Regular Languages are Testable with a Constant Number of Queries

Noga Alon *
Department of Mathematics,
Tel Aviv University,
Tel Aviv 69978, Israel
noga@math.tau.ac.il

Michael Krivelevich †
DIMACS Center
Rutgers University
Piscataway, NJ 08854, USA
mkrivele@dimacs.rutgers.edu

Ilan Newman
Department of Computer Science
University of Haifa, Israel
ilan@mathcs2.haifa.ac.il

Mario Szegedy
AT&T Labs-Research
Florham Park, NJ 07932, USA
ms@research.att.com

Abstract

We continue the study of combinatorial property testing, initiated by Goldreich, Goldwasser and Ron in [7]. The subject of this paper is testing regular languages. Our main result is as follows. For a regular language $L \in \{0,1\}^*$ and an integer n there exists a randomized algorithm which always accepts a word w of length n if $w \in L$, and rejects it with high probability if w has to be modified in at least ϵn positions to create a word in L. The algorithm queries $\tilde{O}(1/\epsilon)$ bits of w. This query complexity is shown to be optimal up to a factor poly-logarithmic in $1/\epsilon$. We also discuss testability of more complex languages and show, in particular, that the query complexity required for testing context-free languages cannot be bounded by any function of ϵ. The problem of testing regular languages can be viewed as a part of a very general approach, seeking to probe testability of properties defined by logical means.

1 Introduction

Property testing deals with the question of deciding whether a given input x satisfies a prescribed property P or is "far" from any input satisfying it. Let P be a property, i.e. a non-empty family of binary words. A word w of length n is called ϵ-far from satisfying P, if no word w' of the same length, which differs from w in no more than ϵn places, satisfies P. An *ϵ-test* for P is a randomized algorithm, which given the quantity n and the ability to make queries about the value of any desired bit of an input word w of length n, distinguishes with probability at least $2/3$ between the case of $w \in P$ and the case of w being ϵ-far from satisfying P. Finally, we say that property P is *testable* if for every fixed $\epsilon > 0$ there exists an ϵ-test for P whose total number of queries is bounded only by a function of ϵ, which is independent of the length of the input word.

Property testing was defined by Goldreich et. al [7] (inspired by [13]). It emerges naturally in the context of PAC learning, program checking [6, 3, 10, 13], probabilistically checkable proofs [2] and approximation algorithms [7].

In [7], the authors mainly consider graph properties, such as bipartiteness and show (among other things) the quite surprising fact that testing bipartiteness can be done by randomly testing a polynomial in $1/\epsilon$ number of edges of the graph, answering the above question with constant probability of failure. They also raise the question of obtaining general results as when there is, for every $\epsilon > 0$, an ϵ-test for a property using $c(\epsilon)$ queries with constant probability of failure. We call properties of this type *ϵ-testable*. So far, such answers are quite sparse; some interesting examples are given in [7], several additional ones can be obtained by applying the Regularity Lemma as we show in a subsequent paper [1].

In this paper we address testability of formal languages (see [8] as a general reference). A language $L \subseteq \{0,1\}^*$ is a property which is usually viewed as a sequence of Boolean functions $f_n : \{0,1\}^n \to \{0,1\}$, with $f_n^{-1}(1) = L \cap \{0,1\}^n = L_n$. Our main result states that all regular languages are testable with query complexity only $\tilde{O}(1/\epsilon)$. We also show that this complexity is optimal up to a factor poly-logarithmic in $1/\epsilon$. This positive result cannot be extended

*Research supported by a USA Israeli BSF grant, by a grant from the Israel Science Foundation and by the Hermann Minkowski Minerva Center for Geometry at Tel Aviv University. Part of this research was performed during a visit at AT&T Labs-Research.

†Supported by a DIMACS postdoctoral fellowship. Part of this research was performed during a visit at AT&T Labs-Research.

to context-free languages, for there is an example to a very simple context-free language which is not testable.

Since regular languages can be characterized using second order monadic logic, we thus obtain a large set of logically defined objects which are testable. In [1] we provide testable graph properties described by logical means as well. These results indicate a strong interrelation between testability and logic. Although our result on regular languages can be viewed as a separate result having no logical bearing at all, our opinion is that logic does provide the right context for testability problems, which may lead to the discovery of further classes of testable properties.

The rest of this paper is organized as follows. In Section 2 we present the proof of the main result showing that every regular language is testable. In Section 3 we show that the upper bound of $\tilde{O}(1/\epsilon)$ for query complexity of testing regular languages, obtained in Theorem 1, is tight up to a polylogarithmic factor. Section 4 is devoted to the discussion of testability of context-free languages. There we show in particular that there exist non-testable context-free languages. The final Section 5 contains some concluding remarks and outlines new research directions.

2 Testing Regular Languages

In this section we prove the main result of the paper. Whenever needed, n is assumed to be large enough and $\epsilon > 0$ to be small enough. All logarithms are binary unless stated explicitly otherwise.

We start by recalling the standard definition of a regular language, based on finite automatons. This definition is convenient for algorithmic purposes.

Definition 2.1 *A deterministic finite automaton (DFA) M over $\{0,1\}$ with states $Q = \{q_1, \ldots, q_m\}$ is given by a function $\delta : \{0,1\} \times Q \to Q$ together with a set $F \subseteq Q$. One of the states, q_1 is called the initial state. The states belonging to the set F are called accepting states, δ is called the transition function.*

We can extend the transition function δ to $\{0,1\}^*$ recursively as follows:

$$\begin{aligned} \delta(\emptyset, q) &= q; \\ \delta(u0, q) &= \delta(0, \delta(u, q)) \\ \delta(u1, q) &= \delta(1, \delta(u, q)). \end{aligned}$$

We then say that M accepts a word u if $\delta(u, q_1) \in F$. M rejects u means that $\delta(u, q_1) \in Q \setminus F$. Finally, the language accepted by M, denoted by L_M, is the set of all $u \in \{0,1\}^*$ accepted by M. We use the following definition of regular languages:

Definition 2.2 *A language is regular iff there exists a finite automaton that accepts it.*

Therefore, we assume in this section that a regular language L is given by its automaton M so that $L = L_M$.

A word w of length n defines a sequence of states $(q_{i_0}, \ldots, q_{i_n})$ in the following natural way: $q_{i_0} = q_1$, and for $1 \leq j \leq n$, $q_{i_j} = \delta(w[1] \ldots w[j], q_1)$. This sequence describes how the automaton M moves while reading w. Later in the paper we will occasionally refer to this sequence as the *traversal path* of w.

A finite automaton M defines a directed graph $G(M)$ by $V(G(M)) = Q$ and $E(G(M)) = \{(q_i, q_j) \mid \delta(0, q_i) = q_j\} \cup \{(q_i, q_j) \mid \delta(1, q_i) = q_j\}$. The *period* $g(G)$ of a directed graph G is the greatest common divisor of cycle lengths in G. If G is acyclic, we set $g(G) = \infty$.

We will use the following lemma about directed graphs.

Lemma 2.3 *Let $G = (V, E)$ be a strongly connected directed graph with a finite period $g(G)$. Then there exist a partition $V(G) = V_0 \cup \ldots V_{g-1}$ and a constant $m = m(G)$ which does not exceed $3|V|^2$ 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 integer $n \geq m$, if $n = (j-i) \bmod g$, then there exists a directed path from u to v in G of length n.*

Proof. To prove part 1, fix an arbitrary vertex $z \in V$ and for each $0 \leq i \leq g-1$, let V_i be the set of all those points which are reachable from v by a directed, (not necessarily simple), path of length $i \bmod g$. Note that since any closed (directed) walk in G is a disjoint union of cycles, the length of each such walk is divisible by g. This implies that the sets V_i are pairwise disjoint. Indeed, assume this is false and suppose w lies in $V_i \cap V_j$ with $i \neq j$. As G is strongly connected there is a path p_1 from w to z, and by definition there is a path p_2 of length $i \bmod g$ from z to w as well as a path p_3 of length $j \bmod g$ from z to w. Now the number of edges of either $p_1 \cup p_2$ or $p_1 \cup p_3$ is not divisible by g, which is impossible. Therefore the sets V_i form, indeed, a partition of V. For $u \in V_i$ and $v \in V_j$, the union of any (directed) path from z to u with a (directed) path from u to v forms a path from z to v, and as any such path must have length $j \bmod g$ the assertion of part 1 follows.

We next prove part 2. It is well known that for any set of positive integers a_i whose greatest common divisor is g there is a smallest number t such that every integer $s \geq t$ which is divisible by g is a linear combination with non-negative coefficients of the numbers a_i. Moreover, it is known (see [9], [5]), that t is smaller than the square of the maximal number a_i. Fix a closed (directed) walk in G, that visits all vertices and whose length is at most $|V|^2$. (This is easily obtained by numbering the vertices of G arbitrarily as $v_0, v_1, \ldots, v_{r-1}$

and by concatenating directed paths from v_i to v_{i+1} for each $0 \leq i \leq r-1$, where the indices are taken modulo r.) By following this closed walk and by traversing each directed cycle as many times as desired, we conclude that every integer which is divisible by g and exceeds $2|V|^2$ is a length of a closed walk passing through all vertices of the graph. Given, now, a vertex $u \in V_i$, a vertex $v \in V_j$ and an integer $n > 3|V|^2$ satisfying $n = (j-i) \bmod g$, fix a shortest path p from u to v, and note that its length l satisfies $l = (j-i) \bmod g$ and $l < |V| (\leq |V|^2)$. Adding to p a closed walk of length $n-l$ from v to itself we obtain the required path, completing the proof. □

We call the constant m from the above lemma the *reachability constant* of G and denote it by $m(G)$. In the sequel we assume that m is divisible by g.

Using Lemma 2.3, for a given n it is easy to check (basically by calculating $n \bmod g$) whether the language L_M contains any words of length exactly n. If $L_M \cap \{0,1\}^n = \emptyset$, a testing algorithm can reject any input without reading it at all. Therefore, we can assume that we are in the nontrivial case $L_M \cap \{0,1\}^n \neq \emptyset$.

Given a word $w \in \{0,1\}^n$, a sub-word (run) w' of w starting at position i is called *feasible* for language L_M, if there exists a state $q \in Q$ such that q is reachable from q_1 in G in exactly $i-1$ steps and there is a path in G from the state $\delta(w', q)$ to at least one of the accepting states. Otherwise, w' is called *infeasible*. Using Lemma 2.3 it is easy to check in time $O(|Q||w'|)$ whether a given run w' is feasible. Of course, finding an infeasible run in w proves that $w \notin L$. Our aim is to show that if a given word w of length n is far from any word of length n in L, then many short runs of w are infeasible. Thus a choice of a small number of random runs of w almost surely contains an infeasible run. First we treat the following basic case:

Lemma 2.4 *Let M satisfy the following assumptions:*

1. *M has a unique accepting state q_{acc};*
2. *The set of states of the automaton, Q, can be partitioned into two parts, C and D so that*
   - *$q_1, q_{acc} \in C$;*
   - *the subgraph of $G(M)$ induced on C is strongly connected;*
   - *no edges in $G(M)$ go from D to C (but edges can go from C to D).*

   (Note that D may be empty.)

Let m be the reachability constant of $G[C]$. Assume that the language $L = L_M$ contains some words of length n. If a word w of length $|w| = n$ satisfies $dist(w, L) \geq \epsilon n$, then there exists an integer $1 \leq i \leq \log_2(6m/\epsilon)$ such that the number of infeasible runs of w of length 2^{i+1} is at least $\frac{2^{i-4}\epsilon n}{m \log_2(\frac{1}{\epsilon})}$.

Proof. Our intention is to construct a sequence $(R_j)_{j=1,\ldots}$ of disjoint feasible runs such that none of these runs (but the last one) can be extended forward, i.e. if we add the bit of w immediately following R_j, the resulting run, R'_j is infeasible. For reasons to become obvious later we also want these runs to be in the interval $[m+1, n-m]$.

A natural way to construct such a sequence is to repeat the following procedure starting from coordinate $m+1$. For $j = 1, \ldots$ originally the current run, R_j is empty. If the addition of the current bit to R_j creates a feasible run, then we add this bit to R_j and move the current pointer one position to the right. In the opposite case we still move the current pointer one position to the right, increase j by one, and start a new empty current run. We finish the procedure, when the current pointer reaches $n-m+1$. Note that for some j-s R_j may be empty. Denote by h the number of runs found by the above algorithm. Clearly,

$$|w| = n = 2m + \sum_{j=1}^{h} (1 + |R_j|).$$

For every $1 \leq j \leq h$ fix a state $q_{i_j} \in C$ so that $\delta(R_j, q_{i_j}) \in C$ and such that q_{i_j} is reachable from q_1 in $c_j - 1$ steps, where c_j is the starting position of run R_j (recall that even empty R_j-s have starting positions).

Next we inductively construct a word $w^* \in L$ such that $dist(w, w^*) \leq 3hm$. We maintain during the induction that for $j = 1, \ldots$ the word w_j we construct is feasible starting from position 1, and it ends in position $c_j - 1$. Take w_0 to be a word of length m which is feasible starting from position 1. Assume we have already defined a word w_{j-1} feasible from position 1 and ending in position $c_j - 1$. Let $\delta(w_{j-1}, q_1) = p_j$. As both p_j and q_{i_j} are reachable from q_1 by a path of length $c_j - 1$, according to Lemma 2.3 we can change the last m bits in w_{j-1} so that we get a word u_j for which $\delta(u_j, q_1) = q_{i_j}$. If $j < h$ we define w_j as a concatenation of u_j, R_j, and an extra bit b_j such that $\delta(u_j R_j b_j, q_1) \in C$. Note that since C is strongly connected such a b_j always exists. In the case $j = h$ we define w_h as a concatenation of u_h, R_h, and a word v of length m so that $\delta(u_h R_h v, q_1) = q_{acc}$. Such v exists because of Lemma 2.3, our assumption that L contains at least one word of length n, and because $\delta(u_h R_h, q_1) \in C$. Since $\delta(u_h R_h v, q_1) = \delta(w^*, q_1) = q_{acc}$, we get $w^* \in L$.

Following the above argument we can see that for $1 \leq j \leq h-1$ one has $dist(w_j, w[1, c_{j+1} - 1]) \leq m + jm$. In the $j = h$ case we changed at most $2m$ bits of w. Therefore $dist(w, w^*) \leq (2+h)m \leq 3hm$, as we claimed.

Recalling that $dist(w, L) \geq \epsilon n$, we conclude that $h \geq \epsilon n/(3m)$.

Each run R_j in the sequence $(R_j)_{j=1}^{h-1}$ of feasible runs defined above corresponds to an infeasible run R'_j, obtained from R_j by adding the first bit after R_j. All these infeasible runs are disjoint. Let $a = \log(6m/\epsilon)$. For $1 \leq i \leq a$, denote by s_i the number of runs R'_j whose length falls in the interval $[2^{i-1}+1, 2^i]$. As $|\{1 \leq j \leq h-1 : |R_j| > 6m/\epsilon\}| < \epsilon n/(6m)$, we get $\sum_{i=1}^{a} s_i \geq h - \epsilon n/(6m) \geq \epsilon n/(6m)$. Therefore there exists an index i for which $s_i \geq \epsilon n/(6am)$. Consider all infeasible runs R'_j with $|R'_j| \in [2^{i-1}+1, 2^i]$, but the three leftmost and the three rightmost ones. Note that if a run contains an infeasible sub-run then it is infeasible by itself. Therefore each infeasible interval of length at most 2^i is contained in at least 2^i infeasible runs of length 2^{i+1} containing at least one bit of the interval in their first half. As R'_j are disjoint, each infeasible interval of length 2^{i+1} contains at most two of the R'_js of length at least $2^{i-1}+1$. Thus, we a get a total of at least $(2^i/2)(\epsilon n/(6am) - 6) \geq (2^{i-4}\epsilon n)/(m \log_2(1/\epsilon))$ infeasible runs of length 2^{i+1}. The lemma is proven. □

Now our aim is to reduce the general case to the above described case. For a given DFA M with a graph $G = G(M)$, we denote by $\mathcal{C}(G)$ the *graph of components* of G, whose vertices correspond to maximal by inclusion strongly connected components of G and whose directed edges connect components of G, which are connected by some edge in G. Note that some of the vertices of $\mathcal{C}(G)$ may represent single vertices of G that do not belong to any strongly connected subgraph of G with at least two vertices. From now on we reserve k for the number of vertices of $\mathcal{C}(G)$. We may assume that all vertices of G are reachable from the initial state q_1. Then $\mathcal{C}(G)$ is an acyclic graph in which there exists a directed path from a component C_1, containing q_1, to every other component.

Our next step is to describe how a word $w \in L_M$ of length n can move along the automaton. If a word w belongs to L, it traverses G starting from q_1 and ending in one of the accepting states. Accordingly, w traverses $\mathcal{C}(G)$ starting from C_1 and ending in a component containing an accepting state. For this reason, we call a path A in $\mathcal{C}(G)$ *admissible*, if it starts at C_1 and ends at a component with an accepting state. Given an admissible path $A = (C_{i_1}, \ldots, C_{i_t})$ in $\mathcal{C}(G)$, a sequence $P = (p_j^1, p_j^2)_{j=1}^t$ of pairs of vertices of G (states of M) is called an *admissible sequence of portals* if it satisfies the following restrictions:

1. $p_j^1, p_j^2 \in C_{i_j}$ for every $1 \leq j \leq t$;
2. $p_1^1 = q_1$;
3. $p_t^2 \in F$ (i.e., p_t^2 is an accepting state of M);
4. For every $2 \leq j \leq t$ one has $(p_{j-1}^2, p_j^1) \in E(G)$.

The idea behind the above definition of admissible portals is simple: given an admissible path A, an admissible sequence P of portals defines how a word $w \in L$ moves from one strongly connected component of A to the next one, starting from the initial state q_1 and ending in an accepting state.

Now, given an admissible path A and a corresponding admissible sequence P of portals, we say that an increasing sequence of integers $\Pi = (n_j)_{j=1}^{t+1}$ forms an *admissible partition* with respect to (A, P) if the following holds:

1. $n_1 = 0$;
2. for every $1 \leq j \leq t$, if C_{i_j} is strongly connected, then there exists a path from p_j^1 to p_j^2 in C_{i_j} of length $n_{j+1} - n_j - 1$;
3. $n_{t+1} = n + 1$.

The meaning of the partition $\Pi = (n_j)_{j=1}^{t+1}$ is as follows. If $w \in L$ and w traverses M in accordance with (A, P), then for each $1 \leq j \leq t$, the value of n_j indicates that w arrives to component C_{i_j} for the first time after n_j bits. For convenience we also set $n_{t+1} = n+1$. Thus, for each $1 \leq j \leq t$, the word w stays in C_{i_j} in the interval $[n_j + 1, n_{j+1} - 1]$. Note that it is possible in principle that for a given admissible path A and a corresponding admissible sequence of portals P there is no corresponding admissible partition Π.

A triplet (A, P, Π), where A is an admissible path, P is a corresponding admissible sequence of portals and Π is a corresponding admissible partition, will be called an *admissible triplet*. It is clear from the definition of an admissible triplet that a word $w \in L$ traverses G in accordance with a scenario suggested by one of the admissible triplets. Therefore, in order to get convinced that $w \notin L$, it is enough to check that w does not fit any admissible triplet.

Fix an admissible triplet (A, P, Π), where $A = (C_{i_1}, \ldots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$ and $\Pi = (n_j)_{j=1}^{t+1}$. For all $1 \leq j \leq t$, we define a language L_j in the following way. First, we create an automaton M_j. The set of states of M_j is obtained by adding to C_{i_j} a new state f_j. The initial state of M_j and its unique accepting state are p_j^1 and p_j^2, respectively. For each $q \in C_{i_j}$ and $\alpha \in \{0, 1\}$, if $\delta_M(\alpha, q) \in C_{i_j}$, we set $\delta_{M_j}(\alpha, q) = \delta_M(\alpha, q)$. Otherwise, $\delta_{M_j}(\alpha, q) = f_j$. We also define $\delta_{M_j}(0, f_j) = \delta_{M_j}(1, f_j) = f_j$. Finally, we set L_j to be the language accepted by M_j. Note that M_j satisfies the conditions of Lemma 2.4.

Given a word w of length $|w| = n$, we define t sub-words of it, $w^1, \ldots, w^t$, by setting $w^j = w[n_j+1] \ldots w[n_{j+1}-1]$, where $1 \leq j \leq t$. Note that $|w^j| = n_{j+1} - n_j - 1$.

Lemma 2.5 *Let (A, P, Π) be an admissible triplet , where $A = (C_{i_1}, \ldots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$, $\Pi = (n_j)_{j=1}^{t+1}$. Let w be a word of length n satisfying $dist(w, l) \geq \epsilon n$. Define languages $(L_j)_{j=1}^t$ and words $(w_j)_{j=1}^t$ as described above. Then there exists an index j, $1 \leq j \leq t$, for which $dist(w^j, L_j) \geq \frac{\epsilon n - k}{k}$.*

Proof. Assume this is not the case. For every $1 \leq j \leq t$, if $n_{j+1}-n_j \geq 2$, let $w^{j*} \in L_j$ be a word of length $n_{j+1}-n_j-1$ for which $dist(w^j, w^{j*}) < (\epsilon n - k)/k$. If $n_{j+1}-n_j = 1$, we set $w^{j*} = \emptyset$. Also, for $1 \leq j \leq t-1$ choose $\alpha_j \in \{0,1\}$ so that $\delta_M(\alpha_j, p_j^2) = p_{j+1}^1$. Then by construction the word $w^* = w^{1*}\alpha_1 w^{2*} \dots \alpha_{t-1} w^{t*}$ belongs to L and $dist(w, w^*) \leq t-1+\sum_{j=1}^t dist(w^j, w^{j*}) \leq t-1+t(\epsilon n-k)/k < \epsilon n$ – a contradiction. □

Now we present a key idea of the proof. Ideally, we would like to test whether an input word w of length n fits any admissible triplet. In the positive case, i.e. when $w \in L_M$, the traversal path of w in M defines naturally an admissible triplet which w will obviously fit. In the negative case, i.e. when $dist(w, L) \geq \epsilon n$, Lemma 2.5 implies that for every admissible triplet (A, P, Π), at least one of the sub-words w^j is very far from the corresponding language L_j. Then by Lemma 2.4 w^j contains many short infeasible runs, and thus sampling a small number of random runs will catch one of then with high probability. However, the problem is that the total number of admissible triplets clearly depends on n (in fact, it is polynomial in n), which makes the task of applying directly the union bound on the probability of not catching an infeasible run impossible. We circumvent this difficulty in the following way. We place evenly in $1, \dots n$ a bounded number of *transition intervals* T_s of a bounded length and postulate that a transition between components of $\mathcal{C}(G)$ should happen inside these transition intervals. Then we show that if $w \in L$, it can be modified slightly to meet this restriction, whereas if $dist(w, L) \geq \epsilon n$, for any choice of such an admissible triplet, w is far from fitting it. As the number of admissible triplets under consideration is bounded by a function of ϵ only, we can apply the union bound to estimate the probability of failure.

Denote $m = \max_j(m(C_j))$, $l = lcm(\{g(G[C_j])\})$, where j runs over all strongly connected components of G, corresponding to vertices of $\mathcal{C}(G)$. Let also $S = 129km\log(1/\epsilon)/\epsilon$. We place S transition intervals $\{T_s\}_{s=1}^S$ evenly in $[n]$, where the length of each transition interval T_s is $|T_s| = (k-1)(l+m)$. For $1 \leq i \leq \log(12km/\epsilon)$ define $r_i = \frac{128k^2 m \log^2(\frac{1}{\epsilon})}{2^i \epsilon}$.

ALGORITHM
Input: a word w of length $|w| = n$;

1. For each $1 \leq i \leq \log(12km/\epsilon)$ choose r_i random runs in w of length 2^{i+1} each;
2. For each admissible triplet (A, P, Π) with $A = (C_{i_1}, \dots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$, $\Pi = (n_j)_{j=1}^{t+1}$ such that for all $2 \leq j \leq t-1$ one has $n_j \in T_s$ for some $1 \leq s \leq S$, do the following:
   - form the automata M_j, $1 \leq j \leq t$ as described above;
   - Discard those chosen runs which are at distance at most $\epsilon n/(128km\log(1/\epsilon))$ from some n_j;
   - For each remaining run R, if R falls between n_j and n_{j+1}, check whether it is feasible for the automata M_j starting at $b - n_j + 1$, where b is the first coordinate of R in w;
3. If for some admissible triplet all checked runs turned out to be feasible, output "YES", otherwise (i.e, in the case where for all admissible triplets at least one infeasible interval has been found) output "NO".

Lemma 2.6 *If $dist(w, L) \geq \epsilon n$, then the above algorithm outputs "NO" with probability at least $3/4$. If $w \in L$, then the algorithm always outputs "YES".*

Proof. Let us first assume that $dist(w, L) \geq \epsilon n$. The number of feasible triplets (A, P, Π) for which all partition points fall into the union of transition intervals $\bigcup_{s=1}^S T_s$ can be estimated from above by

$$2^k |V(G(M))|^{2k} \left(S(k-1)(l+m)\right)^{k-1}$$

(first choose an admissible path in $\mathcal{C}(G)$, the number of admissible paths is at most 2^k as any subset of vertices of $\mathcal{C}(G)$ defines at most one path spanning it; then choose portals, the total number of chosen portals is at most $2k$, therefore there are at most $|V(G)|^{2k}$ possible choices for portals; then for a fixed pair (A, P) there are at most $S|T_s|$ choices for each n_j, where $1 \leq j \leq t-1$ and $t \leq k$). It is easy to see that for a small enough $\epsilon > 0$, the above expression is at most $(1/\epsilon)^k$. Thus we need to check at most $(1/\epsilon)^k$ admissible triplets.

Let (A, P, Π) be an admissible triplet satisfying the restriction formulated in Step 2 of the above algorithm. Write $A = (C_{i_1}, \dots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$, $\Pi = (n_j)_{j=1}^{t+1}$. Then the triplet defines automata $(M_j)_{j=1}^t$ and languages $(L_j)_{j=1}^t$ as described before. By Lemma 2.5 for some $1 \leq j \leq t$ one has $dist(w^j, L_j) \geq (\epsilon n - k)/k > \epsilon n/(2k)$. Then by Lemma 2.4 there exists an i, $1 \leq i \leq \log(12km/\epsilon)$ so that w^j contains at least $(2^{i-4}\epsilon n/(2km\log(2k/\epsilon)) \geq 3 \cdot 2^{i-7}\epsilon n/(km\log(1/\epsilon))$ infeasible runs of length 2^{i+1}. At least $(2^{i-7}/(km))\epsilon n/\log(1/\epsilon)$ of them touch neither the first nor the last $\epsilon n/(128km\log(1/\epsilon))$ bits of the interval $[n_j, n_{j+1}-1]$. Obviously, if a random sample contains one of these infeasible runs, then it provides a certificate for the fact that w does not fit this admissible triplet. A random sample of r_i runs of length 2^{i+1} misses all of these infeasible runs with probability at most

$$\left(1 - \frac{2^{i-7}}{kmn}\frac{\epsilon n}{\log(\frac{1}{\epsilon})}\right)^{r_i} < e^{-k\log(\frac{1}{\epsilon})} < \frac{1}{4}e^{-k\ln(1/\epsilon)} = \frac{\epsilon^k}{4}\,.$$

Thus by the union bound we conclude that in this case a random sample does not contain a "witness" for each feasible triplet with probability at most $1/4$.

Now we need to show that if $w \in L$, then there exists an admissible triplet which passes successfully the test of the above algorithm. A traversal of w in M defines a triplet (A, P, Π) as follows: $A = (C_{i_1}, \ldots, C_{i_t})$, where $C_{i_1}, \ldots, C_{i_t}$ are components from $\mathcal{C}(G)$, ordered according to the order of their traversal by w; $P = (p_j^1, p_j^2)_{j=1}^t$, where p_j^1 (resp. p_j^2) is the first (resp. the last) state of C_{i_j} visited by w; $\Pi = (n_j)_{j=1}^{t+1}$, where $n_1 = 0$, $n_{t+1} = n + 1$, and for $2 \leq j \leq t$, n_j is set to be the first time w enters C_{i_j} while traversing M. Note that this partition does not necessary meet the requirement stated in Step 2 of the algorithm. We show that the desired triplet can be obtained from the actual triplet of w by modifying only the third component of it. Define a new partition $\Pi' = (n'_j)_{j=1}^{t+1}$ in the following way. Let $n'_1 = n_1 = 0$. For each $2 \leq j \leq t$ choose a transition interval T_s closest to n_j. If C_{i_j} is a strongly connected component, we choose n'_j as the leftmost coordinate in T_s satisfying the following restrictions: (a) $n'_j \equiv n_j \bmod l$; (b) $n'_j - n'_{j-1} > m$. If C_{i_j} is a singleton without loops we set $n'_j = n'_{j-1} + 1$. As $|T_s| = (k-1)(l+m)$, such an n'_j always exists. Finally, we set $n'_{t+1} = n_{t+1} = n + 1$.

Note that the obtained triplet (A, P, Π') is admissible. Indeed, for every $1 \leq j \leq t$ we have $n'_{j+1} - n'_j \equiv n_{j+1} - n_j \bmod l$, thus implying $n'_{j+1} - n'_j \equiv n_{j+1} - n_j \bmod g(G[C_{i_j}])$, if C_{i_j} is strongly connected. As there exists a path from p_j^1 to p_j^2 in C_{i_j} of length $n_{j+1} - n_j - 1$, there also exists a path of length $n'_{j+1} - n'_j - 1$. This implies the admissibility of Π' and hence the admissibility of (A, P, Π').

Let now R be a run of w inside $[n'_j + \epsilon n/(128km\log(1/\epsilon)), n'_{j+1} - \epsilon n/(128km\log(1/\epsilon))]$ and let b be its first coordinate. Since we placed S transition intervals $\{T_s\}$ evenly in $[n]$, we have $|n'_j - n_j| \leq n/S + |T_s| = \epsilon n/(129km\log(/1/\epsilon)) + (k-1)(l+m)$. Therefore, R falls also completely inside $[n_j+m, n_{j+1}-1]$. (We remark at this point that the purpose of discarding marginal runs at Step 2 of the algorithm is to achieve that each one of the remaining runs will fall completely not only within $[n'_j, n'_{j+1}]$, but also within $[n_j, n_{j+1}]$. As we will see immediately this guarantees that R will be feasible for the corresponding automaton M_j. Without this deletion, with positive probability one of the sampled runs R can hit a true transition point n_j of w, thus making it impossible to attribute R to one particular automaton M_j. Therefore, with positive probability the algorithm would fail in the positive case. Discarding marginal runs allows us to get a one-sided error algorithm).

As $w \in L$, there exists a state $q \in C_{i_j}$ so that $\delta(R, q) \in C_{i_j}$. Also, q is reachable from p_j^1 (the initial state of C_{i_j}) in $b - n_j \geq m$ steps. According to the choice of n'_j we have $n'_j \equiv n_j \bmod g_j$, where g_j is the period of C_{i_j}. But then by Lemma 2.3 q is reachable from p_j^1 in $b - n'_j$ ($\geq m$) steps. This shows that R is feasible for M_j, starting at $b - n_j + 1$. Thus, if $w \in L$, the above algorithm always outputs "YES". □

Finally, the number of bits of w queried by our algorithm is at most

$$\sum_{i=1}^{\log(12km/\epsilon)} 2^{i+1} r_i = \sum_{i=1}^{\log(12km/\epsilon)} 2^{i+1} \frac{128k^2 m \log^2(\frac{1}{\epsilon})}{2^i \epsilon} < \frac{300k^2 m \log^3(\frac{1}{\epsilon})}{\epsilon}.$$

We have thus proven the following theorem.

Theorem 1 *For every regular language L, every integer n and every small enough $\epsilon > 0$, there exists a one-sided error ϵ-testing algorithm for $L \cap \{0,1\}^n$, whose query complexity is $c\log^3(1/\epsilon)/\epsilon$, where the constant $c > 0$ depends only on L.*

3 Lower bound for regular languages

In many testability questions, it is quite natural to expect a lower bound of order $1/\epsilon$ for the query complexity of testing. This is usually proven by taking a positive example of size n and perturbing it in randomly chosen ϵn places to create a negative instance which is hard to distinguish from the positive one. Regular languages are not an exception in this respect, as shown by the next proposition and its fairly simple proof.

Proposition 1 *Let L be the regular language over the alphabet $\{0,1\}$ defined by $L = \{1^n \mid n \geq 1\}$. For any n an ϵ-test for $L \cap \{0,1\}^n$ has query complexity at least $\frac{1}{3\epsilon}$.*

Proof. Our proof is based on following reformulation of the renowned principle of Yao [14], saying that if there exists a probability distribution on the union Ω 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 Ω according to this distribution, then d is a lower bound on the query complexity of any randomized testing algorithm.

Define a distribution on the set of positive and negative instances of length n as follows. The word 1^n gets probability $1/2$. Next we partition the index set $[1, n]$ into $t = 1/\epsilon$ parts $I_1, \ldots, I_t$, each of size ϵn, and for each $1 \leq i \leq t$ give probability $1/(2t)$ to the vector y_i created from 1^n by flipping all bits in I_i from 1 to 0. Note that $dist(y_i, L) = \epsilon n$, hence all y_i are negative instances. Now we apply the above mentioned principle of Yao. Let A be a deterministic ϵ-testing algorithm with query complexity d. If A is incorrect on the word 1^n, then it is already incorrect with probability at least $1/2$. Otherwise, it should accept the input if all

d tested bits equal to 1. Therefore it accepts as well at least $t-d$ of the inputs y_i. This shows that A gives an incorrect answer with probability at least $(t-d)/(2t) < 1/3$, implying $d > t/3$. □.

4 Testability of context-free languages

Having essentially completed the analysis of testability of regular languages, it is quite natural to try to make one step further and to address testability of the much more complex class of context-free languages (see, e.g., [8] for a background information). It turns out that the general situation changes drastically here as compared to the case of regular languages. We show that there exist quite simple context-free languages which are *not* testable. Then we turn our attention to one particular family of context-free languages – the so-called Dyck languages. We prove that the first language in this family, D_1, is testable in time polynomial in $1/\epsilon$, while all other languages in the family are already non-testable. All relevant definitions and proofs follow.

4.1 Some context-free languages are non-testable

As we have already mentioned, not all context-free languages are testable. This is proven in the following proposition.

Proposition 2 *The context-free language $L = \{vv^R uu^R\}$, where w^R denotes the reversal of the word w for each w, is not testable.*

Proof. Let n be divisible by 6. We again define a distribution D on the union of positive and negative examples in the following way. With probability $1/2$ a negative instance is chosen uniformly at random from among all negative instances (i.e. those words $w \in \{0,1\}^n$ which are at distance at least ϵn from L). With probability $1/2$, we pick uniformly at random an integer k in the interval $[n/6, n/3]$ and then select a positive example uniformly among words $vv^R uu^R$ with $|v| = k$. We use the above mentioned Yao's principle again. It is enough to restrict our analysis to non-adaptive algorithms, as the existence of an adaptive testing algorithm with a constant query complexity implies the existence of a non-adaptive one. Let A be an ϵ-testing algorithm which queries a set S of d bits non-adaptively. We argue that for any fixed vector $f : S \to \{0,1\}$, the probability of a positive instance to be consistent with f on S is almost equal to the probability of a negative instance to be consistent with f on S and both are almost equal to $1/2^d$ (here we choose instances according to D). For negative instances, observe that a word of length n generated uniformly at random is within distance ϵn of some word in L with exponentially small probability. Hence, a uniform distribution on the negative instances can be approximated by a uniform distribution on all 2^n words.

For positive instances, notice that for a fixed set S of d bits, for all but at most $2\binom{d}{2}$ choices of k none of the pairs of bits from S is symmetric with respect to position k or position $n/2+k$. Conditioning on a choice of k for which S does not contain a symmetric pair, any fixed vector f has probability exactly $1/2^d$ to appear as a restriction of a positive instance on S. The above implies that for every fixed f, the probability of getting f on S is $(1+o(1))1/2^{d+1}$ for both positive and negative instances. Thus

$$\begin{aligned} Pr[A \text{ fails on } x] &= \sum_f Pr[(A \text{ fails on } x) \wedge (x|_S = f)] \\ &\geq \frac{1}{2} - o(1). \qquad \square \end{aligned}$$

A more careful analysis shows that in fact the language above does not have a testing algorithm asking less than $O(\sqrt{n})$ queries, for some fixed positive ϵ. We omit the details.

4.2 Testability of the Dyck languages

It would be extremely nice to determine exactly which context-free languages are testable. At present we seem to be very far from fulfilling this task. However, we are able to solve this question completely for one family of context-free languages – the so called Dyck languages.

For an integer $n \geq 1$, the *Dyck language of order n*, denoted by D_n, is the language over the alphabet of $2n$ symbols $\{a_1, b_1, \ldots, a_n, b_n\}$, grouped into n ordered pairs $(a_1, b_1), \ldots, (a_n, b_n)$. The language D_n is defined by the following productions:

1. $S \to a_i S b_i$ for $i = 1, \ldots, n$;
2. $S \to SS$;
3. $S \to \gamma$,

where γ denotes the empty word. Though the words of D_n are not binary according to the above definition, we can easily encode them and the grammar describing them using only 0's and 1's. Thus we may still assume that we are in the framework of languages over the binary alphabet. We can interpret D_n as the language with n distinct pairs of brackets, where a word w belongs to D_n iff it forms a balanced bracket expression. The most basic and well known language in this family is D_1, where we have only one pair of brackets. Dyck languages play an important role in the theory of context-free languages (see, e.g., [4] for a relevant discussion) and therefore the task of exploring their testability is interesting.

Our first goal in this subsection is to show that the language D_1 is testable. Let us introduce a suitable notation.

First, for the sake of simplicity we denote the brackets a_1, b_1 of D_1 by $0, 1$, respectively. Assume that n is a large enough even number (obviously, for odd n we have $D_1 \cap \{0,1\}^n = \emptyset$, thus there is nothing to test in this case). Let w be a binary word of length n. For $1 \leq i \leq n$, we denote by $x(w,i)$ the number of 0's in the first i positions of w. Also, $y(w,i)$ stands for the number of $1's$ in the first i positions of w. We have the following claims.

Claim 4.1 *The word w belongs to D_1 if and only if the following two conditions hold: (a) $x(w,i) \geq y(w,i)$ for every $1 \leq i \leq n$; (b) $x(w,n) = y(w,n)$.*

Proof. Follows easily from the definition of D_1, for example, by induction on the length of w. We omit a detailed proof. □

Claim 4.2 *If w satisfies (a) $y(w,i) - x(w,i) \leq s_1$ for every $1 \leq i \leq n$; (b) $x(w,n) - y(w,n) \leq s_2$, then $dist(w, D_1) \leq s_1 + s_2/2 + 1$.*

Proof. Observe first that by Claim 4.1 a word w is in D_1 if and only if we can partition its letters into pairwise disjoint pairs, so that the left letter in each pair is a zero, and the right letter is a one. Consider the bipartite graph whose classes of vertices are the set of all 1's and the set of all 0's in w, where each 1 is connected to all zeros to its left. By assumption (a) and the defect form of Hall's theorem, this graph contains a matching of size at least $y(w,n) - s_1$. By assumption (b), $y(w,n) \geq n/2 - s_2/2$. Therefore, there are at least $n/2 - s_2/2 - s_1$ disjoint pairs of letters in w, where in each pair there is a zero on the left and a one on the right. Let us pair the remaining elements of w arbitrarily, where all pairs but at most one consist of either two 0's or two 1's. By changing, now, when needed, the left entry of each such pair to 0 and its right entry to 1 we obtain a word in D_1, and the total number of changes performed is at most $(s_2 + 2s_1 - 2)/2 + 2 = s_1 + s_2/2 + 1$, completing the proof. □

Claim 4.3 *a) If for some $1 \leq i \leq n$ one has $y(w,i) - x(w,i) \geq s$, then $dist(w, D_1) \geq s/2$; b) If $x(w,n) - y(w,n) \geq s$, then $dist(w, D_1) \geq s/2$.*

Proof. Follows immediately from Claim 4.1. □

We conclude from the above three claims that a word w is far from D_1 if and only if for some coordinate i it deviates significantly from the necessary and sufficient conditions provided by Claim 4.2. This observation is used in the analysis of an algorithm for testing D_1, proposed below.

Set

$$d = \frac{C \log\left(\frac{1}{\epsilon}\right)}{\epsilon^2};$$

$$\Delta = \frac{C \log\left(\frac{1}{\epsilon}\right)}{8\epsilon},$$

where $C > 0$ is a sufficiently large constant, whose value will be chosen later, and assume d is an even integer. In what follows we omit all floor and ceiling signs, to simplify the presentation.

ALGORITHM
Input: a word w of length $|w| = n$;

1. Choose d bits of $1, \ldots, n$ uniformly at random. Let S be the corresponding sample of w of length d;
2. If $dist(S, D_1 \cap \{0,1\}^d) < \Delta$, output "YES", otherwise output "NO".

Lemma 4.4 *The above algorithm outputs a correct answer with probability at least $2/3$.*

Proof. It will be somewhat easier to analyze a variation of the above algorithm, in which instead of choosing d bits of w uniformly at random we choose each bit of w to be represented in the sample S independently and with probability $p = d/n$. This results in a certain loss of rigorousity in the analysis. One can show however that the above two models for generating S are equivalent for our purpose here. We thus stick to the more convenient model of choosing each bit with probability p. As we have already mentioned, we set

$$p = \frac{d}{n} = \frac{C \log\left(\frac{1}{\epsilon}\right)}{\epsilon^2 n}.$$

Consider first the positive case $w \in D_1$. Set $t = C/\epsilon$ and assume for simplicity that t as well as n/t are integers. For $1 \leq j \leq t$, let X_j be the number of 0's in S, sampled from the interval $[1, nj/t]$. Let also Y_j denote the number of 1's in S, sampled from the same interval. Both X_j and Y_j are binomial random variables with parameters $x(w, nj/t)$ and p, and $y(w, nj/t)$ and p, respectively. As $w \in D_1$, we get by Claim 4.1 that $x(w, nj/t) \geq y(w, nj/t)$, implying $EX_j \geq EY_j$. Applying standard bounds on the tails of binomial distribution, we obtain:

$$Pr[Y_j \geq X_j + \frac{\Delta}{2}] \leq Pr[X_j \leq EX_j - \frac{\Delta}{4}] +$$

$$Pr[Y_j \geq EY_j + \frac{\Delta}{4}] \leq 2^{-\Omega(\Delta^2/np)} = 2^{-\Omega(C \log(1/\epsilon))}. \quad (1)$$

For $1 \leq j \leq t-1$, set $Z_j = Y_{j+1} - Y_j$. Note that $EZ_j \leq np/t$. Using similar argumentation as above, we get

$$Pr[Z_j \geq \frac{2np}{t}] \leq 2^{-\Omega(np/t)} = 2^{-\Omega(\log(1/\epsilon)/\epsilon)}. \quad (2)$$

As $w \in D_1$, we have by Claim 4.1 $x(w,n) = y(w,n) = n/2$. Hence

$$Pr[X_t \geq \frac{np}{2} + \frac{\Delta}{8}] \leq 2^{-\Omega(\Delta^2/np)} = 2^{-\Omega(C \log(1/\epsilon))}. \quad (3)$$

Finally, we have the following estimate on the distribution of the sample size $|S|$:

$$Pr[||S| - np| \geq \frac{\Delta}{4}] \leq 2^{-\Omega(\Delta^2/np)} = 2^{-\Omega(C\log(1/\epsilon))} . \tag{4}$$

Choosing C large enough and recalling the definition of t, we derive from (1)–(4) that with probability at least $2/3$ the following events hold simultaneously:

1. $\max_{1\leq j\leq t}(Y_j - X_j) \leq \frac{\Delta}{2}$;
2. $\max_{1\leq j\leq t} Z_j \leq \frac{2np}{t}$;
3. $X_t \leq \frac{np}{2} + \frac{\Delta}{8}$;
4. $|S| \geq np - \frac{\Delta}{4}$.

Assume that the above four conditions happen. Then we claim that $dist(S, D_1) < \Delta$. Indeed, the first two conditions guarantee that for all $1 \leq i \leq |S|$ we have $y(S,i) - x(S,i) \leq \Delta/2 + 2np/t \leq 2\Delta/3$. The last two conditions provide $x(S,|S|) - y(S,|S|) = X_t - Y_t = 2X_t - |S| \leq \Delta/2$. Therefore, by Claim 4.2 $dist(S, D_1) < \Delta$. Thus, if $w \in D_1$, our algorithm will accept w with probability at least $2/3$, as required.

Let us now consider the negative case. Assume that $dist(w, D_1 \cap \{0,1\}^n) \geq \epsilon n$. By Claim 4.2 we have then that at least one of the following two conditions holds: a) there exists an index $1 \leq i \leq n$, for which $y(w,i) - x(w,i) \geq \epsilon n/2$; b) $x(w,n) - y(w,n) \geq \epsilon n/2$. In the former case, let X, Y be the number of 0's, 1's, respectively, of S, sampled from the interval $[1,i]$. Let also k be the number of elements from $[1,i]$ chosen to S. Then $X = x(S,k)$, $Y = y(S,k)$. Both X and Y are binomially distributed with parameters $x(w,i)$ and p, and $y(w,i)$ and p, respectively. It follows from the definition of i that $EY - EX \geq \epsilon np/2$. But then we have

$$Pr[y(S,k) - x(S,k) \leq 2\Delta] = Pr[Y - X \leq 2\Delta] \leq Pr[X \geq EX + (\frac{\epsilon np}{4} - \Delta)] + Pr[Y \leq EY - (\frac{\epsilon np}{4} - \Delta)] = 2^{-\Omega((\epsilon np/4 - \Delta)^2/(np))} .$$

Choosing the constant C to be sufficiently large and recalling the definitions of p and Δ, we see that the above probability is at most $1/3$. But if $y(S,k) - x(S,k) \geq 2\Delta$, it follows from Claim 4.3 that $dist(S, D_1) \geq \Delta$.

If $x(w,n) - y(w,n) \geq \epsilon n/2$, we obtain, using similar arguments:

$$Pr[x(S,|S|) - y(S,|S|) \leq 2\Delta] = 2^{-\Omega((\epsilon np/4 - \Delta)^2/(np))} .$$

The above probability can be made at most $1/3$ by the choice of C. But if $x(S,|S|) - y(S,|S|) \geq 2\Delta$, it follows from Claim 4.3 that $dist(S, D_1) \geq \Delta$. Thus in both cases we obtain that our algorithm rejects w with probability at least $2/3$. This completes the proof of Lemma 4.4. □.

By Lemma 4.4 we have the following result about the testability of the Dyck language D_1.

Theorem 2 *For every integer n and every small enough $\epsilon > 0$, there exists an ϵ-testing algorithm for $D_1 \cap \{0,1\}^n$, whose query complexity is $C\log(1/\epsilon)/\epsilon^2$ for some absolute constant $C > 0$.*

The reader has possibly noticed one significant difference between the algorithm of Section 2 for testing regular languages and our algorithm for testing D_1. While the algorithm for testing regular languages has a one-sided error, the algorithm of this section has a two-sided error. This is not a coincidence. We can show that there is *no* one-sided error algorithm for testing membership in D_1, whose number of queries is bounded by a function of ϵ only. Indeed, assume that A is a one-sided error algorithm for testing D_1. Consider its execution on the input word $u = 0^{n/2+\epsilon n}1^{n/2-\epsilon n}$. It is easy to see that $dist(u, D_1) \geq \epsilon n$. Therefore, A must reject u with probability at least $2/3$. Fix any sequence of coin tosses which makes A reject u and denote by Q the corresponding set of queried bits of u. We claim that if $|Q \cap [1, n/2 + \epsilon n]| \leq n/2 - \epsilon n$, then there exists a word w of length n from D_1, for which $w[i] = u[i]$ for all $i \in Q$. To prove this claim, we may clearly assume that $|Q \cap [1, n/2 + \epsilon n]| = n/2 - \epsilon n$. Define w as follows. For all $i > n/2 + \epsilon n$ we set $w[i] = 1$. Now, we take the first $\epsilon n/2$ indices i in $[1, n/2 + \epsilon n] \setminus Q$ and set $w[i] = 0$. For the last $\epsilon n/2$ indices i in $[1, n/2 + \epsilon n] \setminus Q$ we set $w[i] = 1$. Also, $w[i] = u[i]$ for all $i \in Q$. It is obvious that w satisfies the sufficient condition for the membership in D_1, given by Claim 4.1, and therefore $w \in D_1$. As A is assumed to be a one-sided error algorithm, it should always accept every $w \in D_1$. But then we must have $|Q \cap [1, n/2 + \epsilon n]| > n/2 - \epsilon n$, implying that A queries a linear in n number of bits. We have proven the following statement.

Proposition 3 *Membership in D_1 is not testable by a one-sided error algorithm.*

Our next goal is to prove that all other Dyck languages, namely D_k for all $k \geq 2$ are non-testable. We will present a detailed proof of this statement only for $k = 2$, but this clearly implies the result for all $k \geq 3$.

For the sake of clarity of exposition we replace the symbols a_1, b_1, a_2, b_2 in the definition of D_2 by $0, 1, 2, 3$, respectively. Then D_2 is defined by the following context-free grammar: $S \to 0S1 \mid 2S3 \mid SS \mid \gamma$, where γ is the empty word. Having in mind the above mentioned bracket interpretation of the Dyck languages, we will sometimes refer to $0, 2$ as left brackets and to $1, 3$ as right brackets. Note that we do not use an encoding of D_2 as a language over $\{0,1\}$,

but rather over an alphabet of size 4. Clearly, non-testability of D_2 as defined above will imply non-testability of any binary encoding of D_2.

Theorem 3 *The language D_2 is not testable.*

Proof. Let n be a large enough integer, divisible by 8. We denote $L_n = D_2 \cap \{0,1,2,3\}^n$. Using Yao's principle, we assign a probability distribution on inputs of length n and show that any deterministic algorithm probing $d = O(1)$ bits outputs an incorrect answer with probability $0.5 \pm o(1)$.

Positive instances are generated according to the distribution P as follows: choose k uniformly at random in the range $n/8, \ldots, n/4$. Given k the word of length n is $w = 0^k 1^k v$ where v is of length $n-2k$ generated by: for $i = 1, ...(n-2k)/2$ choose $v[i]$ at random from $0, 2$ and $v[n-2k+1-i] = v[i]+1$.

Negative instances are chosen as follows: the process is very similar to the positive case except that we do not have the restriction on $v[n-2k+1-i] = v[i]+1$. Namely, we choose k at random in the range $n/8, \ldots, n/4$. Given k, a word of length n is $w = 0^k 1^k v$, where v is of length $n-2k$ generated by: for $i = 1, ...(n-2k)/2$ choose $v[i]$ at random from $0, 2$ and for $i = 1, ...(n-2k)/2$ choose $v[n-2k+1-i]$ at random from $1, 3$. Let us denote by N the distribution at this stage. Note that the words that are generated may be of distance less than ϵn from L_n (in fact some words in L_n are generated too). Hence we further condition N on the event that the word is of distance at least ϵn from L_n.

The probability distribution over all inputs of length n is is now defined by choosing with probability 1/2 positive instances, generated as above, and with probability 1/2 negative instances, chosen according to the above described process.

Claim 4.5 *With exponentially small in n probability an instance generated according to N is ϵn-close to some word in the language L_n.*

Proof. Fix k and let $w = 0^k 1^k v$ be a word of length n generated by N. If $u \in L_n$ is of distance $s \leq \epsilon n$ from w then there is a set $I \subseteq [1,n],\ |I| = s$ of places where u differs from w. However, in w, $v[1], ..., v[\frac{n-2k}{2}] \in \{0,2\}$, namely, are left brackets. In u this would lead to at least $\frac{n-2k}{2} - s$ unchanged places which are left brackets and hence their matching right brackets must come to the right of them. Fixing the way we change the symbols in I, some of the matching right brackets may come from within the set I. However this still leaves at least $\frac{n-2k}{2} - 2s$ original left brackets that must be matched with at least $\frac{n-2k}{2} - 2s$ right brackets that come from places in the range $n - \frac{n-2k}{2} + 1, ..., n$. By the same argument, at least $\frac{n-2k}{2} - 3s$ of these places are unchanged right brackets. Fixing I and the way we change the symbols in I, this fixes at least $\frac{n-2k}{2} - 3s$ places in $v[1], ..., v[\frac{n-2k}{2}]$ and at least $\frac{n-2k}{2} - 3s$ places in the range $v[\frac{n-2k}{2}+1], ..., v[n-2k]$ that need to match. However, by the way we generated w this matching occurs with probability of at most $2^{-\frac{n-2k}{2}+3s}$. Altogether there are $\sum_{s \leq \epsilon n} \binom{n}{s}$ ways to fix I and for each I there are 3^s ways to change the symbols in I. Hence this yields a bound of $\sum_{s \leq \epsilon n} \binom{n}{s} 3^s 2^{-\frac{n-2k}{2}+3s}$ on the probability that a word w, generated according to N, is of a distance at most ϵn from L_n. For ϵ small enough this is clearly exponentially small. □

Claim 4.6 *Let $S \subseteq [n],\ |S| = d = O(1)$ be a fixed set of places and let w be generated according to either N or P. Then S contains a pair $i < j$ symmetric with respect to the midpoint of the last $n-2k$ letters of the word with probability*

Proof. For each distinct pair $i, j \in D$ there is a unique k for which i, j are symmetric with respect to the above defined midpoint. Hence the above probability is bounded by $\binom{d}{2}\frac{8}{n}$. □

We now return to the proof of Theorem 3. Let A be an algorithm for testing L_n that queries at most $d = O(1)$ queries. As $d = O(1)$ we may assume that A is non-adaptive, namely, it queries some fixed set of places S of size d. For any possible set of answers $f : S \longrightarrow \{0,1,2,3\}$ and an input w let f_w denote the event that w is consistent with f on S. Let $NoSym$ be the event that S contains no symmetric pair with respect to $(n-2k)/2$. Also, let F_0 denote all these f's on which the algorithm answers 'negative' and let F_1 be all these f's on which it answers 'positive'.

The total error of the algorithm is

$$\sum_{f \in F_0} Pr[f_w \wedge (w \text{ posit.})] + \sum_{f \in F_1} Pr[f_w \wedge (w \text{ negat.})] \geq$$
$$Pr[NoSym](\sum_{f \in F_0} Pr[f_w \wedge (w \text{ positive})|NoSym] +$$
$$\sum_{f \in F_1} Pr[f_w \wedge (w \text{ negative})|NoSym])$$

However, given that S contains no symmetric pairs, $Pr[f_w \wedge (w\ is\ negative)]$ is essentially equal to $Pr[f_w \wedge (w\ is\ positive)]$ (these probabilities would be exactly equal if negative w would be generated according to N. Claim 4.5 asserts that N is exponentially close to the real distribution on negative instances). Hence each is of these probabilities is $0.5Pr[f_w|NoSym] \pm o(1)$.

Plugging this into the sum above, and using Claim 4.6 we get that the error probability is bounded from below by $Pr[NoSym]\sum_f (0.5 \pm o(1))Pr[f_w|NoSym] \geq (1 - o(1))(0.5 \pm o(1)) \geq 0.5 - o(1)$. □

5 Concluding remarks

The main technical achievement of this paper is a proof of testability of regular languages. A possible continuation of the research is to describe other classes of testable languages and to formulate sufficient conditions for a context-free language to be testable (recall that in Proposition 2 we have shown that not all context-free languages are testable).

One of the most natural ways to describe large classes of testable combinatorial properties is by putting some restrictions on the logical formulas that define them. In particular we can restrict the arity of the participating relations, the number of quantifier alternations, the order of the logical expression (first order, second order), etc.

The result of the present paper is an example to this approach, since regular languages are exactly those that can be expressed in second order monadic logic with a unary predicate and an embedded linear order. Another example can be found in a sequel of this paper [1], which addresses testability of graph properties defined by sentences in first order logic with binary predicates, and which complements the class of graph properties shown to be testable by Goldreich et al [7]. Analogous results for predicates of higher arities would be desirable to obtain, but technical difficulties arise when the arity is greater than two.

As a long term goal we propose a systematic study of the testability of logically defined classes. Since many different types of logical frameworks are known, to find out which one is suited for this study is a challenge. Virtually all single problems that have been looked at so far have the perspective of being captured by a more general logically defined class with members that have the same testability properties.

A very different avenue is to try to develop general *combinatorial* techniques for proving lower bounds for the query complexity of testing arbitrary properties, possibly by finding analogs to the block sensitivity [12] and the Fourier analysis [11] approaches for decision tree complexity. At present we have no candidates for combinatorial conditions that would be both necessary and sufficient for ϵ-testability.

Acknowledgment We would like to thank Oded Goldreich for helpful comments.

References

[1] N. Alon, E. Fischer, M. Krivelevich, M. Szegedy, Efficient testing of large graphs, Proceedings of the 40^{th} IEEE FOCS (1999), to appear.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, Proof verification and intractability of approximation problems, Proceedings of the 33^{th} IEEE FOCS (1992), 14–23.

[3] M. Blum, M. Luby, R. Rubinfeld, Self-testing/correcting with applications to numerical problems. J. Computer System Sciences 47 (1994), 549–595.

[4] M. D. Davis, R. Sigal and E. Weyuker, Computability, Complexity and Languages: Fundamentals of Theoretical Computer Science, Academic Press, 1994.

[5] J. Dixmier, Proof of a conjecture by Erdős and Graham concerning the problem of Frobenius, J. Number Theory 34 (1990), no. 2, 198–209.

[6] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, A. Wigderson, Self-testing/correcting for polynomials and for approximate functions, Proceedings of the 23^{th} ACM STOC (1991), 32–42.

[7] O. Goldreich, S. Goldwasser and D. Ron, Property testing and its connections to learning and approximation. To appear in J. of the ACM, also: Proceedings of the 37^{th} IEEE FOCS (1996), 339–348.

[8] J.E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

[9] M. Lewin, A bound for a solution of a linear Diophantine problem, J. London Math. Soc. (2) 6 (1972), 61–69.

[10] R. Lipton, New directions in testing, Unpublished manuscript, 1989.

[11] N. Nisan, M. Szegedy, On the degree of Boolean functions as real polynomials, Proceedings of the 24^{th} ACM STOC (1992), 462–467.

[12] N. Nisan, CREW PRAMs and decision trees, Proceedings of the 21^{st} ACM STOC (1989), 327–335.

[13] R. Rubinfeld, M. Sudan, Robust characterization of polynomials with applications to program testing. SIAM J. of Computing, 25(2) (1996), 252-271.

[14] A. C. Yao, Probabilistic computation, towards a unified measure of complexity. Proceedings of the 18^{th} IEEE FOCS (1977), 222–227.