Electronic Colloquium on Computational Complexity, Report No. 152 (2005)

Languages that are Recognized by Simple Counter Automata are not necessarily Testable

Oded Lachish
University of Haifa
Haifa, Israel
loded@cs.haifa.ac.il

Ilan Newman
University of Haifa
Haifa, Israel
ilan@cs.haifa.ac.il

Abstract

Combinatorial property testing deals with the following relaxation of decision problems: Given a fixed property and an input f, one wants to decide whether f satisfies the property or is 'far' from satisfying the property. It has been shown that regular languages are testable, and that there exist context free language which are not testable. We show that there exists a language that is accepted by a deterministic counter automaton for which any 1/25-test requires $\Omega(poly(\log n))$ queries. Thus, proving that even if we restrict ourselves to, perhaps the simplest possible stack automaton model, we do not ensure testability.

ISSN 1433-8092

1 Introduction

Combinatorial property testing deals with the following relaxation of decision problems: Given a fixed property and an input f, one wants to decide whether f satisfies the property or is 'far' from satisfying the property. This notion was first introduced in the work of Blum, Luby and Rubinfeld [1], and was explicitly formulated for the first time by Rubinfeld and Sudan [2]. Goldreich, Goldwasser and Ron [3] have started a rigorous study of what later became known as 'combinatorial property testing'. Since then much work has been done, both on designing efficient algorithms for specific properties, and on identifying natural classes of properties that are efficiently testable (that is, with $O(1)$ queries). For detailed surveys on the subject see [4] and [5].

One of the most important questions in the field is to classify the classes of languages that are testable. In [6] it has been shown that regular languages are testable, and that there exist context free languages which are not testable (see also [7]). On the other hand some CFL languages that are not regular are testable (with $O(1)$-queries). One such example is the first Dyck language, that is, the language of balanced parentheses. Looking more carefully, this language resides in a very low complexity level within the class of CFL languages. Hence, a natural question is whether there exists a computational model stronger than state machines and weaker than stack machines such that all the languages that are accepted by such model are testable (or at least testable with relatively small number of queries). Perhaps the simplest model inside the class of CFL languages is that of a deterministic counter-automaton (also known as one-symbol push down automaton). The first Dyck language can indeed be accepted by such an automaton. We show that there exists a language that is accepted by a counter automaton but is not testable. Thus even this slight generalization of regularity does not ensure testability. On the other hand, we prove that the language that we construct is ϵ-testable with $poly(\log n)$ queries, even for ϵ as small as $1/poly(\log n)$. We leave open the question of whether this upper bound applies to all the languages that are accepted by deterministic counter automata.

2 Preliminaries

2.1 Simple Counter Automaton

A *simple counter automaton* also known as a deterministic one symbol push down automaton (*1-symbol-PDA*) is a finite state automaton equipped with a counter. The possible counter operations are increment, decrement and do nothing, and the only feedback from the counter is whether it is currently 0 or positive (larger than 0). Thus such an automaton, running on a string ω reads an input character at a time, and based on its current state and whether the counter is 0, it jumps to the next state and increments/decrements the counter or leaves it unchanged (decrementing the counter is allowed only if the counter is not 0). Such an automaton accepts a string ω if starting with counter configuration at 0 it reads all the input characters and ends with the counter at 0.

It is quite obvious that such an automaton is equivalent to a deterministic pushdown automaton with one symbol stack (and a read-only bottom symbol to indicate empty stack). This model of computation can recognize a very restricted subset of context free languages. Still, some interesting languages are recognized by such an automaton, e.g. D_1, the language of balanced parentheses. Formal definition and discussion on variants of counter automatons can be found in [8].

2.2 Notations

For a string $\alpha \in \Sigma^n$ and an integer $i \in [n]$, ($[n] = \{1, \ldots, n\}$) we denote by α_i the i-th symbol of α, that is $\alpha = \alpha_1 \ldots \alpha_n$. Given a set $S \subseteq [n]$ such that $S = \{i_1, i_2, \ldots, i_m\}$ and $i_1 < i_2 < \cdots < i_m$ we define $\alpha_S = \alpha_{i_1}\alpha_{i_2} \ldots \alpha_{i_m}$. For two integers $j > i$ we denote by $[i, j] = \{\ell |\ i \leq \ell \leq j\}$.

For a given string $\alpha \in \{0,1\}^n$, an interval $[i, j] \subseteq [n]$ is said to be homogeneous if all the letters in $\alpha_{[i,j]}$ are the same.

For two strings α, β we denote by $dist(\alpha, \beta)$ the Hamming distance between α and β. Namely, $dist(\alpha, \beta) = |\{i |\ \alpha_i \neq \beta_i\}|$. For a property $\mathcal{P} \subseteq \Sigma^n$ and a string $\alpha \in \Sigma^n$, $dist(\alpha, \mathcal{P}) = min\{dist(\alpha, \beta) |\ \beta \in \mathcal{P}\}$ denotes the distance from α to $\mathcal{P}$. We say that α is ϵ-*far* from $\mathcal{P}$ if $dist(\alpha, \mathcal{P}) \geq \epsilon n$, otherwise we say that α is ϵ-*close* to $\mathcal{P}$.

2.3 The Language $\mathcal{L}$

We construct here a language $\mathcal{L}$ which is accepted by a simple counter automaton but is not testable. $\mathcal{L}$ is the language of all possible concatenations of strings of the sort 0^t1^t. Formally:

Definition 2.1. *$\mathcal{L}$ is the minimal language of all strings such that:*

1. *The empty string belongs to $\mathcal{L}$.*
2. *For every positive integer t we have $0^t1^t \in \mathcal{L}$.*
3. *If $\alpha_1, \alpha_2 \in \mathcal{L}$ then $\alpha_1\alpha_2 \in \mathcal{L}$.*

Where $\alpha_1\alpha_2$ denotes the concatenation of α_1,α_2.

We remark that the language $\mathcal{L}$ is isomorphic to a subset of D_1, the first Dyck language, which is the language of properly balanced parentheses of one type.

Let $\mathcal{L}_n$ be the property $\mathcal{L}_n = \mathcal{L} \cap \{0,1\}^n$.

3 Lower Bound for Testing $\mathcal{L}_n$

Theorem 3.1. *Any non adaptive, two sided error, $1/48$-testing algorithm for $\mathcal{L}_n$, uses $\Omega(\log n / \log\log n)$ queries.*

3.1 Proof Outline

We prove the theorem by using Yao's principle. That is, we construct a distribution $\mathcal{D}$ over legitimate instances (strings that are either in $\mathcal{L}_n$, or strings that are $1/48$-*far* from $\mathcal{L}_n$) and prove that any non adaptive *deterministic* tester, that uses $o(\log n / \log\log n)$ queries, gives an incorrect answer with probability strictly greater than $1/3$.

The construction of distribution $\mathcal{D}$ is based on the following sets:

$$BAD_\ell = \left\{0^\ell 1^\ell 0^\ell 1^\ell 0^\ell 1^{3\ell} 0^{3\ell} 1^\ell,\ 0^{2\ell} 1^{2\ell} 0^{3\ell} 1^\ell 0^\ell 1^{3\ell}\right\},$$

$$GOOD_\ell = \left\{\left(0^{2\ell}1^{2\ell}\right)^3, \left(0^\ell 1^\ell\right)^6\right\}.$$

where ℓ is an integer. Note that each one of the 4 strings is of length 12ℓ. We refer to strings selected from these sets as 'phrase strings'. We view the phrase strings as being composed of 12 disjoint intervals of length ℓ, which we refer to as 'phrase segments'. By the definition of the 'phrase strings' each 'phrase segment' is an homogeneous substring. It is easily verified that for every integer ℓ, every string in BAD_ℓ is at least 1/48-*far* from $\mathcal{L}_{12\ell}$.

The idea behind the construction of $\mathcal{D}$ and the intuition of the lower bound is illustrated here. Assume that Alg is an algorithm for ϵ testing $\mathcal{L}_n$. Then in particular Alg should distinguish with high probability between a string uniformly selected from BAD_ℓ and a string uniformly selected from $GOOD_\ell$. Note however that any single fixed query can't distinguish between the two cases. In order to distinguish between the two cases Alg must query more than one query. Moreover, at least two of the queries must be in two different 'phrase segments' excluding the first and last.

In the construction of distribution $\mathcal{D}$ we select with probability 1/2 whether the string we choose will be a positive instance or a negative instance. We select a positive instance by concatenating a set of strings uniformly and independently selected from $GOOD_\ell$ with strings from $\mathcal{L}$. We construct negative instance in the same manner except that we replace the selection of strings from $GOOD_\ell$, by selecting strings from BAD_ℓ. Thus the only way to distinguish between a positive instance and a negative instance is if at least two queries are located in the same phrase string, but in different phrase segments (excluding the first and last). $\mathcal{D}$ will be such that if the number of queries that is used is $o(\log n/\log\log n)$, then with high probability there will be no two queries in two different phrase segments that belong to the same phrase string. Since each phrase string is selected independently this ensures that the tester will have no chance of knowing whether the string is a positive instance or a negative one.

3.2 The Distribution $\mathcal{D}$

Let $\mathcal{D}_N$ be a distribution over $\{0,1\}^n$ that is defined by the following process of generating a string $\alpha \in \{0,1\}^n$:

1. Uniformly select a constant $s \in [(\log n)/1000, (\log n)/10]$ and set $\ell = 2^s$.
2. Independently and uniformly select integers $b \in [6\ell]$, until the first time that the integers $b_1, \ldots, b_r$ selected satisfy $\sum_{i=1}^{r}(2b_i + 12\ell) \geq n - 24\ell$.
3. Independently uniformly select r strings $\beta_1, \ldots, \beta_r \in BAD_\ell$.
4. For each $i \in [r]$ set $B_i = 0^{b_i}1^{b_i}\beta_i$. We refer to B_i as the i'th 'block string'. We refer to the substring $0^{b_i}1^{b_i}$ as the 'buffer string' and β_i as the 'phrase'.
5. Set $\alpha = B_1 \cdots B_r 0^t 1^t$, where $t = (n - \sum_{i=1}^{r}|B_i|)/2$.

Let $\mathcal{D}_P$ be a distribution over $\{0,1\}^n$ that is defined in the same manner as $\mathcal{D}_N$ with the exception that in the third stage we select independently and uniformly r strings $\beta_1, \ldots, \beta_r \in GOOD_\ell$.

Observe that $\mathcal{D}_P$ assigns strictly positive probability only to strings in $\mathcal{L}_n$.

Lemma 3.2. *The Distribution $\mathcal{D}_N$ is supported on strings in $\{0,1\}^n$ that are 1/48-far from $\mathcal{L}_n$.*

Proof: Recall that when a string α is selected according to $\mathcal{D}_N$ it contains r disjoint substrings $\beta_i \in BAD_\ell,\ i = 1, \ldots, r$. In addition as $b_i \leq 6\ell$ it follows that the total accumulated length of the phrase strings in α is at least $n/4$.

Since each $\beta \in BAD_\ell$ contains the substring $1^\ell 0^\ell 1^{3\ell}$ it is easy to verify that the closest word $\alpha' \in \mathcal{L}_n$ to α must be different from α, in every phrase, in at least ℓ places. Thus α' must differ from α in a 1/48-fraction of the places in the whole word. ■

3.3 Proof of Main Theorem

Let Alg be any fixed deterministic algorithm that uses $d = o(\log n/\log\log n)$ queries. We will show that the error Alg has when trying to distinguish between the case that the input is drawn from $\mathcal{D}_P$ and the case it is drawn from the distribution $\mathcal{D}_N$ is at least 1/3.

As the lower bound is proved for the non adaptive case, we may assume that the queries $Q = \{q_i,\ i = 1,\ldots,d\}$ are fixed in advance and renumbered such that $q_1 < q_2 \cdots < q_d$. Let $\Delta_i = q_{i+1} - q_i,\ i = 1,\ldots d-1$ be the distances between consecutive queries.

We first show that conditioned on a certain 'good' event $\mathcal{B}$, the error of Alg is indeed 1/2. We will then show that the event $\mathcal{B}$ happens with very high probability.

Let $\mathcal{B}$ be the event that the integers $\ell, b_1, \ldots, b_r$ (that are selected in steps 1 and 2 in the definition of D_N and D_P) are such that for every $i < j \leq d$, if q_i, q_j are in the same phrase of α then they are also in the same phrase segment. Let ERR be the event that Alg errs on α chosen according to $\mathcal{D}$.

Claim 3.3. $Prob(ERR\ |\mathcal{B}) = 1/2$.

Proof: Note that when an instance is selected according to $\mathcal{D}$ the values $\ell, b_1, \cdots, b_N$ determine whether event $\mathcal{B}$ is satisfied (regardless of the values of the block strings).

Let $\beta \in \{0,1\}^d$ be any vector of answers to the d queries. It is enough to show that for every $\beta \in \{0,1\}^d$, $\mathrm{Prob}_{\mathcal{D}_P}(\alpha_Q = \beta\ |\mathcal{B}) = \mathrm{Prob}_{\mathcal{D}_N}(\alpha_Q = \beta\ |\mathcal{B})$.

Indeed assume that $\mathcal{B}$ is satisfied. We may assume w.l.o.g that all queries are in phrases of α as the distribution that is induced on queries that are outside phrases is identical for $\mathcal{D}_P$ and $\mathcal{D}_N$. We can partition $Q = \{q_1, \ldots, q_d\}$ into disjoint sequences $Q_1 = \{q_1, \ldots, q_{i_1}\}$, $Q_2 = \{q_{i_1+1}, \ldots, q_{i_2}\}$, $\ldots$, $Q_u = \{q_{i_u+1}, \ldots, q_d\}$ such that, for every j, all queries in Q_j lie in the same phrase of α while for $j \neq j'$ $Q_j, Q_{j'}$ are in different phrases.

As $\mathcal{B}$ is satisfied, all queries in every Q_j lie in the same *phrase segment* of α. According to the way α is constructed (in both $\mathcal{D}_P, \mathcal{D}_N$) this implies that $\alpha_{Q_j} = 1^{|Q_j|}$ with probability 1/2 and $0^{|Q_j|}$ with probability 1/2, unless Q_j is in the first or last phrase segment in which case α_{Q_j} is deterministically set in the same way for $\mathcal{D}_P$ and $\mathcal{D}_N$. ■

The proof of the theorem now follows from Lemma 3.4 below. ■

Lemma 3.4.

$$\mathrm{Prob}(\mathcal{B}) \geq 1 - o(1).$$

Proof:

Let $\mathcal{A}$ be the event that the integer ℓ, chosen according to the distribution $\mathcal{D}$, satisfies ($\Delta_i \leq \ell/\log n$) or ($\Delta_i \geq 24\ell$) for every $i = 1, \ldots d-1$. We think of $\mathcal{A}$ as asserting that either the queries are extremely close, or they are extremely far (lie in different phrases in the word α).

Claim 3.5.

$$\mathrm{Prob}(\mathcal{A}) \geq 1 - o(1).$$

Proof. Recall that $\ell = 2^s$ and $\Delta_i = q_{i+1} - q_i$ for every $i \in [d]$. Let q_i, q_{i+1} be two consecutive queries and set $p_i = \mathrm{Prob}_{\mathcal{D}}\left(\frac{\ell}{\log n} \leq \Delta_i \leq 24\ell\right)$ then

$$p_i = \mathrm{Prob}_{\mathcal{D}}\left(\frac{2^s}{\log n} \leq \Delta_i \leq 24 \cdot 2^s\right).$$

By a simple manipulation we get

$$p_i = \text{Prob}_{\mathcal{D}}\left(\log \Delta_i - \log 24 \leq s \leq \log \Delta_i + \log\log n\right).$$

Since s is distributed uniformly in $\left[\lfloor \frac{1}{1000}\log n \rfloor, \lfloor \frac{1}{10}\log n \rfloor\right]$

$$p_i \leq \frac{\log\log n + \log 24}{\log n} \cdot \frac{1000}{99}$$

By the union bound we have for $d = o(\log n/\log\log n)$:

$$\text{Prob}_{\mathcal{D}}(\mathcal{A}) \geq 1 - d \cdot p_i = 1 - o(1).$$

■

Claim 3.6.

$$\text{Prob}\left(\mathcal{B} \mid \mathcal{A}\right) > 1 - o(1).$$

Proof. Assume that $\ell, b_1, \ldots, b_r$ is such that for some $i < j$, q_i, q_j are in the same phrase of α. Assume also that $\mathcal{A}$ happens, then $|q_j - q_i| < 12\ell$ (as those two queries lie in the same phrase) and $\Delta_k < \ell/\log n$ for every $k \in [i,j]$ (as $\mathcal{A}$ holds).

According to the definition of $\mathcal{D}$ the block sizes are at most 24ℓ. Therefore, q_i and q_j, are in a block substring B which starts in location $b \in [max(0, q_i - 24\ell), q_i]$. Since $d = o(\frac{\log n}{\log\log n})$ by assumption, then $|q_j - q_i| \leq \Delta_k \cdot d = o(\frac{\ell}{\log\log n})$. Recall that every phrase segment is of size ℓ and hence q_i and q_j could either be in the same phrase segment or in two consecutive segments. Which one of the cases happens is entirely determined by the sizes of the buffers $(b_1, \ldots, b_r)$. The probability that $[q_i, q_j]$ intersects two specific consecutive phrase segments is at most $(q_j - q_i)/(6\ell)$. In the worst case there are 11 pairs of consecutive phrase segments in a phrase, thus, by the union bound, the probability that $[q_j, q_i]$ is not in the same phrase segment is at most $11(q_j - q_i)/(6\ell)$.

This analysis is true for any $i < j$ for which q_i, q_j are in the same phrase. Given ℓ and $b_1, \ldots, b_N$ we can partition $Q = \{q_1, \ldots, q_d\}$ into subsets $Q_1 = \{q_1, \ldots, q_{i_1}\}$, $Q_2 = \{q_{i_1+1}, \ldots, q_{i_2}\}$, $\ldots$, $Q_u = \{q_{i_{u-1}+1}, \ldots, q_d\}$, where for any j all queries in Q_j are in the same phrase. Assuming that $\mathcal{A}$ holds the analysis above holds for every pair of indices $i_{j-1}+1, i_j$. Hence, by the union bound we have that:

$$\text{Prob}_{\mathcal{D}}\left(\mathcal{B} \mid \mathcal{A}\right) > 1 - \frac{11(q_{i_1} - q_1)}{6\ell} - \frac{11(q_{i_2} - q_{i_1+1})}{6\ell} - \cdots - \frac{11(q_d - q_{i_{u-1}+1})}{6\ell} = 1 - \frac{11}{6\ell} \cdot \frac{d \cdot \ell}{\log n}$$

Since $d = o(\log n/\log\log n)$ the claim is proved and so is the Lemma. ■

We note that for 1-sided error algorithms the lower bound is $\Omega(n)$ since there is no witness of length $o(n)$ that the string $0^{\frac{n}{4}}1^{\frac{3n}{4}}$ is not in $\mathcal{L}_n$.

4 Upper Bound for testing $\mathcal{L}_n$

We construct here a non-adaptive 2-sided error ϵ-test for $\mathcal{L}_n$, for every $\epsilon \geq \frac{20}{\log n}$ and query complexity $poly(\log n)$. The idea behind the algorithm is the following: Suppose that by querying a string $\alpha \in \{0,1\}^n$ we find a substring $uvwx$ where $|x|, |u| << |v| << |w|$ and such that u contains a

$'1'$, x contains a $'0'$, v contains mostly 0's and w contains mostly 1's. This, then provides a witness that the string α is not in $\mathcal{L}_n$. We show that if a string α is ϵ-far from $\mathcal{L}_n$, then it contains many such substrings. In order to 'find' such substrings we look for them in different length scales. This motivates the following definitions.

Definition 4.1. [Basic Substring]*: A substring $\alpha_{[x,y]}$ of $\alpha \in \{0,1\}^n$ is called "Basic" if x is odd and one of the following is satisfied.*

1. *$\alpha_{[x,y]}$ is a maximal substring of the sort 0^s1^t, $s \geq 0$, $t \geq 0$ and $s+t$ is even.*
2. *$\alpha_{[x,y]} = 10$.*

A basic substring $\alpha_{[x,y]}$ is called a "good" substring if $\alpha_{[x,y]} = 0^s1^s$ and a "bad" substring otherwise.

Definition 4.2. [Short Substring]*: A substring $\alpha_{[x,y]}$ of $\alpha \in \{0,1\}^n$ is called "short" if $y - x \leq \log^3(n)$.*

Fact 4.3. *Every string in $\{0,1\}^n$ is a concatenation of basic substrings. A string $\alpha \in \{0,1\}^n$ is in $\mathcal{L}_n$ if and only if each one of its basic substrings are good substrings.*

The goal of the tester is to find a 'witness' that a string is not in $\mathcal{L}_n$. Bad substrings would have been a natural choice if it was true that every string that is far from $\mathcal{L}_n$ has many 'short' bad substrings. However bad substrings alone are not sufficient as witnesses since a string can be far from $\mathcal{L}_n$ and have only long bad substrings. The main part of the test will be to 'catch' such cases by finding a suitable "probabilistic" witness. This will result in a 2-sided test.

In order to deal with bad substrings of different scales we introduce the following collection of partitions of a string into substrings.

For each $k \in [2, \frac{\log n}{\log\log n} - 2]$ let $\mathcal{P}_k$ be the partition of $[n]$ into contiguous intervals each of length $n/\log^k n$. We assume for simplicity here, that $n/\log^k n$ is even for each $k \in [2, \frac{\log n}{\log\log n} - 2]$. Note that by this definition each such interval begins in an odd location and ends in an even location. We denote the i'th interval in $\mathcal{P}_k$ by $part_k(i)$ and we set $p_k = |\mathcal{P}_k| = log^k n$. We also use the notation $part_k([x,y])$ to denote the interval that contains all intervals $part_k(i)$ for $i \in [x,y]$ and denote by $|part_k([x,y])|$ the length of $part_k([x,y])$, that is $|part_k([x,y])| = (y-x+1) \cdot n/p_k$.

For every string in $\beta \in \{0,1\}^*$ we set $N_1(\beta) = \frac{|\{i|\beta_i=1\}|}{|\beta|}$ and $N_0(\beta) = \frac{|\{i|\beta_i=0\}|}{|\beta|}$. That is $N_1(\beta)$ is the fraction of 1's in β and $N_1(\beta)$ is the fraction of 0's in β. A color of a string β, denoted $Col(\beta)$ is $'0'$ if $N_1(\beta) < \frac{1}{2\log n}$, $'1'$ if $N_0(\beta) < \frac{1}{2\log n}$ and $'G'$ (for gray) otherwise. Namely, if $Col(\beta) \in \{0,1\}$ then β contains almost only $Col(\beta)$ letters, while if $Col(G) = G$ then β contains a significant amount of both 0's and 1's. Obviously a coloring of a string roughly approximates it. For a $\alpha \in \{0,1\}^n$ every partition $\mathcal{P}_k$ naturally defines an approximation by $Col_k(\alpha) = Col(\alpha_{part_k(1)})Col(\alpha_{part_k(2)}) \ldots Col(\alpha_{part_k(p_k)})$.

Definition 4.4. [Basic Subcoloring]*: A substring $\beta_{[x,y]}$ of a string $\beta \in \{0,G,1\}^m$ is called a basic subcoloring if $\beta_{[x,y]}$ is a maximal substring of the sort $0^sG^z1^t$, where $z \in \{0,1\}$,$s \geq 0$, $t \geq 0$. We call a basic subcoloring "bad subcoloring" if $|s-t| \geq 7$ and "good subcoloring" otherwise.*

Fact 4.5. *Given a string $\alpha \in \{0,1\}^n$ if there exists $k \in [2, \frac{\log n}{\log\log n} - 2]$ such that $Col_k(\alpha)$ has a bad subcoloring then $\alpha \notin \mathcal{L}_n$.*

We note that there exist strings $\alpha \notin \mathcal{L}_n$, such that for every $k \in [2, \frac{\log n}{\log\log n} - 2]$, $Col_k(\alpha)$ doesn't have a bad subcoloring. An immediate result of Theorem 4.9 which we state and prove further on is that such strings are not far from $\mathcal{L}_n$.

We continue this section as follows. In subsection 4.1 we introduce an algorithm and prove that it is the claimed tester. The main part of the proof is the completeness of the algorithm which is presented in subsection 4.2.

4.1 Algorithm $\mathcal{A}$ for testing $\mathcal{L}_n$

The algorithm consists of two phases. The goal of the first phase is to find a "short" bad substring. The goal of the second phase is to find an evidence of a "long" bad substring. Given a string of length n the first is done by uniformly and independently selecting a number of short subintervals of $[n]$ and querying each location in a selected subinterval. The algorithm rejects if it finds a bad substring in this phase. In the second phase; for every $k \in [2, \frac{\log n}{\log\log n} - 2]$, the algorithm uniformly and independently selects a number of short subintervals of $[p_k]$ and then for every location i in each one of the selected subintervals it approximates the color of the corresponding part. The approximated color will be a letter in $\{0, \tilde{0}, G, \tilde{1}, 1\}$. The $\tilde{0}$ should be interpreted as almost all zeros and $\tilde{1}$ should be in the same manner. All other letters retain their original meaning. The reason two new letters are added is that we want the probability that $'G'$ is mistaken to be $'0'$ or $'1'$ and the probability a $'0'$ or $'1'$ is mistaken to be $'G'$ to be very low. The algorithm then rejects if a bad subcoloring is found. The approximation is done as follows;

Approximator

Input: a string $\alpha \in \{0,1\}^n$, an interval $I \subseteq [n]$;

1. Independently and uniformly select $r = \log^3 n$ entries in I. Let n_1 be the number of entries i thus selected, for which $\alpha_i = 1$.
2.
   - If $\frac{n_1}{r} = 0$ return 0.
   - If $0 < \frac{n_1}{r} < \frac{1}{\log n}$ return $\tilde{0}$.
   - If $\frac{1}{\log n} \leq \frac{n_1}{r} \leq 1 - \frac{1}{\log n}$ return G.
   - If $1 - \frac{1}{\log n} < \frac{n_1}{r} < 1$ return $\tilde{1}$.
   - If $\frac{n_1}{r} = 1$ return 1.

Algorithm $\mathcal{A}$

Input: a string $\alpha \in \{0,1\}^n$ and the string length n;

1. Repeat the following $\log^3 n$ independent times:
   - Select an integer s uniformly from $[n - (2\log^3 n + 1)]$ and query α for each location $q \in [s, s + 2\log^3 n + 1]$.
   - If $\alpha_{[s,s+2\log^3 n]}$ has a bad substring then reject.

2. For each $k \in [2, \frac{\log n}{\log\log n} - 2]\}$ repeat the following independently $\log^3 n \cdot \log\log n$ times:

   - Select an integer s uniformly from $[p_k - (2\log^2 n + 1)]$ (if $k = 2$ set $s = 1$) and for each $i \in [2\log^2 n + 1]$ set $\beta_i = \mathit{Approximator}(\mathit{part}_k(s+i))$ and $\beta = \beta_1\beta_2 \ldots \beta_r$, where $r = 2\log^2 n + 1$.
   - If β contains a substring of the form $\{\vdash, \tilde{0}, G, \tilde{1}, 1\}\{0, \tilde{0}\}^x G^y \{\tilde{1}, 1\}^z \{0, \tilde{0}, G, \tilde{1}, \dashv\}$, where $\vdash, \dashv$ are the beginning and end of the string respectively, $y \in \{0, 1\}$ and $|x - z| > 6$ then reject.

3. If not rejected then accept.

Theorem 4.6. *Algorithm $\mathcal{A}$ is a 2-sided error, non-adaptive, $(\frac{20}{\log n}, \mathit{poly}(\log n))$-tester for $\mathcal{L}_n$.*

Note that algorithm $\mathcal{A}$ can be run by first asking all the queries and then deciding to reject or accept. Thus the algorithm is non-adaptive.

We first bound the query complexity of algorithm $\mathcal{A}$. In phase 1 an order of $\log^3 n$ queries are used in each one of the $\log^3 n$ iterations. In phase 2 an order of $\log^2 n$ calls to the approximator are used in $O(\log n/\log\log n)$ iterations. In each call to the approximator $\log^3 n$ queries are used. Thus the algorithm uses $O(\log^6 n)$ queries. It remains to prove the completeness and soundness of the algorithm. In order to achieve this goal we first show that with high probability every called to the *Approximator* "behaves well".

We say that algorithm $\mathcal{A}$ "approximates well" if on every call to the Approximator with a string $\alpha \in \{0,1\}^n$ and an interval $I \subseteq [n]$ the following happens. If $Col(\alpha_I) = 0$ the Approximator returns 0. If $Col(\alpha_I) = 1$ the Approximator returns 1. If $Col(\alpha_I) = G$ the Approximator returns an answer from $\{\tilde{0}, G, \tilde{1}\}$.

Claim 4.7. *Algorithm $\mathcal{A}$ "approximates well" with probability at least $8/9$.*

Proof. The proof is immediate from the Chernoff bound and the union bound. ■

Definition 4.8. [Adequate subcoloring]*: A basic subcoloring $Col_k(\alpha)_{[x,y]}$ (bad or good) is called adequate if $(\log n)/2 \leq y - x \leq \log^2 n$.*

Given $\alpha \in \{0,1\}^n$ we denote by $Bad(\alpha)$ the sum over the lengths of every short bad substring of α, whose length is at most $\log^3 n$. For each $k \in [2, \frac{\log n}{\log\log n} - 2]$ we denote by $Bad_k(\alpha)$ the sum of the lengths of every adequate bad subcolorings of $Col_k(\alpha)$.

Theorem 4.9. *If α is $20/\log n$-far from $\mathcal{L}_n$ then $Bad(\alpha) \geq n/\log^2 n$ or there exists $k \in [2, \frac{\log n}{\log\log n} - 2]$ such that $Bad_k(\alpha) \geq p_k/\log^2 n$.*

We prove this theorem in subsection 4.2, next we will show how it implies the completeness of algorithm $\mathcal{A}$.

Lemma 4.10. *If α is $20/\log n$-far from $\mathcal{L}_n$ then algorithm $\mathcal{A}$ rejects α with probability at least $2/3$.*

Proof. Let $\alpha \in \{0,1\}^n$ be $20/\log n$-far from $\mathcal{L}_n$. By Theorem 4.9 either $Bad(\alpha) \geq n/\log^2 n$ or there exists a $k \in [\frac{\log n}{\log\log n} - 2]$ such that $Bad_k(\alpha) \geq p_k/\log^2 n$. We first prove that if $Bad(\alpha) \geq n/\log^2 n$ then algorithm $\mathcal{A}$ rejects with probability at least $2/3$.

Assume that $Bad(\alpha) \geq n/\log^2 n$. By phase 1 if an integer s is selected in the interval $[x - log^3 n, y - log^3 n]$ such that $\alpha_{[x,y]}$ is a short bad substring then there is a bad substring in $\alpha_{[s,s+2log^3 n+1]}$. Since the algorithm queries each location in $[s, s + 2log^3 n + 1]$ it rejects. Set W_1 to be the set of all locations that are in an interval $[x - log^3 n, y - log^3 n]$ for which $\alpha_{[x,y]}$ is a short bad substring. Note that, by definition, every two bad substrings of α are pairwise disjoint. Thus since $Bad(\alpha) \geq n/\log^2 n$ we get that $|W_1| \geq \frac{n-2log^3 n}{\log^2 n}$. The probability that algorithm $\mathcal{A}$ rejects is at least the probability that $s \in W_1$ for one of the integers s selected in phase 1. Since the algorithm selects uniformly and independently $\log^3 n$ such integers from $[n - 2\log^3 n]$, the probability that the algorithm rejects is at least $1 - (1 - \frac{|W_1|}{n-(2\log^3 n-1)})^{\log^3 n} \geq \frac{8}{9}$.

Assume that α is such that $Bad_k(\alpha) \geq p_k/\log^2 n$ for some $k \in [2, \frac{\log n}{\log\log n} - 2]$. Let *Catch* be the event that in phase 2 algorithm $\mathcal{A}$ selects an integer s in an interval $[x - log^2 n, y - log^2 n]$ such that $Col_k(\alpha)_{[x,y]}$ is an adequate bad subcoloring. Note that if *Catch* occurs then algorithm $\mathcal{A}$ calls the Approximator for every interval $part_k(i)$ such that $i \in [x-1, y+1]$ and $Col_k(\alpha)_{[x,y]}$ is an adequate bad subcoloring. Set W_k to be the set of all locations that are in an interval $[x - log^3 n, y - log^3 n]$ such that $Col_k(\alpha)_{[x,y]}$ is an adequate bad subcoloring. By definition every two bad subcolorings of $Col_k(\alpha)$ are disjoint. Hence since $Bad_k(\alpha) \geq p_k/\log^2 n$ we get that $|W_k| = \frac{p_k - 2log^3 n}{2\log n}$. The probability that event *Catch* occurs is the probability that $s \in W_k$ for one of the integers s selected in phase 2. Since in phase 2 algorithm $\mathcal{A}$ selects uniformly and independently $5(\log^2 n)$ locations from $[n - 2\log^3 n]$ the probability that Catch Occurs is at least $1 - (1 - \frac{|W_k|}{n-(2\log^2 n} + 1))^{\log^3 n} \geq \frac{8}{9}$.

Observe that if event Catch occurs and the algorithm approximates well then it rejects. According to Claim 4.7 and the union bound the probability that this happens is at least 2/3. ■

Lemma 4.11. *If $\alpha \in \mathcal{L}_n$ then algorithm $\mathcal{A}$ accepts α with probability at least 2/3.*

Proof. Let α be a string in $\mathcal{L}_n$. It is immediate that algorithm $\mathcal{A}$ does not reject in phase 1, since by Fact 4.3, α does not have bad substrings. Thus it remains to prove that algorithm $\mathcal{A}$ does not reject in phase 2.

We show that if algorithm $\mathcal{A}$ approximates well then it does not reject α in phase 2. Assume for contradiction that algorithm $\mathcal{A}$ approximates well and rejects. Then according to the rejection condition in phase 2 the algorithm found for some $k \geq 2$, a substring β of $Col_k(\alpha)$ of the form $\{\vdash, \tilde{0}, G, \tilde{1}, 1\}\{0, \tilde{0}\}^x G^y \{\tilde{1}, 1\}^z \{0, \tilde{0}, G, \tilde{1}, \dashv\}$ such that $y \in \{0, 1\}$ and $|y - z| > 6$. Since algorithm $\mathcal{A}$ approximated well it is easy to see that α has a bad substring in the substring that corresponds to the parts of β, in contradiction to the assumption that $\alpha \in \mathcal{L}_n$. ■

4.2 Proof of Theorem 4.9

In order to prove that Theorem 4.9 we introduce an algorithm that on a string $\alpha \in \{0, 1\}^n$ it finds a new string $\beta \in \mathcal{L}_n$ by changing the letters in specific locations of α. We show that if $Bad(\alpha) < n/\log^2 n$ and $Bad_k(\alpha) < p_k/\log^2 n$ for every $k \in [2, \frac{\log n}{\log\log n} - 2]$, this algorithm changes less than $20/\log n$ fraction of the locations of α. This implies the theorem.

An informal description of this algorithm is presented here before its formal statement. The algorithm uses two strings the input string α and the output string β which is initially set to M^n, where M is a special symbol. The algorithm has 4 major phases. In each one of these phases the algorithm selects some intervals and changes β in the corresponding intervals to words in $\mathcal{L}$. These changes are done so that each location is β is changed **exactly** once. More specifically, in

the first phase, for every partition starting with the coarsest one ($\mathcal{P}_2$), and ending with the most refined ($\mathcal{P}_m$ for $m = \frac{\log n}{\log\log n} - 2$), the algorithm takes each subinterval $[w,x]$ for which $Col_k(\alpha)_{[w,x]}$ is an adequate good subcoloring and changes $\beta_{part_m([w,x])}$, or the part of it that does not intersects previously changed locations, into a string in $\mathcal{L}$. In the second phase for every $[w,x]$ for which $\alpha_{[w,x]}$ is a short good substring the algorithm changes $\beta_{[w,x]}$ into part of a string in $\mathcal{L}$. The third stage of the algorithm is the same as the first except that is it done for $[w,x]$ for which $Col_k(\alpha)_{[w,x]}$ is an adequate bad subcoloring. In the last stage every interval $[w,x]$ that is a maximal unchanged interval in β is changed into a string in $\mathcal{L}$. Follows is a formal description of the algorithm, denoted as Algorithm Fix.

Algorithm *Fix*

1. Set $\beta = M^n$.
2. For $k = 2$ to $k = \log n/\log\log n - 2$ do,
   - For each interval $[w,x]$ such that $Col_k(\alpha)_{[w,x]}$ is an adequate good subcoloring of $Col_k(\alpha)$ and for each $[y,z] \subseteq part_k([w,x])$ such that $\beta_{[y,z]}$ is a maximal substring of the form M^*, set $\beta_{part_k([y,z])} = 0^t1^t$, where $t = |part_k([y,z])|/2$.
3. For each interval $[w,x]$ for which $\alpha_{[w,x]}$ is a short good substring and each $[y,z] \subseteq part_k([w,x])$ such that $\beta_{[y,z]}$ is a maximal substring of the form M^*, set $\beta_{[y,z]} = 0^t1^t$, where $t = (z-y)/2$.
4. For $k = 2$ to $k = \log n/\log\log n - 2$ do,
   - For each interval $[w,x]$ such that $Col_k(\alpha)_{[w,x]}$ is an adequate bad subcoloring of $Col_k(\alpha)$ and for each $[y,z] \subseteq part_k([w,x])$ such that $\beta_{[y,z]}$ is a maximal substring of the form M^* set $\beta_{part_k([y,z])} = 0^t1^t$, where $t = |part_k([y,z])|/2$.
5. For each interval $[x,y]$ such that $\beta_{[x,y]}$ is a maximal substring of the form M^* set $\beta_{[x,y]} = 0^t1^t$, where $t = (y-x)/2$.
6. Return β.

Claim 4.12. *Given a string $\alpha \in \{0,1\}^n$ algorithm Fix returns a string $\beta \in \mathcal{L}_n$.*

Proof. Let α and β be as in the claim. Every substring of β that is changed at any step during phases 2 to 4 is 0^t1^t for some t and starts at an odd place. Hence at the end of phase 4, β is a concatenation of strings from $\mathcal{L}$ and strings of the form M^*. Thus after phase 4 the substrings containing only M's start in odd locations and have even length. Hence after phase 5, β is a concatenation of strings in $\mathcal{L}$. ■

Lemma 4.13. *If $Bad(\alpha) < n/\log^2 n$ and $Bad_k(\alpha) < p_k/\log^2 n$ for every $k \in [\frac{\log n}{\log\log n} - 2]$, Algorithm Fix returns a string $\beta \in \mathcal{L}_n$ such that $dist(\alpha,\beta) < \frac{20n}{\log n}$.*

In order to prove Lemma 4.13 we need some observations.

Fact 4.14. *For every s,t,u,k such that $part_{k+1}([s,t]) \subseteq part_k(u)$ the following is satisfied.*

- *If* $Col_k(\alpha)_u = 1$ *and* $Col_{k+1}(\alpha)_{[s,t]} = 0^{t-s+1}$ *then* $t - s \leq 5$.
- *If* $Col_k(\alpha)_u = 0$ *and* $Col_{k+1}(\alpha)_{[s,t]} = 1^{t-s+1}$ *then* $t - s \leq 5$.
- *If* $part_{k+1}([s,t]) = part_k(u)$ *and* $Col_{k+1}(\alpha)_{[s,t]} = 1^{t-s+1}$ *then* $Col_k(\alpha)_u = 1$.
- *If* $part_{k+1}([s,t]) = part_k(u)$ *and* $Col_{k+1}(\alpha)_{[s,t]} = 0^{t-s+1}$ *then* $Col_k(\alpha)_u = 0$.

Claim 4.15. *If for* $k < m$, $Col_k(\alpha)_{[w,x]}$ *and* $Col_m(\alpha)_{[y,z]}$ *are adequate good subcolorings and* $part_k([w,x]) \cap part_m([y,z]) \neq \phi$ *then* **exactly** *one of the following is true.*

1. $part_m([y,z]) \subseteq part_k([w-1,x+1])$.
2. $part_m([y,z-4] \cap part_k([w-1,x+1]) = \phi$.
3. $part_m([y+4,z] \cap part_k([w-1,x+1]) = \phi$.

Proof. Let $\alpha \in \{0,1\}^n$ be such that there exists adequate good subcolorings $Col_m(\alpha)_{[w,x]}$ and $Col_k(\alpha)_{[y,z]}$ such that $k < m$ and $part_k([w,x]) \cap part_m([y,z]) \neq \phi$.

If $m > k+1$ then case 1 is satisfied due to the upper bound on $|part_k([y,z])|$ as an adequate good subcoloring. Hence we may assume that $m = k+1$.

According to corollary 4.14 if $|part_m([w,x]) \cap part_k([y,z])| > 4n/p_m$, that is the intersection contains more than 4 parts of $\mathcal{P}_{k+1}$, then there exist $u_1, u_2 \in [w,x]$ and $v_1, v_2 \in [y,z]$ such that: $Col_k(\alpha)_{u_1} = 1$, $Col_k(\alpha)_{u_2} = 0$, $Col_m(\alpha)_{v_1} \in \{G,1\}$, $Col_m(\alpha)_{v_2} \in \{0,G\}$, $part_m(u_1) \subseteq part_k(v_1)$ and $part_m(u_2) \subseteq part_k(v_2)$. Hence in this case it can't be that $part_k(y-1) \in part_m([w,x])$ or $part_k(z+1) \in part_m([w,x])$, since this would imply that either $Col_k(\alpha)_{y-1} = 0$ or $Col_k(\alpha)_{z+1} = 1$ in contradiction to $Col_k(\alpha)_{[y,z]}$ being a good subcoloring. ■

Claim 4.16. *If* $Col_k(\alpha)_{[w,x]}$ *is an adequate good subcoloring,* $\alpha_{[y,z]}$ *is a good substring and* $part_k([w,x]) \cap [y,z] \neq \phi$ *then* **exactly** *one of the following is true.*

1. $[y,z] \subseteq part_k([w-1,x+1])$.
2. $[y,z-\log n] \cap part_k([w-1,x+1]) = \phi$.
3. $[y+\log n,z] \cap part_k([w-1,x+1]) = \phi$.

Proof. The proof is similar to the proof of claim 4.15. ■

Proof of Lemma 4.13 Let $\alpha \in \{0,1\}^n$ be such that $Bad(\alpha) < n/\log^2 n$ and $Bad_k(\alpha) < p_k/\log^2 n$ for every $k \in [2, \frac{\log n}{\log\log n} - 2]$. Let β be the string used by algorithm Fix on α.

According to definition, algorithm Fix selects intervals $[x,y] \subseteq [n]$ such that x is odd and $y-x$ is even and sets $\beta_{[x,y]} = 0^t 1^t$, where $t = (y-x)/2$. We say this change is light if $\frac{dist(\beta_{[x,y]}, \alpha_{[x,y]})}{y-x} < \frac{10}{\log n}$ and heavy otherwise. We next show that the sum over all the lengths of intervals that required a heavy change is at most $\frac{10n}{\log n}$.

We first consider phase 2 of the algorithm. In this phase the algorithm selects adequate good subcolorings and changes the corresponding substrings in β that is still in M^* to strings in $\mathcal{L}$. Note that for each k the corresponding intervals in the k'th partition are pairwise disjoint.

In the first iteration of phase 2 of the algorithm, $(k = 2)$, by definition all the changes are light. In a certain step $k = j \geq 3$, an adequate good subcoloring of the j partition may be disjoint of any

previously changed interval, or may intersect previous intervals in one of the three cases of Claim 4.15. In the case that the interval in step j is disjoint of previously changed intervals, as well as in the 2nd and 3rd cases of Claim 4.15, the changes due to the corresponding interval are light (as the subcoloring is good and there is small or none intersection with previous changes). If the first case of Claim 4.15, occurs, an interval of the j'th partition, I_j has significant intersection with an interval of a previous partition I^*. The changes due to I_j are made only in the parts that are disjoint of this intersection, and hence may be heavy. However, the corresponding length (of the parts that are changed, that is, the parts that are disjoint of the previously changed interval) is at most $\frac{2}{\log n}$-fraction of the length of I^* if I^* belongs to the $j-1$ partition, or $\frac{2}{\log^2 n}$-fraction if I^* belongs to the $j-2$ partition, and in general $\frac{2}{\log^s n}$-fraction of I^* if I^* belongs to the $j-s$ partition. Thus, if we let A_i denote the sum of all the lengths of heavy changes made during the i'th step of phase 2, B_i denote the sum of all the lengths of light changes in the $i$the step of phase 2, we get that $A_i \leq \frac{2}{\log n}(A_{i-1}+B_{i-1}) + \frac{2}{\log^2 n}(A_{i-2}+B_{i-2}) + \ldots$ which implies that the amount of heavy changes during phase 2 of the algorithm is bounded by $\sum A_i \leq 4n/\log n$.

By claim 4.16 and reasoning similar to the above the sum of heavy changes in phase 3 is at most $\frac{2n}{\log n}$.

In phase 4 of the algorithm, every substring that is changed is contained in an interval $[x,y]$ such that $Col_k(\alpha)_{[x,y]}$ is a bad subcoloring for some $k \in [2, \frac{\log n}{\log\log n} - 2]$. By the assumption on the lengths of such subcolorings, the sum of the intervals changed in this phase is less than $\frac{n}{2\log n}$.

In phase 5, for every substring in β that is still untouched, we change it to be in $\mathcal{L}$. Note that every such string must be part of a bad substring. All those strings that are part of short bad substrings contribute at most $n/\log^2 n$ changes (in particular the heavy changes).

For each bad substring $\alpha_{[x,y]}$ such that $y - x > \log^3 n$ (that is, a bad but not a short bad substring), there is an adequate subcoloring $Col_k(\alpha)_{[w,z]}$ such that $[x,y] \subseteq part_k(w-1, z+1)$. Thus, the changes in the substring $\alpha_{[x,y]}$ in phase 5, contributes at most $\frac{2}{\log n}$ of the length of $|part_k(k-1, z+1)|$. In particular, the heavy changes in this phase are bounded by $\frac{2n}{\log n}$.

Thus the sum over all heavy changes is less than $\frac{10n}{\log n}$. By definition, light changes contribute a total of at most $\frac{10n}{\log n}$. ■

References

[1] M. Luby M. Blum and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, (47):549–595, 1993.

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

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

[4] D. Ron. *Property testing (a tutorial)*. Kluwer Press, 2001.

[5] E. Fischer. The art of uninformed decisions: A primer to property testing. *The computational complexity column of The Bulletin of the European Association for Theoretical Computer Science*, (75):97–126, 2001.

[6] N. Alon M. Krivelevich I. Newman and M. Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, (30):1842–1862, 2001.

[7] M. Parnas D. Ron and R. Rubinfeld. *Testing Parenthesis Languages*, volume 2129. Springer-Verlag Heidelberg, 2001.

[8] M. Paterson L.G. Valiant. Deterministic one-counter automata. *Jornal of Computer and System Science*, (10):340–350, 1975.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'