On the Complexity of Parity Word Automata

Valerie King[1]* and Orna Kupferman[2] and Moshe Y. Vardi[3]**

[1] University of Victoria
Department of Computer Science, P.O. Box 3055, Victoria, BC, Canada V8W 3P6
Email: val@csr.csc.uvic.ca, URL: http://www.csr.uvic.ca/~val
[2] Hebrew University
School of Computer Science and Engineering, Jerusalem 91904, Israel
Email: orna@cs.huji.ac.il, URL: http://www.cs.huji.ac.il/~orna
[3] Rice University
Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email: vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi

Abstract. Different types of nondeterministic automata on infinite words differ in their succinctness and in the complexity for their nonemptiness problem. A simple translation of a parity automaton to an equivalent Büchi automaton is quadratic: a parity automaton with n states, m transitions, and index k may result in a Büchi automaton of size $O((n+m)k)$. The best known algorithm for the nonemptiness problem of parity automata goes through Büchi automata, leading to a complexity of $O((n+m)k)$. In this paper we show that while the translation of parity automata to Büchi automata cannot be improved, the special structure of the acceptance condition of parity automata can be used in order to solve the nonemptiness problem directly, with a dynamic graph algorithm of complexity $O((n+m)\log k)$.

1 Introduction

Today's rapid development of complex and safety-critical systems requires reliable verification methods. The automata-theoretic approach to system verification uses the theory of automata on infinite objects [Büc62,McN66,Rab69] as a unifying paradigm for the specification, verification, and synthesis of nonterminating systems. By translating specifications to automata, we reduce questions about systems and their specifications to questions about automata. More specifically, questions such as satisfiability of specifications and correctness of systems with respect to their specifications are reduced to questions such as nonemptiness and language containment [VW86,Kur94,VW94]. The automata-theoretic approach separates the logical and the algorithmic aspects of reasoning about systems. The translation of specifications to automata handles the logic and shifts all the algorithmic difficulties to automata-theoretic problems.

---

* This work was done while the author was visiting the Hebrew University.
** Supported in part by NSF grants CCR-9700061 and CCR-9988322, and by a grant from the Intel Corporation.

Like automata on finite words, automata on infinite words either accept or reject an input word. Since a run on an infinite word does not have a final state, acceptance is determined with respect to the set of states visited infinitely often during the run. For example, in *Büchi* automata [Büc62], some of the states are designated as accepting states, and a run is accepting if it visits states from the accepting set infinitely often. In *parity* automata [Mos84,EJ91], the acceptance condition is a partition $\{F_1, F_2, \ldots, F_{2k}\}$ of the state space, and a run is accepting if the minimal index i for which the set F_i is visited infinitely often is even. In *Rabin* automata [Rab69], the acceptance condition is a set $\{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$ of pairs of sets of states, and a run is accepting if there is a pair $\langle G_i, B_i\rangle$ for which the set G_i is visited visited infinitely often and the set B_i is visited only finitely often. The number k appearing in the acceptance condition of parity and Rabin automata is called the *index* of the automaton.

The type of the automaton influences its succinctness. For example, a simple transformation of a parity or a Rabin nondeterministic automaton with n states, m transitions, and index k to an equivalent nondeterministic Büchi automaton composes k copies of the automaton, and thus results in an automaton of size $O((n+m)k)$. The type of the automaton also influences the difficulty of answering questions about it. In particular, while the nonemptiness problem for nondeterministic Büchi automata can be easily reduced to the reachability problem (the automaton is nonempty iff there is an accepting state reachable from an initial state and from itself), and can therefore be solved in linear time or NLOGSPACE, there is no straightforward linear-time solution to the nonemptiness problem for parity and Rabin nondeterministic automata. An algorithm that first translates the parity or Rabin automaton to an equivalent Büchi automaton has time complexity $O((n+m)k)$, which is also the best known complexity for the problem. In particular, when $k = n$, the above implies that while the nonemptiness problem for Büchi word automata can be solved in linear time, the best known algorithm for parity automata is quadratic.

Parity automata are particularly useful in the context of verification. This follows from the fact that the parity acceptance condition can naturally recognize languages given by fixed-point expressions [Koz83,EJ91], and many properties of systems are naturally specified by means of fixed points. This is especially relevant for the branching case, where alternating parity tree automata are exactly as expressive as the μ-calculus [EJ91,JW95]. The parity acceptance condition can be viewed as a special case of the Rabin acceptance condition. Indeed, a parity condition $\{F_1, F_2, \ldots, F_{2k}\}$ is equivalent to a Rabin condition $\{\langle F_2, F_1\rangle, \langle F_4, F_1 \cup F_2 \cup F_3\rangle, \ldots, \langle F_{2k}, F_1 \cup \ldots \cup F_{2k-1}\rangle\}$. Some algorithms for parity automata use this equivalence and do not try to make use of the fact that the parity condition has much more structure in it. The question whether we can somehow exploit this structure is important and interesting, and is the key to some fundamental questions in automata and verification. In particular, while the nonemptiness problem for nondeterministic Rabin tree automata is

NP-complete [EJ88], we know that for parity automata the problem is in UP $\cap$ co-UP [Jur98][1], and its precise complexity is an open problem.

It is shown in [SN99] that the blow up in the translation of parity automata to Büchi automata cannot be avoided. For that, Seidl and Niwiński", show a family $\mathcal{L}_n$ of languages such that $\mathcal{L}_n$ can be recognized by a parity automaton with n states, $2n+1$ transitions, and index n, yet the smallest Büchi automaton for $\mathcal{L}_n$ has $O(n^2)$ states and transitions. It follows that nonemptiness algorithms for parity automata that use a translation to Büchi automata cannot be improved.

In this paper we show that while the special structure of parity automata does not prevent them from defining rich languages succinctly, it is helpful when the nonemptiness problem is considered. We present an algorithm that circumvent the translation to Büchi automata: given a parity automaton with n states, m transitions, and index k, our algorithm solves the nonemptiness problem in time complexity $O((n+m)\log k)$. This improves the $O((n+m)k)$ known algorithm.

As detailed in Section 3, our algorithm is a variation of an algorithm for hierarchical clustering [Tar82], and is heavily based on the special structure of the acceptance condition of parity automata. In particular, it cannot be extended to Rabin automata. The algorithm handles a known-in-advance sequence of insertions of edges, thus it is an *off-line partially dynamic* algorithm.

While parity word automata do not have the same practical importance as parity tree automata, we believe that our results are interesting, from both a theoretical and practical points of view: this is not the first time that dynamic graph algorithms are used to improve the complexity of a nonemptiness problem of automata on infinite words. In [HT96], Henzinger and Telle developed an algorithm, called *lock-step search*, for the maintenance of the strongly connected components of a directed graph under edge deletion, and use the algorithm in order to improve the complexity of the nonemptiness problem for Streett automata. A dynamic algorithm was recently used in [BGS00] in order to find and analyze the strongly connected components of a graph of size n with $O(n \log n)$ symbolic steps, improving the quadratic complexity of this highly practical problem. So, we believe that dynamic graph algorithms can be used further to improve the complexity of problems from automata theory and verification. Creating a class of problems where dynamic graph algorithms have proven to be useful has clear practical importance. In particular, the same ideas used in our algorithm may be useful in the branching case.

2 Preliminaries

Given an alphabet Σ, an *infinite word over* Σ is an infinite sequence $w = \sigma_0 \cdot \sigma_1 \cdot \sigma_2 \cdots$ of letters in Σ. An *automaton over infinite words* (word automaton, for short) is $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and α is an acceptance condition (a condition that defines a subset of Q^ω;

[1] The class UP is a subset of NP, where each word accepted by the Turing machine has a unique accepting run.

we define several acceptance conditions below). Intuitively, $\delta(q,\sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ. Since $\mathcal{A}$ may have several initial states and since the transition function may specify many possible transitions for each state and letter, $\mathcal{A}$ may be *nondeterministic*.

Given an input infinite word $w = \sigma_0 \cdot \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$, a *run* of $\mathcal{A}$ on w can be viewed as a function $r : \mathbb{N} \to Q$ where $r(0) \in Q_0$ (i.e., the run starts in one of the initial states) and for every $i \geq 0$, we have $r(i+1) \in \delta(r(i), \sigma_i)$ (i.e., the run obeys the transition function). Each run r induces a set $inf(r)$ of states that r visits *infinitely often*. Formally,

$$inf(r) = \{q \in Q \ : \ \text{for infinitely many } i \in \mathbb{N}, \ \text{we have } r(i) = q\}.$$

As Q is finite, it is guaranteed that $inf(r) \neq \emptyset$. The run r *accepts* w iff it satisfies the acceptance condition α. We consider here three acceptance conditions.

- A run r satisfies a *Büchi* acceptance condition $\alpha \subseteq Q$ if and only if $inf(r) \cap \alpha \neq \emptyset$.
- A run r satisfies a *parity* acceptance condition $\alpha = \{F_1, F_2, \ldots, F_{2k}\}$, where the F_i's are a partition of Q, iff the minimal index i for which $inf(r) \cap F_i \neq \emptyset$ is even. For $1 \leq i \leq 2k$, we use the notation $\tilde{F_i} = F_1 \cup F_2 \cup \ldots F_i$. Note that $\tilde{F_1} \subseteq \tilde{F_2} \subseteq \cdots \subseteq \tilde{F_{2k}} = Q$.
- A run r satisfies a *Rabin* acceptance condition $\alpha = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$, where for $1 \leq i \leq k$, $G_i \subseteq Q$ and $B_i \subseteq Q$, if and only if there exists a pair $\langle G_i, B_i\rangle \in \alpha$ for which $inf(r) \cap G_i \neq \emptyset$ and $inf(r) \cap B_i = \emptyset$.

The number k appearing in a parity or Rabin condition is called the *index* of the automaton. Note that the acceptance condition of an (either a parity or Rabin) automaton with index k involves $2k$ sets. An automaton $\mathcal{A}$ accepts an input word w iff there exists a run r of $\mathcal{A}$ on w such that r accepts w. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of infinite words that $\mathcal{A}$ accepts. Thus, each word automaton defines a subset of Σ^ω. We say that an automaton $\mathcal{A}$ is *nonempty* iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$. For simplicity, we assume that all the states in the automaton are *reachable*; that is, for every state q, there is a word w and a run r of $\mathcal{A}$ on w such that r visits q. We also assume that no state is *redundant*; that is, for every state q, there is a word w and a run r of $\mathcal{A}$ on w such that r accepts w in a run that visits q. Note that we can omit states that are redundant or not reachable and get an equivalent smaller automaton.

A *labeled directed graph* $G = \langle D, V, E, l\rangle$ consists of a domain D, a set V of vertices, a set $E \subseteq V \times V$ of edges, and a labeling function $l : V \to D$ that maps each vertex to a value in D. We assume that $D = \{1, \ldots, n\}$ for some $n \geq 1$. A *path* in G is a sequence $v_1, v_2, \ldots, v_n$ of vertices such that $n \geq 1$ and $\langle v_i, v_{i+1}\rangle \in E$ for all $i \geq 1$. We say that vertex v is reachable from vertex v' iff there is a path $v_1, v_2, \ldots, v_n$ in G with $v' = v_1$ and $v = v_n$. A *cycle* is a path $v_1, v_2, \ldots, v_n$ with $n \geq 2$ and $v_1 = v_n$. A *strongly connected component* (SCC) of G is a set of states $C \subseteq V$ such that for all v and v' in C, the vertex v is reachable from v'. An SCC C is *maximal* (MSCC) if for all vertices $v \notin C$, the set $C \cup \{v\}$ is no longer a SCC. An SCC is *nontrivial* if it contains a cycle. Note that a single vertex with a self loop is a nontrivial SCC. Each graph G has a unique partition into MSCC. By [Tar72], this partition can be found in time $O(|V|+|E|)$.

With each automaton $\mathcal{A}=\langle \Sigma, Q, \delta, Q_0, \alpha\rangle$, we can associate a graph $G_{\mathcal{A}}=\langle D_\alpha, Q, E_\delta, l_\alpha\rangle$, where for $q,q'\in Q$, we have $E_\delta(q,q')$ if there is $\sigma\in\Sigma$ such that $q'\in\delta(q,\sigma)$. The labels of the graph encodes the acceptance condition α. For example, if $\mathcal{A}$ is a Büchi automaton, we can have $D_\alpha=\{1,2\}$ and $l_\alpha(q)=1$ iff $q\in\alpha$. Similarly, if $\mathcal{A}$ is a parity automaton with index k, we can have $D_\alpha=\{1,\ldots,2k\}$ and $l_\alpha(q)=i$ for the i such that $q\in F_i$. When we refer to the number of states and edges of an automaton $\mathcal{A}$, we mean $|Q|$ and $|E_\delta|$.

Theorem 1. [CES86] *The nonemptiness problem for Büchi automata can be solved in linear time.*

Proof: It is easy to see that a Büchi automaton $\mathcal{A}$ is nonempty iff there is a state $q\in\alpha$ that is reachable in $\mathcal{A}$ from Q_0 and from itself. First, such a state q witnesses an accepting run of $\mathcal{A}$ that first leads from Q_0 to q and then visits q infinitely often. Also, if some word is accepted by $\mathcal{A}$, then there must be a run of $\mathcal{A}$ that visits infinitely often some state in α that is reachable from Q_0. Being visited more than once, this state is also reachable from itself. We can test for the existence of q as above by partitioning the graph $G_{\mathcal{A}}$ into MSCC's and looking for a component that is reachable from Q_0 and whose intersection with α is not empty. $\square$

Note that by guessing a state q as in the proof of Theorem 1 and performing two reachability tests, the nonemptiness problem for Büchi automata can also be solved in NLOGSPACE, and is in fact NLOGSPACE-complete [VW94]. In this paper, however, we study the time complexity of the nonemptiness problem.

It is easy to see that a Büchi acceptance condition α is equivalent to a parity condition $\{\emptyset,\alpha\}$ and that a parity condition $\{F_1,F_2,\ldots,F_{2k}\}$ is equivalent to the Rabin condition $\{\langle F_2,\tilde{F}_1\rangle,\ldots,\langle F_{2k},\tilde{F}_{2k-1}\rangle\}$. It follows that one can easily translate a given Büchi automaton to an equivalent parity automaton, and can translate a given parity automaton to an equivalent Rabin automaton. It turns out that for nondeterministic automata, translations are possible also in the other directions. We describe here the translations of Rabin and parity nondeterministic automata to Büchi nondeterministic automata.

Theorem 2. [Cho74] *Given a Rabin automaton $\mathcal{A}$ with n states, m edges, and index k, there is a Büchi automaton $\mathcal{A}'$ with $O(nk)$ states and $O(mk)$ edges such that $\mathcal{L}(\mathcal{A}')=\mathcal{L}(\mathcal{A})$.*

Proof: Let $\mathcal{A}=\langle\Sigma,Q,\delta,Q_0,\alpha\rangle$ with $\alpha=\{\langle G_1,B_1\rangle,\ldots,\langle G_k,B_k\rangle\}$. For every $1\le i\le k$, let $Q_i=(Q\setminus B_i)\times\{i\}$. We define the Büchi automaton $\mathcal{A}'=\langle\Sigma,Q',\delta',Q_0,\alpha'\rangle$, where

- $Q'=Q\cup\bigcup_{1\le i\le k}Q_i$.
- For every $q\in Q$ and $\sigma\in\Sigma$, we have

- $\delta'(q,\sigma) = \delta(q,\sigma) \cup \bigcup_{1 \leq i \leq k}((\delta(q,\sigma) \setminus B_i) \times \{i\})$.
- For every $1 \leq i \leq k$, we have $\delta'(\langle q, i\rangle, \sigma) = (\delta(q,\sigma) \setminus B_i) \times \{i\}$.

– $\alpha' = \bigcup_{1 \leq i \leq k} G_i \times \{i\}$.

Thus, $\mathcal{A}'$ consists of $k+1$ copies of $\mathcal{A}$. One copy ("the initial copy") is full and it contains all the states in Q. Then, k copies are partial: every such copy is associated with a pair $\langle G_i, B_i\rangle$, its states are labeled i, and it contains all the states in $Q \setminus B_i$. A run of $\mathcal{A}'$ starts at the initial copy. The run can nondeterministically choose between staying in the initial copy or moving to one of the other k copies. Once a run of $\mathcal{A}'$ moves to a copy associated with the i'th pair, it cannot visit states from B_i. Indeed, Q_i does not contain such states. The acceptance condition of $\mathcal{A}'$ guarantees that an accepting run eventually leaves the initial copy and moves to some Q_i, where it visits infinitely many states from G_i. □

Since a parity acceptance condition can be translated to a Rabin acceptance condition, Theorem 2 implies the following theorem.

Theorem 3. *Given a parity automaton $\mathcal{A}$ with n states, m edges, and index k, there is a Büchi automaton $\mathcal{A}'$ with $O(nk)$ states and $O(mk)$ edges such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Theorem 1 together with Theorems 2 and 3 imply an $O((n+m)k)$ solution to the nonemptiness problem for Rabin and parity automata with n states, m edges, and index k. In this paper we show that while the blow up in the translation of parity and Rabin automata to Büchi automata cannot be avoided [SN99], one can solve the nonemptiness problem for a parity automaton with n states, m edges, and index k, in time $O((n+m)\log k)$.

3 An Efficient Solution to the Nonemptiness Problem

In this section we present an algorithm of running time $O((n+m)\log k)$ for solving the nonemptiness problem for a parity automaton with n states, m edges, and index k. The idea of the algorithm is as follows. Consider a parity automaton $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha\rangle$, with $\alpha = \{F_1, \ldots, F_{2k}\}$. It is easy to see that the automaton $\mathcal{A}$ is nonempty if there is $1 \leq i \leq k$ such that the graph G_i obtained from $\mathcal{A}$ by omitting states in $F_1 \cup \ldots \cup F_{2i-1}$ contains a MSCC with a node in F_{2i}. Since the graph G_i is contained in the graph G_{i-1}, the search for a candidate i, if starts from $i = k$ and proceeds to $i = 1$, involves a sequence of increasing graphs. Proceeding from G_i to G_{i-1}, we need to calculate the MSCC's of G_{i-1}. The MSCC's of G_i *refine* these of G_{i-1}, in the sense that if two vertices belong to the same MSCC in G_i, they also belong to the same MSCC in G_{i-1}. Once we need to calculate the MSCC's of G_{i-1}, we have already calculated these of G_i. So, there is a hope that we can do better than calculating the MSCC's of G_{i-1} from scratch, which is indeed what our algorithm does.

Our algorithm is a variation of an algorithm for hierarchical clustering [Tar82]. Consider a directed graph with m weighted edges and n vertices. A *hierarchical* *decomposition* of the graph is a process in which the graph's edges are added one at a time in order of weight. Tarjan's algorithm constructs a decomposition tree whose leaves are the vertices of the graph and whose internal nodes are associated with MSCC's that are formed as the hierarchical-decomposition process proceeds. Thus, the children of a node associated with a component C are the subcomponents that coalesce to form C. The running time of the algorithm is $O(m \log n)$. Our variation of the algorithm corresponds to the special case of the graphs G_i described above: it inserts edges in batches, rather than one at a time, and it simplifies steps and data structures that are not relevant to the nonemptiness check. In particular, once the algorithm detects a nontrivial MSCC in which the minimal label of all vertices is even, it terminates with a positive reply.

The formal description of the algorithm is below, given in terms of the labeled directed graph that corresponds to $\mathcal{A}$. Given a labeled directed graph $G = \langle D, V, E, l \rangle$ with $D \subseteq \{1, 2, 3, \ldots, 2k\}$, the *even-cycle problem* is to determine whether there is a cycle C in G such that $\min_{v \in C}\{l(v)\}$ is even. It is easy to see that the nonemptiness problem for a parity automaton $\mathcal{A}$ is equivalent to the even-cycle problem for $G_{\mathcal{A}}$. Indeed, an accepting run of $\mathcal{A}$ induces an even cycle, and vice versa. Also, recall that we assume that all the states in $\mathcal{A}$, and hence also in $G_{\mathcal{A}}$, are reachable; thus we do not have to worry about the cycle being reachable.

Theorem 4. *Let $G = \langle \{1, 2, 3, \ldots, 2k\}, V, E, l \rangle$ be a labeled directed graph with $|V| = n$ and $|E| = m$. The even-cycle problem for G can be solved in time $O((n + m) \log k)$.*

Proof: For $i \in \{1, \ldots, 2k\}$, let $V_i \subseteq V$ be the set of vertices v with $l(v) \geq i$, and let $E_i = E \cap (V_i \times V_i)$. Thus, E_i contains only edges whose both endpoints have labels greater than or equal to i. Then, let $G_i = \langle \{i, \ldots, 2k\}, V_i, E_i, l \rangle$ be G when restricted to vertices and edges in V_i and E_i.

We first claim that the answer to the even-cycle problem is positive iff there is some even $i \in \{1, \ldots, 2k\}$ and a nontrivial MSCC of G_i that contains a vertex labeled i. Indeed, since $\min_{v \in V_i}\{l(v)\} \geq i$, such an MSCC witnesses a cycle C in G for which $\min_{v \in C}\{l(v)\} = i$, which is even. Also, if there is a cycle C in G such that $\min_{v \in C}\{l(v)\}$ is some even i, this cycle belongs to a nontrivial MSCC of G_i. We call i a *witness* for G.

For all i and j with $i < j$, we have that $V_j \subseteq V_i$, thus the graph G_j is a subgraph of G_i. Consequently, the MSCC's of G_j *refine* these of G_i, in the sense that if two vertices belong to the same MSCC in G_j, they also belong to the same MSCC in G_i. This refinement is the key to our algorithm. Following the analysis above, the algorithm searches for $i \in \{1, \ldots, 2k\}$ such that i is even and there is a nontrivial MSCC of G_i that contains a vertex labeled i. Intuitively, when the algorithm examines a candidate i, it either terminates with "Yes", in case i is a witness, or examine further candidates $j \neq i$. We distinguish between two cases. If $j > i$, the examination of j proceeds with a subgraph of G_i, consisting of vertices and edges that belong to the nontrivial MSCC's of G_j. If $j < i$, the examination of j proceeds with a *compressed version* of G_j, in which vertices that belong to the same MSCC of G_i are represented by a single vertex. Consequently, the graphs we consider have fewer states and edges. More precisely, for all $j_1 > i$ and $j_2 < i$, the number of edges in the two graphs required for checking j_1 and j_2 is not greater than the number of edges in the graph required for checking i.

Formally, we solve the even-cycle problem by the recursive routine $\mathbf{solve}(\langle D, V, E, l\rangle, i, j)$ described below. The routine gets as input a labeled directed graph and two indices i and j in D. The routine outputs "Yes" if there is a witness in $\{i, \ldots, j\}$ for the even-cycle problem for $\langle D, V, E, l\rangle$, and outputs "No" otherwise. To solve the even-cycle problem for a given graph G, we call $\mathbf{solve}(G, 1, 2k)$. The idea behind $\mathbf{solve}(G, i, j)$ is as follows. Given G and $i \le j$ (if $j < i$, no witness $i \le l \le j$ exists and the routine terminates with "No"), the routine first finds the MSCC's of G_{mid}, where mid is an index midway between i and j. If mid is even and there is a nontrivial MSCC of G_{mid} with a vertex labeled mid, the routine stops and returns "Yes" (mid is the witness). Otherwise, we call $\mathbf{solve}(G_\perp, mid+1, j)$ and $\mathbf{solve}(G_\top, i, mid-1)$, where the call for $\mathbf{solve}(G_\perp, mid+1, j)$ searches for a witness in $\{mid+1, \ldots, j\}$ and $G_\perp$ is a subgraph of G_{mid}, and the call for $\mathbf{solve}(G_\top, i, mid-1)$ searches for a witness in $\{i, \ldots, mid-1\}$ and $G_\top$ is a compressed version of G, in which a set of vertices that belong to the same MSCC of G_{mid} is represented by a single vertex.

procedure $\mathbf{solve}(\langle D, V, E, l\rangle, i, j)$

1. If $j < i$, return "No" and STOP.
2. $mid := \lceil \frac{i+j}{2} \rceil$.
3. (a) $V_{mid} := \{v \in V \mid l(v) \ge mid\}$.
   (b) $E_{mid} := E \cap (V_{mid} \times V_{mid})$.
4. Find the MSCC's of $G_{mid} = \langle D, V, E_{mid}, l\rangle$.
   For each $v \in V$, let $C_{mid}(v)$ denote the MSCC containing v in G_{mid}, and let $\mathcal{C}_{mid}$ denote the set of all (trivial and nontrivial) MSCC's in G_{mid}. Note that each vertex $v \in V \setminus V_{mid}$ induces the trivial MSCC $\{v\} \in \mathcal{C}_{mid}$.
5. If mid is even and there is $v \in V_{mid}$ such that $l(v) = mid$ and $C_{mid}(v)$ is nontrivial, output "Yes" and STOP.
6. (*Search for a witness in* $\{mid+1, \ldots, j\}$.)
   (a) $V_\perp := \{v \in V_{mid} \mid l(v) \ge mid+1 \text{ and } C_{mid}(v) \text{ is nontrivial }\}$.
   (b) $E_\perp := \{\langle u, v\rangle \in E_{mid} \cap (V_\perp \times V_\perp) \mid C_{mid}(u) = C_{mid}(v)\}$.
   (c) $\mathbf{solve}(\langle D, V_\perp, E_\perp, l\rangle, mid+1, j)$.
7. (*Search for a witness in* $\{i, \ldots, mid-1\}$.)
   (a) $V_\top := \mathcal{C}_{mid}$.
   (b) $E_\top := \{\langle C_{mid}(u), C_{mid}(v)\rangle \mid \langle u, v\rangle \in E \text{ and } (\langle u, v\rangle \notin E_{mid} \text{ or } C_{mid}(u) \neq C_{mid}(v))\}$.
   (c) For $C \in \mathcal{C}_{mid}$, define $l_\top(C) = \min_{v \in C}\{l(v)\}$.
   (d) $\mathbf{solve}(\langle D, V_\top, E_\top, l_\top\rangle, i, mid-1)$.

Let us explain Steps (6) and (7) in more detail. In Step (6), the graph $G_\perp$ removes an edge from G_{mid} if the edge has a vertex labeled mid or if it connects vertices of different MSCCs. Indeed, such edges cannot participate in an MSCC that witnesses a cycle C for which $\min_{v\in C}\{l(v)\}$ is in $mid+1,\ldots,j$. In Step (7), the vertices of $G_\top$ are the MSCC's of G_{mid} and there is an edge between two MSCC's C_u and C_v if there are vertices $u \in C_u$ and $v \in C_v$ such that u or v are not in V_{mid} (in which case the corresponding MSCC is a singleton and is trivial) and $\langle u,v\rangle \in E$, or both u and v are in V_{mid} and they belong to different MSCC's (in which case $C_u \neq C_v$). Indeed, searching for a witness in $i,\ldots,mid-1$, we can ignore the internal structure of MSCC's in $\mathcal{C}_{mid}$.

We now prove formally that for every graph G and indices i and j, the procedure **solve**(G,i,j) outputs "Yes" iff there is an even $i \leq w \leq j$ and there is a cycle C in G such that w is the minimal label in C. Since the initial call is to **solve**$(G,1,2k)$ and all the cycles in G are reachable, this implies the correctness of the algorithm. The proof proceeds by induction on $j-i$. When $j-i<0$, thus $j<i$, no $i \leq w \leq j$ exists, and the procedure indeed outputs "No". For the induction step, assume that the correctness claim holds for $j-i \leq k$, and consider the case $j-i = k+1$. Let $mid = \lceil \frac{i+j}{2} \rceil$. Clearly, there is an even $i \leq w \leq j$ and there is a cycle C in G such that w is the minimal label in C iff one of the following holds:

(a) mid is even and there is a cycle C in G such that mid is the minimal label in C,
(b) there is an even $mid+1 \leq w \leq j$ and there is a cycle C in G such that w is the minimal label in C, or
(c) there is an even $i \leq w \leq mid-1$ and there is a cycle C in G such that w is the minimal label in C.

We prove that **solve**(G,i,j) outputs "Yes" in Steps (5), (6), or (7), iff (a), (b), or (c) holds, respectively.

- Assume that (a) holds. Then, the cycle C exists in G_{mid}, the conditions in Step (5) are satisfied, and the algorithm outputs "Yes". Assume now that the algorithm outputs "Yes" is Step (5). Then, mid is even and there is a nontrivial MSCC in G_{mid} that contains a vertex labeled mid. By the definition of G_{mid}, this implies the existence of a cycle C in G such that mid is the minimal label in C. Hence, (a) holds.
- Assume that (b) holds. Then, the cycle C exists in $\langle D, V_\perp, E_\perp, l\rangle$. Indeed, since the minimal index in C is $mid+1 \leq w \leq j$, the vertices and edges that are removed from G do not participate in C. So, by the induction hypothesis, the algorithm outputs "Yes" in Step (6). Assume that the algorithm outputs "Yes" in Step (6). Then, by the induction hypothesis, there is an even $mid+1 \leq w \leq j$ and there is a cycle C in $\langle D, V_\perp, E_\perp, l\rangle$ such that w is the minimal label in C. Since the cycle C is also a cycle of G, it follows that (b) holds.
- Assume that (c) holds. Let $i \leq w \leq mid-1$ and $C = v_1, v_2, \ldots, v_n$ be such that w is even and C is a circle in G for which w is the minimal label in C (that is, $v_1 = v_n$). Let $C' = C_{mid}(v_1), C_{mid}(v_2), \ldots, C_{mid}(v_n)$ be the sequence of MSCCs in $\mathcal{C}_{mid}$ induced by C. For all $1 \leq k \leq n-1$, we have that $C_{mid}(v_k) = C_{mid}(v_{k+1})$ or $E_\top(C_{mid}(v_k), C_{mid}(v_{k+1}))$. Hence, the

sequence obtained from C' by omitting successive repetitions of the same MSCC in $\mathcal{C}_{mid}$ is a cycle $C_\top$ in $\langle D, V_\top, E_\top, l_\top\rangle$. Since the minimal label in C is w, then, by definition of $l_\top$, the same holds for $C_\top$. So, by the induction hypothesis, the algorithm outputs "Yes" in Step (7). Assume that the algorithm outputs "Yes" in Step (7). Then, by the induction hypothesis, there is an even $i \leq w \leq mid - 1$ and there is a cycle $C_\top$ in $\langle D, V_\top, E_\top, l_\top\rangle$ such that w is the minimal label in $C_\top$. By replacing each vertex in $C_\top$ (which is a MSCC in G) by the appropriate path in G, we can get from $C_\top$ the cycle required for (b) to hold.

We now analyze the time complexity of the algorithm. First, note that since all the states in the graph G are reachable, then all the intermediate graphs $G_\perp$ and $G_\top$ created by the procedure are such that $|E_\perp| \geq |V_\perp|$ and $|E_\top| \geq |V_\top|$. Also, note that for every graph G, each edge $\langle u, v\rangle \in E$ contributes an edge to at most one of $E_\perp$ and $E_\top$. That is, $|E_\perp| + |E_\top| \leq |E|$. Indeed, an edge $\langle u, v\rangle \in E$ is in $E_\perp$ only if $\langle u, v\rangle \in E_{mid}$ and $C_{mid}(u) = C_{mid}(v)$, and it contributes an edge to $E_\top$ only if it satisfies the complementary condition, namely $\langle u, v\rangle \notin E_{mid}$ or $C_{mid}(u) \neq C_{mid}(v)$.

Let $R(s, d)$ denote the cost of **solve**$(\langle D, V, E, l\rangle, i, j)$, where $s = |E|$ and $d = j - i + 1$ (i.e., d is the size of the set $\{i, \ldots, j\}$ within which a witness is searched). Since we can find the MSCC's of a graph in time $O(|V| + |E|)$ and $s = |E| \geq |V|$, the cost of steps (1-5) of the algorithm is $O(s)$, thus the cost of **solve**(G, i, j) is $O(s)$ plus the cost of the two recursive calls to **solve** in steps (6) and (7). Hence, we can describe $R(s, d)$, for all $s \geq 0$, by a recurrence as follows.

- $R(s, 0) = O(1)$.
- For all $d \geq 1$, we have $R(s, d) = O(s) + R(s_\perp, \lfloor\frac{d-1}{2}\rfloor) + R(s_\top, \lceil\frac{d-1}{2}\rceil)$, with $s_\perp + s_\top \leq s$.

It is easy to see that $R(s, 1) = R(s, 2) = O(s)$. We prove that for all $d > 2$, we have $R(s, d) = O(s \log d)$. In particular, the cost of **solve**$(\langle D, V, E, l\rangle, 1, 2k)$ is $O((|V| + |E|) \log k)$.

Let b be a constant such that for all s, we have $R(s, 2) \leq bs$. Note that the cost of steps (1-5) is then at most bs. We prove that there is $c \geq b$ such that for all $d \geq 2$, we have that $R(s, d) \leq cs \log_2 d$, thus $R(s, d) = O(s \log_2 d)$. The proof proceeds by an induction on d. Recall that $R(s, 2) \leq bs$, hence $R(s, 2) \leq cs \log_2 2$, and the induction claim holds for the base $d = 2$. Now, assume that for all $2 \leq d' \leq d$, we have that $R(s, d') \leq cs \log_2 d'$. By the recurrence above, $R(s, d+1) \leq cs + R(s_\perp, \lfloor\frac{d}{2}\rfloor) + R(s_\top, \lceil\frac{d}{2}\rceil)$, for $s_\perp$ and $s_\top$ with $s_\perp + s_\top \leq s$. Then, by the induction hypothesis, $R(s, d+1) \leq cs + cs_\perp \log_2\lfloor\frac{d}{2}\rfloor + cs_\top \log_2\lceil\frac{d}{2}\rceil \leq cs(1 + \log_2\lceil\frac{d}{2}\rceil) \leq cs \log_2(d + 1)$, and we are done.

□

References

[BGS00] R. Bloem, H.N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. In *Formal Methods in Computer Aided Design*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

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

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

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

[HT96] M. Henzinger and J.A. Telle. Faster algorithms for the nonemptiness of Streett automata and for communication protocol pruning. In *Proc. 5th Scandinavian Workshop on Algorithm Theory*, volume 1097 of *Lecture Notes in Computer Science*, pages 10–20. Springer-Verlag, 1996.

[Jur98] M. Jurdzinski. Deciding the winner in parity games is in up $\cap$ co-up. *Information Processing Letters*, 68(3):119–124, 1998.

[JW95] D. Janin and I. Walukiewicz. Automata for the modal μ-calculus and related results. In *Proc. 20th International Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 552–562. Springer-Verlag, 1995.

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

[Kur94] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes.* Princeton Univ. Press, 1994.

[McN66] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

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

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

[SN99] H. Seidl and D. Niwiński. On distributive fixed-point expressions. *Theoretical Informatics and Applications*, 33(4–5):427–446, 1999.

[Tar72] R.E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.

[Tar82] R.E. Tarjan. A hierarchical clustering algorithm using strong components. *Information Processing Letters*, 14:26–29, 1982.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.

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