Safraless Decision Procedures*

Orna Kupferman† Hebrew University

Moshe Y. Vardi‡ Rice University

December 22, 2005

Abstract

The automata-theoretic approach is one of the most fundamental approaches to developing decision procedures in mathematical logics. To decide whether a formula in a logic with the tree-model property is satisfiable, one constructs an automaton that accepts all (or enough) tree models of the formula and then checks that the language of this automaton is nonempty. The standard approach translates formulas into alternating parity tree automata, which are then translated, via Safra's determinization construction, into nondeterministic parity automata. This approach is not amenable to implementation because of the difficulty of implementing Safra's construction and the nonemptiness test for nondeterministic parity tree automata.

In this work we offer an alternative to the standard automata-theoretic approach. The crux of our approach is avoiding the use of Safra's construction and of nondeterministic parity tree automata. Our approach goes instead via universal co-Büchi tree automata and nondeterministic Büchi tree automata. While our translations have the same complexity as the standard approach, they are significantly simpler, less difficult to implement, and have practical advantages like being amenable to optimizations and a symbolic implementation.

1 Introduction

The automata-theoretic approach is one of the most fundamental approaches to developing decision procedures in mathematical logics [Rab69]. It is based on the fact that many logics enjoy the *tree-model property*; if a formula in the logic is satisfiable then it has a tree (or a tree-like) model [Var97]. To decide whether a formula ψ in such a logic is satisfiable, one constructs an automaton $\mathcal{A}_\psi$ that accepts all (or enough) tree models of ψ and then checks that the language of $\mathcal{A}_\psi$ is nonempty.

*A preliminary version of this paper appears in the Proceedings of the 46th IEEE Symposium on Foundations of Computer Science.

†Address:School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il. Supported in part by BSF grant 9800096 and by a grant from Minerva.

‡Address: Department of Computer Science, Rice University, Houston, TX 77251-1892, U.S.A., Email: vardi@cs.rice.edu. Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, EIA-0086264, and ANI-0216467, by BSF grant 9800096, by Texas ATP grant 003604-0058-2003, and by a grant from the Intel Corporation.

The automata-theoretic approach was developed first for monadic logics over finite words [Büc60, Elg61, Tra62]. It was then extended to infinite words in [Büc62], to finite trees in [TW68], and finally generalized to infinite trees in [Rab69]. Following Rabin's fundamental result, SnS, the monadic theory of infinite trees, served for many years as a proxy for the automata-theoretic approach – to show decidability of a logic one could simply demonstrate an effective reduction of that logic to SnS, e.g., [Gab72, KP84]. Unfortunately, the complexity of SnS is known to be nonelementary (i.e., it cannot be bounded by a stack of exponential of a fixed height) [Mey75]. Thus, in the early 1980s, when decidability of highly expressive logics became of practical interest in areas such as formal verification and AI [GL94, Koz83], and complexity-theoretic considerations started to play a greater role, the original automata-theoretic idea was revived; by going from various logics to automata directly, decision procedures of elementary complexity were obtained for many logics, e.g., [SE84, Str82, VW86].

By the mid 1980s, the focus was on using automata to obtain tighter upper bounds. This required progress in the underlying automata-theoretic techniques. Such breakthrough progress was attained by Safra [Saf88], who described an optimal determinization construction for automata on infinite words, and by Emerson and Jutla [EJ88] and Pnueli and Rosner [PR89], who described improved algorithms for parity tree automata (the term "parity" refers to the accompanying acceptance condition of the automaton). Further simplification was obtained by the introduction of alternating automata on infinite trees [EJ91, MS85]. In the now standard approach for checking whether a formula ψ is satisfiable, one follows these steps: (1) construct an alternating parity tree automaton $\mathcal{A}_\psi$ that accepts all (or enough) tree models of ψ, (The translation from formulas to alternating parity tree automata is well known (c.f., [KVW00]) and will not be addressed in this paper.) (2) translate this automaton to a nondeterministic parity tree automaton $\mathcal{A}^n_\psi$, and (3) check that the language of $\mathcal{A}^n_\psi$ is nonempty.

While the now standard automata-theoretic approach yielded significantly improved upper bounds (in some cases reducing the upper time bound from octuply exponential [Str82] to singly exponential [Var98]), it proved to be not too amenable to implementation. First, the translation in step (2) is very complicated: removing alternation from alternating tree automata involves determinization of word automata, and Safra's construction proved quite resistant to efficient implementation [THB95]. An alternative removal of alternation is described in [MS95]. Like Safra's construction, however, this translation is very complicated [ATW05]. Second, the best-known algorithms for parity-tree-automata emptiness are exponential [Jur00]. Thus, while highly optimized software packages for automata on finite words and finite trees have been developed over the last few years [EKM98], no such software has been developed for automata on infinite trees.

In this paper we offer an alternative to the standard automata-theoretic approach. The crux of our approach is avoiding the use of Safra's construction and of nondeterministic parity tree automata. In the approach described here, one checks whether a formula ψ is satisfiable by following these steps: (1) construct an alternating parity tree automaton $\mathcal{A}_\psi$ that accepts all (or enough) tree models of ψ, (2) reduce[1] $\mathcal{A}_\psi$ to a uni-

[1]We use "reduce A_1 to A_2", rather than "translate A_1 to A_2" to indicate that A_1 need not be equivalent to A_2, yet the language of $\mathcal{A}_1$ is empty iff the language of $\mathcal{A}_2$ is empty.

versal co-Büchi automaton $\mathcal{A}^c_\psi$, (3) reduce $\mathcal{A}^c_\psi$ to an alternating weak tree automaton $\mathcal{A}^w_\psi$, (4) translate $\mathcal{A}^w_\psi$ to a nondeterministic Büchi tree automaton $\mathcal{A}^n_\psi$, and (5) check that the language of $\mathcal{A}^n_\psi$ is nonempty. The key is avoiding Safra's construction, by using universal co-Büchi automata instead of deterministic parity automata.[2] Universal automata have the desired property, enjoyed also by deterministic automata but not by nondeterministic automata, of having the ability to run over all branches of an input tree. In addition, the co-Büchi acceptance condition is much simpler than the parity condition. This enables us to solve the nonemptiness problem for universal co-Büchi tree automata by reducing them into nondeterministic Büchi tree automata (the reduction goes through alternating weak tree automata [MSS88], and there is no need for the parity acceptance condition). The nonemptiness problem for nondeterministic Büchi tree automata is much simpler than the nonemptiness problem for nondeterministic parity tree automata and it can be solved symbolically and in quadratic time [VW86]. We also show that in some cases (in particular, the *realizability and synthesis* [PR89] problems for LTL specifications), it is possible to skip the construction of an alternating parity automaton and go directly to a universal co-Büchi automaton.

Our translations and reductions are significantly simpler than the standard approach, making them less difficult to implement, both explicitly and symbolically. These advantages are obtained with no increase in the complexity. In fact, as discussed in Section 6, our construction is amenable to several optimization techniques.

2 Preliminaries

Given a set D of directions, a *D-tree* is a set $T \subseteq D^*$ such that if $x \cdot c \in T$, where $x \in D^*$ and $c \in D$, then also $x \in T$. If $T = D^*$, we say that T is a full D-tree. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $x \in T$, the nodes $x \cdot c$, for $c \in D$, are the *successors* of x. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or there exists a unique $c \in D$ such that $x \cdot c \in \pi$. Given an alphabet Σ, a *Σ-labeled D-tree* is a pair $\langle T, \tau \rangle$ where T is a tree and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ.

A *transducer* is a labeled finite graph with a designated start node, where the edges are labeled by D and the nodes are labeled by Σ. A Σ-labeled D-tree is *regular* if it is the unwinding of some transducer. More formally, a transducer is a tuple $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L \rangle$, where D is a finite set of directions, Σ is a finite alphabet, S is a finite set of states, $s_{in} \in S$ is an initial state, $\eta : S \times D \to S$ is a deterministic transition function, and $L : S \to \Sigma$ is a labeling function. We define $\eta : D^* \to S$ in the standard way: $\eta(\varepsilon) = s_{in}$, and for $x \in D^*$ and $d \in D$, we have $\eta(x \cdot d) = \eta(\eta(x), d)$. Intuitively, A Σ-labeled D-tree $\langle D^*, \tau \rangle$ is regular if there exists a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L \rangle$ such that for every $x \in D^*$, we have $\tau(x) = L(\eta(x))$. We then say that the size of the regular tree $\langle D^*, \tau \rangle$, denoted $\|\tau\|$, is $|S|$, the number of states of $\mathcal{T}$.

[2] A note to readers who are discouraged by the fact our method goes via several intermediate automata: it is possible to combine the reductions into one construction, and in fact we describe here also a direct translation of universal co-Büchi automata into nondeterministic Büchi automata. In practice, however, it is beneficial to have many intermediate automata, as each intermediate automaton undergoes optimization constructions that are suitable for its particular type [Fri03, FW02, GKSV03].

For a set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas **true** (an empty conjunction) and **false** (an empty disjunction). For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning **true** to elements in Y and assigning **false** to elements in $X \setminus Y$ makes θ true. An *Alternating tree automaton* is $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$, where Σ is the input alphabet, D is a set of directions, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$ is a transition function, $q_{in} \in Q$ is an initial state, and α specifies the acceptance condition (a condition that defines a subset of Q^ω; we define several types of acceptance conditions below).

The alternating automaton $\mathcal{A}$ runs on Σ-labeled full D-trees. A *run* of $\mathcal{A}$ over a Σ-labeled D-tree $\langle T, \tau \rangle$ is a $(T \times Q)$-labeled $\mathbb{N}$-tree $\langle T_r, r \rangle$. Each node of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T. The labels of a node and its successors have to satisfy the transition function. Formally, $\langle T_r, r \rangle$ satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = \langle \varepsilon, q_{in} \rangle$.
2. Let $y \in T_r$ with $r(y) = \langle x, q \rangle$ and $\delta(q, \tau(x)) = \theta$. Then there is a (possibly empty) set $S = \{(c_0, q_0), (c_1, q_1), \ldots, (c_{n-1}, q_{n-1})\} \subseteq D \times Q$, such that S satisfies θ, and for all $0 \leq i \leq n-1$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = \langle x \cdot c_i, q_i \rangle$.

For example, if $\langle T, \tau \rangle$ is a $\{0,1\}$-tree with $\tau(\varepsilon) = a$ and $\delta(q_{in}, a) = ((0, q_1) \vee (0, q_2)) \wedge ((0, q_3) \vee (1, q_2))$, then, at level 1, the run $\langle T_r, r \rangle$ includes a node labeled $(0, q_1)$ or a node labeled $(0, q_2)$, and includes a node labeled $(0, q_3)$ or a node labeled $(1, q_2)$. Note that if, for some y, the transition function δ has the value **true**, then y need not have successors. Also, δ can never have the value **false** in a run.

A run $\langle T_r, r \rangle$ is accepting if all its infinite paths satisfy the acceptance condition. Given a run $\langle T_r, r \rangle$ and an infinite path $\pi \subseteq T_r$, let $inf(\pi) \subseteq Q$ be such that $q \in inf(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $r(y) \in T \times \{q\}$. That is, $inf(\pi)$ contains exactly all the states that appear infinitely often in π. We consider here three acceptance conditions defined as follows[3]

- A path π satisfies a *Büchi* acceptance condition $\alpha \subseteq Q$ if and only if $inf(\pi) \cap \alpha \neq \emptyset$.
- A path π satisfies a *co-Büchi* acceptance condition $\alpha \subseteq Q$ if and only if $inf(\pi) \cap \alpha = \emptyset$.
- A path π satisfies a *parity* acceptance condition $\alpha = \{F_1, F_2, \ldots, F_h\}$ with $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_h = Q$ iff the minimal index i for which $inf(\pi) \cap F_i \neq \emptyset$ is even. The number h of sets in α is called the *index* of the automaton.

For the three conditions, an automaton accepts a tree iff there exists a run that accepts it. We denote by $\mathcal{L}(\mathcal{A})$ the set of all Σ-labeled trees that $\mathcal{A}$ accepts.

[3] In the proof of Theorem 4.3, we also refer to the Rabin and Streett conditions, but their definition is irrelevant for the proof.

Below we discuss some special cases of alternation automata. The alternating automaton $\mathcal{A}$ is *nondeterministic* if for all the formulas that appear in δ, if (c_1, q_1) and (c_2, q_2) are conjunctively related, then $c_1 \neq c_2$. (i.e., if the transition is rewritten in disjunctive normal form, there is at most one element of $\{c\} \times Q$, for each $c \in D$, in each disjunct). The automaton $\mathcal{A}$ is *universal* if all the formulas that appear in δ are conjunctions of atoms in $D \times Q$, and $\mathcal{A}$ is *deterministic* if it is both nondeterministic and universal. The automaton $\mathcal{A}$ is a *word* automaton if $|D| = 1$.

In [MSS86], Muller et al. introduce *alternating weak tree automata*. In a weak automaton, we have a Büchi acceptance condition $\alpha \subseteq Q$ and there exists a partition of Q into disjoint sets, $Q_1, \ldots, Q_m$, such that for each set Q_i, either $Q_i \subseteq \alpha$, in which case Q_i is an *accepting set*, or $Q_i \cap \alpha = \emptyset$, in which case Q_i is a *rejecting set*. In addition, there exists a partial order $\leq$ on the collection of the Q_i's such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, \sigma)$, for some $\sigma \in \Sigma$, we have $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. It follows that every infinite path of a run of an alternating weak automaton ultimately gets "trapped" within some Q_i. The path then satisfies the acceptance condition if and only if Q_i is an accepting set.

We denote each of the different types of automata by three letter acronyms in $\{D, N, U, A\} \times \{B, C, P, R, S, W\} \times \{W, T\}$, where the first letter describes the branching mode of the automaton (deterministic, nondeterministic, universal, or alternating), the second letter describes the acceptance condition (Büchi, co-Büchi, parity, Rabin, Streett, or weak), and the third letter describes the object over which the automaton runs (words or trees). For example, APT are alternating parity tree automata and UCT are universal co-Büchi tree automata.

3 From APT to NBT via UCT

UCT are a special case of APT: the transition function of a UCT contains only conjunctions and the acceptance condition corresponds to a parity condition of index 2. UCT are indeed strictly less expressive than APT. Consider for example the language $\mathcal{L}$ of $\{0, 1\}$-labeled trees where $\langle T, \tau \rangle \in \mathcal{L}$ iff there is a path $\pi \subseteq T$ such that for infinitely many $x \in \pi$, we have $\tau(x) = 0$. It is easy to construct an APT (in fact, even an NBT [Rab70]) that recognizes $\mathcal{L}$. By [Rab70], however, no NBT can recognize the complement of $\mathcal{L}$. Hence, by [MSS86], no UCT can recognize $\mathcal{L}$.

In this section we show that though UCT are less expressive than APT, they are very powerful. On the one hand, the emptiness problem for APT is easily reducible to the emptiness problem for UCT. On the other hand, it is easy to translate UCT into NBT so that emptiness is preserved (that is, the NBT is empty iff the UCT is empty). Thus, as discussed in Section 1, traditional decidability algorithms that end up in a complicated APT nonemptiness check, can be much simplified. We also show that UCT are useful for tasks traditionally assigned to APT. Thus, in many cases it is possible to skip the construction of an APT and go directly to a UCT. This includes the realizability and synthesis problems for LTL specifications [PR89], and the problem of translating an LTL specification into a DPW with a minimal index. We will discuss these applications in Section 5.

3.1 From APT to UCT

Consider an APT $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$. Recall that the transition function $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$ maps a state and a letter to a formula in $\mathcal{B}^+(D \times Q)$. A *restriction* of δ is a partial function $\eta : Q \to 2^{D \times Q}$. For a letter $\sigma \in \Sigma$, we say that a restriction η is *relevant* to σ if for all $q \in Q$ for which $\delta(q, \sigma)$ is satisfiable (i.e., $\delta(q, \sigma)$ is not **false**), the set $\eta(q)$ satisfies $\delta(q, \sigma)$. If $\delta(q, \sigma)$ is not satisfiable, then $\eta(q)$ is undefined. Intuitively, by choosing the atoms that are going to be satisfied, η removes the nondeterminism in δ. Let F be the set of restrictions of δ. Note that $|F|$ is exponential in $|\delta|$. A *running strategy* of $\mathcal{A}$ for a Σ-labeled D-tree $\langle T, \tau \rangle$ is an F-labeled tree $\langle T, f \rangle$. We say that $\langle T, f \rangle$ is *relevant* to $\langle T, \tau \rangle$ if for all $x \in T$, the restriction $f(x)$ is relevant to $\tau(x)$. When $\langle T, f \rangle$ is relevant to $\langle T, \tau \rangle$, it induces a unique (up to the order of siblings in the run tree) run $\langle T_f, r_f \rangle$ of $\mathcal{A}$ on $\langle T, \tau \rangle$: whenever the run $\langle T_f, r_f \rangle$ is in state q as it reads a node $x \in T$, it proceeds according to $f(x)(q)$. Formally, $\langle T_f, r_f \rangle$ is a $(T \times Q)$-labeled $\mathbb{N}$-tree that satisfies the following:

1. $\varepsilon \in T_f$ and $r_f(\varepsilon) = (\varepsilon, q_{in})$.
2. Consider a node $y \in T_f$ with $r_f(y) = (x, q)$. Let $f(x)(q) = \{(c_0, q_0), (c_1, q_1), \ldots, (c_{n-1}, q_{n-1})\} \subseteq D \times Q$. For all $0 \leq i \leq n-1$, we have $y \cdot i \in T_r$ and $r_f(y \cdot i) = \langle x \cdot c_i, q_i \rangle$. The only children of y in T_f are these required for the satisfaction of the above.

We say that a running strategy $\langle T, f \rangle$ is *good* for $\langle T, \tau \rangle$ if $\langle T, f \rangle$ is relevant to $\langle T, \tau \rangle$ and the run $\langle T_f, r_f \rangle$ is accepting. Note that a node x of $\langle T, f \rangle$ may be read by several copies of $\mathcal{A}$. All these copies proceed according to the restriction $f(x)$, regardless the history of the run so far. Thus, the run $\langle T_f, r_f \rangle$ is *memoryless*. By [EJ91], an APT $\mathcal{A}$ accepts $\langle T, \tau \rangle$ iff $\mathcal{A}$ has a memoryless accepting run on $\langle T, \tau \rangle$. Hence the following theorem.

Theorem 3.1 [EJ91] *The APT $\mathcal{A}$ accepts $\langle T, \tau \rangle$ iff there exists a running strategy $\langle T, f \rangle$ that is good for $\langle T, \tau \rangle$.*

Annotating input trees with restrictions enables us to transform an APT to a UCT with polynomially many states: let $\Sigma' \subseteq \Sigma \times F$ be such that for all $\langle \sigma, \eta \rangle \in \Sigma'$, we have that η is relevant to σ. Note that since we restrict attention to pairs in which η is relevant to σ, the size of Σ' is still exponential in $|\delta|$. We refer to a Σ'-labeled tree as $\langle T, (\tau, f) \rangle$, where τ and f are the projections of Σ' on Σ and F, respectively.

Theorem 3.2 *Let $\mathcal{A}$ be an APT with n states, transition function of size m, and index h. There is a UCT $\mathcal{A}'$ with $O(nh)$ states and alphabet of size $2^{O(m)}$ such that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$.*

Proof: The UCT $\mathcal{A}'$ accepts a Σ'-labeled D-tree iff $\mathcal{A}$ accepts its projection on Σ. For that, $\mathcal{A}'$ accepts a tree $\langle T, (\tau, f) \rangle$ iff $\langle T, f \rangle$ is good for $\langle T, \tau \rangle$. By Theorem 3.1, it then follows that $\mathcal{A}'$ accepts $\langle T, (\tau, f) \rangle$ iff $\mathcal{A}$ accepts $\langle T, \tau \rangle$. Note that since Σ' contains only pairs $\langle \sigma, \eta \rangle$ for which η is relevant to σ, it must be that f is relevant to τ, thus $\mathcal{A}'$ only has to check that all the paths in the run tree $\langle T_f, r_f \rangle$ satisfy the parity acceptance

condition. Since the running strategy $\langle T, f\rangle$ removes the nondeterminism in δ, the construction of $\mathcal{A}'$ is similar to a translation of a universal parity tree automaton into a universal co-Büchi tree automaton, which is dual to the known translation of Rabin (or co-parity) word automata to Büchi word automata [Cho74].

Formally, let $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha\rangle$ with $\alpha = \{F_1, F_2, \ldots, F_{2h}\}$, and let $F_0 = \emptyset$. We define the UCT $\mathcal{A}' = \langle \Sigma', Q \times \{0, \ldots, h-1\}, \delta', \langle q_{in}, 0\rangle, \alpha'\rangle$, where[4]

- For every $q \in Q$, $\sigma \in \Sigma$, and $\eta \in F$, we have
  - $\delta'(\langle q, 0\rangle, \langle \sigma, \eta\rangle) = \bigwedge_{0 \le i < h} \bigwedge_{(c,s)\in(\eta(q)\setminus(D\times F_{2i}))} (c, \langle s, i\rangle)$.
  - For every $1 \le i < h$, we have $\delta'(\langle q, i\rangle, \langle \sigma, \eta\rangle) = \bigwedge_{(c,s)\in(\eta(q)\setminus(D\times F_{2i}))} (c, \langle s, i\rangle)$.
- $\alpha' = \bigcup_{0\le i<h}(F_{2i+1} \times \{i\})$.

The automaton $\mathcal{A}'$ consists of h copies of $\mathcal{A}$, with the states of the i-th copy being labeled by i, for $0 \le i \le h-1$. A copy associated with index i, for $0 \le i \le h-1$, checks that if a path in the run $\langle T_f, r_f\rangle$ visits F_{2i} only finitely often, then the path also visits F_{2i+1} only finitely often. The run of $\mathcal{A}'$ starts at the "master copy", $Q \times \{0\}$, and it branches as suggested by the restriction in the input. From the master copy, $\mathcal{A}'$ branches to the other copies: for each transition of $\mathcal{A}$ suggested by η, the automaton $\mathcal{A}'$ branches to the master copy and to all the other $h-1$ copies. Once $\mathcal{A}'$ moves to a copy associated with index i, it stays there forever, unless when it has to move to a state from F_{2i}. The acceptance condition of $\mathcal{A}'$ guarantees that if $\mathcal{A}'$ stays in the i-th copy forever (in which case it reaches the i-th copy in a suffix of a path of $\langle T_f, r_f\rangle$ that has no visits to F_{2i}, indicating that corresponding path visits F_{2i} only finitely often), it visits only finitely many states in F_{2i+1}. □

As discussed in Section 1, APT are of special interest as it is possible to translate μ-calculus formulas into APT. By translating other types of alternating tree automata into UCT, our approach can be applied to other temporal logics as well. We describe such two cases below. The logic CTL* is weaker than the μ-calculus and CTL* formulas can be translated into alternating *hesitant* automata [KVW00]. Since the acceptance condition of a hesitant automaton is similar to a Rabin condition with a single pair, the construction of the UCT of Theorem 3.2 in that case involves only a linear blow-up in the state space.

An extension of the standard μ-calculus, called the *full μ-calculus*, which includes both forward and backward modalities, is studied in [Var98]. It is shown there that a full μ-calculus formula can be translated into a *two-way* APT, denoted 2APT. The emptiness problem for 2APT is solved in [Var98] via a reduction to NPT that uses Safra's construction. A closer examination of the construction in [Var98] shows that it proceeds in two steps: (1) It is shown that a 2APT $\mathcal{A}$ can be translated into a UPT $\mathcal{A}'$ over a larger alphabet such that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$; (2) The UPT $\mathcal{A}'$ is translated to an NPT using Safra's construction. Using Theorem 3.2, we can translate $\mathcal{A}'$ into a UCT and skip step (2). Thus, we obtain a Safraless decision procedure for the full μ-calculus.

[4]Note that an empty conjunction evaluates to **false**.

3.2 From UCT to NBT

We now describe an emptiness preserving translation of UCT to NBT. The correctness proof of the construction is given in Section 4. There, we also suggest to use AWT as an intermediate step in the construction. While this adds a step to our chain of reductions, it enables further optimizations of the result.

Theorem 3.3 *Let $\mathcal{A}$ be a UCT with n states. There is an NBT $\mathcal{A}'$ over the same alphabet such that $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$, and the number of states in $\mathcal{A}'$ is $2^{O(n^2 \log n)}$.*

Proof: Let $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$, and let $k = (2n!)n^{2n}3^n(n+1)/n!$. Note that k is $2^{O(n \log n)}$. Let $\mathcal{R}$ be the set of functions $f : Q \to \{0, \ldots, k\}$ in which $f(q)$ is even for all $q \in \alpha$. For $g \in \mathcal{R}$, let $odd(g) = \{q : g(q) \text{ is odd}\}$. We define $\mathcal{A}' = \langle \Sigma, D, Q', q'_{in}, \delta', \alpha' \rangle$, where

- $Q' = 2^Q \times 2^Q \times \mathcal{R}$.
- $q'_{in} = \langle \{q_{in}\}, \emptyset, g_0 \rangle$, where g_0 maps all states to k.
- For $q \in Q$, $\sigma \in \Sigma$, and $c \in D$, let $\delta(q, \sigma, c) = \delta(q, \sigma) \cap (\{c\} \times Q)$. For two functions g and g' in $\mathcal{R}$, a letter σ, and direction $c \in D$, we say that g' *covers* $\langle g, \sigma, c \rangle$ if for all q and q' in Q, if $q' \in \delta(q, \sigma, c)$, then $g'(q') \leq g(q)$. Then, for all $\langle S, O, g \rangle \in Q'$ and $\sigma \in \Sigma$, we define δ as follows.
  - If $O \neq \emptyset$, then $\delta'(\langle S, O, g \rangle, \sigma) =$
$$\bigwedge_{c \in D} \bigvee_{g_c \text{ covers } \langle g, \sigma, c \rangle} \langle \delta(S, \sigma, c), \delta(O, \sigma, c) \setminus odd(g_c), g_c \rangle.$$
  - If $O = \emptyset$, then $\delta'(\langle S, O, g \rangle, \sigma) =$
$$\bigwedge_{c \in D} \bigvee_{g_c \text{ covers } \langle g, \sigma, c \rangle} \langle \delta(S, \sigma, c), \delta(S, \sigma, c) \setminus odd(g_c), g_c \rangle.$$
- $\alpha' = 2^Q \times \{\emptyset\} \times \mathcal{R}$.

□

3.3 Complexity

Combining Theorems 3.2 and 3.3, we get the desired reduction from the nonemptiness problem for APT to the nonemptiness problem for NBT:

Theorem 3.4 *Let $\mathcal{A}$ be an APT with n states, transition function of size m, and index h. There is an NBT $\mathcal{A}'$ with $2^{O(n^2 h^2 \log nh)}$ states and alphabet of size $2^{O(m)}$ such that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$.*

We now analyze the complexity of the nonemptiness algorithm for APT that follows.

Theorem 3.5 *The nonemptiness problem for an APT with n states, transition function of size m, and index h can be solved in time $2^{O(\log|D|+m+n^2h^2\log nh)}$.*

Proof: By Theorem 3.4, the NBT induced by the APT has $2^{O(n^2h^2\log nh)}$ states and alphabet of size $2^{O(m)}$. The transitions of the NBT are such that the successors of a certain state in a particular direction are independent of its successors in other directions. Thus, the transition function of the NBT specifies for each state, letter, and direction, a set of possible states, and it is therefore of size $2^{O(\log|D|+m+n^2h^2\log nh)}$. The nonemptiness problem for NBT can be solved in time quadratic in the size of the transition function [VW86], so we get $2^{O(\log|D|+m+n^2h^2\log nh)}$. □

This coincides with the known upper bound that is based on Safra's construction. Indeed, there, one first constructs a DPT with $(nh)^{O(nh)}$ states and index $O(nh)$. The alphabet size of the DPT is $2^{O(m)}$. Since the DPT is deterministic, the size of its transition function is the product of its state space size, alphabet size, and branching degree, which is $2^{O(\log|D|+m+nh\log(nh))}$ The nonemptiness problem for DPT with transition function of size x, state space of size y, and index z requires time $xy^{O(z)}$ [Jur00], so we get $2^{O(\log|D|+m+n^2h^2\log nh))}$, which coincides with our bound. The main advantage of our approach is the simplicity of the algorithm; the complexity analysis here just serves to show that this simplicity does not involve a worse upper bound.

4 A proof of the UCT to NBT construction

Recall that runs of alternating tree automata are labeled trees. By merging nodes that are roots of identical subtrees, it is possible to maintain runs in graphs. In Section 4.1, we prove a bounded-size run graphs property for UCT. In Section 4.2, we show how the bounded-size property enables a simple translation of UCT to AWT. In Section 4.3, we translate these AWT to NBT. Combining the translations results in the construction presented in Theorem 3.3.

4.1 Useful Observations

Consider a UCT $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha\rangle$. Recall that a run $\langle T_r, r\rangle$ of $\mathcal{A}$ on a Σ-labeled D-tree $\langle T, \tau\rangle$ is a $(T \times Q)$-labeled tree in which a node y with $r(y) = \langle x, q\rangle$ stands for a copy of $\mathcal{A}$ that visits the state q when it reads the node x. Assume that $\langle T, \tau\rangle$ is regular, and is generated by a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L\rangle$. For two nodes y_1 and y_2 in T_r, with $r(y_1) = \langle x_1, q_1\rangle$ and $r(y_2) = \langle x_2, q_2\rangle$, we say that y_1 and y_2 are *similar* iff $q_1 = q_2$ and $\eta(x_1) = \eta(x_2)$. By merging similar nodes into a single vertex, we can represent the run $\langle T_r, r\rangle$ by a finite graph $G_r = \langle V, E\rangle$, where $V = S \times Q$ and $E(\langle s, q\rangle, \langle s', q'\rangle)$ iff there is $c \in D$ such that $(c, q') \in \delta(q, L(s))$ and $\eta(s, c) = s'$. We restrict G_r to vertices reachable from the vertex $\langle s_{in}, q_{in}\rangle$. We refer to G_r as the *run graph of $\mathcal{A}$ on $\mathcal{T}$*. A run graph of $\mathcal{A}$ is then a run graph of $\mathcal{A}$ on some transducer $\mathcal{T}$. We say that G_r is accepting iff every infinite path of G_r has only finitely many α-vertices (vertices in $S \times \alpha$). Since $\mathcal{A}$ is universal and $\mathcal{T}$ is deterministic, the run $\langle T_r, r\rangle$

is *memoryless* in the sense that the merging does not introduce to G_r paths that do not exist in $\langle T_r, r\rangle$, and thus, it preserves acceptance. Formally, we have the following:

Lemma 4.1 *Consider a UCT $\mathcal{A}$. Let $\langle T, \tau\rangle$ be a tree generated by a transducer $\mathcal{T}$. The run tree $\langle T_r, r\rangle$ of $\mathcal{A}$ on $\langle T, \tau\rangle$ is accepting iff the run graph G_r of $\mathcal{A}$ on $\mathcal{T}$ is accepting.*

Proof: We say that a path $\pi = y_0, y_1 \cdot y_2 \cdots$ of $\langle T_r, r\rangle$ corresponds to a path $\pi' = \langle s_0, q_0\rangle, \langle s_1, q_1\rangle, \langle s_2, q_2\rangle, \ldots$ of G_r iff $s_0 = s_{in}$, $q_0 = q_{in}$, and there is a path $x_0, x_1, x_2, \ldots$ of T, with $x_{i+1} = x_i \cdot c_i$, such that for all $i \geq 0$, we have that $r(y_i) = \langle x_i, q_i\rangle$ and $\eta(s_i, c_i) = s_{i+1}$. Thus, π' describes the states of $\mathcal{T}$ and $\mathcal{A}$ that the copy of $\mathcal{A}$ whose evolution is recorded in the path π visits. Clearly, π has infinitely many nodes y_i with $r(y_i) \in T \times \alpha$ iff π' visits infinitely many α-vertices. By the definition of G_r, each path of $\langle T_r, r\rangle$ corresponds to a single path of G_r. Also, each path π' of G_r has at least one path π of $\langle T_r, r\rangle$ such that π corresponds to π'. To see this, note that since $\langle T, \tau\rangle$ is induced by $\mathcal{T}$, then $T = D^*$ and for all $x \in D^*$, we have that $\tau(x) = L(\eta(x))$. In addition, by the definition of G_r, for all $i \geq 0$ there is $c_i \in D$ such that $(c_i, q_{i+1}) \in \delta(q_i, L(s_i))$ and $\eta(s_i, c_i) = s_{i+1}$; the sequence of these x_i's induces a path $x_0, x_1, x_2, x_3, \ldots$ of T, with $x_{i+1} = x_i \cdot c_i$. The run of $\mathcal{A}$ on $\langle T, \tau\rangle$ contains a copy that reads this path and visits $q_0, q_1, q_2, \ldots$, and the path π of $\langle T_r, r\rangle$ describes this copy. Hence, $\langle T_r, r\rangle$ has an infinite path that visits infinitely many states in α iff G_r has an infinite path with infinitely many α-vertices, and we are done. □

Note that G_r is finite, and its size is bounded by $S \times Q$. We now bound S and get a bounded-size run-graph property for UCT. The size of S depends on the blow-up involved in NBW determinization. By [Saf88], an NBW with n states can be translated to an equivalent deterministic Streett word automaton (DSW) with $2^{O(n \log n)}$ states. Here, we need an exact bound, so we analyze the complexity of the construction in [Saf88] carefully:

Lemma 4.2 *Given an NBW with n states, it is possible to construct an equivalent DSW with $(2n!)n^{2n}3^n(n+1)/n!$ states.*

Proof: By [Saf88, Saf89], the state space of the DSW is the set of tuples $\langle t, \pi, i_1, i_2\rangle$, where t is a labeled ordered tree over n nodes (a tree in which the successors of each node are ordered), π is a permutation of $1, \ldots, n$, and $1 \leq i_1, i_2 \leq n+1$. Each node of t is labeled by a number in $\{1, \ldots, n\}$ (the names of the node) and a color in $\{0, 1, 2\}$. In addition, each node is labeled by a subset of $\{1, \ldots, n\}$, corresponding to the subset of states of the NBW associated with the node. The trees t are such that if a node is labeled with a subset containing state $i \in \{1, \ldots, n\}$, then so are its ancestors. Also, the state i cannot belong to the subsets labeling other nodes of the same level. Therefore, the labeling of the nodes by subsets of $\{1, \ldots, n\}$ can be encoded by a function that maps a state $i \in \{1, \ldots, n\}$ to the lowest leftmost node such that i belongs to its labeled subset. Thus, the number of different labels is the product of n^n (for the name), 3^n (for the color), and n^n (for the subsets). There are $cat(n-1)$ ordered trees over n nodes, where cat stands for Catalan number. The explicit formula for $cat(n)$ is $(2n!)/(n!(n+1)!)$. This, together with the $n^n 3^n n^n$ factor for the possible labels of the nodes, and the $n!(n+1)^2$ factor for π, i_1, and i_2, gives a

$(2n!)2^{2n\log n}3^n n!(n+1)^2/(n!(n+1)!)$ bound, which equals $(2n!)n^{2n}3^n(n+1)/n!$. □

Note that applying Stirling's Approximation $n! \approx \sqrt{2\pi n}(n/e)^n$, we can approximate the bound in Theorem 4.2 by $\sqrt{4\pi n}(2n/e)^{2n}n^{2n}3^n(n+1)/\sqrt{2\pi n}(n/e)^n$, which, for $n \geq 9$, is bounded by $n^{3n}6^n$.

We can now obtain a bounded-size run-graph property for UCT.

Theorem 4.3 *A UCT $\mathcal{A}$ with n states is not empty iff $\mathcal{A}$ has an accepting run graph with at most $(2n!)n^{2n+1}3^n(n+1)/n!$ vertices.*

Proof: Assume first that $\mathcal{A}$ has an accepting run graph G_r (of any size) on some transducer $\mathcal{T}$. Let $\langle T, \tau\rangle$ be the tree generated by $\mathcal{T}$. Thus, $T = D^*$ and for all $x \in D^*$ we have that $\tau(x) = L(\eta(x))$. Consider the run $\langle T_r, r\rangle$ of $\mathcal{A}$ on $\langle T, \tau\rangle$. By Lemma 4.1, $\langle T_r, r\rangle$ is accepting. Hence, $\mathcal{A}$ is not empty.

For the other direction, consider the UCT $\mathcal{A}$. By [EJ91], there is a DRT $\mathcal{A}^d$ equivalent to $\mathcal{A}$, which is constructed as follows. Let $\mathcal{A}'$ be an NBW that runs over a branch of an input tree for $\mathcal{A}$ and checks whether $\mathcal{A}$ has a rejecting path over this branch. The NBW $\mathcal{A}'$ has the same state space as $\mathcal{A}$. Let $\mathcal{A}''$ be a DSW $\mathcal{A}''$ that is equivalent to $\mathcal{A}'$ (by Lemma /4.2). Now, we complement $\mathcal{A}''$ (by dualizing its acceptance condition) and run the complementary DRW over all branches of the input tree to check that all paths of the run tree of $\mathcal{A}$ are accepting. This yields the DRT $\mathcal{A}^d$ that is equivalent to $\mathcal{A}$.

By Lemma 4.2, the DRT $\mathcal{A}^d$ has at most $n' = (2n!)n^{2n}3^n(n+1)/n!$ states. By [Eme85], an NRT with n' states is not empty iff it accepts a regular tree generated by a transducer with n' states. The state space of the run graph of $\mathcal{A}$ on such a transducer is then bounded by $nn' = (2n!)n^{2n+1}3^n(n+1)/n!$. Since the run of $\mathcal{A}$ on the tree is accepting, Lemma 4.1 implies that so is the run graph. □

We note that an improvement in the upper bound of Theorem 4.3 would lead to an improvement in the complexity of our decision procedure. In fact, as we further discuss in Section 7, even an improvement in the width of such a run graph would improve the complexity of the decision procedure.

Consider a graph $G \subseteq G_r$. We say that a vertex $\langle s, q\rangle$ is *finite* in G iff all the paths that start at $\langle s, q\rangle$ are finite. We say that a vertex $\langle s, q\rangle$ is α-*free* in G iff all the vertices in G that are reachable from $\langle s, q\rangle$ are not α-vertices. Note that, in particular, an α-free vertex is not an α-vertex.

Given a run $\langle T_r, r\rangle$, we define a sequence $G_0 \supseteq G_1 \supseteq G_2 \supseteq \ldots$ of graphs, subgraphs of G_r, as follows.

- $G_0 = G_r$.
- $G_{2i+1} = G_{2i} \setminus \{\langle s, q\rangle \mid \langle s, q\rangle \text{ is finite in } G_{2i}\}$.
- $G_{2i+2} = G_{2i+1} \setminus \{\langle s, q\rangle \mid \langle s, q\rangle \text{ is } \alpha\text{-free in } G_{2i+1}\}$.

Lemma 4.4 *A run graph $G_r = \langle V, E\rangle$ is accepting iff there is $l \leq |V| + 1$ for which G_l is empty.*

Proof: Assume first that G_r is accepting. We prove that for all $l \geq 1$, the graph G_l has at most $|V| + 1 - l$ vertices. In particular, $G_{|V|+1}$ has at most 0 vertices, so there is $l \leq |V|+1$ for which G_l is empty. The proof proceeds by an induction on l. Clearly, G_1 has at most $|V|$ vertices. For the induction step, we prove that (1) for all $i \geq 1$, if G_{2i} is not empty, then it contains at least one finite vertex, and (2) for all $i \geq 0$, if G_{2i+1} is not empty, then it contains at least one α-free vertex. It follows that the transition from G_l to G_{l+1} involves a removal of at least one vertex, and we are done. We start with Claim (2). Consider the graph G_{2i}. If G_{2i} contains only finite vertices, then G_{2i+1} is empty, and we are done. We prove that if G_{2i} contains a vertex that is not finite, then there must be some α-free vertex in G_{2i+1}. To see this, assume, by way of contradiction, that G_{2i} contains a vertex $\langle s_0, q_0 \rangle$ that is not finite and no vertex in G_{2i+1} is α-free. Consider the graph G_{2i+1}. All the vertices in G_{2i+1} are not finite, and therefore, each of the vertices in G_{2i+1} has at least one successor. Consider the vertex $\langle s_0, q_0 \rangle$ in G_{2i+1}. Since, by the assumption, it is not α-free, there exists an α-vertex $\langle s'_0, q'_0 \rangle$ reachable from $\langle s_0, q_0 \rangle$. Let $\langle s_1, q_1 \rangle$ be a successor of $\langle s'_0, q'_0 \rangle$. By the assumption, $\langle s_1, q_1 \rangle$ is also not α-free. Hence, there exists an α-vertex $\langle s'_1, q'_1 \rangle$ reachable from $\langle s_1, q_1 \rangle$. Let $\langle s_2, q_2 \rangle$ be a successor of $\langle s'_1, q'_1 \rangle$. By the assumption, $\langle s_2, q_2 \rangle$ is also not α-free. Thus, we can continue similarly and construct an infinite sequence of vertices $\langle s_j, q_j \rangle$, $\langle s'_j, q'_j \rangle$ such that for all j, the vertex $\langle s'_j, q'_j \rangle$ is an α-vertex reachable from $\langle s_j, q_j \rangle$, and $\langle s_{j+1}, q_{j+1} \rangle$ is a successor of $\langle s'_j, q'_j \rangle$. Such a sequence, however, corresponds to a path in G_r that visits α infinitely often, contradicting the assumption that G_r is an accepting run graph.

It is left to prove Claim (1). Assume by way of contradiction that there is $i \geq 1$ such that G_{2i} is not empty and yet it contains no finite vertex. Then, $G_{2i+1} = G_{2i}$. Recall that G_{2i} is obtained from G_{2i-1} by removing all the α-free vertices. Therefore, G_{2i} contains no α-free vertex. Hence G_{2i+1} contains no α-free either, contradicting Claim (2).

Assume now that G_r is rejecting. Then, G_r contains an infinite path π with infinitely many α-vertices. We prove that for all $i \geq 0$, all the vertices $\langle s, q \rangle$ in π are in G_{2i}. The proof proceeds by induction on i. The vertices in π are clearly members of G_0. Also, if all the vertices in π are members of G_{2i}, it must be that they are neither finite nor α-free in G_{2i+1}, so they stay in G_{2i+2}. □

Let G_r be an accepting run graph. Given a vertex $\langle s, q \rangle$ in G_r, the *rank* of $\langle s, q \rangle$, denoted $rank(s, q)$, is defined as follows:

$$rank(s,q) = \begin{cases} 2i & \text{If } \langle s,q \rangle \text{ is finite in } G_{2i}. \\ 2i+1 & \text{If } \langle s,q \rangle \text{ is } \alpha\text{-free in } G_{2i+1}. \end{cases}$$

By Lemma 4.4, there is $l \leq |V| + 1$ for which G_l is empty, Therefore, every vertex gets a well-defined rank, smaller than $|V|$.

Lemma 4.5 *Consider a run graph $G_r = \langle V, E \rangle$.*

1. *For every vertex $\langle s, q \rangle$ in G_r and $i \leq |V|$, we have $\langle s, q \rangle \notin G_i$ iff $rank(s, q) < i$.*
2. *For every two vertices $\langle s, q \rangle \neq \langle s', q' \rangle$ in G_r, if $\langle s', q' \rangle$ is reachable from $\langle s, q \rangle$, then $rank(s', q') \leq rank(s, q)$.*

Proof: We start with Claim (1): for every vertex $\langle s, q\rangle$ in G_r and $i \leq |V|$, we have $\langle s, q\rangle \notin G_i$ iff $rank(s, q) < i$.

We first prove that if $rank(s, q) < i$ then $\langle s, q\rangle \notin G_i$. Let $rank(s, q) = j$. By the definition of ranks, $\langle s, q\rangle$ is finite or α-free in G_j. Hence, $\langle s, q\rangle \notin G_{j+1}$. Hence, as $i > j$, also $\langle s, q\rangle \notin G_i$.

For the other direction, we proceed by an induction on i. Since $G_0 = G_r$, the case where $i = 0$ is immediate. For the induction step, consider $i \leq |V|$ and assume the lemma holds for all $j < i$. Consider a vertex $\langle s, q\rangle \notin G_i$. If $\langle s, q\rangle \notin G_{i-1}$, the lemma's requirement follows from the induction hypothesis. Otherwise, $\langle s, q\rangle \in G_{i-1}$ and we distinguish between two cases. If i is even, then $\langle s, q\rangle$ is α-free in G_{i-1}. Accordingly, $rank(s, q) = i - 1$, and we are done. If i is odd, then $\langle s, q\rangle$ is finite in G_{i-1}. Accordingly, $rank(s, q) = i - 1$, and we are done too.

We now prove Claim (2): for every two vertices $\langle s, q\rangle \neq \langle s', q'\rangle$ in G_r, if $\langle s', q'\rangle$ is reachable from $\langle s, q\rangle$, then $rank(s', q') \leq rank(s, q)$.

If $rank(s, q) = i$ is odd, then $\langle s, q\rangle$ is α-free in G_i. Hence, either $\langle x', q'\rangle$ is not in G_i, in which case, by Claim (1), its rank is strictly smaller than i, or $\langle x', q'\rangle$ is in G_i, in which case, being reachable from $\langle s, q\rangle$, it must by α-free in G_i and have rank i.

If $rank(s, q) = i$ is even, then $\langle s, q\rangle$ is finite in G_i. Hence, either $\langle x', q'\rangle$ is not in G_i, in which case, by Claim (1), its rank is strictly smaller than i, or $\langle x', q'\rangle$ is in G_i, in which case, being reachable from $\langle s, q\rangle$, it must be finite in G_i and have rank i. □

Note that no α-vertex gets an odd rank. Hence, by Lemma 4.5, we have the following.

Lemma 4.6 *In every infinite path in an accepting run graph G_r, there exists a vertex $\langle s, q\rangle$ with an odd rank such that all the vertices $\langle s', q'\rangle$ on the path that are reachable from $\langle s, q\rangle$ have $rank(s', q') = rank(s, q)$.*

4.2 From UCT to AWT

For an integer k, let $[k] = \{0, \ldots, k\}$, and let $[k]^{even}$ and $[k]^{odd}$ be the restriction of $[k]$ to its even and odd members, respectively.

Theorem 4.7 *Let $\mathcal{A}$ be a UCT with n states. There is an AWT $\mathcal{A}'$ over the same alphabet such that $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$, and the number of states in $\mathcal{A}'$ is $2^{O(n \log n)}$.*

Proof: Let $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha\rangle$, and let $k = (2n!)n^{2n+1}3^n(n+1)/n!$. The AWT $\mathcal{A}'$ accepts all the regular trees $\langle T, \tau\rangle \in \mathcal{L}(\mathcal{A})$ that are generated by a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L\rangle$ with at most $(2n!)n^{2n+1}3^n(n+1)/n!$ states. Note that the run graph of $\mathcal{A}$ on such $\langle T, \tau\rangle$ is accepting and is of size most k. By Theorem 4.3, we have that $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$. We define $\mathcal{A}' = \langle \Sigma, D, Q', q'_{in}, \delta', \alpha'\rangle$, where

- $Q' = Q \times [k]$. Intuitively, when $\mathcal{A}'$ is in state $\langle q, i\rangle$ as it reads the node $x \in T$, it guesses that the rank of the vertex $\langle \eta(x), q\rangle$ of G_r is i. An exception is the initial state q'_{in} explained below.
- $q'_{in} = \langle q_{in}, k\rangle$. That is, q_{in} is paired with k, which is an upper bound on the rank of $\langle \eta(\varepsilon), q_{in}\rangle$.

- We define δ' by means of a function

$$release : \mathcal{B}^+(D \times Q) \times [k] \to \mathcal{B}^+(D \times Q').$$

Given a formula $\theta \in \mathcal{B}^+(D \times Q)$, and a rank $i \in [k]$, the formula $release(\theta, i)$ is obtained from θ by replacing an atom (c, q) by the disjunction $\bigvee_{i' \leq i}(c, \langle q, i' \rangle)$. For example, $release((1, q) \wedge (2, s), 2) = ((1, \langle q, 2 \rangle) \vee (1, \langle q, 1 \rangle) \vee (1, \langle q, 0 \rangle)) \wedge ((2, \langle s, 2 \rangle) \vee (2, \langle s, 1 \rangle) \vee (2, \langle s, 0 \rangle))$.

Now, $\delta' : Q' \times \Sigma \to \mathcal{B}^+(D \times Q')$ is defined, for a state $\langle q, i \rangle \in Q'$ and $\sigma \in \Sigma$, as follows.

$$\delta'(\langle q, i \rangle, \sigma) = \begin{cases} release(\delta(q, \sigma), i) & \text{If } q \notin \alpha \text{ or } i \text{ is even.} \\ \textbf{false} & \text{If } q \in \alpha \text{ and } i \text{ is odd.} \end{cases}$$

That is, if the current guessed rank is i then, by employing $release$, the run can move in its successors to every rank that is smaller than or equal to i. If, however, $q \in \alpha$ and the current guessed rank is odd, then, by the definition of ranks, the current guessed rank is wrong, and the run is rejecting.

- $\alpha' = Q \times [k]^{odd}$. That is, infinitely many guessed ranks along each path should be odd.

It is easy to see that $\mathcal{A}'$ is weak: each rank $i \in [k]$ induces the set $Q_i = Q \times \{i\}$ in the partition. The acceptance condition α' then requires the run to get stuck in a set associated with an odd rank.

We prove that $\mathcal{A}'$ accepts all the regular trees $\langle T, \tau \rangle \in \mathcal{L}(\mathcal{A})$ that are generated by a transducer $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L \rangle$ with at most $(2n!)n^{2n+1}3^n(n+1)/n!$ states. Note that the run graph of $\mathcal{A}$ on such $\langle T, \tau \rangle$ is accepting and is of size most k. By Theorem 4.3, we then have that $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

We first prove that $L(\mathcal{A}') \subseteq L(\mathcal{A})$. Consider a tree $\langle T, \tau \rangle$ accepted by $\mathcal{A}'$. Let $\langle T_r, r' \rangle$ be the accepting run of $\mathcal{A}'$ on $\langle T, \tau \rangle$. Consider the $T \times Q$-labeled tree $\langle T_r, r \rangle$ where for all $y \in T_r$ with $r'(y) = (x, \langle q, i \rangle)$, we have $r(y) = (x, q)$. Thus, $\langle T_r, r \rangle$ projects out the $[k]$ element of the labels of $\langle T_r, r' \rangle$. It is easy to see that $\langle T_r, r \rangle$ is a run of $\mathcal{A}$ on $\langle T, \tau \rangle$. Indeed, the transitions of $\mathcal{A}'$ only annotate transitions of $\mathcal{A}$ by ranks. We show that $\langle T_r, r \rangle$ is an accepting run. Since $\langle T_r, r' \rangle$ is accepting, then, by the definition of α', each infinite path of $\langle T_r, r' \rangle$ gets trapped in a set $Q \times \{i\}$ for some odd i. By the definition of δ', no accepting run can visit a state $\langle q, i \rangle$ with an odd i and $q \in \alpha$. Hence, the infinite path actually gets trapped in the subset $(Q \setminus \alpha) \times \{i\}$ of $Q \times \{i\}$. Consequently, in $\langle T_r, r \rangle$, all the paths visits states in α only finitely often, and we are done.

It is left to prove that if $\mathcal{T} = \langle D, \Sigma, S, s_{in}, \eta, L \rangle$ is a transducer with at most $(2n!)n^{2n}3^n(n+1)/n!$ states and the run graph of $\mathcal{A}$ on $\mathcal{T}$ is accepting, then $\mathcal{A}'$ accepts the regular tree generated by $\mathcal{T}$. Let $\mathcal{T}$ be as above and let G_r be the accepting run graph of $\mathcal{A}$ on $\mathcal{T}$. Consider the $(T \times Q')$-labeled $\mathbb{N}$-tree $\langle T'_r, r' \rangle$ defined as follows.

- $\varepsilon \in T'_r$ and $r'(\varepsilon) = (\epsilon, \langle q_{in}, k \rangle)$.

- Let $y \in T'_r$ be such that $r'(y) = (x, \langle q, i\rangle)$. By the definition of $\langle T'_r, r'\rangle$ so far, $\langle \eta(x), q\rangle$ is a vertex in G_r . Let $\delta(q, \tau(x)) = \{\langle c_1, q_1\rangle, \ldots, \langle c_m, q_m\rangle\}$. By the definition of G_r, the vertex $\langle \eta(x), q\rangle$ has successors $\langle s_1, q_1\rangle, \ldots, \langle s_m, q_m\rangle$ such that for all $1 \leq i \leq m$, we have that $\eta(\eta(x), c_i) = s_i$. Then, for all $1 \leq i \leq m$, we have $y \cdot i \in T'_r$, and $r'(y \cdot i) = (x \cdot c_i, \langle q_i, rank(\eta(x_i), q_i)\rangle)$.

We claim that $\langle T'_r, r'\rangle$ is an accepting run of $\mathcal{A}'$ on $\langle T, \tau'\rangle$. We first prove that $\langle T'_r, r'\rangle$ is a legal run. Since $q'_{in} = \langle q_{in}, k\rangle$, the root of T'_r is labeled legally. We now consider the other nodes of T'_r. Let $\{(s_1, q_1), \ldots, (s_m, q_m)\}$ be the successors of (ε, q_{in}) in G_r, with $s_i = \eta(s_{in}, c_i)$. As k is the maximal rank that a vertex can get, each successor (s_i, q_i) has $rank(s_i, q_i) \leq k$. Thus, as k is even, the set $\{(c_1, \langle q_1, rank(x_1, q_1)\rangle)$, $\ldots$, $(c_m, \langle q_m, rank(x_m, q_m)\rangle)\}$ satisfies $\delta'(\langle q_{in}, k\rangle, \tau(\varepsilon))$. Hence, the first level of T'_r is labeled legally. For the other levels, consider a node $y \in T'_r$ such that $y \neq \varepsilon$. Let $r'(y) = (x, \langle q, j\rangle)$. By the definition of $\langle T'_r, r'\rangle$, we have that $(\eta(x), q)$ is a vertex of G_r with $rank(\eta(x), q) = j$. Let $\{(s_1, q_1), \ldots, (s_m, q_m)\}$ be the successors of $(\eta(x), q)$ in G_r with $s_i = \eta(s_{in}, c_i)$. By Lemma 4.5, for all $1 \leq i \leq m$, we have $rank(s_i, q_i) \leq j$. Also, by the definition of ranks, it cannot be that $q \in \alpha$ and j is odd. Therefore, the set $\{(c_1, \langle q_1, rank(\eta(x_1), q_1)\rangle), \ldots, (c_m, \langle q_m, rank(\eta(x_m), q_m)\rangle)\}$ satisfies $\delta'(\langle q, j\rangle, \tau(x))$. Hence, the tree $\langle T'_r, r'\rangle$ is a legal run of $\mathcal{A}'$ on $\langle T, \tau'\rangle$. Finally, by Lemma 4.6, each infinite path of $\langle T'_r, r'\rangle$ gets trapped in a set with an odd index, thus $\langle T'_r, r'\rangle$ is accepting. □

4.3 From AWT to NBT

In [MH84], Miyano and Hayashi describe a translation of ABW to NBW. In Theorem 4.8 below (see also [Mos84]), we present (a technical variant of) their translation, adapted to tree automata,

Theorem 4.8 *Let $\mathcal{A}$ be an ABT with n states. There is an NBT $\mathcal{A}'$ with $2^{O(n)}$ states, such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Proof: The automaton $\mathcal{A}'$ guesses a subset construction applied to a run of $\mathcal{A}$. At a given node x of a run of $\mathcal{A}'$, it keeps in its memory the set of states in which the various copies of $\mathcal{A}$ visit node x in the guessed run. In order to make sure that every infinite path visits states in α infinitely often, $\mathcal{A}'$ keeps track of states that "owe" a visit to α. Let $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha\rangle$. Then $\mathcal{A}' = \langle \Sigma, D, 2^Q \times 2^Q, \langle \{q_{in}\}, \emptyset\rangle, \delta', 2^Q \times \{\emptyset\}\rangle$, where δ' is defined as follows. We first need the following notation. For a set $S \subseteq Q$ and a letter $\sigma \in \Sigma$, let $sat(S, \sigma)$ be the set of subsets of $D \times Q$ that satisfy $\bigwedge_{q \in S} \delta(q, \sigma)$. Also, for two sets $O \subseteq S \subseteq Q$ and a letter $\sigma \in \Sigma$, let $pair\_sat(S, O, \sigma)$ be such that $\langle S', O'\rangle \in pair\_sat(S, O, \sigma)$ iff $S' \in sat(S, \sigma)$, $O' \subseteq S'$, and $O' \in sat(O, \sigma)$. Finally, for a direction $c \in D$, we have $S'_c = \{s : (c, s) \in S'\}$ and $O_c = \{o : (c, o) \in O'\}$.

Now, δ is defined, for all $\langle S, O\rangle \in 2^Q \times 2^Q$ and $\sigma \in \Sigma$, as follows.

- If $O \neq \emptyset$, then

$$\delta'(\langle S, O\rangle, \sigma) = \bigvee_{\substack{\langle S', O'\rangle \in \\ pair\_sat(S, O, \sigma)}} \bigwedge_{c \in D} (c, \langle S'_c, O'_c \setminus \alpha\rangle).$$

Thus, $\mathcal{A}'$ sends to direction c the set S'_c of states that are sent to direction c (in different copies) in the guessed run. Each such S'_c is paired with a subset O'_c of S'_c of the states that still owe a visit to α.

- If $O = \emptyset$, then

$$\delta'(\langle S, O\rangle, \sigma) = \bigvee_{S' \in sat(S,\sigma)} \bigwedge_{c \in D} (c, \langle S'_c, S'_c \setminus \alpha\rangle).$$

Thus, when no state owes a visit to α, the requirement to visit α is reinforced on all the states in S'_c.

□

4.4 Complexity

Combining Theorems 4.7 and 4.8, one can reduce the nonemptiness problem for UCT to the nonemptiness problem for NBT. Consider a UCT $\mathcal{A}$ with n states. If we translate $\mathcal{A}$ to an NBT by going through the AWT we have obtained in Theorem 4.7, we end up with an NBT with $2^{2^{O(n \log n)}}$ states, as the AWT has $2^{O(n \log n)}$ states. In order to complete the proof of Theorem 3.3, we now exploit the special structure of the AWT in order to get an NBT with only $2^{O(n^2 \log n)}$ states.

Theorem 4.9 *Let $\mathcal{A}$ be a UCT with n states. There is an NBT $\mathcal{A}'$ over the same alphabet such that $\mathcal{L}(\mathcal{A}') \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$, and the number of states in $\mathcal{A}'$ is $2^{O(n^2 \log n)}$.*

Proof: Let $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha\rangle$ with $|Q| = n$. Let $k = (2n!)n^{2n+1}3^n(n+1)/n!$ Consider a state $\langle S, O\rangle$ of the NBT constructed from $\mathcal{A}$ as described above. Each of the sets S and O is a subset of $Q \times [k]$. We say that a set $P \subseteq Q \times [k]$ is *consistent* iff for every two states $\langle q, i\rangle$ and $\langle q', i'\rangle$ in P, if $q = q'$ then $i = i'$. We claim the following: (1) Restricting the states of the NBT to pairs $\langle S, O\rangle$ for which S is a consistent subset of $Q \times [k]$ is allowable; that is, the resulting NBT is equivalent. (2) There are $2^{O(n^2 \log n)}$ consistent subsets of $Q \times [k]$.

In order to prove Claim (1), recall that the AWT visiting a state $\langle q, i\rangle$ when reading a node $x \in T$ corresponds to a guess that the rank of the vertex $\langle \eta(x), q\rangle$ of an accepting run graph G_r is i. Since every vertex in G_r has a unique rank, the copies of AWT that are generated in an accepting run that corresponds to G_r are consistent, in the sense that the different copies that read the same node x agree on the rank that $\langle \eta(x), q\rangle$ has in G_r. When the NBT visits a state $\langle S, O\rangle$, all the states in S correspond to copies of the AWT that read the same node. Hence, a state $\langle S, O\rangle$ for which S is inconsistent corresponds to a node in the run of the AWT whose copies are inconsistent. Hence, the NBT can ignore states $\langle S, O\rangle$ with inconsistent S.

In order to prove Claim (2), observe that we can characterize a consistent set by the projection of its pairs on Q, augmented by an assignment $f : Q \to [k]$. Since there are 2^n such projections and $k^n = 2^{O(n^2 \log n)}$ such assignments, we are done.

By the two claims, as O is always a subset of S, we can restrict the state space of the NBT to $2^{O(n^2 \log n)}$ states. The construction that follows is described in the proof of Theorem 3.3. □

5 More Applications

In Section 3, we show how UCT can help in solving the emptiness problem for APT. One immediate application is the decidability problem for μ-calculus, which is easily reduced to APT emptiness [EJ91, KVW00]. Another immediate application is the *language-containment* problem for NPT: given two NPT $\mathcal{A}_1$ and $\mathcal{A}_2$, we can check whether $\mathcal{L}(\mathcal{A}_1)$ is contained in $\mathcal{L}(\mathcal{A}_2)$ by checking the nonemptiness of the intersection of $\mathcal{A}_1$ with the complement of $\mathcal{A}_2$. Since it is easy to complement $\mathcal{A}_2$ by dualizing it [MS87], it is easy to define this intersection as an APT.

In this section we describe more, less immediate, applications of our approach. In particular, we show that UCT are often useful for tasks traditionally assigned to APT. Thus, in many cases it is possible to skip the construction of an APT and go directly to a UCT. We demonstrate this below with the realizability and synthesis problems for LTL, and the problem of translating LTL formulas into deterministic parity automata of a minimal index (in particular, translating LTL formulas into DBW). We note that these problems involve both a decision problem (namely, is the formula realizable? can the formula be translated into a DPW with a given index?) as well as a construction problem (namely, construct a realizing strategy; construct an equivalent DPW). As discussed in Section 6.2, while our Safraless approach simplifies the algorithms and improves the complexity of the decidability problems, the fact it uses a simplified class of automata (that is, co-Büchi rather than parity) causes the constructions to have more states than these constructed by the traditional algorithm.

5.1 LTL Realizability and Synthesis

Given an LTL formula ψ over the sets I and O of input and output signals, the *realizability problem* for ψ is to decide whether there is a *strategy* $f : (2^I)^* \to 2^O$, generated by a transducer[5] such that all the computations of the system generated by f satisfy ψ [PR89]. Formally, a computation $\rho \in (2^{I \cup O})^\omega$ is generated by f if $\rho = (i_0 \cup o_0), (i_1 \cup o_1), (i_2 \cup o_2), \ldots$ and for all $j \geq 1$, we have $o_j = f(i_0 \cdot i_1 \cdots i_{j-1})$.

The traditional algorithm for solving the realizability problem translates the LTL formula into an NBW, applies Safra's construction in order to get a DPW A_ψ for it, expands $\mathcal{A}_\psi$ to a DPT $\mathcal{A}_{\forall\psi}$ that accepts all the trees all of whose branches satisfy ψ, and then checks the nonemptiness of $\mathcal{A}_{\forall\psi}$ with respect to *I-exhaustive* $2^{I \cup O}$-labeled 2^I-trees, namely $2^{I \cup O}$-labeled 2^I-trees that contain, for each word $w \in (2^I)^\omega$, at least one path whose projection on 2^I is w [PR89]. Thus, the algorithm applies Safra's determinization construction, and has to solve the nonemptiness problem for DPT. For ψ of length n, the DPW $\mathcal{A}_\psi$ has $2^{2^{O(n \log n)}}$ states and index $2^{O(n)}$. This is also the size of the DPT $\mathcal{A}_{\forall\psi}$, making the overall complexity doubly-exponential, which matches the lower bound in [Ros92]. We now show how UCW can be used instead of DPW. Intuitively, universal automata have the desired property, enjoyed also by deterministic automata but not by nondeterministic automata, of having the ability to run over all branches of an input tree. In addition, since complementation of LTL is trivial, the

[5] It is known that if some transducer that generates f exists, then there is also a finite-state transducer.

known translations of LTL into NBW can be used in order to translate LTL into UCW[6]. Formally, we have the following.

Theorem 5.1 *The realizability problem for an LTL formula can be reduced to the nonemptiness problem for a UCT with exponentially many states.*

Proof: A strategy $f : (2^I)^* \to 2^O$ can be viewed as a 2^O-labeled 2^I-tree. We define a UCT $\mathcal{S}_\psi$ such that $\mathcal{S}_\psi$ accepts a 2^O-labeled 2^I-tree $\langle T, \tau \rangle$ iff τ is a good strategy for ψ.

Let $\mathcal{A}_{\neg\psi} = \langle 2^{I \cup O}, Q, q_{in}, \delta, \alpha \rangle$ be an NBW for $\neg\psi$ [VW94]. Thus, $\mathcal{A}_{\neg\psi}$ accepts exactly all the words in $(2^{I \cup O})^\omega$ that do not satisfy ψ. Then, $\mathcal{S}_\psi = \langle 2^O, 2^I, Q, q_{in}, \delta', \alpha \rangle$, where for every $q \in Q$ and $o \in 2^O$, we have $\delta'(q, o) = \bigwedge_{i \in 2^I} \bigwedge_{q' \in \delta(q, i \cup o)} (i, q')$. Thus, from state q, reading the output assignment $o \in 2^O$, the automaton $\mathcal{S}_\psi$ branches to each direction $i \in 2^I$, with all the states q' to which δ branches when it reads $i \cup o$ in state q. It is not hard to see that $\mathcal{S}_\psi$ accepts a 2^O-labeled 2^I-tree $\langle T, \tau \rangle$ iff for all the paths $\{\varepsilon, i_0, i_0 \cdot i_1, i_0 \cdot i_1 \cdot i_2, \ldots\}$ of T, the infinite word $(i_0 \cup \tau(\varepsilon)), (i_1 \cup \tau(i_0)), (i_2 \cup \tau(i_0 \cdot i_1)), \ldots$ is not accepted by $\mathcal{A}_{\neg\psi}$; thus all the computations generated by τ satisfy ψ. Since the size of $\mathcal{A}_{\neg\psi}$ is exponential in the length of ψ, so is $\mathcal{S}_\psi$, and we are done. □

For an LTL formula of length n, the size of the automaton S_ψ is $2^{O(n)}$, making the overall complexity doubly-exponential, matching the complexity of the traditional algorithm , as well as the lower bound [Ros92].

The *synthesis problem* for an LTL formula ψ is to find a a transducer that generates a strategy realizing ψ. Known algorithms for the nonemptiness problem can be easily extended to return a transducer [Rab70]. The algorithm we present here also enjoys this property, thus it can be used to solved not only the realizability problem but also the synthesis problem.

In the *supervisory-control* problem, one has to disable some of the controllable transitions of a given system in order for it to satisfy a given specification [RW89]. The problem is similar to the synthesis problem: in both problems the goal is to synthesize a correct system, where in supervisory control, some information about the system is already given. Formally, the system, also called a *plant*, is a nondeterminstic transducer $\mathcal{T} = \langle 2^I, 2^O, S, S_{in}, \eta, L \rangle$, with $S_{in} \subseteq S$ and $\eta : S \times 2^I \to 2^S$. Note that $\mathcal{T}$ may have several possible computations on a given sequence of inputs. A control strategy for $\mathcal{T}$ is a function $f : (2^O)^* \to 2^I$ that maps the sequence of outputs the system has generated so far to an input enabled by the environment. We say that a computation over I and O of is *consistent* with f and $\mathcal{T}$ if the inputs follow f and the outputs follow $\mathcal{T}$. Formally. a computation $\rho \in (2^{I \cup O})^\omega$ is consistent with f and $\mathcal{T}$ if $\rho = (i_0 \cup o_0), (i_1 \cup o_1), (i_2 \cup o_2), \ldots$ is such that there is a sequence $s_0, s_1, s_2, \ldots$ of states of $\mathcal{T}$ such that $s_0 \in S_{in}$, $i_0 = f(\epsilon)$, and $o_0 = L(s_0)$, and for all $j \geq 1$, we have $s_j \in \eta(s_{j-1}, i_{j-1})$, $i_j = f(o_0 \cdot o_1 \cdots o_{j-1})$, and $o_j = L(s_j)$.

Our Safraless solution to the synthesis problem, which is described in the proof of Theorem 5.1, can be adjusted to solve also the supervisory-control problem. Essentially, rather than defining $\mathcal{S}_\psi$ to branch to all the directions in 2^I and read letters

[6] For an application of these properties in the area of infinite games, see [ATM03].

in 2^O, we define it to run over an unwinding of the system, and has in its alphabet information about whether particular inputs are disabled or enabled. Formally, $\mathcal{S}_\psi = \langle 2^I, 2^O, S \times Q, S_{in} \times \{q_{in}\}, \delta', \alpha \rangle$, where for every $s \in S$, $q \in Q$, and $i \in 2^I$, we have

$$\delta'(\langle s, q \rangle, i) = \bigwedge_{o \in 2^O} \bigwedge_{s' \in \eta(s,i): L(s') = o} \bigwedge_{q' \in \delta(q, i \cup L(s))} (o, \langle s', q' \rangle).$$

Thus, from state $\langle s, q \rangle$, reading the input assignment $i \in 2^I$, the automaton $\mathcal{S}_\psi$ branches to each direction $o \in 2^O$ with all the states $\langle s', q' \rangle$ such that $\mathcal{T}$ may branch to s' from s when it reads i, the output in s' is o, and δ branches to q' when it reads $i \cup L(s)$ in state q.

The synthesis problem often arise in a setting in which the system has *incomplete information* about the environment. As described in [KV00], the solution to the synthesis problem with incomplete information for branching temporal logic can be reduced to the nonemptiness problem of APT, thus our Safraless procedure for the latter is of use also in this context.

5.2 Translation of LTL formulas to DBW

DBW form a strict subset of NBW [Lan69]. For an LTL formula ψ, we say that ψ is in DBW if the language of words satisfying ψ can be recognized by a DBW. Not all LTL formulas are in DBW. For example, while there is no DBW for the language of all words over the alphabet $\{a, b\}$ that contains only finitely many a's, it is easy to specify this language with the LTL formula $FG\neg a$ ("eventually always not a"). It turned out that an LTL formula ψ is in DBW iff ψ has an equivalent alternation-free μ-calculus formula [KV98a]. Current methods for deciding whether an LTL formula ψ is in DBW start with a construction of an NBW for ψ [VW94], then determinize it to a DPW using Safra's determinization construction, and then check whether the DPW has an equivalent DBW [KPB94]. In this section we describe how UCW can be used instead of DPW.

Theorem 5.2 *The problem of deciding whether an LTL formula is in DBW can be reduced to the nonemptiness problem of a UCT with exponentially many states.*

Proof: By [Lan69], an ω-regular language $\mathcal{L} \subseteq \Sigma^\omega$ is in DBW iff there is some regular language $R \subseteq \Sigma^*$ such that $\mathcal{L} = limR$; i.e., $w \in \mathcal{L}$ iff w has infinitely many prefixes in R. A regular language R can be represented by a $\{0,1\}$-labeled Σ-tree $\langle \Sigma^*, f_R \rangle$ where for all $x \in \Sigma^*$, we have $f_R(w) = 1$ iff $x \in R$. For an LTL formula ψ over AP, we say that a $\{0,1\}$-labeled (2^{AP})-tree $\langle (2^{AP})^*, f \rangle$ is a *DBW witness* for ψ if for all paths $w \subseteq (2^{AP})^\omega$, we have that w satisfies ψ iff there are infinitely many $x \in w$ with $f(x) = 1$. By [Lan69], ψ can be recognized by a DBW iff it has a DBW witness. We construct a UCT over $\{0,1\}$-labeled (2^{AP})-trees that accepts exactly all the DBW witnesses of ψ.

Let $\mathcal{A}^+ = \langle 2^{AP}, Q^+, q^+_{in}, \delta^+, \alpha^+ \rangle$ and $\mathcal{A}^- = \langle 2^{AP}, Q^-, q^-_{in}, \delta^-, \alpha^- \rangle$ be NBWs

for ψ and $\neg\psi$, respectively [VW94][7]. Thus, $\mathcal{A}^+$ accepts exactly all the words in $(2^{AP})^\omega$ that satisfy ψ, and $\mathcal{A}^-$ complements it. We define an NBW $\mathcal{S}_\psi$ over the alphabet $2^{AP} \times \{0,1\}$. For a word $w \in 2^{AP} \times \{0,1\}$, we use $w[1]$ and $w[2]$ in order to refer to the projection of w on 2^{AP} and $\{0,1\}$, respectively. The NBW $\mathcal{S}_\psi$ accepts a word w if one of the following holds.

- $w[1]$ satisfies ψ and $w[2]$ has only finitely many 1's, or
- $w[1]$ does not satisfy ψ and $w[2]$ has infinitely many 1's.

It is easy to define $\mathcal{S}_\psi$ as a union of two NBWs, the first obtained by intersecting $\mathcal{A}^+$ with an NBW for "finitely many 1's", and the second obtained by intersecting $\mathcal{A}^-$ with an NBW for "infinitely many 1's". By dualizing $\mathcal{S}_\psi$, we get a UCW $\tilde{\mathcal{S}}_\psi$ that accepts exactly all the words $w \in 2^{AP} \times \{0,1\}$ for which $w[1]$ satisfies ψ iff $w[2]$ has infinitely many 1's. We can run $\tilde{\mathcal{S}}_\psi$ on $\{0,1\}$-labeled 2^{AP}-trees so that it accepts exactly all the DBW witnesses for ψ. Formally, if $\tilde{\mathcal{S}}_\psi = \langle 2^{AP} \times \{0,1\}, Q, q_{in}, \delta, \alpha\rangle$, then the UCT is $\mathcal{W}_\psi = \langle \{0,1\}, 2^{AP}, Q, q_{in}, \delta, \alpha\rangle$ where for all $q \in Q$ and $\tau \in \{0,1\}$, we have

$$\delta(q,\tau) = \bigwedge_{\sigma \in 2^{AP}} \bigwedge_{s \in \delta(q,\langle\sigma,\tau\rangle)} (\sigma, s).$$

□

Now, suppose that ψ is in DBW and $\mathcal{W}_\psi$ is not empty. Then, by [Rab70], the nonemptiness test of its equivalent NBT returns a transducer $\mathcal{T}$ with edges labeled by 2^{AP} and states labeled by $\{0,1\}$. Note that $\mathcal{T}$ is deterministic. By defining the states of $\mathcal{T}$ with label 1 to be accepting, we get a DBW equivalent to ψ. Thus, when our test returns a positive answer, it can also translate the LTL formula into an equivalent DBW.

5.3 Translating LTL formulas to DPW

While DPW are not less expressive than NPW, fixing the index of a DPW does result in a strict subset of NPW. In fact, if we denote the set of DPW with index h by DPW$[h]$, then DPW$[h]$ form a strict subset of DPW$[h+1]$ [Kam85]. Since it is possible to specify the languages described in [Kam85] by LTL formulas, the above strict hierarchy remains valid when we restrict attention to languages that are generated by LTL formulas. Recall that the Büchi acceptance condition is a special case of the parity acceptance condition. Indeed, a Büchi condition α is equivalent to a parity condition $\{\emptyset, \alpha, Q\}$ of index 3. In this section we extend the reasoning described in Section 5.2 to parity automata with a fixed index. For an LTL formula ψ and an integer $h \geq 2$, we say that ψ is of index h if the language of words satisfying ψ can be recognized by a DPW$[h]$. Current methods for deciding whether an LTL formula ψ is of index h start with a construction of an NBW for ψ [VW94], then determinize it to a DPW using

[7] Using the construction in [VW94], the NBWs $\mathcal{A}^+$ and $\mathcal{A}^-$ differ only in their sets of initial states. Our construction, however, does not make a use of this fact, thus $\mathcal{A}^+$ and $\mathcal{A}^-$ can be optimized, and we assume, for simplicity, that each of them has a single initial state.

Safra's determinization construction, and then check whether the DPW has an equivalent DPW$[h]$ [KPB95, KPBV95]. In this section we describe how UCW can be used instead of DPW. In case the LTL formula has an equivalent DPW$[h]$, our procedure returns it without first constructing a DPW of a larger index.

Theorem 5.3 *The problem of deciding the minimal index of an LTL formula can be reduced to the nonemptiness problem of a UCT with exponentially many states.*

Proof: For an LTL formula ψ over AP, we say that a regular $\{1, 2, \ldots, h\}$-labeled (2^{AP})-tree $\langle (2^{AP})^*, f\rangle$ is a *DPW*$[h]$ *witness* for ψ if for all paths $w \subseteq (2^{AP})^\omega$, we have that w satisfies ψ iff the minimal letter c such that infinitely many $x \in w$ have $f(x) = c$ is even. It is easy to see that a DPW$[h]$ for ψ induces a DPW$[h]$ witness. Indeed, if $\mathcal{A}$ is a DPW$[h]$ with $\alpha = \{F_1, F_2, \ldots, F_h\}$ for ψ and the single run of $\mathcal{A}$ on a word $w \in (2^{AP})^*$ leads to a state in F_i, we define $f(w)$ to be i. Also, since we restrict attention to regular trees, a DPW$[h]$ witness induces a DPW$[h]$ for ψ. We construct a UCT over $\{1, \ldots, h\}$-labeled (2^{AP})-trees that accepts exactly all the DPW$[h]$ witnesses of ψ.

Let $\mathcal{A}^+ = \langle 2^{AP}, Q^+, q^+_{in}, \delta^+, \alpha^+\rangle$ and $\mathcal{A}^- = \langle 2^{AP}, Q^-, q^-_{in}, \delta^-, \alpha^-\rangle$ be NBWs for ψ and $\neg\psi$, respectively [VW94]. Thus, $\mathcal{A}^+$ accepts exactly all the words in $(2^{AP})^\omega$ that satisfy ψ, and $\mathcal{A}^-$ complements it. We define an NBW $\mathcal{S}_\psi$ over the alphabet $2^{AP} \times \{1, \ldots, h\}$. For a word $w \in 2^{AP} \times \{1, \ldots, h\}$, we use $w[1]$ and $w[2]$ in order to refer to the projection of w on 2^{AP} and $\{1, \ldots, h\}$, respectively. The NBW $\mathcal{S}_\psi$ accepts a word w if one of the following holds.

- $w[1]$ satisfies ψ and the minimum letter that appears infinitely often in $w[2]$ is odd, or
- $w[1]$ does not satisfy ψ and the minimum letter that appears infinitely often in $w[2]$ is even.

It is easy to define $\mathcal{S}_\psi$ as a union of two NBWs, the first obtained by intersecting $\mathcal{A}^+$ with an NBW for "the minimum letter that appears infinitely often is odd", and the second obtained by intersecting $\mathcal{A}^-$ with an NBW for "the minimum letter that appears infinitely often is even". By dualizing $\mathcal{S}_\psi$, we get a UCW $\tilde{\mathcal{S}}_\psi$ that accepts exactly all the words $w \in 2^{AP} \times \{1, \ldots, h\}$ for which $w[1]$ satisfies ψ iff the minimum letter that appears infinitely often in $w[2]$ is even. We can run $\tilde{\mathcal{S}}_\psi$ on $\{1, \ldots, h\}$-labeled 2^{AP}-trees so that it accepts exactly all the DPW$[h]$ witnesses for ψ. Formally, if $\tilde{\mathcal{S}}_\psi = \langle 2^{AP} \times \{1, \ldots, h\}, Q, q_{in}, \delta, \alpha\rangle$, then the UCT is $\mathcal{W}_\psi = \langle \{1, \ldots, h\}, 2^{AP}, Q, q_{in}, \delta, \alpha\rangle$ where for all $q \in Q$ and $\tau \in \{0, 1\}$, we have

$$\delta(q, \tau) = \bigwedge_{\sigma \in 2^{AP}} \bigwedge_{s \in \delta(q, \langle \sigma, \tau\rangle)} (\sigma, s).$$

□

As in the case of DBW, when the NBT equivalent to $\mathcal{S}_\psi$ is not empty, its emptiness test returns a regular transducer with edges labeled (2^{AP}) and states labeled $\{1, 2, \ldots, h\}$, and hence a DPW$[h]$ for ψ. For ψ of length n, Safra's determinization construction results in a DPW $\mathcal{A}_\psi$ of index $2^{O(n)}$. Thus, all LTL formulas can

be translated to a DPW with an exponential index. This suggests a Safraless determinization construction for LTL formulas. Moreover, it suggests a Safraless optimized determinization construction, where one starts with a small guess for the index of ψ and increase the guess when necessary.

We note that in all the three applications described in this section, we used the fact that we are able, given a specification ψ, to construct an NBW for $\neg\psi$ of size exponential in $|\psi|$. When ψ is given in terms of an LTL formula, this exponential blow-up is similar to the exponential blow-up in the translation of ψ to an NBW, making our bounds similar, and even better, than known bounds to the problems. While it is possible to apply our constructions to a specification ψ that is given in terms of an NBW, the exponential blow-up in that case is not acceptable. Indeed, since going from an LTL formula to a DPW involves a doubly-exponential blow-up whereas going from NBW to DPW involves only an exponential blow-up, the traditional solutions that use Safra's determinization construction perform exponentially better for ψ that is given in terms of an NBW. It is an open problem whether a Safraless solution for the applications described in this section exists also for specifications that are given in terms of NBW. A positive answer applies to the realizability problem. Indeed, given an NBW $\mathcal{A}$, the system has a strategy $f : (2^I)* \to 2^O$ that realizes $\mathcal{A}$ iff the environment does not have a strategy $g : (2^O)^* \to 2^I$ that realizes the complement of $\mathcal{A}$. Now, the complement of $\mathcal{A}$ is a UCW, for which the construction we describe does apply. Thus, realizability of NBW specifications can be reduced to realizabilty of UCW specifications.

Another point that is joint to the three applications has to do with the size of the generated transducer, in case the decision procedure returns a positive answer. We will get back to this point in Section 6.

6 In Practice

As discussed in Section 1, the intricacy of current constructions, which use Safra's determinization, is reflected in the fact there is no implementation for them. The lack of a simple implementation is not due to a lack of need: implementations of realizability algorithms exist, but they have to either restrict the specification to one that generates "easy to determinize" automata [ST03, WMBSV05] or give up completeness [HRS05]. As we argue in this section, the simplicity of our construction not only makes it amenable to implementation, but also enables important practical advantages over the existing algorithms.

6.1 A symbolic implementation

Safra's determinization construction involves complicated data structures: each state in the deterministic automaton is associated with a labeled ordered tree. Consequently, even though recent work describes a symbolic algorithm for the nonemptiness problem for NPT [BKV04], there is no symbolic implementation of decision procedures that are based on Safra's determinization and NPT. Our construction, on the other hand, can be implemented symbolically. Indeed, the state space of the NBT constructed in Theorem 3.3 consists of sets of states and a ranking function, it can be encoded by Boolean

variables, and the NBT's transitions can be encoded by relations on these variables and a primed version of them. The fixpoint solution for the nonemptiness problem of NBT (c.f., [VW86]) then yields a symbolic solution to the original UCT nonemptiness problem. Moreover, when applied for the solution of the realizability problem, the BDDs that are generated by the symbolic decision procedure can be used to generate a symbolic witness strategy. The Boolean nature of BDDs then makes it very easy to go from this BDD to a sequential circuit for the strategy. It is known that a BDD can be viewed as an expression (in DAG form) that uses the "if then else" as a single ternary operator. Thus, a BDD can be viewed as a circuit built from if-then-else gates. More advantages of the symbolic approach are described in [HRS05]. As mentioned above, [HRS05] also suggests a symbolic solution for the LTL synthesis problem. However, the need to circumvent Safra's determinization causes the algorithm in [HRS05] to be complete only for a subset of LTL. Our approach circumvents Safra's determinization without giving up completeness.

6.2 An incremental approach

Recall that our construction is based on the fact we can bound the maximal rank that a vertex of G_r can get by k – the bound on the size of the run graphs of $\mathcal{A}$ we consider. Often, the sequence $G_0, G_1, G_2, \ldots$ of graphs converges to the empty graph very quickly, making the bound on the maximal rank much smaller (see [GKSV03] for an analysis and experimental results for the case of UCW). Accordingly, we suggest to regard k as a parameter in the construction, start with a small parameter, and increase it if necessary. Let us describe the incremental algorithm that follows in more detail.

Consider the combined construction described in Theorem 3.3. Starting with a UCT $\mathcal{A}$ with state space Q, we constructed an NBT $\mathcal{A}'$ with state space $2^Q \times 2^Q \times \mathcal{R}$, where $\mathcal{R}$ is the set of functions $f : Q \to [k]$ in which $f(q)$ is even for all $q \in \alpha$. For $l \leq k$, let $\mathcal{R}[l]$ be the restriction of $\mathcal{R}$ to functions with range $[l]$, and let $\mathcal{A}'[l]$ be the NBT $\mathcal{A}'$ with k being replaced by l. Recall that the NBT $\mathcal{A}'[l]$ is empty iff all the run graphs of $\mathcal{A}$ of size at most l are not accepting. Thus, coming to check the emptiness of $\mathcal{A}$, a possible heuristic would be to proceed as follows: start with a small l and check the nonemptiness of $\mathcal{A}'[l]$. If $\mathcal{A}'[l]$ is not empty, then $\mathcal{A}$ is not empty, and we can terminate with a "nonempty" output. Otherwise, increase l, and repeat the procedure. When $l = k$ and $\mathcal{A}'[l]$ is still empty, we can terminate with an "empty" output.

It is important to note that it is possible to take advantage of the work done during the emptiness test of $\mathcal{A}'[l_1]$, when testing emptiness of $\mathcal{A}'[l_2]$, for $l_2 > l_1$. To see this, note that the state space of $\mathcal{A}'[l_2]$ consists of the union of $2^Q \times 2^Q \times \mathcal{R}[l_1]$ (the state space of $\mathcal{A}'[l_1]$) with $2^Q \times 2^Q \times (\mathcal{R}[l_2] \setminus \mathcal{R}[l_1])$ (states whose $f \in \mathcal{R}[l_2]$ has a state that is mapped to a rank greater than l_1). Also, since ranks can only decrease, once the NBT $\mathcal{A}'[l_2]$ reaches a state of $\mathcal{A}'[l_1]$, it stays in such states forever. So, if we have already checked the nonemptiness of $\mathcal{A}'[l_1]$ and have recorded the classification of its states to empty and nonempty, the additional work needed in the nonemptiness test of $\mathcal{A}'[l_2]$ concerns only states in $2^Q \times 2^Q \times (\mathcal{R}[l_2] \setminus \mathcal{R}[l_1])$.

The incremental approach circumvents the fact that the k-fold blow-up that is introduced in the translation of a UCT to an AWT occurs for all UCT. With the incremental algorithm, the k-fold blow occurs only in the worst case. As shown in [GKSV03],

experimental results show that in the case of word automata the construction ends up with a small k. A point in favor of the Safrafull approach has to do with the bound on the size of a "nonemptiness witness" in case the APT (or the UCT) is not empty. Known algorithms for the nonemptiness problem of nondeterministic tree automata can be easily extended to return a witness to the automaton's nonemptiness. Such a witness is a transducer that generates a tree accepted by the automaton whose nonemptiness is checked (in the case of realizability, the witness is a synthesized strategy; in the case of LTL determinization, the witness is a DBW or a DPW equivalent to the LTL formula). The size of the witness is linear in the state space of the automaton. Both the Safrafull and the Safraless approaches reduce the original problem to the nonemptiness problem of a nondeterministic automaton. The Safraless approach avoids the parity condition and uses instead the Büchi condition. This makes the nonemptiness test easier. Indeed, the nonemptiness algorithm for NPT is exponential in the index of the NPT, while the nonemptiness algorithm for NBT is quadratic. On the other hand, if we restrict attention to the bound on the size of the state space of the automaton (and thus, the size of a witness), then the parity condition has an advantage: the Safraless approach translates a UCT with n states to an NBT with $2^{O(n^2 \log n)}$ states, whereas the Safrafull approach results in an NPT with $2^{O(n \log n)}$ states. Such a Safraless bound on the size of a small witness is still an open problem. With the incremental algorithm, however, we expect the NBT whose emptiness we check to be much smaller than an NPT constructed with no optimizations.

7 Discussion

In [KV01], we used alternating co-Büchi word automata in order to avoid Safra's construction in complementation of Büchi word automata. As here, the approach in [KV01] involves an analysis of ranks. Alternating word automata are closely related to nondeterministic tree automata and the analysis in [KV01] have proven to be useful also for solving the nonemptiness problem for nondeterministic parity tree automata [KV98b]. By now, the simple construction in [KV01] has become the standard complementation construction [Tho98], has been implemented [GKSV03, Mer00], has led to tighter and new Safraless complementation constructions for richer types of automata on infinite words [FKV04, KV04], and has led to further implementations of alternating automata [Fin01].

Since the bounded-width property trivially holds for runs of word automata, the analysis in [KV98b, KV01] is much simpler than the one required for alternating tree automata, and indeed the problem of a Safraless decision procedure for them was left open. In this work we solved this problem and showed how universal co-Büchi automata can be used in order to circumvent Safra's determinization and the parity acceptance condition. Below we discuss a related theoretical point that is still open.

Our construction avoids the complicated determinization construction of Safra, but its correctness proof makes use of the bounded-size run graph property, which in turn makes use of Safra's determinization. It is an open problem whether we can have a Safraless proof, and whether such a proof can improve the construction further. Consider an infinite run tree $\langle T_r, r \rangle$ of a UCT. We say that two nodes y_1 and

y_2 of T_r are *similar* if $r(y_1) = r(y_2)$. Thus, similar nodes correspond to different copies of the UCT, possibly with a different past, but with the same present: if $r(y_1) = r(y_2) = \langle x, q\rangle$, then both copies have to accept the subtree with root x from the state q. Runs of UCT are memoryless in the sense that two copies of the UCT that read similar nodes have the same future. Thus, by merging similar nodes on the run tree, one gets a run DAG G_r of the UCT, which is accepting iff $\langle T_r, r\rangle$ is accepting. Recall that the bounded-size run graph property enables us to bound the maximal rank that a vertex can get. The run DAG G_r is infinite, but we can also show (see [KV01] for the case of words) that bounding its width (the number of different vertices in each level) by an integer k leads to a ranking function in which the maximal rank is $2k$. In order to get a bounded-width DAG property, we need not bound the width of all run DAGs—we only need to show that if the UCT is not empty then it has a accepting run DAG of width at most k. We conjecture that a UCT is not empty iff it accepts a tree in which nodes that are visited by the same set of states (recall that each node of the input tree may be visited by several copies of the UCT) are roots of identical subtrees. Our conjecture leads to an $n2^n$ bound on the width, for a UCT with n states. Proving the conjuncture will not only make the proof Safraless, but will also reduce the maximal rank that a vertex can get, and thus improves the construction further.

Acknowledgement We thank Nir Piterman for helpful discussions and comments on an early draft of this paper.

References

[ATM03] R. Alur, S. La Torre, and P. Madhusudan. Modular strategies for infinite games on recursive game graphs. In *Computer-Aided Verification, Proc. 15th International Conference*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2003.

[ATW05] C. Schulte Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of Büchi automata. In *Proc. 10th International Conference on the Implementation and Application of Automata*, 2005.

[BKV04] D. Bustan, O. Kupferman, and M.Y. Vardi. A measured collapse of the modal μ-calculus alternation hierarchy. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Computer Science*, pages 522–533. Springer-Verlag, 2004.

[Büc60] J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. International Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and System Sciences*, 8:117–141, 1974.

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 328–337, White Plains, October 1988.

[EJ91] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 368–377, San Juan, October 1991.

[EKM98] J. Elgaard, N. Klarlund, and A. Möller. Mona 1.x: new techniques for WS1S and WS2S. In *Computer Aided Verification, Proc. 10th International Conference*, volume 1427 of *Lecture Notes in Computer Science*, pages 516–520. Springer-Verlag, Berlin, 1998.

[Elg61] C. Elgot. Decision problems of finite-automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.

[Eme85] E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 79–87. Springer-Verlag, 1985.

[Fin01] B. Finkbeiner. Symbolic refinement checking with nondeterministic BDDs. In *Tools and algorithms for the construction and analysis of systems*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[FKV04] E. Friedgut, O. Kupferman, and M.Y. Vardi. Büchi complementation made tighter. In *2nd International Symposium on Automated Technology for Verification and Analysis*, volume 3299 of *Lecture Notes in Computer Science*, pages 64–78. Springer-Verlag, 2004.

[Fri03] C. Fritz. Constructing Büchi automata from linear temporal logic using simulation relations for alternating bchi automata. In *Proc. 8th Intl. Conference on Implementation and Application of Automata*, number 2759 in Lecture Notes in Computer Science, pages 35–48. Springer-Verlag, 2003.

[FW02] C. Fritz and T. Wilke. State space reductions for alternating Büchi automata: Quotienting by simulation equivalences. In *Proc. 22th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 2556 of *Lecture Notes in Computer Science*, pages 157–169, December 2002.

[Gab72] D.M. Gabbay. Applications of trees to intermediate logics i. *J. Symbolic Logic*, 37:135–138, 1972.

[GKSV03] S. Gurumurthy, O. Kupferman, F. Somenzi, and M.Y. Vardi. On complementing nondeterministic Büchi automata. In *12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 96–110. Springer-Verlag, 2003.

[GL94] G. De Giacomo and M. Lenzerini. Concept languages with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 411–415. John Wiley and Sons, 1994.

[HRS05] A. Harding, M. Ryan, and P. Schobbens. A new algorithm for strategy synthesis in ltl games. In *11th International Conference on Tools and algorithms for the construction and analysis of systems*, volume 3440 of *Lecture Notes in Computer Science*, pages 477–492. Springer-Verlag, 2005.

[Jur00] M. Jurdzinski. Small progress measures for solving parity games. In *17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer-Verlag, 2000.

[Kam85] M. Kaminski. A classification of ω-regular languages. *Theoretical Computer Science*, 36:217–229, 1985.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KP84] D. Kozen and R. Parikh. A decision procedure for the propositional μ-calculus. In *Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 313–325. Springer-Verlag, 1984.

[KPB94] S.C. Krishnan, A. Puri, and R.K. Brayton. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *Algorithms and Computations*, volume 834 of *Lecture Notes in Computer Science*, pages 378–386. Springer-Verlag, 1994.

[KPB95] S.C. Krishnan, A. Puri, and R.K. Brayton. Structural complexity of ω-automata. In *Symposium on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[KPBV95] S.C. Krishnan, A. Puri, R.K. Brayton, and P.P. Varaiya. The Rabin index and chain automata, with applications to automata and games. In *Computer Aided Verification, Proc. 7th International Conference*, pages 253–266, Liege, July 1995.

[KV98a] O. Kupferman and M.Y. Vardi. Freedom, weakness, and determinism: from linear-time to branching-time. In *Proc. 13th IEEE Symp. on Logic in Computer Science*, pages 81–92, June 1998.

[KV98b] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 224–233, Dallas, 1998.

[KV00] O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, January 2000.

[KV01] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. on Computational Logic*, 2(2):408–429, July 2001.

[KV04] O. Kupferman and M.Y. Vardi. Complementation constructions for nondeterministic automata on infinite words. In *11th International Conference on Tools and algorithms for the construction and analysis of systems*, volume 3440 of *Lecture Notes in Computer Science*, pages 206–221. Springer-Verlag, 2004.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.

[Lan69] L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.

[Mer00] S. Merz. Weak alternating automata in Isabelle/HOL. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics: 13th International Conference*, volume 1869 of *Lecture Notes in Computer Science*, pages 423–440. Springer-Verlag, 2000.

[Mey75] A. R. Meyer. Weak monadic second order theory of successor is not elementary recursive. In *Proc. Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 132–154. Springer-Verlag, 1975.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[Mos84] A.W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, volume 208 of *Lecture Notes in Computer Science*, pages 157–168. Springer-Verlag, 1984.

[MS85] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. In *Automata on Infinite Words*, volume 192, pages 100–107. Lecture Notes in Computer Science, Springer-Verlag, 1985.

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[MS95] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

[MSS86] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th International Colloquium on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*. Springer-Verlag, 1986.

[MSS88] D.E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings 3rd IEEE Symp. on Logic in Computer Science*, pages 422–427, Edinburgh, July 1988.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, Austin, January 1989.

[Rab69] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

[Rab70] M.O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Math. Logic and Foundations of Set Theory*, pages 1–23. North Holland, 1970.

[Ros92] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1992.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *IEEE Transactions on Control Theory*, 77:81–98, 1989.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.

[Saf89] S. Safra. *Complexity of automata on infinite objects*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1989.

[SE84] R.S. Street and E.A. Emerson. An elementary decision procedure for the μ-calculus. In *Proc. 11th International Colloquium on Automata, Languages and Programming*, volume 172, pages 465–472. Lecture Notes in Computer Science, Springer-Verlag, July 1984.

[ST03] R. Sebastiani and S. Tonetta. "more deterministic" vs. "smaller" büchi automata for efficient ltl model checking. In *12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 126–140. Springer-Verlag, 2003.

[Str82] R.S. Streett. Propositional dynamic logic of looping and converse. *Information and Control*, 54:121–141, 1982.

[THB95] S. Tasiran, R. Hojati, and R.K. Brayton. Language containment using nondeterministic omega-automata. In *Proc. of 8th CHARME: Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 987 of *Lecture Notes in Computer Science*, pages 261–277, Frankfurt, October 1995. Springer-Verlag.

[Tho98] W. Thomas. Complementation of Büchi automata revisited. *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–122, 1998.

[Tra62] B.A. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Math. J*, 3:101–131, 1962. Russian; English translation in: AMS Transl. 59 (1966), 23-55.

[TW68] J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–81, 1968.

[Var97] M.Y. Vardi. What makes modal logic so robustly decidable? In N. Immerman and Ph.G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, pages 149–183. American Mathematical Society, 1997.

[Var98] M.Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th International Coll. on Automata, Languages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer-Verlag, Berlin, July 1998.

[VW86] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–221, April 1986.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[WMBSV05] G. Wang, A. Mishchenko, R. Brayton, and A. Sangiovanni-Vincentelli. Synthesizing FSMs according to co-Büchi properties. Technical report, UC Berkeley, 2005.