Deciding the First Level of the μ-calculus Alternation Hierarchy

Ralf Küsters and Thomas Wilke

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu 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 [2] 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]: 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.

Detailed proofs of all the results presented in this paper can be found in our technical report [6].

2 Basic Notions and Main Result

In this section we recall basic notions and state the main result.

2.1 Kripke Trees

We will interpret μ-calculus and MSOL-formulas (see below) over Kripke trees.

A *Kripke tree* t is a tuple (W, R, ρ, λ) where W is a nonempty set of *worlds,* $R \subseteq W \times W$ is an *accessibility relation* such that (W, R) is a tree rooted at $\rho \in W$, 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. Usually, we refer to the components of the tuple (W, R, ρ, λ) by W_t, R_t, etc. The set of all Kripke trees is denoted $\mathcal{T}$ and the set of Kripke trees labeled with subsets of P for some $P \subseteq \mathcal{P}$ is referred to by $\mathcal{T}_P$. 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.*

Two Kripke trees t and t' are called *bisimilar* ($t \cong t'$ for short) when there exists a relation $R \subseteq W_t \times W_{t'}$, called a *bisimulation relation*, such that $(\rho_t, \rho_{t'}) \in R$ and for every $(v, v') \in R$ and $p \in \mathcal{P}$:

- $p \in \lambda_t(v)$ iff $p \in \lambda_{t'}(v')$,
- whenever $(v, u) \in E_t$ for some u, then there exists u' such that $(v', u') \in E_{t'}$ and $(u, u') \in R$,
- whenever $(v', u') \in E_{t'}$ for some u', then there exists u such that $(v, u) \in E_t$ and $(u, u') \in R$.

A set S of Kripke trees is *closed under bisimulation* if $t \in S$ and $t' \cong t$ implies $t' \in S$ for every Kripke tree t and t'.

2.2 Modal μ-calculus

Modal μ-calculus [5] is modal logic augmented by operators for least and greatest fixed points. For simplicity, this paper only deals with the unimodal case, but all definitions, results, and proofs easily extend to the multi-modal case.

We fix a countably infinite supply $\mathcal{P}$ of propositional variables. The formulas of modal μ-calculus are built from the constant symbols $\perp$ 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, where propositional variables bound by a fixed point operator occur only positive. The set of μ-calculus formulas is denoted by L_μ. The set of L_μ formulas that do not contain the greatest fixed point operator is denoted by Σ_1. Analogously, Π_1 shall denote the set of all L_μ formulas that do not contain the least fixed point operator (see, for example, [9] for the definition of the whole fixed point alternation hierarchy)

In general, L_μ formulas are interpreted in Kripke structures. Without loss of generality, we may only consider (arbitrary branching) Kripke trees as defined above. The interpretation of L_μ formulas on Kripke trees is defined as usual [5].

2.3 Monadic Second Order Logic (MSOL)

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 semantics of MSOL is defined as follows, where V stands for a valuation, which assigns to every first-order variable an element of W_t and to every second-order variable a subset of W_t. The constant sr is interpreted as ρ_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 [4].

2.4 Parity Tree Automata

Parity tree automata are finite-state devices designed to accept or reject Kripke trees.

A *parity tree automaton* $\mathbf{A}$ is a tuple (Q, q_I, δ, Ω), 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 $p \in \mathcal{P}$; 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 automaton obtained from $\mathbf{A}$ by setting the initial state of $\mathbf{A}$ to q. We call a parity tree automaton which only uses priority 1 a *1-automaton.* Analogously, we define *0-automata.*

The computational behavior of parity tree automata is explained using the notion of a run. Assume $\mathbf{A}$ is a parity tree automaton and t a pointed Kripke structure. A *run* of $\mathbf{A}$ on t is a $(W_t \times Q)$-vertex-labeled tree $\mathbf{r} = (V_r, E_r, \rho_r, \lambda_r)$ such that ρ_r is labeled (ρ_t, 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 $p \in \lambda_t(w)$, and if $\delta(q) = \neg p$, then $p \notin \lambda_t(w)$.
- 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 $w' \in \mathrm{Suc}_t(w)$.
- If $\delta(q) = \Box q'$, then for every $w' \in \mathrm{Suc}_t(w)$ 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 $\lambda(v'') = (w, q'')$.

For a node v in $\mathbf{r}$ with $\lambda_r(v) = (w, q)$, define $\Omega_r(v) := \Omega_A(q)$. 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 $\Omega_r(v_0)\Omega_r(v_1)\Omega_r(v_2)\cdots$ is even. The run $\mathbf{r}$ is *accepting* if every infinite branch through $\mathbf{r}$ is accepting. A Kripke tree is *accepted* by $\mathbf{A}$ if there exists an accepting run of $\mathbf{A}$ on the Kripke tree. 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 Kripke tree that holds in ϕ is accepted by $\mathbf{A}$ and vice versa.

The part of the one-to-one correspondence between the fixed point alternation hierarchy of the modal μ-calculus and the hierarchy of parity tree automata relevant for this paper reads as follows:

Fact 1 *1. For every L_μ formula one can construct in linear time an equivalent parity tree automaton. For Σ_1-formulas (Π_1-formulas) one obtains 1-automata (0-automata).*

2. *Conversely, for every parity tree automaton, there exists an equivalent L_μ formula; 1- and 0-automata are equivalent to Σ_1- and Π_1-formulas, respectively.*

2.5 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. 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).

Most 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

$$T(\mathbf{A}) \cap (\mathcal{T}_P \setminus \Sigma_1(\mathbf{A})) \stackrel{?}{=} \emptyset \tag{1}$$

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 complementation 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 The Characteristic Equivalence Relation

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. 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 $t' \in T(\mathbf{A}_q)$.

Obviously, the index of the characteristic equivalence relation $\equiv_\mathbf{A}$ can be bounded exponentially in the size of $\mathbf{A}$. Also, $\equiv_\mathbf{A}$ saturates $T(\mathbf{A})$.

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.

Formally, 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

$$\begin{gathered}\lambda_t(\rho_t) = \lambda_{t'}(\rho_{t'}), \text{ and} \\ \{[t \downarrow w]_\equiv \mid w \in \mathrm{Suc}_t(\rho_t)\} = \{[t' \downarrow w]_\equiv \mid w \in \mathrm{Suc}_{t'}(\rho_{t'})\}.\end{gathered}$$

One can easily show:

Lemma 1 *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.)*

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 tree automaton $\mathbf{A}$ over P there exists a unique function $f_\mathbf{A} \colon 2^{C_A} \times 2^P \to C_A$, where $C_\mathbf{A} := \{[t]_{\equiv_\mathbf{A}} \mid t \text{ is a tree }\}$ denotes the set of $\equiv_\mathbf{A}$-equivalence classes, such that the following holds: Given a tree t and the sets $P' = \lambda_t(\rho_t)$ and $C = \{[t \downarrow w]_{\equiv_\mathbf{A}} \mid w \in \mathrm{Suc}_t(\rho_t)\}$ we have

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

4 Bottom-up Tree Automata on Infinite Trees

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 world of this frontier a certain set of states, depending on the label of the world and on whether the world is a leaf or not. Then in a bottom-up powerset fashion the ancestors are labeled with sets of states according to the transition function. (All descendants of worlds of the frontier are labeled with the empty set.) A tree is accepted if the set of states the root is labeled with is non-empty and a subset of the set of final states of the automaton.

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 i) there does not exist a path between two different worlds in $\mathcal{F}$ ($\mathcal{F}$ is an anti-chain), and ii) every maximum path in t starting from ρ_t contains a world 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 $\beta(\rho_t) \neq \emptyset$ and $\beta(\rho_t) \subseteq F$. The set of trees accepted by $\mathbf{B}$ is denoted $T(\mathbf{B}) := \{t \in \mathcal{T}_P \mid \text{there exists an accepting run of } \mathbf{B} \text{ on } t\}$.

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 (see [6] for an example). Nevertheless, we can show the following.

Proposition 1 *1. The languages accepted by bottom-up tree automata are definable in MSOL.*
2. Whenever the language accepted by a bottom-up tree automaton is closed under bisimulation, then it is Σ_1-definable.

Proving the MSOL-definability is straightforward. To show 2., we specify a 1-automaton and prove that it is equivalent to the given bottom-up tree automaton provided that the language accepted by the bottom-up automaton is bisimulation closed. The difficult part is to show that the language accepted by the 1-automaton is contained in the one accepted by the bottom-up automaton: We start with an accepting run of the 1-automaton on t, and construct an accepting run of the bottom-up automaton on a tree t' which is bisimilar to t. Now, using the closure property, we conclude that t is accepted by the bottom-up automaton.

5 The Decidability Criterion

Given a parity tree automaton $\mathbf{A}$, we define a bottom-up tree automaton $\mathbf{B_A} = (C_\mathbf{A}, P, \{[t]_{\equiv_\mathbf{A}} \mid t \in T(\mathbf{A})\}, f_\mathbf{A})$, where $f_\mathbf{A}$ is specified as at the end of Section 3. We call the language accepted by $\mathbf{B_A}$ 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,*

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

The implication from right to left follows from Proposition 1: If $T(\mathbf{A}) = \Sigma_1(\mathbf{A})$, then $\Sigma_1(\mathbf{A})$ is bisimulation closed since every language accepted by a parity tree automaton has this property. Proposition 1 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 2 *For every parity tree automaton* $\mathbf{A}$*,*

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

This lemma is proved by transfinite induction. To this end, we characterize the run β of $\mathbf{B_A}$ as the fixed point of a certain mapping which mimics the bottom-up computation of $\mathbf{B_A}$ on a tree t starting from a frontier $\mathcal{F}$ in t. Then, we show that for every $w \in \hat{\mathcal{F}}$, $[t \downarrow w]_{\equiv_\mathbf{A}} \in \beta(w)$. Given that β is an accepting run, this implies $[t]_{\equiv_\mathbf{A}} \in \{[t']_{\equiv_\mathbf{A}} \mid t' \in T(\mathbf{A})\}$, and thus, $t \in T(\mathbf{A})$.

It remains to show:

Lemma 3 *If* $T(\mathbf{A})$ *is* Σ_1*-definable, then* $T(\mathbf{A}) \subseteq \Sigma_1(\mathbf{A})$.

The idea of the proof is as follows. Given that $T(\mathbf{A})$ is Σ_1-definable, there exists a 1-automaton $\mathbf{A}'$ accepting $T(\mathbf{A})$. For every accepting run r of $\mathbf{A}'$ on some tree t, we know that every path in r is finite. In other words, there exists a frontier $\mathcal{F}$ in t such that all descendants of worlds in $\mathcal{F}$ are not occupied by states of $\mathbf{A}'$. Now, we use $\mathcal{F}$ to construct an accepting run of $\mathbf{B_A}$ on t exploiting that "below" $\mathcal{F}$ we can change the structure of the tree anyhow without leaving $T(\mathbf{A})$.

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 (1). 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.

First, we define top-down tree automata. A *top-down tree automaton* $\mathbf{D}$ is a tuple (Q, P, I, δ), defined just as for bottom-up tree automata, except that $I \subseteq Q$ is the set of *initial* states. A *run* r of $\mathbf{D}$ on $t \in \mathcal{T}_P$ is a mapping from W_t into Q such that $r(w) = \delta(\{r(w') \mid w' \in \mathrm{Suc}_t(w')\}, \lambda_t(w))$ for every $w \in W_t$. The run is called *accepting* if $r(\rho_t) \in I$. 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 $T(\mathbf{D})$.

As in the case of bottom-up tree automata, the languages accepted by top-down tree automata (even those induced by parity tree automata) are not bisimulation closed; see [6] for an example. However, we show:

Proposition 2 *1. The language accepted by a top-down tree automaton is definable in MSOL.*
2. Whenever the language accepted by a top-down tree automaton is closed under bisimulation, then it is Π_1-definable.

The main statement of this section is:

Proposition 3 *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 is of size linear in the size of* $\mathbf{B}$.

The proof works as follows. Given a bottom-up tree automaton $\mathbf{B} = (Q, P, F, \delta)$, we show that the top-down automaton

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

accepts $\mathcal{T}_P \setminus T(\mathbf{B})$. This is done in two steps. First, one shows the statement for finitely branching trees. Then, one assumes that the symmetric difference of $\mathcal{T}_P \setminus T(\mathbf{B})$ and $T(\mathbf{D_B})$ contains an infinite branching tree, and shows this implies a contradiction: Since both sets are MSOL-definable, their symmetric difference is MSOL-definable, and thus, it must contain a finitely branching tree due to the finite model property of MSOL. However, this contradicts the statement shown in the first step.— In the first step, finitely branching trees are considered to iteratively construct an accepting run of $\mathbf{D_B}$ on t given that $t \in \mathcal{T}_P \setminus T(\mathbf{B})$.

As a corollary of Theorem 2, Lemma 2, and (the proof of) Proposition 3, we obtain:

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

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

Also, from (the proof of) Proposition 3 and Theorem 2 we can derive a characterization of Π_1-definability. Given a parity tree automaton $\mathbf{A}$ over the propositional variables P, define the top-down tree automaton $\mathbf{D_A} := (C_\mathbf{A}, P, \{[t]_{\equiv_\mathbf{A}} \mid t \in T(\mathbf{A})\}, f_\mathbf{A})$. We call the set $\Pi_1(\mathbf{A}) := T(\mathbf{D_A})$ the *Π_1-test language of* $\mathbf{A}$. Now, one can show:

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

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

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 $\mathbf{D} := \mathbf{D}_{\mathbf{B_A}}$.

The proof of this proceeds in two steps. First, we show a "small branching property" for $T(\mathbf{A}) \cap T(\mathbf{D})$. And then, we describe how a typical emptiness test for parity tree automata can be modified so as to work for our problem.

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, [3].

Lemma 4 *Every tree language recognized by a parity tree automaton with n states has the n-branching property.*

We prove a similar statement for top-down automaton induced by parity tree automata.

Lemma 5 *For every parity tree automaton* $\mathbf{A}$ *with* n *states, the language accepted by* $\mathbf{D}$*, as defined above, has the* $2n$*-branching property.*

To prove this lemma, we introduce so-called modal top-down automata (see [6]) and show that $\mathbf{D}$ is equivalent to a modal top-down automaton with $2n$ states, given that $\mathbf{A}$ has n states. We then show that modal top-down automata with n' states have the n'-branching property.

As a consequence, we can note the following.

Corollary 4 *For every parity tree automaton* $\mathbf{A}$ *with* n *states, the set* $T(\mathbf{A}) \cap T(\mathbf{D})$ *has the* $(n+2n)$*-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.

Emptiness test. Consider a typical emptiness test for a parity tree automaton $\mathbf{A}$ with n states, for instance, the one described in [3]. 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 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 $f_{\mathbf{A}}(\mathcal{C}, P') = C$ for all $\mathcal{C} \subseteq C_{\mathbf{A}}$, $|\mathcal{C}| \leq 3n$, and $P' \subseteq P$.) 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 [3, 5], and deterministic exponential time is closed under complementation.

Proposition 4 L_μ-TAUTOLOGY *is polynomial-time reducible to* Σ_1-DEFINABILITY.

The idea of the proof is as follows. Let p be some new propositional variable. Define π_1 by $\pi_1 = \Diamond\nu X(p \wedge \Box X)$. 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 $\phi^* = \Diamond(\neg p \wedge \neg\phi) \wedge \pi_1$. Then, we show that ϕ is a tautology iff ϕ^* is equivalent to a Σ_1-formula.

References

1. André Arnold. The μ-calculus alternation-depth hierarchy is strict on binary trees. *Theoretical Informatics and Applications*, 33:329–339, 1999.
2. 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.
3. 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.
4. 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.
5. Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
6. R. Küsters and T. Wilke. Deciding the First Level of the μ-calculus Alternation Hierarchy. Technical Report 0209, Institut für Informatik und Praktische Mathematik, CAU Kiel, Germany, 2002. Available from http://www.informatik.uni-kiel.de/reports/2002/0209.html.
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.