Avoiding Determinization[*]

Orna Kupferman[†]
Hebrew University

Abstract

Automata on infinite objects are extensively used in system specification, verification, and synthesis. While some applications of the automata-theoretic approach have been well accepted by the industry, some have not yet been reduced to practice. Applications that involve determinization of automata on infinite words have been doomed to belong to the second category. This has to do with the intricacy of Safra's optimal determinization construction, the fact that the state space that results from determinization is awfully complex and is not amenable to optimizations and a symbolic implementation, and the fact that determinization requires the introduction of acceptance conditions that are more complex than the Büchi acceptance condition. Examples of applications that involve determinization and belong to the unfortunate second category include model checking of ω-regular properties, decidability of branching temporal logics, and synthesis and control of open systems.

We offer an alternative to the standard automata-theoretic approach. The crux of our approach is avoiding determinization. Our approach goes instead via universal co-Büchi automata. Like nondeterministic automata, universal automata may have several runs on every input. Here, however, an input is accepted if all of the runs are accepting. We show how the use of universal automata simplifies significantly known complementation constructions for automata on infinite words, known decision procedures for branching temporal logics, known synthesis algorithms, and other applications that are now based on determinization. Our algorithms are less difficult to implement and have practical advantages like being amenable to optimizations and a symbolic implementation.

[*]The Safraless plot described in this paper is based on joint work with Moshe Y. Vardi, appearing in [21, 24].

[†]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.

1 Introduction

Finite automata on infinite objects were first introduced in the 60's. Motivated by decision problems in mathematics and logic, Büchi, McNaughton, and Rabin developed a framework for reasoning about infinite words and infinite trees [4, 28, 36]. The framework has proved to be very powerful. Automata, and their tight relation to second-order monadic logics were the key to the solution of several fundamental decision problems in mathematics and logic [47]. Today, automata on infinite objects are used for specification and verification of nonterminating systems [49, 27, 51]. The automata-theoretic approach separates the logical and the combinatorial aspects of reasoning about systems. The translation of specifications to automata handles the logic and shifts all the combinatorial difficulties to automata-theoretic problems.

While some applications of the automata-theoretic approach have been well accepted by the industry, some have not yet been reduced to practice. As we detail below, applications that involve determinization is of automata on infinite words have been doomed to belong to the second category. This has to do with the intricacy of Safra's optimal determinization construction, the fact that the state space that results from determinization is awfully complex and is not amenable to optimizations and a symbolic implementation, and the fact that determinization requires the introduction of acceptance conditions that are more complex than the Büchi acceptance condition.

Let us examine some examples, and let us start with perhaps the most successful and influential application of automata theory in formal verification : linear time model checking [51]. In the automata-theoretic approach to model checking, we check the correctness of a system with respect to a specification by checking containment of the language of the system in the language of an automaton that accepts exactly all computations that satisfy the specification. In order to check the latter, we check that the intersection of the system with an automaton that accepts exactly all the computations that violate the specification is empty. For instance, LTL model checking usually proceeds by translating the negation of an LTL formula into a Büchi automaton.

Difficulties start when properties are specified by ω-regular automata. Then, one needs to complement the property automaton. Efforts for developing complementation constructions for nondeterministic Büchi automata started early in the 60s. Büchi suggested a complementation construction that involved a doubly-exponential blow-up in the state space. Thus, complementing an automaton with n states resulted in an automaton with $2^{2^{O(n)}}$ states [4]. Büchi's motivation was decidability of S1S. In the mid 80s, when complementation became of practical interest in formal verification, and complexity-theoretic considerations started to play a greater role, the problem was re-examined and a construction with $2^{O(n^2)}$ states was suggested in [42].

Only in [40], however, Safra introduced an optimal determinization construction, which also enabled a $2^{O(n \log n)}$ complementation construction, matching the known lower bound [30]. Safra's determinization construction is beautiful. In order to obtain the optimal bound, Safra defined each state of the deterministic automaton to be a tree of subset constructions that cleverly maintains the essential information about all possible runs (unlike automata on finite words, the set of states reachable in all possible runs does not provide sufficient information, as we also need information about the set of states that have been visited along each run). While being the heart of many complexity results in verification, the construction in [40] is complicated and difficult to implement. Efforts to implement it [46, 2] have to cope with the awfully complex state space of the deterministic automaton, which is amenable to optimizations and a symbolic representation. Almost 20 years have passed since the introduction of Safra's construction, and no implementation that can handle automata with more than 8 states exists.

Due to the lack of a simple complementation construction, users are typically required to specify the property by deterministic Büchi automata (it is easy to complement a deterministic automaton [26]), or to supply the automaton for the negation of the property [16]. Similarly, specification formalisms like ETL [53], which have automata within the logic, involve complementation of automata, and the difficulty of complementing Büchi automata is an obstacle to practical use [3]. In fact, even when the properties are specified in LTL, complementation is useful: the translators from LTL into automata have reached a remarkable level of sophistication (cf. [43, 11]). Even though complementation of the automata is not explicitly required, the translations are so involved that it is useful to checks their correctness, which involves complementation[1]. Complementation is interesting in practice also because it enables refinement and optimization techniques that are based on language containment rather than simulation. Thus, an effective algorithm for the complementation of Büchi automata is of significant practical value.

Let us move on to another important application — decidability of branching temporal logics. Using automata on infinite trees, Rabin was able to prove the decidability of SnS, the monadic theory of infinite trees. In fact, SnS decidability was the motivation for extending the automata-theoretic framework to infinite trees [36]. The complexity of SnS decidability is known to be nonelementary [29]. Thus, while decidability of many logics has been established by demonstrating an effective reduction to SnS, this approach was no longer appealing when decidability became of practical interest in areas such as formal verification and AI [12, 19]. This is when 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., [44, 45, 50].

By the mid 1980s, the focus was on using automata to obtain tighter upper bounds. Safra's optimal determinization construction has led to a breakthrough progress also in the branching setting. Indeed, the translation of branching temporal logic formulas to automata on infinite trees typically involves determinization of automata on infinite words that are associated with linear requirements in the formula. More progress was attained by improved algorithms for the nonemptiness problem of nondeterministic tree automata [8, 35]. The introduction of alternating automata on infinite trees [9, 31] simplified this approach further. In the now standard approach for checking whether a formula ψ is satisfiable, one constructs 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 simple and well known, c.f., [9, 25]), and then checks that the language of $\mathcal{A}_\psi$ is nonempty.

While the above approach yielded significantly improved upper bounds (in some cases reducing the upper time bound from octuply exponential [45] to singly exponential [48]), it proved to be not too amenable to implementation. First, checking the nonemptiness of alternating parity tree automata requires their translation to nondeterministic parity tree automata. Such removal of alternation involves determinization of word automata, and thus involves Safra's construction[2]. Second, the best-known algorithms for nonemptiness of nondeterministic parity tree automata are exponential [17]. Implementing them on top of the messy state space that results from Safra's determinization is practically impossible.

As a final example, consider the synthesis problem for

[1]For an LTL formula ψ, one typically checks that both the intersection of $\mathcal{A}_\psi$ with $\mathcal{A}_{\neg\psi}$ and the intersection of their complementary automata are empty.

[2]An alternative construction for removal of alternation is described in [33]. Like Safra's construction, however, this construction is very complicated and we know of no implementation of it.

linear specifications. When a system is reactive, it interacts with the environment, and a correct system should satisfy the specification with respect to all environments. As argued in [1, 6, 35], the right way to approach synthesis of reactive systems is to consider the situation as a (possibly infinite) game between the environment and the system. A correct system can be then viewed as a winning strategy in this game. The traditional algorithm for finding a winning strategy for the system transforms the specification into a parity automaton over trees that embody all the possible inputs to the system. The system is realizable precisely when this tree automaton is not empty [35]. A finite generator of an infinite tree accepted by this automaton can be viewed as a finite-state system realizing the specification. This is closely related to the approach taken, e.g., in [38], to solve Church's solvability problem [5].

In spite of the rich theory developed for system synthesis, little of this theory has been reduced to practice. Some people argue that this is because the realizability problem for LTL specifications is 2EXPTIME-complete [35, 39], but this argument is not compelling. First, experience with verification shows that even nonelementary algorithms can be practical, since the worst-case complexity does not arise often (c.f., the model-checking tool MONA [7]). Furthermore, in some sense, synthesis is not harder than verification. Indeed, realizable specifications for which the solution of the synthesis problem is doubly exponential, require systems of doubly exponential size for their realization [39]. While the verification of such systems is linear in the size of the system, is doubly exponential in the specification. We believe that, as with satisfiability, the main reason for the lack of practical impact of synthesis theory is the fact the generation of the tree automaton uses Safra's determinization construction, and the check for its emptiness requires an execution of a parity-tree-automata emptiness algorithm on top of its messy state space. 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 [41, 52] or give up completeness [15].

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. Instead, we use universal automata. Like nondeterministic automata, universal automata may have several runs on every input. Here, however, an input is accepted if all of the runs are accepting. Universal automata are sufficiently strong to play the role that deterministic automata play in the current algorithms. Complementing a nondeterministic automaton can be done by dualizing its acceptance condition and viewing it as a universal automaton. In addition, universal automata have the desired property, enjoyed by deterministic automata but not by nondeterministic automata, of having the ability to run over all branches of an input tree – this is required for both satisfiability and synthesis. Using universal automata, we can also avoid the parity acceptance condition. For complementation, dualizing the Büchi acceptance condition, one gets the co-Büchi condition. This observation is helpful also in the context of synthesis, as a universal co-Büchi automaton for a required behavior ψ can be obtained by dualizing a nondeterministic Büchi automaton for $\neg\psi$. For satisfiability, we show that an alternating parity tree automaton can be reduced[3] to a universal co-Büchi tree automaton.

By analyzing runs of universal co-Büchi word automata, we are able to translate them to nondeterministic Büchi automata. Such a translation completes a complementation construction for nondeterministic Büchi word automata. An analysis of runs of universal co-Büchi tree automata is more sophisticated, and it enables us to reduce universal co-Büchi tree automata to nondeterministic Büchi tree automata. Such a reduction completes a solution for the decidability and synthesis problems. 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, in some cases, the complexity is improved). In addition, they give rise to several significant optimizations and heuristics.

The idea of avoiding determinization was first suggested in the context of complementation in [18], which described a $2^{O(n \log n)}$ Safraless complementation construction. The analysis of runs of universal co-Büchi automata that we do here is similar to the progress-measures introduced there. Unfortunately, the complementation construction in [18] is complicated and we know of no implementation of it. We believe that the simplicity of our approach follows from the fact we explicitly use universal co-Büchi automata as an intermediate step (in fact, as described in [21], it is possible to decompose our construction further and use alternating weak automata as another intermediate step). A more recent effort to avoid determinization in the context of synthesis is described in [14]. There, the challenge is to cope with the fact that when objectives in two-player games are ω-regular, current solutions construct the product of the game with a deterministic automaton for the objective. Instead, Henzinger and Piterman suggest to leave the objective automaton nondeterministic, but to restructure its state space and transitions so that taking its product with the system does solve the original game. The construction of "good-for-games automata" involves an inevitable exponential blow up, but is much simpler than Safra's determinization.

[3]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.

2 Safraless Complementation

Given an alphabet Σ, an *infinite word over* Σ is an infinite sequence $w = \sigma_0 \cdot \sigma_1 \cdot \sigma_2 \cdots$ of letters in Σ. We denote by w^l the suffix $\sigma_l \cdot \sigma_{l+1} \cdot \sigma_{l+2} \cdots$ of w. An *automaton on infinite words* is $\mathcal{A} = \langle \Sigma, Q, Q_{in}, \rho, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\rho : Q \times \Sigma \to 2^Q$ is a transition function, $Q_{in} \subseteq Q$ is a set of initial states, and α is an acceptance condition (a condition that defines a subset of Q^ω). Intuitively, $\rho(q, \sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ. Since the transition function of $\mathcal{A}$ may specify many possible transitions for each state and letter, $\mathcal{A}$ is not *deterministic*. If ρ is such that for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\rho(q, \sigma)| = 1$, then $\mathcal{A}$ is a deterministic automaton.

A *run* of $\mathcal{A}$ on w is a function $r : \mathbb{N} \to Q$ where $r(0) \in Q_{in}$ (i.e., the run starts in an initial state) and for every $l \geq 0$, we have $r(l+1) \in \rho(r(l), \sigma_l)$ (i.e., the run obeys the transition function). Acceptance is defined according to the set $Inf(r)$ of states that r visits *infinitely often*, i.e., $Inf(r) = \{q \in Q : r(l) = q$ for infinitely many $l \in \mathbb{N}\}$. As Q is finite, it is guaranteed that $Inf(r) \neq \emptyset$. The way we refer to $Inf(r)$ depends on the acceptance condition of $\mathcal{A}$. In *Büchi automata*, $\alpha \subseteq Q$, and r is accepting iff $Inf(r) \cap \alpha \neq \emptyset$. Dually, in *co-Büchi automata*, $\alpha \subseteq Q$, and r is accepting iff $Inf(r) \cap \alpha = \emptyset$.

Since $\mathcal{A}$ is not deterministic, it may have many runs on w. In contrast, a deterministic automaton has a single run on w. There are two dual ways in which we can refer to the many runs. When $\mathcal{A}$ is a *nondeterministic* automaton, it accepts an input word w iff there exists an accepting run of $\mathcal{A}$ on w. When $\mathcal{A}$ is a *universal* automaton, it accepts an input word w iff all the runs of $\mathcal{A}$ on w are accepting.

We denote each of the different types of automata (some will be defined only in the sequel) by three letter acronyms in $\{D, N, U, A\} \times \{B, C, P, R, S\} \times \{W, T\}$, 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, or Streett), and the third letter describes the object over which the automaton runs (words or trees). For example, APT stands for an alternating parity tree automaton and UCW stands for a universal co-Büchi word automaton.

Our Safraless complementation construction proceeds as follows. In order to complement an NBW, first dualize the transition function and the acceptance condition, and then translate the resulting UCW automaton back to an NBW. By [32], the dual automaton accepts the complementary language, and therefore, so does the nondeterministic automaton we end up with. Thus, rather than determinization, complementation is based on a translation of universal automata to nondeterministic ones. We now give the technical details of the construction.

Consider a UCW $\mathcal{A} = \langle \Sigma, Q, Q_{in}, \delta, \alpha \rangle$. The runs of $\mathcal{A}$ on a word $w = \sigma_0 \cdot \sigma_1 \cdots$ can be arranged in an infinite DAG (directed acyclic graph) $G = \langle V, E \rangle$, where

- $V \subseteq Q \times \mathbb{N}$ is such that $\langle q, l \rangle \in V$ iff some run of $\mathcal{A}$ on w has $q_l = q$. For example, the first level of G contains the vertices $Q_{in} \times \{0\}$.
- $E \subseteq \bigcup_{l \geq 0} (Q \times \{l\}) \times (Q \times \{l+1\})$ is such that $E(\langle q, l \rangle, \langle q', l+1 \rangle)$ iff $\langle q, l \rangle \in V$ and $q' \in \delta(q, \sigma_l)$.

Thus, G embodies exactly all the runs of $\mathcal{A}$ on w. We call G the *run* DAG of $\mathcal{A}$ on w, and we say that G is *accepting* if all its paths satisfy the acceptance condition α. Note that $\mathcal{A}$ accepts w iff G is accepting. We say that a vertex $\langle q', l' \rangle$ is a *successor* of a vertex $\langle q, l \rangle$ iff $E(\langle q, l \rangle, \langle q', l' \rangle)$. We say that $\langle q', l' \rangle$ is *reachable* from $\langle q, l \rangle$ iff there exists a sequence $\langle q_0, l_0 \rangle, \langle q_1, l_1 \rangle, \langle q_2, l_2 \rangle, \ldots$ of successive vertices such that $\langle q, l \rangle = \langle q_0, l_0 \rangle$, and there exists $i \geq 0$ such that $\langle q', l' \rangle = \langle q_i, l_i \rangle$. For a set $S \subseteq Q$, we say that a vertex $\langle q, l \rangle$ of G is an *S-vertex* if $q \in S$.

Consider a (possibly finite) DAG $G' \subseteq G$. We say that a vertex $\langle q, l \rangle$ is *finite* in G' if only finitely many vertices in G' are reachable from $\langle q, l \rangle$. For a set $S \subseteq Q$, we say that a vertex $\langle q, l \rangle$ is *S-free* in G' if all the vertices in G' that are reachable from $\langle q, l \rangle$ are not S-vertices. Note that, in particular, an S-free vertex is not an S-vertex. We say that a level l of G' is of *width* $d \geq 0$ if there are d vertices of the form $\langle q, l \rangle$ in G'. Finally, the *width* of G' is the maximal $d \geq 0$ such that there are infinitely many levels l of width d. The *α-less* width of a level of $\mathcal{G}$ is defined similarly, restricted to vertices $\langle q, l \rangle$ for which $q \notin \alpha$. Note that the width of G is at most n and the α-less width of G_r is at most $n - |\alpha|$.

Runs of UCW were studied in [21]. For $x \in \mathbb{N}$, let $[x]$ denote the set $\{0, 1, \ldots, x\}$, and let $[x]^{odd}$ and $[x]^{even}$ denote the set of odd and even members of $[x]$, respectively. A *co-Büchi-ranking* for G (*C-ranking*, for short) is a function $f : V \to [2n]$ that satisfies the following two conditions:

1. For all vertices $\langle q, l \rangle \in V$, if $f(\langle q, l \rangle)$ is odd, then $q \notin \alpha$.
2. For all edges $\langle \langle q, l \rangle, \langle q', l+1 \rangle \rangle \in E$, we have $f(\langle q', l+1 \rangle) \leq f(\langle q, l \rangle)$.

Thus, a C-ranking associates with each vertex in G a rank in $[2n]$ so that the ranks along paths do not increase, and α-vertices get only even ranks. We say that a vertex $\langle q, l \rangle$ is an *odd vertex* if $f(\langle q, l \rangle)$ is odd. Note that each path in G eventually gets trapped in some rank. We say that the C-ranking f is an *odd C-ranking* if all the paths of G eventually get trapped in an odd rank. Formally, f is odd iff for all paths $\langle q_0, 0 \rangle, \langle q_1, 1 \rangle, \langle q_2, 2 \rangle, \ldots$ in G, there is $l \geq 0$

such that $f(\langle q_l, l\rangle)$ is odd, and for all $l' \geq l$, we have $f(\langle q_{l'}, l'\rangle) = f(\langle q_l, l\rangle)$. Note that, equivalently, f is odd if every path of G has infinitely many odd vertices.

Lemma 2.1 [21] *The following are equivalent.*

1. *All the paths of G have only finitely many α-vertices.*
2. *There is an odd C-ranking for G.*

Proof: Assume first that there is an odd C-ranking for G. Then, every path in G eventually gets trapped in an odd rank. Hence, as α-vertices get only even ranks, all the paths of G visit α only finitely often, and we are done.

For the other direction, given an accepting run DAG G, we define an infinite sequence $G_0 \supseteq G_1 \supseteq G_2 \supseteq \ldots$ of DAGs inductively as follows.

- $G_0 = G$.
- $G_{2i+1} = G_{2i} \setminus \{\langle q, l\rangle \mid \langle q, l\rangle$ is finite in $G_{2i}\}$.
- $G_{2i+2} = G_{2i+1} \setminus \{\langle q, l\rangle \mid \langle q, l\rangle$ is α-free in $G_{2i+1}\}$.

It is shown in [21] that for every $i \geq 0$, the transition from G_{2i+1} to G_{2i+2} involves the removal of an infinite path from G_{2i+1}. Since the width of G_0 is bounded by n, it follows that the width of G_{2i} is at most $n - i$. Hence, G_{2n} is finite, and G_{2n+1} is empty. In fact, as argued in [13], the α-less width of G_{2i} is at most $n - (|\alpha| + i)$, implying that $G_{2(n-|\alpha|)+1}$ is already empty. Since $|\alpha| \geq 1$, we can therefore assume that G_{2n-1} is empty.

Each vertex $\langle q, l\rangle$ in G has a unique index $i \geq 1$ such that $\langle q, l\rangle$ is either finite in G_{2i} or α-free in G_{2i+1}. Thus, the sequence of DAGs induces a function $rank : V \to [2n-2]$, defined as follows.

$$rank(q, l) = \left[\begin{array}{ll} 2i & \text{If } \langle q, l\rangle \text{ is finite in } G_{2i}. \\ 2i+1 & \text{If } \langle q, l\rangle \text{ is } \alpha\text{-free in } G_{2i+1}. \end{array} \right.$$

It is shown in [21] that the function *rank* is an odd C-ranking. □

We now use C-ranking in order to translate UCWs to NBWs.

Theorem 2.2 [21] *Let $\mathcal{A}$ be a* UCW *with n states. There is an* NBW *$\mathcal{A}'$ with $2^{O(n \log n)}$ states such that $L(\mathcal{A}') = L(\mathcal{A})$.*

Proof: Let $\mathcal{A} = \langle \Sigma, Q, Q_{in}, \delta, \alpha\rangle$. When $\mathcal{A}'$ reads a word w, it guesses an odd C-ranking for the run DAG G of $\mathcal{A}$ on w. At a given point of a run of $\mathcal{A}'$, it keeps in its memory a whole level of G and a guess for the rank of the vertices at this level. In order to make sure that all the paths of G visit infinitely many odd vertices, $\mathcal{A}'$ remembers the set of states that owe a visit to an odd vertex.

Before we define $\mathcal{A}'$, we need some notations. A *level ranking* for $\mathcal{A}$ is a function $g : Q \to [2n-2]$, such that if $g(q)$ is odd, then $q \notin \alpha$. Let $\mathcal{R}$ be the set of all level rankings. For a subset S of Q and a letter σ, let $\delta(S, \sigma) = \bigcup_{s \in S} \delta(s, \sigma)$. Note that if level l in G, for $l \geq 0$, contains the states in S, and the $(l+1)$-th letter in w is σ, then level $l+1$ of G contains the states in $\delta(S, \sigma)$.

For two level rankings g and g' in $\mathcal{R}$ and a letter σ, we say that g' *covers* $\langle g, \sigma\rangle$ if for all q and q' in Q, if $q' \in \delta(q, \sigma)$, then $g'(q') \leq g(q)$. Thus, if g describes the ranks of the vertices of level l, and the $(l+1)$-th letter in w is σ, then g' is a possible level ranking for level $l+1$. Finally, for $g \in \mathcal{R}$, let $odd(g) = \{q : g(q) \in [2n-2]^{odd}\}$. Thus, a state of Q is in $odd(g)$ if has an odd rank.

Now, $\mathcal{A}' = \langle \Sigma, Q', Q'_{in}, \delta', \alpha'\rangle$, where

- $Q' = 2^Q \times 2^Q \times \mathcal{R}$, where a state $\langle S, O, g\rangle \in Q'$ indicates that the current level of the DAG contains the states in S, the set $O \subseteq S$ contains states along paths that have not visited an odd vertex since the last time O has been empty, and g is the guessed level ranking for the current level.
- $Q'_{in} = \{Q_{in}\} \times \{\emptyset\} \times \mathcal{R}$.
- δ' is defined, for all $\langle S, O, g\rangle \in Q'$ and $\sigma \in \Sigma$, as follows.
  - If $O \neq \emptyset$, then $\delta'(\langle S, O, g\rangle, \sigma) = \{\langle \delta(S, \sigma), \delta(O, \sigma) \setminus odd(g'), g'\rangle : g'$ covers $\langle g, \sigma\rangle\}$.
  - If $O = \emptyset$, then $\delta'(\langle S, O, g\rangle, \sigma) = \{\langle \delta(S, \sigma), \delta(S, \sigma) \setminus odd(g'), g'\rangle : g'$ covers $\langle g, \sigma\rangle\}$.
- $\alpha' = 2^Q \times \{\emptyset\} \times \mathcal{R}$.

Consider a state $\langle S, O, g\rangle \in Q'$. Since $O \subseteq S$, there are at most 3^n pairs S and O that can be members of the same state. In addition, since there are at most $(2n-1)^n$ level rankings, the number of states in $\mathcal{A}'$ is at most $3^n \cdot (2n-1)^n$, which is $2^{O(n \log n)}$ □

Corollary 2.3 *Let $\mathcal{A}$ be an NBW with n states. There is an NBW $\tilde{\mathcal{A}}$ with $2^{O(n \log n)}$ states such that $L(\tilde{\mathcal{A}}) = \Sigma^\omega \setminus L(\mathcal{A})$.*

2.1 Remarks

2.1.1 A tighter construction Both Safra's construction and our construction results in $\tilde{\mathcal{A}}$ with $2^{O(n \log n)}$ states, matching Michel's lower bound. A careful analysis, however, of the exact blow-up in Safra's and Michel's bounds reveals an exponential gap in the constants hiding in the $O()$ notations: while the upper bound on the number of states in Safra's complementary automaton is n^{2n}, Michel's lower bound involves only an $n!$ blow up, which is roughly $(n/e)^n$. The construction above does better: since $(1 + \frac{x}{n})^n = e^x$, the $3^n \cdot (2n-1)^n$ bound in Theorem 2.2 is equal to $(6n)^n/\sqrt{e}$. This is still far from Michel's lower bound.

In [10] we improved the construction further and described a construction that results in an NBW with at most $(0.96n)^n$ states. The idea is as follows. Let k be the maximal odd rank that some vertex in G has. There is a level l in G such that all the odd ranks below k appear in all the levels above l. Intuitively, it follows from the fact that odd ranks correspond to vertices that are α-free, and there is a level l that has an α-free vertex in all the intermediate DAGs G_i, for each odd i below k. This observation suggests that the NBW $\mathcal{A}'$ can guess the level l and restrict the level rankings g that are guessed for the levels above it to level rankings in which all odd ranks below k appear. In addition, in a state $\langle S, O, g\rangle$, the level ranking g need not refer to states not in S. The above considerations significantly reduce the the number of potential level rankings, and lead to the $(0.96n)^n$ bound.

We note that the lower bound for NBW complementation was recently tightened too: a new technique by Yan implies a $(0.76n)^n * poly(n)$ lower bound [54]. Thus, there is still a gap between the upper and lower bounds, but it is less significant than the gap between Safra's and Michel's bound.

2.1.2 Finding the minimal rank required

A drawback of our construction in Theorem 2.2 is that it never performs better than its worst-case complexity. Indeed, the $3^n \cdot (2n-1)^n$ blow-up is introduced regardless of the structure of $\mathcal{A}$ and would occur even if, say, $\mathcal{A}$ is a deterministic automaton. In order to circumvent such an unnecessary blow up, we suggest to first calculate the *minimal rank required for* $\mathcal{A}$ (formally defined below), and then to construct $\mathcal{A}'$ with respect to this rank.

For every $j \in [n]$, we define the NBW $\mathcal{A}'_j$ as a restriction of $\mathcal{A}'$ from Theorem 2.2 to states $\langle S, O, g\rangle$ in which $g : Q \to [2j-2]$. Thus, $\mathcal{A}'_j$ restricts the runs of $\mathcal{A}'$ to guess only ranks smaller than $2j-2$. It is easy to see that for every j, the language of $\mathcal{A}'_j$ is contained in the language of $\mathcal{A}'$. On the other hand, the language of $\mathcal{A}'_j$ contains only these words in $L(\mathcal{A}')$ for which G_{2j+1} is empty. The minimal rank required for $\mathcal{A}$ is the minimal $j \in [n]$ for which $L(\mathcal{A}) \subseteq L(\mathcal{A}'_j)$.

As demonstrated in the experimental results in [13], this minimal rank is often significantly smaller than n. Also, the size of $\mathcal{A}'_j$ is only $3^n(2j-1)^n$. As discussed in [21], the problem of finding the minimal rank required for $\mathcal{A}$ is PSPACE-complete.

2.1.3 An incremental approach

As stated above, the problem of finding the minimal rank required for $\mathcal{A}$ requires space that is polynomial in $\mathcal{A}$. Nevertheless, the automaton $\mathcal{A}$ is typically small, and the bottle-neck of the computation is usually the application of $\mathcal{A}'$ (e.g., taking its product with a system with a large state space). Thus, finding the minimal rank j required for $\mathcal{A}$ and using $\mathcal{A}'_j$ instead of $\mathcal{A}'$ may be of great practical importance.

Moreover, for the language-containment application, one need not calculate the minimal required rank and can check the containment of S in $\mathcal{A}$ by checking the emptiness of $S \cap \mathcal{A}'_j$ for increasing j's. In Section 4.1.2, we describe the incremental approach in more detail. There, we also note that it is possible to take advantage of the work done during the emptiness test of $S \cap \mathcal{A}'_j$, when testing emptiness of $S \cap \mathcal{A}'_{j'}$, for $j' > j$.

2.1.4 More heuristics

In [13], we used the fact that the state space of $\mathcal{A}'$ is simple and suggested an arsenal of optimization techniques that can be applied to it. The optimizations make use of the fact that the construction of $\mathcal{A}'$ from $\mathcal{A}$ may use an intermediate alternating week automaton, and are applied on both the intermediate automaton and the final NBW. The optimizations involve techniques of rank reduction (described above), height reductions (repeatedly removing a minimal strongly connected component, as long as such a removal does not change the language of the intermediate automaton), as well as direct and fair simulation. As detailed in [13], the construction and the optimizations have been implemented and they significantly reduce the size of the state space of $\mathcal{A}'$.

2.1.5 Safraless complementation of nondeterministic Rabin and Streett automata

In [23], we extended the ranking technique to Rabin and Streett automata, and use the analysis in order to describe simple complementation constructions for NRW and NSW. Thus, also in these classes, it is possible to avoid determinization and complement automata by dualizing them to universal automata.

3 Safraless Decision Procedures

The key idea in [21] is that when all the paths of a DAG have only finitely many α-vertices, and finite vertices are removed, the removal of α-free vertices results in a DAG with a strictly smaller width. Consequently, when the width of the DAG is bounded by some $k \geq 1$ (and in the case of a run DAG of a UCW with n states we know that $k \leq n$), iterative removal of finite and α-free vertices results in an empty DAG after at most k iterations. This is why every vertex can be associated with a finite rank bounded by $2k$, and the translation of UCWs to NBWs proceeds by letting the NBW guess the ranks.

In order to solve the satisfiability problem for branching temporal logics, we reduce the satisfiability problem to the emptiness problem of UCTs. We then solve the emptiness problem for UCTs by reducing them to NBTs. Runs of a UCT can also be arranged in a run DAG, and as in the linear case, when all the paths of the DAG have only finitely many α-vertices, and finite vertices are removed, the removal of α-free vertices results in a DAG with a strictly smaller width. In the case of tree automata, however, one crucial factor is missing: we do not have a bound on the width of the run DAG. Indeed, since the UCT runs on trees whose width is not bounded, the width of the run DAG is not bounded either.

The way we solve this problem is as follows. Recall that our motivation is the satisfiability problem. Therefore, we do not have to construct an NBT $\mathcal{A}'$ that is equivalent to the UCT $\mathcal{A}$, and we only need an NBT that is emptiness-equivalent to the UCT (that is, $L(\mathcal{A}') \neq \emptyset$ iff $L(\mathcal{A}) \neq \emptyset$). Accordingly, the NBTs we construct are parameterized by a parameter k and the NBT $\mathcal{A}'_k$ accepts only trees whose accepting run graph (graph, rather than a DAG, as we have to refer to its size and not to its width) has at most k vertices. By a bounded model property for UCTs, we know that $L(\mathcal{A}) \neq \emptyset$ iff $L(\mathcal{A}'_k) \neq \emptyset$ for $k = n^{2n+3}$. The bounded model property relies on Safra's determinization construction (k above depends on the number of states in a DSW equivalent to an NBW induced by $\mathcal{A}$), but it is only the correctness of the our construction that relies on Safra's construction: once we have k, the construction is independent of the intricacy of Safra's construction.

Below we describe the construction briefly. The full details, as well as the definitions of trees, alternating tree automata, and run graphs, can be found in [24]. The input to the satisfiability problem is a μ-calculus formula ψ [19]. The formula ψ can be translated to an APT with no blow up [9]. Such translations exist for several other branching temporal logics [25]. Thus, the problem we need to solve is APT emptiness, and we start with a reduction of APT emptiness to UCT emptiness.

UCTs are a special case of APTs and are strictly less expressive than APTs. The emptiness problem for APTs can be still easily reduced to the emptiness problem for UCTs. The idea is to enrich the alphabet of the APT by a "strategy component" — information on how nondeterminizm is going to be resolved in the current transition. Then, guesses of the alternating automaton are reduced to guesses about the input letter, and the automaton becomes universal, over a richer alphabet. In addition, by changing the state space of the automaton, the parity condition is replaced by a co-Büchi condition. Formally, we have the following.

Theorem 3.1 [24] *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 $L(\mathcal{A}) \neq \emptyset$ iff $L(\mathcal{A}') \neq \emptyset$.*

As explained above, in order to reduce the UCT to an NBT, we first need a bounded-model property for UCTs, which also implies a bound on the size of an accepting run graph. Let $det(n) = n^{2n+2}$.

Theorem 3.2 [24] *A UCT $\mathcal{A}$ with n states is not empty iff $\mathcal{A}$ has an accepting run graph with at most $n \cdot det(n)$ vertices.*

By translating the UCT to an NBT that accepts exactly all trees whose accepting run graph has at most $n \cdot det(n)$ vertices, we finally get the following.

Theorem 3.3 [24] *Let $\mathcal{A}$ be a UCT with n states. There is an NBT $\mathcal{A}'$ over the same alphabet such that $L(\mathcal{A}') \neq \emptyset$ iff $L(\mathcal{A}) \neq \emptyset$, and the number of states in $\mathcal{A}'$ is $2^{O(n^2 \log n)}$.*

Combining Theorems 3.1 and 3.3, we get the desired reduction from the nonemptiness problem for APT to the nonemptiness problem for NBT:

Theorem 3.4 [24] *Let $\mathcal{A}$ be an APT with n states, transition function of size m, and index h. There is an NBT $\mathcal{A}'$ with $2^{O(n^2h^2 \log nh)}$ states and alphabet of size $2^{O(m)}$ such that $L(\mathcal{A}) \neq \emptyset$ iff $L(\mathcal{A}') \neq \emptyset$.*

The complexity of the nonemptiness algorithm for APT that follows coincides with the known one. The main advantage of our approach is the fact it avoids Safra's determinization and the need to solve parity games on top of it, and the fact it enables an incremental and symbolic implementation. We will get back to this point in Section 4.

We note that an improvement of the upper bound of NBW determinization would lead to an improvement in the complexity of our decision procedure. Indeed, $det(n)$ is the blow up that determinization involves, and it affects the range of ranks that the NBT has to guess. In fact, the bound described here is better than the one in [24], and the improvement is due to Piterman's recent determinization construction [34].

4 Safraless 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[4] such that all the computations of the system generated by f satisfy ψ [35]. 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 $o_j = f(i_0 \cdot i_1 \cdots i_{j-1})$.

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

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 [35]. 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 [39]. 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 known translations of LTL into NBW can be used in order to translate LTL into UCW. Formally, we have the following.

Theorem 4.1 [24] *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 $\neg\psi$ [51]. 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 [39].

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 [37]. 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 (as well as related richer problems, like supervisory-control and synthesis with incomplete information). 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. We believe, however, that this drawback is compensated by the practical advantages, discussed below, of our approach.

4.1 Remarks

4.1.1 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, 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., [50]) 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. In [15], the authors suggest a symbolic solution for the LTL synthesis problem. However, the need to circumvent Safra's determinization causes the algorithm in [15] to be complete only for a subset of LTL. Our approach circumvents Safra's determinization without giving up completeness.

4.1.2 An incremental approach Our construction is based on the fact we can bound the maximal rank that a vertex of G can get by $k = n \cdot det(n)$ — the bound on the size of the run graphs of $\mathcal{A}$. Often, the bound on the maximal rank much smaller. Accordingly, as in the linear case, 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 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 blow-up that is introduced in the translation of a UCT to an NBT occurs for all UCT. With the incremental algorithm, the blow occurs only in the worst case. As shown in [13], experimental results show that in the case of word automata the construction typically ends up with a small k.

4.1.3 Ranks for generalized universal co-Büchi automata In [22, 20], we extended the ranking analysis to universal *generalized co-Büchi* word and tree automata. Consequently, we can handles LTL formulas by translating them to nondeterministic generalized Büchi automata. This leads to an exponential improvement in the complexity of the algorithm. Since our Safraless approach uses a "Safraful" bound on the size of run graph, the extension to generalized co-Büchi automata required, on top of the above analysis of runs, also an extension of Safra's determinization construction to nondeterministic generalized Büchi automata. The extension leads to an exponential improvement (with respect to an approach that first translates the generalized Büchi automaton to a Büchi automaton) in that construction.

4.1.4 Compositional Synthesis A drawback of current theory of system synthesis is that it assumes that one gets a comprehensive set of temporal assertions as a starting point. This cannot be realistic in practice. A more realistic approach would be to assume an *evolving* formal specification: temporal assertions can be added, deleted, or modified. Accordingly, there is a need to develop compositional synthesis algorithms. Such algorithms can, for example, refine designs when provided with additional temporal properties.

In [20], we describe such an algorithm. Given a specification ψ, we first check its realizability. Suppose now that we get an additional specification ψ'. We can, of course, simply check the realizability of $\psi \wedge \psi'$ from scratch. Instead, we suggest to first check also the realizability of ψ'. We then show how, thanks to the simple structure of the NBT obtained in the Safraless approach, much of the work used in checking the realizability of ψ and ψ' in isolation can be reused in checking the realizability of $\psi \wedge \psi'$.

References

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 16th ICALP*, LNCS 372, pages 1–17, 1989.

[2] C. Schulte Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of Büchi automata. In *Proc. 10th ICIAA*, LNCS 3845, pages 262–272, 2005.

[3] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification logic. In *Proc. 8th TACAS*, LNCS 2280, pages 296–211, 2002.

[4] 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.

[5] A. Church. Logic, arithmetics, and automata. In *Proc. International Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.

[6] D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989.

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

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

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

[10] E. Friedgut, O. Kupferman, and M.Y. Vardi. Büchi complementation made tighter. In *Proc. 2nd ATVA*, LNCS 3299, pages 64–78, 2004.

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

[12] G. De 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, 1994.

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

[14] T.A. Henzinger and N. Piterman. Solving games without determinization, *Submitted*, 2006.

[15] 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.

[16] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International Editions, 1991.

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

[18] N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *Proc. 32nd FOCS*, pages 358–367, 1991.

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

[20] O. Kupferman, N. Piterman, and M.Y. Vardi. Safraless compositional synthesis. In *Proc. 18th CAV*, LNCS, 2006.

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

[22] O. Kupferman and M.Y. Vardi. From complementation to certification. In *Proc. 10th TACAS*, LNCS 2988, pages 591–606, 2004.

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

[24] O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th FOCS*, pages 531–540, 2005.

[25] 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.

[26] R.P. Kurshan. Complementing deterministic Büchi automata in polynomial time. *Journal of Computer and System Science*, 35:59–71, 1987.

[27] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.

[28] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *I& C*, 9:521–530, 1966.

[29] A. R. Meyer. Weak monadic second order theory of successor is not elementary recursive. In *Proc. ICALP*, LNCS 453, pages 132–154, 1975.

[30] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

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

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

[33] 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.

[34] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *25th LICS*, 2006.

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

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

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

[38] M.O. Rabin. Automata on infinite objects and Church's problem. *Amer. Mathematical Society*, 1972.

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

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

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

[42] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[43] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proc. 12th CAV*, LNCS 1855, pages 248–263, 2000.

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

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

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

[47] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 133–191, 1990.

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

[49] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, pages 332–344, 1986.

[50] 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.

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

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

[53] P. Wolper. Temporal logic can be more expressive. *I& C*, 56(1–2):72–99, 1983.

[54] Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *Proc. 33rd ICALP*, LNCS 4052, pages 589–600, 2006.