On Complementing Nondeterministic Büchi Automata

Sankar Gurumurthy[1], Orna Kupferman[2⋆],
Fabio Somenzi[1⋆⋆], and Moshe Y. Vardi[3⋆⋆⋆]

[1] University of Colorado at Boulder
[2] Hebrew University
[3] Rice University

Abstract. Several optimal algorithms have been proposed for the complementation of nondeterministic Büchi word automata. Due to the intricacy of the problem and the exponential blow-up that complementation involves, these algorithms have never been used in practice, even though an effective complementation construction would be of significant practical value. Recently, Kupferman and Vardi described a complementation algorithm that goes through weak alternating automata and that seems simpler than previous algorithms. We combine their algorithm with known and new minimization techniques. Our approach is based on optimizations of both the intermediate weak alternating automaton and the final nondeterministic automaton, and involves techniques of rank and height reductions, as well as direct and fair simulation.

1 Introduction

Efforts for developing simple complementation algorithms for nondeterministic Büchi automata started early in the 60s, motivated by decision problems of second order logics. In [5], Büchi suggested a complementation construction that involved a complicated combinatorial argument and 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. In [22], Sistla, Vardi, and Wolper suggested an improved construction, with $2^{O(n^2)}$ states. Only in [20], however, Safra introduced an optimal determinization construction, which also enabled a $2^{O(n \log n)}$ complementation construction, matching the known lower bound [18]. Another $2^{O(n \log n)}$ construction was suggested by Klarlund in [10], which circumvented the need for determinization. While being the heart of many complexity results in verification, the constructions in [20, 10] are complicated and difficult to program. We know of no implementation of Klarlund's algorithm, and the implementation of Safra's algorithm [24] has to cope with the involved structure of the states in the complementary automaton.

The lack of a simple implementation is not due to a lack of need. In the automata-theoretic approach to verification, we check correctness of a system with respect to a

⋆ Supported in part by NSF grant CCR-9988172.
⋆⋆ Supported in part by SRC contract 2002-TJ-920 and NSF grant CCR-99-71195.
⋆⋆⋆ Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, and EIA-0086264, and by a grant from the Intel Corporation.

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 [15, 25] usually proceeds by translating the negation of an LTL formula into a Büchi automaton. When properties are specified by ω-regular automata, one needs to complement the property automaton. Due to the lack of a simple complementation construction, the user is typically required to specify the property by deterministic Büchi automata [14] (it is easy to complement a deterministic automaton), or to supply the automaton for the negation of the property [9]. Similarly, specification formalisms like ETL [26], 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. [23, 8]). 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 would be of significant practical value.

In [12], Kupferman and Vardi describe a complementation procedure that is simpler than those in [20, 10]. The key idea of [12] is to go via *weak alternating automata*. In an alternating automaton [6], both existential and universal branching modes are allowed, and the transitions are given as Boolean formulas over the set of states. For example, a transition $\delta(q, \sigma) = q_1 \vee (q_2 \wedge q_3)$ means that when the automaton is in state q and it reads a letter σ, it should accept the suffix of the word either from state q_1 or from both states q_2 and q_3. Let α be the set of the automaton's accepting states. In a weak automaton, each strongly connected component of the graph induced by the transition function is either accepting (trivial, or contained in α) or rejecting (its intersection with α is empty). Since the strongly connected components are partially ordered, each path in the run eventually gets trapped in one of them. The run is accepting if all paths get trapped in accepting components. The *height* of a weak automaton is the maximal number of alternations between accepting and rejecting components in a path in the graph of the automaton, plus one.

The rich structure of alternating automata makes their complementation trivial—one only has to dualize the transition function and the acceptance condition. Removing alternation from Büchi automata involves a simple extension of the subset construction [19]. Unfortunately, by dualizing the given nondeterministic Büchi automaton, one gets a universal co-Büchi automaton, creating a gap in the construction. This gap is closed in [12], whose complementation construction consists of the following steps.

(1) Dualize the given nondeterministic Büchi automaton $\mathcal{B}$, and obtain a universal co-Büchi automaton $\mathcal{C}$ for the complement language. This step is trivial and involves no blow up.

[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) Translate $\mathcal{C}$ to an alternating weak automaton $\mathcal{W}$ accepting the same language. If $\mathcal{C}$ has n states, then $\mathcal{W}$ has $O(n^2)$ states.
(3) Translate $\mathcal{W}$ to a nondeterministic Büchi automaton $\mathcal{M}$. This step follows the exponential subset construction of [19]. The state space of $\mathcal{M}$ can be restricted to consistent subsets[2], making the overall blow up $2^{O(n \log n)}$ rather than $2^{O(n^2)}$.

In this paper we study and describe an arsenal of optimization techniques that can be applied in the last two steps. The techniques can be partitioned into the following classes.

Rank reduction. The translation in Step (2) is based on an analysis of the accepting runs of $\mathcal{C}$. Each vertex of the run is associated with a *rank* in the range $\{0, \ldots, 2n\}$. Like the progress measure of [10], the rank of a vertex indicates how easy it is to prove that all the paths that start at the vertex visit α only finitely often. The rank of a universal co-Büchi automaton $\mathcal{C}$ is the maximal rank of a vertex in an accepting run of $\mathcal{C}$.

If the state space of $\mathcal{C}$ is Q and its rank is k, the state space of $\mathcal{W}$ can be restricted to $Q \times \{0, \ldots, k\}$. Hence, finding and/or reducing the rank of $\mathcal{C}$ is desirable. We study ranks of languages, namely the minimal rank of a universal co-Büchi automaton that recognizes their complement. We show that, surprisingly, the rank of all ω-regular languages is 3 (a nice corollary, also proved in [16], is that all ω-regular languages can be recognized by an alternating weak automaton of height 3). Reducing the rank to 3, however, has a flavor of determinization, and involves an exponential blow-up in the state space. Accordingly, we prefer the approach of finding the rank k of $\mathcal{C}$. We show that the rank of $\mathcal{C}$ is bounded by $2(n - |\alpha|)$, and that there are automata for which this bound is tight. As suggested in [12], the rank is often smaller. We find the rank by checking for language equivalence between $\mathcal{W}$ and its restrictions to $Q \times \{0, \ldots, j\}$, for $j < 2(n - |\alpha|)$.

Minimization of $\mathcal{W}$. Once we found the rank k of $\mathcal{C}$ and restrict the state space of $\mathcal{W}$ accordingly, we minimize $\mathcal{W}$ further. The transition function of $\mathcal{W}$ as described in [12] is of size $|\delta|k^2$, where δ is the transition function of $\mathcal{C}$. It is suggested in [12] to simplify it and obtain a function of size $3|\delta|k$. We simplify it further to $2|\delta|k$. The simplification is based on simulation minimization we apply to $\mathcal{W}$, and which often reduces the state space and the transitions even more. Our simulation relation is similar to the alternating simulation of [2], extended to automata with acceptance conditions on the states (direct simulation) as well as an extension of it in which acceptance conditions are moved to the arcs. Finally, we reduce the height of $\mathcal{W}$ by repeatedly removing its minimal strongly connected component, as long as such a removal does not change its language.

Minimization of $\mathcal{M}$. Once $\mathcal{M}$ is produced by the subset construction, we apply further simplification techniques to it. The first is the fair simulation minimization of [8], and the second is similar to the height reduction described for $\mathcal{W}$, performed on the strongly connected components of $\mathcal{M}$. We note that the same reductions are applied also to the nondeterministic Büchi automaton $\mathcal{B}$ with which we start.

As shown in [18], complementation of a nondeterministic Büchi automaton with n states may involve a $2^{O(n \log n)}$ blow up. Accordingly, we measure the efficiency of

[2] We describe the consistency condition in Section 3.

our optimizations by the following two criteria: (1) we would like the result of complementing a nondeterministic Büchi automaton derived from an LTL formula to be comparable with what we get by negating the formula and then translating to a nondeterministic Büchi automaton. (2) we would like the result of complementing a nondeterministic Büchi automaton twice to be comparable with the original automaton. We demonstrate the effectiveness of our construction by examining several examples for which our construction produces the minimal nondeterministic Büchi automaton. We have implemented our procedure as an extension of the Wring translator from LTL to Büchi automata [23, 8], and our experimental results are reported in Section 7.

2 Preliminaries

Let $\mathcal{B}^+(Q)$ denote the set of positive Boolean formulas over Q. An *alternating automaton on infinite words* $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$ consists of a finite alphabet Σ, a finite set of states Q, an initial state $q_{in} \in Q$, a transition function $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$, and an acceptance condition $\alpha \subseteq Q$. For $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$ and $q \in Q$, let $\mathcal{A}^q = \langle \Sigma, Q, q, \delta, \alpha \rangle$. That is, $\mathcal{A}^q$ is obtained from $\mathcal{A}$ by making q the initial state.

A run of an alternating automaton $\mathcal{A}$ on a word $\sigma \in \Sigma^\omega$ is a Q-labeled tree $\langle T_r, r \rangle$, where T_r is a prefix-closed subset of $\mathbb{N}^*$, and $r : T_r \to Q$ is a labeling function. A run of $\mathcal{A}$ on $\sigma = \sigma_0, \sigma_1, \ldots$ satisfies the following conditions: (1) $r(\varepsilon) = q_{in}$. (2) For a tree node $t \in T_r$ such that $r(t) = q$ and $\delta(q, \sigma_i) = \beta$, there is a subset $Q_t \subseteq Q$ that satisfies β, and such that the successors of t are labeled by the elements of Q_t.

A run is accepting if all its infinite paths satisfy the acceptance condition. In a *Büchi* automaton, a path satisfies α if it intersects α infinitely often. In a *co-Büchi* automaton, a path satisfies α if it intersects it finitely many times. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on w. The words accepted by $\mathcal{A}$ form the language of $\mathcal{A}$, denoted by $L(\mathcal{A})$.

Complementation of an alternating automaton is accomplished by dualizing its transition function, and changing the acceptance condition from Büchi to co-Büchi or vice versa. Dualization consists of exchanging $\wedge$ with $\vee$, and true with false in δ.

A positive Boolean formula has a unique minimal DNF. Therefore $\delta(q, \sigma_i) \in \mathcal{B}^+(Q)$ identifies a set of sets of states $\Delta(q, \sigma_i) \subseteq 2^Q$. For instance, if $\delta(q_0, \sigma_0) = (q_1 \wedge (q_2 \vee q_3)) \vee (q_1 \wedge q_2 \wedge q_4)$, then $\Delta(q_0, \sigma_0) = \{\{q_1, q_2\}, \{q_1, q_3\}\}$. The Boolean formulas true and false translate into $\{\emptyset\}$ and $\emptyset$, respectively. The choice of $Q_t \subseteq Q$ required by the definition of run can always be restricted so that $Q_t \in \Delta(q, \sigma_i)$.

Nondeterministic automata are alternating automata in which each $C \in \Delta(q, l)$ is a singleton for every $q \in Q$ and $l \in \Sigma$. Universal automata are alternating automata in which $\Delta(q, l)$ is a singleton for every $q \in Q$ and $l \in \Sigma$. Deterministic automata are at the same time nondeterministic and universal.

A *maximal strongly connected component* (MSCC) of a directed graph is a maximal subgraph such that each node in the subgraph has a path to every node in the subgraph. A trivial MSCC contains one node and no arcs. We assume that all the trivial MSCCs of an automaton are contained in α. A *weak* alternating automaton is such that each MSCC of its transition graph is either disjoint from α or contained in it. From a weak alternating automaton with co-Büchi acceptance $\mathcal{A}$ one obtains a weak alternating automaton with Büchi acceptance $\mathcal{A}'$ such that $L(\mathcal{A}') = L(\mathcal{A})$ simply by taking $\alpha' = Q \setminus \alpha$.

We use three-letter abbreviations to designate types of automata: The first letter characterizes the transition structure and is one of "D" (deterministic), "N" (nondeterministic), "U" (universal), and "A" (alternating). The second letter identifies the acceptance condition and is one of "B" (Büchi), "C" (co-Büchi), and "W" (weak). Finally, the third letter designates the objects accepted by the automata; in this paper we are only concerned with "W" (infinite words). Hence, NBW designates a nondeterministic Büchi automaton, UCW designates a universal co-Büchi automaton, and AWW designates a weak alternating automaton, all on infinite words.

3 Ranks and Complementation

In this section we review the relevant technical details of [12]. Consider a UCW $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$ obtained by dualizing NBW $\mathcal{B}$, and a word w. Let $|Q| = n$. The run of $\mathcal{A}$ on w can be represented by a directed acyclic graph (DAG) $G_r = \langle V, E \rangle$, where

- $V \subseteq Q \times \mathbb{N}$ is such that $\langle q, l \rangle \in V$ iff there exists $x \in T_r$ with $|x| = l$ and $r(x) = q$. For example, $\langle q_{in}, 0 \rangle$ is the only vertex of G_r in $Q \times \{0\}$.
- $E \subseteq \bigcup_{l \geq 0} (Q \times \{l\}) \times (Q \times \{l+1\})$ is such that $E(\langle q, l \rangle, \langle q', l+1 \rangle)$ iff there exists $x \in T_r$ with $|x| = l$, $r(x) = q$, and $r(x \cdot c) = q'$ for some $c \in \mathbb{N}$.

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 $\langle q_0, l_0 \rangle, \ldots, \langle q_i, l_i \rangle$ of successive vertices such that $\langle q, l \rangle = \langle q_0, l_0 \rangle$, and $\langle q', l' \rangle = \langle q_i, l_i \rangle$. Finally, we say that a vertex $\langle q, l \rangle$ is an *α-vertex* iff $q \in \alpha$. It is easy to see that $\langle T_r, r \rangle$ is accepting iff all paths in G_r have only finitely many α-vertices.

Consider a (possibly finite) DAG $G \subseteq G_r$. We say that a vertex $\langle q, l \rangle$ is *finite* in G iff only finitely many vertices in G are reachable from $\langle q, l \rangle$. We say that a vertex $\langle q, i \rangle$ is *α-free* in G iff all the vertices in G that are reachable from $\langle q, l \rangle$ are not α-vertices. Finally, we say that the *width* of G is k if k is the maximal number for which there are infinitely many levels l such that there are k vertices of the form $\langle q, l \rangle$ in G. Note that the width of G_r is at most n. Given an accepting run DAG G_r, we define an infinite sequence $G_0 \supseteq G_1 \supseteq G_2 \supseteq \ldots$ of DAGs inductively as follows.

- $G_0 = G_r$.
- $G_{2i+1} = G_{2i} \setminus \{\langle q, l \rangle \mid \langle q, l \rangle$ 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 [12] 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.

Each vertex $\langle q, l \rangle$ in G_r 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}. Given a vertex $\langle q, l \rangle$, we define the *rank* of $\langle q, l \rangle$, denoted $rank(q, l)$, as follows.

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

For $k \in \mathbb{N}$, let $[k]$ denote the set $\{0, 1, \ldots, k\}$, and let $[k]^{odd}$ denote the set of odd members of $[k]$. By the above, the rank of every vertex in G_r is in $[2n]$. Recall that when the run is accepting, all the paths in G_r visit only finitely many α-vertices. Intuitively, $rank(q, l)$ hints at how difficult it is to prove that all the paths of G_r that visit the vertex $\langle q, l\rangle$ visit only finitely many α-vertices. Easiest to prove are vertices that are finite in G_0. Accordingly, they get the minimal rank 0. Then come vertices that are α-free in the graph G_1. These vertices get the rank 1. The process repeats until all vertices get some rank.

We say that an integer j is a *required rank* for a UCW $\mathcal{A}$ if there exists a word $w \in \mathcal{L}(\mathcal{A})$ such that some vertex in the run of $\mathcal{A}$ on w gets rank j. Then, the *rank of* $\mathcal{A}$ is the maximal rank required for $\mathcal{A}$. The annotation of runs with ranks is used in order to translate UCW into AWW:

Theorem 1. *Let $\mathcal{A}$ be a UCW with n states and rank k. There is an AWW $\mathcal{A}'$ with $n(k+1)$ states such that $L(\mathcal{A}') = L(\mathcal{A})$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha\rangle$. We define $\mathcal{A}' = \langle \Sigma, Q', q'_{in}, \delta', \alpha'\rangle$, where

- $Q' = Q \times [k]$. Intuitively, $\mathcal{A}'$ is in state $\langle q, i\rangle$, if it guesses that in the accepting run of $\mathcal{A}$ on w, the rank of $\langle q, l\rangle$ 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 $\langle q_{in}, 0\rangle$.
- We define δ' by means of a function $release : \mathcal{B}^+(Q) \times [k] \to \mathcal{B}^+(Q')$. Given a formula $\theta \in \mathcal{B}^+(Q)$, and a rank $i \in [k]$, the formula $release(\theta, i)$ is obtained from θ by replacing an atom q by the disjunction $\bigvee_{i' \leq i} \langle q, i'\rangle$. For example, $release(q_3 \wedge q_5, 2) = (\langle q_3, 2\rangle \vee \langle q_3, 1\rangle \vee \langle q_3, 0\rangle) \wedge (\langle q_5, 2\rangle \vee \langle q_5, 1\rangle \vee \langle q_5, 0\rangle)$.
  Now, $\delta' : Q' \times \Sigma \to \mathcal{B}^+(Q')$ is defined, for a state $\langle q, i\rangle \in Q'$ and $\sigma \in \Sigma$, as follows.
  $$\delta'(\langle q, i\rangle, \sigma) = \begin{cases} release(\delta(q, \sigma), i) & \text{If } q \notin \alpha \text{ or } i \text{ is even.} \\ \mathsf{false} & \text{If } q \in \alpha \text{ and } i \text{ is odd.} \end{cases}$$
  That is, if the current guessed rank is i then, by employing $release$, the run can move to its successors at any rank that is smaller than 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 a path should be odd.

To see that the automaton $\mathcal{A}'$ is weak, note that each set $Q \times \{i\}$ is a collection of strongly connected components that agree on their classification as accepting or rejecting. Indeed, membership in α' depends on the parity of i, and the transitions in δ' are such that from a state in $Q \times \{i\}$ the automaton $\mathcal{A}'$ proceeds only to states in $Q \times \{j\}$, for $j \leq i$. $\square$

Once we know how to translate UCW to AWW, complementation is reduced to removal of alternation from ABW (recall that AWW are a special case of ABW). In [19], Miyano and Hayashi describe such a translation. We present (a simplified version of) their translation in Theorem 2 below.

Theorem 2. [19] *Let $\mathcal{A}$ be an alternating Büchi automaton. There is a nondeterministic Büchi automaton $\mathcal{A}'$, with exponentially many states, such that $L(\mathcal{A}') = L(\mathcal{A})$.*

Proof. The automaton $\mathcal{A}'$ guesses a run of $\mathcal{A}$. At a given point of a run of $\mathcal{A}'$, it keeps in its memory a whole level of the run tree of $\mathcal{A}$. As it reads the next input letter, it guesses the next level of the run tree of $\mathcal{A}$. 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 $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$. Then $\mathcal{A}' = \langle \Sigma, 2^Q \times 2^Q, \langle \{q_{in}\}, \emptyset \rangle, \delta', 2^Q \times \{\emptyset\} \rangle$, where 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 $\delta'(\langle S, O \rangle, \sigma) = \{\langle S', O' \setminus \alpha \rangle \mid S'$ satisfies $\bigwedge_{q \in S} \delta(q, \sigma), O' \subseteq S'$, and O' satisfies $\bigwedge_{q \in O} \delta(q, \sigma)\}$.
– If $O = \emptyset$, then $\delta'(\langle S, O \rangle, \sigma) = \{\langle S', S' \setminus \alpha \rangle \mid S'$ satisfies $\bigwedge_{q \in S} \delta(q, \sigma)\}$.

$\square$

For an NBW $\mathcal{B}$, the *rank* of $\mathcal{B}$ is the rank of its dual UCW. Complementing an NBW $\mathcal{B}$ with n states and rank k, its dual UCW has n states and rank k as well, the AWW $\mathcal{W}$ constructed in Theorem 1 has $O(nk)$ states, and the final NBW $\mathcal{M}$ constructed in Theorem 2 has $2^{O(nk)}$ states. By [18, 20], however, an optimal complementation construction for nondeterministic Büchi automata results in an automaton with $2^{O(n \log n)}$ states, which may be smaller. Let $\mathcal{B} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$. Consider a state $\langle S, O \rangle$ of $\mathcal{M}$. Each of the sets S and O is a subset of $Q \times [k]$. We say that $P \subseteq Q \times [k]$ is *consistent* iff for every two states $\langle q, i \rangle$ and $\langle q', i' \rangle$ in P, if $q = q'$ then $i = i'$. It is shown in [12] that restricting the state space of $\mathcal{M}$ to pairs $\langle S, O \rangle$ for which S is a consistent subset of $Q \times [k]$ is allowable; that is, the resulting $\mathcal{M}$ still complements $\mathcal{B}$. Since there are $2^{O(n \log k)}$ consistent subsets of $Q \times [k]$, we have the following.

Theorem 3. *Let $\mathcal{A}$ be an NBW with n states and rank k. There is an NBW $\mathcal{A}'$ with $2^{O(n \log k)}$ states such that $L(\mathcal{A}') = comp(L(\mathcal{A}))$.*

4 Ranks of Automata and Languages

Consider a UCW $\mathcal{A}$ with n states and a word $w \in \Sigma^\omega$. Let $G_0, G_1, \ldots, G_{2n+1}$ be the sequence of DAGs constructed in [12] for the run of $\mathcal{A}$ on w. Recall that the transition from G_{2i+1} to G_{2i+2} involves a removal of an infinite path from G_{2i+1}, which is why the width of G_{2i} is at most $n - i$. As noted to us by Doron Bustan, all the vertices in the removed path are not α-vertices. Hence, one could argue that the $n - i$ bound on the width of G_{2i} holds also for a tighter definition of width: let the α-less width of G_i be the maximal number k for which there are infinitely many levels l such that there are k vertices not in α of the form $\langle q, l \rangle$. With this tighter definition, the α-less width of G_0 is bounded by $n - |\alpha|$, implying that the α-less width of G_{2i} is at most $n - (|\alpha| + i)$. In particular, the α-less width of $G_{2(n-|\alpha|)}$ is at most 0. Hence $G_{2(n-|\alpha|)}$ has only finitely many vertices that are not α-vertices. Since G_0 is accepting, then, by König's Lemma, $G_{2(n-|\alpha|)}$ also has only finitely many α vertices. It follows that $G_{2(n-|\alpha|)}$ is finite, implying that all vertices get ranks in $0, \ldots, 2(n - |\alpha|)$.

In practice, the transition from G_{2i} to G_{2i+2} often reduces the width by more than one. One may wonder whether it is possible to tighten the analysis above even more in order to show that a rank of $2(n - |\alpha|)$ is never required. Recall that an integer j is a required rank for $\mathcal{A}$ if there exists a word $w \in \mathcal{L}(\mathcal{A})$ such that some vertex in the run of $\mathcal{A}$ on w gets rank j. Equivalently, the α-less width of G_j (with G_0 being the run DAG of $\mathcal{A}$ on w) is strictly larger than 0. As follows from Theorems 1 and 3, the rank of $\mathcal{A}$ plays an important role in the sizes of equivalent AWW and NBW for it. It is shown in [12] that the problem of finding the rank of a UCW $\mathcal{A}$ is PSPACE-complete. By the above, the rank of $\mathcal{A}$ is at most $2(n - |\alpha|)$. By the following theorem, there are cases in which this bound is tight.

Theorem 4. *There is a family $\mathcal{A}_1, \mathcal{A}_2, \ldots$ of UCW such that $\mathcal{A}_n$ has $n+1$ states, acceptance set of size* 1*, and rank $2n$.*

We now turn to study ranks of ω-regular languages. For an ω-regular language $\mathcal{L}$, we say that the rank of $\mathcal{L}$ is k iff there is a UCW of rank at most k for $comp(\mathcal{L})$. It is tempting to think that ranks induce an infinite hierarchy $\mathcal{R}_0 \subset \mathcal{R}_1 \subset \cdots$ of languages, with $\mathcal{R}_i$ containing all languages of rank i. We show that the hierarchy collapses at $\mathcal{R}_3$ (that is, all ω-regular languages have rank at most 3) and characterize its four levels. For a definition of safety and co-safety languages, see [1, 21].

Theorem 5. *$\mathcal{R}_3 = \omega$-regular languages, $\mathcal{R}_2 =$ DBW, $\mathcal{R}_1 =$ co-safety languages, and $\mathcal{R}_0 =$ safety languages.*

The hierarchy induced by ranks is closely related to a hierarchy induced by heights of AWW. Intuitively, the height of an AWW is the number of accepting and rejecting layers it has. Formally, the *height* of an AWW $\mathcal{A}$ is the number of alternations between accepting and rejecting components in the graph of $\mathcal{A}$, plus one, where the constants true and false are counted as accepting and rejecting components, respectively. For an integer k, let AWW[k] denote the set of AWW of height at most k, or the ω-regular languages accepted by such automata. Theorem 5 implies Theorem 6 below, which was proved first in [16]. Note that Theorem 5 is stronger than Theorem 6 and does not follow from it.

Theorem 6. AWW[3] $= \omega$*-regular languages,* AWW[2] $=$ *DBW* $\cup$ *DCW, and* AWW[1] $=$ *safety or co-safety languages.*

The results in this section imply that procedures for rank reduction that modify the given UCW are much stronger than those that calculate its rank. On the other hand, the reduction of the rank to 3 involves determinization, which we are trying to avoid, and which may cause an exponential blow-up. In view of this trade-off between the size of UCW and their ranks, our efforts focus on calculating the rank of the given UCW, rather than on modifying it.

5 Simplifying Alternating Büchi Automata

The construction of Theorem 2 may cause an exponential blow-up. Hence, before applying it, we try to simplify the AWW $\mathcal{W}$ in three ways: by simulation minimization, by computing the rank of the UCW $\mathcal{C}$, and by removing redundant MSCCs.

5.1 Simulation Minimization

We recall that for an ABW $\Delta(q, l)$ is a set of sets. Each member of $\Delta(q, l)$ is a conjunction of states. We define simulation between alternating automata in terms of a game as in [2]. Let $\mathcal{A}_A = \langle \Sigma, Q_A, q_{iA}, \delta_A, \alpha_A \rangle$ and $\mathcal{A}_P = \langle \Sigma, Q_P, q_{iP}, \delta_P, \alpha_P \rangle$ be two ABWs; automaton $\mathcal{A}_P$ simulates automaton $\mathcal{A}_A$ if, given players P and A, P has a winning strategy for the following game. The positions of the game are the elements of $Q_A \times Q_P$; the initial position is (q_{iA}, q_{iP}), and the possible successors of a position (s_A, s_P) are all pairs (t_A, t_P) obtained by application of the following rule:

– A chooses a letter $l \in \Sigma$ and a set of states $C_A \in \Delta_A(s_A, l)$;
– P chooses a set of states $C_P \in \Delta_P(s_P, l)$;
– A chooses $t_P \in C_P$;
– P chooses $t_A \in C_A$.

A player who has to choose from an empty set loses. If this never happens, the play is infinite. The winner of an infinite play depends on whether one considers *direct* simulation or *fair* simulation. For direct simulation, A wins iff for some position (s_A, s_P) encountered, $s_A \in \alpha_A$ and $s_P \notin \alpha_P$. For fair simulation, A wins iff there are infinitely many positions such that $s_A \in \alpha_A$, but only finitely many positions such that $s_P \in \alpha_P$. P wins if A does not. As in the case of NBWs, direct simulation implies fair simulation, and fair simulation implies language containment; the converse is not true [2].

Theorem 7. *Let $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$ and $\mathcal{A}' = \langle \Sigma, Q', q'_{in}, \delta', \alpha' \rangle$ be two ABWs. If q_{in} direct simulates q'_{in}, then q_{in} fair simulates q'_{in}. If q_{in} fair simulates q'_{in}, then $L(\mathcal{A}) \supseteq L(\mathcal{A}')$.*

If two states q_1 and q_2 are such that each simulates the other, we say that q_1 and q_2 are *simulation equivalent*. Two ABWs are simulation equivalent if their initial states are. Of particular interest to us is the case in which the two automata are $\mathcal{A}^{q_1}$ and $\mathcal{A}^{q_2}$ for $q_1, q_2 \in Q$; that is, we are interested in the simulation relation on the states of ABW $\mathcal{A}$. The "layered" structure of the AWW $\mathcal{W}$ implies the existence of a nontrivial simulation relation.

Theorem 8. *Let $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$ be a UCW with rank k; let $\mathcal{A}'$ be the equivalent AWW of Theorem 1. Then, for every $\langle q, j \rangle \in Q \times \{0, \ldots, k\}$ and $i \in \{0, \ldots, j\}$, if j is even or $q \notin \alpha$, then $\langle q, j \rangle$ fair simulates $\langle q, i \rangle$ in $\mathcal{A}'$. If in addition j is odd or i is even, then $\langle q, j \rangle$ direct simulates $\langle q, i \rangle$.*

The simulation of Theorem 8 allows us to improve on [12, Remark 4.2] and reduce the size of the transition relation of $\mathcal{W}$ from $3|\delta|k$ to $2|\delta|k$, where δ is the transition function and k is the rank of the UCW $\mathcal{C}$.

Theorem 9. *If in Theorem 1, $\mathsf{release}(\theta, i)$ is redefined so that an atom q is replaced by $\langle q, i \rangle \vee \langle q, i-1 \rangle$ if $i > 0$, and by $\langle q, 0 \rangle$ for $i = 0$, then $L(\mathcal{A}') = L(\mathcal{A})$.*

In general, simulations between states of an ABW can be used to merge states (in case of simulation equivalence), remove transitions, or simplify transitions.[3] The last

[3] This is in contrast to [7], which only considers simulation equivalence quotients. Besides, its model of alternating automata with existential and universal states makes even direct simulation unsafe for minimization.

use is specific to alternating automata: Suppose $C \in \Delta(q_i, \sigma_j)$ contains two states in direct simulation relation. Then, the simulating one can be removed because acceptance from the simulated state guarantees acceptance from the simulating one.

Theorem 10. *Let $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$ be an ABW. Let q_1 and q_2 be two states in Q such that q_2 direct simulates q_1. Suppose $\{q_1, q_2\} \subseteq C \in \Delta(q, l)$, for some $q \in Q$ and $l \in \Sigma$. Then the automaton $\mathcal{A}'$ obtained from $\mathcal{A}$ by replacing C in $\Delta(q, l)$ with $C' = C \setminus \{q_2\}$ is direct simulation equivalent to $\mathcal{A}$.*

Theorem 11. *Let $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$ be an ABW. Let $C_1, C_2 \in \Delta(q, l)$, for some $q \in Q$, $l \in \Sigma$. Suppose that $C_1 \neq C_2$, and that $\forall q_1 \in C_1$, $\exists q_2 \in C_2$ such that q_1 direct simulates q_2. Then the automaton $\mathcal{A}'$ obtained from $\mathcal{A}$ by replacing $\Delta(q, l)$ with $\Delta'(q, l) = \Delta(q, l) \setminus \{C_2\}$ is direct simulation equivalent to $\mathcal{A}$.*

Two simulation equivalent states q_1 and q_2 are merged by the following steps: (1) for every letter l, $\delta(q_1, l)$ is replaced by $\delta(q_1, l) \vee \delta(q_2, l)$; (2) q_2 is replaced by q_1 throughout δ; (3) q_1 is made initial if q_2 is; (4) q_2 is dropped.

Corollary 1. *Let $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$ be an ABW. If two states $q_1, q_2 \in Q$ are direct simulation equivalent, the automaton obtained by merging q_1 and q_2 is simulation equivalent to $\mathcal{A}$.*

The computation of the direct simulation relation is based on the following observation [2]. Let S be a simulation relation on the states of an ABW over alphabet Σ. Then $(u, v) \in S$ implies

$$\forall l \in \Sigma \,.\, \forall C \in \Delta(u, l) \,.\, \exists C' \in \Delta(v, l) \,.\, \forall v' \in C' \,.\, \exists u' \in C \,.(u', v') \in S \ .$$

We can therefore compute the direct simulation relation as a greatest fixpoint by starting with all the pairs of states (u, v) such that acceptance of u implies acceptance of v, and removing pairs that violate the condition above.

5.2 Simulation with Accepting Arcs

The definition of direct simulation given in Section 5.1 assumes that $u \in \alpha$ implies $v \in \alpha$. However, we may compute a larger relation by considering the acceptance conditions to be on the arcs. Let every set of states $C \in \Delta(q, l)$ be a *transition* out of $q \in Q$ enabled by $l \in \Sigma$. An *arc* of transition C is the pair (q, q'), for some state $q' \in C$. An arc (q, q') is accepting if $q' \in \alpha$. We can modify the definition of direct simulation as follows. Player A wins an infinite play if for some position (s_A, s_P), the arc (s_A, t_A) of C_A is accepting, but the arc (s_P, t_P) is not. Player P wins if A does not.

This approach may lead to simplifications not allowed by the original definition of direct simulation. However, Theorems 10 and 11 do not hold when acceptance conditions are moved to the arcs. Consider an AWW with $\Sigma = \{0\}$, $Q = \{a, b\}$, $q_{in} = a$, $\delta(a, 0) = a \wedge b$, $\delta(b, 0) = b$, and $\alpha = \{a\}$. Here b direct simulates a when acceptance is on the arcs. In this case the only accepting arc is the self-loop on a. However, $\delta(a, 0)$ cannot be simplified to a lest the language changes from empty to Σ^ω. To obviate this problem, while computing the direct simulation relation with accepting arcs, we mark all the arcs that are used to justify the relation itself. We then allow simplification of a transition according to Theorem 10 only if the arcs to be removed are not marked.

5.3 Simplification Based on Language Containment

Theorem 8 gives conditions under which $\langle q, j\rangle$ simulates $\langle q, i\rangle$ for $j > i$. However, no such general result can be proved for $j < i$. To determine the rank of the UCW $\mathcal{C}$ obtained by dualization of the given NBW $\mathcal{B}$, and hence the required height of the AWW $\mathcal{W}$, we resort to a language containment check. Specifically, since the rank is bounded by $2(n - |\alpha|)$, we apply the construction of Theorem 1 with $k = 2(n - |\alpha|)$ to build AWW $\mathcal{W}'$ such that $L(\mathcal{W}') = L(\mathcal{C})$. The construction of Theorem 2 applied to $\mathcal{W}'$ yields $\mathcal{M}'$.

To check whether $k \in \{0, 2, \dots, 2(n - |\alpha| - 1)\}$ is the rank of $\mathcal{C}$, we restrict $\mathcal{W}'$ to $Q \times \{0, \dots, k\}$, make $\langle q_{in}, k\rangle$ initial, and call the result $\mathcal{W}''$. We then obtain an AWW $\mathcal{D}$ for $comp(L(\mathcal{W}''))$ by dualization of $\mathcal{W}''$, and apply Theorem 2 to it to produce $\mathcal{M}''$. Since we know that $L(\mathcal{W}'') \subseteq L(\mathcal{W}')$, if the intersection of $\mathcal{M}'$ and $\mathcal{M}''$ is empty, then k is an upper bound on the rank of $\mathcal{C}$. If one tries the possible values of k in increasing order, the first time the intersection is empty, k is the rank of $\mathcal{C}$, and $\mathcal{W} = \mathcal{W}''$. It is important to note that the restriction to consistent subsets is allowed when converting $\mathcal{W}$ to NBW, but is not allowed when converting $\mathcal{D}$. This makes the determination of the rank a particularly expensive operation. To partially offset this cost, simulation minimization is always applied to $\mathcal{D}$ before the subset construction.

The language-containment approach can be used to further simplify $\mathcal{W}$. Specifically, we try to remove an MSCC from $\mathcal{W}$, and all the transitions with at least one destination state in the chosen MSCC. This guarantees that the language of the resulting automaton is contained in the language of the original one. A single language containment check then suffices to check whether the language remains the same. The MSCCs are examined in topological order from terminal to initial. If the language does not change, the removal of the MSCC is greedily accepted. We refer to this process as *pruning the AWW*.

5.4 Simplification Procedure

If the NBW $\mathcal{B}$ is weak, so is the UCW $\mathcal{C}$. Hence, the construction of Theorem 1 is not required, because a UWW is a special case of AWW. Since $\mathcal{B}$ has been minimized, no further simplification of $\mathcal{W} = \mathcal{C}$ is attempted. Testing this special case avoids the potentially expensive simplification of $\mathcal{W}$ and makes complementation of NWB efficient. This is practically relevant because many natural specifications induce weak automata [11, 4]. (In [17] it is shown that the intersection of ACTL and LTL is UCW[1], which is included in UWW.)

If $\mathcal{C}$ is not weak, first its rank is determined, and $\mathcal{W}$ is built accordingly, simplifying transitions as discussed in Section 5.2, and applying Corollary 1, and Theorems 10–11. The states with index 0 are included only if $\mathcal{C}$ has at least one transition equal to `true`. (Otherwise, no accepting path can visit them.) Pruning based on language containment (see Section 5.3) is then performed as the last optimization of the AWW before computing the NBW equivalent to $\mathcal{W}$.

If $\mathcal{B}$ is a DBW that is not weak, the resulting AWW is an NWW, and the subset construction does not change it. In such a case, our algorithm behaves like the one of [13]. In some cases, simplification of an AWW also produces an NWW, making the subset construction redundant.

6 Simplification of Nondeterministic Büchi Automata

The complementation algorithm starts and ends with two NBWs, $\mathcal{B}$ and $\mathcal{M}$. It is important to minimize both. For $\mathcal{B}$, every simplification is likely to alleviate the burden for the successive stages of the computation. For $\mathcal{M}$, minimization recovers inefficiencies due, in particular, to the subset construction. In this section we describe how this minimization is carried out. Two procedures are applied to the NBWs $\mathcal{B}$ and $\mathcal{M}$. One is fair simulation minimization [8]. The other is a pruning technique akin to the one described in Section 5.3, but based on checking direct simulation, rather than language containment. Its objective is to reduce the height of the NBW, and it works as follows.

1. Mark all states simulated by an initial state as initial.
2. Process MSCCs that intersect α in topological order from sources to sinks.
3. Remove arcs out of MSCC and compute simulation relation for result.
4. If initial states with path to MSCC are simulated by initial states without a path to the MSCC, make all the states in the MSCC non-accepting.
5. Minimize automaton if some MSCCs were made non-accepting; otherwise, make non-initial all states that were made initial in the first step.

We rely on the fact that direct simulation minimization removes from the initial states a state that is simulated by another initial state. Hence, we end up with only one initial state if we started with one.

7 Experimental Results

We have implemented the complementation algorithm presented in this paper as an extension of the Wring translator from LTL to Büchi automata [23, 8], which is written in Perl. All experiments were run on an IBM IntelliStation running Linux with a 1.7 GHz Pentium 4 CPU and 1 GB of RAM. Complementation experiments were allotted 1 minute if the input NBW was weak, and 2 minutes if it was not.

We use a set of 1000 LTL formulas distributed with Wring to evaluate the complementation algorithm. Two types of comparisons were conducted. In the first, each formula is converted by Wring into a Büchi automaton whose complement is then computed if it has exactly one fairness constraint. (Wring produces generalized Büchi automata, which may have 0, 1, or more sets of accepting states. Our implementation of the complementation algorithm only deals with one set of accepting states.) The complement is compared to the automaton obtained by translating the negation of the LTL formula. In the second comparison, the automaton obtained from an LTL formula is compared to the complement of its complement. Table 1 summarizes our results with regard to the quality of the automata produced by the complementation algorithm. For the two experiments, the table reports the ratios of total numbers of states and transitions produced by the complementation procedure and those in the reference automata.

Several steps in the translation from LTL to automaton are order dependent. Since Wring's data structures heavily rely on hash tables, even minimal differences in two runs like the addition of a diagnostic print command may cause some differences in the

Table 1. Our complementation procedure produces small automata

experiment	states	trans.
negation	1.09	1.26
double complementation	1.13	2.23

results. Hence, the number of automata with one set of accepting states presents small fluctuations in the various experiments. The same applies to most quantities we report.

Table 2 compares variants of the complementation algorithm ranging from the basic procedure presented in [12] (*base*) to the procedure that implements all the improvements described in this paper (*all*). Table 3 defines all variants in terms for their features, and Table 4 summarizes each feature used to define the methods and refers to the section of this paper that discusses it.

The first column of Table 2 designates the algorithm variant. Columns *weak* and *timeout weak* report the number of automata from those with one accepting set that were found to be weak and how many of those timed out. Columns *strong* and *timeout strong* do the same for the automata that were not weak. The next column gives the total CPU time in seconds. Columns 7 and 8 give the average number of states and transitions in $\mathcal{M}$ for the cases that completed. For comparison, the average numbers of states and transitions of the input automaton $\mathcal{B}$ are 6.04 and 12.23, respectively. The last two columns report the average ratio between the size of $\mathcal{M}$ before and after optimization ($\mathcal{M}$ *opt. ratio*), and the total number of states of the AWWs.

A few observations can be made about the data in Table 2. First, checking the input automaton $\mathcal{B}$ for weakness is a simple way to dramatically improve performance. However, method w+, that adds this simple check to the base approach, can only complete 10 automata that are not weak: Though there seems to remain considerable room for improvement in the complementation of automata that are not weak, the optimizations presented in this paper triple the number of successes.

Comparing the average sizes of the automata obtained with the several variants is hindered by the fact that the largest automata tend to cause the most timeouts. Comparing variants that produce about the same number of timeouts, however, shows that more optimization tends to produce smaller automata. It is also instructive to examine the effects of optimization of the NBW $\mathcal{M}$ produced by the subset construction of Theorem 2. The variants that skip direct simulation minimization of the AWW $\mathcal{W}$ have higher $\mathcal{M}$ opt. ratios because the final optimization has to make up for the "sloppiness" of the preceding stage. While fair simulation minimization of $\mathcal{M}$ discharges its duties well, minimization of $\mathcal{W}$ leads to a more robust solution.

Acknowledgment. Doron Bustan called our attention to the improved bound on the rank of a UCW.

References

[1] B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.

Table 2. Experimental results

method	weak	timeouts weak	strong	timeouts strong	time	states	trans.	$\mathcal{M}$ opt. ratio	$\mathcal{W}$ states
base	406	215	67	56	47303	4.08	7.05	6.03	2901
+w	404	4	70	60	9556	5.96	14.03	31.82	4636
+t9	405	4	69	49	7672	6.07	13.67	60.22	2495
+ds	405	4	68	53	10233	5.96	13.36	2.11	2907
+lc	405	3	69	59	9240	6.02	13.52	53.05	2309
–lc	405	4	68	38	6263	6.48	14.93	49.12	3536
–hr	405	3	68	39	6129	6.38	14.71	1.94	3603
–arc	404	4	69	53	6267	5.95	13.36	1.65	2456
all	406	3	68	39	6568	6.02	13.83	6.82	2470

Table 3. Definition of methods compared in Table 2

method	$\mathcal{B}$ sim	weak test	Thm. 9	$\mathcal{C}$ bound	$\mathcal{C}$ rank	$\mathcal{W}$ arc	$\mathcal{W}$ sim	$\mathcal{W}$ lc	$\mathcal{M}$ hr	$\mathcal{M}$ sim
base	✓			✓	✓				✓	✓
+w	✓	✓		✓	✓				✓	✓
+t9	✓	✓	✓	✓	✓				✓	✓
+ds	✓	✓		✓	✓	✓	✓		✓	✓
+lc	✓	✓		✓	✓			✓	✓	✓
–lc	✓	✓	✓	✓		✓	✓		✓	✓
–hr	✓	✓	✓	✓		✓	✓			✓
–arc	✓	✓	✓	✓	✓		✓	✓	✓	✓
all	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 4. Feature description

feature	description	section
$\mathcal{B}$ sim	fair simulation minimization of $\mathcal{B}$	6
weak test	simplified treatment for weak $\mathcal{B}$	5.4
Thm. 9	reduce the number of transitions of $\mathcal{W}$	5.1
$\mathcal{C}$ bound	use of $2(n - \lvert\alpha\rvert)$ as bound for the rank of $\mathcal{C}$	4
$\mathcal{C}$ rank	exact computation of the rank of $\mathcal{C}$	5.3
$\mathcal{W}$ arc	simulation minimization of $\mathcal{W}$ with accepting arcs	5.2
$\mathcal{W}$ sim	direct simulation minimization of $\mathcal{W}$	5.1
$\mathcal{W}$ lc	removal of MSCCs by language containment	5.3
$\mathcal{M}$ hr	height reduction of $\mathcal{M}$	6
$\mathcal{M}$ sim	fair simulation minimization of $\mathcal{M}$	6

[2] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. *Concurrency Theory*, pages 163–178, 1998. LNCS 1466.

[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 language. *TACAS*, pages 296–311, 2002. LNCS 2280.

[4] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of

linear time logic properties. *CAV*, pages 222–235, 1999. LNCS 1633.
[5] J. R. Büchi. On a decision method in restricted second order arithmetic. *1960 International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.
[6] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *JACM*, 28(1):114–133, 1981.
[7] C. Fritz and T. Wilke. State space reductions for alternating Büchi automata. *FSTTCS*, pages 157–168, 2002. LNCS 2556.
[8] S. Gurumurthy, R. Bloem, and F. Somenzi. Fair simulation minimization. *CAV*, pages 610–623, 2002. LNCS 2404.
[9] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.
[10] N. Klarlund. Progress measures for complementation of ω-automata with application to temporal logic. *FOCS*, pages 358–367, 1991.
[11] O. Kupferman and M. Y. Vardi. Relating linear and branching model checking. *IFIP PROCOMET*, pages 304–326, 1998.
[12] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM TOCL*, 2(3):408–429, 2001.
[13] R. P. Kurshan. Complementing deterministic Büchi automata in polynomial time. *Journal of Computer and System Sciences*, 35:59–71, 1987.
[14] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, 1994.
[15] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. *POPL*, pages 97–107, 1985.
[16] C. Löding and W. Thomas. Alternating automata and logics over infinite words. *TCS*, pages 521–535, 2000. LNCS 1872.
[17] M. Maidl. The common fragment of CTL and LTL. *FOCS*, pages 643–652, 2000.
[18] M. Michel. Complementation is more difficult with automata on infinite words. Manuscript, CNET, Paris, 1988.
[19] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *TCS*, 32:321–330, 1984.
[20] S. Safra. On the complexity of ω-automata. *FOCS*, pages 319–327, 1988.
[21] A. P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects in Computing*, 6:495–511, 1994.
[22] A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *TCS*, 49:217–237, 1987.
[23] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. *CAV*, pages 248–263, 2000. LNCS 1855.
[24] S. Tasiran, R. Hojati, and R. K. Brayton. Language containment using non-deterministic omega-automata. *CHARME*, pages 261–277, 1995. LNCS 987.
[25] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *LICS*, pages 322–331, 1986.
[26] P. Wolper. Temporal logic can be more expressive. *I&C*, 56(1-2):72–99, 1983.