}_r$ denote the set of ancestors of worlds in $\mathcal{F}_r$ including $\mathcal{F}_r$. In other words, $\hat{\mathcal{F}}_r$ is the set of worlds of t_r.

Claim I. $t'_r \cong t$.

Proof of Claim I. We first show that t_r and $t|_{\hat{\mathcal{F}}}$ are bisimilar, where $t|_{\hat{\mathcal{F}}}$ is the subtree of t which only contains worlds in $\hat{\mathcal{F}}$. The bisimulation relation $R \subseteq W_{t_r} \times W_{t|_{\hat{\mathcal{F}}}}$ between t_r and $t|_{\hat{\mathcal{F}}}$ is defined as follows: $(v, w) \in R$ iff $\lambda_r(v) = (w, q)$ for some q. Note that due to (IV), if $\lambda_r(v) = (w, q)$, then $w \in \hat{\mathcal{F}}$. We show that R satisfies the required conditions. First note that $(\rho_{\mathbf{t_r}}, \rho_{\mathbf{t}}) \in R$ due to (I). Now, assume that $(v, w) \in R$, i.e., $\lambda_r(v) = (w, q)$ for some q.

- By definition of R, $\lambda_{t_r}(v) = \lambda_t(w)$.
- Assume $(v, v') \in E_{t_r}$ with $\lambda_r(v') = (w', q')$. From (IV) it follows that $w' \in \mathrm{Suc}_t(w)$.
- Assume $(w, w') \in E_t$. From (III), (b) it follows that there exists $(v, v') \in E_{t_r}$ with $\lambda_r(v') = (w', q')$ for some q'.

Now it is straightforward to prove that t'_r and t are bisimilar. This completes the proof of Claim I.

Claim II. $t'_r \in T(\mathbf{B})$.

Proof of Claim II. We construct a mapping $\beta\colon W_{t'_r} \to 2^Q$ of $\mathbf{B}$ on t'_r as follows: For every $v \in \hat{\mathcal{F}}_r$, $\beta(v) := q$ if $\lambda_r(v) = (w, q)$ for some w and q. For every $v \notin \hat{\mathcal{F}}_r$, $\beta(v) := \emptyset$. The frontier $\mathcal{F}_\beta$ of β is $\mathcal{F}_r$. Using (I), (II), and (III), it follows that β is an accepting run. This completes the proof of Claim II. □

4.5 MSOL-definability of Bottom-up Tree Automata

A tree set T is *MSOL-definable* if there exists an MSOL-sentence φ_T such that $t \in T$ iff $t \models \varphi_T$ for every Kripke tree t. We show the following.

Proposition 4 *Every language accepted by a bottom-up tree automaton is MSOL-definable.*

Proof. Let $\mathbf{B} = (Q, P, F, \delta)$ be a bottom-up tree automaton. Let γ and Γ be defined as usual. We will define an MSOL-sentence $\varphi_{\mathbf{B}}$ such that $t \in T(\mathbf{B})$ iff $t \models \varphi_{\mathbf{B}}$.

We need some auxiliary MSOL formulas. First, we define the predicates leaf and inner node:

$$\begin{aligned} \mathsf{leaf}(x) &= \neg\exists y\ s(x, y), \\ \mathsf{inner}(x) &= \neg\mathsf{leaf}(x). \end{aligned}$$

A set of worlds closed under predecessors is defined as follows:

$$\mathsf{closed}(X) = \forall x \forall y\ \big(s(x, y) \wedge X(y) \to X(x)\big).$$

The set of ancestors X of a set of worlds Y is defined by the following formula:

$$\mathsf{anc}(X, Y) = Y \subseteq X \wedge \mathsf{closed}(X) \wedge \forall Z\big(Y \subseteq Z \wedge \mathsf{closed}(Z) \wedge Z \subseteq X \to Z = X\big).$$

The following formula describes a set of worlds that is closed under predecessors and every inner node in this set has a successor in this set.

$$\mathsf{maxclosed}(X) = \mathsf{closed}(X) \wedge \forall x\ \Big(X(x) \wedge \mathsf{inner}(x) \to \exists y\ \big(s(x, y) \wedge X(y)\big)\Big).$$

With this a maximum path X containing x can be defined as follows.

$$\begin{aligned}\mathsf{maxpath}(X,x) \;=\;& X(x) \wedge \mathsf{maxclosed}(X) \wedge \\ & \forall Y\big(Y(x) \wedge \mathsf{maxclosed}(Y) \wedge Y \subseteq X \to Y = X\big).\end{aligned}$$

To say that there exists a non-empty path from x to y we write

$$\mathsf{reach}(x,y) = \exists z \exists X\ \big(s(x,z) \wedge \mathsf{maxpath}(X,z) \wedge X(z) \wedge X(y)\big).$$

A frontier, as introduced in Section 4.2, is defined by

$$\begin{aligned}\mathsf{frontier}(X) \;=\;& \forall x \forall y\ \big(X(x) \wedge X(y) \to \neg\mathsf{reach}(x,y)\big) \wedge \\ & \forall Y\ \Big(\mathsf{maxpath}(Y,\mathsf{sr}) \to \exists x\ \big(Y(x) \wedge X(x)\big)\Big).\end{aligned}$$

We are prepared to specify $\varphi_{\mathbf{B}}$. It is defined to be the existential closure of $\varphi'_{\mathbf{B}}$; see below for an informal description. For $P' \subseteq P$, we define $P'(x) := \bigwedge_{p \in P'} p(x) \wedge \bigwedge_{p \notin P'} \neg p(x)$.

$$\begin{aligned}\varphi'_{\mathbf{B}} \;=\;& \forall x \bigvee_{q \in 2^Q} X_q(x)\ \wedge && (8)\\ & \forall x \bigwedge_{\substack{q,q' \in 2^Q \\ q' \neq q}} \neg(X_q(x) \wedge X_{q'}(x))\ \wedge && (9)\\ & \bigvee_{\substack{q \subseteq 2^F \\ q \neq \emptyset}} X_q(\mathsf{sr})\ \wedge && (10)\\ & \exists F \exists \hat{F}\Big(\mathsf{frontier}(F) \wedge \mathsf{anc}(\hat{F},F)\ \wedge && (11)\\ & \quad \forall x \bigwedge_{P' \subseteq P} \big(F(x) \wedge \mathsf{leaf}(x) \wedge P'(x) \to X_{\{\gamma(P')\}}(x)\big)\ \wedge && (12)\\ & \quad \forall x \bigwedge_{P' \subseteq P} \big(F(x) \wedge \mathsf{inner}(x) \wedge P'(x) \to X_{\Gamma(P')}(x)\big)\ \wedge && (13)\\ & \quad \forall x \bigwedge_{\substack{P' \subseteq P \\ Q' \subseteq 2^Q}} \big(\neg F(x) \wedge \hat{F}(x) \wedge P'(x) \wedge \mathsf{sucstates}(x,Q') && (14)\\ & \qquad\qquad \to X_{\Delta(Q',P')}(x)\big)\ \wedge \\ & \quad \forall x\big(\neg\hat{F}(x) \to X_\emptyset(x)\big)\Big), && (15)\end{aligned}$$

where

$$\mathsf{sucstates}(x, Q') \;=\; \forall y \,\big(s(x,y) \to \bigvee_{q \in Q'} X_q(y)\big) \;\wedge\; \bigwedge_{q \in Q'} \exists y \,\big(s(x,y) \wedge X_q(y)\big).$$

With the variables X_q, $q \in 2^Q$, we encode the run $\beta\colon W_t \to 2^Q$. In (8) and (9), we guarantee that every node in W_t is mapped to exactly one element of 2^Q. With (10), we say that the run is accepting. In (11) to (14), the conditions 1. to 3. in the definition of a run are described, where $\mathsf{sucstates}(x, Q')$ says that the set of states the successors of x are labelled with is Q'. Finally, condition 4. in the definition of a run is captured by (15).

It should be clear that $\varphi_{\mathbf{B}}$ describes the set of trees accepted by $\mathbf{B}$. □

5 Decidability Criterion

We consider the bottom-up tree automaton $\mathbf{B}_{\mathbf{A}} := (C_{\mathbf{A}}, P, \{[t]_{\equiv_{\mathbf{A}}} \mid t \in T(\mathbf{A})\}, f_{\mathbf{A}})$, where $f_{\mathbf{A}}$ is defined as at the end of Section 3. Let γ, Γ, and Δ be defined as in the previous section, where δ is replaced by $f_{\mathbf{A}}$. We call the language accepted by $\mathbf{B}_{\mathbf{A}}$ the Σ_1-*test language* of $\mathbf{A}$ and denote it by $\Sigma_1(\mathbf{A})$.

The following theorem provides the criterion for deciding Σ_1-DEFINABILITY.

Theorem 2 *Let* $\mathbf{A}$ *be a parity tree automaton. Then,*

$$T(\mathbf{A}) \textit{ is } \Sigma_1\textit{-definable} \quad \textit{iff} \quad T(\mathbf{A}) = \Sigma_1(\mathbf{A}).$$

The implication from right to left follows from Proposition 3. If $T(\mathbf{A}) = \Sigma_1(\mathbf{A})$, $\Sigma_1(\mathbf{A})$ is bisimulation closed since every language accepted by a parity tree automaton has this property [6]. Then, Proposition 3 implies that $\Sigma_1(\mathbf{A})$ is Σ_1-definable, and hence, so is $T(\mathbf{A})$.

For the implication in the other direction we first show:

Lemma 5 *For every parity tree automaton* $\mathbf{A}$*,*

$$\Sigma_1(\mathbf{A}) \subseteq T(\mathbf{A}).$$

PROOF. It is easy to show by transfinite induction that $[t \downarrow w]_{\equiv_{\mathbf{A}}} \in \mathcal{R}^{\tau}_{\alpha_\beta}(w)$ for every ordinal τ and every $w \in D_\tau$, where $\mathcal{R}$ and D_τ are defined as in Section 4.3.

With Proposition 1 we obtain that $[t]_{\equiv_{\mathbf{A}}} \in \mathcal{R}^{\tau}_{\alpha_\beta}(\rho_{\mathbf{t}}) = \beta(\rho_{\mathbf{t}})$. Since β is an accepting run, this implies that $t \in T(\mathbf{A})$. □

It remains to show:

Lemma 6 *If $T(\mathbf{A})$ is Σ_1-definable, then $T(\mathbf{A}) \subseteq \Sigma_1(\mathbf{A})$.*

PROOF. Assume that $T(\mathbf{A})$ is Σ_1-definable. Let $\mathbf{A}'$ be an 1-automaton accepting $T(\mathbf{A})$, $t \in T(\mathbf{A})$, and r be an accepting run of $\mathbf{A}'$ on t. We need to construct an accepting run β of $\mathbf{B}_{\mathbf{A}}$ on t.

For $w \in W_t$, define

$$\beta_r(w) := \{q \mid \text{ there exists } w \in W_t \text{ with } (w, q) \in V_r\}.$$

Since $\mathbf{A}'$ is a 1-automaton we know:

Claim I. On every path in t only a finite number of worlds w have a nonempty value $\beta_r(w)$.

To construct β, we define the following sets:

- $F_r := \{w \in W_t \mid \beta_r(w) = \emptyset$ and there exists w' and w'' with $(w', w) \in E_t$, $(w, w'') \in E_t$, and $\beta_r(w') \neq \emptyset\}$,
- F'_r contains all descendants of worlds in F_r (excluding the worlds in F_r).
- $L_r := \{w \in W_t \mid w$ is a leaf, and $\beta_r(w) \neq \emptyset$ or there exists w' with $(w', w) \in E_t$ and $\beta_r(w') \neq \emptyset\}$

We define β as follows:

- $\beta(w) := \emptyset$ if $w \in F'_r$,
- $\beta(w) := \{\gamma(\lambda_t(w))\}$ if $w \in L_r$,
- $\beta(w) := \Gamma(\lambda_t(w))$ if $w \in F_r$,
- $\beta(w) := \Delta(\{\beta(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w))$ otherwise.

Claim II. β is an accepting run of $\mathbf{B_A}$ on t.

Proof of the claim. We first show that $\mathcal{F} := F_r \cup L_r$ is a frontier. Let $\hat{\mathcal{F}}$ be defined as in Section 4.3. By the definition of β on worlds in $\mathcal{F}$, Proposition 2 immediately implies that β is a run. It then remains to show that β is an accepting run.

Assume that $\mathcal{F}$ is not a frontier. First, assume that $\mathcal{F}$ is not an anti-chain. Thus, there exist $w, w' \in \mathcal{F}$, $w \neq w'$, such that there is a nonempty path from w to w' in t. We can conclude that $w \in F_r$. Since $\beta_r(w) = \emptyset$, it follows that $\beta_r(w'') = \emptyset$ for all w'' on the path from w to w'. Thus, $w' \notin \mathcal{F}$. Consequently, $\mathcal{F}$ is an anti-chain, and we can assume that there exists a maximum path π in t starting from $\rho_\mathbf{t}$ which does not contain a world in $\mathcal{F}$. Since r is a run of $\mathbf{A}$ on t, $\beta_r(\rho_\mathbf{t})$ contains $\mathbf{A}$'s initial state. Thus, $\beta_r(\rho_\mathbf{t}) \neq \emptyset$. If π is infinite, by Claim I there must exist two successive worlds w and w' in π such that $\beta_r(w) \neq \emptyset$ and $\beta_r(w') = \emptyset$. But then, $w' \in F_r \subseteq \mathcal{F}$, a contradiction. If π is finite, two cases can occur. First, there exists an inner node w' as before, which again lead to a contradiction. Second, the end node w of π, which is a leaf, belongs to $L_r \subseteq \mathcal{F}$, a contradiction. Thus, $\mathcal{F}$ is a frontier.

It remains to show that β is an *accepting* run of $\mathbf{B_A}$. Proposition 2 implies that $\beta(\rho_\mathbf{t}) \neq \emptyset$. We need to show that $\beta(\rho_\mathbf{t}) \subseteq \{[t]_{\equiv_\mathbf{A}} \mid t \in T(\mathbf{A})\}$.

Let $C \in \beta(\rho_\mathbf{t})$. By transfinite induction, one shows that for $C \in \beta(\rho_\mathbf{t})$ there exists a mapping $\hat{\beta}$ from $\hat{\mathcal{F}}$ into $C_\mathbf{A}$ such that

- $\hat{\beta}(\rho_\mathbf{t}) = C$,
- $\hat{\beta}(w) \in \beta(w)$ for every $\hat{\mathcal{F}}$,
- $\hat{\beta}(w) = f_\mathbf{A}(\{\beta(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w))$ for every $w \in \hat{\mathcal{F}} \setminus \mathcal{F}$.

Let t' be a tree obtained from t as follows: For every $w \in F_r$, replace the subtree $t \downarrow w$ in t by a tree belonging to the class $\hat{\beta}(w) \in \beta(w) = \Gamma(\lambda_t(w))$.

It is easy to see that r is a run on t'. Thus, $t' \in T(\mathbf{A})$. By transfinite induction, one shows that $\hat{\beta}(w) = [t' \downarrow w]_{\equiv_\mathbf{A}}$ for every $w \in \hat{\mathcal{F}}$. This implies that $C = \hat{\beta}(\rho_\mathbf{t}) = [t']_{\equiv_\mathbf{A}} \in \{[t]_{\equiv_\mathbf{A}} \mid t \in T(\mathbf{A})\}$. □

6 Complementing Bottom-up Tree Automata

We show that the complement of a language accepted by a bottom-up tree automaton is accepted by a top-down tree automaton of the same size, which

will be used to decide the emptiness problem (5). We also note that from the correspondence between the complement of bottom-up tree automata and top-down tree automata, one can easily derive a decidability criterion for Π_1-Definability, similar to the one for Σ_1-Definability.

6.1 Top-down Tree Automata

A *top-down tree automaton* $\mathbf{D}$ is a tuple (Q, P, I, δ), where Q is the finite set of states, P is a finite set of propositional variables, $I \subseteq Q$ is the set of initial states, and δ is a mapping from $2^Q \times 2^P$ into Q.

A *run* r of $\mathbf{D}$ on $t \in \mathcal{T}_P$ is a mapping from W_t into Q such that $r(w) = \delta(\{r(w') \mid w' \in \mathrm{Suc}_t(w')\}, \lambda_t(w))$. The run is called *accepting* if $r(\rho_\mathbf{t}) \in I$. A tree $t \in \mathcal{T}_P$ is accepted by $\mathbf{D}$, if there exists an accepting run of $\mathbf{D}$ on t. The set of trees accepted by $\mathbf{D}$ is denoted $T(\mathbf{D})$.

Similar to the case for bottom-up tree automata, one shows the following proposition.

Proposition 5 *Every language accepted by a top-down tree automaton is MSOL-definable.*

As in the proof of Proposition 4, one can specify a MSOL-formulas that encodes a run of a top-down automaton.

Also, similar to Proposition 3, we prove:

Proposition 6 *Whenever the language accepted by a top-down tree automaton is closed under bisimulation, then it is* Π_1*-definable.*

Proof. Given a top-down tree automaton $\mathbf{D} = (Q, P, I, \delta)$ such that $T(\mathbf{D})$ is bisimulation closed, one shows, analogously to the proof of Proposition 3, that the 0-automaton $\mathbf{A} = (Q_\mathbf{A}, q_I, \delta_\mathbf{A}, \Omega_\mathbf{A})$ accepts $T(\mathbf{D})$, where q_I is a new state, $Q_\mathbf{A} := Q \cup \{q_I\}$, $\Omega_\mathbf{A}$ assigns to every state priority 0, and $\delta_\mathbf{A}$ is defined as follows: For $P' \subseteq P$, let $\bigwedge P' := \bigwedge_{p \in P'} p \wedge \bigwedge_{p \notin P'} \neg p$. Now,

$$\delta_\mathbf{A}(q_I) := \bigvee_{q \in I} q,$$

and for every $q \in Q$,

$$\delta_\mathbf{A}(q) \;:=\; \bigvee_{\substack{P' \subseteq P \\ Q' \subseteq Q \\ q = \delta(Q', P')}} \bigwedge P' \wedge \Box \bigvee_{q' \in Q'} q' \wedge \bigwedge_{q' \in Q} \Diamond q'.$$

□

In Section 6.4, we present an example of a top-down tree automaton induced by a parity tree automaton whose language is not bisimulation closed.

6.2 Complementation using Top-down Tree Automata

In this section, we show that the complement of the language accepted by the bottom-up automaton $\mathbf{B} = (Q, P, F, \delta)$ is accepted by the top-down automaton

$$\mathbf{D_B} = (Q, P, Q \setminus F, \delta).$$

In the following two lemmas this is proved for finitely branching trees. We then generalize the statement to infinitely branching trees using the finite model property of MSOL.

Lemma 7 *For every finitely branching tree $t \in \mathcal{T}_P$, $t \in T(\mathbf{D_B})$ implies $t \in \mathcal{T}_P \setminus T(\mathbf{B})$.*

Proof. Assume that $t \in \mathcal{T}_P$ is finitely branching and $t \in T(\mathbf{D_B})$. Let r be an accepting run of $\mathbf{D_B}$ on t. We will show that for every run β of $\mathbf{B}$ on t, $r(\rho_\mathbf{t}) \in \beta(\rho_\mathbf{t})$. Since $r(\rho_\mathbf{t}) \notin F$, we then conclude that β is not accepting. Thus, $t \notin T(\mathbf{B})$.

Let β be a run on t. Let $\mathcal{F}_\beta$ and $\hat{\mathcal{F}}_\beta$ be defined as in Section 4.3. It is easy to see that $r(w) \in \beta(w)$, for all $w \in \mathcal{F}_\beta$. By transfinite induction and using Proposition 1, we obtain that $r(w) \in \beta(w)$ for every $w \in \hat{\mathcal{F}}_\beta$. In particular, $r(\rho_\mathbf{t}) \in \beta(\rho_\mathbf{t})$. □

The following lemma states the implication in the other direction.

Lemma 8 *For every finitely branching tree $t \in \mathcal{T}_P$, $t \in \mathcal{T}_P \setminus T(\mathbf{B})$ implies $t \in T(\mathbf{D_B})$.*

Proof. Assume that $t \in \mathcal{T}_P$ is finitely branching and $t \in \mathcal{T}_P \setminus T(\mathbf{B})$.

For every $n \geq 0$, we define the set $\mathcal{F}_n := \{w \mid$ the path from $\rho_\mathbf{t}$ to w has length n or this path has length $< n$ and w is a leaf$\}$. Obviously, $\mathcal{F}_n$ is a frontier in t. As usual, let $\hat{\mathcal{F}}_n$ denote the ancestors of worlds in $\mathcal{F}_n$ including the worlds in $\mathcal{F}_n$. Note that $\hat{\mathcal{F}}_n$ is finite since t is finitely branching.

For $n \geq 0$ an n-valuation υ is a mapping from $\hat{\mathcal{F}}_n$ into Q such that $\upsilon(\rho_\mathbf{t}) \notin F$, and for every $w \in \hat{\mathcal{F}}_n$,

- $\upsilon(w) := \delta(\emptyset, \lambda_t(w))$ $(= [t \downarrow w]_{\equiv_\mathbf{A}})$ if $w \in \mathcal{F}_n$ is a leaf,

- $\upsilon(w) := \delta(Q', \lambda_t(w))$ for some $Q' \subseteq Q$ and $w \in F_n$ is not a leaf, and
- $\upsilon(w) := \delta(\{\upsilon(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w))$ if $w \notin \mathcal{F}_n$.

Let $\mathcal{V}_n := \{\upsilon \mid \upsilon \text{ is a } n\text{-valuation}\}$ be the set of all n-valuations. Obviously, $\mathcal{V}_n$ is a finite set.

Because $t \notin T(\mathbf{B})$, we can conclude that for every $n \geq 0$, $\mathcal{V}_n \neq \emptyset$. Otherwise, it is easy to see that $\mathcal{R}^{\xi}_{\alpha_{\mathcal{F}_n}}$, see Proposition 2, is an accepting run of $\mathbf{B}$ on t.

For $D \subseteq \hat{\mathcal{F}}_n$, let $\mathcal{V}_n|_D$ denote the set of n-valuations in $\mathcal{V}_n$ restricted to D. Then, it is an easy exercise to prove that for every $n \geq 0$, $\mathcal{V}_n \supseteq \mathcal{V}_{n+1}|_{\hat{\mathcal{F}}_n} \neq \emptyset$.

As a consequence, there exists a $\gamma_0 \in \mathcal{V}_0$ that occurs in $\mathcal{V}_n|_{\mathcal{F}_0}$ for infinitely many $n \geq 1$; here we use that the sets $\mathcal{V}_n$ are finite. Let $n_0, n_1, n_2, \ldots$ be these n's. For every $i \geq 0$, define $\mathcal{V}^0_{n_i} := \{\gamma \in \mathcal{V}_{n_i} \mid \gamma|_{\mathcal{F}_0} = \gamma_0\}$. We know that these sets are finite and nonempty. Also, for every $i \geq 0$, $\mathcal{V}^0_{n_i} \supseteq \mathcal{V}^0_{n_{i+1}}|_{\hat{\mathcal{F}}_{n_i}} \neq \emptyset$. As before, we can conclude that there exists a $\gamma_1 \in \mathcal{V}^0_{n_0}$ which belongs to $\mathcal{V}^0_{n_i}|_{\mathcal{F}_{n_0}}$ for infinitely many $i \geq 1$. Iterating this argument, we obtain a sequence of valuations $\gamma_0, \gamma_1, \gamma_2, \ldots$ and indices $i_0 < i_1 < i_2 < \cdots$ such that $\gamma_j \in \mathcal{V}_{i_j}$ and $\gamma_{j+1}|_{\mathcal{F}_{i_j}} = \gamma_j$ for every $j \geq 0$. Define γ to be the limes of the γ_j, i.e., for every $w \in W_t$, $\gamma(w) := \gamma_j(w)$ if $w \in \mathcal{F}_{i_j}$ for j minimal. It is straightforward to verify that γ is a run of $\mathbf{D_B}$ on t. In particular, $t \in T(\mathbf{D_B})$. □

Now assume that there exists an infinite branching tree $t \in \mathcal{T}_P$ that belongs to the symmetric difference of $\mathcal{T}_P \setminus T(\mathbf{B})$ and $T(\mathbf{D_B})$, i.e.,

$$t \in \Big(\big(\mathcal{T}_P \setminus T(\mathbf{B})\big) \setminus T(\mathbf{D_B})\Big) \cup \Big(T(\mathbf{D_B}) \setminus \big(\mathcal{T}_P \setminus T(\mathbf{B})\big)\Big) =: L.$$

We know that $T(\mathbf{B})$ and $T(\mathbf{D_B})$ are MSOL-definable. Thus, L is MSOL-definable. It is well-known that MSOL formulas have the finite model property. Thus, if $L \neq \emptyset$, then L contains a finite branching tree. But the two lemmas from above tell us that on finite branching trees the sets $T(\mathbf{B})$ and $T(\mathbf{D_B})$ coincide. Consequently, $L = \emptyset$. This proves

$$\mathcal{T}_P \setminus T(\mathbf{B}) \;=\; T(\mathbf{D_B}), \tag{16}$$

and thus, we obtain:

Proposition 7 *For every bottom-up tree automaton* $\mathbf{B}$ *there exists a top-down tree automaton which accepts the complement* $\mathcal{T}_P \setminus T(\mathbf{B})$ *of* $T(\mathbf{B})$*, and which is of size linear in the size of* $\mathbf{B}$*.*

As a corollary of Theorem 2, Lemma 5, and (the proof of) Proposition 7, we obtain:

Corollary 2 *For every parity tree automaton* $\mathbf{A}$*,*

$$T(\mathbf{A}) \text{ is } \Sigma_1\text{-definable} \quad \text{iff} \quad T(\mathbf{A}) \cap T(\mathbf{D}_{\mathbf{B}_\mathbf{A}}) = \emptyset.$$

6.3 Π_1-Definability

In this section, we derive a characterization of Π_1-definability from Theorem 2 and (16).

Let $\mathbf{A}$ be a parity tree automaton over the propositional variables P. Define the top-down tree automaton $\mathbf{D}_\mathbf{A} := (C_\mathbf{A}, P, \{[t]_{\equiv_\mathbf{A}} \mid t \in T(\mathbf{A})\}, f_\mathbf{A})$. We call $\Pi_1(\mathbf{A}) := T(\mathbf{D}_\mathbf{A})$ the Π_1-test language of $\mathbf{A}$. Then, Π_1-definability can be characterized as follows.

Theorem 3 *Let* $\mathbf{A}$ *be a parity tree automaton. Then,*

$$T(\mathbf{A}) \text{ is } \Pi_1\text{-definable} \quad \text{iff} \quad T(\mathbf{A}) = \Pi_1(\mathbf{A}).$$

To prove the theorem, let $\overline{\mathbf{A}}$ denote the dual automaton of $\mathbf{A}$, i.e., in the transition function $\bot$ and $\top$, $\wedge$ and $\vee$, $\Box$ and $\Diamond$ are exchanged, and the priority of every state is increased by 1. It is well known that $T(\overline{\mathbf{A}}) = \mathcal{T}_P \setminus T(\mathbf{A})$. Observe:

$$\begin{array}{lll} T(\mathbf{A}) \text{ is } \Pi_1\text{-definable} & \text{iff} & \mathcal{T}_P \setminus T(\mathbf{A}) \text{ is } \Sigma_1\text{-definable} \\ & \text{iff} & T(\overline{\mathbf{A}}) \text{ is } \Sigma_1\text{-definable} \\ & \text{iff} & T(\overline{\mathbf{A}}) = \Sigma_1(\overline{\mathbf{A}}) \\ & \text{iff} & T(\mathbf{A}) = \mathcal{T}_P \setminus \Sigma_1(\overline{\mathbf{A}}) \\ & \text{iff} & T(\mathbf{A}) = T(\mathbf{D}_{\mathbf{B}_{\overline{\mathbf{A}}}}). \end{array}$$

Hence, we are done if we can show that $T(\mathbf{D}_{\mathbf{B}_{\overline{\mathbf{A}}}}) = T(\mathbf{D}_\mathbf{A})$. In fact, we show $\mathbf{D}_{\mathbf{B}_{\overline{\mathbf{A}}}} = \mathbf{D}_\mathbf{A}$.

Recall that $\mathbf{D}_{\mathbf{B}_{\overline{\mathbf{A}}}} = (C_{\overline{\mathbf{A}}}, P, \{[t]_{\equiv_{\overline{\mathbf{A}}}} \mid t \notin T(\overline{\mathbf{A}})\}, f_{\overline{\mathbf{A}}})$.

Lemma 9 *For every parity automaton* $\mathbf{A}$ *over* P *it holds:*

1. $[t]_{\equiv_\mathbf{A}} = [t]_{\equiv_{\overline{\mathbf{A}}}}$ *for every Kripke tree* t*. In particular,* $C_\mathbf{A} = C_{\overline{\mathbf{A}}}$*.*
2. $f_\mathbf{A} = f_{\overline{\mathbf{A}}}$*.*

PROOF. The first proposition is obvious. For the second proposition observe that for every $\mathcal{C} \subseteq C_{\mathbf{A}}$, $C \in C_{\mathbf{A}}$, and every $P' \subseteq P$,

$$f_{\mathbf{A}}(\mathcal{C}, P') = C \quad \text{iff} \quad \text{there exists } t \in C \text{ with } \lambda_t(\rho_{\mathbf{t}}) = P' \text{ and } \mathcal{C} = \{[t \downarrow w]_{\equiv_{\mathbf{A}}} \mid w \in \mathrm{Suc}_t(\rho_{\mathbf{t}})\}.$$

Because of 1., we know that $[t \downarrow w]_{\equiv_{\mathbf{A}}} = [t \downarrow w]_{\equiv_{\overline{\mathbf{A}}}}$. Thus, $f_{\mathbf{A}} = f_{\overline{\mathbf{A}}}$. □

From this lemma, we immediately obtain that $\mathbf{D}_{\mathbf{B}_{\overline{\mathbf{A}}}} = \mathbf{D}_{\mathbf{A}}$, which completes the proof of the theorem.

As a corollary of the proof, we obtain that the complement of every top-down tree automaton induced by a parity tree automaton is definable by a bottom-up tree automaton.

Corollary 3 *For every parity tree automaton* $\mathbf{A}$ *over* P*,*

$$\mathcal{T}_P \setminus T(\mathbf{D}_{\mathbf{A}}) = T(\mathbf{B}_{\overline{\mathbf{A}}}).$$

PROOF. We know that $T(\mathbf{D}_{\mathbf{A}}) = T(\mathbf{D}_{\mathbf{B}_{\overline{\mathbf{A}}}})$. Thus, $\mathcal{T}_P \setminus T(\mathbf{D}_{\mathbf{A}}) = \mathcal{T}_P \setminus T(\mathbf{D}_{\mathbf{B}_{\overline{\mathbf{A}}}})$. By (16), $\mathcal{T}_P \setminus T(\mathbf{D}_{\mathbf{B}_{\overline{\mathbf{A}}}}) = T(\mathbf{B}_{\overline{\mathbf{A}}})$. □

6.4 A Non-bisimulation Closed Top-down Automaton

We show that there exist top-down tree automaton induced by parity tree automaton that are not bisimulation closed. More precisely, we show the following.

Proposition 8 *There exists a parity tree automaton* $\mathbf{A}$ *such that* $T(\mathbf{D}_{\mathbf{A}})$ *is not bisimulation closed.*

To simplify the presentation of the proof we consider trees that are labelled by letters $a, b, c, \ldots$ instead of subsets of $\mathcal{P}$.

We will construct a parity tree automaton $\mathbf{A}$ which accepts those trees t with $\lambda_t(\rho_{\mathbf{t}}) = a$ and there exist $w_b, w_c \in \mathrm{Suc}_t(\rho_{\mathbf{t}})$ such that both $t \downarrow w_b$ and $t \downarrow w_c$ have an infinite path and on every maximum path in $t \downarrow w_b$ and $t \downarrow w_c$ one eventually only sees b's and c's, respectively. In particular, $w_b \neq w_c$.

This automaton can be defined as follows:

$$\mathbf{A} = (\{q_I, q_b, q_b', q_c, q_c'\}, \{a, b, c\}, q_I, \delta, \Omega),$$

where Ω maps q, q_b, and q_c to 1, and q'_b and q'_c to 0. The transition function δ is defined as follows:

$$\begin{aligned}
\delta(q_I) &= a \wedge \Diamond q_b \wedge \Diamond q_c, \\
\delta(q_b) &= (b \wedge \Box q'_b \wedge \Diamond 1) \vee (\Box q_b \wedge \Diamond 1), \\
\delta(q'_b) &= (b \wedge \Box q'_b \wedge \Diamond 1) \vee (\Box q_b \wedge \Diamond 1), \\
\delta(q_c) &= (c \wedge \Box q'_c \wedge \Diamond 1) \vee (\Box q_c \wedge \Diamond 1), \\
\delta(q'_c) &= (c \wedge \Box q'_c \wedge \Diamond 1) \vee (\Box q_c \wedge \Diamond 1).
\end{aligned}$$

Obviously, $\mathbf{A}$ accepts the language described above. It is an easy exercise to check that there are four equivalence classes C_1, C_2, and C_3, where $C_1 = T(\mathbf{A})$, $C_2 = T(\mathbf{A}_{q_b})$, $C_3 = T(\mathbf{A}_{q_c})$, and C_4 contains the remaining trees.

Consider a binary tree t' where every world is labelled with a. This tree is accepted by $\mathbf{D}_\mathbf{A}$: Labeling the root with C_1, the worlds of the left subtree with C_2, and the worlds of the right subtree with C_3, is an accepting run of $\mathbf{D}_\mathbf{A}$ on t.

Conversely, the unary tree t' where every world is labelled with a is not accepted by $\mathbf{D}_\mathbf{A}$: The transition applied at the root must be of the form $f_\mathbf{A}(\mathcal{C}, a) = C_1$. But this only holds if $\mathcal{C}$ contains both C_2 and C_3. However, the root of t' only has one successor.

We have shown that even though t and t' are bisimilar, t is accepted by $\mathbf{D}_\mathbf{A}$ but t' is not. This completes the proof of Proposition 8.

7 Complexity

The aim of this section is to show the complexity upper and lower bound claimed in Theorem 1. We first show the upper bound and then present the proof of the lower bound.

7.1 Upper Bound

Due to Fact 1 we can assume that instead of a μ-calculus formula we have given a parity tree automaton $\mathbf{A}$. Also, Corollary 2 tells us that it suffices to show that the emptiness problem $T(\mathbf{A}) \cap T(\mathbf{D}) \stackrel{?}{=} \emptyset$ is decidable in deterministic exponential time, where $\mathbf{D} := \mathbf{D}_{\mathbf{B}_\mathbf{A}}$.

The proof of this proceeds in three steps. First, we characterize $\mathbf{D}$ by yet another automaton model. Second, we show a "small branching property"

for $T(\mathbf{A}) \cap T(\mathbf{D})$. And, third, we describe how a typical emptiness test for parity tree automata can be modified so as to work for our problem.

7.1.1 Modal Top-down Automata

A *modal top-down automaton* is a tuple

$$(Q, P, \mathcal{Q}_A, \alpha_I, \delta) \tag{17}$$

where Q and P are as usual, $\mathcal{Q}_A \subseteq 2^Q$ is a set of *admissible subsets,* α_I is a propositional formula over Q, the so-called *initial condition,* and δ maps every state from Q to a transition condition (defined below).

A *run* of such an automaton on a given tree is a labeling of the worlds of the tree with admissible subsets such that the following conditions are satisfied. First, the root is labelled with a set satisfying the initial condition. Second, if a world is labelled with $Q' \subseteq Q$, then, for every $q \in Q'$, the transition condition $\delta(q)$ is satisfied in this node, which is defined in the straightforward way.

We want to show that the automaton $\mathbf{D}$ is equivalent to a simple modal top-down automaton. Assume $\mathbf{A}$ is of the form $(Q, P, q_I, \delta, \Omega)$. For every tree t, let $tp(t) = \{q \in Q \mid t \in T(\mathbf{A}_q)\}$ be the *type* of t. Then, for all trees t, t', $t \equiv_{\mathbf{A}} t'$ iff $tp(t) = tp(t)'$. Thus, when we set $Tp_{\mathbf{A}} = \{tp(t) \mid t \text{ is a tree}\}$, then $Tp_{\mathbf{A}}$ corresponds to $C_{\mathbf{A}}$. That is, we can rephrase $\mathbf{D}$ in terms of types. The state set becomes $Tp_{\mathbf{A}}$, the final state set is the set of all types q_I does not belong to, and the transition function f is of the form $f\colon 2^{Tp_{\mathbf{A}}} \times 2^P \to Tp_{\mathbf{A}}$ and satisfies certain conditions depending on the transition function of $\mathbf{A}$. Let $\mathcal{Q} \subseteq Tp_{\mathbf{A}}$. Then $q \in f(\mathcal{Q}, P')$ if and only if

- $\delta(q) = \top$,
- $\delta(q) = q'$ and $q' \in f(\mathcal{Q}, P')$,
- $\delta(q) = q' \wedge q''$ and $q', q'' \in f(\mathcal{Q}, P')$,
- $\delta(q) = q' \vee q''$ and $q' \in f(\mathcal{Q}, P')$ or $q'' \in f(\mathcal{Q}, P')$,
- $\delta(q) = \Diamond q'$ and there exists $Q' \in \mathcal{Q}$ such that $q' \in Q'$, or
- $\delta(q) = \Box q'$ and for all $Q' \in \mathcal{Q}$, $q' \in Q'$.

This can be easily described by a modal top-down automaton:

Proposition 9 *Let* $\mathbf{A}$ *be an arbitrary parity tree automaton with* n *states. Then there exists a modal top-down automaton* $\mathbf{D}$ *with* $2n$ *states equivalent to* $\mathbf{D}_{\mathbf{B_A}}$.

PROOF. Let $\mathbf{A}$ be a usual. We simply take $(Q', P, \mathcal{Q}_A, \alpha_I, \delta')$ where the individual components are defined as follows. The state set is defined by $Q' = Q \cup \bar{Q}$ for a disjoint copy $\bar{Q} = \{\bar{q} \mid q \in Q\}$ of Q. For every tree t, $\mathcal{Q}_A$ contains the set $tp(t) \cup \{\bar{q} \mid q \notin tp(t)\}$. The initial condition simply says $\bar{q}_I$. The transition function δ' coincides with δ on Q and for every $q \notin Q$, $\delta'(\bar{q})$ is the condition dual to $\delta(q)$, that is, where $\top$ and $\bot$, $\vee$ and $\wedge$, $\Diamond$ and $\Box$, q' and $\bar{q}'$ are exchanged. Then, clearly, the automaton works as required. $\Box$

7.1.2 Small Branching Property

We define an ordering on trees. When t and t' are trees, then $t \sqsubseteq t'$ if t is a subgraph (in the usual sense) of t' and has the same root as t', that is, t results from t' by removing subtrees. Given t and t' such that $t \sqsubseteq t'$, we use $[t, t']$ to denote $\{t^* \mid t \sqsubseteq t^* \sqsubseteq t'\}$.

We say that a tree language T has the *n-branching property* if for every tree $t \in T$, there exists a tree $t_0 \sqsubseteq t$ of branching degree $\leq n$ such that $[t_0, t] \subseteq T$.

A typical emptiness test for parity tree automata starts with a statement similar to the following, see, for instance, [5].

Lemma 10 *Every tree language recognized by a parity tree automaton with* n *states has the* n*-branching property.*

We prove a similar statement for modal top-down automata.

Lemma 11 *Every tree language recognized by a modal top-down automaton with* n *states has the* n*-branching property.*

PROOF. To see this, assume $\mathbf{M}$ is a modal top-down automaton as above and assume that in a run of $\mathbf{M}$ on some tree t a world is labelled Q' and the set of labelling of its successors is $\mathcal{Q}$. It is now easy to see that we can cut off subtrees rooted at the successors of the considered world and still satisfy all the transition conditions as long as we keep for each $q \in Q'$ with $\delta(q) = \Diamond q'$ a successor whose labelling contains q'. This yields an n-branching property. $\Box$

As a consequence, we can not the following.

Corollary 4 *Let T and T' be tree languages recognized by a parity tree automaton with n states and modal top-down automaton with n' states. Then $T \cap T'$ has the $(n + n')$-branching property.*

For the proof of this, just consider the union of the two trees that are guaranteed to exist by the individual small branching properties.

7.1.3 Emptiness Test

Consider a typical emptiness test for a parity tree automaton $\mathbf{A}$ with n states, for instance, the one described in [5]. In the first step, one exploits the n-branching property. Using Safra's construction, one constructs a non-deterministic Rabin tree automaton $\mathbf{A}'$ with $2^{O(n \log n)}$ states and $O(n)$ pairs which is equivalent to $\mathbf{A}$ on all $\leq n$ branching trees. This can be an automaton with transitions depending on the branching degree. In the second step, one solves the emptiness test for this automaton, for instance, by reducing it to the problem of finding the winner in an appropriate two-player infinite game.

In order to check whether or not $T(\mathbf{A}) \cap T(\mathbf{D}) = \emptyset$, as above, one can proceed in a similar fashion. By Lemmas 10 and 11 and Corollary 4 we know that $T(\mathbf{A}) \cap T(\mathbf{D})$ has the $3n$-branching property. Therefore, in a first step, we construct a non-deterministic Rabin tree automaton $\mathbf{A}'$ as above which is equivalent to $\mathbf{A}$ on all $\leq 3n$ branching trees. Second, we convert $\mathbf{D}$ to a non-deterministic tree automaton $\mathbf{D}'$ of exponential size with trivial acceptance condition (0-acceptance) which is equivalent to $\mathbf{D}$ on all $\leq 3n$ branching trees. (Observe that this is possible since we can check in deterministic exponential time whether or not a subset of the state set of $\mathbf{A}$ constitutes a type.) Third, we form, in a straightforward manner, a product of $\mathbf{A}'$ and $\mathbf{D}'$ that recognizes exactly all $\leq 3n$ branching trees in $T(\mathbf{A}) \cap T(\mathbf{D})$. Finally, we perform an emptiness test for this product automaton, for instance, by reducing it to a winner problem for an appropriate game.

7.2 Lower Bound

The proof of the lower bound is quite simple. We reduce the tautology problem for modal μ-calculus, L_μ-TAUTOLOGY, to Σ_1-DEFINABILITY. This proves hardness for deterministic exponential time, because the satisfiability

problem for modal μ-calculus, L_μ-SATISFIABILITY, is complete for deterministic exponential time [5], and deterministic exponential time is closed under complementation.

Proposition 10 *L_μ-TAUTOLOGY is polynomial-time reducible to Σ_1-DEFINABILITY.*

PROOF. Let p be some new propositional variable. Define π_1 by

$$\pi_1 = \Diamond \nu X (p \wedge \Box X) \tag{18}$$

Clearly, the property defined by π_1 is Π_1- but not Σ_1-definable, see [1].

Now, let ϕ be an arbitrary L_μ-formula. Consider the formula ϕ^* defined by

$$\phi^* = \Diamond(\neg p \wedge \neg \phi) \wedge \pi_1 \tag{19}$$

We claim that ϕ is a tautology iff ϕ^* is Σ_1-definable.

If ϕ is a tautology, then $\neg\phi$ is not satisfiable, and, hence, ϕ^* is not satisfiable. Therefore, ϕ^* is equivalent to FALSE, which is a Σ_1-formula.

Conversely, assume ϕ^* is equivalent to a Σ_1-formula ψ, and ϕ is not a tautology. We construct a Σ_1-formula χ equivalent to π_1, which would be a contradiction.

Since ϕ is no tautology, $\neg p \wedge \neg\phi$ has a model, say the tree t' with root w'. For an arbitrary tree t with root w, let t^* be the tree which is obtained from t by adding a disjoint copy of t' at w. Formally, if t'' is a copy of t' with root w'' and $W_{t''}$ disjoint from W_t, then

$$\begin{aligned} W_{t^*} &= W_t \cup W_{t''} \\ R_{t^*} &= R_t \cup R_{t''} \cup \{(w, w'')\} \\ \lambda_{t^*} &= \lambda_t \cup \lambda_{t''} \end{aligned}$$

By unwinding the fixed point operators in ψ and using basic propositional logic, we can rewrite ψ as a disjunction of formulas of the form

$$\alpha \wedge \bigwedge_{i<n} \Diamond \rho_i \wedge \Box \sigma \tag{20}$$

where α is a propositional formula and the ρ_i's and σ are arbitrary Σ_1-formulas. (See, for instance, [2].) After introducing a further case distinction, we can even assume that each ρ_i is either of the form $p \wedge \rho_i'$ or $\neg p \wedge \rho_i'$.

Let ψ' the result of rewriting ψ according to the above remarks. That is, we assume that ψ' is

- equivalent to ϕ^*,
- a Σ_1-formula,
- a disjunction of formulas of the above form, (20), where α is propositional and each ρ_i is of the form $p \wedge \rho_i'$ or $\neg p \wedge \rho_i'$.

In the following, the individual disjuncts of ψ' will be referred to as disjuncts. These disjuncts are divided into the disjuncts that hold true in at least one tree of the form t^* and are called productive disjuncts, and the remaining disjuncts, which are called unproductive disjuncts.

Let ψ'' be the formula which is obtained from ψ' by removing the unproductive disjuncts. The conjuncts of ψ'' will be referred to as propositional conjuncts, positive diamond conjuncts, negative diamond conjuncts, and box conjuncts, respectively.

Note that this definition ensures that $t' \models \sigma$ for every formula σ for which $\Box\sigma$ is a box conjunct. Similarly, $t' \models \rho$ for every formula ρ for which $\Diamond(\neg p \wedge \rho)$ is a positive diamond conjunct.

The desired formula χ is obtained from ψ'' by removing every negative diamond conjunct. To complete the proof, we only need to show that χ is equivalent to π_1.

For one direction, assume $t \models \pi_1$. By construction, $t^* \models \phi^*$. Thus $t^* \models \psi$, hence $t^* \models \psi'$, and thus $t^* \models \psi''$. Now, observe the following. First, every propositional conjunct that holds in t^* also holds in t. Second, every box conjunct which holds in t^* also holds in t, because w has fewer successors in t than in t^*. Third, any formula of the form $p \wedge \rho_i$ can only hold true in the successors of w which are present in t and not in the other successor, which is a copy of w' and satisfies $\neg p$. So every positive diamond conjunct which holds in t^* also holds in t. From these three observations, we can conclude $t \models \chi$.

Conversely, assume $t \models \chi$. Observe the following. First, every propositional conjunct that holds in t^* also holds in (t, w). Second, the above note implies that every box conjunct which holds in (t, w) also holds in t^*. Third, the above note implies also that every negative diamond conjunct which holds in t also holds in t^*. Therefore, $t^* \models \psi''$. Consequently, $t^* \models \psi'$, hence $t^* \models \psi$, and thus $t^* \models \pi_1$. Since the copy of w' in t^* satisfies $\neg p$, we obtain $t \models \pi_1$. $\Box$

References

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

[2] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking (extended abstract). In *6th International Conference on Computer Aided Verification (CAV '94)*, number 818 in Lecture Notes in Computer Science, pages 142–155. Springer-Verlag, 1994.

[3] J. C. Bradfield. The Modal mu-calculus Alternation Hierarchy is Strict. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory. 7th International Conference*, volume 1119 of *LNCS*, pages 232–246. Springer-Verlag, August 1996.

[4] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extend abstract). In *1st IEEE Symposium on Logic in Computer Science*, pages 267–278, Cambridge, Massachusetts, 1986.

[5] E.A. Emerson and C.S. Jutla. The Complexity of Tree Automata and Logics of Programs (Exteded Abstract). In *IEEE Symposium on Foundations of Computer Science* (FOCS'88), pages 328–337, Los Alamitos, California, October 1988. IEEE Computer Society Press.

[6] David Janin and Igor Walukiewicz. On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic. In U. Montanari and V. Sassone, editors, *CONCUR'96: Concurrency Theory*, volume 1119 of *LNCS*, pages 263–277. Springer-Verlag, August 1996.

[7] L.H. Landweber. Decision problems for ω-automata. *Math. Systems Theory*, 3:376–384, 1969.

[8] Giacomo Lenzi. A Hierarchy Theorem for the μ-Calculus. In F. Meyer auf der Heide and B. Monien, editors, *Automata, Languages and Programming. 23rd International Colloquium. ICALP '96*, volume 1099 of *LNCS*, pages 87–97. Springer-Verlag, July 1996.

[9] Damian Niwiński. On Fixed-Point Clones (Extended Abstract). In Laurent Kott, editor, *Automata, Languages and Programming. 13th International Colloquium*, volume 226 of *LNCS*, pages 464–473. Springer-Verlag, July 1986.

[10] M. Otto. Eliminating Recursion in the mu-Calculus. In *16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of *Lecture Notes in Computer Science*, pages 531–540. Springer-Verlag, 1999.

[11] T. F. Urbański. On Deciding if Deterministic Rabin Language Is in Büchi Class. In U. Montanari, J.D.P. Rolim, and E. Welzl, editors, *Automata, Languages and Programming, 27th International Colloquium (ICALP 2000)*, volume 1853 of *Lecture Notes in Computer Science*, pages 663–674. Springer-Verlag, 2000.

[12] I. Walukiewicz, 2002. Personal communication.

[13] T. Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bull. Soc. Math. Belg.*, 8(2), May 2001.