Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 345 (2005) 83 – 100
Theoretical Computer Science
www.elsevier.com/locate/tcs

From complementation to certification

Orna Kupferman[a,*,1], Moshe Y. Vardi[b,2]

[a] *School of Engineering and Computer Science, Hebrew University, Jerusalem 91904, Israel*
[b] *Department of Computer Science, Rice University, Houston, TX 77251-1892, USA*

Abstract

In the automata-theoretic approach to model checking we check the emptiness of the product of a system S with an automaton $\mathscr{A}_{\neg\psi}$ for the complemented specification. This gives rise to two automata-theoretic problems: *complementation* of word automata, which is used in order to generate $\mathscr{A}_{\neg\psi}$, and the *emptiness* problem, to which model checking is reduced. Both problems have numerous other applications, and have been extensively studied for nondeterministic Büchi word automata (NBW). Nondeterministic *generalized* Büchi word automata (NGBW) have become popular in specification and verification and are now used in applications traditionally assigned to NBW. This is due to their richer acceptance condition, which leads to automata with fewer states and a simpler underlying structure.

In this paper we analyze runs of NGBW and use the analysis in order to describe a new complementation construction and a symbolic emptiness algorithm for NGBW. The complementation construction exponentially improves the best known construction for NGBW and is easy to implement. The emptiness algorithm is almost identical to a known variant of the Emerson–Lei algorithm, and our contribution is the strong relation we draw between the complementation construction and the emptiness algorithm—both naturally follow from the analysis of the runs, which easily implies their correctness. This relation leads to a new *certified* model-checking procedure, where a positive answer to the model-checking query is accompanied by a certificate whose correctness can be checked by

*Corresponding author.
E-mail addresses: orna@cs.huji.ac.il (O. Kupferman), vardi@cs.rice.edu (M.Y. Vardi)
URLs: http://www.cs.huji.ac.il/∼orna, http://www.cs.rice.edu/∼vardi.
[1] Supported in part by BSF Grant 9800096, and by a grant from Minerva.
[2] Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, and IIS-9978135, by BSF Grant 9800096, and by a Grant from the Intel Corporation.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.07.021

methods independent of the model checker. Unlike certificates generated in previous works on certified model checking, our analysis enables us to generate a certificate that can be checked automatically and symbolically.

1. Introduction

In *model checking*, we check whether all the computations of a given system S satisfy a specification ψ. The system is usually given as a labeled state-transition graph and ψ is either a formula in LTL or a word automaton. In the automata-theoretic approach to model checking [20,37], one constructs an automaton $\mathcal{A}_{\neg\psi}$ for the negation of ψ and takes its product with S. The system S is correct with respect to ψ if this product is empty.

When ψ is given as an LTL formula, the construction of $\mathcal{A}_{\neg\psi}$ is relatively straightforward, we first negate ψ and then apply on $\neg\psi$ one of the many known translations of LTL formulas to word automata (cf. [37,10,35]). When the specification is given as an automaton $\mathcal{A}_{\psi}$, the task is harder and one needs to complement the automaton. The product of $\mathcal{A}_{\neg\psi}$ and S can be viewed as a word automaton $\mathcal{A}_{S\times\neg\psi}$, and it is empty iff no computation of S violates ψ. Thus, the model-checking problem gives rise to two automata-theoretic problems: *complementation* of word automata, which is used in order to generate $\mathcal{A}_{\neg\psi}$ from $\mathcal{A}_{\psi}$, and the *emptiness* problem, to which model checking is reduced. Both problems have numerous other applications. First, refinement and optimization techniques that are based on language containment rather than simulation involve complementation and emptiness [19]. In addition, complementation is used in specification formalisms like ETL [38,37], which have automata within the logic, and emptiness is used for satisfiability, planning and synthesis [4,11,23].

Complementation and emptiness have been extensively studied for *nondeterministic Büchi word automata* (NBW, for short). The Büchi acceptance condition consists of a subset F of the state space, and a run of the automaton is accepting iff it visits F infinitely often. Consider an NBW $\mathcal{A}$ with n states. In [4], Büchi described a doubly-exponential complementation construction, which was improved in [34] to a construction with $2O(n2)$ states. Only in [32], Safra introduced an asymptotically optimal determinization construction, which also enabled a $2O(n \log n)$ complementation construction, matching the known lower bound [25]. Another $2O(n \log n)$ construction was suggested in [16], which circumvents the need for determinization. The optimal constructions in [32,16] are complicated, making their implementation very difficult [36]. In [17], we suggested an optimal complementation construction that is based on alternating automata. This construction is considerably simpler, making it the first construction to be implemented [24,12]. [3] The emptiness problem for NBW can be easily solved in linear time and NLOGSPACE [37]. The easy

[3] In [24], the theorem prover Isabelle/HOL is used in order to prove that the ω-regular languages are closed under complementation. This is done by defining, giving an NBW $\mathcal{A}$, the complementing automaton as constructed in [17], and proving that it indeed complements $\mathcal{A}$.

algorithms, however, are based on depth-first search, and cannot be implemented symbolically, which is very desirable in practice. Emerson and Lei's algorithm for evaluation of μ-calculus formulas suggests a quadratic symbolic algorithm for the problem [7], and many variants of it have been suggested and studied (cf. [13–15]). More involved algorithms with only $O(n \log n)$ [3] and $O(n)$ [9] symbolic steps are known too, but it is not clear that these algorithms are better in practice then the Emerson–Lei algorithm [31].

The *generalized Büchi* acceptance condition consists of a set $\{F_1, \ldots, F_k\}$ of subsets of the state space, and a run of the automaton is accepting iff it visits F_i infinitely often, for all $1 \leqslant i \leqslant k$. The number *k* of sets is the *index* of the automaton. The richer acceptance condition leads to automata with fewer states and simpler underlying structure. For example, the traditional translation of an LTL formula ψ to an NBW results in an automaton with state space $2cl(\psi) \times 2cl(\psi)$ [37]; the set $cl(\psi)$ is the set of ψ's subformulas and each state consists of a "local component", which checks satisfaction of local requirements and an "eventuality component", which checks satisfaction of eventualities. Using the generalized Büchi condition, it is easier to handle the different eventuality requirements, there is no need to the eventuality component, and the state space of the automaton is $2cl(\psi)$ [10]. Nondeterministic generalized Büchi word automata (NGBW, for short) have become popular in specification and verification and are now used in applications traditionally assigned to NBW [20]. Once the NGBW is constructed, it is easy to translate it to an equivalent NBW, and then apply the known algorithms for NBW. For an NGBW with *n* states and index *k*, the constructed NBW has $O(nk)$ states [6].

In this paper, we analyze runs of NGBW, and use the analysis in order to suggest a new complementation construction and a symbolic emptiness algorithm for them. Recall that an NGBW $\mathcal{A}$ rejects a word w if every run of $\mathcal{A}$ has a set F_i in the acceptance condition that is visited only finitely often. The runs of $\mathcal{A}$ can be arranged in a DAG (directed acyclic graph). We show that $\mathcal{A}$ rejects w iff it is possible to label the vertices of the DAG by ranks so that some local conditions on the ranks of vertices and their successors are met. Intuitively, the ranks measure the distance from a position from which no states in F_i are visited.

The complementation construction that follows from the analysis results in an NBW with $2O(n \log nk)$ states. This exponentially improves current complementation constructions, which first translate the NGBW into an NBW with $O(nk)$ states, and ends up in an NBW with $2O(nk \log nk)$ states. Like the construction in [17], our construction is simple and easy to implement. The extension of the reasoning in [17] to NGBW is not trivial and was left open in [12]. (A trivial extension of [17] to NGBW does exist, but results in an NBW with $2O(nk \log nk)$ states. The technical achievement of the construction here is a simultaneous handling of all the sets in the acceptance condition, which is the key to the improved complexity.) The emptiness algorithm that follows from the analysis is almost identical to the OWCTY algorithm of [8] for symbolic detection of bad cycles. Our contribution is the strong relation we draw between the complementation construction and the emptiness algorithm—both naturally follow from the analysis of the runs, which easily implies their correctness.

Beyond the theoretical contribution of the relation between complementation and emptiness, it gives rise to a new *certified* model-checking procedure. As discussed in [27,29,30], it is desirable to accompany a positive answer of a model checker by a proof whose correctness can be verified by methods that are independent of the model checker. As in the case of proof-carrying codes (cf. [28]), such a proof certifies that the system was verified, and checking the certificate is much easier than the original verification task. For a discussion of other applications of certificates, see [27].

Recall that model checking is reduced to checking the emptiness of the product $\mathcal{A}_{S\times\neg\psi}$ of the system S and the complemented specification $\mathcal{A}_{\neg\psi}$. We show that the ranks we associate with the vertices in the run DAG of $\mathcal{A}_{S\times\neg\psi}$ constitute a certificate that the product of S and $\mathcal{A}_{\neg\psi}$ is indeed empty. Moreover, by adding to our symbolic emptiness algorithm an algebraic decision diagram (ADD) that maintains the ranks, the certificate is generated symbolically (ADDs extend OBDDs by allowing the leaves to have values from arbitrary domains, thus they maintain functions that are not necessarily Boolean. Thus, while OBDDs represent Boolean functions, ADDs represent pseudo-Boolean functions [1]). Once the certificate is generated, it can be easily checked, automatically and symbolically, or manually, and it involves only local checks and no fixed points. Unlike the certificates in [27,29,30], whose goal is to provide the user with a deductive proof to ponder, our goal is to generate a compact certificate that can be verified automatically. Since a deductive proof usually consists of a long list of assertions, we believe that machine-checkable certificates are more appropriate in the verification of large systems. The generation of certificates that can be checked automatically is possible thanks to the analysis of runs, which bounds the domain of the well-founded sets that are used in the deductive certificates generated in previous works. As explained in Section 6, our method can generate, for users that are interested in a manual check, also "list-based" proofs.

2. Preliminaries

A *word automaton* is $\mathcal{A} = \langle \Sigma, Q, \delta, Q_{in}, \alpha\rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2Q$ is a transition function, $Q_{in} \subseteq Q$ is a set of initial states, and α is an acceptance condition that defines a subset of $Q\omega$.

Given an input word $w = \sigma_0 \cdot \sigma_1 \cdots$ in $\Sigma\omega$, a *run* of $\mathcal{A}$ on w is a word $r = q_0, q_1, \ldots$ in $Q\omega$ such that $q_0 \in Q_{in}$ and for every $i \geqslant 0$, we have $q_{i+1} \in \delta(q_i, \sigma_i)$; i.e., the run starts in the initial state and obeys the transition function. Since the transition function may specify many possible transitions for each state and letter, $\mathcal{A}$ may have several runs on w. A run is accepting iff it satisfies the acceptance condition α. We consider here the *generalized Büchi* acceptance condition, where $\alpha = \{F_1, \ldots, F_k\}$ is a set of subsets of Q. The number k of sets is the *index* of α (or $\mathcal{A}$). For a run r, let $\mathit{inf}(r)$ denote the set of states that r visits infinitely often. That is,

$$\mathit{inf}(r) = \{q \in Q : q_i = q \text{ for infinitely many } i \geqslant 0\}.$$

As Q is finite, it is guaranteed that $\mathit{inf}(r) \neq \emptyset$. A run r is accepting iff $\mathit{inf}(r) \cap F_j \neq \emptyset$ for all $1 \leqslant j \leqslant k$. That is, r is accepting if every set in α is visited infinitely often. The *generalized co-Büchi* acceptance condition dualizes the generalized Büchi condition. Thus, again $\alpha = \{F_1, F_2, \ldots, F_k\}$ is a set of subsets of Q, but a run r is accepting if $\mathit{inf}(r) \cap F_j = \emptyset$ for some $1 \leqslant j \leqslant k$. Thus, r visits some set in α only finitely often.

If the automaton $\mathcal{A}$ is *nondeterministic*, then it accepts an input word w iff it has an accepting run on w. If $\mathcal{A}$ is *universal*, then it accepts w iff all its runs on w are accepting. The *language* of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$ is the set of words that $\mathcal{A}$ accepts. Dualizing a nondeterministic generalized Büchi automaton (NGBW, for short) amounts to viewing it as a universal generalized co-Büchi automaton (UGCW, for short). It is easy to see that by dualizing $\mathcal{A}$, we get an automaton that accepts its complementary language. Note that nondeterministic Büchi automata (NBW, for short) are a special case of NGBW, with $k = 1$.

In the linear-time approach to model checking, we check whether all the computations of a given system S satisfy a specification ψ. The system is usually given as a labeled state-transition graph and ψ is either a formula in LTL or a word automaton (traditionally, NBW or NGBW). LTL formulas can be translated to word automata. The original translation in [37] uses NBW. More recent translations use NGBW. For example, it is shown in [10] that an LTL formula ψ of length m can be translated to an NGBW $\mathcal{A}_\psi$ that accepts exactly all the words that satisfy ψ. The automaton $\mathcal{A}_\psi$ has $2O(m)$ states and index m.

3. Ranks for UGCW

Let $\mathcal{A} = \langle \Sigma, Q, Q_{in}, \delta, \alpha \rangle$ be a universal generalized co-Büchi automaton with $\alpha = \{F_1, \ldots, F_k\}$. Let $|Q| = n$. 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) $G_r = \langle V, E \rangle$, 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_r contains the vertices $Q_{in} \times \{0\}$.
- $E \subseteq \bigcup_{l \geqslant 0} (Q \times \{l\}) \times (Q \times \{l+1\})$ 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_r embodies exactly all the runs of $\mathcal{A}$ on w. We call G_r the *run* DAG of $\mathcal{A}$ on w. For a set $F \subseteq Q$, we say that a vertex $\langle q, l \rangle$ in G_r is an *F-vertex* iff $q \in F$. We say that G_r is *accepting* if each path π in G_r has an index $1 \leqslant j \leqslant k$ such that π contains only finitely many F_j-vertices. It is easy to see that $\mathcal{A}$ accepts w iff G_r is accepting.

Let $[2n]$ denote the set $\{0, 1, \ldots, 2n\}$, and let $[2n]$*odd* and $[2n]$*even* denote the set of odd and even members of $[2n]$, respectively. Also, let $R = [2n]even \cup ([2n]odd \times \{1, \ldots, k\})$, and $\leqslant$ be the lexicographical order on the elements of R. We refer to the members of R in $[2n]$*even* as *even ranks* and refer to the members of R in $[2n]odd \times \{j\}$ as *odd ranks with index j*.

A *ranking* for G_r is a function $f : V \to R$ that satisfies the following conditions:

1. For all vertices $\langle q, l \rangle \in V$, if $f(\langle q, l \rangle) = \langle 2i+1, j \rangle$, then $q \notin F_j$.
2. For all edges $\langle \langle q, l \rangle, \langle q', l+1 \rangle \rangle \in E$, we have $f(\langle q', l+1 \rangle) \leqslant f(\langle q, l \rangle)$.

Thus, a ranking associates with each vertex in G_r a rank in R so that ranks along paths decrease monotonically, and F_j-vertices cannot get an odd rank with index j. Note that each path in G_r eventually gets trapped in some rank. We say that the ranking f is an *odd ranking* if all the paths of G_r 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_r, there is $l \geqslant 0$ such that $f(\langle q_l, l \rangle)$ is odd, and for all $l' \geqslant l$, 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_r has infinitely many vertices with odd ranks.

In the rest of this section we prove that G_r is accepting iff it has an odd ranking. 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 $\langle q, l \rangle$. For a set $F \subseteq Q$, we say that a vertex $\langle q, l \rangle$

is *F-free* in G iff all the vertices in G that are reachable from $\langle q, l\rangle$ are not F-vertices. Note that, in particular, $\langle q, l\rangle$ is not an F-vertex.

We define an infinite sequence of DAGs $G_0 \supseteq G_11 \supseteq G_12 \supseteq \dots G_1k \supseteq G_1k+1 \supseteq G_31 \supseteq \dots G_3k+1 \supseteq G_51 \dots$ as follows. To simplify notations, we sometimes refer to $G_{2i+1}k+1$ as G_{2i+2}. Thus, $G_2 = G_1k+1$, $G_4 = G_3k+1$, and so on.

- $G_0 = G_r$.
- $G_{2i+1}1 = G_{2i} \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is finite in } G_{2i}\}$.
- $G_{2i+1}j+1 = G_{2i+1}j \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is } F_j\text{-free in } G_{2i+1}j\}$, for $1 \leqslant j \leqslant k$.

Lemma 3.1. *For every $i \geqslant 0$, there exists l_i such that for all $l \geqslant l_i$, there are at most $n - i$ vertices of the form $\langle q, l\rangle$ in G_{2i}.*

Proof. We prove the lemma by an induction on i. The case where $i = 0$ follows from the definition of G_0. Indeed, in G_r all levels $l \geqslant 0$ have at most n vertices of the form $\langle q, l\rangle$. Assume that the lemma's requirement holds for i, we prove it for $i+1$. Consider the DAG G_{2i}. We distinguish between two cases. First, if G_{2i} is finite, then $G_{2i+1}1$ is empty, G_{2i+2} is empty as well, and we are done. Otherwise, we claim that there must be some F_j-free vertex in $G_{2i+1}j$, for some $1 \leqslant j \leqslant k$. To see this, assume, by way of contradiction, that G_{2i} is infinite and all the vertices in $G_{2i+1}j$ are not F_j-free, for all $1 \leqslant j \leqslant k$. Note that then $G_{2i+1}j = G_{2i+1}1$ for all $1 \leqslant j \leqslant k$. Thus, all the vertices in $G_{2i+1}1$ are not F_j-free, for all $1 \leqslant j \leqslant k$. Since G_{2i} is infinite, $G_{2i+1}1$ is also infinite. Also, each vertex in $G_{2i+1}1$ has at least one successor. Consider some vertex $\langle q_0, l_0\rangle$ in $G_{2i+1}1$. Since, by the assumption, it is not F_1-free, there exists an F_1-vertex $\langle q'_0, l'_0\rangle$ reachable from $\langle q_0, l_0\rangle$. Let $\langle q_1, l_1\rangle$ be a successor of $\langle q'_0, l'_0\rangle$. By the assumption, $\langle q_1, l_1\rangle$ is not F_2-free. Hence, there exists an F_2-vertex $\langle q'_1, l'_1\rangle$ reachable from $\langle q_1, l_1\rangle$. Let $\langle q_2, q_2\rangle$ be a successor of $\langle q'_1, l'_1\rangle$. By the assumption, $\langle q_2, l_2\rangle$ is not F_3-free. Thus, we can continue similarly and construct an infinite sequence of vertices $\langle q_h, l_h\rangle$ and $\langle q'_h, l'_h\rangle$ such that for all h, the vertex $\langle q'_h, l'_h\rangle$ is a $F_{(h \bmod k)+1}$-vertex reachable from $\langle q_h, l_h\rangle$, and $\langle q_{h+1}, l_{h+1}\rangle$ is a successor of $\langle q'_h, l'_h\rangle$. Such a sequence, however, corresponds to a path in G_r that visits F_j infinitely often, for all $1 \leqslant j \leqslant k$, contradicting the assumption that G_r is accepting.

So, let j be the minimal index for which there is an F_j-free vertex in $G_{2i+1}j$, and let $\langle q, l\rangle$ be such a vertex. By the minimality of j, we have that $G_{2i+1}j$ is equal to $G_{2i+1}1$, and it contains no finite vertices. Hence, every F_j-free vertex in $G_{2i+1}j$ has a successor, which is also F_j-free, thus we can assume without loss of generality that $l \geqslant l_i$. We claim that taking $l_{i+1} = l$ satisfies the lemma's requirement. That is, we claim that for all $x \geqslant l$, there are at most $n-(i+1)$ vertices of the form $\langle q, x\rangle$ in G_{2i+2}. Recall that $\langle q, l\rangle$ is not finite in G_{2i}. Thus, there are infinitely many vertices in G_{2i} that are reachable from $\langle q, l\rangle$. Hence, by König's lemma, G_{2i} contains an infinite path $\langle q, l\rangle, \langle q_1, l+1\rangle, \langle q_2, l+2\rangle, \dots$. For all $x \geqslant 1$, the vertex $\langle q_x, l+x\rangle$ has infinitely many vertices reachable from it in G_{2i} and thus, it is not finite in G_{2i}. Therefore, the path $\langle q, l\rangle, \langle q_1, l+1\rangle, \langle q_2, l+2\rangle, \dots$ exists also in $G_{2i+1}1$. Recall that $\langle q, l\rangle$ is F_j-free in $G_{2i+1}j$. Hence, being reachable from $\langle q, l\rangle$, all the vertices $\langle q_x, l+x\rangle$ on the path are F_j-free as well. Therefore, they are not in $G_{2i+1}j+1$. It follows that for all $x \geqslant l$, the number of vertices of the form $\langle q, x\rangle$ in $G_{2i+1}j+1$ (and hence also in G_{2i+2}) is strictly smaller than their number in G_{2i}. Hence, by the induction hypothesis, we are done. □

Lemma 3.1 implies that G_{2n} is finite, and $G_{2n+1}1$ is empty.

Each vertex $\langle q, l\rangle$ in G_r has a unique $i \geqslant 1$ such that $\langle q, l\rangle$ is either finite in G_{2i} or F_j-free in $G_{2i+1}j$, for some $1 \leqslant j \leqslant k$. This induces a function $f : V \rightarrow R$ defined as follows:

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

For an odd rank $\eta = \langle 2i+1, j\rangle$, we refer to $G_{2i+1}j$ as G_η.

Lemma 3.2. *For every vertex $\langle q, l\rangle$ in G_r and $\eta \in R$, if $\langle q, l\rangle \notin G_\eta$, then $f(\langle q, l\rangle) < \eta$.*

Proof. We prove the lemma by an induction on η. Since $G_0 = G_r$, the case where $\eta = 0$ is immediate. For the induction step, we distinguish between two cases. For the case $\eta = \langle 2i+1, j\rangle$, consider a vertex $\langle q, l\rangle \notin G_{2i+1}j+1$. If $\langle q, l\rangle \notin G_{2i+1}j$, the lemma's requirement follows from the induction hypothesis. If $\langle q, l\rangle \in G_{2i+1}j$, then $\langle q, l\rangle$ is F_j-free in $G_{2i+1}j$. Accordingly, $f(\langle q, l\rangle) = \langle 2i+1, j\rangle$, meeting the lemma's requirement. For the case $\eta = 2i$, consider a vertex $\langle q, l\rangle \notin G_{2i+1}1$. If $\langle q, l\rangle \notin G_{2i}$, the lemma's requirement follows from the induction hypothesis. If $\langle q, l\rangle \in G_{2i}$, then $\langle q, l\rangle$ is finite in G_{2i}. Accordingly, $f(\langle q, l\rangle) = 2i$, meeting the lemma's requirement. $\square$

Lemma 3.3. *For every two vertices $\langle q, l\rangle$ and $\langle q', l'\rangle$ in G_r, if $\langle q', l'\rangle$ is reachable from $\langle q, l\rangle$, then $f(\langle q', l'\rangle) \leqslant f(\langle q, l\rangle)$.*

Proof. We distinguish between two cases. If $f(\langle q, l\rangle) = 2i$, then $\langle q, l\rangle$ is finite in G_{2i}. Hence, either $\langle q', l'\rangle$ is not in G_{2i}, in which case, by Lemma 3.2, we have that $f(\langle q', l'\rangle) < 2i$, or $\langle q', l'\rangle$ is in G_{2i}, in which case, being reachable from $\langle q, l\rangle$, it must be finite in G_i, with $f(\langle q', l'\rangle) = 2i$, and we are done. If $f(\langle q, l\rangle) = \langle 2i+1, j\rangle$, then $\langle q, l\rangle$ is F_j-free in $G_{2i+1}j$. Hence, either $\langle q', l'\rangle$ is not in $G_{2i+1}j$, in which case, by Lemma 3.2, we have that $f(\langle q', l'\rangle) < \langle 2i+1, j\rangle$, or $\langle q', l'\rangle$ is in $G_{2i+1}j$, in which case, being reachable from $\langle q, l\rangle$, it must be F_j-free in $G_{2i+1}j$, in which case $f(\langle q', l'\rangle) = \langle 2i+1, j\rangle$, and we are done. $\square$

Lemma 3.4. *For every infinite path in G_r, there exists an index $1 \leqslant j \leqslant k$ and a vertex $\langle q, l\rangle$ with an odd rank with index j such that all the vertices $\langle q', l'\rangle$ on the path that are reachable from $\langle q, l\rangle$ have $f(\langle q', l'\rangle) = f(\langle q, l\rangle)$.*

Proof. By Lemma 3.3, in every infinite path in G_r, there exists a vertex $\langle q, l\rangle$ such that all the vertices $\langle q', l'\rangle$ on the path that are reachable from $\langle q, l\rangle$ have $f(q', l') = f(\langle q, l\rangle)$. We need to prove that $f(\langle q, l\rangle)$ is odd. Assume, by way of contradiction, that $f(\langle q, l\rangle)$ is some even i. Thus, $\langle q, l\rangle$ is finite in G_i. Then, all the vertices $\langle q', l'\rangle$ on the path that are reachable from $\langle q, l\rangle$ also have $f(\langle q', l'\rangle) = i$. By Lemma 3.2, they all belong to G_i. Since the path is infinite, there are infinitely many such vertices, contradicting the fact that $\langle q, l\rangle$ is finite in G_i. $\square$

We can now conclude with Theorem 3.5.

Theorem 3.5. *G_r is accepting iff it has an odd ranking.*

Proof. Assume first that there is an odd ranking for G_r. Then, every path in G_r eventually gets trapped in some odd rank $\langle 2i+1, j\rangle$. Hence, as F_j-vertices cannot get this rank, the path visits F_j only finitely often, and we are done.

For the other direction, note that Lemma 3.3, together with the fact that a vertex gets an odd rank with index *j* only if it is F_j-free, imply that the function *f* described above is a ranking. Lemma 3.4 then implies that the ranking is odd. □

We note that the reasoning above is similar to the one described for co-Büchi automata in [17]. The extension to the case of generalized co-Büchi is not trivial and involves a refinement of the DAG G_{2i+2}. In particular, the minimality of *j* in the proof of Lemma 3.1 is crucial for its correctness.

4. Complementation of NGBW

Theorem 3.5 implies that a UGCW $\mathcal{A}$ accepts a word w iff there is an odd ranking for the run DAG G_r of $\mathcal{A}$ on w—a ranking in which every infinite path in G_r has infinitely many vertices with an odd rank. Intuitively, the theorem suggests that the requirements imposed by the generalized co-Büchi condition (finitely often, for some set in α) can be reduced to a new condition of a simpler type (infinitely often, for vertices with an odd rank). Recall that by dualizing an NGBW, we get a UGCW for the complementary language. Theorem 3.5 enables us to translate this UGCW to an NBW, resulting in the complementation construction described.

Theorem 4.1. *Let $\mathcal{A}$ be an NGBW with n states and index k. There is an NBW $\mathcal{A}'$ with* $2O(n\log kn)$ *states such that* $\mathcal{L}(\mathcal{A}') = \Sigma\omega \setminus \mathcal{L}(\mathcal{A})$.

Proof. Let $\tilde{\mathcal{A}}$ be the UGCW that dualizes $\mathcal{A}$. The UGCW $\tilde{\mathcal{A}}$ accepts exactly all words rejected by $\mathcal{A}$. We obtain $\mathcal{A}'$ by translating $\tilde{\mathcal{A}}$ to an NBW. When $\mathcal{A}'$ reads a word w, it guesses a ranking for the run DAG G_r of $\tilde{\mathcal{A}}$ on w. At a given point of a run of $\mathcal{A}'$, it keeps in its memory a whole level of G_r and a guess for the ranks of the vertices at this level. In order to check that the ranking is odd, $\mathcal{A}'$ keeps track of states that owe a visit to vertices with odd ranks.

Before we define $\mathcal{A}'$, we first need some notations. A *level ranking* for $\mathcal{A}$ is a function $g: Q \to R$, such that if $g(q)$ is odd with index *j*, then $q \notin F_j$. Let $\mathcal{R}$ be the set of all level rankings. For $S \subseteq Q$ and a letter σ, let $\delta(S,\sigma) = \bigcup_{s\in S}\delta(s,\sigma)$. Note that if level $l-1$ in G_r contains the states in *S*, and the *l*th letter in w is σ, then level *l* of G_r contains the states in $\delta(S,\sigma)$. For two level rankings *g* and g' in $\mathcal{R}$, a set $S \subseteq Q$, and a letter σ, we say that g' *covers* $\langle g, S, \sigma\rangle$ if for all $q \in S$ and $q' \in Q$, if $q' \in \delta(q,\sigma)$, then $g'(q') \leqslant g(q)$. Thus, if the vertices of level $l-1$ contain exactly all the states in *S*, then *g* describes the ranks of these vertices, and the *l*th letter in w is σ, then g' is a possible level ranking for level *l*. Finally, for $g \in \mathcal{R}$, let $odd(g) = \{q : g(q) \in [2n]odd \times \{1,\ldots,k\}\}$. Thus, $odd(g)$ contains states to which *g* gives an odd rank.

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

- $Q' = 2Q \times 2Q \times \mathcal{R}$. A state $\langle S, O, g\rangle \in Q'$ indicates that the current level of the DAG contains the states in *S* and the guessed level ranking for the current level is *g*. The set

$O \subseteq S$ contains states along paths that have not visited a vertex with an odd rank since the last time O has been empty.

- $q'_{in} = \langle \{q'_{in}\}, \emptyset, f_{in} \rangle$, where $f_{in}(q) = 2n$ for all $q \in Q$.
- 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 $\langle g, S, \sigma \rangle\}$.
  - 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 $\langle g, S, \sigma \rangle\}$.
- $\alpha' = 2Q \times \{\emptyset\} \times \mathcal{R}$.

Thus, when $\mathcal{A}'$ reads the lth letter in the input, it guesses the level ranking for level l in the run DAG. This level ranking should cover the level ranking of level $l-1$. In addition, in the O component, $\mathcal{A}'$ keeps track of states along paths that owe a visit to a vertex with an odd rank. When all the paths of the DAG have visited a vertex with an odd rank, the set O becomes empty, and is initiated according to the states in the current level and its ranking. The acceptance condition then checks that there are infinitely many levels in which O become empty.

We now prove the correctness of the construction formally. For that, we prove that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\tilde{A})$, for the UGCW $\tilde{A}$ that complements $\mathcal{A}$. We first prove that $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\tilde{A})$. Consider a word $w = \sigma_0 \cdot \sigma_1 \cdots$ accepted by $\mathcal{A}'$. Let $r = \langle S_0, O_0, g_0 \rangle, \langle S_1, O_1, g_1 \rangle, \ldots$ be an accepting run of $\mathcal{A}'$ on w. Consider the DAG $G_r = \langle V, E \rangle$, where $V \subseteq Q \times \mathbb{N}$ is such that $\langle q, l \rangle \in V$ iff $q \in S_l$, and $E \subseteq \bigcup_{l \geqslant 0} (Q \times \{l\}) \times (Q \times \{l+1\})$ is such that $E(\langle q, l \rangle, \langle q', l+1 \rangle)$ iff $\langle q, l \rangle \in V$ and $q' \in \delta(q, \sigma_l)$. By the definition of δ', the DAG G_r embodies exactly all the runs of $\tilde{A}$ on w. Indeed the S-component of the states of $\mathcal{A}'$ follows the subset construction, and S_l contains exactly all the states that are teachable from q_{in} by reading $\sigma_0 \cdots \sigma_{l-1}$. So, G_r is the run DAG of $\tilde{A}$ on w. By Theorem 3.5, it is left to prove that G_r has an odd ranking. Let $f : V \to [2n]$ be such that $f(\langle q, l \rangle) = g_l(q)$. Since, g_l is a level ranking for $\mathcal{A}$, then whenever $g_l(q)$ is odd with index j, it is guaranteed that $q \notin F_j$. Also, in the definition of δ', we require g_{l+1} to cover $\langle g_l, S_l, \sigma_l \rangle$. Therefore, f is a ranking for G_r. We prove that f is an odd ranking. Let $l, l' \geqslant 0$ be such that $l' > l$, $O_l = \emptyset$, and $O_{l''} \neq \emptyset$ for $l < l'' \leqslant l$. By the definition of δ', the set $O_{l'}$ is a subset of $S_{l'}$ containing states q such that there is a path in G_r that starts in level l, reaches $\langle q, l' \rangle$ and does not visit a vertex with an odd rank. Also, whenever O_l is empty, O_{l+1} is "recharged" with states $q \in S_l$ for which the rank of $\langle q, l \rangle$ is even. Since r is accepting, it visits infinitely many states with an empty O component, indicating that every path of G_r has infinitely many vertices with odd ranks, thus f is an odd ranking.

It is left to prove that $\mathcal{L}(\tilde{A}) \subseteq \mathcal{L}(\mathcal{A}')$. Consider a word w accepted by $\tilde{A}$. Let G_r be the accepting run DAG of $\tilde{A}$ on w. By Theorem 3.5, there is an odd ranking f for G_r. Consider a state $\langle S_l, O_l, g_l \rangle$ of $\mathcal{A}'$. By the definition of δ', fixing the level ranking g_{l+1} induces a unique S_{l+1} and O_{l+1}. Indeed, both components follow the subset construction, with the O_{l+1} component subtracting states in $odd(g_{l+1})$. For $l \geqslant 0$, let $g_l : Q \to [2n]$ be such that for all states q for which $\langle q, l \rangle$ is a vertex of G_r, we have that $g_l(q) = f(\langle q, l \rangle)$ (for states for which $\langle q, l \rangle$ is not a vertex of G_r, the value of $g_l(q)$ is not important, and can be set arbitrarily). Since f is an odd ranking for G_r, the function g_l is a level ranking for $\mathcal{A}$. Let $r = \langle S_0, O_0, g_0 \rangle, \langle S_1, O_1, g_1 \rangle, \ldots$ be the run of $\mathcal{A}'$ that is induced by the level rankings

$g_0, g_1, \ldots$. Since f is odd, every path of G_r has infinitely many vertices with odd ranks. Therefore, the O-components along r are repeatedly being emptied, and r is accepting.

Finally, since there are at most $(k(2n+1))n$ level rankings, the number of states in $\mathcal{A}'$ is at most $22n \cdot (k(2n+1))n = 2\mathrm{O}(n \log kn)$. $\square$

Note that the previous complementation construction for NGBW involves a $2\mathrm{O}(nk \log nk)$ blow up, as it first translates the NGBW into an NBW with $\mathrm{O}(nk)$ states, and complementing an NBW with m states results in an NBW with $2\mathrm{O}(m \log m)$ states [32,25]. Thus, our construction improves the previous construction exponentially.

In [17], we used a simpler analysis of ranks in order to translate an alternating co-Büchi automaton (ACW, for short) $\mathcal{A}$ with n states to an alternating weak automaton (AWW, for short) $\mathcal{A}'$ with $\mathrm{O}(n2)$ states. In the case of a co-Büchi acceptance condition, odd ranks need not be coupled with an index. Accordingly, the state space of $\mathcal{A}'$ consists on pairs $\langle q, r\rangle$, where q is a state of $\mathcal{A}$ and i is a rank in $[2n]$. A visit of $\mathcal{A}'$ in the state $\langle q, i\rangle$ when it reads the lth letter in the input word indicates that the rank of the vertex $\langle q, l\rangle$ in an accepting run DAG is i. In a similar way, our analysis here can also be used in order to translate an alternating generalized co-Büchi word automaton (AGCW, for short) with n states and index k to an AWW with $\mathrm{O}(n2k)$ states. Here, the states also consist of pairs $\langle q, i\rangle$, with i being a rank in R. Formally, we have the following (see Appendix A for the detailed construction).

Theorem 4.2. *Let $\mathcal{A}$ be an alternating generalized co-Büchi automaton with n states and index k. There is a weak alternating automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$ and the number of states in $\mathcal{A}'$ is* $\mathrm{O}(n2k)$.

Note that the result of a construction that first translates the AGCW into an ACW and then uses [17] is an AWW with $\mathrm{O}(n2k2)$ states. This, as well as the improved complementation construction, suggests that when it is possible to use both NGBW and NBW, one should prefer an NGBW with fewer states, even if its index is large. The above give rise to the following problem:

Given an NBW, find an equivalent NGBW with fewer states.

In particular, it would be interesting to look for a variant of the translation in [26], of an alternating Büchi word automaton with state space Q into an NBW with state space $2Q \times 2Q$, that will end up in an NGBW with state space $2Q$.

5. Model checking

Recall that the model-checking problem is reduced to the emptiness problem of an NGBW $\mathcal{A}_{S\times\neg\psi}$ over a single-letter alphabet. Equivalently, we can check the nonemptiness of the UGCW $\tilde{\mathcal{A}}_{S\times\neg\psi}$ that dualizes $\mathcal{A}_{S\times\neg\psi}$. Indeed, since $\mathcal{A}_{S\times\neg\psi}$ has a single-letter alphabet, it is empty iff $\tilde{A}_{S\times\neg\psi}$ is not empty (see also [21]).

In this section we describe a symbolic algorithm for a single-letter UGCW non-emptiness. The algorithm is induced by the analysis in Section 3, and its correctness follows immediately from Theorem 3.5. The algorithm is a variant of Emerson–Lei algorithm and is similar to the OWCTY algorithm of [8] for detecting bad-cycles (see also [14,15]). The tight relation between the complementation construction and the emptiness procedure is very interesting, as it shows that progress in the emptiness procedure can be measured by means of the ranks that are used in the construction of the complementary automaton. As we describe in the next section, this observation gives rise to a new *certified* model-checking procedure.

Consider a single-letter UGCW $\mathcal{A} = \langle \{a\}, Q, \delta, Q_{in}, \alpha \rangle$. The analysis in Section 3 associates ranks with members of the infinite set $Q \times \mathbb{N}$. On the other hand, nonemptiness algorithms handle the finite state set Q. Accordingly, we first associate ranks with states:

Lemma 5.1. *Consider a UGCW $\mathcal{A}$ over a single-letter alphabet. Then, for every state q of $\mathcal{A}$, all the vertices in $\{q\} \times \mathbb{N}$ have the same rank.*

Proof. Consider a state q and two levels l_1 and l_2 such that $\langle q, l_1 \rangle$ and $\langle q, l_2 \rangle$ are vertices in G_r. Recall that the DAG G_r embodies all the runs of $\mathcal{A}$ on the input word. Since $\mathcal{A}$ is a single-letter UGCW, the sub-DAG with root $\langle q, l_1 \rangle$ coincides with the sub-DAG with root $\langle q, l_2 \rangle$. Indeed, both embody exactly all the runs of $\mathcal{A}$ with initial state q on $a\omega$. Thus, all the sub-DAGs of G_r with roots in $\{q\} \times \mathbb{N}$ coincide. Hence, it is easy to prove by an induction on $r \in R$ that for all states q and $r \in R$, either all vertices in $\{q\} \times \mathbb{N}$ get rank r, or no vertex in $\{q\} \times \mathbb{N}$ gets rank r. Indeed, again by an induction on i and j, for each DAG G_{2i} or $G_{2i+1}j$ in the sequence of DAGs, either all vertices in $\{q\} \times \mathbb{N}$ are in the DAG, or no vertex is in it. □

For a state q of $\mathcal{A}$, the *rank of q*, denoted *rank*(q), is the rank of the vertices $\{q\} \times \mathbb{N}$ in G_r. We are now ready to describe the nonemptiness procedure that follows. The procedure, described in Fig. 1, gets as input the UGCW $\mathcal{A}$ and calculates the set b of all the states q such that $\mathcal{A}$ with initial state q is empty. The UGCW $\mathcal{A}$ is then not empty iff $Q_{in} \cap b = \emptyset$. The algorithm uses the following set-based operations (all easily implemented by means of OBDDs).

- The operator *pre* : $2Q \to 2Q$. Given a set γ of states, the set *pre*(γ) contains all states that have an immediate successor in γ. Formally, $q \in pre(\gamma)$ iff $\delta(q, a) \cap \gamma \neq \emptyset$ (in temporal logic, $q \models EX\gamma$).
- The operator *until* : $2Q \times 2Q \to 2Q$. Given two sets η and γ of states, the set *until*(η, γ) contains all states that reach a state in $\gamma \cap \eta$ via states in η. Formally, $q \in until(\eta, \gamma)$ iff there are $q_0, \ldots, q_l$ such that $q_0 = q$, for all $0 \leqslant i < l$, we have that $q_{i+1} \in \delta(q_i, a)$ and $q_i \in \eta$, and $q_l \in \eta \cap \gamma$ (in temporal logic, $q \models E\eta U(\eta \wedge \gamma)$). Note that the operator *until* can be implemented by repeatedly applying the *pre* and intersection operators, until a fixpoint is reached.

Note that the set b is monotonically decreasing during the execution of the procedure NonEmpty. Intuitively, b contains all states that have not yet been ranked. At initialization, b contains all the states, and in each iteration it is intersected with some set. We say that a state q is removed from b in iteration $i[0]$ if q is removed from b during the internal while loop of the ith external while loop. We say that q is removed from b in iteration $(i[1], j)$ if q is removed from b during the jth internal for loop of the ith external while loop. Lemma 5.2 then follows directly from the definition of ranks.

```
procedure NonEmpty($\mathcal{A}$)
  $b := Q$;
  while $b$ changes do
    while $b$ changes do $b := b \cap pre(b)$;
    for $j = 1 \ldots k$ do $b := b \cap until(b, F_j)$;
  if $Q_{in} \cap b = \emptyset$ then return("not empty") else return("empty");
```

Fig. 1. A nonemptiness procedure.

Lemma 5.2. *Consider a state q in $\mathcal{A}$.*
- *The state q is removed from b in iteration $i[0]$ iff $rank(q) = 2i$.*
- *The state q is removed from b in iteration $(i[1], j)$ iff $rank(q) = \langle 2i + 1, j\rangle$.*

By Lemma 5.2, a state q is removed from b during the execution of the procedure if it has a well-defined rank, which holds, by Theorem 3.5, if $\mathcal{A}$ with initial state q is not empty. Thus, Lemma 5.2, together with the analysis in Section 3, naturally induce the algorithm and immediately imply its correctness. (The only, minor, difference between our algorithm and the OWCTY algorithm [8], is that in our algorithm the internal while loop precedes the internal for loop, rather than the other way around. Since the purpose of the internal while loop is to eliminate quickly states that cannot be on a cycle, it makes sense to apply it as soon as possible.)

Remark 5.3. When ψ is a safety property, the automaton $\mathcal{A}_\psi$ is a looping automaton (all infinite runs are accepting), and the automaton $\mathcal{A}_{\neg\psi}$ can be defined as a nondeterministic automaton on finite words [33,18]. Thus, a run of $\mathcal{A}_{\neg\psi}$ is accepting iff it reaches a set F of accepting states. Accordingly (assuming that the system has no fairness conditions), the automaton $\tilde{\mathcal{A}}_{S\times\neg\psi}$ is a universal automaton in which all runs except these that reach F are accepting. As a result, we need a much simpler nonemptiness procedure, which corresponds to backwards traversal. Thus, it initializes b with $Q \setminus F$, follows with the single while loop **while** b changes **do** $b := b \cap pre(b)$, and returns "not empty" when $Q_{in} \subseteq b$.

6. Certified model checking

Recall that $\tilde{A}_{S\times\neg\psi}$ with initial state q is not empty iff $rank(q)$ is well defined, and hence belongs to R. Thus, beyond a correctness proof, the analysis in Section 3 can be used in order to accompany the output of the procedure described in Fig. 1 by a certificate, namely the odd ranking, that $\tilde{A}_{S\times\neg\psi}$ is indeed not empty, and S satisfies ψ. In this section we describe how to generate and check such a certificate.

By definition, a function $f : V \to R$ is an odd ranking if F_j-vertices do not get an odd rank with index j, ranks along paths decrease monotonically, and all the paths of G_r eventually get trapped in an odd rank. The number of vertices along a path that get an even rank depends on the input word and is in general unbounded. Accordingly, checking whether a given function f is an odd ranking involves, in addition to local checks, also a check for eventualities, which involves a fixed-point computation. We now show that when $\mathcal{A}$ is a single-letter automaton, as is the case with $\tilde{A}_{S\times\neg\psi}$, it is possible to bound the number of vertices that get even ranks. Let $\mathcal{A} = \langle\{a\}, Q, \delta, Q_{in}, \alpha\rangle$ be a single-letter UGCW. Consider a vertex $\langle q, l\rangle$ that is finite in G_{2i}. Let *height*(q, l) be the length of the longest path from $\langle q, l\rangle$ to a leaf of G_{2i}.

Lemma 6.1. *For all vertices $\langle q, l\rangle$, we have that height$(q, l) \in \{0, \ldots, n-1\}$.*

Proof. Assume by way of contradiction that G_{2i} contains a vertex $\langle q, l\rangle$ such that *height*$(q, l) \geqslant n$. Then, the longest path from $\langle q, l\rangle$ to a leaf of G_{2i} contains at least one state q' that repeats at least twice. Thus, there is a path in G_{2i} that starts in $\langle q, l\rangle$, reaches a vertex $\langle q', l_1\rangle$ with $l_1 \geqslant l$ and continues to a vertex $\langle q', l_2\rangle$ with $l_2 > l_1$. As argued in the proof of Lemma 5.1, the sub-DAG with root $\langle q', l_1\rangle$ coincides with the sub-DAG with root $\langle q', l_2\rangle$. Hence, there is a path in G_{2i} that starts in $\langle q', l_2\rangle$ and reaches a vertex $\langle q', l_3\rangle$ with $l_3 > l_2$. By repeating this argument, we obtain an infinite path in G_{2i} that starts in $\langle q, l\rangle$, contradicting the fact that $\langle q, l\rangle$ is finite. □

Following Lemma 6.1, we refine our set of ranks to $R = ([2n]even \times \{1, \ldots, n\}) \cup ([2n]odd \times \{1, \ldots, k\})$. We say that a function $f : V \to R$ is a *bounded odd ranking* if the following holds:

1. For all vertices $\langle q, l\rangle \in V$, if $f(\langle q, l\rangle) = \langle 2i+1, j\rangle$, then $q \notin F_j$.
2. Consider an edge $\langle\langle q, l\rangle, \langle q', l+1\rangle\rangle \in E$.
   (a) If $f(\langle q, l\rangle)$ is odd, then $f(\langle q', l+1\rangle) \leqslant f(\langle q, l\rangle)$.
   (b) If $f(\langle q, l\rangle)$ is even, then $f(\langle q', l+1\rangle) < f(\langle q, l\rangle)$.

Thus, in a bounded odd ranking, the rank of successors of a vertex $\langle q, l\rangle$ with an even rank must be strictly smaller than the rank of $\langle q, l\rangle$.

Theorem 6.2. *Let $\mathcal{A}$ be a single-letter automaton. Then, G_r is accepting iff it has a bounded odd ranking.*

Proof. Each bounded odd ranking is also an odd ranking (with the height component being ignored). Thus, the direction from right to left follows from Theorem 3.5. For the other direction, we refine the function *rank* : $V \to R$ to account for heights of vertices. Thus, *rank*(q, l), for $\langle q, l\rangle$ that is finite with height h in G_{2i} is $\langle 2i, h\rangle$. It is easy to see that, as with odd ranking, the first two conditions on rank being a bounded odd ranking hold. For the third condition, consider a vertex $\langle q, l\rangle$ with rank $\langle 2i, h\rangle$. By the definition of height, the successors of $\langle q, l\rangle$ in G_{2i} have heights that are strictly smaller than h. By Lemma 3.2, the successors of $\langle q, l\rangle$ that are not in G_{2i} have rank that is strictly smaller than $2i$. Thus, all the successors of $\langle q, l\rangle$ have ranks that are strictly smaller than $\langle 2i, h\rangle$, and we are done. □

It turns out that the nonemptiness procedure actually accounts for heights too: we say that a state q is removed from b in iteration $(i[0], h)$ if q is removed from b during the ith external while loop and its hth internal while loop (we start to count iterations from 0). Lemma 6.3 then follows directly from the definition of ranks.

```
procedure Certified NonEmpty($\mathcal{A}$)
  $b := Q$; $f := \emptyset$; $i := -1$;
  while $b$ changes do
    $i := i + 1$;
    $h := 0$;
    while $b$ changes do
      $f := f \cup assign(b \setminus pre(b), (2i, h))$;
      $b := b \cap pre(b)$;
      $h := h + 1$;
    for $j = 1 \ldots k$ do
      $f := f \cup assign(b \setminus until(b, F_j), (2i + 1, j))$;
      $b := b \cap until(b, F_j)$;
  if $Q_{in} \cap b = \emptyset$ then return(“not empty with certificate” $f$) else return
   (“empty”);
```

Fig. 2. A nonemptiness procedure that generates a certificate.

Lemma 6.3. *Consider a state q in $\mathcal{A}$.*
- *The state q is removed from b in iteration $(i[0], h)$ iff rank$(q) = \langle 2i, h\rangle$.*
- *The state q is removed from b in iteration $(i[1], j)$ iff rank$(q) = \langle 2i + 1, j\rangle$.*

In a symbolic implementation of the procedure NonEmpty, we maintain *b* in an OBDD. By maintaining in addition an ADD that maps states to ranks, we generate a certificate that can be used to certify the model-checking procedure. Let $f : Q \to R$ be a partial function from *Q* to *R*. The procedure Certified_Nonempty described in Fig. 2 gets as input a single letter UGCW $\mathcal{A}$ and calculates, in addition to the set *b*, also a function *f* that describes the odd ranking, which is returned in case no state of Q_{in} is in *b*.[4] The procedure uses the operator *assign*, which given a set $\gamma \subseteq Q$ and a rank $r \in R$, returns a function in which all the states in γ are assigned *r*. In addition, initializing *f* to $\emptyset$ corresponds to an empty function, and $\cup$ between two functions with disjoint domains returns their union.

The procedure can be easily implemented symbolically, with *f* being maintained in an ADD. Note that *R* consists of pairs, thus in some ADD implementations, where the domain of the ADD is restricted to single values, we have to encode *R*, which is easy.

Once the procedure Certified_Nonempty terminates and *f* is returned to the user, she can check that *f* represent a bounded odd ranking and that all the states in Q_{in} have a rank. Note how the use of heights, which enables us to consider bounded odd rankings, is essential here, as checking *f* involves only local checks (a comparison of a rank of vertices and their successors) and no fixed points. As described in the procedure Check_Certificate in Fig. 3, the check can be done automatically and symbolically. The procedure uses the following Boolean functions:
- *undef* : $2Q \to \{\textbf{true}, \textbf{false}\}$. Given a set γ of states, *undef*(γ) is true iff *f* does not assign a rank to some state in γ; i.e., $\gamma \cap comp(f - 1(R)) \neq \emptyset$.
- *oops*: $\{1, \ldots, k\} \times 2Q \to \{\textbf{true}, \textbf{false}\}$. Given an index $1 \leqslant j \leqslant k$ and a set γ of states, *oops*(j, γ) is true iff there is $q \in \gamma$ with an odd rank with index *j*; i.e., $f(q) \in [2n]odd \times \{j\}$.

[4] In case the intersection of Q_{in} and *b* is not empty, it is possible to enhance the procedure to return an evidence to the emptiness of $\mathcal{A}$; this is similar to the known generation of counterexamples and we do not discuss it here.

```
procedure Check_Certificate($\mathcal{A}, f$)
  if $undef(Q_{in})$ then return("incorrect certificate");
  for $j = 1 \ldots k$ do
    if $oops(j, F_j)$ then return("incorrect certificate");
  for all $q \in Q$ and $q' \in \delta(q, a)$ do
    if $f(q)$ is odd and $f(q') > f(q)$ then return ("incorrect certificate");
    if $f(q)$ is even and $f(q') \geq f(q)$ then return ("incorrect certificate");
  return("correct certificate");
```

Fig. 3. Verifying that a certificate is correct.

The symbolic implementation of *oops* checks whether the intersection of $f - 1([2n]odd \times \{j\})$ and b is empty.
The correctness of the procedure Check_Certificate follows immediately from Theorem 6.2.

In the case of LTL model checking, the automaton $\mathcal{A}$ is $\tilde{\mathcal{A}}_{S\times\neg\psi}$ and its state space consists of pairs $\langle s, P\rangle$, where s is a state of S and $P \subseteq cl(\psi)$ is a set of LTL formulas. The automaton $\tilde{\mathcal{A}}_{S\times\neg\psi}$ with initial state $\langle s, P\rangle$ is not empty if each path that starts in s violates at least one formula in P. Each set in the generalized Büchi acceptance condition corresponds to a formula of the form $\varphi U\theta$ and consists of all the states $\langle s, P\rangle$ in which P contains θ or does not contain $\varphi U\theta$. The rank of a state $\langle s, P\rangle$ explains how one of the formulas in P is not satisfied. If the rank is even, the explanation is transferred, via local conditions, to the successors of s. If the rank is odd, the particular formula that is not satisfied is recorded (by means of the index of the odd rank). Hence, for users that prefer to get a certificate that is similar to deductive proofs generated by proof-theoretic approaches to verification [22] (as in [29,30]), it is possible to present the certificate as a list of states and how they satisfy (that is, do not satisfy the negation of) relevant subformulas of ψ.

Remark 6.4. As discussed in Remark 5.3, when ψ is a safety property, nonemptiness of $\tilde{\mathcal{A}}_{S\times\neg\psi}$ can be checked by backwards traversal and the nonemptiness procedure consists of the single loop **while** b changes **do** $b := b \cap pre(b)$. In this case, we can take b itself as the certificate, and there is no need to compute ranks. To check that b is a correct certificate, one has to check that $Q_{in} \subseteq b$, $b \cap F = \emptyset$, and $b = b \cap pre(b)$. Thus, while the computation of b involves a fixed-point, checking that it is indeed a fixed-point involves only local checks.

7. Discussion

We described a new complementation construction for NGBW. The analysis behind the complementation construction led to a symbolic certified model-checking procedure. Our certificate enables the user to verify the emptiness of an NGBW. A naive alternative way to construct a certificate for NBW emptiness[5] is to construct an ROBDD R for the (non-reflexive) transitive closure of the edge relation E of the NBW. In order to use R as

[5] In the discussion here, we ignore, for simplicity sake, the fact the Büchi condition may be generalized.

a certificate, we need to do the following:

- Check that R contains the transitive closure of E; thus, $E \subseteq R$ and $R \circ E \subseteq R$, where $\circ$ is relational composition.
- Check that the NBW is empty; thus, check that $(\exists x, y)(Init(x) \wedge F(y) \wedge (x = y \vee R(x, y)) \wedge R(y, y))$ is empty, with *Init* and F being ROBDDs for the set of initial and accepting states, respectively.

The drawback of this simple alternative certificate is the complexity of generating and checking R: if there are n state variables, then R requires $2n$ ROBDD variables. It is true that the OWCTY algorithm, and hence also our certificate, refer to E, which also requires an ROBDD with $2n$ variables. But while E is the transition relation, which is a "natural relation" and may not be constructed explicitly (in particular, one can use conjunctive partitioning [2]), this is not the case for R. Efforts to implement computation of the transitive closure are not successful. While the computation can converge in time logarithmic in the state space, the ROBDDs explode [5]. In addition to the ROBDD that our approach uses in order to refer to E, we also need an ADD with n variables and a domain of size $O(n)$, which corresponds to a total of $n + O(\log n)$ ROBDD variables.

Another open problem refers to the particular nonemptiness algorithm we were able to relate to the complementation construction and to augment with a certificate. Recall that our algorithm is similar to the OWCTY algorithm, which is a variant of the quadratic Emerson–Lei algorithm. As discussed in Section 1, more recent algorithms solve the nonemptiness problem with a sub-quadratic number of symbolic steps [3,9]. We believe that it is possible to generate certificates that are based on the behavior of these algorithms. As is the case with our certificate, the symbolic nature of the algorithms should also enable a symbolic generation and verification of the certificate in terms of ADDs. On the other hand, among the known symbolic algorithms, only Emerson–Lei is suitable for infinite graphs, which is essential to the rank analysis. Therefore, we believe that only variants of the Emerson–Lei algorithm, such as OWCTY, can be related to complementation. Note, however, that Emerson–Lei cannot be applied directly to complementation, since, unlike OWCTY, it does not yield bounded ranks.

Appendix A. From AGCW to AWW

Theorem A.1. *Let $\mathcal{A}$ be an alternating generalized co-Büchi automaton with n states and index k. There is a weak alternating automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$ and the number of states in $\mathcal{A}'$ is $O(n2k)$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$, with $\alpha = \{F_1, \ldots, F_k\}$, and let $n = |Q|$. Recall that $R = [2n]even \cup ([2n]odd \times \{1, \ldots, k\})$ is the set of all possible ranks, and $\leqslant$ is the lexicographical order on the elements of R. We define $\mathcal{A}' = \langle \Sigma, Q', q'_{in}, \delta', \alpha' \rangle$, where

- $Q' = Q \times R$. Intuitively, when the automaton is in state $\langle q, i \rangle$ as it reads the letter σ_l (the l'th letter in the input), then it guesses that in a memoryless 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.
- $q'_{in} = \langle q_{in}, 2n \rangle$. That is, q_{in} is paired with $2n$, which is an upper bound on the rank of $\langle q_{in}, 0 \rangle$.

- We define δ' by means of a function

$$\mathit{release}: \mathcal{B}+(Q) \times R \to \mathcal{B}+(Q').$$

Given a formula $\theta \in \mathcal{B}+(Q)$, and a rank $i \in R$, the formula $\mathit{release}(\theta, i)$ is obtained from θ by replacing an atom q by the disjunction $\bigvee_{i' \leqslant i} \langle q, i' \rangle$. For example, if $k = 3$, then $\mathit{release}(q_3 \wedge q_5, 2) = (\langle q_3, 2\rangle \vee \langle q_3, (1,3)\rangle \vee \langle q_3, (1,2)\rangle \vee \langle q_3, (1,1)\rangle \vee \langle q_3, 0\rangle) \wedge (\langle q_5, 2\rangle \vee \langle q_5, (1,3)\rangle \vee \langle q_5, (1,2)\rangle \vee \langle q_5, (1,1)\rangle \vee \langle q_5, 0\rangle)$. 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:

$$\delta'(\langle q, i\rangle, \sigma) = \begin{cases} \mathit{release}(\delta(q,\sigma), i) & \text{If } q \notin F_j \text{ or } i \text{ is not odd with index } j. \\ \textbf{false} & \text{If } q \in F_j \text{ and } i \text{ is odd with index } j. \end{cases}$$

That is, if the current guessed rank is *i* then, by employing *release*, the run can move in its successors to any rank that is smaller than *i*. If, however, $q \in F_j$ and the current guessed rank is odd with index *j*, then, by the definition of ranks, the current guessed rank is wrong, and the run is rejecting.

- $\alpha' = Q \times ([2n]\mathit{odd} \times \{1, \ldots, k\})$. That is, infinitely many guessed ranks along each path should be odd. This guarantees that the guessed ranking is odd. □

References

[1] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, F. Somenzi, Algebraic decision diagrams and their applications, FMSD 10 (2/3) (1997) 171–206.

[2] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, M. Yoeli, Methodology and system for practical formal verification of reactive hardware, in: Proc. sixth CAV, Lecture Notes in Computer Science, Vol. 818, 1994, pp. 182–193.

[3] R. Bloem, H.N. Gabow, F. Somenzi, An algorithm for strongly connected component analysis in $n \log n$ symbolic steps, in: Proc. FMCAD, Lecture Notes in Computer Science, Vol. 1954, 2000, pp. 37–54.

[4] J.R. Büchi, On a decision method in restricted second order arithmetic, in: Proc. Internat. Congr. Logic, Method and Philosophical Sciences, Vol. 1960, Stanford University Press, Stanford, 1962, pp. 1–12.

[5] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, Symbolic model checking: 1020 states and beyond, Inform. and Comput. 98 (2) (1992) 142–170.

[6] Y. Choueka, Theories of automata on ω-tapes: a simplified approach, J. Comput. System Sci. 8 (1974) 117–141.

[7] E.A. Emerson, C.-L. Lei, Efficient model checking in fragments of the propositional μ-calculus, in: Proc. 1st LICS, 1986, pp. 267–278.

[8] K. Fisler, R. Fraer, G. Kamhi, M.Y. Vardi, Z. Yang, Is there a best symbolic cycle-detection algorithm? in: Proc. seventh TACAS, Lecture Notes in Computer Science, Vol. 2031, 2001, pp. 420–434.

[9] R. Gentilini, C. Piazza, A. Policriti, Computing strongly connected components in a linear number of symbolic steps, in: Proc. 14th SODA, 2003, pp. 573–582.

[10] R. Gerth, D. Peled, M.Y. Vardi, P. Wolper, Simple on-the-fly automatic verification of linear temporal logic, in: Protocol Specification, Testing, and Verification, August 1995, Chapman & Hall, London, pp. 3–18.

[11] G. De Giacomo, M.Y. Vardi, Automata-theoretic approach to planning for temporally extended goals, in: Proc. fifth ECP, Lecture Notes in Artificial Intelligence, Vol. 1809, 2000, pp. 226–238.

[12] S. Gurumurthy, O. Kupferman, F. Somenzi, M.Y. Vardi, On complementing nondeterministic Büchi automata, in: Proc. 12th CHARME, Lecture Notes in Computer Science, 2003.

[13] R.H. Hardin, R.P. Kurshan, S.K. Shukla, M.Y. Vardi, A new heuristic for bad cycle detection using BDDs, in: Proc. ninth CAV, Lecture Notes in Computer Science, Vol. 1254, 1997, pp. 268–278.

[14] R. Hojati, H. Touati, R. Kurshan, R. Brayton, Efficient ω-regula language containment, in: Proc. fourth CAV, Lecture Notes in Computer Science, Vol. 663, 1992.

[15] Y. Kesten, A. Pnueli, L. Raviv, Algorithmic verification of linear temporal logic specifications, in: Proc. 25th ICALP, Lecture Notes in Computer Science, Vol. 1443, 1998, pp. 1–16.

[16] N. Klarlund, Progress measures for complementation of ω-automata with applications to temporal logic, in: Proc. 32nd FOCS, 1991, pp. 358–367.

[17] O. Kupferman, M.Y. Vardi, Weak alternating automata are not that weak, ACM Trans. Comput. Logic 2001 (2) (2001) 408–429.

[18] O. Kupferman, M.Y. Vardi, Model checking of safety properties, Formal Methods System Design 19 (3) (2001) 291–314.

[19] R.P. Kurshan, The complexity of verification, in: 26th STOC, 1994, pp. 365–371.

[20] R.P. Kurshan, Computer Aided Verification of Coordinating Processes, Princeton University Press, Princeton, NJ, 1994.

[21] Z. Manna, A. Pnueli, Specification and verification of concurrent programs by $\forall$-automata, in: Proc. 14th POPL, 1987, pp. 1–12.

[22] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems: Specification, Berlin, January 1992.

[23] Z. Manna, P. Wolper, Synthesis of communicating processes from temporal logic specifications, ACM TOPLAS 6 (1) (1984) 68–93.

[24] S. Merz, Weak alternating automata in Isabelle/HOL, in: Theorem Proving in Higher Order Logics: 13th International Conference, Lecture Notes in Computer Science, Vol. 1869, 2000, pp. 423–440.

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

[26] S. Miyano, T. Hayashi, Alternating finite automata on ω-words, Theoret. Comput. Sci. 32 (1984) 321–330.

[27] K.S. Namjoshi, Certifying model checkers, in: Proc. 13th CAV, Lecture Notes in Computer Science, Vol. 2102, 2001, pp. 2–13.

[28] G.C. Necula, Proof-carrying code, in: Proc. 24th POPL, 1997, pp. 106–119.

[29] D. Peled, A. Pnueli, L.D. Zuck, From falsification to verification, in: Proc. 21st FST&TCS, Lecture Notes in Computer Science, Vol. 2245, 2001, pp. 292–304.

[30] D. Peled, L.D. Zuck, From model checking to a temporal proof. in: Proc. eighth SPIN Workshop on Model Checking of Software, Lecture Notes in Computer Science, Vol. 2057, 2001, pp. 1–14.

[31] K. Ravi, R. Bloem, F. Somenzi, A comparative study of symbolic algorithms for the computation of fair cycles, in: Proc. FMCAD, Lecture Notes in Computer Science, Vol. 1954, 2000, pp. 143–160.

[32] S. Safra, On the complexity of ω-automata, in: 29th FOCS, 1988, pp. 319–327.

[33] A.P. Sistla, Safety, liveness and fairness in temporal logic, Formal Aspects Comput. 6 (1994) 495–511.

[34] A.P. Sistla, M.Y. Vardi, P. Wolper, The complementation problem for Büchi automata with applications to temporal logic, Theoret. Comput. Sci. 49 (1987) 217–237.

[35] F. Somenzi, R. Bloem, Efficient Büchi automata from LTL formulae, in: Proc. 12th CAV, Lecture Notes in Computer Science, Vol. 1855, 2000, pp. 248–263.

[36] S. Tasiran, R. Hojati, R.K. Brayton, Language containment using non-deterministic ω-automata, in: Proc. eighth CHARME, Lecture Notes in Computer Science, Vol. 987, 1995, pp. 261–277.

[37] M.Y. Vardi, P. Wolper, Reasoning about infinite computations, Inform. and Comput. 115 (1) (1994) 1–37.

[38] P. Wolper, Temporal logic can be more expressive, Inform. and Control 56 (1–2) (1983) 72–99.