Safraless Decision Procedures

Orna Kupferman*
Hebrew University

Moshe Y. Vardi†
Rice University

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 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. Our approach goes instead via universal co-Büchi tree automata and nondeterministic Büchi tree automata. Our translations are significantly simpler than the standard approach, less difficult to implement, and have practical advantages like being amenable to optimizations and a symbolic implementation. We also show that our approach yields better complexity bounds.

1 Introduction

The automata-theoretic approach is one of the most fundamental approaches to developing decision procedures in mathematical logics [34]. 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 [46]. 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.

The automata-theoretic approach was developed first for monadic logics over finite words [1, 6, 44]. It was then extended to infinite words in [2], to finite trees in [43], and finally generalized to infinite trees in [34]. 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., [14, 20]. 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) [26]. Thus, in the early 1980s, when decidability of highly expressive logics became of practical interest in areas such as formal verification and AI [15, 19], 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., [38, 39, 48].

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 [37], who described an optimal determinization construction for automata on infinite words, and by Emerson and Jutla [8] and Pnueli and Rosner [33], 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 [9, 31]. 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., [22]) and will not be addressed in this paper.) (2) translate this automaton to a nondeterministic parity tree automaton $\mathcal{A}^n_\psi$ using Safra's construction, 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

*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.

Proceedings of the 2005 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)
0-7695-2468-0/05 $20.00 © 2005 IEEE

reducing the upper time bound from octuply exponential [39] to singly exponential [47]), it proved to be not too amenable to implementation. First, Safra's construction proved quite resistant to efficient implementation [41]. Second, the best-known algorithms for parity-tree-automata emptiness are exponential [18]. Thus, while highly optimized software packages for automata on finite words and finite trees have been developed over the last few years [5], no such software has been developed for automata on infinite trees[1].

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 $\mathcal{A}_\psi$ to a universal 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 [30], 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 [48]. We also show that in some cases, (in particular, the *realizability and synthesis* [33] 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. We do show that, in addition, our approach yields better complexity bounds. Finally, as discussed in Section 6, our construction is amenable to several optimization techniques. Due to lack of space, some of the proofs are omitted in this version. A full version exists in the authors' URLs.

[1] An alternative translation of alternating tree automata to nondeterministic tree automata is described in [32]. Like Safra's construction, however, this translation is complicated.

[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-B¨uchi automata into nondeterministic B¨uchi automata. In practice, however, it is benefi cial to have many intermediate automata, as each intermediate automaton undergoes optimization constructions that are suitable for its particular type [12, 13, 16].

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 $\mathcal{T}$.

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.

- 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.

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 [29], 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 $\{D, N, U, A\} \times \{B, C, P, W\} \times \{W, T\}$, 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, 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 [35]) that recognizes $\mathcal{L}$. By [35], however, no NBT can recognize the complement of $\mathcal{L}$. Hence, by [29], 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.

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 [9], 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 [9] *The APT $\mathcal{A}$ accepts $\langle T, \tau\rangle$ iff there exists a running strategy $\langle T, f\rangle$ 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, index h, and alphabet Σ. There is a UCT $\mathcal{A}'$ with $O(nh)$ states and alphabet Σ' 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 [4]. □

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. In particular, in the full version we describe the application for the logics CTL* and *full μ-calculus*, which includes both forward and backward modalities [47].

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 $k = n2^{n \log n + 2n}$. 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 [48], so we get $2^{O(\log|D|+m+n^2h^2\log nh)}$. □

This improves 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 $2^{\log|D|+O(m+nh\log(nh))}$ The nonemptiness problem for DPT with transition function of size x and index y requires time $x^{O(y)}$ [18], so we get $2^{O(nh(\log|D|+m+nh\log nh))}$. This is worse than our $2^{O(\log|D|+m+n^2h^2\log nh)}$ bound. At any rate, the main advantage of our approach is the simplicity of the algorithm; the complexity analysis here just serves to show that this simplicity also yields a better 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 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.*

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.

Theorem 4.2 *A UCT $\mathcal{A}$ with n states is not empty iff $\mathcal{A}$ has an accepting run graph with at most $n2^{n\log n+2n}$ 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 [37][3], there is an NPT of size $n' = 2^{n\log n+2n}$ equivalent to $\mathcal{A}$. By [7], an NPT 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 $nn' = n2^{n\log n+2n}$. Since the run of $\mathcal{A}$ on the tree is accepting, Lemma 4.1 implies that so is the run graph. □

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.

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

[3] The complexity analysis in [37] is approximated, and the size of the equivalent NPT is $2^{O(n\log n)}$. Here we need an explicit bound, so we analyze Safra's construction. The state space of the NPT is the set of ordered trees over n nodes (trees in which the successors of each node are ordered) in which each node is labeled by a number in $\{1, \ldots, n\}$. There are $cat(n-1)$ ordered trees over n nodes, where cat stands for Catalan number. The explicit formula for $cat(n)$ is $(2n!)!/(n!(n+1)!)$. The asymptotic form is $4^n/(\sqrt{(}\pi)n^{3/2})$ [45], which is bounded by 4^n. This, together with the n^n factor for the possible labels of the nodes, gives the $2^{(n\log n+2n)}$ bound.

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

Lemma 4.3 *A run graph $G_r = \langle V, E\rangle$ is accepting iff there is $l \leq |V| + 1$ for which G_l is empty.*

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 4.3, there is $l \leq |V| + 1$ for which G_l is empty, Therefore, every vertex gets a well-defined rank, smaller than $|V|$. Since ranks along paths cannot increase and no α-vertex gets an odd rank, we have the following.

Lemma 4.4 *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 4.5 *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 $k = n2^{n\log n+2n}$. 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 $2^{n\log n+2n}$ 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 4.2, 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. Correctness then follows from Lemma 4.4. □

4.3 From AWT to NBT

In [27], Miyano and Hayashi describe a translation of ABW to NBW. In Theorem 4.6 below (see also [28]), we present (a technical variant of) their translation, adopted to tree automata,

Theorem 4.6 *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 4.5 and 4.6, 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 4.5, 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 4.7 *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 $k = n2^{n \log n + 2n}$. 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 $Q \times [k]$.

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 [9, 22]. 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, it is easy to define this intersection as an APT.

In the full version 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 problem for LTL.

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 finite-state transducer, such that all the computations of the system generated by f satisfy ψ [33]. 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 [33]. Thus, the algorithm applies Safra's construction, and has to solve the nonemptiness problem for DPT. We now show how UCW can be used instead of DPW.

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 $\neg\psi$ [49]. 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 $\delta'(q, o) = \bigwedge_{i \in 2^I} \bigwedge_{s \in \delta(q, i \cup o)} (i, s)$. 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 s 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 (in fact, as in the case of the satisfiability problem, the exact bound we get is better than the one obtained by the traditional algorithm) as well as the lower bound [36].

The *synthesis problem* for an LTL formula ψ is to find a strategy that realizes ψ. Known algorithms for the nonemptiness problem can be easily extended to return a witness to the automaton nonemptiness. 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. We 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 [40, 50] or give up completeness [17]. 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.

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 [3], 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., [48]) then yields a symbolic solution to the original UCT nonemptiness problem. Moreover, when applied for the solution of the realizability problem, the BDD that is generated by the symbolic procedure maintains a witness strategy. The Boolean nature of BDDs then makes it very easy to go from this BDD to a sequential circuit for the strategy. More advantages of the symbolic aproach are described in [17]. As mentioned above, [17] also suggests a symbolic solution for the LTL synthesis problem. However, the need to circumvent Safra's determinization causes the algorithm in [17] to be complete only for a subset of LTL. Our approach circumvents Safra's determinization without giving up completeness.

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 [16] 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 [16], 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 aproach has to do with the bound on the size of a "nonemptiness witness" in case the APT (or the UCT) is not empty. 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). 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 aproach translates a UCT with n states to an NBT with $2^{O(n^2 \log n)}$ states, whereas the Safrafull aproach 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 [23], 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 [23] involves an analysis of ranks. Alternating word automata are closely related to nondeterministic tree automata and the analysis in [23] have proven to be useful also for solving the nonemptiness problem for nondeterministic parity tree automata [21]. By now, the simple construction in [23] has become the standard complementation construction [42], has been implemented [16, 25], has led to tighter and new Safraless complementation constructions for richer types of automata on infinite words [11, 24], and has led to further implementations of alternating automata [10].

Since the bounded-width property trivially holds for runs of word automata, the analysis in [21, 23] 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 $r(y_1) = r(y_2)$. 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 [23] 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.

References

[1] J. B¨uchi. Weak second-order arithmetic and fi nite automata. *Zeit. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

[2] J. B¨uchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pages 1–12, Stanford, 1962.

[3] D. Bustan, O. Kupferman, and M. Vardi. A measured collapse of the modal μ-calculus alternation hierarchy. In *Proc. 21st STACS*, LNCS 2996, pages 522–533, 2004.

[4] Y. Choueka. Theories of automata on ω-tapes: A simplifi ed approach. *JCSS*, 8:117–141, 1974.

[5] J. Elgaard, N. Klarlund, and A. M¨oller. Mona 1.x: new techniques for WS1S and WS2S. In *Proc. 10th CAV*, LNCS 1427, pages 516–520, 1998.

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

[7] E. Emerson. Automata, tableaux, and temporal logics. In *Proc. WLOP*, LNCS 193, pages 79–87, 1985.

[8] E. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *29th FOCS*, pages 328–337, 1988.

[9] E. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd FOCS*, pages 368–377, 1991.

[10] B. Finkbeiner. Symbolic refi nement checking with nondeterministic BDDs. In *Proc. 7th TACAS*, LNCS 2031, 2001.

[11] E. Friedgut, O. Kupferman, and M. Vardi. B¨uchi complementation made tighter. In *Proc. 2nd ATVA*, LNCS 3299, pages 64–78, 2004.

[12] C. Fritz. Constructing B¨uchi automata from linear temporal logic using simulation relations for alternating bchi automata. In *Proc. 8th Intl. Conference on Implementation and Application of Automata*, LNCS 2759, pages 35–48, 2003.

[13] C. Fritz and T. Wilke. State space reductions for alternating B¨uchi automata: Quotienting by simulation equivalences. In *Proc. 22th FST & TCS*, LNCS 2556, pages 157–169, 2002.

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

[15] G. D. Giacomo and M. Lenzerini. Concept languages with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. 11th ECAI-94*, pages 411–415. John Wiley and Sons, 1994.

[16] S. Gurumurthy, O. Kupferman, F. Somenzi, and M. Vardi. On complementing nondeterministic Büchi automata. In *12th CHARME*, LNCS 2860, pages 96–110, 2003.

[17] A. Harding, M. Ryan, and P. Schobbens. A new algorithm for strategy synthesis in ltl games. In *Proc. 11th TACAS*, LNCS 3440, pages 477-492, 2005.

[18] M. Jurdzinski. Small progress measures for solving parity games. In *17th STACS*, LNCS 1770, pages 290–301, 2000.

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

[20] D. Kozen and R. Parikh. A decision procedure for the propositional μ-calculus. In *Logics of Programs*, LNCS 164, pages 313–325, 1984.

[21] O. Kupferman and M. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th STOC*, pages 224–233, 1998.

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

[23] O. Kupferman and M. Vardi. Weak alternating automata are not that weak. *ACM TOCL*, 2(3):408–429, 2001.

[24] O. Kupferman and M. Vardi. Complementation constructions for nondeterministic automata on infinite words. In *Proc. 11th TACAS*, LNCS 3440, pages 206-221, 2005.

[25] S. Merz. Weak alternating automata in Isabelle/HOL. In *Proc. 13th TPHOL*, LNCS 1869, pages 423–440, 2000.

[26] A. R. Meyer. Weak monadic second order theory of successor is not elementary recursive. In *Proc. Logic Colloquium*, LNM 453, pages 132–154, 1975.

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

[28] A. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, LNCS 208, pages 157–168, 1984.

[29] D. Muller, A. Saoudi, and P. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th ICALP*, LNCS 226, 1986.

[30] D. 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 *Proc. 3rd LICS*, pages 422–427, Edinburgh, July 1988.

[31] D. Muller and P. Schupp. Alternating automata on infinite trees. In *Automata on Infinite Words*, LNCS 192, pages 100–107, 1985.

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

[33] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.

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

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

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

[37] S. Safra. On the complexity of ω-automata. In *Proc. 29th FOCS*, pages 319–327, White Plains, October 1988.

[38] R. Street and E. Emerson. An elementary decision procedure for the μ-calculus. In *Proc. 11th ICALP*, LNCS 172, pages 465–472, July 1984.

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

[40] R. Sebastiani and S. Tonetta. "more deterministic" vs. "smaller" büchi automata for efficient ltl model checking. In *12th CHARME*, LNCS 2860, pages 126–140, 2003.

[41] S. Tasiran, R. Hojati, and R. Brayton. Language containment using non-deterministic omega-automata. In *Proc. 8th CHARME*, LNCS 987, pages 261–277, 1995.

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

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

[44] B. 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.

[45] I. Vardi. *Computational Recreations in Mathematica*. Addison-Wesley, Redwood City, CA, 1991.

[46] M. Vardi. What makes modal logic so robustly decidable? In *Descriptive Complexity and Finite Models*, pages 149–183. American Mathematical Society, 1997.

[47] M. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th ICALP*, LNCS 1443, pages 628–641, 1998.

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

[49] M. Vardi and P. Wolper. Reasoning about infinite computations. *I& C*, 115(1):1–37, November 1994.

[50] G. Wang, A. Mishchenko, R. Brayton, and A. Sangiovanni-Vincentelli. Synthesizing FSMs according to co-Büchi properties. Technical Report *UC BErkeley*, 2005.