Finding Shortest Witnesses to the Nonemptiness of Automata on Infinite Words

Orna Kupferman and Sarai Sheinvald-Faragy

Hebrew University, School of Engineering and Computer Science, Jerusalem 91904, Israel
Email: {orna,surke}@cs.huji.ac.il

Abstract. In the automata-theoretic approach to formal verification, the satisfiability and the model-checking problems for linear temporal logics are reduced to the nonemptiness problem of automata on infinite words. Modifying the nonemptiness algorithm to return a shortest witness to the nonemptiness (that is, a word of the form uv^ω that is accepted by the automaton and for which $|uv|$ is minimal) has applications in synthesis and counterexample analysis. Unlike shortest accepting runs, which have been studied in the literature, the definition of shortest witnesses is semantic and is independent on the specification formalism of the property or the system. In particular, its robustness makes it appropriate for analyzing counterexamples of concurrent systems.

We study the problem of finding shortest witnesses in automata with various types of concurrency. We show that while finding shortest witnesses is more complex than just checking nonemptiness in the nondeterministic and in the concurrent models of computation, it is not more complex in the alternating model. It follows that when the system is the composition of concurrent components, finding a shortest counterexample to its correctness is not harder than finding some counterexample. Our results give a computational motivation to translating temporal logic formulas to alternating automata, rather than going all the way to nondeterministic automata.

1 Introduction

The automata-theoretic approach to formal verification uses the theory of automata as a unifying paradigm for system specification, verification, and synthesis. Two fundamental problems in formal verification are reduced to the nonemptiness problem of automata on infinite words: the *satisfiability* problem for a linear temporal logic (LTL) formula ψ is reduced to the nonemptiness problem of an automaton $\mathcal{A}_\psi$ that accepts exactly all the computations that satisfy ψ [5, 35], and the *model-checking* problem of a system $\mathcal{S}$ with respect to ψ is reduced to checking the emptiness of the product of $\mathcal{S}$ with an automaton $\mathcal{A}_{\neg\psi}$ that accepts exactly all the computations that violate ψ [35]. Verification methods based on these reductions have been implemented in both academic and industrial automated-verification tools.

Modifying the nonemptiness algorithm to return a *witness* to the nonemptiness of the automaton does not involve an additional computational price and is beneficial in both applications: in the case of satisfiability, the witness is a computation that satisfies the formula. In particular, when the formula describes the desired behavior of a closed

system, finding a witness amounts to *synthesis* [6]. In the case of model checking, a witness points to a computation of the system that violates the specification, and it helps the user to detect errors in the system. Further applications of witnesses exist in *abstraction*, where refinement is directed by an analysis of counterexamples [7], and *vacuity*, where a positive answer from the model checker is accompanied by a trace in which the specification is satisfied non-vacuously [24].

A witness to the nonemptiness of an automaton on infinite words is a word of the form uv^ω, for finite words u and v. The length of the witness is $|u| + |v|$. Modifying the emptiness algorithm further to return a shortest witness (that is, one with a minimal length) has useful applications. One of the weaknesses of automated synthesis is that it may produce systems that are needlessly complicated [2]. Returning a minimal witness amounts to returning the most optimal system that satisfies the specification. In the context of model checking, short witnesses enable the user to find errors in the system as soon as they appear, and they give a compact explanation to the incorrectness of system. Using counterexamples for refinement of an abstract system, shorter counterexamples point better to elements that have to be refined. Finally, using witnesses in the context of vacuity, short witnesses explain better how formulas have been satisfied in a non-vacuous way.

Finding a shortest witness to the nonemptiness of an automaton has the flavor of finding shortest paths in graphs. Indeed, previous work on short witnesses studies the problem of finding minimal fair cycles in graphs [8, 14, 32]. Nevertheless, the fact that an automaton corresponds to a *labeled* graph, makes things more complicated. To see this, consider for example the deterministic Büchi automaton $\mathcal{A}$ in Figure 1. In the Büchi acceptance condition, a run is accepting if it visits the set of accepting states infinitely often. The shortest accepting run that witnesses the nonemptiness of $\mathcal{A}$ is

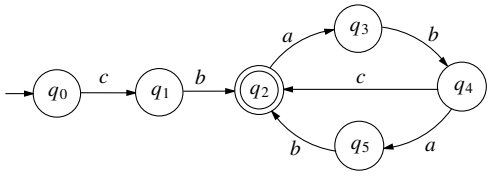

Fig. 1. While the shortest accepting run in $\mathcal{A}$ is $q_0, q_1, (q_2, q_3, q_4)^\omega$, the shortest witness to its nonemptiness is $c(ba)^\omega$.

$q_0, q_1, (q_2, q_3, q_4)^\omega$, which witnesses the membership of the word $cb(abc)^\omega$, of length 5, in the language of $\mathcal{A}$. The automaton $\mathcal{A}$, however, has an even shorter witness to its nonemptiness. Indeed, while the accepting run $q_0, q_1, (q_2, q_3, q_4, q_5)^\omega$ is of length 6, it witnesses the membership of the word $c(ba)^\omega$, which is of length 3.

For the applications mentioned above, it is the shortest witness, rather than the shortest accepting run, that we want to return to the user[1]. Indeed, in the case of synthesis, the shortest witness points to the most optimal system that satisfies the specification, and in the case of model checking, the shortest witness is the shortest computation that violates the property. In particular, in the case of model checking, the automaton is the product of the specification automaton with the system, and considering shortest accepting runs rather than shortest witnesses is sensitive to the structure of the specification automaton.

The length of a shortest witness is a robust measure, as it is independent of the specification formalism: every language $L \subseteq \Sigma^\omega$ has a shortest member, and this member is independent of whether L is specified by an LTL formula, or by a particular type of an automaton. In [32], the authors point to the fact that a shortest witness may not coincide with a shortest accepting run, and studied automata for which the two measures coincide (that is, the shortest witness is read along a shortest run). Here, we take a different approach, and refer to the length of the witness directly, for various specification formalisms. Note that the shortest-witness measure is especially appropriate when we consider the intersection of several automata, as in the case of model checking a system that is given by means of its underlying components. There, the shortest accepting run is defined with respect to the product of the components of the system. A shortest witness, on the other hand, is independent of the presentation of the system, and can be defined with respect to the underlying components.

Classical models of computations, such as Turing machines and automata, have been enriched with features to capture concurrency. *Nondeterminism*, for example, amounts to letting several processes run over the input word, each following different nondeterministic choices. In the case of nondeterminism, no cooperation between the spawned processes takes place, except when time comes to decide whether the input should be accepted. Then, the input is accepted if some process accepts it. A dual type of cooperation is allowed in *universal* automata. There, the input word is accepted if all the processes accept it. It turned out that such limited cooperation is sufficient to make nondeterministic and universal automata exponentially more succinct than deterministic automata, and to make their combination, namely alternating automata, doubly exponentially more succinct than deterministic automata [9]. As studied in [10], enriching automata with real concurrency, where the spawned processes can cooperate all along the computation (technically, a *concurrent* automaton consists of several components that run concurrently, and the transitions of a component depend on the states of the other components), results in even more succinct automata.

The automata-theoretic approach to formal verification was originally developed with nondeterministic automata, and is based on an exponential translation of LTL formulas to nondeterministic Büchi automata [35]. In recent years, however, more and more algorithms and tools are based on *alternating* automata. A significant advantage of alternating automata is the straightforward (and linear) translation of LTL formulas to alternating Büchi automata [26, 34]. Solving the nonemptiness problem for alternating

[1] One can consider an even shorter description, where, for example, a subword $aaa \ .^n. \ a$ is represented by a^n, with n encoded in binary. Then, the description of the word may be logarithmically shorter. We will refer also to such *compressed* descriptions.

Büchi automata is done by translating them to nondeterministic Büchi automata [27]. The translation involves an exponential blow-up. Thus, alternating automata do not lead to an improved complexity, but they do suggest cleaner algorithms with practical advantages: an ability to minimize both the intermediate alternating automaton and the nondeterministic one [12], an ability to use the structure of the alternating automaton in order to generate minimal nondeterministic automata [15], and more.

We study the problem of finding shortest witnesses to Büchi automata with various types of concurrency. The input to the *shortest-witness problem* is an automaton and an integer k, given in binary. The problem is to determine whether a witness uv^ω such that $|uv| \leq k$ exists[2]. We start with nondeterministic automata, and show that the witness problem for them is NP-complete — more complex than the NLOGSPACE-complete nonemptiness problem[3]. We describe a heuristic that does better than checking all candidates, and is based on the observation it is possible to transfer letters from the prefix u of the witness to its cycle v, and vice versa. A similar idea is used in [14] in the context of shortest accepting runs, but is more significant in the context of shortest witnesses. We then show that the increased complexity with respect to the nonemptiness problem is carried over also to concurrent automata, where the witness problem is NEXPTIME-complete — more complex than the PSPACE-complete nonemptiness problem for them [10, 21]. It follows that our heuristic can be applied to the nondeterministic automaton obtained by removal of concurrency, but we cannot hope to do much better.

Our main result is that for alternating automata, one can do better, and the shortest-witness problem is not more complex than the nonemptiness problem. The technical point is that while alternating automata are sufficiently strong to count to k with $O(\log k)$ states [3], they are sufficiently weak to let us analyze the run on a word of the form uv^ω by carefully analyzing the run of each of the processes in isolation. This leads to a PSPACE algorithm to the shortest-witness problem for alternating automata. From a practical point of view, our results give another good reason to translate LTL formulas to alternating automata, rather than going all the way to nondeterministic ones. Indeed, not only the algorithm for alternating automata is cleaner, but also it improves the complexity from NEXPTIME to PSPACE. Also, in the context of model checking, our algorithm shows that when the system is given by a set of components (that is, when the language of the system is the intersection of the languages of its underlying components, in which case it can be efficiently translated to an alternating automaton), it is better to avoid the generation of the product system and reason about the components in isolation. ¿From a theoretical point of view, our results extend previous study on the computational price of different types of concurrency. We will go back to this point in Section 6.

[2] We specify the shortest witness problem as a decision problem rather than an optimization problem in order to analyze its complexity in terms of the classical complexity classes. By bounding the length of the shortest witness and performing a binary search for it, our results imply also tight bounds in complexity classes like $\mathrm{FP}^{\mathrm{NP}[\log]}$, which refer to the problem of computing the length of the shortest witness.

[3] This result is different from the NP-completeness result in [8] for the shortest accepting run problem for a nondeterministic generalized Büchi automaton. We refer to this point lengthily in Section 3.

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. We denote by w^l the suffix $\sigma_l \cdot \sigma_{l+1} \cdot \sigma_{l+2} \cdots$ of w. An *automaton on infinite words* is $\mathcal{A} = \langle \Sigma, Q, Q_0, \rho, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\rho : Q \times \Sigma \to 2^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and α is an acceptance condition (a condition that defines a subset of Q^ω). Intuitively, $\rho(q, \sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ. Since $\mathcal{A}$ has several initial states and the transition function of $\mathcal{A}$ may specify many possible transitions for each state and letter, $\mathcal{A}$ is not *deterministic*. If $|Q_0| = 1$ and ρ is such that for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\rho(q, \sigma)| \leq 1$, then $\mathcal{A}$ is *deterministic*.

A *run* of $\mathcal{A}$ on w is a function $r : \mathbb{N} \to Q$ where $r(0) \in Q_0$ (i.e., the run starts in an initial state) and for every $l \geq 0$, we have $r(l+1) \in \rho(r(l), \sigma_l)$. (i.e., the run obeys the transition function). Acceptance is defined according to the set $Inf(r)$ of states that r visits *infinitely often*, i.e., $Inf(r) = \{q \in Q : \text{for infinitely many } l \in \mathbb{N}, \text{we have } r(l) = q\}$. As Q is finite, it is guaranteed that $Inf(r) \neq \emptyset$. The way we refer to $Inf(r)$ depends on the acceptance condition of $\mathcal{A}$. Several acceptance conditions are studied in the literature. We consider here *Büchi* automata, where $\alpha \subseteq Q$, and r is accepting if it visits α infinitely often. Formally, $Inf(r) \cap \alpha \neq \emptyset$.

Since $\mathcal{A}$ is not deterministic, it may have many runs on w. In contrast, a deterministic automaton has a single run on w. There are two dual ways in which we can refer to the many runs. When $\mathcal{A}$ is an *existential* automaton (or simply a *nondeterministic* automaton, as we shall call it in the sequel), it accepts an input word w iff there exists an accepting run of $\mathcal{A}$ on w. When $\mathcal{A}$ is a *universal* automaton, it accepts an input word w iff all the runs of $\mathcal{A}$ on w are accepting. *Alternation* was studied in [9] in the context of Turing machines and in [4, 9, 27] for finite automata. In particular, [27] studied alternating automata on infinite words. Alternation enables us to have both existential and universal branching choices.

For a given set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$). For technical convenience, we do not allow the formulas in $\mathcal{B}^+(Q)$ to use the constant **true**, where we also allow the formulas **true** and **false**. For $Y \subseteq X$, we say that Y *satisfies* a formula $\theta \in \mathcal{B}^+(X)$ iff the truth assignment that assigns *true* to the members of Y and assigns *false* to the members of $X \setminus Y$ satisfies θ. For example, the sets $\{q_1, q_3\}$ and $\{q_2, q_3\}$ both satisfy the formula $(q_1 \vee q_2) \wedge q_3$, while the set $\{q_1, q_2\}$ does not satisfy this formula.

Consider an automaton $\mathcal{A}$ as above. We can represent ρ using $\mathcal{B}^+(Q)$. For example, a transition $\rho(q, \sigma) = \{q_1, q_2, q_3\}$ of a nondeterministic automaton $\mathcal{A}$ can be written as $\rho(q, \sigma) = q_1 \vee q_2 \vee q_3$. If $\mathcal{A}$ is universal, the transition can be written as $\rho(q, \sigma) = q_1 \wedge q_2 \wedge q_3$. While transitions of nondeterministic and universal automata correspond to disjunctions and conjunctions, respectively, transitions of alternating automata can be arbitrary formulas in $\mathcal{B}^+(Q)$. We can have, for instance, a transition $\delta(q, \sigma) = (q_1 \wedge q_2) \vee (q_3 \wedge q_4)$, meaning that the automaton accepts a suffix w^i of w from state q, if it accepts w^{i+1} from both q_1 and q_2 or from both q_3 and q_4. Such a transition combines existential and universal choices.

Formally, an *alternating automaton on infinite words* is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Σ, Q, q_0, and α are as in automata, and $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is a transition function. When $\mathcal{A}$ runs on an input word, it generates (an unbounded number of) processes that read the input word. The joint behavior of these processes can be described in a tree; thus a run of an alternating automaton is a tree in which each node is labeled by a state of $\mathcal{A}$. A node in level l that is labeled q, corresponds to a process of $\mathcal{A}$ that visits the state q and has to accept the suffix w^l of w. As proven in [11], runs of alternating Büchi automata are *memoryless* in the sense that if $\mathcal{A}$ accepts a word w, then it also accepts w in a memoryless run — one in which two processes that are in the same state and have to accept the same suffix proceed in the same way[4]. Accordingly, we restrict attention to memoryless runs and define a run of an alternating Büchi automaton to be a DAG (directed acyclic graph).

Formally, a *run*-DAG of $\mathcal{A}$ on an input word $w = \sigma_0 \cdot \sigma_1 \cdots$, is $G_r = \langle V, E \rangle$, where $V \subseteq Q \times \mathbb{N}$ and $E \subseteq \bigcup_{l \geq 0} (Q \times \{l\}) \times (Q \times \{l+1\})$ are such that $\langle q_0, 0 \rangle \in V$, and for every vertex $\langle q, l \rangle \in V$, there is a set $S = \{q_1, \ldots, q_k\}$ such that S satisfies $\delta(q, \sigma_l)$ and for all $1 \leq c \leq k$, we have $\langle q_c, l+1 \rangle \in V$ and $E(\langle q, l \rangle, \langle q_c, l+1 \rangle)$.

A run-DAG is *accepting* iff all its paths[5], which are labeled by words in Q^ω, satisfy the acceptance condition. A word w is accepted iff there exists an accepting run-DAG on it. Note that while conjunctions in the transition function of $\mathcal{A}$ are reflected in branches of G_r, disjunctions are reflected in the fact we can have many run-DAGs on the same word.

A run of an alternating automaton is a tree $r : T_r \to Q$ for some $T_r \subseteq \mathbb{N}^*$. Formally, a tree is a (finite or infinite) nonempty prefix-closed set $T \subseteq \mathbb{N}^*$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $x \in T$, the nodes $x \cdot c \in T$ where $c \in \mathbb{N}$ are the *children* of x. A node with no children is a *leaf*. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\epsilon \in \pi$ and for every $x \in \pi$, either x is a leaf, or there exists a unique $c \in \mathbb{N}$ such that $x \cdot c \in \pi$. Given a finite set Σ, a *Σ-labeled tree* is a pair $\langle T, V \rangle$ where T is a tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ. A run of $\mathcal{A}$ on an infinite word $w = \sigma_0 \cdot \sigma_1 \cdots$ is a Q-labeled tree $\langle T_r, r \rangle$ such that the following hold:

- $r(\varepsilon) = q_0$.
- Let $x \in T_r$ with $r(x) = q$ and $\delta(q, \sigma_{|x|}) = \theta$. There is a (possibly empty) set $S = \{q_1, \ldots, q_k\}$ such that S satisfies θ and for all $1 \leq c \leq k$, we have $x \cdot c \in T_r$ and $r(x \cdot c) = q_c$.

For example, if $\delta(q_0, \sigma_0) = (q_1 \vee q_2) \wedge (q_3 \vee q_4)$, then possible runs of $\mathcal{A}$ on w have a root labeled q_0, have one node in level 1 labeled q_1 or q_2, and have another node in level 1 labeled q_3 or q_4. Note that if $\theta =$ **true**, then x need not have children. This is the reason why T_r may have leaves. Also, since there exists no set S as required for $\theta =$ **false**, we cannot have a run that takes a transition with $\theta =$ **false**.

[4] [11] proves a stronger result, namely the existence of memoryless accepting runs for parity alternating automata. Since the Büchi acceptance condition is a special case of the parity acceptance condition, the result cited above follows.

[5] Recall that we do not allow the formulas in $\mathcal{B}^+(Q)$ to use the constant **true**, thus all the paths of G_r are infinite.

A run $\langle T_r, r\rangle$ is *accepting* iff all its infinite paths, which are labeled by words in Q^ω, satisfy the acceptance condition. A word w is accepted iff there exists an accepting run on it. Note that while conjunctions in the transition function of $\mathcal{A}$ are reflected in branches of $\langle T_r, r\rangle$, disjunctions are reflected in the fact we can have many runs on the same word.

Recall that when an alternating automaton runs on an input word, it spawns to several processes. All these processes take part in the task of deciding whether the word belongs to the language. No cooperation, however, between the processes takes place, except when time comes to decide whether the input should be accepted. A different type of concurrency is one in which the processes cooperate all along the run. This type of concurrency exists in *nondeterministic Büchi automata with bounded concurrency* (concurrent Büchi automata, for short), introduced in [10][6]. A CBW is a tuple $\mathcal{A} = \langle \Sigma, \mathcal{A}_1, \ldots, \mathcal{A}_n\rangle$ consisting of an alphabet Σ and n *components* $\mathcal{A}_1, \ldots, \mathcal{A}_n$, for some $n \geq 1$. Each component $\mathcal{A}_i$ is a tuple $\langle Q_i, Q_i^0, \delta_i, \alpha_i\rangle$, where Q_i is a finite set of states, and we require the state sets of the different components to be pairwise disjoint. Let $Q = \bigcup_{j=1}^{n} Q_j$. The set $Q_i^0 \subseteq Q_i$ is the set of initial states, $\delta_i : Q_i \times \Sigma \times \mathcal{B}(Q) \to 2^{Q_i}$ is a transition relation, where $\mathcal{B}(Q)$ denotes the set of all Boolean propositional formulas over Q, and $\alpha \in \mathcal{B}(Q)$ is a Büchi acceptance condition. Note that while each component of Q has its own states and transitions, its transitions depend not only on the component's current state but also on the current states of the other components. Also, the Büchi acceptance condition refers to the states of all components.

A *configuration* of $\mathcal{A}$ is a tuple $c = \langle q_1, q_2, \ldots, q_n\rangle \in Q_1 \times Q_2 \times \cdots \times Q_n$, describing the current state of each of the components. A configuration is *initial* if for all $1 \leq i \leq n$, we have $q_i \in Q_i^0$. We use C to denote the set of all configurations of $\mathcal{A}$, and C_0 to denote the set of all its initial configurations. For a propositional formula θ in $\mathcal{B}(Q)$ and a configuration $c = \langle q_1, q_2, \ldots, q_n\rangle$, we say that c *satisfies* θ if assigning **true** to states in c and **false** to states not in c makes θ true. For example, $s_1 \wedge (t_1 \vee t_2)$, with $s_1 \in Q_1$ and $\{t_1, t_2\} \subseteq Q_2$, is satisfied by every configuration in which $\mathcal{A}_1$ is in state s_1 and $\mathcal{A}_2$ is in either t_1 or t_2.

Given two configurations $c = \langle q_1, q_2, \ldots, q_n\rangle$ and $c' = \langle q_1', q_2', \ldots, q_n'\rangle$, and a letter $\sigma \in \Sigma$, we say that c' is a *σ-successor of c*, if for all $1 \leq i \leq n$ there is $\theta_i \in \mathcal{B}(Q)$ such that c satisfies θ_i and $q_i' \in \delta_i(q_i, \sigma, \theta_i)$. In other words, a σ-successor configuration is obtained by simultaneously reading σ in all the components. A run of $\mathcal{A}$ on an input word $w = \sigma_0, \sigma_1, \ldots$ is a function $r : \mathbb{N} \to C$ where $r(0) \in C_0$ and for every $l \geq 0$, we have $r(l+1)$ is a σ_l successor of $r(l)$. Acceptance is defined according to the set $Inf(r)$ of configurations that r visits *infinitely often*. A run is accepting if at least one configuration in this set satisfies α.

We use NBW, ABW, and CBW to denote nondeterministic, alternating, and concurrent Büchi automata, respectively. For all types of automata, the language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of infinite words that $\mathcal{A}$ accepts. Thus, each word automaton defines a subset of Σ^ω. For all types, we refer also to automata on finite words (denoted NFW,

[6] The basic motivation for this model comes from the *statecharts* of [17], which can be viewed as nondeterministic automata with both concurrency and hierarchy. Our goal here is to study the role of concurrency, and we eliminate the hierarchy.

AFW, and CFW). There, acceptance is defined according to the last state visited by the run (or a process of the run, in case of the alternating and concurrent models).

A *witness* to the nonemptiness of an automaton $\mathcal{A}$ is a word $w \in L(\mathcal{A})$. For an automaton $\mathcal{A}$ on infinite words, we define a *shortest witness* for $\mathcal{A}$ to be a word $uv^\omega \in L(\mathcal{A})$ such that $|uv|$ is minimal; that is, for all words $u'(v')^\omega \in L(\mathcal{A})$, we have $|uv| \leq |u'v'|$. We refer to $|uv|$ as the length of the witness. For example, as discussed in Section 1, the shortest witness for the automaton of Figure 1 is $c(ba)^\omega$, which is of length 3. The *shortest-witness* problem is to decide, given an automaton $\mathcal{A}$ and an integer $k \geq 0$, given in binary, whether $\mathcal{A}$ has a witness of length at most k.

It is easy to see that for an NBW with n states, the length of a shortest witness is bounded by $2n$. In the case of ABW and CBW, a shortest witness may be exponentially longer than the size of the automaton. Intuitively, the above follows from the fact that intersection of automata can be modeled with no blow-up by both alternating and concurrent automata. For a concrete example, consider, given $k \geq 1$, the intersection of k automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$ defined as follows. For $i \geq 1$, let p_i be the i-th prime number. Let $\mathcal{A}_i$, for $1 \leq i \leq k$, be an NBW that accepts exactly all words of the form $(a^{d_i}b)^\omega$, for $d_i = 0 \bmod p_i$. For example, $\mathcal{A}_1$ accepts $((aa)^*b)^\omega$, $\mathcal{A}_2$ accepts $((aaa)^*b)^\omega$, $\mathcal{A}_3$ accepts $((aaaaa)^*b)^\omega$, and so on. It is easy to see that $\mathcal{A}_i$ needs $O(p_i)$ states. Since $p_i = O(i \log i)$ [20], the size of all components together is polynomial in k. On the other hand, the shortest witness to the nonemptiness of their intersection is $(a^{2\cdot 3\cdot 5\cdots p_k}b)^\omega$. Since $2 \cdot 3 \cdot 5 \cdots p_k$ is exponential in k, we get that the shortest witness is exponential in the size of the components.

Remark 1 Recall that the shortest-witness problem gets as input both an automaton $\mathcal{A}$ and an integer k. Since k is given in binary, an algorithm that is based on checking the membership in $\mathcal{A}$ of all words uv^ω with $|uv| \leq k$, is exponential in the input. By the above discussion, the length of a shortest witness is at most exponential in the size of $\mathcal{A}$. Thus, with k given in binary, $\mathcal{A}$ is always the dominant part of the input (otherwise, we can reduce the shortest-witness problem to the nonemptiness problem, which is independent of k). Consequently, the complexities we get to our decision problem correspond to the complexities of the optimization problem in which only $\mathcal{A}$ is given, and a shortest witness has to be found. □

A naive approach for finding a shortest witness first checks the nonemptiness of the automaton and then tries witnesses of increasing lengths. Our main results in this paper are that for nondeterministic and concurrent automata, one can proceed with algorithms that are likely to perform in average better than the naive algorithm, yet it is impossible to go below the NP and NEXPTIME complexities of the naive approach. On the other hand, for alternating automata, where the naive approach also yields an NEXPTIME algorithm, we are able to suggest a PSPACE algorithm.

3 The Shortest-Witness Problem for Nondeterministic Automata

We start with NBW and prove that the shortest-witness problem for them is NP-complete. The result is technically easy, but is of interest, as it highlights the difference between

the shortest-accepting-run and the shortest-witness problems. The shortest-accepting-run problem for NBW can be reduced to the problem of finding shortest paths in a graph and can therefore be solved in polynomial time. On the other hand, it is proven in [8] that the shortest-accepting-run problem for nondeterministic *generalized* Büchi automata (NGBW) is NP-complete. In an NGBW, the acceptance condition is a set of sets of states, and a run is accepting if it visits all sets infinitely often. By the above, hardness of the shortest-accepting-run problem in NP crucially depends on the use of the generalized Büchi condition. An NGBW $\mathcal{A}$ can be translated to an NBW $\mathcal{A}'$ with only a polynomial blow up. How come, then, that the problem is in PTIME for NBW and is NP-hard for NGBW? Well, while $\mathcal{A}'$ accepts the same language as $\mathcal{A}$, it has a different structure, and a shortest accepting run for it says nothing about a shortest accepting run for $\mathcal{A}$. This is a drawback of the shortest-accepting-run measure, which depends on the specification formalism. The shortest-witness measure, on the other hand, is independent of the specification formalism, and the solution of the shortest-witness problem can (and indeed does) involve translations between different specification formalisms.

Theorem 2. *The shortest-witness problem for NBW is NP-complete.*

Proof: We first prove membership in NP. Consider an NBW $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$. When $\mathcal{A}$ is not empty, a witness uv^ω is induced by a simple path (labeled u) from Q_0 to a state in α that is reachable from itself (by a simple cycle labeled v). Thus, the length $|uv|$ of a shortest witness is bounded by $2|Q|$. Since the membership-problem for NBW can be solved in polynomial time, membership in NP follows.

For the lower bound, we do a reduction from the Hamiltonian-cycle problem. There, we are given a graph $G = \langle V, E \rangle$ and we have to decide whether there exists a simple cycle traversing all vertices in V. Given G, let $V = \{1, \ldots, n\}$. We define an NBW $\mathcal{A} = \langle E, V \times V, \{\langle 1, 1 \rangle\}, \delta, \{\langle n, n \rangle\} \rangle$, where $\delta(\langle i, j \rangle, (i, h))$ is $\langle h, (j \bmod n) + 1 \rangle$ if $i = j$, and is $\langle h, j \rangle$ if $i \neq j$. Intuitively, a state $\langle i, j \rangle$ indicates that $\mathcal{A}$ traverses a path that now visits vertex i, and is waiting for a visit in vertex j. Accordingly, from state $\langle i, j \rangle$, the NBW $\mathcal{A}$ can read only edges with source i, and it updates the first element of the successor state to be the target of the edge. In addition, if $i = j$, then the path visits the vertex for which $\mathcal{A}$ waits, and it updates the second element of the successor state to be the next vertex. Consequently, $\mathcal{A}$ visits the state $\langle n, n \rangle$ infinitely many times iff the traversed path has visited all vertices infinitely often.

It is easy to see that $\mathcal{A}$ has a witness of length (at most) n iff G has a Hamiltonian cycle. Indeed, if G has a Hamiltonian cycle C, then for a word w read along C from vertex 1, we have $w^\omega \in L(A)$. For the other direction, assume that $\mathcal{A}$ has a witness uv^ω of length n. An accepting run r on uv^ω visits $\langle n, n \rangle$ infinitely often. By the definition of δ, the run r also visits the states $\langle i, i \rangle$ infinitely often, for all $1 \leq i \leq n$. The transitions to each of these states are labeled by different letters. Therefore, v must include at least n different letters, and can include only n letters only if G has a Hamiltonian cycle. □

Remark 3 Membership in NP holds also for nondeterministic generalized Büchi automata. There, the length of a shortest witness can be bounded by $2k|Q|$, where k is the index of the automaton. Note that the automaton $\mathcal{A}$ used in the hardness proof is deterministic. Moreover, with some more technicality (going to an accepting loop from

the state $\langle n, n \rangle$), it is possible to modify $\mathcal{A}$ to be a Büchi automaton with $\alpha = Q$ (also known as a *looping* automaton). Thus, NP-hardness holds already for deterministic looping automata. □

Remark 4 As mentioned in Section 1, a compressed description of a witness is such that subwords consisting of a block of m repetitions of the same subword x are represented by x^m, with m given in binary. For example, if the witness is $aabaab(cccb)^\omega$, a compressed description for it is $(a^2b)^2(c^3b)^\omega$. In the *shortest compressed witness* problem, we are given an automaton $\mathcal{A}$ and an integer k, given in binary, and we have to decide whether a witness of compressed length k exists.

Since the length of a witness is bounded by $2|Q|$, so is the length of a compressed witness, which implies that the shortest compressed-witness problem is in NP. In addition, since our NP-hardness proof for the shortest-witness problem imposes a witness with no repeated letters, NP-hardness holds also for the shortest compressed-witness problem. □

The NP-completeness of the shortest-witness problem for NBW implies that a polynomial algorithm for finding a shortest witness is unlikely to exist. A naive algorithm for finding a shortest witness for a given NBW $\mathcal{A}$ goes over all words of the form uv^ω such that $|uv|$ is bounded by $2|Q|$, and returns the shortest such word that is accepted by $\mathcal{A}$. In the full version, we tighten the $2|Q|$ bound to $|Q|$ and describe an exponential time and polynomial space algorithm that has a better running time than the naive algorithm. Essentially, the improved algorithm is based on the observation that the we can choose the location where the loop starts in such a way that the paths traversed along u and v are disjoint. This observation is also used in [14] in the context of shortest accepting runs, but its applications are more significant in our setting. Formally, we have the following.

Proposition 1. *Consider an NBW $\mathcal{A} = \langle \sigma, Q, \delta, Q_0, \alpha \rangle$. For a state $q \in Q$, let u_q be a word labeling a shortest path from Q_0 to q, and let v_q be a shortest word such that v_q^ω is accepted by $\mathcal{A}$ with initial state q. Then, a shortest witness for $\mathcal{A}$ is $u_q v_q^\omega$, for some $q \in Q$.*

4 The Shortest-Witness Problem for Concurrent Automata

We now turn to study the shortest-witness problem for concurrent automata. Note that for concurrent automata, shortest accepting runs are not defined[7]. On the other hand, the definition of shortest witnesses is semantic, and we can refer to the shortest witnesses of concurrent automata.

Theorem 5. *The shortest-witness problem for CBW is NEXPTIME-complete.*

[7] One can define shortest accepting runs for CBW by referring to the product of the components, but this gives up the exponential succinctness of the concurrent model.

Proof: A CBW $\mathcal{A}$ can be translated to an NBW with an exponential blow up [10]. Thus, membership in NEXPTIME follows from Theorem 2. For the lower bound, we do a reduction from the *succinct Hamiltonian cycle* problem. A succinct representation of a graph with 2^n nodes is a Boolean circuit C with $2n$ input gates. The graph represented by C is $G_C = \langle\{0, \ldots, 2^n - 1\}, E\rangle$, where $E(i, j)$ iff C has value 1 on the input (of length $2n$) that has the n-bit binary encoding of the integers i and j. The *succinct Hamiltonian circle* problem is to determine, given a circuit C, whether the graph G_C has a Hamiltonian cycle. Like many other problems on succinct graphs whose "non-succinct version" (that is, over graphs given enumeratively) is NP-complete, the succinct Hamiltonian cycle problem is NEXPTIME-complete [16].

Given C with $2n$ input gates, we construct a CBW $\mathcal{A}$ with $|C| + n$ components, each with two states, such that $\mathcal{A}$ has a witness of length at most $n2^n$ iff G_C has a Hamiltonian cycle. We partition the components of $\mathcal{A}$ to $2n$ input components, $|C|-2n$ internal-gate components, and n counting components. Recall that each component has two states, thus we refer to the *value* of a component, which is either 0 or 1, according to the state it visits.

The alphabet of $\mathcal{A}$ is $\{0, 1\}$, and an input word $w = w_0, w_1, \ldots$ describes an attempt to encode a path $v_0, v_1, \ldots$ in G_C, where the vertex v_i is encoded in the subword $w_{in}, \ldots, w_{(i+1)n-1}$. The word w encodes a path if its partition to blocks of length n indeed describes a path. Thus, there is an edge between v_i and v_{i+1} for all $i \geq 0$. Equivalently, the value of C on $w_{in}, \ldots, w_{(i+2)n-1}$ is 1, for all $i \geq 0$.

Accordingly, $\mathcal{A}$ proceeds as follows. The $2n$ input components maintain the last edge that was taken, and the internal-gate components maintain the value of the internal gates of C with respect to this edge. The automaton $\mathcal{A}$ updates its guess for the next edge whenever a block in the input word starts (that is, once every n letters). The update consists of the following steps: a check that the component of the output gate (the one that maintains the value of C) is 1, a transfer of the values of the input components $n + 1, \ldots 2n$ to the input components $1, \ldots, n$, a guess for the new values of the input components $n + 1, \ldots, 2n$, and a guess for the value of the internal-gate components. Once the update has been performed, the input and internal-gate components do not change their value until $\mathcal{A}$ finishes reading the current block. They may, however, get stuck (and do so in case of a bad guess), during the reading of the current block. Technically, when the current block is read, the input components $n + 1, \ldots, 2n$ check that the guess for the next vertex is correct (that is, in the i-th letter of the block, the input component $n+i+1$ expects to read the letter that agrees with its value. If this is not the case, the component gets stuck, and the run is rejected. In addition, each internal-gate components checks that its guessed value corresponds to the semantics of the gate with which it is associated. For example, an internal-gate component associated with an and gate, stays in its state if its value is the conjunction of the values of the components associated with its operands. If this is not the case, the component gets stuck, and the run is rejecting. For the initial configuration, $\mathcal{A}$ guesses values for all input components. Note that the components of $\mathcal{A}$ make use of its concurrency: the transitions of one component depends on the values of other components.

By the above, $\mathcal{A}$ accepts a word only if it encodes a path in C_G. It is left to describe how $\mathcal{A}$ takes care of the path being a Hamiltonian cycle. This is where the counter

components enter the picture. The job of these components is similar to the job of the second element in the pair in the state space of the NBW described in the proof of Theorem 2. While there blowing up the state space of the NBW by the number of vertices is not a problem, here we cannot blow-up the state space by a factor of 2^n, and instead we use the ability of concurrent automata to count to 2^n with n components. Formally, whenever a block is read and the vertex in this block agrees with the value of the counter (that is, the value of the input components $n+1, \ldots, 2n$ agree with the value of the counter components), $\mathcal{A}$ increases the value of the counter by 1. □

Remark 6 Note that our reduction involves a generation of a CBW that accepts a word iff it is a path in a graph represented succinctly. Thus, the technique we developed for the shortest-witness problem is useful for proving NEXPTIME-hardness of a family of problems for which the corresponding problem on NBW is NP-hard [30]. □

5 The Shortest-Witness Problem for Alternating Automata

In the concurrent case, our results indicate that one cannot do better than translating the CBW to an NBW and solving the shortest-witness problem with respect to the NBW. ABW can also be translated to NBW with an exponential blow up [27]. Solving the shortest-witness problem by translating the ABW to an NBW would then result in an NEXPTIME algorithm. In this section we show that the translation to NBW can be avoided, and that a direct algorithm on the ABW requires only polynomial space.

Consider an ABW $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$. For two sets $S, S' \subseteq Q$, let $\mathcal{A}[S, S'] = \langle \Sigma, Q, S, \delta, S' \rangle$ be the alternating automaton on finite words obtained from $\mathcal{A}$ by defining S to be the set of *conjunctively related* initial states and S' to be its set of final states[8]. For simplicity, when $S = \{q\}$ is a singleton, we denote the automaton by $\mathcal{A}[q, S']$. Note that for every sets S, S', and S'', with $S' \subseteq S''$, we have that $L(\mathcal{A}[S, S']) \subseteq L(\mathcal{A}[S, S''])$, and $L(\mathcal{A}[S'', S]) \subseteq L(\mathcal{A}[S', S])$.

For $m \geq 1$, we say that a function $f : Q \to 2^Q \setminus \{\emptyset\}$ is *m-cyclic on $\mathcal{A}$* if there exists a set $Q' \subseteq Q$ such that

1. $q_0 \in Q'$,
2. $\bigcup_{q \in Q'} f(q) \subseteq Q'$, and
3. there exists a word w such that $|w| = m$ and $w \in \bigcap_{q \in Q'} L(\mathcal{A}[q, f(q)])$.

We say that Q' is a *core* of f. Thus, f is m-cyclic on $\mathcal{A}$ if there is a set Q' of states that contains q_0, the application of f on a state in Q' results in states in Q', and there is a word w of length m such that for all states in Q', the word w is accepted by the AFW with initial states q and accepting set $f(q)$. Intuitively, m-cyclic functions reduce the existence of a witness to the nonemptiness of the alternating automaton to the existence of the same witness to the nonemptiness of several automata.

[8] An alternating automaton with a set of conjunctively related initial states has to accept the input word from all the initial states. Automata with a set of conjunctively related initial states can be easily translated to an automaton with a single initial state: the transition from the new initial state is a conjunction of the transitions from the states in the set of initial states.

In order to describe how we use m-cyclic cycles in order to solve the shortest-witness problem for ABW, let us first consider a special case of the Büchi condition, where $\alpha = Q$. In such automata (also known as *looping* automata), every infinite run is accepting. Also, let us first handle the case where the witness is of the form v^ω.

Lemma 1. *Consider an alternating looping automaton $\mathcal{A}$. For all $m \geq 1$, there exists a witness $w^\omega \in L(A)$ such that $|w| = m$ iff there exists an m-cyclic function on $\mathcal{A}$.*

Proof: Let $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, Q \rangle$. For the first direction, let $w^\omega \in L(A)$ be such that $|w| = m$. Consider an accepting memoryless run r of $\mathcal{A}$ on w^ω. Let $G_r = \langle V, E \rangle$ be the run DAG of r. For each $i \geq 0$, let Q_i denote the set of states in the (im)-th level of G_r; thus, $Q_i = \{q : \langle q, im \rangle \in V\}$. Since the run is accepting, $Q_i \neq \emptyset$ for all $i \geq 0$. Note that for all $i \geq 0$, the set Q_i is the set of states that r visits after reading the prefix w^i of w^ω. For a state $q \in Q_i$, let $S_{q,i}$ denote the subset of Q_{i+1} that contains all states reachable from q. Formally, $q' \in S_{q,i}$ iff $\langle q', (i+1)m \rangle$ is reachable from $\langle q, im \rangle$. Since r is memoryless, then for all $i, j \geq 0$ and $q \in Q_i \cap Q_j$, we have that $S_{q,i} = S_{q,j}$. Indeed, both sets contain the set of states that q "generates" when it reads the prefix w of w^ω. For a state q for which $q \in Q_i$ for some $i \geq 0$, let $S_q = S_{q,i}$. As argued, the definition of S_q is independent of i.

We construct an m-cyclic function f as follows. Let $Q' = \bigcup_{i \geq 0} Q_i$. Thus, the core of f is the set of states that r visits in levels $0, m, 2m, 3m, \ldots$. For $q \in Q'$, we define $f(q) = S_q$. For $q \notin Q'$, we define $f(q)$ arbitrarily. We claim that the three conditions for f being m-cyclic hold. First, since $Q_0 = \{q_0\}$, we have that $q_0 \in Q'$. Second, if $q' \in f(q)$ then $q' \in Q'$, and hence $\bigcup_{q \in Q'} f(q) \subseteq Q'$. Finally, for each $q \in Q_i$, we have that $w \in \mathcal{A}[q, S_q]$, so, by the definition of f, we also have that $w \in L(\mathcal{A}[q, f(q)])$. Hence, as $|w| = m$, there exists a word of length m in $\bigcap_{q \in Q'} L(\mathcal{A}[q, f(q)])$.

For the other direction, let f be an m-cyclic function on $\mathcal{A}$, and let Q' be a core for f. Since f is m-cyclic, there exists a word w of length m in $\bigcap_{q \in Q'} L(\mathcal{A}[q, f(q)])$. We claim that $\mathcal{A}$ accepts w^ω. We define a run r of $\mathcal{A}$ on w^ω as follows. Let r_0 be an accepting run of $\mathcal{A}[q_0, f(q_0)]$ on w. Such a run exists, as by the definition of f, we have that $w \in L(\mathcal{A}[q_0, f(q_0)])$. Let S_1 be the set of states that r_0 visits after reading w. Let r_1 be an accepting run of $\mathcal{A}[S_1, \bigcup_{q \in S_1} f(q)]$ on w. Again, such a run exists, as $S_1 \subseteq Q'$, and hence $w \in \mathcal{L}(\mathcal{A}[q, f(q)])$ for all $q \in S_1$. We continue in the same manner to obtain, for each $i \geq 0$, a set $S_i \subseteq Q'$ and an accepting run r_i of $\mathcal{A}[S_i, \bigcup_{q \in S_i} f(q)]$ on w. Since there are 2^n subsets of Q, it is guaranteed that there are $j' < j \leq 2^n$ such that $S_j = S_{j'}$. The run obtained by concatenating $r_0, r_1, \ldots, r_{j'-1}$ and then repeatedly concatenating $r_{j'}, \ldots, r_{j-1}$ is an infinite accepting run of $\mathcal{A}$ on w^ω. □

We can now extend m-cyclic function to ABW. In [29], Muller et al. introduce *weak alternating automata* (AWW). In an AWW, the acceptance condition is $\alpha \subseteq Q$, and there exists a partition of Q into disjoint sets, Q_i, such that for each set Q_i, either $Q_i \subseteq \alpha$, in which case Q_i is an *accepting set*, or $Q_i \cap \alpha = \emptyset$, in which case Q_i is a *rejecting set*. In addition, there exists a partial order $\leq$ on the collection of the Q_i's such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, \sigma)$, for some $\sigma \in \Sigma$, we have $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. It follows that every infinite path of a run of a AWW ultimately gets

"trapped" within some Q_i. The path then satisfies the acceptance condition if and only if Q_i is an accepting set. Thus, we can view a AWW with an acceptance condition α as both a Büchi automaton with an acceptance condition α, and a co-Büchi automaton with an acceptance condition $Q \setminus \alpha$. Indeed, a run gets trapped in an accepting set iff it visits infinitely many states in α, which is true iff it visits only finitely many states in $Q \setminus \alpha$.

The translation of LTL formulas to alternating automata results in weak automata [26]. Also, ABW can be translated to an AWW with a quadratic blow up [23]. Accordingly, we are going to use m-cyclic functions of AWW in our algorithm for the shortest-witness problem for ABW.

We say that an m-cyclic function f has a *rejecting cycle* if for every core Q' for it, there exists a sequence $q_1, q_2, \ldots, q_k$, with $k \leq |Q|$, of states in $Q' \setminus \alpha$ such that $q_{(i \bmod k)+1} \in f(q_i)$ for all $1 \leq i \leq k$. Note that a rejecting cycle refers only to states in Q' and requires some path in the run induced by f to visit $Q \setminus \alpha$ infinitely often. As we argue in Lemma 2 below, the weakness of the automaton guarantees that the fact a rejecting cycle refers only to states "sampled" by f does not prevent it from characterizing acceptance.

Lemma 2. *Consider an AWW $\mathcal{A}$. For every $m \geq 1$, there exists a word $w^\omega \in L(A)$ such that $|w| = m$ iff there exists an m-cyclic function f on $\mathcal{A}$ that does not have a rejecting cycle.*

Proof: Let $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$. For the first direction, let $w^\omega \in L(\mathcal{A})$ be such that $|w| = m$. Let r be a memoryless accepting run of $\mathcal{A}$ on w and let $G_r = \langle V, E \rangle$ be its run DAG. We claim that the m-cyclic function f defined in the proof of Lemma 1 does not have a rejecting cycle. Assume by way of contradiction that f has a rejecting cycle $q_1, \ldots, q_k$. Since for each $1 \leq i < k$ we have that $q_{(i \bmod k)+1} \in f(q_i)$, then, by the way we have defined f, there exists an index $j \geq 0$ and a vertex $\langle q_1, j \cdot m \rangle$, reachable from $\langle q_0, 0 \rangle$, from which there is a path π_1 to the vertex $\langle q_2, (j+1) \cdot m \rangle$. Again, by the definition of f, there exists a path π_2 from $\langle q_2, (j+1) \cdot m \rangle$ to $\langle q_3, (j+2) \cdot m \rangle$. We can continue in a similar way and generates an infinite path in G_r that visits infinitely many states in $Q \setminus \alpha$, contradicting the fact that G_r is accepting.

For the other direction, let f be an m-cyclic function on $\mathcal{A}$ that does not have a rejecting cycle, and let Q' be the core of f. We claim that the run r constructed in the proof of Lemma 1 is accepting. Assume by way of contradiction that r is rejecting. Then, the DAG G_r has a path $\pi = \langle q_0, 0 \rangle, \langle q_1, 1 \rangle, \ldots$ such that there is an index $l \geq 0$ such that $q_j \notin \alpha$ for all $j \geq l$. By the definition of r, we have that $Q_i \subseteq Q'$ for all $i \geq 0$. Let j be such that $j \cdot m > l$. The sequence $q_{jm}, q_{(j+1)m}, q_{(j+2)m}, \ldots$ is such that for all $i \geq 1$, we have that $q_{(j+i)m} \in Q' \setminus \alpha$, and $q_{(j+i+1)m} \in f(q_{(j+i)m})$. By the pigeonhole principle, there exist i' and i with $i' < i < n$ such that $q_{(j+i')m} = q_{(j+i)m}$. Therefore, the sequence $q_{(j+i')m}, q_{(j+i'+1)m}, \ldots, q_{(j+i-1)m}$ is a rejecting cycle in f, and we reach a contradiction. □

Theorem 7. *The shortest witness problem for ABW is PSPACE-complete.*

Proof: An ABW with n states can be translated to an NBW with 3^n states [27]. Therefore, since by Proposition 1 length of a shortest witness for an NBW is bounded the

number of its states, an ABW with n states is empty iff it has no witness of length at most 3^n. Hence, hardness in PSPACE follows from the PSPACE hardness of the nonemptiness problem for ABW. [27].

Let $\mathcal{A}$ be an ABW with n states, and let k be an integer given in binary. We describe an algorithm in NPSPACE for deciding whether $\mathcal{A}$ has a witness of length at most k. Since NPSPACE=PSPACE [31], we are done. First, the algorithm answers "no" if $\mathcal{A}$ is empty and answers "yes" if $\mathcal{A}$ is not empty and $k > 3^n$. Since the nonemptiness problem is in PSPACE, this can be done in PSPACE.

Otherwise, let $\mathcal{A}' = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ be an AWW equivalent to $\mathcal{A}$. By [23], such an AWW with at most $2n^2$ states exists. The algorithm guesses an integer $0 < m \leq k$, a function $f : Q \to 2^Q \setminus \{\emptyset\}$ and a set $Q' \in Q$. The function f consists of $|Q|$ sets of elements in Q, each of size at most $|Q|$. Hence f can be encoded in space of size $O(n^4 \log n)$. The algorithm then checks that f is m-cyclic as follows. It first checks that $q_0 \subseteq Q'$, and that $\bigcup_{q \in Q'} f(q) \subseteq Q'$. These checks are both performed in polynomial space. It then constructs an AFW $\mathcal{A}_m$ of size $O(\log m)$ that accepts all words $w \in \Sigma^m$ [3]. The algorithm then constructs an AFW $\mathcal{A}[q, f(q)]$ for each $q \in Q'$, and checks that the intersection $L(\bigcap_{q \in Q'} \mathcal{A}[q, f(q)]) \cap L(\mathcal{A}_m)$ is not empty. Since alternation can model intersection, the latter can be checked in PSPACE. Note that the intersection is not empty iff an m-cyclic function on $\mathcal{A}$ exists. It is left to check that f has no rejecting cycle. For that, the algorithm goes over each sequence $q_1, \ldots, q_k$, with $k \leq |Q|$ and checks that it is not a rejecting cycle. If all sequences pass the check successfully, the algorithm returns "yes". By Lemma 2, there exists a witness $v^\omega \in L(A)$ of length $m \leq k$ iff there is an instance of the algorithm that answers "yes".

We now expand the algorithm to account for cases where the witness is of the form uv^ω. The algorithm first guesses the length of u by guessing an integer $t < m$. It then guesses a sequence of t letters and a run of $\mathcal{A}$ on it. Let Q'_0 be the set of states the algorithm visits after reading the t letters without getting stuck, and let $\mathcal{A}''$ be the ABW obtained from $\mathcal{A}$ by defining Q'_0 to be its set of initial states. The algorithm now proceeds as in the case of witnesses of the form v^ω, with respect to witnesses of length $m - t$. □

Recall that the definition of a shortest witness is semantic and depends on the language of the automaton rather than its structure. Thus, the shortest-witness problem can be defined with respect to any specification formalism that defines ω-regular languages. LTL formulas can be linearly translated to AWW. Hence, together with the PSPACE lower bound for LTL satisfiability, Theorem 7, implies the following.

Theorem 8. *The shortest-witness problem for LTL is PSPACE-complete.*

It follows that finding a shortest witness for the satisfiability of an LTL formula is not harder than just checking its satisfiability.

6 Discussion

We studied the shortest-witness problem in three different models of concurrency. From a theoretical point of view, our results show that the limited concurrency between the

processes of an alternating automaton makes reasoning about them simpler than concurrent automata: the shortest-witness problem for ABW is PSPACE-complete, like the nonemptiness problem for it, whereas for concurrent automata, the problem is NEXPTIME-complete. Note that while cooperation between the processes of an alternating automaton is limited, this model does not bound the number of processes that run on the input. Thus, alternation cannot be easily simulated by concurrency. In fact, the translations between the two models involve an exponential blow up in both directions [10].

It is interesting to compare our results with previous results about the computational price of the two models. In Figure 2, we describe such results. The table in the figure refers to six problems that involve automata. Simulation and fair-simulation consider labeled transition systems, which can be viewed as automata with labels at the states rather than on the transitions. In fair simulation, we refer to the definition of [13]; in the case of alternating systems, we refer to the alternating simulation of [1]; the complexity of fair alternating simulation is still open. ELTL satisfiability is the problem of deciding the satisfiability of an LTL formula in which automata are used as temporal modalities.

	nonemptiness	universality	simulation	fair-simulation	ELTL satisfiability	shortest witness
NBW	NLOGSPACE [35]	PSPACE [33]	PTIME [28]	PSPACE [22]	PSPACE [19]	NPTIME Th. 2
CBW	PSPACE [10]	NEXPSPACE [10]	EXPTIME [18]	EXPSPACE [18]	EXPSPACE [19]	NEXPTIME Th. 5
ABW	PSPACE [27]	PSPACE [27]	PTIME [1]	?	EXPSPACE [19]	PSPACE Th. 7

Fig. 2. The computational price of different types of concurrency.

As can be seen from the table, all problems become exponentially more complex in the concurrent setting. On the other hand, for some problems, special algorithms for the alternating setting are less complex than an algorithm that first removes alternation, which involves an exponential blow up. The shortest-witness problem falls in this category.

¿From a practical point of view, our results indicate that alternation can be useful not only thanks to the straightforward translation of temporal logic formulas to alternating automata, but also because of computational considerations. In particular, in case the system is given symbolically by a set of underlying components, and the specification is given by an LTL formula, it is better to translate the formula to an ABW (rather than an NBW) and search for a shortest witness in this setting. Indeed, the intersection of the underlying components can be modeled by an alternating automaton, and the complexity is PSPACE, like the PSPACE complexity for the model-checking problem (note that in the symbolic setting described above, model checking is PSPACE in both the LTL formula and the underlying components).

Another application of the shortest-witness problem is synthesis. In the last years, researches have developed methods for coping with the implementation difficulties of the synthesis problem: its high complexity, and the fact its solution involves deter-

minization of automata on infinite words [25]. With these problems being challenged, there is now room for studying the size of the synthesized system, and developing automated-synthesis algorithms for generating optimal systems. Our study in this paper handles the case of a closed system. In future research, we plan to study the case of open systems, where the problem is to find minimal transducers that generate correct systems.

References

1. R. Alur, T.A. Henzinger, O. Kupferman, and M.Y. Vardi. Alternating refinement relations. In *Proc. 9th CONCUR*, LNCS 1466, pages 163–178, 1998.
2. A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal model genration. In *Proc. 2nd CAV*, LNCS 531, pages 197–203, 1990.
3. J.C. Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Mathematical Systems Theory*, 26(3):237–269, 1993.
4. J.A. Brzozowski and E. Leiss. Finite automata and sequential networks. *TCS*, 10:19–35, 1980.
5. J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. International Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.
6. E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on LP*, LNCS 131, pages 52–71, 1981.
7. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.
8. E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32nd DAC*, pages 427–432. IEEE Computer Society, 1995.
9. A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.
10. D. Drusinsky and D. Harel. On the power of bounded concurrency I: Finite automata. *J. ACM*, 41(3):517–539, 1994.
11. E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd FOCS*, pages 368–377, 1991.
12. S. Gurumurthy, O. Kupferman, F. Somenzi, and M.Y. Vardi. On complementing nondeterministic Büchi automata. In *Proc. 12th CHARME*, LNCS 2860, pages 96–110, 2003.
13. O. Grumberg and D.E. Long. Model checking and modular verification. *ACM TOPLAS*, 16(3):843–871, 1994.
14. P. Gastin, P. Moro, and M. Zeitoun. Minimization of counterexamples in spin. In *SPIN Workshop on Model Checking of Software*, pages 92–108, 2004.
15. P. Gastin and D. Oddoux. Fast LTL to büchi automata translation. In *Proc. 13th CAV*, LNCS 2102, pages 53–65, 2001.
16. H. Galperin and A. Wigderson. Succinct representations of graphs. *I& C*, 56(3):183–198, 1983.
17. D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Computer Prog.*, 8:231–274, July 1987.
18. D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. *I & C*, 173:1–19, 2002.
19. D. Harel, R. Rosner, and M.Y. Vardi. On the power of bounded concurrency iii: Reasoning about programs. In *Proc. 5th LICS*, pages 478-488, 1990.

20. G.A. Jones, J.M. Jones, and J.M. Tyrer-Jones. *Elementary Number Theory*. Springer Undergraduate Mathematics Series, Berlin, 1998.
21. D. Kozen. Lower bounds for natural proof systems. In *Proc. 18th FOCS*, pages 254–266, 1977.
22. O. Kupferman and M.Y. Vardi. Verification of fair transition systems. *CJTCS*, 1998(2), 1998.
23. O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM TOCL*, 2(2):408–429, July 2001.
24. O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. *J. STTT*, 4(2):224–233, February 2003.
25. O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th FOCS*, pages 531–540, 2005.
26. O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, March 2000.
27. S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *TCS*, 32:321–330, 1984.
28. R. Milner. *A Calculus of Communicating Systems*, LNCS 92, 1980.
29. D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th ICALP*, LNCS 226, pages 275 – 283, 1986.
30. C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
31. W.J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal on Computer and System Sciences*, 4:177–192, 1970.
32. V. Schuppan and A. Biere. Shortest counterexamples for symbolic model checking of LTL with past. In *Proc 11th TACAS*, LNCS 3440, pages 493–509, 2005.
33. A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *TCS*, 49:217–237, 1987.
34. M.Y. Vardi. Alternating automata and program verification. In *Computer Science Today – Recent Trends and Developments*, LNCS 1000, pages 471–485, 1995.
35. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *I& C*, 115(1):1–37, November 1994.