Regular Languages are Testable with a Constant Number of Queries [*]

Noga Alon [†] Michael Krivelevich [‡] Ilan Newman [§] Mario Szegedy [¶]

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

[*] A preliminary version of this paper appeared in the Proceedings of the 40^{th} Symposium on Foundation of Computer Science (FOCS'99), IEEE Press 1999, 645–655.

[†] Department of Mathematics, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, Israel, and AT&T Labs–Research, Florham Park, NJ 07932, USA. Email: noga@math.tau.ac.il. 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.

[‡] Department of Mathematics, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: krivelev@math.tau.ac.il Part of this research was performed when this author was with DIMACS Center, Rutgers University, Piscataway NJ, 08854, USA and AT&T Labs–Research, Florham Park, NJ 07932, USA. Research supported in part by a DIMACS Postdoctoral Fellowship.

[§] Department of Computer Science, University of Haifa, Haifa, Israel. E-mail: ilan@cs.haifa.ac.il. Part of this research was performed when this author was visiting AT& T Labs – Research, Florham Park, NJ 07932, USA.

[¶] School of Mathematics, Institute for Advanced Study, Olden Lane, Princeton, NJ 08540, USA. E-mail: szegedy@math.ias.edu. Part of this research was performed when this author was with AT&T Labs–Research, Florham Park, NJ 07932, USA.

the case of w being ϵ-far from satisfying P. Finally, we say that property P is (c, ϵ)-*testable* if for every $\epsilon > 0$ there exists an ϵ-test for P whose total number of queries is bounded by c.

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 to when there is, for every $\epsilon > 0$, an ϵ-test for a property using $c = c(\epsilon)$ queries (i.e c is a function of ϵ but independent of n) 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 $f_n : \{0,1\}^n \rightarrow \{0,1\}$, with $f_n^{-1}(1) = L \cap \{0,1\}^n = L_n$. Our main result states that all regular languages are ϵ-testable with query complexity only $\tilde{O}(1/\epsilon)$. We also show that this complexity is optimal up to a factor polylogarithmic in $1/\epsilon$. This positive result cannot be extended to context-free languages, for there is an example of 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 the 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. We also discuss testability of the Dyck 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, namely that regular languages are $(\tilde{O}(\frac{1}{\epsilon}), \epsilon)$-testable. As this result is asymptotic, we assume that n is big enough with respect to $\frac{1}{\epsilon}$ (and with respect to any other constant that depends only on the fixed language we are working with). All logarithms are binary unless stated explicitly otherwise.

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

Definition 2.1 *A deterministic finite automaton (DFA) M over $\{0,1\}$ with states $Q = \{q_1, \dots, q_m\}$ is given by a function $\delta : Q \times \{0,1\} \to Q$ 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. Let γ denote the empty word. Then

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

Thus, if M starts in a state q and processes string u, then it ends up in a state $\delta(q,u)$.

We then say that M accepts a word u if $\delta(q_1,u) \in F$. M rejects u means that $\delta(q_1,u) \in Q \setminus F$. 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 $(q_{i_0}, \dots, q_{i_n})$ in the following natural way: $q_{i_0} = q_1$, and for $1 \le j \le n$, $q_{i_j} = \delta(q_1, w[1] \dots w[j])$. 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 $E(G(M)) = \{(q_i,q_j) \mid \delta(q_i,0) = q_j\} \cup \{(q_i,q_j) \mid \delta(q_i,1) = q_j\}$. 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 nonempty, strongly connected directed graph with a finite period $g(G)$. Then there exist a partition $V(G) = V_0 \cup \dots V_{g-1}$ and a constant $m = m(G)$ which does not exceed $3|V|^2$ such that:*

1. *For every $0 \le i,j \le g-1$ 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 $0 \le i,j \le g-1$ and for every $u \in V_i$, $v \in V_j$ and for every integer $r \ge m$, if $r = (j-i) \pmod g$, then there exists a directed path from u to v in G of length r.*

Proof. To prove part 1, fix an arbitrary vertex $z \in V$ and for each $0 \le i \le g-1$, let V_i be the set of all those vertices 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. Consider any set of positive integers $\{a_i\}$ whose greatest common divisor is g. It is well known that 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 integer 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 $v_0, v_1, \ldots, v_{k-1}$ and by concatenating directed paths from v_i to v_{i+1} for each $0 \leq i \leq k-1$, where the indices are taken modulo k). Associate now the set of cycle lengths in this walk with the set of positive integers $\{a_i\}$ as above. Then, following this closed walk and 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 $r > 3|V|^2$ satisfying $r = (j-i) \bmod g$, 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 $r-l$ 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.

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 non-trivial case $L_M \cap \{0,1\}^n \neq \emptyset$.

We now introduce a key definition for the sequel:

Definition 2.4 *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 of length $n - (|w'| + i - 1)$ in G from the state $\delta(q, w')$ to at least one of the accepting states. Otherwise, w' is called* infeasible*.*

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:

Definition 2.5 *We call an automaton M* 'essentially strongly connected' *if*

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.)

Lemma 2.6 *Assume that the language $L = L_M$ contains some words of length n, and that M is essentially strongly connected with C and D being the partition of the states of M as in Definition 2.5. Let m be the reachability constant of $G[C]$. Assume also that $\epsilon n \geq 64m\log(4m/\epsilon)$. Then if for a word w of length $|w| = n$, $dist(w, L) \geq \epsilon n$, then there exists an integer $1 \leq i \leq \log(4m/\epsilon)$ such that the number of infeasible runs of w of length 2^{i+1} is at least $\frac{2^{i-4}\epsilon n}{m\log(4m/\epsilon)}$.*

Proof.

Our intention is to construct a sequence $(R_j)_{j=1,\ldots}$ of disjoint infeasible runs, each being minimal in the sense that each of its prefixes is feasible, and so that each is a subword of the given word w. We then show that we can concatenate these subwords to form a word in the language that is not too far from w ('not too far' will essentially depend on the number of runs that we have constructed). This in turn will show that if $dist(w, L) \geq \epsilon n$ then there is a lower bound on the number of these infeasible runs.

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 $m + 1$: Let R_1 be the shortest infeasible run starting from $w[m + 1]$ and ending before $w[n - m + 1]$. If there is no such run we stop. Assume that we have constructed so far $R_1, \ldots, R_{j-1}$ ending at $w[c_{j-1}]$, next we construct R_j by taking the minimal infeasible run starting at $w[c_{j-1} + 1]$ and ending before $w[n - m + 1]$. Again if there is no such run we stop.

Assume we have constructed in this way runs $R_1, \ldots, R_h$. Note that each run is a subword of w, the runs are pairwise disjoint and their concatenation in order forms a (continuous) subword of w. Also, note that by the definition of each run R_j being minimal infeasible, its prefix $R_j^{(-)}$, obtained by discarding the last bit of R_j is feasible. This, in turn, implies that R'_j which is obtained from R_j by flipping its last bit is feasible. In addition, by Definition 2.4, this means that for each R'_j there is a state $q_{i_j} \in C$ so that $\delta(q_{i_j}, R'_j) \in C$ and such that q_{i_j} is reachable from q_1 in $c_{j-1} + 1$ steps.

Next we inductively construct a word $w^* \in L$ such that $dist(w, w^*) \leq hm + 2m + 2$. Assuming that $dist(w, L) \geq \epsilon n$ this will imply a lower bound on h. The general idea is to 'glue' together the R'_j for $j = 1, \ldots, h$, each being feasible and yet very close to a subword of w (except for the last bit in each). The only concern is to glue the pieces together so that as a whole word it will be feasible. This will require an extra change of m bits per run, plus some additional $2m$ bits at the end of the word.

We maintain during the induction that for $j = 0, \ldots$ the word w_j we construct is feasible starting from position 1, and it ends in position c_j. For the base case, let $c_0 = m$ and let w_0 to be any 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 $\delta(q_1, w_{j-1}) = p_j$. 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 $\delta(q_1, u_j) = q_{i_j}$. We now define w_j as a concatenation of u_j and R'_j. Let w_h be the final word that is defined in this way, ending at place c_h. Now the reason we have stopped with R_h is either that there is no infeasible run starting at $c_h + 1$, in which case, changing the last m bits of w_h and concatenating to it the remaining suffix of w (that starts at position $c_h + 1$), exactly as in the case of adding R'_j, yields the required w^*. The other possible reason for stopping growing R_h is when there is a minimal infeasible run that start at $c_h + 1$ and ends after position $n - m + 1$. Let R be that run, and let R' be the run obtained by flipping the last bit of R. As was the case with any R'_j, R' is feasible from position $c_h + 1$. Hence there is a feasible word u of which R' is a prefix, u is of length $n - c_h$ and so that $\delta(q_{i_h}, u) = q_{acc}$. We can construct w^* from w_h and u exactly as we have constructed w^* form w_h and the suffix of w in the previous case.

By the definition of w^*, $w^* \in L$. Following the inductive construction of w^* it follows that for $1 \leq j \leq h,\ dist(w_j, w[1, c_j]) \leq jm + 1$. Then to get from w_h to w^* we concatenate R' which is either a subword of w (in the first case previously discussed) or it is a subword of w where one bit was changed (in the second case), following by changing m bits at the end of w_h and possibly additional m bits at the end of u. Therefore $dist(w, w^*) \leq hm + 2m + 2$, as we claimed.

Recalling that $dist(w, L) \geq \epsilon n$, we conclude that $h \geq \frac{\epsilon n - 2}{m} - 2 \geq \epsilon n/(2m)$ (the last inequality is by our assumptions that $\epsilon n \geq 64m \log(4m/\epsilon)$). This already shows that if $dist(w, L) \geq \epsilon n$ then there are $\Omega(\epsilon n)$ many disjoint infeasible runs in w. However, we need a stronger dependence as stated in the lemma. We achieve this in the following way.

Let $a = \log(4m/\epsilon)$. For $1 \leq i \leq a$, denote by s_i the number of runs in $\{R_j\}_{j=1}^h$ whose length falls in the interval $[2^{i-1} + 1, 2^i]$. As $|\{R_j :\ 1 \leq j \leq h, |R_j| > 4m/\epsilon\}| < \epsilon n/(4m)$, we get $\sum_{i=1}^a s_i \geq h - \epsilon n/(4m) \geq \epsilon n/(4m)$. Therefore there exists an index i for which $s_i \geq \epsilon n/(4am)$. Consider all infeasible runs R_j with $|R_j| \in [2^{i-1} + 1, 2^i]$. Note that if a run contains an infeasible sub-run then it is infeasible by itself. Now, each infeasible run of length between $2^{i-1} + 1$ and 2^i is contained in at least 2^i infeasible runs of length 2^{i+1}, except maybe, for the first two and the last two runs (these with the two smallest j's and these with the two largest j's). As R_j are disjoint, each infeasible run of length 2^{i+1} contains at most three of the R_js of length at least $2^{i-1} + 1$. Thus, we a get a total of at least $(2^i/3)(s_i - 4)$ infeasible runs of length at most 2^{i+1}. By our assumption on the parameters this number is: $(2^i/3)(s_i - 4) \geq \frac{2^i}{3}(\frac{\epsilon n}{4am} - 4) = 2^{i-4}(\frac{\epsilon n}{am} + \frac{1}{3}\frac{\epsilon n - 64am}{am}) \geq \frac{2^{i-4}\epsilon n}{m \log(4m/\epsilon)}$, as claimed. □

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 with no self loops, that do not belong to any strongly connected subgraph of G with at least two vertices. All other components have non empty paths inside them and will be called *truly connected*. From now on we reserve k for the number of vertices of $\mathcal{C}(G)$ and set $V = V(G)$. 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. Denote $m = \max_j(m(C_j))$, $l = lcm(\{g(G[C_j])\})$, where j runs over all truly connected components of G, corresponding to vertices of $\mathcal{C}(G)$. We will assume in the sequel that the following relation are satisfied between the parameters:

Condition (*)

- $\frac{\epsilon n}{2k} \geq 64m \log \frac{8mk}{\epsilon}$.
- $\epsilon n > 8km$

- $\epsilon \log(1/\epsilon) < \frac{1}{258k^2|V|^2 m(l+m)}$

clearly, for any fixed k, m, l for ϵ small enough and n large enough condition (*) holds.

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. The pair (p_j^1, p_j^2) are the first and last states that are traversed in C_{i_j}.

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 $1 \leq j \leq t$, 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 (this could happen if the path A and the set of portals P correspond to no word of length n).

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 $A = (C_{i_1}, \dots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$ and $\Pi = (n_j)_{j=1}^{t+1}$. For all $1 \le j \le t$, we define a language L_j that contains all words that traverse in M from p_j^1 to p_j^2. This is done formally by defining an automaton M_j as follows: 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 $\delta_M(q, \alpha) \in C_{i_j}$, we set $\delta_{M_j}(q, \alpha) = \delta_M(q, \alpha)$. Otherwise, $\delta_{M_j}(q, \alpha) = f_j$. We also define $\delta_{M_j}(f_j, 0) = \delta_{M_j}(f_j, 1) = f_j$. Namely, in M_j all transitions within C_{i_j} remain the same. All transitions going to other components now go to f_j which has a loop to itself. Thus, M_j is essentially strongly connected as in Definition 2.5 with $D = \{f_j\}$. Then L_j is the language accepted by M_j.

Given the fixed admissible triplet (A, P, Π) and a word w of length $|w| = n$, we define t sub-words of it, $w^1, \dots, w^t$, by setting $w^j = w[n_j + 1] \dots w[n_{j+1} - 1]$, where $1 \le j \le t$. Note that $|w^j| = n_{j+1} - n_j - 1$. Namely, if w were to path through M according to the partition Π then the substring w_j corresponds to the portion of the traversal path of w that lies within the component C_{i_j}.

Lemma 2.7 *Let (A, P, Π) be an admissible triplet , where $A = (C_{i_1}, \dots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$, $\Pi = (n_j)_{j=1}^{t+1}$. Let w be a word of length n satisfying $dist(w, L) \ge \epsilon n$. Define languages $(L_j)_{j=1}^t$ and words $(w_j)_{j=1}^t$ as described above. Then there exists an index j, $1 \le j \le t$, for which $dist(w^j, L_j) \ge \frac{\epsilon n - k}{k}$.*

Proof. Assume this is not the case. Let $\Pi = (n_j)_{j=1}^{t+1}$ be the partition and recall that $t \le k$. For every $1 \le j \le t$, if $n_{j+1} - n_j \ge 2$, 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 $w^{j*} = \gamma$ (the empty word). Also, for $1 \le j \le t - 1$ choose $\alpha_j \in \{0, 1\}$ so that $\delta_M(p_j^2, \alpha_j) = p_{j+1}^1$. Then by construction the word $w^* = w^{1*}\alpha_1 w^{2*} \dots \alpha_{t-1} w^{t*}$ belongs to L and $dist(w, w^*) \le t - 1 + \sum_{j=1}^t dist(w^j, w^{j*}) \le t - 1 + t(\epsilon n - k)/k < \epsilon n$ – 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 $dist(w, L) \ge \epsilon n$, Lemma 2.7 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 2.6 w^j contains many short infeasible runs, and thus sampling a small number of random runs will catch one of them with high probability. However, the problem is that the total number of admissible triplets clearly depends on 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 (depending only on ϵ and the parameters of M) 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 $dist(w, L) \ge \epsilon n$, 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.

Recall that $m = \max_j(m(C_j))$, $l = lcm(\{g(G[C_j])\})$, where j runs over all truly connected components of G, corresponding to vertices of $\mathcal{C}(G)$. Let $S = 129km\log(1/\epsilon)/\epsilon$. We place S transition intervals

$\{T_s = [a_s, b_s]\}_{s=1}^S$ evenly in $[n]$, where the length of each transition interval T_s is $|T_s| = (k-1)(l+m)$. For $1 \le i \le \log(8km/\epsilon)$ define $r_i = \frac{2^{8-i}k^2 m \log^2(\frac{1}{\epsilon})}{\epsilon}$.

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

1. For each $1 \le i \le \log(8km/\epsilon)$ choose r_i random runs in w of length 2^{i+1} each;
2. For each admissible triplet (A, P, Π) with $A = (C_{i_1}, \ldots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$, $\Pi = (n_j)_{j=1}^{t+1}$ such that for all $2 \le j \le t$ one has $n_j \in T_s$ for some $1 \le s \le S$, do the following:
    - Form the automata M_j, $1 \le j \le t$, as described above.
    - Discard those chosen runs which end or begin at place p for which $|p - n_j| \le \epsilon n/(128km\log(1/\epsilon))$. Namely, those runs which have one of their ends closer than $\epsilon n/(128km\log(1/\epsilon))$ from some $n_j \in \Pi$.
    - For each remaining run R, if R falls between n_j and n_{j+1}, check whether it is feasible for the automaton M_j starting at $b - n_j + 1$ where b is the first coordinate of R in w. Namely, $b - n_j + 1$ is the place where R starts relative to n_j, which is the the place w "enters" M_j.
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 run has been found) output "NO".

Lemma 2.8 *If $dist(w, L) \ge \epsilon n$, then the above algorithm outputs "NO" with probability at least $3/4$. If $w \in L$, then the algorithm always outputs "YES".*

Proof. The proof contains two independent parts, in the first we consider the case of an input w with $dist(w, L) \ge \epsilon n$, on which the algorithm should answer 'NO' (with high probability). The other part treats the case where $w \in L$, for which the algorithm should answer 'YES'.

Let us first assume that $dist(w, L) \ge \epsilon n$. The number of admissible 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

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

(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 $|V|^{2k}$ possible choices for portals; then for a fixed pair (A, P) there are at most $S|T_s|$ choices for each n_j, where $2 \le j \le t$ and $t \le k$). For ϵ satisfying condition (*) and S as above, this expression is at most $(1/\epsilon)^{2k}$. Thus we need to check at most $(1/\epsilon)^{2k}$ admissible triplets.

Let (A, P, Π) be an admissible triplet satisfying the restriction formulated in Step 2 of the above algorithm. Write $A = (C_{i_1}, \ldots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$, $\Pi = (n_j)_{j=1}^{t+1}$. Then the triplet defines automata $(M_j)_{j=1}^t$ and languages $(L_j)_{j=1}^t$ as described before. By Lemma 2.7 for some $1 \le j \le t$ one has $dist(w^j, L_j) \ge (\epsilon n - k)/k > \epsilon n/(2k)$. Then by Lemma 2.6 there exists an i, $1 \le i \le \log(8km/\epsilon)$ so that

w^j contains at least $(2^{i-4}\epsilon n/(2km\log(8km/\epsilon)) \geq 2^{i-6}\epsilon n/(km\log(1/\epsilon))$ infeasible runs of length 2^{i+1}. At most $\alpha = \epsilon n/(128km\log(1/\epsilon))$ of them may touch the last α bits of the interval $[n_j, n_{j+1}-1]$, and at most α of them may touch the first α bits of this interval. Hence there are at least $2^{i-6}\epsilon n/(km\log(1/\epsilon))-2\alpha \geq (2^{i-7}\epsilon n)/(km\log(1/\epsilon))$ of them that 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

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

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. This completes the proof for the case of $dist(w, L) \geq \epsilon n$.

We now address the case for which $w \in L$. We need to show that in this case the algorithm answers 'YES'. For this is is enough 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 naturally 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. However, this partition does not necessarily meet the requirement stated in Step 2 of the algorithm: In the true traversal of w in M the transitions from C_{i_j} to $C_{i_{j+1}}$ might occur outside the transition intervals T_s. We show that the desired triplet can be obtained from the actual triplet, (A, P, Π), of w by modifying only the third component of it. This modified triplet would then correspond to a different word $w' \in L$ (which is quite close to w) that makes all the transitions inside the postulated transition intervals. In addition, we will take care that no query is made to bits in which w' differs from w. Hence, the algorithm will actually be consistent with both. This is in fact the reason for discarding the runs that are too close to some n_j in Step 2 of the algorithm. Intuitively, this is done as follows: Assume n_j is not in a transition interval, then we either make the traversal in $C_{i_{j-1}}$ longer so to end in p_{j-1}^2 in a transition interval, or we shorten the traversal in $C_{i_{j-1}}$ so to enter a transition interval, depending on where the closest transition interval is. Formally this is done as follows. Define a new partition $\Pi' = (n'_j)_{j=1}^{t+1}$ where $n'_1 = n_1 = 0$. For each $2 \leq j \leq t$ choose a transition interval T_s closest to n_j. If C_{i_j} is a truly 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 truly 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 may start in a place where w is in $C_{i_{j-1}}$ and end in a place where w is in C_{i_j}, 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 $\delta(q, R) \in C_{i_j}$. Also, q is reachable from p^1_j (the initial state of C_{i_j}) in $b - n_j \geq m$ steps (b is the first coordinate of R). 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^1_j 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". $\Box$

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

$$\sum_{i=1}^{\log(8km/\epsilon)} 2^{i+1} r_i = \sum_{i=1}^{\log(8km/\epsilon)} 2^{i+1} \frac{2^{8-i}k^2 m \log^2(\frac{1}{\epsilon})}{\epsilon} < \frac{2^{10}k^2 m \log^3(\frac{1}{\epsilon})}{\epsilon} .$$

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.*

A final note about the dependence of the complexity on the parameters is in place here. In the proof M is considered fixed, as the algorithm is tailored for a fixed given language. However, in the calculation above we have kept the dependence of the query complexity on the parameters of M explicit. One has to take in mind though that the estimates hold only when condition (*) holds. In particular we require (third item in (*)), that $1/(\epsilon\log(1/\epsilon)) = \Omega(k^2|V|^2 m(l+m))$.

Another note is about the running time of the algorithm (rather then just its query complexity). The dominating term in Step 1 and the first two subsets of Step 2 of the algorithm is the query complexity. In the last substeps, each run has to be checked against M_j. Each such check involves checking whether there is a word u and a word v (of suitable lengths) so that $uRv \in L$. Checking whether there are such u, v is done directly by Lemma 2.3 in case the length of u and v are longer than m, or by checking all 2^m words if one of them is shorter than m.

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 the 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 $d > t/3$. □.

The main idea of the proof of the above proposition can be used to get an $\Omega(1/\epsilon)$ lower bound on the query complexity of testing any non-trivial regular language, with a natural definition of non-trivial. This is proven in the next proposition. A somewhat paradoxical feature of its proof is that our main positive result (Theorem 1) and its proof are used here to get a negative result.

For a language L let $L_n = L \cap \{0,1\}^n$.

Definition 3.1 *A language L is non-trivial if there exists a constant $0 < \epsilon_0 < 1$, so that for infinitely many values of n the set L_n is non-empty, and there exists a word $w \in \{0,1\}^n$ so that $dist(w, L_n) \geq \epsilon_0 n$.*

Proposition 2 *Let L be a non-trivial regular language. Then for all sufficiently small $\epsilon > 0$, any ϵ-testing algorithm for L requires $\Omega(1/\epsilon)$ queries.*

Proof. The proof here is essentially a generalization of the proof of Proposition 1. We thus present it in a somewhat abridged form.

Let n be large enough. Assume $L_n \neq \emptyset$, and $w \in \{0,1\}^n$ is such that $dist(w, L_n) \geq \epsilon_0 n$. We may clearly assume that the constant ϵ_0 is as small as needed for our purposes. Our main result, Theorem 1, and its proof imply that with probability at least 2/3, a random choice of a set of runs, built as described at Step 1 of the testing algorithm of Theorem 1, and having total length $\tilde{O}(1/\epsilon_0)$, will cause the algorithm to reject w. As we have noticed, the testing algorithm has one sided error, i.e., it always accepts a word from L. Thus, if we choose a random set of runs as above, it will cause to reject w with probability 2/3 and it will not coincide with any word $u \in L_n$ (for otherwise, it would reject u too).

Each such random set of runs is just a random set of intervals in $\{1, \ldots, n\}$ (of length as defined in Step 1 of the testing algorithm) of total length bounded by $\tilde{O}(1/\epsilon_0)$. Notice that two such random sets intersect with probability $\tilde{O}(1/(\epsilon_0^2 n))$. Therefore if we choose $\tilde{\Omega}(\epsilon^2 n)$ such subsets at random, then we expect that $\tilde{O}(\epsilon_0^2 n)$ pairs of them will intersect, and that 2/3 of the members will reject w. This implies that there exists a family $\mathcal{S}$ of $\tilde{\Omega}(\epsilon_0^2)n$ pairwise disjoint sets of runs so that for each member of $\mathcal{S}$, no word of L_n coincides with w on this set. Fix now ϵ_0 and let $\epsilon > 0$ be small enough compared to ϵ_0. We partition the family $\mathcal{S}$ into $t = (c/\epsilon)$ subfamilies $\mathcal{S}_1, \ldots, \mathcal{S}_t$, each of cardinality ϵn, where the constant c depends on ϵ_0 only and is thus independent of ϵ. Let u be a word in L_n. For each $1 \leq i \leq t$, the word w_i is obtained from u by changing the bits of u, corresponding to $\mathcal{S}_i$, to those from w. It follows then that $dist(w_i, L_n) \geq \epsilon n$. Indeed, to transform w_i into a word in L_n, at least one bit has to be changed in every member of $\mathcal{S}_i$.

Now, as in the proof of Proposition 1, we define a probability distribution on the union of positive and negative examples. The word u gets probability $1/2$, and each one of the t words $w_1, \ldots, w_t$ gets probability $1/(2t)$. A simple argument, essentially identical to that in the proof of Proposition 1, shows that any deterministic algorithm needs to query at least $\Omega(t) = \Omega(1/\epsilon)$ bits of the input word to be successful with probability at least $2/3$ on the defined probability distribution. Applying Yao's principle, we get the desired result. □

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* ϵ-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.

Theorem 2 *Any ϵ-testing algorithm for the context-free language $L = \{vv^R uu^R\}$, where w^R denotes the reversal of a word w, requires $\Omega(\sqrt{n})$ queries in order to have error of at most $1/3$.*

Proof. Let n be divisible by 6. We again define a distribution D on the union of positive and negative inputs in the following way. 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). We refer to this distribution as N. Positive instances are generated according to a distribution P defined as follows: we pick uniformly at random an integer k in the interval $[n/6+1, n/3]$ and then select a positive example uniformly among words $vv^R uu^R$ with $|v| = k$. Finally the distribution D on all inputs is defined as follows: with probability $1/2$ we choose a positive input according to P and with probability $1/2$ we choose a negative input according to N. We note that a positive instance is actually a pair (k, w) (the same word w may be generated using different k's).

We use the above mentioned Yao's principle again. Let A be a deterministic ϵ-testing algorithm for L. We show that for any such A, if its maximum number of queries is $d = o(\sqrt{n})$, then its expected error with respect to D is at least $\frac{1}{2} - o(1)$. Indeed, let A be such an algorithm. We can view A as a binary decision tree, where each node represents a query to a certain place, and the two outgoing edges, labeled with 0 or 1, represent possible answers. Each leaf of A represents the end of a possible computation, and is labeled 'positive' or 'negative' according to the decision of the algorithm. Tracing the path from the root to a node of A, we can associate with each node t of A a pair (Q_t, f_t), where $Q_t \subseteq \{1, \ldots, n\}$ is a set of queries to the input word, and $f_t : Q_t \to \{0,1\}$ is a vector of answers received by the algorithm.. We may obviously assume that A is a full binary tree of height d and has thus 2^d leaves. Then $|Q_t| = d$ for each leaf t of A.

We will use the following notation. For a subset $Q \subseteq \{1, \ldots, n\}$ and a function $f : Q \to \{0,1\}$, let

$$\begin{aligned} E^-(Q,f) &= \{w \in \{0,1\}^n, dist(w,L) \geq \epsilon n,\ w \text{ coincides with } f \text{ on } Q\}\ , \\ E^+(Q,f) &= \{w \in \{0,1\}^n \cap L :\ w \text{ coincides with } f \text{ on } Q\}\ , \end{aligned}$$

i.e., $E^-(Q,f)$ $(E^+(Q,f))$ is the set of all negative (resp. positive) instances of length n consistent with the pair (Q,f). Also, if D is a probability distribution on the set of binary strings of length n and $E \subseteq \{0,1\}^n$ is a subset, we define $Pr^D[E] = \sum_{w \in E} Pr^D[w]$.

Let T_1 be the set of all leaves of A labeled 'positive', let T_0 be the set of all leaves of T labeled 'negative'. Then the total error of the algorithm A on the distribution D is

$$\sum_{t \in T_1} Pr^D[E^-(Q_t, f_t)] + \sum_{t \in T_0} Pr^D[E^+(Q_t, f_t)]\ .$$

The theorem follows from the following two claims.

Claim 4.1 *For every subset $Q \subset \{1, \ldots, n\}$ of cardinality $|Q| = o(n)$ and for every function $f : Q \to \{0,1\}$,*

$$Pr^D[E^-(Q,f)] \geq \left(\frac{1}{2} - o(1)\right) 2^{-|Q|}\ .$$

Claim 4.2 *For every subset $Q \subset \{1, \ldots, n\}$ of cardinality $|Q| = o(\sqrt{n})$ and for every function $f : Q \to \{0,1\}$,*

$$Pr^D[E^+(Q,f)] \geq \left(\frac{1}{2} - o(1)\right) 2^{-|Q|}\ .$$

Based on Claims 4.1, 4.2, we can estimate the error of the algorithm A by

$$\begin{aligned} \sum_{t \in T_1} Pr^D[E^-(Q_t, f_t)] + \sum_{t \in T_0} Pr^D[E^+(Q_t, f_t)] &\geq \sum_{t \in T_1} \left(\frac{1}{2} - o(1)\right) 2^{-|Q_t|} + \sum_{t \in T_0} \left(\frac{1}{2} - o(1)\right) 2^{-|Q_t|} \\ &= \left(\frac{1}{2} - o(1)\right) \frac{|T_1| + |T_0|}{2^d} \geq \frac{1}{2} - o(1)\ . \end{aligned}$$

The theorem follows. □

We now present the proofs of Claims 4.1 and 4.2.

Proof of Claim 4.1: Notice first that L has at most $2^{n/2}n/2$ words of length n (first choose a word of length $n/2$ and then cut it into two parts v and u, thus getting a word $w = vv^Ruu^R \in L$). Therefore the number of words of length n at distance less than ϵn from L is at most $|L \cap \{0,1\}^n| \sum_{i=0}^{\epsilon n} \binom{n}{i} \leq 2^{n/2+2\epsilon \log(1/\epsilon)n}$. We get

$$\begin{aligned} |E^-(Q,f)| &\geq |\{w \in \{0,1\}^n : w(Q) = f\}| - |\{w \in \{0,1\}^n : dist(w,L) < \epsilon n\}| \\ &\geq 2^{n-|Q|} - 2^{n/2+2\epsilon \log(1/\epsilon)n} = (1-o(1))2^{n-|Q|} \ . \end{aligned}$$

It follows then from the definition of D that

$$Pr^D[E^-(Q,f)] = \frac{1}{2} Pr^N[E^-(Q,f)] \geq \frac{1}{2} \frac{|E^-(Q,f)|}{2^n} \geq \left(\frac{1}{2} - o(1)\right) 2^{-|Q|} \ . \qquad \Box$$

Proof of Claim 4.2: It follows from the definition of the distribution D that for a word $w \in L \cap \{0,1\}^n$,

$$Pr^D(w) = \frac{1}{2} Pr^P(w) = \frac{1}{2} \frac{|\{w = vv^Ruu^R : \ |v| = k, \frac{n}{6}+1 \leq k \leq \frac{n}{3}\}|}{\frac{n}{6} 2^{n/2}} \ .$$

Recall that $E^+(Q,f)$ is the set of words in L for which are consistent with f on the set of queries Q, Hence,

$$Pr^D[E^+(Q,f)] = \frac{1}{\frac{n}{6}2^{n/2+1}} \sum_{k=n/6+1}^{n/3} |\{w \in \{0,1\}^n : w(Q) = f, w = vv^Ruu^R, |v| = k\}| \ .$$

Now observe that for each of the $\binom{d}{2}$ pairs of places in Q there are at most two choices of k, for which the pair is symmetric with respect to k or to $n/2 + k$. This implies that for $n/6 - 2\binom{d}{2} = (1 - o(1))n/6$ choices of k, the set Q does not contain a pair symmetric with respect to k or $n/2+k$. For each such k,

$$|\{w \in \{0,1\}^n : w(Q) = f, w = vv^Ruu^R, |v| = k\}| = 2^{n/2-|Q|} \ .$$

Therefore,

$$Pr^D[E^+(Q,f)] \geq \frac{(1-o(1))\frac{n}{6}2^{n/2-|Q|}}{\frac{n}{6}2^{n/2+1}} = \left(\frac{1}{2} - o(1)\right) 2^{-|Q|} \ . \qquad \Box$$

As a concluding remark to this subsection we would like to note that in the next subsection (Theorem 4) we will give another proof to the fact that not all context-free languages are testable by showing the non-testability of the Dyck language D_2. However, we preferred to give Theorem 2 as well due to the following reasons. First, the language discussed in Theorem 2 is simpler and more natural than the Dyck language D_2. Secondly, the lower bound of Theorem 2 is better than that of Theorem 4. The proofs of these two theorems have many common points, so the reader may view Theorem 2 as a "warm-up" for Theorem 4.

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. $S \rightarrow a_i S b_i$ for $i = 1, \ldots, n$;

2. $S \rightarrow SS$;

3. $S \rightarrow \gamma$,

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 4.3 *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 4.4 *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 4.3 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 two classes of vertices are the set of indices i for which $w[i] = 0$ and the set of indices i for which $w[i] = 1$, respectively, where each i with $w[i] = 1$ is connected to all $1 \leq j < i$ for which $w[j] = 0$. 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 4.5 *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 4.3. □

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 4.4. This observation is used in the analysis of an algorithm for testing D_1, proposed below.

Set

$$\begin{aligned} d &= \frac{C \log\left(\frac{1}{\epsilon}\right)}{\epsilon^2} \ ; \\ \Delta &= \frac{C \log\left(\frac{1}{\epsilon}\right)}{8\epsilon} \ , \end{aligned}$$

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 a sample S of bits in the following way: For each bit of w, independently and with probability $p = d/n$ choose it to be in S. Then, if S contains more then $d + \Delta/4$ bits, answer 'YES' without querying any bit. Else,
2. If $dist(S, D_1 \cap \{0,1\}^{d'}) < \Delta$, where $d' = |S|$, output "YES", otherwise output "NO".

Lemma 4.6 *The above algorithm outputs a correct answer with probability at least $2/3$.*

Proof. As we have already mentioned, we set

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

The proof contains two independent parts, in the first we prove that the algorithm is correct (with probability 2/3) for $w \in D_1$ and in the second part we prove that the algorithm has a bounded error for words w for which $dist(w, D_1) \geq \epsilon n$.

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 4.3 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 4.3 $x(w, n) = y(w, n) = n/2$. 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 $|S|$:

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

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 are satisfied. 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 4.4 $dist(S, D_1) < \Delta$. Thus, if $w \in D_1$, our algorithm will accept w with probability at least 2/3, as required. This ends the first part of the proof.

Let us now consider the negative case. Assume that $dist(w, D_1 \cap \{0,1\}^n) \geq \epsilon n$. By Claim 4.4 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

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

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 $1/6$. But if $y(S,k) - x(S,k) \geq 2\Delta$, it follows from Claim 4.5 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 $1/6$ by the choice of C. But if $x(S,|S|) - y(S,|S|) \geq 2\Delta$, it follows from Claim 4.5 that $dist(S, D_1) \geq \Delta$. Thus in both cases we obtain that our algorithm accepts w with probability at most $1/6$. In addition, the algorithm may accept w (in each of the cases), when $|S| > d + \Delta/4$ (first item in the algorithm). However, by equation (4) this may be bounded by $1/6$ (choosing C as in the first part). Hence the algorithm rejects w with probability at least $2/3$. This completes the proof of Lemma 4.6. □.

By Lemma 4.6 we have the following result about the testability of the Dyck language D_1.

Theorem 3 *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 ϵn indices i in $[1, n/2+\epsilon n] \setminus Q$ and set $w[i] = 0$. For the last ϵn indices i in $[1, n/2+\epsilon n] \setminus Q$ we set $w[i] = 1$. Also, $w[i] = u[i]$ for all $i \in Q$. Now, w satisfies the sufficient condition for the membership in D_1, given by Claim 4.3. Indeed, at any point j in $[1, n/2+\epsilon n]$ the number of 0's in the first j bits of w is at least as large as the number of 1's. Also, for $j \geq n/2 + \epsilon n$ we have $x(w,j) = n/2$ and $y(w,j) = \epsilon n + (j - n/2 - \epsilon n) = j - n/2$. 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 *Any one-sided error ϵ-test for membership in D_1 queries $\Omega(n)$ bits on words of length n.*

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: $S \rightarrow 0S1 \mid 2S3 \mid SS \mid \gamma$, 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 that is obtained by a fixed binary encoding of $\{0, 1, 2, 3\}$.

Theorem 4 *The language D_2 is not ϵ-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 $0.5 \pm o(1)$. Both positive and negative words will be composed of three parts: The first which is a sequence of matching $0/1$ (brackets of the first kind) followed by a sequence of $0/2$ (left brackets) and a sequence of $1/3$ (right brackets).

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 then set $v[n-2k+1-i] = v[i]+1$.

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 $1/2$ a positive instance, generated as above, and with probability $1/2$ a negative instance, chosen according to the above described process.

Claim 4.7 *The probability that an instance generated according to N is ϵn-close to some word in L_n is exponentially small in n.*

Proof. Fix k and let $w = 0^k 1^k v$ be a word of length n generated by N. For such fixed k the three parts of w are the first part of matching $0/1$ of length $2k$, the second part which is a random sequence of $0/2$ of length $\frac{n-2k}{2}$ and the third part which is a random sequence of $1/3$ of length $\frac{n-2k}{2}$. Let us denote by N_1, N_2, N_3 these three disjoint sets of indices of w.

We will bound from above the number of words w of length n of the form $w = 0^k 1^k (0/2)^{\frac{n-2k}{2}} (1/3)^{\frac{n-2k}{2}}$ which are at distance at most ϵn from L_n. First we choose the value of w on N_2, which gives $2^{\frac{n-2k}{2}}$ possibilities. Then we choose (at most) ϵn bits of w to be changed to get a word from L_n ($\binom{n}{\epsilon n}$ choices) and set those bits ($4^{\epsilon n}$ possibilities). At this point, the only part of w still to be set is its value of N_3, where we are allowed to use only right brackets $1, 3$. The word to be obtained should belong to L_n. It is easy to see that there is at most one way to complete the current word to a word in L_n using right brackets only. Hence the number of such words altogether is at most $2^{\frac{n-2k}{2}}\binom{n}{\epsilon n}4^{\epsilon n}$. The total number of words w of the form $0^k1^k(0/2)^{\frac{n-2k}{2}}(1/3)^{\frac{n-2k}{2}}$ is 2^{n-2k}, and each such word gets the same probability in the distribution N. Therefore the probability that a word chosen according to N is ϵn-close to L_n can be estimated from above by

$$\Sigma_{k=n/8}^{n/4}\ \frac{8}{n}\cdot\frac{2^{\frac{n-2k}{2}}\binom{n}{\epsilon n}4^{\epsilon n}}{2^{n-2k}} \leq max_k(2^{O(\epsilon\log(\frac{1}{\epsilon}))n+2\epsilon n-\frac{n}{2}+k}) \leq 2^{-n/5}\ ,$$

for small enough $\epsilon > 0$ as promised. □

Claim 4.8 *Let $S \subseteq [n], |S| = d$, be a fixed set of places and let k be chosen uniformly at random in the range $n/8, ..., n/4$. Then S contains a pair $i < j$ symmetric with respect to $(n-2k)/2$ with probability at most $\binom{d}{2}\frac{8}{n}$.*

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

We now return to the proof of Theorem 4. 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 (as every adaptive A can be made non adaptive by querying ahead at most 2^d possible queries defined by two possible branchings after each adaptive query. We then look at these $2^d = O(1)$ queries as our S). 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 'NO' and let F_1 be all these f's on which it answers 'YES'. Finally denote by (w *positive*) and (w *negative*) the events that a random w is a positive instance and a negative instance, respectively.

The total error of the algorithm is

$$\sum_{f\in F_0} Prob[f_w \wedge (w \text{ positive})] + \sum_{f\in F_1} Prob[f_w \wedge (w \text{ negative})] \geq$$

$$Prob[NoSym]\left(\sum_{f\in F_0} Prob[f_w \wedge (w \text{ positive})|NoSym] + \sum_{f\in F_1} Prob[f_w \wedge (w \text{ negative})|NoSym]\right)$$

However, given that S contains no symmetric pairs, for a fixed f, $Prob[f_w \wedge (w\ is\ negative)]$ is essentially equal to $Prob[f_w \wedge (w\ is\ positive)]$ (these probabilities would be exactly equal if negative w would be generated according to N. Claim 4.7 asserts that N is exponentially close to the real distribution on negative instances). Hence each is of these probabilities is $0.5Prob[f_w|NoSym] \pm o(1)$.

Plugging this into the sum above, and using Claim 4.8 we get that the error probability is bounded from below by $Prob(NoSym)\sum_f(0.5 \pm o(1))Prob[f_w|NoSym] \geq (1-\binom{d}{2}\frac{8}{n})(0.5 \pm o(1)) \geq 0.5 - o(1)$. □

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 Theorem 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. We are also grateful to the anonymous referees for their careful reading.

References

[1] N. Alon, E. Fischer, M. Krivelevich and M. Szegedy, Efficient testing of large graphs, Proceedings of the 40^{th} IEEE FOCS, IEEE (1999), 656–666. Also: Combinatorica, to appear.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, Proof verification and intractability of approximation problems, Journal of the ACM, 45(3) (1998), 501-555.

[3] M. Blum, M. Luby and 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 and 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. J. of the ACM 45 (1998), 653–750. 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 and 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 and 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.