ω-regular Languages are Testable with a Constant Number of Queries [⋆]

Hana Chockler [1]

College of Computer and Information Science, Northeastern University, Boston MA 02115, U.S.A.

Orna Kupferman

School of Engineering and Computer Science, Hebrew University, Jerusalem 91904, Israel.

Abstract

We continue the study of combinatorial property testing. For a property ψ, an ε-test for ψ, for $0 < \varepsilon \leq 1$, is a randomized algorithm that given an input x, returns "yes" if x satisfies ψ, and returns "no" with high probability if x is ε-far from satisfying ψ, where ε-far essentially means that an ε-fraction of x needs to be changed in order for it to satisfy ψ. In [AKNS99], Alon et al. show that regular languages are ε-testable with a constant (depends on ψ and ε and independent of x) number of queries. We extend the result in [AKNS99] to ω-regular languages: given a nondeterministic Büchi automaton A on infinite words and a small $\varepsilon > 0$, we describe an algorithm that gets as input an infinite lasso-shape word of the form $x \cdot y^\omega$, for finite words x and y, samples only a constant number of letters in x and y, returns "yes" if $w \in L(A)$, and returns "no" with probability 2/3 if w is ε-far from $L(A)$. We also discuss the applicability of property testing to formal verification, where ω-regular languages are used for the specification of the behavior of nonterminating reactive systems, and computations correspond to lasso-shape words.

[⋆] Preliminary version appeared in the Proceedings of the 6-th International Workshop on Randomization and Approximation Techniques in Computer Science, 2002.

Email addresses: `hanac@ccs.neu.edu` (Hana Chockler), `orna@cs.huji.ac.il` (Orna Kupferman).

[1] Much of this work was done while the first author was in Hebrew University of Jerusalem, Israel

1 Introduction

Property testing was first introduced in [RS96], where Rubinfeld and Sudan checked whether a given function computes a low-degree polynomial or is far from computing it. The work in [RS96] have led to the study of combinatorial property testing, defined by Goldreich et al. in [GGR98]. Generally speaking, given a property ψ, an input x, and $0 < \varepsilon \leq 1$, we say that x is ε-far from satisfying ψ if we need to change an ε-fraction of x in order for it to satisfy ψ. For example, a graph with n vertices is ε-far from being bipartite if we need to change at least εn^2 entries in the graph's adjacency matrix in order to make it bipartite [2]. Then, an ε-test for ψ is a randomized algorithm that given ψ, x, and ε, behaves as follows.

- If x satisfies ψ, the algorithm returns "yes" with probability at least $2/3$.
- If x is ε-far from ψ, the algorithm returns "no" with probability at least $2/3$.

An ε-test may have a one-sided error, in which case if x satisfies ψ, the algorithm always returns "yes". In both cases, the algorithm has no obligation for x that neither satisfies ψ nor is ε-far from ψ. We say that a property ψ is *ε-testable* if there exists an ε-test for ψ that uses only $f(\varepsilon)$ queries on the input, where f is independent of the size of the input. [3] It turned out that several properties are ε-testable. For example, it is possible to check bipartiteness by randomly testing $poly(1/\varepsilon)$ edges of the graph [GGR98], and similar results hold for k-connectivity, acyclicity, k-colorability, and more [GR97,AK99,GR99,BR00,PRR01,PRS01].

Recently, there have been several general results on ε-testability of certain classes of properties, especially graph properties [GT03,KKR03]. One of the few general results for non-graph properties is described in [AKNS99], which studies the testability of formal languages. For a word $w \in \{0,1\}^n$ and a regular language L, we say that w is ε-far from a language $L \subseteq \{0,1\}^*$, if no word of length n that differs from w in at most εn positions is a member of L. Alon et al. proved that regular languages are ε-testable with a one-sided error and with query complexity $\tilde{O}(1/\varepsilon)$. More precisely, for every deterministic automaton A on finite words, integer n, and small enough $\varepsilon > 0$, there is an algorithm that gets as input a word $w \in \{0,1\}^n$, samples only $c \log^3(1/\varepsilon)/\varepsilon$ letters in w, where c depends only on A, returns "yes" if $w \in L(A)$, and returns "no" with probability 2/3 if w is ε-far from $L(A)$ [AKNS99].

[2] The definition of ε-far depends on the size of the input. Thus, if a graph G with m edges is given by an adjacency list, rather than an adjacency matrix, then G is ε-far from satisfying a property if we need to change at least εm edges (rather than εn^2 edges) in order for it to satisfy the property [PR99].

[3] Alternative definitions of ε-test allow a number of queries that depend on the input (usually in some sub-linear way), and other bounds on the error. The definition above is commonly used in the literature, and is the one we use in this paper.

In this paper we extend the result of [AKNS99] to *ω-regular languages*. An ω-regular language over an alphabet Σ is a set $L \subseteq \Sigma^\omega$ of infinite words over Σ. ω-regular languages are described by automata on infinite words, first introduced in the 1960's. Motivated by decision problems in mathematical logic, Büchi, McNaughton, and Rabin developed a framework of automata on infinite words and infinite trees [Büc62,McN66,Rab69]. The framework has proven to be very powerful. Automata, and their tight relation to second-order monadic logics were the key to the solution of several fundamental decision problems in mathematical logic [Tho90]. Today, automata on infinite objects are used for specification and verification of nonterminating programs. 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. There are various ways to refer to this set. In *Büchi* automata, some of the states are designated as accepting states, and a run is accepting iff it visits states from the accepting set infinitely often [Büc62].

Nondeterministic Büchi automata recognize all the ω-regular languages [Lan69] and our algorithm assumes that L is given by such an automaton. The input to our algorithm are infinite words. We consider infinite words that have a finite representation. A general such representation maps each letter $\sigma \in \Sigma$ to a predicate $P_\sigma \subseteq \mathbb{N}$ that describes the positions of the word that are labeled σ. A special case we consider here is of *lasso-shape* (also known as *ultimately periodic*) infinite words, which are of the form $x \cdot y^\omega$, for $x \in \Sigma^*$ and $y \in \Sigma^*$. Thus, every lasso-shaped word has a position from which it is cyclic. As we discuss in Section 7, this special case is of particular interest in the context of specification and verification. In particular, it is easy to see that the language of a Büchi automaton is not empty iff the automaton accepts some lasso-shape word. Following similar considerations, if a system violates an ω-regular property, it has a lasso-shape computation violating the property [CD88,VW94]. Given a lasso-shaped word w, our algorithm tests the membership of w in the language of a nondeterministic Büchi automaton [4].

For some problems on automata, the transition from finite to infinite words is complicated. For example, one cannot determinize Büchi automata [Lan69], making the complementation problem for nondeterministic Büchi automata very challenging [Saf88]. For other problems, the transition is simple. For example, while the nonemptiness problem for automata on finite words can be reduced to one reachability test (from an initial state to the accepting set α), the nonemptiness problem for Büchi automata can be reduced to $2|\alpha|$ reachability tests (from an initial to an accepting state and from the accepting state to itself). It is easy to see then, that

[4] So, we actually extend [AKNS99] by three aspects: we consider a general (rather than binary) alphabet, we consider languages given by nondeterministic (rather than deterministic) automata, and we consider ω-regular, rather than regular, languages. It is not hard to see that the algorithm in [AKNS99] can be applied also to general alphabets and nondeterministic automata, thus the only real contribution is the extension to infinite words.

given an oracle for the nonemptiness problem for automata on finite words, the nonemptiness problem for Büchi automata can be solved by $2|\alpha|$ calls to the oracle.

Consider a nondeterministic Büchi automaton A and a lasso-shape word $w = x \cdot y^\omega$. As we explain in Section 4, the membership of w in A can be reduced to a sequence of membership tests for automata on finite words, where the ε-test of [AKNS99] can be used as an oracle. The problem with this simple reduction is that the number of calls to the oracle depends on the length, $|x| + |y|$, of w. Finding an algorithm with a query complexity that does not depend on w turns out to be much more difficult, and is the main technical contribution of this paper. Essentially, we show that for every word w there is a set D of positions such that the size of D depends only on A and the following holds: if $w \in L(A)$ then there is a word $v \in L(A)$ such that v differs from w only in positions in D and the membership of v in $L(A)$ can be verified by a constant number of applications of (some variant of) the algorithm of [AKNS99]. Moreover, if w is ε-far from $L(A)$, then the above check would fail for all the words v that differ from w only in positions in D. The full details are described in Section 4. The query complexity of our algorithm is $\tilde{O}(1/\varepsilon)$, as the one of [AKNS99]. In addition, we study the special case where the language of the Büchi automaton is a *safety language*; that is, every word not in L has a finite "bad" prefix that cannot be extended to a word in L [5]. We also prove an $\Omega(1/\varepsilon)$ lower bound for the problem [6].

We hope that the ε-test for ω-regular languages would stimulate further efforts to apply the study of combinatorial property testing to *formal verification*. In formal verification, we verify that a system meets a desired behavior by executing an algorithm that checks whether a mathematical model of the system satisfies a formal specification that describes the behavior [CGP99]. Almost all current efforts and heuristics to cope with the large state spaces that commercial formal-verification tools handle do not deviate from the strict definition of formal verification, where the algorithm is not allowed to err. We believe that a major improvement of currently used heuristics should involve a deviation from the strict definition of formal verification. The setting of property testing seems very appealing for this task: the specifications are small, the systems are exceedingly large, and it is the complexity in terms of the system that we wish to bound, which is exactly what property testing does. In Section 7, we discuss this direction in detail.

[5] We note that the result of [AKNS99] does not immediately apply ε-testability for *safety properties*, even though such properties can be characterized by a regular language of bad prefixes. The reason is that there is no *a-priori* bound on the length of the prefix that needs to be checked.

[6] As discussed in [AKNS99], a lower bound of order $1/\varepsilon$ for the query complexity of testing is quite expected in general. In our case, given the $\Omega(1/\varepsilon)$ lower bound for testing of regular languages, the lower bound is even more expected.

2 Definitions

2.1 Automata

A finite word over an alphabet Σ is a finite sequence $w \in \Sigma^*$ of letters from Σ. We can view a finite word as a function $w : \{1, \ldots, n\} \to \Sigma$, where n is the length of w. An infinite word over Σ is an infinite sequence $w \in \Sigma^\omega$ of letters from Σ, and it can be viewed as a function $w : \mathbb{N} \setminus \{0\} \to \Sigma$. For a word $w \in \Sigma^\omega$ and positions $0 \leq x \leq y$ we denote by $w[x, y]$ the sub-word of w that starts at position x and ends at position y. A *nondeterministic automaton* A is $A = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, where Σ is an alphabet, Q is a set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition relation, $q_0 \in Q$ is the initial states, and $\alpha \subseteq Q$ is a set of accepting states. Given a finite word $w \in \Sigma^*$, a run r of A on w is a function $r : \{0, \ldots, n\} \to Q$ such that $r(0) = q_0$ and for all $0 \leq i \leq n$, we have $r(i + 1) \in \delta(r(i), w(i))$. The run r is *accepting* iff $r(n) \in \alpha$. If for all $q \in Q$ and $\sigma \in \Sigma$ we have that $|\delta(q, \sigma)| = 1$, then A is *deterministic*.

The automaton A can also get as input infinite words over Σ. Given such a word $w \in \Sigma^\omega$, a run r of A on w is a function $r : \mathbb{N} \to Q$ such that $r(0) = q_0$ and for all $i \geq 0$, we have $r(i + 1) \in \delta(r(i), w(i))$. Since the run has no final states, acceptance is determined with respect to the set $\mathit{inf}(r)$, of states that appear in r infinitely often. Formally, $q \in \mathit{inf}(r)$ iff $r(i) = q$ for infinitely many i's. When A is a *Büchi* automaton, the run r is *accepting* iff $\mathit{inf}(r) \cap \alpha \neq \emptyset$ [Büc62]. That is, a run is accepting iff it visits some accepting state infinitely often. Otherwise, r is rejecting. A path in A that corresponds to an accepting run is called an *accepting path*. The language of A, denoted $L(A)$, is the set of words w such that there is an accepting run of A on w. Note that $L(A) \subseteq \Sigma^*$ for automata on finite words and $L(A) \subseteq \Sigma^\omega$ for automata on infinite words. We assume that $L(A) \neq \emptyset$.

An automaton A induces a directed graph $G_A = \langle V, E \rangle$ in the following way. The set of vertices of $G(A)$ is $V = Q$, and for each q and q' in V, we have $\langle q, q' \rangle \in E$ iff there exists $\sigma \in \Sigma$ such that $q' \in \delta(q, \sigma)$. For a graph G, the *period* of G is the greatest common divisor of cycle lengths in G. Note that if A is a Büchi automaton with $L(A) \neq \emptyset$, then there is at least one cycle in G_A, thus the period of G_A is a finite $g \geq 1$.

2.2 Infinite words

We say that an infinite word $w \in \Sigma^\omega$ is *lasso-shaped* if there are $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ such that $w = w_1 \cdot (w_2)^\omega$. That is, there exists a position from which w is cyclic. The word w_1 is called the *prefix* of w, and the word w_2 is called the *lasso* of w. When $|w_1| = n_1$ and $|w_2| = n_2$, we say that w is (n_1, n_2)*-lasso-shaped.* As we discuss in Section 7, lasso-shaped words are of special interest in the context of formal verification.

For two finite words w and v of the same length, the *distance* between w and v, denoted $dist(w,v)$, is the number of letters that have to be changed in w in order to obtain v [AKNS99]. This definition is a straightforward extension of the *Hamming distance* for the case of general finite alphabet. We say that two finite words w and v of the same length n are *ε-far*, for $0 < \varepsilon \leq 1$, if $\frac{dist(w,v)}{n} \geq \varepsilon$.

For infinite words, the number of letters that have to be changed in one word in order to obtain the other can be infinite, thus we cannot extend the definition of [AKNS99] to infinite words in a straightforward way. Instead, the definition of distance should refer to the finite representation of an infinite word, thus to the prefix and lasso of lasso-shaped words. Consider the lasso-shape word $w = 01(10)^\omega$. One can represent w also as the lasso-shape words $0110(10)^\omega$ or $011(01)^\omega$. Generally, a lasso-shaped word $w_1 \cdot (w_2)^\omega$ can be represented as $w_1 \cdot (w_2)^i \cdot w_3 \cdot ((w_4 \cdot w_3)^j)^\omega$, for some $i \geq 0$, $j \geq 1$, and some partition of w_2 into w_3 and w_4. When we define the distance between lasso-shaped words, we want our definition of distance to be insensitive to a particular representation: the distance between different representations of the same word should be 0. Let w and v be two lasso-shaped words with prefixes of length n_1 and n_1' and lassos of length n_2 and n_2', respectively. Without loss of generality, assume that $n_1' \geq n_1$.

Definition 2.1 *Let $i = n_1' - n_1$ and $n = lcm(n_2, n_2')$ (least common multiplier). We define*

$$dist(w,v) = dist(w[n_1 + i + 1, n_1 + i + n], v[n_1' + 1, n_1' + n]).$$

*We say that a lasso-shaped word w is ε-*far *from a lasso-shaped word v if $dist(w,v) \geq \varepsilon n$, where n is the least common multiplier of the lengths of lassos of w and v.*

For a lasso-shaped word w and a language $L \subseteq \Sigma^\omega$, we say that w is ε-far from L if w is ε-far from all lasso-shaped words $v \in L$. Intuitively, we represent w and v as lasso-shaped words with lassos of the same length n, and count the number of letters that should be changed in the lasso of w in order to obtain the lasso of w'. The number i is the "offset", thus the comparison of letters starts from the same place in both words. The following example shows why the "offset" is important in measuring the distance between lasso-shaped words. Consider two words $w = (01)^\omega$, and $v = 1 \cdot (01)^\omega$ and the language $L = \{w : \text{all the letters of } w \text{ in odd positions are } 0\}$. The lasso of w is equal to the lasso of v, however, $w \in L$ and v is very far from L. Our definition of distance does not compare the prefixes, as their weight in the infinite words is negligible, but it does take the lengths of the prefixes into an account, making w very far from v. The following lemma shows that the distance between two different representations of the same lasso-shaped word is 0, as required.

Lemma 2.2 *Let $w = w_1 \cdot (w_2)^\omega$ be a lasso-shaped word. Let n_1 be the length of the prefix w_1, and n_2 be the length of the lasso w_2. Then, for all numbers $k \geq 0$ and $l \geq 1$ and all partitions of w_2 to w_3 and w_4, we have $dist(w_1 \cdot (w_2)^\omega, w_1 \cdot (w_2)^k \cdot w_3 \cdot ((w_4 \cdot w_3)^l)^\omega) = 0$.*

Proof: Let n_3 and n_4 be the lengths of w_3 and w_4, respectively. The lasso of w' is $(w_2)^l$ of length $l \cdot n_2$, thus $n = lcm(n_2, l \cdot n_2) = l \cdot n_2$. The length of the prefix of w' is $n_1 + k \cdot n_2 + n_3$, thus $i = n_1 + k \cdot n_2 + n_3 - n_1 = k \cdot n_2 + n_3$. Recall that $w = w_1 \cdot (w_2)^\omega$. Then

$$dist(w, w') = dist(w[n_1 + k \cdot n_2 + n_3 + 1, n_1 + k \cdot n_2 + n_3 + l \cdot n_2], w[n_1 + k \cdot n_2 + n_3 + 1, n_1 +$$

$\square$

An alternative, perhaps cleaner, way to define ε-farness is to say, for $0 < \varepsilon < 1$, that an infinite word w is ε-far from an infinite word v if

$$\lim_{n \to \infty} \frac{dist(w[1,n], v[1,n])}{n} \geq \varepsilon.$$

Clearly, for general infinite words this limit may not exist at all. However, it is easy to see that for lasso-shaped words this limit always exists and the two definitions are equivalent. Indeed, by Lemma 2.2 we can assume that w and v have the same length of prefix and the same length of lasso parts (otherwise we can unwind the lasso several times and match the lengths of prefixes). Let x be the distance between prefixes of w and v (assuming they are of the same length) and y the distance between lasso parts of w and v, and let m be the length of the lasso of w and v. By Definition 2.1, w is ε-far from v iff $\frac{y}{m} \geq \varepsilon$. Then, for all n the number of letters we have to change in $w[1,n]$ in order to obtain $v[1,n]$ is $O(x + \frac{n}{m}y)$, and dividing by n we get $O(\frac{x}{n} + \frac{y}{m})$, which converges to $\frac{y}{m}$ as $n \to \infty$.

3 Observations on Accepting Runs

In this section we analyze the structure of accepting runs of nondeterministic Büchi automata on infinite words. We argue that for lasso-shaped infinite words it suffices to examine a finite prefix of a word in order to decide its membership in $L(A)$.

Let $A = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$ be a Büchi automaton. For simplicity, we describe our algorithm for the case $\alpha = \{q_{acc}\}$ is a singleton. Later we show how to extend our results to the case of multiple accepting states. We denote by A_{fin} the automaton A viewed as an automaton on finite words. Let C_{acc} be the maximal strongly connected component of G_A that contains q_{acc}. We define the automaton $B_{fin} = \langle \Sigma, C_{acc}, \delta^q_{fin}, q_{acc}, \{q_{acc}\} \rangle$ as the automaton on finite words that is derived from the graph C_{acc}, with initial state and accepting state q_{acc}. Formally,

$B_{\mathit{fin}} = \langle \Sigma, Q \cap C_{acc}, \delta_{\mathit{fin}}, q_{acc}, \{q_{acc}\}\rangle$, where $\delta_{\mathit{fin}} : C_{acc} \times \Sigma \rightarrow 2^{C_{acc}}$ is such that $q' \in \delta_{\mathit{fin}}(q, \sigma)$ iff $q' \in \delta(q, \sigma) \cap C_{acc}$. In Lemma 3.1 we show that for lasso-shaped words, the membership problem of a word in the language can be reduced to the membership problem of two prefixes of a bounded length in languages on finite words.

Lemma 3.1 *An (n_1, n_2)-lasso-shaped word w belongs to $L(A)$ iff there exist $n_1 \leq p \leq n_1 + |Q|n_2$ and $1 \leq i \leq |C_{acc}|$, where $|C_{acc}|$ is the size of the maximal strongly connected component that contains the accepting state, such that $w[1, p] \in L(A_{\mathit{fin}})$, and $w[p + 1, p + i \cdot n_2] \in L(B_{\mathit{fin}})$.*

Proof: Assume first that $w \in L(A)$. We show that then there exist p and i as required. For that, we first define the function $pos : \mathbb{N} \rightarrow \{1, \dots, n_1 + n_2\}$, where for all $i \in \mathbb{N}$ we have

$$pos(i) = \begin{bmatrix} i & \text{if } i \leq n_1. \\ n_1 + 1 + ((i - (n_1 + 1)) \bmod n_2) & \text{if } i > n_1. \end{bmatrix}$$

Intuitively, since w is an (n_1, n_2)-lasso-shaped word, the letter $w(i)$, for all $i \in \mathbb{N}$, is equal to the letter $w(pos(i))$.

In order to reason about the runs of A on w, we consider the graph G that represents the possible runs of A on w. Formally, $G = \langle V, R\rangle$, where $V = Q \times \{1, \dots, n_1 + n_2\}$, and $R(\langle q, i\rangle, \langle q', i'\rangle)$ iff $q' \in \delta(q, w(i))$ and $i' = pos(i + 1)$. We say that a vertex $\langle q, i\rangle$ of G is accepting iff $q \in \alpha$. It is easy to see that w is accepted by A iff the graph G has an accepting vertex $\langle q_{acc}, p\rangle$ such that $\langle q_{acc}, p\rangle$ is reachable from $\langle q_0, 1\rangle$ and from itself.

The vertices of G can be partitioned to V_1 and V_2 with the corresponding induced subgraphs $G_1 = \langle V_1, R_1\rangle$ and $G_2 = \langle V_2, R_2\rangle$. The subgraph G_1 has $V_1 = Q \times \{1, \dots, n_1\}$, $R_1 = R \cap (V_1 \times V_1)$, and it represents the possible runs of A on the prefix of w. The subgraph G_2 has $V_2 = Q \times \{n_1 + 1, \dots, n_1 + n_2\}$, $R_2 = R \cap (V_2 \times V_2)$, and it represents the possible runs of A on the lasso of w. The transitions in $R \cap ((Q \times \{n_1\}) \times (Q \times \{n_1 + 1\}))$ connect the two subgraphs.

Since for $i \leq n_1 + 1$, we have $pos(i) = i$, the graph G_1 does not contain cycles. Since $w \in L(A)$, the graph G_2 contains at least one cycle that contains a vertex $\langle q_{acc}, j\rangle$ for some $n_1 + 1 \leq j \leq n_1 + n_2$, and $\langle q_{acc}, j\rangle$ is also reachable from the vertex $\langle q_0, 1\rangle$ Since the cycles in G_2 are induced by the lasso of w, the length of the cycle is divisible by n_2, and is bounded by $|Q|n_2$, which is the number of vertices in G_2. In order to prove the stronger bound on i we note that cycles that contain $\langle q_{acc}, p\rangle$ are induced by the lasso of w and also by the cycles in the graph of A. The length of (simple) cycles in the graph of A that contain the state q_{acc} is bounded by the size of the maximal strongly connected component C_{acc}, which contains q_{acc}.

Thus, the length of cycles that contain $\langle q_{acc}, p\rangle$ is bounded by $|C_{acc}|n_2$, as required. Since G_1 does not contain cycles, the length of the shortest path in G from $\langle q_0, 1\rangle$ to $\langle q_{acc}, j\rangle$ is at least n_1, in case the vertex $\langle q_{acc}, n_1\rangle$ is reachable in G, and is at most $n_1 + |Q|n_2$, in case the smallest j for which $\langle q_{acc}, j\rangle$ is reachable in G is $n_1 + |Q|n_2$. Let p be the length of this path. It follows that A has an accepting run on w that visits q_{acc} in steps p and $p + i \cdot n_2$, for $n_1 \leq p \leq n_1 + |Q|n_2$, and $q \leq i \leq |C_{acc}|$, thus $w[1, p] \in L(A_{fin})$ and $w[p + 1, p + i \cdot n_2] \in L(B_{fin})$.

Assume now that there exist $n_1 \leq p \leq n_1 + |Q|n_2$ and $1 \leq i \leq |C_{acc}|$ such that $w[1, p] \in L(A_{fin})$ and $w[p + 1, p + i \cdot n_2] \in L(B_{fin})$. Let r_1 be an accepting run of A_{fin} on $w[1, p]$ and let r_2 be an accepting run of B_{fin} on $w[p + 1, p + i \cdot n_2]$. We define the run r in the following way.

$$r(j) = \begin{bmatrix} r_1(j) & j \leq p. \\ r_2(p + (j - p)) \bmod i \cdot n_2 & j > p. \end{bmatrix}$$

Intuitively, once r reaches q_{acc} in position p, it loops there forever, following the behavior of r_2. It is easy to see that r is a legal and accepting run of A on w. $\Box$

So, by iterating over all the possible values of p and i, we can reduce the membership problem for Büchi automata and lasso-shaped infinite words to a sequence of membership tests for finite words. However, since the number of possible values for p depends on n_2, so is the query complexity of an algorithm that is based on such a reduction. In Section 4 we describe the long journey required in order to avoid this dependency in n_2.

4 The Algorithm

In this section we describe an ε-test for lasso-shaped words with respect to properties given by a nondeterministic Büchi automaton. The query complexity of the test is $\tilde{O}(1/\varepsilon)$. The basic idea is to use the ε-test of finite words as an oracle, and reduce an ε-test for an infinite word to a sequence of ε-tests for finite subwords of it. As explained in Section 3, Lemma 3.1 suggests such a reduction, only that the number of calls to the oracle depends on the length of the input word. We now describe how to avoid such a dependency. Fortunately, some of the techniques developed in [AKNS99] in order to ε-test finite words turned out to be useful also for bounding the number of calls to the algorithm in [AKNS99]. In particular, we need the following lemma about strongly connected graphs.

Lemma 4.1 [AKNS99] *Let $G = \langle V, E\rangle$ be a nonempty, strongly connected graph with a finite period g. Then there exists a partition of V to pairwise disjoint sets*

$V(G) = V_0, \dots, V_{g-1}$ and a constant $m \leq 3|V|^2$ such that:

(1) For every $0 \leq i, j \leq g-1$, and for every $u \in V_i$, $v \in V_j$, the length of every directed path from u to v in G is $(j-i) \bmod g$.

(2) For every $0 \leq i, j \leq g-1$, and for every $u \in V_i$, $v \in V_j$, and for every $l \geq m$ such that $l = (j-i) \bmod g$, there exists a directed path from u to v in G of length l.

The proof of Lemma 4.1 follows from the well known fact that for a set of integers $\{a_i\}$ with the greatest common divisor g, each large enough integer (greater that some number m that depends on $\{a_i\}$) that is divisible by g is a linear combination of the numbers $\{a_i\}$ with non-negative coefficients. The integers a_i are the lengths of cycles in G, and the constant m from Lemma 4.1 is called the *reachability constant* of G. It is smaller than the square of the maximal number among $g_1, \dots, g_k$ [Lew72,Dix90].

We use Lemma 4.1 in order to change the input word slightly in a way that enables us to restrict attention to runs of A that visit q_{acc} at specific positions whose number depends on the period of C_{acc} rather than on n_2. This involves two arguments. First, in Lemma 4.2 we show that when w is in $L(A)$, we can change w slightly so that A accepts the resulted word v by visiting q_{acc} at specific positions. Then, in Lemma 4.3, we prove that if w is ε-far from $L(A)$, then all words v that are slightly different from w cannot be accepted by runs that visit q_{acc} in these specific positions. In what follows, we fix g and m to denote the period and the reachability constant of C_{acc}, respectively. Also, let

$$D = \bigcup_{k \geq 0} \{n_1 + (|Q|+k)n_2, \dots, n_1 + (|Q|+k)n_2 + 2m + g - 1\}.$$

As illustrated in Figure 1, when we formalize in Lemmas 4.2 and 4.3 the notion of "change w slightly", we mean that w can be changed only in positions in D (the gray areas in Figure 1).

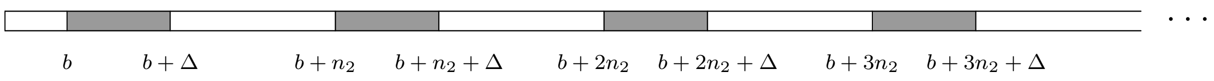

Fig. 1. The positions in which w and v may differ, with $b = n_1 + |Q|n_2$ and $\Delta = 2m + g - 1$.

Lemma 4.2 *Let $w \in L(A)$ be an (n_1, n_2)-lasso-shaped word with $n_2 > 2m + g$. There exists a lasso-shaped word $v \in L(A)$ that satisfies the following.*

(1) For all $j \notin D$, we have $w(j) = v(j)$, and

(2) The length of the prefix of v is p_v, where $n_1 + |Q|n_2 + m \leq p_v \leq n_1 + |Q|n_2 + m + g - 1$, and the length of the lasso of v is $i^v n_2$, where $1 \leq i^v \leq |C_{acc}|$, and we have $v[1, p_v] \in L(A_{fin})$ and $v[p_v + 1, p_v + i^v \cdot n_2] \in L(B_{fin})$.

Proof: Since $w \in L(A)$, then by Lemma 3.1 there exists a run r of A on w and integers $n_1 \leq p \leq n_1 + |Q|n_2$ and $1 \leq i \leq |C_{acc}|$ such that $r(p) = r(p + i \cdot n_2) =$

q_{acc}. In fact, the run constructed in the proof of Lemma 3.1 continues to loop at q_{acc}. Thus, $r(p+j) = r(p + (j \bmod (i \cdot n_2)))$ for all $j > 0$. Note that for all $j \geq p$, $r(j) \in C_{acc}$. Let $V_0, \ldots, V_{g-1}$ be a partition of C_{acc} as in Lemma 4.1. Let $q = r(n_1 + |Q|n_2)$. We know that $q \in C_{acc}$. Let $0 \leq i_1 \leq g-1$ be such that $q \in V_{i_1}$, let i_2 be such that $q_{acc} \in V_{i_2}$, and let $m \leq l \leq m+g-1$ be the integer for which $l = (i_2 - i_1) \bmod g$. Then by Lemma 4.1 there exists a path from q to q_{acc} of length l, implying that there is a word $v_1 \in \Sigma^*$ of length l that belongs to B_{fin}. Let p_1 be this path from q to q_{acc}. Let $q' = r(n_1+|Q|n_2+l)$, and let $q'' = r(n_1+|Q|n_2+l+m)$. There exists a path from q to q' of length l, thus $q' \in V_{i_2}$. There also exists a path from q' to q'' of length m, thus $q'' \in V_{i_3}$ for i_3 such that $m = (i_3 - i_2) \bmod g$. Thus, by Lemma 4.1, there exists a path of length m from q_{acc} to q''. Let p_2 be this path, and let v_2 be a word of length m that corresponds to this path. Now we define v as $w[1, n_1 + |Q|n_2] \cdot (v_1 \cdot v_2 \cdot w[n_1 + |Q|n_2 + m + l + 1, n_1 + (|Q|+1)n_2])^\omega$. Obviously, v differs from w only in letters that are located between the positions $n_1 + (k + |Q|)n_2$ and $n_1 + (k+|Q|)n_2 + m + l$, for all $k \geq 1$. By the construction of v, we have $v \in L(A)$, and there exists an integer p_v such that $n_1 + |Q|n_2 + m \leq p_v \leq n_1 + |Q|n_2 + m + g - 1$ and $v[q, p_v] \in L(A_{\mathit{fin}})$. The word v is $(n_1 + |Q|n_2, n_2)$-lasso-shaped, thus, according to Lemma 3.1, there exists an integer $1 \leq i^v \leq |C_{acc}|$ such that $v[p_v + 1, p_v + i^v \cdot n_2] \in L(B_{\mathit{fin}})$.

We note that $i^v = i$, where i is the parameter derived from Lemma 3.1 for the word w. To see this, consider the run r of A on w that satisfies the conditions of Lemma 3.1. Let r' be the run $r(1), \ldots, r(n_1 + |Q|n_2), (p_1 \cdot p_2, r(n_1|Q|n_2 + m + l + 1), \ldots, r(n_1 + (|Q|+1)n_2))^\omega$. From the construction of r', we have that r' is a legal run of A on v. The length of the loop of r' is the same as the length of the loop of r, and is equal to i. Thus, $i^v = i$. $\square$

In particular, Lemma 4.2 implies that $dist(w, v) \leq m + g$.

Lemma 4.3 *For each $0 < c < 1$, there exists $N_c \in \mathbb{N}$ such that for every (n_1, n_2)-lasso-shaped word w, if w is ε-far from $L(A)$, with $n_2 \geq N_c$, then all infinite words v satisfy one of the following.*

(1) There is $j \notin D$ such that $w(j) \neq v(j)$, or
(2) For all $n_1 + |Q|n_2 + m \leq p \leq n_1 + |Q|n_2 + m + g - 1$ and $1 \leq i \leq |C_{acc}|$, either $L(A_{\mathit{fin}})$ does not contain words of length p, or $v[p+1, p + i \cdot n_2]$ is $c \cdot \varepsilon$-far from $L(B_{\mathit{fin}})$.

Proof: Assume by way of contradiction that there exists $0 < c < 1$ such that for all $N \in \mathbb{N}$ there exist words w and v as follows:

- The word w is (n_1, n_2)-lasso-shaped,
- $n_1 \geq N$ and $n_2 \geq N$,
- the word v differs from w only in letters that are located between $n_1 + (k + |Q|)n_2$ and $n_1 + (k + |Q|)n_2 + 2m + g - 1$, for all $k \geq 0$, and
- there exist integers p and i such that $n_1 + |Q|n_2 + m \leq p \leq n_1 + |Q|n_2 + m + g - 1$

and $1 \leq i \leq |C_{acc}|$, v is $(p, i \cdot n_2)$-lasso-shaped, the language $L(A_{\mathit{fin}})$ contains words of length p, and $dist(v[p+1, p+i \cdot n_2], L(B_{\mathit{fin}})) < c \cdot \varepsilon \cdot i \cdot n_2$.

Let $N = \frac{(2m+g-1)}{\varepsilon(1-c)}$, and let w and v be the words as above. Let $p = n_1 + |Q|n_2 + l$. We know that $m \leq l \leq m+g-1$. We note that $dist(w, v) = dist(w[p+1, p+i \cdot n_2], v[p+1, p+i \cdot n_2]) \leq i(2m+g-1)$. Let u_1 be a word in $L(A_{\mathit{fin}})$ of length p. Recall that the distance $dist(v[p+1, p+i \cdot n_2], L(B_{\mathit{fin}}))$ is defined as the minimum of distances between $v[p+1, p+i \cdot n_2]$ and words in $L(B_{\mathit{fin}})$ of the same length. Let u_2 be the word in $L(B_{\mathit{fin}})$ where this minimum is reached. Then, u_2 is of length $i \cdot n_2$ and $dist(v[p+1, p+i \cdot n_2], u_2) < c \cdot \varepsilon \cdot i \cdot n_2$. Now consider the lasso-shaped word $u = u_1 \cdot (u_2)^\omega$. By Lemma 3.1, $u \in L(A)$. We compute $dist(w, u)$. The least common multiplier of n_2 and $i \cdot n_2$ is $i \cdot n_2$. Thus, by the definition of distance,

$$dist(w, u) = dist(w[p+1, p+i \cdot n_2], u[p+1, n_1 + i \cdot n_2]),$$

Since $dist(w[p+1, p+i \cdot n_2], v[p+1, p+i \cdot n_2]) \leq i(2m+g-1)$, we have that

$$dist(w, u) \leq dist(w[p+1, p+i \cdot n_2], v[p+1, p+i \cdot n_2]) + dist(v[p+1, p+i \cdot n_2], u[p+1, p+i$$

$$i(2m+g-1) + c \cdot \varepsilon \cdot i \cdot n_2.$$

Since $dist(w, u) \geq \varepsilon \cdot i \cdot n_2$, we receive that

$$i(2m+g-1) + c \cdot \varepsilon \cdot i \cdot n_2 > \varepsilon \cdot i \cdot n_2,$$

and thus

$$c > \frac{\varepsilon \cdot n_2 - (2m+g-1)}{\varepsilon n_2},$$

but we chose $c \leq \frac{\varepsilon \cdot n_2 - (2m+g-1)}{\varepsilon n_2}$, and thus we reach a contradiction. This concludes the proof of the lemma. $\square$

In fact, in the proof of Lemma 4.3, we show that $N_c = \frac{2m+g-1}{\varepsilon(1-c)}$.

We are now ready to prove the following theorem, which gives a reduction from infinite lasso-shaped words to finite words, where the complexity of the reduction is independent of the size of the input.

Theorem 4.4 *Consider a Büchi automaton $A = \langle \Sigma, Q, \delta, q_0, \{q_{acc}\} \rangle$. For every (n_1, n_2)-lasso-shaped word $w \in \Sigma^\omega$, the following hold.*

(1) If $w \in L(A)$, then there exists a lasso-shaped word $v \in L(A)$ as follows.
  (a) $dist(w, v) \leq 2m + g - 1$.

(b) For all $j \notin D$ we have $w(j) = v(j)$.

(c) There exist integers $n_1 + |Q|n_2 + m \leq p \leq n_1 + |Q|n_2 + m + g - 1$ and $1 \leq i \leq |C_{acc}|$, such that $v[1, p] \in L(A_{fin})$ and $v[p + 1, p + i \cdot n_2] \in L(B_{fin})$.

(2) If w is ε-far from $L(A)$, then for all lasso-shaped words $v \in \Sigma^\omega$ such that $w(j) = v(j)$ for all $j \notin D$, all integers p and i such that $n_1 + |Q|n_2 + m \leq p \leq n_1 + |Q|n_2 + m + g - 1$ and $1 \leq i \leq |C_{acc}|$, and all $0 < c \leq \frac{\varepsilon n_2 - (2m+g-1)}{\varepsilon n_2}\}$, either $L(A_{fin})$ does not contain words of length p, or $dist(v[p + 1, p + i \cdot n_2], L(B_{fin})) \geq c \cdot \varepsilon \cdot i \cdot n_2$.

Proof: We start with the case $w \in L(A)$. Let v be as in Lemma 4.2. Then v differs from w only in letters that are located between $n_1 + (k + |Q|)n_2$ and $n_1 + (k + |Q|)n_2 + m + g - 1$ for all $k \geq 0$, as required, and there exists an integer p such that $n_1 + n_2 + m \leq p \leq n_1 + |Q|n_2 + m + g - 1$ and there exists a run r_v of A on v such that $r_v(p) = q_{acc}$. Therefore, $v[1, p] \in L(A_{fin})$. In addition, there exists an integer $1 \leq i \leq |C_{acc}|$ such that $r_v(p + i \cdot n_2) = q_{acc}$. Let $r_v(p + 1) = q$. Clearly, q is a successor of q_{acc}. Furthermore, there is a path from q_{acc} to q and from q to q_{acc}. Thus, $v[p + 1, p + i \cdot n_2] \in L(B_{fin})$.

We now consider the case where w is ε-far from $L(A)$. Let $c \leq \frac{\varepsilon \cdot n_2 - (2m+g-1)}{\varepsilon \cdot n_2}$. Then, we have $n_2 > \frac{(2m+g-1)}{\varepsilon(1-c)}$. Then, according to Lemma 4.3, for all v that differ from w only in letters that are located between $n_1 + (k + |Q|)n_2$ and $n_1 + (k + |Q|)n_2 + m + g - 1$ for all $k \geq 0$, and for all integers p and i such that $n_1 + |Q|n_2 + m \leq p \leq n_1 + |Q|n_2 + m + g - 1$ and $1 \leq i \leq |C_{acc}|$, either $L(A_{fin})$ does not contain words of length p, or $v[p + 1, p + i \cdot n_2]$ is $c \cdot \varepsilon$-far from $L(B_{fin})$. □

By the definition of c, it is always greater than 0 and is smaller than 1. If we take $c > 1/2$, Theorem 4.4 holds for all n_2 that are greater than $\frac{3(2m+g-1)}{\varepsilon(1-c)}$.

Theorem 4.4 leads to our ε-test, which is described in Figure 2. The algorithm gets five parameters: a lasso-shaped word w, the length of the prefix of w, the length of the lasso of w, a Büchi automaton A, and $0 < \varepsilon \leq 1$. It invokes two algorithms: an algorithm Check_Length, which gets as input an automaton A_{fin} and an integer p and checks whether the language $L(A_{fin})$ contains words of length p, and an algorithm Fin_Test, which, from reasons we explain below, differs from the algorithm of [AKNS99]. The algorithm Check_Length is deterministic, and it checks whether A_{fin} contains words of length p by computing several *modulo* operations on p. We describe the algorithm Check_Length later in this section. The algorithm Fin_Test gets four parameters: a finite word w', the length of w', an automaton on finite words B_{fin}, and $0 < \varepsilon' \leq 1$. Like the algorithm in [AKNS99], when $w' \in L(B_{fin})$, it outputs "yes", and when w' is ε'-far from $L(B_{fin})$, it outputs "no" with probability at least $2/3$. We will describe the algorithm Fin_Test in more detail later. Essentially, Fin_Test is a simpler version of the algorithm in [AKNS99] that also discards from the random sample the letters of the input that are located in the gray areas in Figure 1. Thus, the algorithm Fin_Test gives the same answer for all words that differ one from another only in letters that are located in these areas. We note that Fin_Test can handle general alphabet and nondeterministic automata, nevertheless it is not hard to see that the algorithm in [AKNS99] can be applied also to general alphabets and nondeterministic automata.

```
procedure LS_Test $(w, n_1, n_2, A, \varepsilon)$
  for $p = n_1 + |Q|n_2 + m$ to $n_1 + |Q|n_2 + m + g - 1$ do
    if Check_Length $(A_{\mathit{fin}}, p)$ then
      for $i = 1$ to $|Q|$ do
        if Fin_Test $(w[p+1, p+i \cdot n_2], p, B_{\mathit{fin}}, \varepsilon - \frac{2m+g-1}{n_2})$ then return "yes";
  return "no".
```

Fig. 2. Testing of Lasso-Shaped Words

The algorithm LS_Test first calls the algorithm Check_Length with the automaton A_{fin} and the length p, for all $n_1 + |Q|n_2 + m \le p \le n_1 + |Q|n_2 + m + g - 1$. Thus, the first part of LS_Test invokes Check_Length at most g times. If LS_Test gets a positive answer for some p, it calls Fin_Test with the word $w[p+1, p+i \cdot n_2]$ of length $i \cdot n_2$, the automaton B_{fin}, and $\varepsilon' = \varepsilon - \frac{2m+g-1}{n_2}$, for all $1 \le i \le |C_{acc}|$. Now, since the the query complexity of Check_Length is 0, and the query complexity of Fin_Test is $\tilde{O}(1/\varepsilon')$, and since m, g, and $|Q|$ do not depend on w, the query complexity of LS_Test is $\tilde{O}(1/\varepsilon)$.

Readers not familiar with [AKNS99] may be happy at this point and wonder about the need to modify the algorithm in [AKNS99]. The need for it arises from three differences between our situation here and the situation in [AKNS99]. The first difference has to do with the fact that LS_Test calls Fin_Test several times, and it returns "yes" if for some p and i the language $L(A_{\mathit{fin}})$ contains words of length p and the corresponding call to Fin_Test returned "yes". Consequently, if we want to bound the probability of error of LS_Test by $1/3$, the probability of error of Fin_Test has to be much lower. The second difference comes from the fact that [AKNS99] considers general automata, that is, automata whose graphs consists of several strongly connected components, whereas in our case the graph of automaton B_{fin} is strongly connected (since the graph of B_{fin} is the maximal strongly connected component C_{acc} of A that contains the accepting state q_{acc}). The third difference has to do with the intervals in the set D of positions in which we allowed a modification of w.

Let us describe the algorithm Fin_Test. For $1 \le i \le \log(4m/\varepsilon)$ we define $r_i = \frac{2^{6-i} m \log 1/\varepsilon}{\varepsilon}$. The algorithm proceeds as follows.

1. For each $1 \le i \le \log(4m/\varepsilon)$ choose xr_i random subwords in w of length 2^{i+1} each, where $x = \lfloor -\log(1 - (\frac{2}{3})^{\frac{1}{\lceil C_{acc} \rceil \cdot g}})/2 \rfloor$.
2. Discard chosen subwords that intersect with D.
3. Check feasibility of the remaining subwords for the automaton B_{fin}. If all these subwords are feasible, accept. Otherwise (at least one infeasible subword is found), reject.

Now we prove that the probability of error of Fin_Test is bounded by $1/4^x$, where x is the parameter from the step **1** of the algorithm. Formally, we prove the following lemma.

Lemma 4.5 *the probability of error for Fin_Test is bounded by $1/4^x$, where x is the factor by which the algorithm increases the number of chosen subwords.*

Proof: Clearly, if the input word belongs to $L(B_{\mathit{fin}})$, then all its subwords are feasible and the algorithm always accepts. We now compute the probability of (erroneous) positive answer in case of w being ε-far from $L(A)$. We need to estimate the number of infeasible subwords among the chosen subwords. We use the following lemma.

Lemma 4.6 [AKNS99] *Assume that the language $L = L(A)$ contains some words of length n, and that A is essentially strongly connected. Let m be the reachability constant of A. Assume that $\varepsilon n \geq 64m\log(4m/\varepsilon)$. Then if for a word w of length $|W| = n$, $dist(w, L) \geq \varepsilon n$, then there exists an integer $1 \leq i \leq \log(4m/\varepsilon)$ such that the number of infeasible subwords of w of length 2^{i+1} is at least $\frac{2^{i-4}\varepsilon n}{m\log(4m/\varepsilon)}$.*

By Lemma 4.6, there exists $1 \leq i \leq \log(4m/\varepsilon)$ such that the number of infeasible subwords of w is at least $\frac{2^{i-4}\varepsilon n}{m\log(4m/\varepsilon)}$. At most $|C_{acc}|(2m + g - 1)$ of them may intersect with intervals $[n_1 + (x + |Q|)n_2, n_1 + (x + |Q|)n_2 + 2m + g - 1]$ for $1 \leq x \leq |C_{acc}|$. For sufficiently large n, we have that at least $\frac{2^{i-4}\varepsilon n}{m\log(4m/\varepsilon)} - |C_{acc}|(2m + g - 1) \geq \frac{2^{i-5}\varepsilon n}{m\log(1/\varepsilon)}$ of the infeasible subwords were not discarded. If a random sample contains one of these subwords, it provides a certificate for the fact that w does not belong to the language. A random sample of $x \cdot r_i$ subwords of length 2^{i+1} misses all these infeasible subwords with probability at most

$$(1 - \frac{1}{n}\frac{2^{i-5}\varepsilon n}{m\log(1/\varepsilon)})^{x \cdot r_i} < e^{-2x} = (e^{-2})^x < \frac{1}{4^x}. \tag{1}$$

□

It remains to show that for integer x such that $x = \lfloor -\frac{\log(1-(\frac{2}{3})^{\frac{1}{|C_{acc}|\cdot g}})}{2} \rfloor$ and for each w that is ε-far from $L(A)$, the algorithm LS_Test outputs "no" with probability at least $2/3$.

Let w be an input word that is ε-far from $L(A)$. The algorithm LS_Test invokes the procedure Check_Length with the automaton A_{fin} and the length $p \cdot g$ times for all possible values of p between $n_1 + |Q|n_2 + m$ and $n_1 + |Q|n_2 + m + g - 1$. In a case of the positive answer, LS_Test invokes Fin_Test with the automaton B_{fin} and the word $w[p + 1, p + i \cdot n_2]$ for all possible values of i between 1 and $|C_{acc}|$. By Lemma 4.3, for each p and i as above, either the language $L(A_{\mathit{fin}})$ does not contain words of length p, or $w[p+1, p+i \cdot n_2]$ is $c \cdot \varepsilon$-far from B_{fin}.

- Assume that the language $L(A_{\mathit{fin}})$ does not contain words of length p. The algorithm Check_Length returns the negative answer with probability 1 (since it is deterministic).
- Assume that $w[p+1, p+i \cdot n_2]$ is $c \cdot \varepsilon$-far from B_{fin} for all $1 \leq i \leq |C_{acc}|$. Then the algorithm LS_Test returns the negative answer iff all calls to Fin_Test with the automaton B_{fin} return the negative answer. Each call to Fin_Test returns the negative answer with probability at least $1 - \frac{1}{4^x}$, by Equation 1. Thus, $|C_{acc}|$ calls to Fin_Test return the negative answer with probability at least $(1 - \frac{1}{4^x})^{|C_{acc}|}$.

Thus, the probability of the negative answer for a specific value of p is at least $(1 - \frac{1}{4^x})^{|C_{acc}|}$, and therefore for g possible values of p the probability of the negative answer is at least $(1 - \frac{1}{4^x})^{|C_{acc}| \cdot g}$, which is at least $2/3$ by the choice of x. Thus, the probability of error for LS_Test with x as above is bounded by $1/3$.

We are now ready to prove the correctness of the algorithm LS_Test.

Theorem 4.7 *For a Büchi automaton $A = \langle \Sigma, Q, \delta, q_0, \{q_{acc}\} \rangle$, and an (n_1, n_2)-lasso-shaped word $w \in \Sigma^\omega$, if $w \in L(A)$, then the algorithm LS_Test always outputs "yes", and if w is ε-far from $L(A)$, then the algorithm LS_Test outputs "no" with probability at least* $2/3$.

Proof: Assume that $w \in L(A)$. Then, by Theorem 4.4, there exists a lasso-shaped word $v \in L(A)$ that is very close to w, and there exist $n_1 + |Q|n_2 + m \leq p \leq n_1 + |Q|n_2 + m + g$ and $1 \leq i \leq |C_{acc}|$ such that $v[1, p] \in L(A_{\mathit{fin}})$ (which means in particular that $L(A_{\mathit{fin}})$ contains words of length p), and $v[p+1, p+i \cdot n_2] \in L(B_{\mathit{fin}})$. Thus, LS_Test returns "yes" for v. It remains to show that LS_Test returns the same answer for w. Recall that LS_Test does not choose subwords that have letters in positions in D, and that v and w may differ only in letters in such positions. It follows that LS_Test does not distinguish between v and w, and we are done.

Assume now that w is ε-far from $L(A)$. Then, by Theorem 4.4, for all words v such that $w(j) = v(j)$ for all $j \notin D$, and for all p and i as above, either $L(A_{\mathit{fin}})$ does not contain words of length p, or $w[p+1, p+i \cdot n_2]$ is ε'-far from $L(B_{\mathit{fin}})$. In the former case we receive negative answer with probability 1 (since Check_Length is deterministic), or each one of the calls to Fin_Test returns the negative answer with probability $(1 - \frac{1}{4^x})$, where x is the factor by which we have increased the number of chosen subwords in step **1**. In particular, for $x = \lfloor -\log(1 - (\frac{2}{3})^{\frac{1}{|C_{acc}| \cdot g}})/2 \rfloor$, LS_Test returns "no" with probability at least $2/3$. $\Box$

Remark 4.8 The algorithm LS_Test can be extended to handle multiple accepting states by running it for each accepting state separately. This increases the running time by at most the size of α. The algorithm LS_Test, as well as the algorithm of [AKNS99], are described for the case of a single initial state. They can be extended to handle multiple initial states by running it for each initial state separately. This increases the running time by at most the number of initial states. □

4.0.0.1 Description of Check_Length The algorithm Check_Length is deterministic. For an automaton A_{fin} and an integer p it checks whether there exist words of length p accepted by A_{fin}. The check is done by considering all possible traversals of a word in the automaton $A_{\in fin}$ described by the list of traversed components and the sequence of entrance and exit states for each component. For each traversal, Check_length checks whether it can fit a word of length p. The algorithm uses Lemma 4.1 and the well known fact that for a set of integers $\{a_i\}$ with the greatest common divisor g, each large enough integer that is divisible by g is a linear combination of the numbers a_i with non-negative coefficients.

Each possible traversal of a word in the automaton A_{fin} can be described by a pair $\langle A, P\rangle$, where A is the list of traversed components and P is the sequence of entrance and exit states for each component. A pair is called *admissible* if the last component in A contains the accepting state q_{acc} (that is, admissible pairs correspond to accepting runs). Note that the number of admissible pairs depends only on the size of A_{fin}, and not on the size of the input word. Let $\langle A, P\rangle$ be an admissible pair, where $A = C_1, \dots, C_k$ is the list of traversed components, and $P = \{\langle p_2^i, p_1^{i+1}\rangle : 1 \leq i \leq k\}$ is the list of "portals". For each i, the state p_1^i is the entrance state to the component, and p_2^i is the exit state of the component. For the last component C_k, p_2^k is the state q_{acc}. We now use Lemma 4.1 for each component separately. For a component C_i, let g_i be the period of C_i, and let $V_0^i, \dots, V_{g_i-1}^i$ be the partition of the states of C_i according to Lemma 4.1. Let V_j^i be the set that contains p_1^i, and V_l^i be the set that contains p_2^i. By Lemma 4.1, all paths from p_1^i to p_2^i are of length $(l - j) \bmod g_i$. Let $rem_i = (l - j) \bmod g_i$ for the component C_i. Thus, a traversal $\langle A, P\rangle$ corresponds to a possible traversal of a word of length p in the automaton A_{fin} iff there exist integers $p_1, \dots, p_k$ such that $\sum_{i=1}^k p_i + (k-1) = p$ and $p_i = rem_i \bmod g_i$ for $1 \leq i \leq k$ (p_i is the length of the traversal in the component C_i, and the term $k - 1$ in the sum is the sum of lengths of edges that connect the components). Therefore, the check of existence of words of length p in the language $L(A_{\mathit{fin}})$ reduces to the check of whether there exist integers $x_1, \dots, x_k$ such that $p = \sum_{i=1}^k x_i g_i + \sum_{i=1}^k rem_i + (k - 1)$. It is well known that if g is the greatest common divisor of $g_1, \dots, g_k$, then there exists t such that each integer that is larger than t and is divisible by g is a linear combination of $\{g_1, \dots, g_k\}$ with nonnegative coefficients. Moreover, t is smaller than the square of the maximal number among $g_1, \dots, g_k$ [Lew72,Dix90]. That is, there exist $x_1, \dots, x_k$ as above iff $p - \sum_{i=1}^k rem_i - (k - 1)$ is divisible by g. Thus, the algorithm Check_Length checks whether $p - \sum_{i=1}^k rem_i - (k - 1)$ is divisible by g for each admissible pair $\langle A, P\rangle$, and returns "yes" if one such pair exists. We note that we assumed that all C_i are truly connected components. In the case that C_i is a state, we do not count it in the sum. The algorithm works correctly for all $p \geq 3|Q|^2$, and for smaller p all possible traversals can be checked directly.

4.1 *Query complexity and running time of the algorithm LS_Test*

The procedure Check_Length does not query the input word. The algorithm LS_Test executes the procedure Fin_Test at most $|C_{acc}g$ times. Each call to the procedure Fin_Test samples $x \cdot r_i$ subwords of length 2^{i+1} for all i between 1 and $\log(4m/\varepsilon)$. Recall that $x = \lfloor -\frac{\log(1-(\frac{2}{3})^{\frac{1}{|C_{acc}|\cdot g}})}{2} \rfloor$. Thus, the number of bits queried by Fin_Test is

$$\sum_{i=1}^{\log(4m/\varepsilon)} 2^{i+1}(-\frac{\log(1-(\frac{2}{3})^{\frac{1}{|C_{acc}|\cdot g}})}{2})\frac{2^{6-i}m\log(1/\varepsilon)}{\varepsilon} < (-\frac{\log(1-(\frac{2}{3})^{\frac{1}{|C_{acc}|\cdot g}})}{2})\frac{2^{8}m\log^2(1/\varepsilon)}{\varepsilon},$$

and the number of bits queried by LS_Test is bounded by the number of bits queried by Fin_Test multiplied by $(|C_{acc}|+1)g$.

The running time of the algorithm depends on the query complexity, and for each queried subword w of the input on the length of the feasibility check. Each such check involves checking whether there are words u and v of matching lengths such that uwv is accepted by the matching automaton on finite words. This check is done using Lemma 4.1 in case the words u and v are longer than m, or by checking all $|\Sigma|^m$ possible words in case one of the words u or v is shorter than m (see also [AKNS99]).

4.2 *Requirements for ε, n_1, and n_2*

Similarly to other property-testing algorithms, our algorithm works correctly for long input words. The restrictions on the size of the input word follow from the inequalities listed below.

(1) $c \leq \frac{\varepsilon\cdot n_2-(2m+g-1)}{\varepsilon\cdot n_2}$.
(2) $\varepsilon < (1/e)^{\log 4m+1}$.
(3) $n_2 > \frac{2^{5-i}m\log 1/\varepsilon \log 4m/\varepsilon |C_{acc}|(2m+g-1)}{\varepsilon}$.

5 Safety Languages

Of special interest in formal verification are *safety properties*, asserting that the observed behavior of the system always stay within some allowed region, in which nothing "bad" happens. Intuitively, a property ψ is a safety property if every violation of ψ occurs after a finite execution of the system. Consider a language L of infinite words over Σ. A finite word x over Σ is a *bad prefix* for L iff for all infinite words y over Σ, the concatenation $x \cdot y$ of x and y is not in L. Thus, a bad prefix for L is a finite word that cannot be extended to an infinite word in L. A language L is a *safety language* if every word not in L has a finite bad prefix [AS85]. For example, if $\Sigma = \{0, 1\}$, then $L = \{0^\omega, 1^\omega\}$ is a safety language. Indeed, every word not in L contains either the sequence 01 or the sequence 10, and a prefix that ends in one of these sequences cannot be extended to a word in L.

In this section we consider ε-testability of safety properties. Given a nondeterministic Büchi automaton A that recognizes a safety language (the latter can be checked in PSPACE [Sis94]), it is possible to construct a deterministic automaton A_{bad} on finite words that accepts exactly all the bad prefixes of $L(A)$ [KV99]. Clearly, an infinite word w belongs to L iff there is no finite prefix of w that belongs to $L(A_{bad})$. Lemma 5.1 below shows that for lasso-shaped words, it is enough to check one prefix.

Lemma 5.1 *Consider nondeterministic Büchi automaton A that recognizes a safety language. Let $|Q|$ be the number of states of the automaton A_{bad}, and let w be an (n_1, n_2)-lasso-shaped word. Then, $w \in L(A)$ iff $w[1, n_1 + |Q|n_2] \notin L(A_{bad})$.*

Proof: The idea of the proof is similar to the one used in Lemma 3.1, and is based on analyzing the possible combinations of a state of A_{bad} with a position of w. Note that if $x \in \Sigma^*$ is a bad prefix, then $x \cdot y$ is a bad prefix for all $y \in \Sigma^*$. Thus, it suffices to show that $w \notin L(A)$ iff there exists a bad prefix of w of length less or equal to $n_1 + |Q|n_2$.

Assume first that $w \notin L(A)$. Let x be the shortest bad prefix of w. The run of A_{bad} on w reaches an accepting state for the first time after $|x|$ steps. The same construction as in Lemma 3.1 shows that if an accepting state is reached, then it is reached within the first $n_1 + |Q|n_2$ steps of the run. Thus, $|x| \leq n_1 + |Q|n_2$. For the other direction, assume that there exists a prefix of w of length $m \leq n_1 + |Q|n_2$ such that $w[1, m] \in A_{bad}$. Then, w has a bad prefix, implying it is not in $L(A)$. $\square$

It follows that an ε-test for the membership of an (n_1, n_2)-lasso shaped word in the language of an automaton A that recognizes a safety property can invokes the ε-test of regular languages of [AKNS99] with the word $w[1, n_1 + |Q|n_2]$ and the automaton A_{bad}.

6 Lower Bound for Lasso-Shaped Infinite Words

In this section we prove a lower bound of $\Omega(1/\varepsilon)$ for query complexity of ε-testing of lasso-shaped infinite words. The proof is similar to the one presented in [AKNS99] for query complexity of ε-testing regular languages. We note that since testing infinite words is a harder problem than testing finite words, and given the fact that there is a lower bound of $\Omega(1/\varepsilon)$ on testing finite words, the same lower bound on testing infinite words is quite expected. However, since our testing algorithm of infinite words is restricted to lasso-shaped words, the lower bound does not follow immediately from the finite case.

The proof uses the renowned principle of Yao [Yao77], saying that if there exists a probability distribution on the union U of positive and negative examples such that any deterministic testing algorithm of query complexity d is correct with probability less than $2/3$ for an input randomly chosen from U according to this distribution, then d is a lower bound on the query complexity of any randomized testing algorithm.

As the proof of lower bound for testing finite words [AKNS99], the proof of lower bound for testing infinite words uses the algorithm that is presented in Section 4 in order to create the required probability distribution. The difference between the finite case and the lasso-shaped case stems from the fact that a finite word obtained by several unwindings of the lasso of length n has a particular structure: it is periodic with period n. The proof of [AKNS99] starts with a word of length m in the language and then changes it in randomly chosen εm places in order to obtain a word which is far from the language. In our case, $m = i \cdot n$ for some $i \geq 1$, however in order to preserve the lasso structure of the input words, we have to perform our random choice of runs not from m, but from n. We start with the following lemma, that demonstrates this idea on a very simple language.

Lemma 6.1 *Let $L = \{1^\omega\}$ be the language over the alphabet $\{0, 1\}$. Then, for any n, an ε-test of a $(0, n)$-lasso-shaped word has query complexity of at least $\frac{1}{3\varepsilon}$.*

Proof: The proof is essentially the same as in [AKNS99] for finite words. Clearly, L is recognized by a Büchi automaton. We define a distribution on the set of positive and negative instances with a prefix of length 0 and a lasso of length n as follows. The word $(1^n)^\omega$ gets the probability $1/2$. Next we partition the index set $[1, \ldots, n]$ into $t = 1/\varepsilon$ parts $I_1, \ldots, I_t$, each of size εn, and for each $1 \leq i \leq t$ we assign the probability $1/(2t)$ to the word y_i that is created from $(1^n)^\omega$ by flipping all bits in I_i from 1 to 0. It is easy to see that $dist(y_i, L) = \varepsilon n$, thus all y_i are negative instances. Now we apply the Yao principle. Let A be a deterministic ε-testing algorithm with query complexity d. If A gives the wrong answer on $(1^n)^\omega$, then it is incorrect with probability at least $1/2$. Otherwise, it should accept the input if all d tested bits are equal to 1. Then it accepts also at least $t - d$ of the inputs y_i. Therefore, A gives an incorrect answer with probability at least $(t - d)/(2t) < 1/3$, implying $d > t/3 = 1/3\varepsilon$. $\square$

The proof of Lemma 6.1 can be generalized to all non-trivial languages that are recognized by Büchi automata in essentially the same way as it is done in [AKNS99]. First, we formally define non-trivial languages.

Definition 6.2 *A language L that is recognized by a Büchi automaton is* non-trivial *if it is infinite, and there exists a constant* $0 < \varepsilon_L < 1$ *such that for all* $n_1, n_2 \geq 0$ *there exists an* (n_1, n_2)*-lasso-shaped word* w *such that* $dist(w, L) \geq (\varepsilon_L n_1, \varepsilon_L n_2)$.

Now we prove the lower bound of $\Omega(1/\varepsilon)$ for the query complexity of non-trivial languages. Essentially, the proof is a generalization of the idea used in Lemma 6.1. For a lasso-shaped word w in the language L, we construct a set of lasso-shaped words v that are ε-far from w by randomly choosing a set of subwords of total length $O(\varepsilon n)$, where n is the length of the lasso, and changing w in these places such that the resulting set "fools" any deterministic algorithm that asks less than $\Omega(1/\varepsilon)$ queries. The main difference between our construction and the construction used in [AKNS99] is that all words v should be lasso-shaped with lasso of length n.

Lemma 6.3 *For each non-trivial language on infinite words L, for all sufficiently small* $\varepsilon > 0$*, any* ε*-testing algorithm for* L *requires* $\Omega(1/\varepsilon)$ *queries.*

Proof: The idea that we use to prove this bound is similar to the one used in [AKNS99]. We choose a positive example with lasso of size n, and change it in randomly chosen εn places, thus creating a negative instance which is hard to distinguish from the positive one.

Let L be a non-trivial language that is recognized by a Büchi automaton. Let w be an $(0, n)$-lasso-shaped word that is ε_0-far from L. Recall that the algorithm LS_Test invokes the algorithm Check_Length for $w[1, p]$ and then the algorithm Fin_Test for $w[p+1, p+i \cdot n_2]$ for all $n_1 + |Q|n_2 + m \leq p \leq n_1 + |Q|n_2 + m + g - 1$ and $1 \leq i \leq |C_{acc}|$, where n_1 is the length of the prefix of w and n_2 is the length of the lasso. In our case, LS_Test invokes Check_Length for all $|Q|n + m \leq p \leq |Q|n + m + g - 1$. Since L is non-trivial, the algorithm Check_Length accepts at least for one value of p. The algorithm Fin_Test chooses a random set of subwords and checks the feasibility of each subword with respect to the corresponding language on finite words. The algorithm has one-sided error, that is, in case of $w \in L$, the algorithm always accepts, and in case of w that is ε_0-far from L the algorithm rejects with probability at least $2/3$. Thus, if we choose a random set of subwords as above, it will cause the algorithm to reject w with probability at least $2/3$ and will not coincide with any word $u \in L$.

According to Theorem 4.4, if w is ε_0-far from L, then there exists a constant $0 < c < 1$ such that either there are no words of length p in L_{fin}, or $w[p+1, p+i \cdot n_2]$ is $c \cdot \varepsilon_0$-far from L^q_{fin} for all $|Q|n + m \leq p \leq |Q|n + m + g - 1$, $1 \leq i \leq |Q|$, and $q \in S$, where L_{fin} denotes the corresponding language on finite words, and L^q_{fin} denotes the language on finite words that is limited to C_{acc} with the initial state q. It is enough to check the case where Check_Length accepts. That is, $w[p+1, p+i \cdot n]$ is ε_0-far from L^q_{fin}. This implies that with a probability at least $2/3$ a random choice of a set of runs built as described in the Fin_Test algorithm and having total length $\tilde{O}(1/\varepsilon_0)$, will cause the algorithm to reject $w[p+1, p+i \cdot n]$ (and thus also w).

Since the algorithm has one-sided error, a random set of runs with probability $2/3$ will cause the algorithm to reject w and will not coincide with any word in L.

Each such run is a set of intervals in $\{p+1, \ldots, p+i \cdot n\}$ of total length bounded by $\tilde{O}(1/\varepsilon_0)$. Let u be a $(0, n)$-lasso-shaped word in L. Our goal is to create a probability distribution over positive and negative instances of L, where u is the positive instance, and negative instances are created from u by changing the bits of u, corresponding to some random run, to those of w. The difficulty that arises here and does not exist in [AKNS99] is that the changed words should also be $(0, n)$-lasso-shaped. That is, not only the chosen runs should be pairwise disjoint, but also their projections on $[1, n]$ should be pairwise disjoint, since for every $1 \leq i \leq n$ and $1 \leq j \leq |Q|$, the indices i and $i + jn$ correspond to the same position on the lasso. Note, however, that for a subword of length r there are at most $r \cdot |Q|$ subwords that correspond to the same position on the lasso. Thus, choosing ε small enough we can ensure that we have sufficiently many sets of runs that are both pairwise disjoint and for each run all subwords in it correspond to different positions on the lasso.

The rest of the proof is as in [AKNS99]. We construct $t = c/\varepsilon$ families of runs S_i, each of cardinality εn, where the constant c depends only on ε_0. For each S_i as above we define a word w_i that is obtained from u by changing the bits of u that correspond to S_i to those of w. Clearly, each w_i is ε-far from L. The distribution is defined as follows. The word u gets the probability $1/2$, and each of the t words w_i gets the probability $1/(2t)$. Then, a deterministic testing algorithm has to query at least $\Omega(t) = \Omega(1/\varepsilon)$ bits of the input word in order to answer correctly with probability at least $2/3$. The lemma now follows from Yao principle.

□

7 Discussion

The main technical contribution of this paper is an ε-test for ω-regular languages and lasso-shaped words. This result is an extension of the ε-test for regular languages presented in [AKNS99]. The extension is not trivial. In fact, already the definition of distance, which is straightforward for finite words, involves subtle considerations, and discussion has to be restricted to infinite words with a finite representation. We describe a reduction from ε-test of infinite words to a constant number of ε-tests for finite words. The main difficulty that is posed by the fact the word is infinite is that, unlike the case of finite words, we do not know the position in the word in which an accepting run of A on the word visits an accepting state. For general (not lasso-shaped) words, this difficulty cannot be circumvented. We show that for lasso-shaped words, we can bound the number of positions where an accepting run can visit an accepting state. Moreover, we show that a word in the language of A can be modified slightly to ensure that an accepting run would visit an accepting state inside a specific interval of a constant length.

Today's rapid development of complex and safety-critical systems requires *formal verification* methods. In *model checking*, we verify that a system meets a desired behavior by executing an algorithm that checks whether a mathematical model of the system satisfies a formal specification that describes the behavior. The systems we verify are non-terminating, and their specifications describe an on-going behavior of the system (c.f., "every request is eventually granted"). The algorithmic nature of model checking makes it fully automatic, and thus attractive to practitioners. At the same time, model checking is very sensitive to the size of the mathematical model of the system. Commercial model-checking tools need to cope with the exceedingly large state spaces that are present in real-life designs, making the so-called *state-explosion problem* perhaps the most challenging issue in computer-aided verification [CGP99].

Almost all previous efforts and heuristics for coping with the state-explosion problem, such as symbolic methods [McM93] and modular verification [dRLP98], do not deviate from the strict definition of model checking, where the algorithm is not allowed to err. Consequently, complexity lower bounds for the model-checking problem apply also for these heuristics. We believe that a major improvement of currently used heuristics should involve a deviation from the strict definition of model checking. The setting of property testing seems very appealing for this task: the specifications are small, the systems are exceedingly large, and it is the complexity in terms of the system that we wish to bound, which is exactly what property testing does. Indeed, the complexity of ε-testing algorithms depends only on ε and on the size of the property, and is independent of the size of the input.

The one-sided error allowed in property testing means that if the system is correct, the testing algorithm always say it is correct, yet when the system is incorrect, the testing algorithm may say it is correct. This at first seems like the unfortunate side, as a model-checking algorithm that reports the correctness of an incorrect system may be more harmful than an algorithm that reports the incorrectness of a correct system. It is now agreed, however, that the anticipated goal of formal verification, of providing a "correctness stamp" on the system, has turned out to involve too many obstacles (e.g., exact modeling of the system, comprehensive specification, etc.), and the primary use of model checking nowadays is *falsification*. There, as in debugging, the goal is to detect errors, rather than to serve as a correctness stamp [BCCZ99,Sip99]. The one-sided error of property testing fits well in this approach: whenever the testing algorithm reports a dissatisfaction of the property, an error indeed exists. In addition, as with all other randomized algorithms, repeated runs can be used in order to reduce the error to any desirable constant.

Naturally, the ultimate goal of applying property testing to model checking is an ε-test that gets as input a Kripke structure that models an entire system (rather than a single computation), and distinguishes between the case where the system satisfies a specification and the case where it is far from satisfying it. Technically, this can be achieved by extending the ε-test here to *ω-regular tree languages*. Our efforts in this direction are still unsuccessful. The difficulty is already in an extension of [AKNS99] to regular tree languages, and it lies in the fact that local changes in the tree influence several paths in it. While the ultimate goal seems hard to achieve, some helpful applications can arise already from our result here.

Our ε-test can replace model checking of lasso-shaped words. While the complexity of the ε-test is independent of n_1 and n_2, the best time complexity known for the model-checking problem of (n_1, n_2)-lasso-shaped words is $O((n_1 + n_2) * m)$, where m is the size of the specification, represented as a nondeterministic Büchi automaton. The restriction to lasso-shape words is not really restrictive in the context of model checking: as proven in [CD88], if a system violates an LTL property ψ, there is a lasso-shaped computation of the system that violates ψ. In addition, counter examples for LTL properties (that is, descriptions of computations that violate the specification) are given by model-checking tools in terms of lasso-shaped computations [VW94], and several *random simulation* algorithms that are based on sampling and checking individual computations of the system consider, or can be easily modify to consider, lasso-shaped words [Wes89,Hol91].

Our ε-test, however, has an additional crucial requirement on the input, namely the ability to perform local queries of the input in a constant time (also known as *random access* to the input). On the other hand, current random-simulation methods construct the sampled computations "on-the-fly," and a naive application of our algorithm on top of them involves storing of the sampled computation and would cause the time and space complexity to depend on the size of the input word. Still, our ε-test is useful in combination with a random simulator as running the model checker on the output of the random simulator causes a significant slow-down in the performance of the simulator [Fix02]. Thus, storing the sampled computations and then running our ε-test on them seems a promising way to reduce the complexity of model checking of sampled computations.

Acknowledgment We thank David Dill, Limor Fix, Ilan Newman, and Dana Ron for helpful discussions.

References

[AK99] N. Alon and M. Krivelevich. Testing k-colorability. *SIAM Journal on Discrete Mathematics*, 1999. accepted for publication.

[AKNS99] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. In *Proc. 40th IEEE Symposium on*

Foundations of Computer Science, pages 645–655, 1999.

[AS85] B. Alpern and F.B. Schneider. Defining liveness. *Information processing letters*, 21:181–185, 1985.

[BCCZ99] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Analysis and Construction of Systems*, volume 1579 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[BR00] M.A. Bender and D. Ron. Testing acyclicity of directed graphs in sublinear time. In *Proc. 27th International Colloqium on Automata, Languages, and Programming*, 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.

[CD88] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 428–437. Springer-Verlag, 1988.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[Dix90] J. Dixmier. Proof of a conjecture by Erdös and Graham concerning the problem of Frobenius. *Journal of Number Theory*, 34(2):198–209, 1990.

[dRLP98] W-P. de Roever, H. Langmaack, and A. Pnueli, editors. *Compositionality: The Significant Difference. Proceedings of Compositionality Workshop*, volume 1536 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[Fix02] L. Fix. Application of ε-test to random simulation. Private communication, 2002.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of ACM*, 45(4):653–750, 1998.

[GR97] O. Goldreich and D. Ron. Property testing in bounded degree graphs. In *Proc. 25th ACM Symposium on the Theory of Computing*, pages 406–415, 1997.

[GR99] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[GT03] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. *Random Struct. Algorithms*, 23(1):23–57, 2003.

[Hol91] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International Editions, 1991.

[KKR03] T. Kaufman, M. Krivelevich, and D. Ron. Tight bounds for testing bipartiteness in general graphs. In *Proceedings of 7th International Workshop on Randomization and Approximation Techniques in Computer Science*, volume 2764 of *Lecture Notes in Computer Science*, pages 341–353, Princeton, NY, August 2003. Springer-Verlag.

[KV99] O. Kupferman and M.Y. Vardi. Model checking of safety properties. In *Computer Aided Verification, Proc. 11th International Conference*, volume 1633 of *Lecture Notes in Computer Science*, pages 172–183. Springer-Verlag, 1999.

[Lan69] L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.

[Lew72] M. Lewin. A bound for a solution of a linear Diophantine problem. *Journal of London Mathematical Society*, 6(2):61–69, 1972.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

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

[PR99] M. Parnas and D. Ron. Testing the diameter of graphs. In *Proc. of RANDOM*, pages 85–96, 1999.

[PRR01] M. Parnas, D. Ron, and R. Rubinfeld. Testing parenthesis languages. In *Proc. 5th International workshop on randomization and approximation techniques in computer science*, pages 261–272, 2001.

[PRS01] M. Parnas, D. Ron, and A. Samorodnitsky. Proclaiming dictators and juntas or testing boolean formulae. In *Proc. 5th International workshop on randomization and approximation techniques in computer science*, pages 273–284, 2001.

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

[RS96] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.

[Sip99] H.B. Sipma. *Diagram-based Verification of Discrete, Real-time and Hybrid Systems*. PhD thesis, Stanford University, Stanford, California, 1999.

[Sis94] A.P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.

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

[Wes89] C.H. West. Protocol validation in complex systems. In *Proc. 8th Symp. on Principles of Distributed Computing*, pages 303–312, 1989.

[Yao77] A.C. Yao. Probabilistic computation, towards a unified measure of complexity. In *Proc. of the 18th IEEE Symp. on Foundations of Computer Science*, pages 222–227, 1977.