Testing Membership in Parenthesis Languages

Michal Parnas
The Academic College
of Tel-Aviv-Yaffo
Tel-Aviv, ISRAEL
michalp@mta.ac.il

Dana Ron*
Department of EE – Systems
Tel-Aviv University
Ramat Aviv, ISRAEL
danar@eng.tau.ac.il

Ronitt Rubinfeld
NEC Research Institute
Princeton, NJ
ronitt@research.nj.nec.com

June 16, 2003

Abstract

We continue the investigation of properties defined by formal languages. This study was initiated by Alon et. al. [AKNS00] who described an algorithm for testing properties defined by regular languages. Alon et. al. also considered several context free languages, and in particular Dyck languages, which contain strings of properly balanced parentheses. They showed that the first Dyck language, which contains strings over a single type of pairs of parentheses, is testable in time independent of n, where n is the length of the input string. However, the second Dyck language, defined over two types of parentheses, requires $\Omega(\log n)$ queries.

Here we describe a sublinear-time algorithm for testing all Dyck languages. Specifically, the running time of our algorithm is $\tilde{O}(n^{2/3}/\epsilon^3)$, where ϵ is the given distance parameter. Furthermore, we improve the lower bound for testing Dyck languages to $\tilde{\Omega}(n^{1/11})$ for constant ϵ. We also describe a testing algorithm for the context free language $L_{\text{REV}} = \{uu^r vv^r : u, v \in \Sigma^*\}$, where Σ is a fixed alphabet. The running time of our algorithm is $\tilde{O}(\sqrt{n}/\epsilon)$, which almost matches the lower bound given by Alon et. al. [AKNS00].

1 Introduction

Property testing [RS96, GGR98] is a relaxation of the standard notion of a decision problem: property testing algorithms distinguish between inputs that have a certain property and those that are *far* from having the property. More precisely, for any fixed property $\mathcal{P}$, a testing algorithm for $\mathcal{P}$ is given query access to the input and a distance parameter ϵ. The algorithm should output "accept" with high probability if the input has the property $\mathcal{P}$, and output "reject" if the input is ϵ-far from having $\mathcal{P}$. By ϵ-far we mean that more than an ϵ–fraction of the input should be modified so that the input obtains the desired property $\mathcal{P}$.

Testing algorithms whose query complexity is sublinear and even independent of the input size, have been designed for testing various algebraic and combinatorial properties (see [Ron01] for a survey).

*Supported by the Israel Science Foundation (grant number 32/00-1).

Motivated by the desire to understand in what sense the complexity of testing properties of strings is related to the complexity of formal languages, Alon et. al. [AKNS00], have shown that all properties defined by regular languages are testable in time that is independent of the input size. Specifically, given a regular language L, they describe an algorithm that tests, using $\tilde{O}(1/\epsilon)$ queries, whether a given string s belongs to L or is ϵ-far from any string in L. This result was later extended by Newman [New00] to properties defined by bounded-width branching programs. However, Alon et. al. [AKNS00] showed that the situation changes quite dramatically for context-free languages. In particular, they prove that there are context-free languages that are not testable even in time square root in the input size. The question remains whether context-free languages can be tested in sublinear time. In this paper, we give evidence for an affirmative answer by presenting sublinear time testers for certain important subclasses of the context-free languages.

Dyck Languages. One important subclass of the context-free languages is the Dyck language, which includes strings of properly balanced parentheses. Strings such as "(( )( ))" belong to this class, whereas strings such as "(( )" or ") (" do not. If we allow more than one type of parentheses then "([ ])" is a balanced string but "([ )]" is not. Formally, the Dyck language D_m contains all balanced strings that contain at most m types of parentheses. Thus for example "(( )( ))" belongs to D_1 and "([ ])" belongs to D_2.

Dyck languages appear in many contexts. For example, these languages describe a property that should be held by commands in most commonly used programming languages, as well as various subsets of the symbols/commands used in latex. Furthermore, Dyck languages play an important role in the theory of context-free languages. As stated by the Chomsky-Schötzenberger Theorem, every context-free language can be mapped to a restricted subset of D_m [Sch63]. A comprehensive discussion of context free languages and Dyck languages can be found in [Har78, Koz97].

Thus testing membership in D_m is a basic and important problem. Alon et. al. [AKNS00], have shown that membership in D_1 can be tested in time $\tilde{O}(1/\epsilon)$, whereas membership in D_2 cannot be tested in less than a logarithmic time in the length n of the string.

Our Results.

- We present an algorithm that tests whether a string s belongs to D_m. The query complexity and running time of the algorithm are $\tilde{O}\left(n^{2/3}/\epsilon^3\right)$, where n is the length of s. The complexity does not depend on m, the number of different types of parentheses.
- We prove a lower bound of $\Omega(n^{1/11}/\log n)$ on the query complexity of any algorithm for testing D_m for $m > 1$.
- We consider the context free language $L_{\mathrm{REV}} = \{uu^r vv^r : u, v \in \Sigma^*\}$, where Σ is any fixed alphabet and u^r denotes the string u in reverse order. We show that L_{REV} can be tested in $\tilde{O}(\sqrt{n}/\epsilon)$ time, where n is the length of the string. Our algorithm almost matches the $\Omega(\sqrt{n})$ lower bound of Alon et. al [AKNS00] on the number of queries required for testing L_{REV}.

The structure of our testing algorithm for D_m. Our testing algorithm for D_m combines *local* checks with *global* checks. Specifically, the first part of the test randomly selects consecutive substrings of the given input string, and checks that they do not constitute a witness to the string not belonging to D_m. The second, more elaborate part of the test, verifies that non-consecutive pairs of substrings that are supposed to contain matching parentheses, in fact do. In particular, the string is partitioned into fixed *blocks* (consecutive substrings), and the algorithm computes various statistics concerning the numbers of opening and closing parentheses in the different blocks. Using these statistics it is possible to determine which pairs of blocks should contain many matching parentheses in case that the string in fact belongs to D_m. The testing algorithm then randomly selects such pairs of blocks and verifies the existence of such a matching between opening parentheses in one block and closing parentheses in the other block.

Organization. In Section 2 we describe the necessary preliminaries. Our testing algorithm for Dyck Languages is presented in Section 3, and the lower bound in Section 4. The testing algorithm for L_{REV} appears in Section 5.

2 Preliminaries

Let $s = s_1 \ldots s_n$ be a string over an alphabet $\Sigma_m = \{0, \ldots, 2m-1\}$ where $2i, 2i+1$ correspond to the i^{th} type of opening and closing parentheses. We will use the following notation for strings and substrings.

Definition 1 (Substrings) *For a string $s = s_1 \ldots s_n$ and $i \leq j$, we let $s_{i,j}$ denote the substring $s_i, s_{i+1}, ..., s_j$. If s', s'' are two strings, then $s's''$ denotes the concatenation of the two strings.*

Definition 2 (Dyck Language) *The Dyck language D_m can be defined recursively as follows:*

1. *The empty string belongs to D_m.*
2. *If $s' \in D_m$, $\sigma = 2i$ is an opening parenthesis and $\tau = 2i+1$ is a matching closing parenthesis, for some $0 \leq i \leq m-1$, then $\sigma s' \tau \in D_m$.*
3. *If $s', s'' \in D_m$, then $s's'' \in D_m$.*

It is clear from the recursive definition of D_m that the parentheses in a string s have a nested structure and are balanced. The first step of our algorithm will test if the string s is a legal string when we view it as a string in D_1, using the test given by [AKNS00]. Furthermore, the algorithm will test if consecutive substrings of s can be extended to a legal string in D_m. The following definitions address these aspects formally.

Definition 3 (Single-Parentheses Mapping) *Given a string s over Σ_m, we define a mapping μ which maps s to a string $\mu(s)$ over $\Sigma_1 = \{0, 1\}$ as follows: Every opening parenthesis is mapped to 0, and every closing parentheses is mapped to 1.*

The following claim is immediate from the definition of D_1.

Claim 1 *For every string s such that $\mu(s) \in D_1$, there exists a unique perfect matching between opening and closing parentheses in s, such that each opening parenthesis s_j is matched to a closing parenthesis s_k, and no two matched pairs "cross". That is, if s_{j_1} is matched to s_{k_1}, and s_{j_2} to s_{k_2} where $j_1 < k_1$, and $j_1 < j_2 < k_2$, then either $k_1 < j_2$ or $k_1 > k_2$. We denote this unique perfect matching by $M(s)$.*

Thus for example if $s =$ "([ )]", so that $s \notin D_2$ but $\mu(s) \in D_1$, then $M(s) = \{(s_1, s_4), (s_2, s_3)\}$.

Definition 4 (Consistency of Substrings) *We say that a substring s' over Σ_m is D_m-*Consistent*, if there exists a string $s \in D_m$ such that s' is a consecutive substring of s.*

Thus for example $s' =$ ")[ ])[" is D_2-consistent because $s =$ "((s'] = "(( )[ ])[ ]" $\in D_2$, while $s' =$ ")[ ))[" is not D_2-consistent.

The second part of our algorithm finds disjoint pairs of substrings such that there exist opening parentheses in the first substring that should be matched to closing parentheses in the second substring. The algorithm verifies that these pairs of parentheses match in type as required. The following concepts will be needed for this part of the algorithm.

Definition 5 (Parentheses Numbers) *For any substring s' of s, define*

$$n_0(s') \overset{\text{def}}{=} \textit{Number of opening parentheses in } s' ,$$

and

$$n_1(s') \overset{\text{def}}{=} \textit{Number of closing parentheses in } s' .$$

Fact 2 *A string s belongs to D_1 if and only if: (1) For every prefix s' of s, $n_0(s') \geq n_1(s')$; (2) $n_0(s) = n_1(s)$.*

The above fact implies that any string s' over $\Sigma_1 = \{0,1\}$ is D_1-consistent, since for such a string there exist integers k and ℓ such that $0^k s' 1^\ell \in D_1$. In this case we can view s' as having an *excess* of k closing parentheses and ℓ opening parentheses, assuming k and ℓ are the smallest integers such that $0^k s' 1^\ell \in D_1$. The following definition extends this notion of excess parentheses in a substring to any alphabet Σ_m.

Definition 6 (Excess numbers) *Let s' be a substring over Σ_m, and let k and ℓ be the smallest integers such that $0^k\,\mu(s')1^\ell \in D_1$. Then k is called the* excess number *of closing parentheses in s', and ℓ is the excess number of opening parentheses in s'. Denote k by $e_1(s')$ and ℓ by $e_0(s')$.*

For example if $s' =$ "][( )])(", then $e_1(s') = 2$ and $e_0(s') = 1$. It is possible to compute the excess numbers from the parentheses numbers as follows.

Claim 3 *The following two equalities hold for every substring s',*

$$e_1(s') = \max_{s'' \textit{prefix of } s'} (n_1(s'') - n_0(s'')) \tag{1}$$

$$e_0(s') = \max_{s'' \textit{suffix of } s'} (n_0(s'') - n_1(s'')) \tag{2}$$

In both cases the maximum is also over the empty prefix (suffix) s'', for which $n_1(s'') - n_0(s'') = 0$.

3 The Algorithm for Testing D_m

In the following subsections we describe several building blocks of our algorithm. Recall that the algorithm has two main parts. Let s be a string over Σ_m. First we test that $\mu(s) \in D_1$. For simplicity of this introductory discussion, assume that if $\mu(s)$ passes this test, then $\mu(s)$ actually belongs to D_1 (and is not only close to a string in D_1). Next we test that consecutive substrings of s are D_m-consistent. In the next stage we estimate the excess numbers for substrings of s. Using these estimates we find pairs of substrings that contain a significant number of matched pairs of parentheses according to the perfect matching $M(s)$ guaranteed by Claim 1, and check that these pairs match in type.

To do the latter, we break the string into $n^{1/3}$ substrings each of length $n^{2/3}$, which we refer to as *blocks*. We define a weighted graph, whose vertices correspond to these blocks, and in which there is an edge between block i and block $j > i$ if and only if the matching $M(s)$ matches between excess opening parentheses in block i to excess closing parentheses in block j. The weight of each edge is simply the number of corresponding matched pairs of excess parentheses. As we show subsequently, this weight can be determined by the values of the excess numbers for every consecutive sequence of blocks. Hence, if we were provided with these *exact* values, we could verify, for randomly selected pairs of blocks that are connected by an edge in the graph, whether their excess parentheses match as required. Since we do not have these exact values, but rather approximate values, we use our estimates of the excess values to construct an approximation of the above graph, and to perform the above verification of matching excess parentheses.

3.1 Checking D_m-Consistency

It is well known that it is possible to check in time $O(n)$ using a stack whether a string s over Σ_m whose length is n belongs to D_m. This is done as follows: The symbols of s are read one by one. If the current symbol read is an opening parenthesis then it is pushed onto the stack. If it is a closing parenthesis, then the top symbol on the stack is popped and compared to the current symbol. The algorithm rejects if the symbol popped (which must be an opening parenthesis) does not match the current symbol. The algorithm also rejects if the stack is empty when trying to pop a symbol, that is, there is a missing matching symbol, or if the stack is not empty after reading all symbols. Otherwise the algorithm accepts. The above algorithm can be easily modified to check whether a substring s' is D_m-consistent. The only two differences are: (1) When reading a closing parenthesis and finding that the stack is empty, the algorithm does not reject but rather continues with the next symbol. (2) If the algorithm has completed reading the string without finding a mismatched pair of parentheses, then it accepts even if the stack is not empty. Thus the algorithm rejects only if it finds a mismatch in the type of parentheses.

3.2 A Preprocessing Stage

An important component of our algorithm is acquiring good estimates of the excess numbers of different substrings of the given input string s. We start by describing a preprocessing step based on which we can obtain such estimates for a fixed set of *basic* substrings of s (of various lengths). By sampling from such a substring s', we obtain estimates of the parentheses numbers $n_0(s')$ and $n_1(s')$. Using these estimates we can derive estimates for the excess numbers of any given substring of s.

Let $r = \log(n^{1/3}/\delta)$, where $0 < \delta < 1$ is a parameter that is set subsequently. For each $j \in \{0, 1, \ldots, r\}$, we consider the partition of s into 2^j consecutive substrings each of length $n/2^j$. We assume for simplicity that n is divisible by $2^r = n^{1/3}/\delta$. Thus the total number of substrings is $O(n^{1/3}/\delta)$, where the longest is the whole string s, and the shortest ones are of length $\delta \cdot n^{2/3}$. We refer to these substrings as the *basic substrings* of s.

For each basic substring s' of length $n/2^j$, we uniformly and independently select a sample of m_j symbols from s', where

$$m_j = c \cdot \frac{n^{2/3}}{2^{2j}} \cdot \frac{\log^3(n/\delta)}{\delta^2},$$

for some sufficiently large constant c. Let m_j^0 be the number of opening parentheses in the sample, and m_j^1 be the number of closing parentheses in the sample. Our estimates of the number of opening and closing parentheses in s' are respectively:

$$\hat{n}_0(s') = \frac{m_j^0}{m_j} \cdot |s'| = \frac{m_j^0}{m_j} \cdot \frac{n}{2^j} \qquad \text{and} \qquad \hat{n}_1(s') = \frac{m_j^1}{m_j} \cdot \frac{n}{2^j}.$$

Lemma 4 *With probability at least $1 - o(1)$, for each of the basic substrings $s' \subseteq s$, $|\hat{n}_0(s') - n_0(s')| \leq (\delta \cdot n^{2/3})/(24 \log(n/\delta))$ and $|\hat{n}_1(s') - n_1(s')| \leq (\delta \cdot n^{2/3})/(24 \log(n/\delta))$. The total size of the sample is $O\left(n^{2/3} \cdot \log^3(n/\delta)/\delta^2\right)$.*

Proof: We prove the bound for $\hat{n}_0(s')$. The bound for $\hat{n}_1(s')$ directly follows since $n_0(s') + n_1(s') = |s'| = \hat{n}_0(s') + \hat{n}_1(s')$. By an additive Chernoff bound, for any fixed substring s' of length $n/2^j$, and for any given $0 < \gamma_j < 1$,

$$\Pr\left[|\hat{n}_0(s') - n_0(s')| > \gamma_j \cdot \frac{n}{2^j}\right] < 2\exp(-2m_j\gamma_j^2). \tag{3}$$

Setting $\gamma_j = \frac{2^j}{n^{1/3}} \cdot \frac{\delta}{24 \log(n/\delta)}$ so that $\gamma_j \cdot \frac{n}{2^j} = \frac{\delta}{24 \log(n/\delta)} \cdot n^{2/3}$, for any such string the probability that $|\hat{n}_0(s') - n_0(s')| > \frac{\delta}{24 \log(n/\delta)} \cdot n^{2/3}$ is at most $2\exp(-2m_j\gamma_j^2) = 1/\text{poly}(n/\delta)$. Since the total number of basic substrings considered is $O(n^{1/3}/\delta)$, all estimates are within the stated error.

Since there are 2^j basic substrings of length $n/2^j$, then the total size of the sample is:

$$\sum_{j=0}^{r} 2^j \cdot m_j = \frac{c \cdot n^{2/3} \cdot \log^3(n/\delta)}{\delta^2} \cdot \sum_{j=0}^{r} \frac{1}{2^j} = O\left(\frac{n^{2/3} \cdot \log^3(n/\delta)}{\delta^2}\right)$$

■

We assume from now on that the quality of our estimates $\hat{n}_0(s')$ and $\hat{n}_1(s')$ is in fact as stated in the lemma for every basic substring s'. We refer to this as the *successful preprocessing assumption.*

Assumption 5 (Successful Preprocessing Assumption) *For each of the basic substrings s', $|\hat{n}_0(s') - n_0(s')| \leq (\delta \cdot n^{2/3})/(24 \log(n/\delta))$, and the same bound holds for $\hat{n}_1(s')$.*

3.3 Obtaining Estimates of Excess numbers

We first consider obtaining estimates for $n_0(s')$ and $n_1(s')$ for substrings s' of s of the form defined in the next claim.

Claim 6 *Let $s' = s_{k,\ell}$ be any substring of s such that $k = t_1 \cdot (\delta \cdot n^{2/3}) + 1$ and $\ell = t_2 \cdot (\delta \cdot n^{2/3})$, for $0 \leq t_1 < t_2 \leq n^{1/3}/\delta$. Then s' is the concatenation of at most $2 \log |s'| + 1$ basic substrings.*

Proof: Let s^b be the longest basic substring such that $s^b \subseteq s'$. Therefore s' is of the form $s' = s^{left}s^bs^{right}$. Consider the substring s^{right}. It is not hard to verify that if we partition s^{right} into basic substrings starting from left to right, and each time choose the longest possible basic substring that is contained in s^{right}, then each such basic substring will cover at least $1/2$ of what remains of s^{right}. Thus the total number of basic substrings needed to cover s^{right} is at most $\log |s'|$. Similarly, the total number of basic substrings needed to cover s^{left} is at most $\log |s'|$ as well. It directly follows that s' is a concatenation of at most $2 \log |s'| + 1$ basic substrings. See Figure 1 for an illustration of the proof. ■

Assume that the substring s' is the concatenation of the basic substrings $s^1, ..., s^t$. Then we can estimate $n_0(s')$ by $\hat{n}_0(s') = \sum_{i=1}^{t} \hat{n}_0(s^i)$, where $\hat{n}_0(s^i)$ is the estimate we got above for the basic substring s^i. Similarly, we can estimate $n_1(s')$.

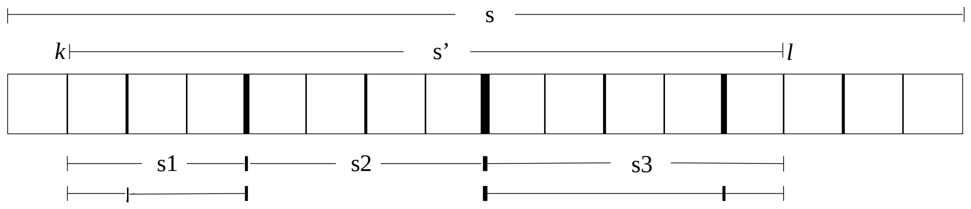

Figure 1: An illustration for Claim 6. The partition of the string s into basic substrings is represented by separating lines with different thickness. Specifically, the middle, thickest line represents the partition of s into the two basic substrings of length $n/2$, the two less thicker lines represent the next partition into four substrings of length $n/4$ and so on. Each square corresponds to a substring of length $\delta \cdot n^{2/3}$, where s is the concatenation of all these substrings (smallest basic substrings). The substring s' starts at index k and ends at index ℓ, where both are multiples of $\delta \cdot n^{2/3}$. In the illustration, $s2 = s^b$, $s1 = s^{left}$ and $s3 = s^{right}$ (note that there were two possible choices for the largest basic substring s^b that is contained in s'). The substrings $s1$ and $s2$ are further partitioned each into two basic substrings, the first of length at least half of $s1$ (respectively, $s2$).

Corollary 7 *Under Assumption 5, for every substring s' as in Claim 6, $|\hat{n}_0(s') - n_0(s')| < \delta \cdot n^{2/3}/4$ and $|\hat{n}_1(s') - n_1(s')| < \delta \cdot n^{2/3}/4$.*

We next consider how to obtain estimates for the excess number of opening parentheses of a given substring $s' = s_{k,\ell}$ (where k and ℓ are assumed to be as in Claim 6), and similarly for the excess number of closing parentheses. To this end we appeal to Claim 3, and use our estimates for the total number of opening and closing parentheses in certain prefixes and suffixes of s'. As we show below, for the purpose of getting an additive estimate of the excess to within $\delta \cdot n^{2/3}$ for any substring, it is enough to use estimates of n_0 and n_1 for prefixes and suffixes of the substring that are multiples of $\delta \cdot n^{2/3}$. Specifically,

Claim 8 *Let $s' = s_{k,\ell}$ be as in Claim 6, and define two sets*

$$\textit{Prefix} = \{s'' | s'' = s_{k,\ell'},\ \ell' = t_2' \cdot (\delta \cdot n^{2/3}) + 1,\ t_1 \le t_2' < t_2\}$$

$$\textit{Suffix} = \{s'' | s'' = s_{k',\ell},\ k' = t_1' \cdot (\delta \cdot n^{2/3}) + 1,\ t_1 < t_1' \le t_2\}.$$

Let

$$\hat{e}_0(s') = \max_{s'' \in \textit{Suffix}} (\hat{n}_0(s'') - \hat{n}_1(s'')), \qquad \hat{e}_1(s') = \max_{s'' \in \textit{Prefix}} (\hat{n}_1(s'') - \hat{n}_0(s'')).$$

Then, under Assumption 5, $|\hat{e}_0(s') - e_0(s')| \le \delta \cdot n^{2/3}$ and $|\hat{e}_1(s') - e_1(s')| \le \delta \cdot n^{2/3}$.

Proof: We prove the claim for $\hat{e}_0(s')$. The proof for $\hat{e}_1(s')$ is analogous. Let $s'' = s_{b,\ell}$ be the suffix of s' for which the maximum is obtained in Equation (2) of Claim 3. (Recall that $s_{b,\ell}$ may be the empty string in which case $b = \ell + 1$.) Let b' be the index closest to b of the form $b' = t_1' \cdot (\delta \cdot n^{2/3}) + 1$, where $t_1 < t_1' \le t_2$. Since by definition of b' we have $|b' - b| \le \delta \cdot n^{2/3}/2$, we know that $n_0(s_{b',\ell}) - n_1(s_{b',\ell}) \ge n_0(s_{b,\ell}) - n_1(s_{b,\ell}) - \delta \cdot n^{2/3}/2$. But by definition of $s_{b,\ell}$, $n_0(s_{b,\ell}) - n_1(s_{b,\ell}) = e_0(s')$, and so

$$n_0(s_{b',\ell}) - n_1(s_{b',\ell}) \ge e_0(s') - \delta \cdot n^{2/3}/2.$$

By Corollary 7, $|\hat{n}_0(s_{b',\ell}) - n_0(s_{b',\ell})| \le \delta \cdot n^{2/3}/4$, and $|\hat{n}_1(s_{b',\ell}) - n_1(s_{b',\ell})| \le \delta \cdot n^{2/3}/4$. Hence, $\hat{n}_0(s_{b',\ell}) - \hat{n}_1(s_{b',\ell}) \ge e_0(s') - \delta \cdot n^{2/3}$. But by definition, $\hat{e}_0(s') \ge \hat{n}_0(s_{b',\ell}) - \hat{n}_1(s_{b',\ell})$, and the claim follows. ■

3.4 The Matching Graph

Before defining the matching graph, we extend the notion of the perfect matching $M(s)$ guaranteed by Claim 1, to strings s for which $\mu(s) \notin D_1$. In this case we do not obtain a *perfect* matching, but rather a matching of all the parentheses in the string that are not excess parentheses with respect to the whole string. Specifically, by definition of the excess numbers, the string $\tilde{s} = 0^{e_1(s)}\mu(s)1^{e_0(s)}$ belongs to D_1. Thus we let $M(s)$ be the restriction of $M(\tilde{s})$ to pairs of parentheses that are *both in* s. For example, if $s =$ "( ]] ([ )", then $M(s)$ matches between s_1 and s_2 and between s_5 and s_6.

In all that follows we assume that $n^{2/3}$ is an even integer. It is not hard to verify that this can be done without loss of generality. We partition the given string s into $n^{1/3}$ consecutive and disjoint substrings, each of length $n^{2/3}$, which we refer to as *blocks*.

Definition 7 (Neighbor Blocks) *We say that two blocks i and j are* neighbors *in a string s, if the matching $M(s)$ matches between excess opening parentheses in block i and excess closing parentheses in block j.*

Definition 8 (The Matching Graph of a String) *Given a string s, we define a weighted graph as follows. The vertices of the graph are the $n^{1/3}$ blocks of s. Two blocks $i < j$ are connected by an edge (i, j) if and only if they are neighbor blocks (as defined above). The weight $w(i, j)$ of the edge (i, j) is the number of excess opening parentheses in block i that are matched by $M(s)$ to excess closing parentheses in block j. The resulting graph is called the* matching graph *of s, and is denoted by $G(s)$. The set of edges of the graph is denoted by $E(G(s))$.*

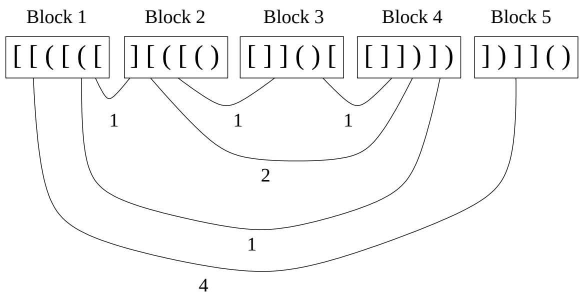

Figure 2: An example of the matching graph of a string in D_2. The string consists of 5 blocks outlined by rectangles, with 6 symbols in each block. The numbers below the edges are the weights of the edges.

Note that since we extended the matching $M(s)$ also to strings s such that $\mu(s) \notin D_1$, then the graph $G(s)$ is well defined also for such strings. By the properties of the matching $M(s)$ which guarantees that matched pairs do not "cross", we get:

Claim 9 *For every string s, the matching graph $G(s)$ is planar, and therefore $|E(G(s))| \leq 3n^{1/3}$.*

It is possible to determine which blocks are neighbors in $G(s)$, and what is the weight of the edge between them, using the excess numbers e_1 and e_0 as follows. We first introduce one more definition.

Definition 9 (Intervals) *For a given string s, let $I_{i,j}$ denote the substring, which we refer to as* interval*, that starts at block i and ends at block j (including both of them). If $j < i$ then $I_{i,j}$ is the empty interval.*

Note that $I_{i,i}$ is just block number i.

Claim 10 *Let s be a given string and let $i < j$ be two blocks in $G(s)$. Define:*

$$x(i,j) \stackrel{\text{def}}{=} \min\{e_1(I_{i+1,j}), e_0(I_{i,i})\} - e_1(I_{i+1,j-1}) \ , \tag{4}$$

where if $I_{i+1,j-1}$ is empty (that is, $j = i+1$), then $e_1(I_{i+1,j-1}) = 0$. Then we have the following: (1) If $x(i,j) > 0$ then i and j are neighbors in $G(s)$. (2) If i and j are neighbors in $G(s)$ then $w(i,j) = x(i,j)$.

Proof: We first observe that for both parts of the claim the premise implies that $e_0(I_{i,i}) > 0$. That is, the i'th block has excess opening parentheses. We next observe that the sequence $e_1(I_{i+1,j})$ is monotonically non-decreasing with j. In particular, $e_1(I_{i+1,j}) \geq e_1(I_{i+1,j-1})$.

Item (1): Suppose that $x(i,j) > 0$. By definition of $x(i,j)$ we have that both $e_0(I_{i,i}) - e_1(I_{i+1,j-1}) > 0$ and $e_1(I_{i+1,j}) - e_1(I_{i+1,j-1}) > 0$ (since $\min\{e_0(I_{i,i}), e_1(I_{i+1,j})\} - e_1(I_{i+1,j-1}) = x(i,j) > 0$). The first inequality implies that there are excess *opening* parentheses in block i that are not matched to closing parentheses in the interval $I_{i+1,j-1}$. The second inequality implies that there are excess *closing* parentheses in block j that are not matched to opening parentheses in the interval $I_{i+1,j-1}$. It follows that there must be excess opening parentheses in block i that are matched to excess closing parentheses in block j, and so i and j are neighbors in $G(s)$ as claimed.

Item (2): Suppose that i and j are neighbors in $G(s)$. We first observe that all $e_1(I_{i+1,j-1})$ excess closing parentheses in the interval $I_{i+1,j-1}$ are matched to excess opening parentheses in block i, and that there are $e_0(I_{i,i}) - e_1(I_{i+1,j-1}) > 0$ additional excess opening parentheses in block i. We consider two subcases (for an illustration see Figure 3):

If $e_0(I_{i,i}) \leq e_1(I_{i+1,j})$, so that $x(i,j) = e_0(I_{i,i}) - e_1(I_{i+1,j-1})$, then all excess opening parentheses in block i are matched to closing parentheses in the interval $I_{i+1,j}$, that is, to closing parentheses in blocks $i+1$ to j. By what we have observed above, $e_1(I_{i+1,j-1})$ of them are matched to parentheses in blocks $i+1$ to $j-1$, and all the remaining $e_0(I_{i,i}) - e_1(I_{i+1,j-1}) = x(i,j)$ excess opening parentheses in block i are matched to closing parentheses in block j. Thus we have verified the second item in case that $e_0(I_{i,i}) \leq e_1(I_{i+1,j})$.

If on the other hand $e_0(I_{i,i}) > e_1(I_{i+1,j})$, so that $x(i,j) = e_1(I_{i+1,j}) - e_1(I_{i+1,j-1})$, then all excess closing parentheses in the interval $I_{i+1,j}$ are matched to excess opening parentheses in block i. Amongst them $e_1(I_{i+1,j-1})$ are from blocks $i+1$ to $j-1$, and all the remaining $e_1(I_{i+1,j}) - e_1(I_{i+1,j-1}) = x(i,j)$ are from block j. Thus we have verified the second item in case that $e_0(I_{i,i}) > e_1(I_{i+1,j})$. ■

It is not hard to verify that based on the symmetry of the matching, if i and j are neighbors in the graph $G(s)$ then also $w(i,j) = \min\{e_0(I_{i,j-1}), e_1(I_{j,j})\} - e_0(I_{i+1,j-1})$. Recall that if $\mu(s) \in D_1$ then by Claim 1 the matching $M(s)$ is a perfect matching between opening and closing parentheses in s. In particular it contains all parentheses that are excess parentheses in the $n^{1/3}$ blocks of s. We thus obtain:

Corollary 11 *Let s be a string such that $\mu(s) \in D_1$. Then,*

$$\sum_{(k,\ell)\in E(G(s)),\ k<\ell} w(k,\ell) = \frac{1}{2}\sum_{i=1}^{n^{1/3}} \left(e_0(I_{i,i}) + e_1(I_{i,i})\right) .$$

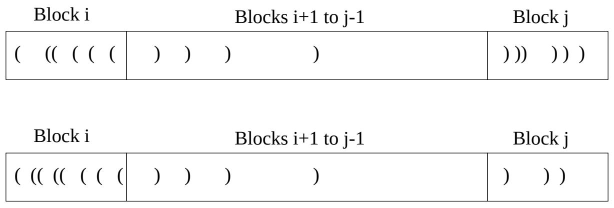

Figure 3: An illustration for Claim 10. The parentheses shown in the figure (all of the same type for simplicity) are all excess parentheses. Specifically, in the top figure $e_0(I_{i,i}) = 6$, $e_1(I_{i+1,j-1}) = 4$ and $e_1(I_{i+1,j}) = 10$. In the bottom figure $e_0(I_{i,i}) = 8$, $e_1(I_{i+1,j-1}) = 4$ and $e_1(I_{i+1,j}) = 7$. In both figures the 4 excess closing parentheses in the interval $I_{i+1,j-1}$ are all matched to the last 4 excess parentheses in block i. The top figure corresponds to the case that $e_0(I_{i,i}) \le e_1(I_{i+1,j})$, so that $x(i,j) = e_0(I_{i,i}) - e_1(I_{i+1,j-1}) = 2$. In this case the remaining 2 excess opening parentheses from block i are matched to the first 2 excess closing parentheses in block j (where these are excess parentheses with respect to all the interval $I_{i+1,j}$). The bottom figure corresponds to the case that $e_0(I_{i,i}) > e_1(I_{i+1,j})$, so that $x(i,j) = e_1(I_{i+1,j}) - e_1(I_{i+1,j-1}) = 3$, and in fact all 3 excess closing parentheses in block j are matched to excess opening parentheses in block i.

We next turn to the case in which we only have estimates of the excess numbers. Here we define a graph based on the estimates we have for the excess numbers. This graph contains only relatively "heavy" edges in order to overcome approximation errors.

Definition 10 (The Approximate Matching Graph) *Given a string s, we partition it into blocks of size $n^{2/3}$, and define a graph $\hat{G}(s)$ whose vertices are the $n^{1/3}$ blocks of s. A pair of blocks $i < j$ will be connected by an edge (i,j) if and only if* $\min\{\hat{e}_1(I_{i+1,j}), e_0(I_{i,i})\} - \hat{e}_1(I_{i+1,j-1}) \ge 4\delta n^{2/3}$*, where $\hat{e}_0$ and $\hat{e}_1$ are as defined in Claim 8, and if $I_{i+1,j-1}$ is empty then $\hat{e}_1(I_{i+1,j-1}) = 0$.*

Note that in the definition above we use the exact value $e_0(I_{i,i})$, as opposed to the approximate values $\hat{e}_1(I_{i+1,j})$. This is done because our testing algorithm will have time to compute the value $e_0(I_{i,i})$ exactly for certain blocks, where as it will only approximate the values $\hat{e}_1(I_{i+1,j})$. The following lemma is central to our algorithm and its analysis.

Lemma 12 *Suppose Assumption 5 holds. Then for any given string s, the graph $\hat{G}(s)$ is a subgraph of $G(s)$, and every vertex in $\hat{G}(s)$ has degree at most $1/(2\delta)$. Furthermore, if $\mu(s)$ is δ-close to a string in D_1, then $\hat{G}(s)$ "accounts for most of the excess" in s. Namely,*

$$\sum_{(k,\ell)\in E(\hat{G}(s)),\ k<\ell} x(k,\ell) \ge \frac{1}{2}\sum_{i=1}^{n^{1/3}} (e_0(I_{i,i}) + e_1(I_{i,i})) \quad - 19\delta n.$$

Proof: By Claim 8, for every interval $I_{i,j}$ of the string s, $|\hat{e}_0(I_{i,j}) - e_0(I_{i,j})| \le \delta \cdot n^{2/3}$, and $|\hat{e}_1(I_{i,j}) - e_1(I_{i,j})| \le \delta \cdot n^{2/3}$. By definition of $\hat{G}(s)$, for every edge $(i,j) \in E(\hat{G}(s))$, we have $\min\{\hat{e}_1(I_{i+1,j}), e_0(I_{i,i})\} - \hat{e}_1(I_{i+1,j-1}) \ge 4\delta n^{2/3}$. Therefore

$$x(i,j) = \min\{e_1(I_{i+1,j}), e_0(I_{i,i})\} - e_1(I_{i+1,j-1}) \ge 4\delta n^{2/3} - 2\delta n^{2/3} = 2\delta n^{2/3}.$$

By Claim 10, this implies that there exists an edge $(i,j) \in E(G(s))$, and that this edge has weight $x(i,j) \ge 2\delta n^{2/3}$. Since this is true for every edge $(i,j) \in E(\hat{G}(s))$, we get that $\hat{G}(s)$ is a subgraph of $G(s)$, and every vertex in $\hat{G}(s)$ has degree at most $n^{2/3}/(2\delta n^{2/3}) = 1/(2\delta)$.

On the other hand, for every edge $(i,j) \in E(G(s))$ such that $w(i,j) = x(i,j) \ge 6\delta n^{2/3}$, we have that

$$\min\{\hat{e}_1(I_{i+1,j}), e_0(I_{i,i})\} - \hat{e}_1(I_{i+1,j-1}) \;\ge\; x(i,j) - 2\delta n^{2/3} \;\ge\; 4\delta n^{2/3}$$

and so (i,j) is an edge in $\hat{G}(s)$ as well. Since $G(s)$ is planar, the total weight of edges $(i,j) \in E(G(s))$ such that $w(i,j) < 6\delta n^{2/3}$, is at most $3n^{1/3} \cdot 6\delta n^{2/3} = 18\delta n$. Hence,

$$\sum_{(k,\ell)\in E(\hat{G}(s)),\, k<\ell} x(k,\ell) \geq \left(\sum_{(k,\ell)\in E(G(s)),\, k<\ell} x(k,\ell) \right) - 18\delta n. \tag{5}$$

If $\mu(s)$ is δ-close to D_1, then $M(s)$ matches all parentheses in s but at most $2\delta n$ parentheses. To verify this, assume in contradiction that more than $2\delta n$ parentheses are left unmatched by $M(s)$. In other words, that $e_0(s) + e_1(s) > 2\delta n$. But in such a case it is necessary to modify *more than* δn symbols in s so as to obtain a string $\tilde{s}$ such that $e_0(\tilde{s}) = e_1(\tilde{s}) = 0$ (so that $\mu(\tilde{s}) \in D_1$). This would contradict the fact that $\mu(s)$ is δ-close to a string in D_1. By definition of $G(s)$ this implies that

$$\sum_{(k,\ell)\in E(G(s)),\, k<\ell} w(k,\ell) \geq \frac{1}{2}\left(\left(\sum_{i=1}^{n^{1/3}} (e_0(I_{i,i}) + e_1(I_{i,i})) \right) - 2\delta n \right). \tag{6}$$

Combining Equations (5) and (6) together with Claim 10, we obtain

$$\sum_{(k,\ell)\in E(\hat{G}(s)),\, k<\ell} x(k,\ell) \geq \frac{1}{2}\sum_{i=1}^{n^{1/3}} (e_0(I_{i,i}) + e_1(I_{i,i})) - 19\delta n$$

and the proof of Lemma 12 is completed. ■

3.5 Matching Between Neighbors

We define matching substrings as follows.

Definition 11 (Matching substrings) *Let s' be a substring of opening parentheses and let s'' be a substring of closing parentheses. We say that s' and s''* match *if $s's'' \in D_m$.*

Given a string s, it is possible to determine for any two neighbor blocks $i < j$ in $G(s)$, which pairs of excess parentheses within these blocks should match. Let $\mathcal{E}_0(i)$ denote the (non-consecutive) substring of excess opening parentheses in block i, and let $\mathcal{E}_1(j)$ denote the substring of excess closing parentheses in block j. By definition, $|\mathcal{E}_0(i)| = e_0(I_{i,i})$ and $|\mathcal{E}_1(j)| = e_1(I_{j,j})$.

We first find $\mathcal{E}_0(i)$ and $\mathcal{E}_1(j)$. This is done by a slight modification to the D_m-consistency procedure (see Subsection 3.1). Namely, when reading block i, the substring $\mathcal{E}_0(i)$ consists of those opening parentheses that are left on the stack when the procedure terminates. On the other hand, the substring $\mathcal{E}_1(j)$ consists of those closing parentheses, that when read, the stack is found to be empty.

Recall that by Claim 10, for every two blocks $i < j$ that are neighbors in $G(s)$, there are $w(i,j) = x(i,j)$ excess opening parentheses in block i that are matched to excess closing parentheses in block j, where $x(i,j)$ is as defined in Claim 10, Equation (4). Note that there are $e_1(I_{i+1,j-1})$ excess opening parentheses in block i that are matched to excess closing parentheses in the interval $I_{i+1,j-1}$. Similarly, there are $e_0(I_{i+1,j-1})$ closing parentheses in block j that are matched to opening parentheses in $I_{i+1,j-1}$. Observe that either all $e_0(I_{i,i})$ excess opening parentheses in block i get matched to excess closing parentheses in blocks $i+1, \cdots, j$, or all $e_1(I_{j,j})$ excess closing parentheses in block j get matched to opening parentheses in blocks $i, \cdots, j-1$. This leads to the following exact matching procedure, with two cases: The first corresponds to the situation when all of the excess closing parenthesis in block j are matched to parentheses in the interval $I_{i,j-1}$. In particular this implies that those parentheses in block j that are matched to parentheses in block i constitute a *suffix* of the excess substring $\mathcal{E}_1(j)$. The second case corresponds to the situation when all of the excess opening parentheses in block i are matched to parentheses in the interval $I_{i+1,j}$, and so a *prefix* of $\mathcal{E}_0(i)$ is matched to a substring of $\mathcal{E}_1(j)$.

In what follows, for a (consecutive) substring s' of $\mathcal{E}_0(i)$, we denote by $F_i(s')$ and $L_i(s')$ the positions in $\mathcal{E}_0(i)$ of the first and last symbols of s', respectively. Similarly, for a substring s'' of $\mathcal{E}_1(j)$, we denote by $F_j(s'')$ and $L_j(s'')$ the positions of the first and last symbols of s'' in $\mathcal{E}_1(j)$ respectively.

Exact Parentheses Matching Procedure(i,j)

1. If $e_1(I_{i+1,j}) < e_0(I_{i,i})$: Let s'' be the suffix of $\mathcal{E}_1(j)$ of length $x(i,j)$, and let s' be the substring of $\mathcal{E}_0(i)$ such that $L_i(s') = e_0(I_{i,i}) - e_1(I_{i+1,j-1})$, where $|s'| = |s''|$.
2. If $e_1(I_{i+1,j}) \geq e_0(I_{i,i})$: Let s' be the prefix of $\mathcal{E}_0(i)$ of length $x(i,j)$, and let s'' be the substring of $\mathcal{E}_1(j)$ such that $F_j(s'') = e_0(I_{i+1,j-1}) + 1$, where $|s''| = |s'|$.
3. If s' and s'' match, then return success, otherwise return fail.

It may be verified that $L_i(s') = e_0(I_{i,i}) - e_1(I_{i+1,j-1})$ and $F_j(s'') = e_0(I_{i+1,j-1}) + 1$, no matter which step of the procedure is applied. Hence, the two cases can actually be merged into one, but the above formulation will be helpful in understanding a variant of this procedure that is presented subsequently.

An Example. Consider for example the string from Figure 2, and the neighboring blocks $i = 1$ and $j = 4$. Then $\mathcal{E}_0(1) =$ "[[(([([", that is, the block consists only of excess parentheses, and $\mathcal{E}_1(4) =$ "])])". Thus, $e_0(I_{1,1}) = 6$. The other relevant values are: $x(1,4) = 2$, $e_1(I_{2,4}) = 3$, and $e_1(I_{2,3}) = 1$. Hence s'' is the suffix of length 2 of $\mathcal{E}_1(4)$, that is, $s'' =$ "])". We also get that $L_i(s') = 6 - 1 = 5$, and so s' is the substring of $\mathcal{E}_0(1)$ of length 2 that ends at position 5, that is $s' =$ "([". The substrings s' and s'' match of course since $s's'' \in D_2$.

Since we only have estimates $\hat{e}_1(I_{i+1,j-1})$ and $\hat{e}_0(I_{i+1,j-1})$ of the excess numbers in the interval $I_{i+1,j-1}$, then we apply the following *partial* matching procedure to any pair of neighbor blocks $i < j$ in $\hat{G}(s)$. The procedure is basically the same as the exact matching procedure, but it searches for a possibly smaller match in a larger range (where the size of the match and the range are determined by the quality of the approximation we have). Thus we define

$$\hat{x}(i,j) \stackrel{\text{def}}{=} \min\{\hat{e}_1(I_{i+1,j}), e_0(I_{i,i})\} - \hat{e}_1(I_{i+1,j-1}) - 2\delta n^{2/3},$$

and look for matching substrings of length $\hat{x}(i,j)$. Furthermore, we only allow matches of locations that have an even number of symbols between them. If $s \in D_m$ and blocks i and j are neighbors in $G(s)$, then the existing matching between excess opening parentheses in block i and excess closing parentheses in block j, should in fact obey this constraint.

Partial Parentheses Matching Procedure(i,j)

1. If $\hat{e}_1(I_{i+1,j}) < e_0(I_{i,i}) - \delta n^{2/3}$: Let $\hat{s}''$ be the suffix of $\mathcal{E}_1(j)$ of length $\hat{x}(i,j)$. Search for a matching substring $\hat{s}'$ of $\mathcal{E}_0(i)$ such that $L_i(\hat{s}')$ is in the range

$$\Big(e_0(I_{i,i}) - \hat{e}_1(I_{i+1,j-1}) - 3\delta n^{2/3}\ ,\ e_0(I_{i,i}) - \hat{e}_1(I_{i+1,j-1}) + \delta n^{2/3}\Big) \qquad (7)$$

and such that $L_i(\hat{s}')$ has opposite parity from the parity of $F_j(\hat{s}'')$. If $\hat{x}(i,j) \leq 0$ then $\hat{s}''$ is the empty string, and a matching exists trivially.

2. If $\hat{e}_1(I_{i+1,j}) \geq e_0(I_{i,i}) + \delta n^{2/3}$: Let $\hat{s}'$ be the prefix of $\mathcal{E}_0(i)$ of length $\hat{x}(i,j)$. Search for a matching substring $\hat{s}''$ of $\mathcal{E}_1(j)$ such that $F_j(\hat{s}'')$ is in the range
$$(\hat{e}_0(I_{i+1,j-1}) + 1 - \delta n^{2/3}\ ,\ \hat{e}_0(I_{i+1,j-1}) + 1 + 3\delta n^{2/3}) \tag{8}$$
where again, $F_j(\hat{s}'')$ should have opposite parity from that of $L_i(\hat{s}')$.

3. If $|\hat{e}_1(I_{i+1,j}) - e_0(I_{i,i})| \leq \delta n^{2/3}$: Search for a matching as described in Step 1 above. If a matching is not found, search for a matching as described in Step 2 above.

4. If a matching was found, then return success, otherwise return fail.

To implement either step in the above procedure, we run a linear time string matching algorithm [KMP77].

Lemma 13 *Assume that Assumption 5 holds. Then we have the following:*

1. *If $s \in D_m$, then for every two neighbor blocks $i < j$ in $\hat{G}(s)$, the partial matching procedure described above succeeds in finding a matching.*

2. *Let s be any given string and consider any three blocks $i < j_1 < j_2$ such that j_1 and j_2 are both neighbors of i in $\hat{G}(s)$. Suppose that the partial matching procedure succeeds in finding a matching between substrings $\hat{s}'$ and $\hat{t}'$ of $\mathcal{E}_0(i)$ and substrings $\hat{s}''$ of $\mathcal{E}_1(j_1)$ and $\hat{t}''$ of $\mathcal{E}_1(j_2)$, respectively. Then, under Assumption 5, $\hat{s}'$ and $\hat{t}'$ overlap by at most $6\delta n^{2/3}$. An analogous statement holds for triples $i_1 < i_2 < j$ such that i_1 and i_2 are both neighbors of j in $\hat{G}(s)$.*

Proof:
Part 1: By Lemma 12, $\hat{G}(s)$ is a subgraph of $G(s)$. In other words, every two blocks $i < j$ that are neighbors in $\hat{G}(s)$, are also neighbors in $G(s)$. If $s \in D_m$, then this implies that the exact matching procedure would succeed in finding a match between substrings s' of $\mathcal{E}_0(i)$ and s'' of $\mathcal{E}_1(j)$ in either Step 1 or Step 2 of the procedure.

We consider the first case, and show that in this case the partial matching procedure can find a match between $\hat{s}'$ and $\hat{s}''$ in Step 1 (the second case is handled analogously). In this case, $e_1(I_{i+1,j}) < e_0(I_{i,i})$, and there is a matching between the suffix s'' of $\mathcal{E}_1(j)$ that has length $x(i,j)$, and the substring s' of $\mathcal{E}_0(i)$ of the same length that ends in position $L_i(s') = e_0(I_{i,i}) - e_1(I_{i+1,j-1})$.

By Claim 8, conditioned on Assumption 5, we know that for every k, ℓ, $|\hat{e}_1(I_{k,\ell}) - e_1(I_{k,\ell})| \leq \delta n^{2/3}$. It follows that $\hat{e}_1(I_{i+1,j}) < e_0(I_{i,i}) + \delta n^{2/3}$ and that
$$x(i,j) - 4\delta n^{2/3} \leq \hat{x}(i,j) \leq x(i,j). \tag{9}$$
Therefore, either $\hat{e}_1(I_{i+1,j}) < e_0(I_{i,i}) - \delta n^{2/3}$, or $|\hat{e}_1(I_{i+1,j}) - e_0(I_{i,i})| \leq \delta n^{2/3}$, and in either case, the procedure tries to find a match as defined in Step 1.

Since $\hat{x}(i,j) \leq x(i,j)$, the substring $\hat{s}''$ defined in Step 1 is a suffix of s''. Since we know that s' matches s'' in this case, then there is a prefix $\hat{s}'$ of s' that matches $\hat{s}''$, and we just have to show that the partial matching procedure can find it. Let $\hat{e}_1(I_{i+1,j}) = e_1(I_{i+1,j}) + y$, and $\hat{e}_1(I_{i+1,j-1}) = e_1(I_{i+1,j-1}) + z$ where $-\delta n^{2/3} \leq y, z \leq \delta n^{2/3}$. Hence,
$$|\hat{s}'| = \hat{x}(i,j) = x(i,j) + y - z - 2\delta n^{2/3} = |s'| + y - z - 2\delta n^{2/3}$$

and so

$$L_i(\hat{s}') = L_i(s') - (y - z - 2\delta n^{2/3}) = e_0(I_{i,i}) - \hat{e}_1(I_{i+1,j-1}) - (y - 2\delta n^{2/3}).$$

The matching substring that the partial matching algorithm searches for is allowed to end at a position in the range $(e_0(I_{i,i}) - \hat{e}_1(I_{i+1,j-1}) - 3\delta n^{2/3}\ ,\ e_0(I_{i,i}) - \hat{e}_1(I_{i+1,j-1}) + \delta n^{2/3})$, which contains $L_i(\hat{s}')$. This ensures that a matching is found.

Part 2: Let $i < j_1 < j_2$ be a given triple as defined in the lemma. We first show that regardless of whether $\mu(s) \in D_1$ or not, given the exact values of $e_0(I_{i+1,j_1-1})$, $e_1(I_{i+1,j_1-1})$, $e_0(I_{i+1,j_2-1})$, and $e_1(I_{i+1,j_2-1})$, the matching defined by the exact matching procedure would not cause any overlaps. This fact will be used to bound the overlap caused by the partial matching procedure.

Let s' and t' be substrings of $\mathcal{E}_0(i)$ that the exact matching procedure tries to match to substrings s'' of $\mathcal{E}_1(j_1)$ and t'' of $\mathcal{E}_1(j_2)$ respectively. We next show the following inequalities:

1. $L_i(t') < F_i(s')$, that is, the exact matching algorithm would not cause any overlap: By definition of the exact matching procedure, (no matter which step is applied), $L_i(t') = e_0(I_{i,i}) - e_1(I_{i+1,j_2-1})$, $L_i(s') = e_0(I_{i,i}) - e_1(I_{i+1,j_1-1})$, and $|s''| = |s'| = x(i, j_1)$. Thus,

$$\begin{aligned} F_i(s') &= e_0(I_{i,i}) - e_1(I_{i+1,j_1-1}) - x(i,j_1) + 1 \\ &= e_0(I_{i,i}) - e_1(I_{i+1,j_1-1}) - (\min\{e_1(I_{i+1,j_1}), e_0(I_{i,i})\} - e_1(I_{i+1,j_1-1})) + 1 \\ &= e_0(I_{i,i}) - \min\{e_1(I_{i+1,j_1}), e_0(I_{i,i})\} + 1 \\ &\geq e_0(I_{i,i}) - e_1(I_{i+1,j_1}) + 1 \end{aligned} \tag{10}$$

   Since $j_1 \leq j_2 - 1$ and $e_1(I_{i+1,j})$ is monotonically non-decreasing with j, we have that $e_1(I_{i+1,j_2-1}) \geq e_1(I_{i+1,j_1})$, and so $L_i(t') < F_i(s')$ as desired.

2. As observed in Part 1 of this proof, $\hat{x}(i,j) \leq x(i,j)$ and so $|\hat{s}''| \leq |s''|$.

3. $L_i(\hat{s}') \geq L_i(s') - 4\delta n^{2/3}$: We first observe that by Lemma 12, $x(i, j_2) = \min\{e_1(I_{i+1,j_2}), e_0(I_{i,i})\} - e_1(I_{i+1,j_2-1}) \geq 6\delta n^{2/3}$. Since $j_1 \leq j_2 - 1$ and $e_1(i+1, j)$ is monotonically non-decreasing with j, necessarily $e_1(I_{i+1,j_1}) \leq e_0(I_{i,i}) - 6\delta n^{2/3}$. Therefore, $\hat{e}_1(I_{i+1,j_1}) \leq e_0(I_{i,i}) - 5\delta n^{2/3}$, and so the partial matching procedures would apply Step 1. Thus, the substring $\hat{s}'$ may end in the worst case in position

$$e_0(I_{i,i}) - \hat{e}_1(I_{i+1,j_1-1}) - 3\delta n^{2/3} \geq e_0(I_{i,i}) - e_1(I_{i+1,j_1-1}) - 4\delta n^{2/3} = L_i(s') - 4\delta n^{2/3}.$$

4. $F_i(\hat{s}') \geq F_i(s') - 4\delta n^{2/3}$: Using the previous inequalities we get,

$$\begin{aligned} F_i(\hat{s}') &= L_i(\hat{s}') - \hat{x}(i,j) \\ &\geq L_i(s') - 4\delta n^{2/3} - \hat{x}(i,j) \\ &= (F_i(s') + x(i,j)) - 4\delta n^{2/3} - \hat{x}(i,j) \\ &\geq F_i(s') - 4\delta n^{2/3} \end{aligned} \tag{11}$$

5. $L_i(\hat{t}') \leq L_i(t') + 2\delta n^{2/3}$: As noted previously, by definition of the exact matching procedure, $L_i(t') = e_0(I_{i,i}) - e_1(I_{i+1,j_2-1}) + 1$. If the match between $\hat{t}'$ and $\hat{t}''$ is found in Step 1 of the partial matching procedure, then

$$L_i(\hat{t}') \leq e_0(I_{i,i}) - \hat{e}_1(I_{i+1,j_2-1}) + \delta n^{2/3} \leq e_0(I_{i,i}) - e_1(I_{i+1,j_2-1}) + 2\delta n^{2/3} = L_i(t') + 2\delta n^{2/3}.$$

   If the match is found in Step 2, then

$$L_i(\hat{t}') = \hat{x}(i,j) \leq e_0(I_{i,i}) - \hat{e}_1(I_{i+1,j_2-1}) - 2\delta n^{2/3} < L_i(t').$$

As a result we get

$$F_i(\hat{s}') \geq F_i(s') - 4\delta n^{2/3} > L_i(t') - 4\delta n^{2/3} \geq L_i(\hat{t}') - 6\delta n^{2/3}.$$

and therefore the overlap is at most $6\delta n^{2/3}$. We have thus proved the claim for triples $i < j_1 < j_2$. The analogous claim for triples $i_1 < i_2 < j$ is proved similarly. ■

3.6 Putting it all together

Algorithm 1 Test if $s \in D_m$

1. *Let* $\delta = \frac{\epsilon}{200}$.
2. D_1**-test:** *Test that* $\mu(s) \in D_1$ *with distance parameter* δ *and confidence* $9/10$. *If this test rejects, then output* reject.
3. *Partition the string* s *into* $n^{1/3}$ *substrings of length* $n^{2/3}$ *each, which we refer to as "blocks".*
4. D_m**-consistency test:** *Select* $100/\epsilon$ *blocks uniformly, and check that they are* D_m*-consistent. If any of the blocks selected is not* D_m*-consistent then output* reject.
5. *Perform the preprocessing step on the basic substrings of* s *(defined based on the above setting of* δ*).*
6. **Matching test:** *Uniformly select* $100/\epsilon$ *blocks and for each find its neighboring blocks in* $\hat{G}(s)$. *For each selected block, and for each of its neighbors, check that their excess parentheses match correctly by invoking the partial matching procedure. If the partial matching procedure fails for any of the selected blocks then output* reject. *Otherwise output* accept.

Theorem 1 *If* $s \in D_m$ *then the above testing algorithm accepts with probability at least* $2/3$, *and if* s *is* ϵ*-far from* D_m *then the above test rejects with probability at least* $2/3$.
The query complexity and running time of the algorithm are $O\left(n^{2/3} \cdot \log^3(n/\epsilon)/\epsilon^3\right)$.

Proof: Consider first the easier case in which $s \in D_m$. The D_1-test (Step 2) passes with probability at least $9/10$, and the D_m-consistency test (Step 4) always passes. By Lemma 12, if Assumption 5 holds, then $\hat{G}(s)$ is a subgraph of $G(s)$. By Lemma 13 (using Assumption 5 once again), for every two neighboring blocks i and j, the matching of excess parentheses must succeed. Since by Lemma 4, Assumption 5 holds with high probability, this part of the theorem follows.

We now turn to the second part of the theorem. We shall show that conditioned on Assumption 5 holding, if s is accepted with probability greater than $1/6$, then it is ϵ-close to some string in D_m. This implies that conditioned on Assumption 5 holding, if s is ϵ-far from D_m then it is rejected with probability at least $5/6$. Since Assumption 5 holds with probability at least $5/6$, this implies that if s is ϵ-far from D_m then it is rejected with probability at least $2/3$, as required. Thus from now on we assume that Assumption 5 holds.

If s is accepted with probability greater than $1/6$ then necessarily it must pass each part of the test with probability greater than $1/6$. This implies that:

1. $\mu(s)$ is δ-close to a string in D_1: otherwise, it would be rejected in Step 2 of the algorithm with probability at least $9/10$;

2. All but at most an $\frac{\epsilon}{4}$-fraction of the blocks of s are D_m-consistent: otherwise, an inconsistent block would be selected in Step 4 with probability greater than 5/6, causing the algorithm to reject in this step with probability greater than 5/6;

3. The fraction of blocks i that have a neighbor j in $\hat{G}(s)$ for which the partial matching procedure would fail if executed on i and j is at most $\epsilon/4$: otherwise, one of these blocks would be selected in Step 6 with probability greater than 5/6, causing the algorithm to reject in this step with probability greater than 5/6;

4. Combining the first item above ($\mu(s)$ is δ-close to a string in D_1) with Assumption 5, we know by Lemma 12, that $\hat{G}(s)$ is a planar graph, and furthermore,

$$\sum_{(k,\ell)\in E(\hat{G}(s)),\, k<\ell} x(k,\ell) \geq \frac{1}{2}\sum_{i=1}^{n^{1/3}}(e_0(I_{i,i}) + e_1(I_{i,i})) - 19\delta n,$$

where $x(k,\ell)$ is as defined in Claim 10, Equation (4).

To show that s is ϵ-close to a string in D_m, we show how to modify s in at most ϵn positions so that it becomes a string in D_m. In particular we show the existence of a nested non-crossing matching between opening and closing parentheses in the modified string, such that every matched pair match in type.

Making all blocks D_m consistent. First we consider all blocks that are not D_m-consistent, and turn them into consistent blocks *without modifying their excess parentheses*. As stated in the discussion following Fact 2, for every block s', the string $\mu(s')$ is D_1-consistent. Hence this modification can be done simply by considering the matching induced on the non-excess parentheses, and modifying at most 1/2 of the non-excess parentheses in the block. Since the fraction of inconsistent blocks is at most an $\epsilon/4$-fraction of the blocks of s, the total number of symbols modified is at most $\frac{\epsilon}{8}n$.

Adjusting matched excess parentheses. Next we need to "fix" the excess parentheses. Consider the graph $\hat{G}(s)$, and for every two blocks $i < j$ that are neighbors in $\hat{G}(s)$, consider (as a mental experiment) the result of running the partial matching algorithm on their excess parentheses substrings $\mathcal{E}_0(i)$ and $\mathcal{E}_1(j)$ as described in Subsection 3.5. Suppose that we succeed and find a matching between substring s' of $\mathcal{E}_0(i)$ and substring s'' of $\mathcal{E}_1(j)$. Then we shall "commit" to the two matched substrings with the exception of the last $6\delta n^{2/3}$ symbols of s' and the first $6\delta n^{2/3}$ symbols of s''. In "committing" we mean that these symbols will not be modified, and that the matching between the respective symbols in s' and s'' will be maintained in all future modifications. Note that by Lemma 13, each excess opening parenthesis is matched in this way to at most one excess closing parenthesis in one of the neighbors of block i.

If the matching algorithm does not succeed then we modify a substring s' of $\mathcal{E}_0(i)$ so that it matches a designated substring s'' of $\mathcal{E}_1(j)$, with the exception of $6\delta n^{2/3}$ consecutive symbols, and we commit to the two matched substrings. More precisely, if $e_1(I_{i,i}) < e_0(I_{i,i})$ (so that $\hat{e}_1(I_{i,i}) < e_0(I_{i,i})+\delta n^{2/3}$), then we let τ be the suffix of $\mathcal{E}_1(j)$ of length $\hat{x}(i,j)-6\delta n^{2/3}$, where $\hat{x}(i,j)$ is as defined in the partial matching procedure. We then modify the substring of $\mathcal{E}_0(i)$ of length $\hat{x}(i,j) - 6\delta n^{2/3}$ that ends at position $e_0(I_{i,i}) - \hat{e}_1(I_{i+1,j-1}) - 6\delta n^{2/3}$ so that it matches τ. If $e_1(I_{i,i}) \geq e_0(I_{i,i})$ then we modify the prefix of length $\hat{x}(i,j) - 6\delta n^{2/3}$ of $\mathcal{E}_0(i)$ so that it matches the substring τ of length $\hat{x}(i,j) - 6\delta n^{2/3}$ of $\mathcal{E}_0(i)$ that starts at position $F_j(\tau) = \hat{e}_0(I_{i+1,j-1}) + 1 + 6\delta n^{2/3}$. It is not hard to verify that in this manner we do not introduce any errors into the matching.

Since the fraction of blocks that have at least one neighbor on which the partial matching procedure fails is at most $\frac{\epsilon}{4}$, the total number of symbols modified in this stage is at most $\frac{\epsilon}{4}n$. Note that since $\hat{G}(s)$ is planar, the matching defined so far is nested as required.

Adjusting non-matched excess parentheses. At this stage the string s is composed of three types of consecutive substrings: (1) substrings inside the blocks that are strings in D_m themselves; (2) excess parentheses in one block that are matched to excess parentheses in another block, to which we committed; (3) excess parentheses that are not matched.

We now show how to change s into a string in D_m, but first let us bound the total number of parentheses of type (3), which must still be modified. The total number of excess parentheses is $\sum_{i=1}^{n^{1/3}} (e_0(I_{i,i}) + e_1(I_{i,i}))$. Let us think of each edge (i,j) in $\hat{G}(s)$ as "having to account for" $x(i,j)$ pairs of excess parentheses. By Lemma 12, the total number of excess parentheses that are not accounted to by the edges of $\hat{G}(s)$ is at most $2 \cdot 19\delta n$. In addition, by definition of the approximate matching procedure, the length of the matched substring corresponding to the edge is $\hat{x}(i,j) \geq x(i,j) - 4\delta n^{2/3}$ (see Equation 9). By our committing strategy, for every edge (i,j) in $\hat{G}(s)$, the number of pairs of symbols among the matched $\hat{x}(i,j)$ pairs that we did not commit to is $6\delta n^{2/3}$. Thus the total number of uncommitted excess parentheses pairs per edge is at most $10\delta n^{2/3}$. Since $\hat{G}(s)$ is planar, the total number of uncommitted pairs is at most $30\delta n$. Hence there are at most $60\delta n$ excess parentheses that are accounted for by the edges of $\hat{G}(s)$, but uncommitted for. If we add the $36\delta n$ parentheses that are not accounted for we get a total of at most $96\delta n$ parentheses of type (3). We now show how to modify these at most $96\delta n$ parentheses, so that we get a string in D_m. Since $\delta = \frac{\epsilon}{200}$ we modify in this step at most $\frac{\epsilon}{2}n$ symbols.

Let t be the string obtained from s by removing all consecutive substrings of type (1). Note that by removing such substrings that are always even in length, we do not change the parity of the length of substrings between two matched excess substrings of type (2). We show how to modify t in a recursive way. Let t' and t'' be two matched substrings such that between them there is only a substring τ of parentheses of type (3), where τ may be empty. Thus, t is of the form $t = \sigma' t' \tau t'' \sigma''$. Note that $|\tau|$ must be even, since the position of the last symbol of t' has opposite parity than that of the position of the last symbol of t''. Therefore, we modify τ so that it is a string in D_m and continue recursively with the string $\sigma'\sigma''$. This string is even in length since $|t|$ and $|\tau|$ are even and $|t'| = |t''|$. Also as noted above by removing consecutive substrings that are even in length, we do not change the parity of the length of substrings between two matched excess substrings of type (2).

Finally we turn to the query complexity and running time of the algorithm. Testing that $\mu(s) \in D_1$ with distance parameter $\epsilon' = \delta$ takes time $O(\log(1/\epsilon')/\epsilon')$ [AKNS00], which is $O(\log(1/\epsilon)/\epsilon)$. Testing that $O(1/\epsilon)$ blocks are D_m-consistent takes time $O\left(n^{2/3}/\epsilon\right)$. The preprocessing step takes time linear in the sample size, which by Lemma 4 is $O\left(n^{2/3} \cdot \log^3(n/\epsilon)/\epsilon^3\right)$. Finally, by Lemma 12 the degree of every vertex (block) in $\hat{G}(s)$ is at most $1/(2\delta) = O(1/\epsilon)$. Therefore the last step takes time $O\left(n^{2/3}/\epsilon^2\right)$. The total running time and query complexity is hence $O\left(n^{2/3} \cdot \log^3(n/\epsilon)/\epsilon^3\right)$. ■

4 A Lower Bound for D_2

In this section we prove a lower bound of $\Omega(n^{1/11}/\log n)$ on the query complexity of any algorithm for testing D_2, and hence for testing all Dyck languages. We first provide such a bound for the language PAR_2 which is defined below and is a variant of D_2, and then discuss how a very similar argument can be applied to obtain the same lower bound for D_2.

Definition 12 (Parenthesis Languages) *The parenthesis language* PAR_m *over strings in* $\Sigma_m \cup \Sigma'$*, where* Σ' *is any alphabet that has no intersection with* Σ_m*, can be defined recursively as follows:*

1. *Any string* $s \in (\Sigma')^*$ *belongs to* PAR_m*.*
2. *If* $s' \in \mathrm{PAR}_m$*,* $\sigma = 2i$ *is an opening parenthesis and* $\tau = 2i+1$ *is a matching closing parenthesis, for some* $0 \le i \le m-1$*, then* $\sigma s' \tau \in \mathrm{PAR}_m$*.*
3. *If* $s', s'' \in \mathrm{PAR}_m$*, then* $s's'' \in \mathrm{PAR}_m$*.*

We will prove the following theorem.

Theorem 2 *Any algorithm for testing* PAR_2 *with distance parameter* $\epsilon \le 2^{-6}$ *and success probability of at least* $2/3$*, requires* $\alpha n^{1/11}/\log n$ *queries, where* $\alpha = e^{-7}$*.*

The high-level structure of our proof is similar to other lower-bound proofs for testing (see for example [GR02, PR02, BR02] which can be traced back to [Yao77]). In order to prove the theorem we define two distributions, called POS_n and NEG_n, on strings over $\Sigma_2 \cup \{\text{'a'}\}$ (that is, there are two types of parentheses and one extra non-parenthesis symbol). Since we have only two types of parentheses, it will be convenient to let $\Sigma_2 = \{(,),[,]\}$. The support of the first distribution, POS_n, contains only strings in PAR_2, while with extremely high probability, a string selected according to the second distribution, NEG_n, is 2^{-6}-far from PAR_2. Roughly speaking, what we show is that an algorithm that asks less than $\alpha n^{1/11}/\log n$ queries cannot distinguish with sufficiently high probability between a string selected according to the first distribution (which should be accepted) and a string selected according to the second distribution (which should almost always be rejected).

4.1 The Two Distributions

In what follows we assume for simplicity that the length n of the strings, is divisible by 32. In both distributions the support of the distributions is only on strings s such that $\mu_1(s) \in \mathrm{PAR}_1$, where we extend the mapping $\mu_1(\cdot)$ defined in Definition 3, so that it maps every 'a' to 'a'. Furthermore, the strings have a relatively simple structure: there are always $n/4$ opening parentheses among the first $n/2$ symbols (the *left half* of the string), and $n/4$ closing parentheses among the last $n/2$ symbols (the *right half* of the string). All other symbols are 'a's. The strings differ only in the actual positions of the parentheses in the string and in their *type*:

Definition 13 (Parenthesis Types) *We say that an opening parenthesis is of* type *0 if it is* '(', *and is of type 1 if it is* '['. *Similarly, we say that a closing parenthesis is of type 0 if it is* ')', *and it is of type 1 if it is* ']'. *Thus,* '(' *and* ')', *and similarly* '[' *and* ']', *are said to have* the same type.

4.1.1 The First Distribution, POS_n.

This distribution is simply uniform over all strings in PAR_2 that have $n/4$ opening parentheses among the first $n/2$ positions, and $n/4$ closing parentheses (of corresponding types) among the last $n/2$ positions. To be precise, a string s is generated in the following manner:

1. Uniformly select a subset $L \subset \{1, \ldots, n/2\}$ such that $|L| = n/4$. These will be the positions in s of the opening parentheses.
2. Uniformly select a subset $R \subset \{n/2+1, \ldots, n\}$ such that $|R| = n/4$. These will be the positions in s of the closing parentheses.

3. Uniformly select a binary string $x \in \{0,1\}^{n/4}$. The string x will be used to determine the type of parentheses.

4. Let $j_1, j_2, \ldots, j_{n/4}$ be the elements of L where $n/2 \geq j_1 > j_2 > \cdots > j_{n/4} \geq 1$. Then, for every $1 \leq i \leq n/4$, if $x_i = 0$ then $s_{j_i} =$'(', and if $x_i = 1$ then $s_{j_i} =$'['.

5. Similarly, consider a sorted order of the indices in R, only here the order is reversed so that $n/2+1 \leq k_1 < k_2 < \cdots < k_{n/4} \leq n$. Then, for every $1 \leq i \leq n/4$, if $x_i = 0$ then $s_{k_i} =$')', and if $x_i = 1$ then $s_{k_i} =$']'.

6. For every $i \notin L \cup R$, let $s_i =$'a'.

Thus, each string in the support of POS_n has probability $\binom{n/2}{n/4}^{-2} \cdot 2^{-n/4}$.

4.1.2 The Second Distribution, NEG_n.

This distribution is similar to POS_n (and in particular its support contains the support of POS_n), with the exception that not all pairs of parentheses (j_i, k_i) as defined above have the same type. In particular, the generating procedure is the same as that of POS_n described above, with the exception of Steps 3 and 5 that are modified below.

1,2. As described for POS_n.

3. Uniformly select a binary string $x \in \{0,1\}^{n/4}$, and a binary string $y \in \{0,1\}^{n/8}$.

4. As described for POS_n.

5. Consider a sorted order of the indices in R so that $k_1 < k_2 < \cdots < k_{n/4}$. Then, for every i such that $1 \leq i \leq n/16$ or $n/4 - n/16 + 1 \leq i \leq n/4$, if $x_i = 0$ then $s_{k_i} =$')', and if $x_i = 1$ then $s_{k_i} =$']'. For every $n/16+1 \leq i \leq n/4 - n/16$, if $y_{i-n/16} = 0$ then $s_{k_i} =$')', and if $y_{i-n/16} = 1$ then $s_{k_i} =$']'.

   That is, as opposed to POS_n, here the string x determines only the type of the first $n/16$ and the last $n/16$ parentheses on the right side of the string, while the string y determines the type of the remaining $n/8$ middle parentheses.

6. As described for POS_n.

Thus, each string in the support of NEG_n has probability $\binom{n/2}{n/4}^{-2} \cdot 2^{-3n/8}$.

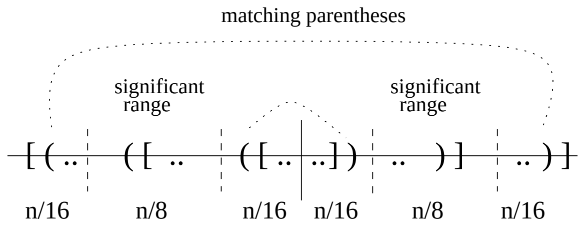

Figure 4: An illustration of strings in the two distributions. The horizontal line represents a string. The central vertical line represents the middle of the string – to the left of it there are only opening parentheses and to the right only closing parentheses. The other dashed vertical lines represent the borders of the regions in which reside the first and last $n/16$ parentheses and the middle $n/8$ parentheses in each side. The middle $n/8$ pairs must match in POS_n and do not necessarily match in NEG_n.

4.1.3 Properties of the Distributions

The following definitions will be central to our analysis.

Definition 14 (Parenthesis Index) *Let s be a string in the support of NEG_n (which in particular contains the support of POS_n), and let $1 \le j \le n/2$ be a position such that s_j is an opening parenthesis. The* parenthesis index *of j in s is the number of opening parentheses $s_{j'}$ such that $j \le j' \le n/2$. We denote the parenthesis index of j in s by $\pi_s(j)$.*

Analogously, for a position $n/2+1 \le k \le n$ such that s_k is a closing parenthesis, the parenthesis index *of k in s is the number of closing parentheses $s_{k'}$ such that $n/2+1 \le k' \le k$.*

Definition 15 (Significant Index, Significant Range) *We say that a parenthesis index $1 \le \pi \le n/4$ is* significant *if $n/16+1 \le \pi \le n/4 - n/16$. Otherwise, it its* non-significant*. We shall call the range of indices between $n/16+1$ and $n/4 - n/16$, the* significant range.

Note that the parenthesis index of a position is not determined by the position itself but rather by the number of parentheses between this position and the middle of the string. Observe that for every string s in the support of POS_n, and for every two positions $1 \le j \le n/2$ and $n/2+1 \le k \le n$, such that s_j is an opening parenthesis and s_k is a closing parenthesis, if $\pi_s(j) = \pi_s(k)$, then s_j and s_k must be of the same type. For a string s in the support of NEG_n, the above is necessarily true only for pairs j, k such that $\pi_s(j) = \pi_s(k) = \pi$ and π is not a significant parenthesis index.

Lemma 14 *Let $\epsilon \le 2^{-6}$. Then the probability that a string generated according to NEG_n is ϵ-far from PAR_2, is at least $1 - \exp(-\Omega(n))$.*

Proof: Consider all possible ways in which a given string s that is generated according to NEG_n can be modified in at most ϵn places. There are $\binom{n}{\epsilon n}$ selections of subsets $C \subset \{1, \ldots, n\}$, $|C| = \epsilon n$, and for each $i \in C$ the symbol s_i can be modified to any one of five symbols (this includes not changing s_i which accounts for the possibility of modifying *less* than ϵn positions). That is, there are

$$\binom{n}{\epsilon n} \cdot 5^{\epsilon n} \le 2^{((1+o(1)) \cdot H(\epsilon) + (\log 5) \cdot \epsilon) \cdot n} \tag{12}$$

possible ways to modify the string, where $H(\cdot)$ is the binary entropy function (that is, $H(\epsilon) = \epsilon \log(1/\epsilon) + (1-\epsilon)\log(1/(1-\epsilon))$).

For each string s in the support of NEG_n, and for each $C \subset \{1, \ldots, n\}$, $|C| = \epsilon n$ and $t \in \{\Sigma_2 \cup \{a\}\}^{\epsilon n}$, let the string $s^{(C,t)}$ be defined as follows: for every $i \notin C$, $s_i^{(C,t)} = s_i$, and for every $i \in C$, $s_i^{(C,t)} = t_i$. We say that the pair (C,t) *corrects* s if $s^{(C,t)}$ is in PAR_2. The probability that a string generated according to NEG_n is ϵ-close to PAR_2 is the probability over the choice of s according to NEG_n that there exists a pair (C,t) that corrects s. We thus consider any particular subset C and any particular string t and show that the probability over the choice of s that (C,t) corrects s is exponentially smaller than the number of pairs (C,t). By applying a union bound we prove the lemma.

We shall actually prove a slightly stronger claim. For a fixed choice of (C,t), consider the process of selecting a string s according to NEG_n. Recall that a string s is generated by first uniformly selecting the sets of parentheses positions L and R, and then randomly setting the types of parentheses in these positions (according to the choice of the strings x and y). We shall show that *for every* choice of L and R, the probability, taken only over the choice of types of parentheses, that the resulting string is corrected by (C,t), is sufficiently small. Details follow.

For a given choice of $L \subset \{1, \ldots, n/2\}$, $|L| = n/4$ and $R \subset \{n/2+1, \ldots, n\}$, $|R| = n/4$, let $S(L, R)$ denote the subset of words in the support of NEG_n that have opening parentheses in the positions in L and closing parentheses in the positions in R. Then either for every string $s \in S(L, R)$ we have that $\mu_1(s^{(C,t)}) \in \mathrm{PAR}_1$, or for every string $s \in S(L, R)$, $\mu_1(s^{(C,t)}) \notin \mathrm{PAR}_1$. In other words, in the latter case, no matter how the types of parentheses are set in the positions determined by L and R, the resulting string is not corrected by (C, t). Thus assume from now on that L and R are such that $\mu_1(s^{(C,t)}) \in \mathrm{PAR}_1$. Note that for every $s \in S(L, R)$, the matching $M(s^{(C,t)})$ is exactly the same. Let us thus denote it by $M(L, R, C, t)$.[1]

Let $n/2 + 1 \leq k_1 < k_2 < \ldots < k_{n/4} \leq n$ be the indices in R in sorted order. Since $|C| = \epsilon n$, the number of indices k_i such that either $k_i \in C$ or k_i is matched by $M(L, R, C, t)$ to some $\ell \in C$ is at most $2\epsilon n$. Therefore, there must be at least $n/8 - 2\epsilon n$ positions k_i for $i \in \{n/16+1, \ldots, n/4 - n/16\}$, such that $k_i \notin C$, *and* $M(L, R, C, t)$ matches k_i to some $j_{i'} \in L$ such that $j_{i'} \notin C$. If (C, t) corrects a string $s \in S(L, R)$, then it must be the case that the strings x and y (as defined in the description of NEG_n) are such that all the above $n/8 - 2\epsilon n$ pairs of positions matched by $M(L, R, C, t)$, have the same type of parentheses within each pair. The probability of this event, taken over the choice of x and y is at most $2^{-(n/8 - 2\epsilon n)}$.

Since the above is true for every L and R (such that $\mu_1(s^{(C,t)}) \in \mathrm{PAR}_1$ for every $s \in S(L, R)$), we obtain a bound on the probability that the given (C, t) corrects a string s generated according to NEG_n.

Applying Equation (12), which gives a bound on the number of choices of (C, t), we see that if we select ϵ so that $(H(\epsilon) + (\log 5) \cdot \epsilon)$ is sufficiently smaller than $(1/8 - 2\epsilon)$, then we are done. A choice of $\epsilon = 1/64$ will do. ■

The following simple claim will be useful later on. It states that with sufficiently high probability over the choice of a string generated by one of the two distributions defined above, the parenthesis index of every position does not deviate by much from its expected value.

Claim 15 *With probability at least* $7/8$ *over the choice of a string* s *according to* POS_n *(similarly,* NEG_n*), for every* $1 \leq j \leq n/2$ *such that* s_j *is an opening parenthesis, and for every* $n/2+1 \leq k \leq n$ *such that* s_k *is a closing parenthesis,*

$$\left| \pi_s(j) - \frac{n/2 - j}{2} \right| \leq \sqrt{\log n \cdot \min\{(n/2 - j), j\}}$$

and

$$\left| \pi_s(k) - \frac{k - n/2}{2} \right| \leq \sqrt{\log n \cdot \min\{(k - n/2), (n - k)\}}$$

Proof: We prove the claim concerning $1 \leq j \leq n/2$. The second claim concerning $n/2 < k \leq n$ is proved analogously. Let us fix an index j and assume, without loss of generality, that $j \leq n/4$, so that $\min\{(n/2 - j), j\} = j$. Recall that for any string in the support of NEG_n, the total number of opening parentheses among the first $n/2$ positions is exactly $n/4$. Hence, $\pi_s(j)$ deviates by more than $\sqrt{\log n \cdot j}$ from $\frac{n/2-j}{2}$ if and only if the number of opening parentheses in s among the first j positions deviates by more than $\sqrt{\log n \cdot j}$ from $\frac{j}{2}$ (the expected number of parentheses). The probability that there are *at most* $\frac{j}{2} - \sqrt{\log n \cdot j}$ parentheses in these positions is

$$\frac{\sum_{i=0}^{j/2 - \sqrt{j \log n}} \binom{j}{i} \cdot \binom{n/2 - j}{n/4 - i}}{\binom{n/2}{n/4}} \leq \frac{\sum_{i=0}^{j/2 - \sqrt{j \log n}} \binom{j}{i} \cdot \binom{n/2 - j}{n/4 - j/2}}{\binom{n/2}{n/4}}$$

[1]This matching clearly does not depend on the type of parentheses in t but only on whether they are opening or closing parentheses, but for simplicity we denote it as if it depends on t.

$$= O\left(\sum_{i=0}^{j/2-\sqrt{j\log n}} \binom{j}{i} \cdot \frac{2^{n/2-j}/\sqrt{n/2-j}}{2^{n/2}/\sqrt{n/2}}\right)$$
$$= O\left(2^j \cdot n^{-2} \cdot 2^{-j}\right) \;=\; O(n^{-2}) \qquad (13)$$

Similarly, the probability that there are *at least* $\frac{j}{2} + \sqrt{\log n \cdot j}$ parentheses in these positions is $O(n^{-2})$ as well. By applying a union bound over all positions j, we get that the probability that there is a large deviation from the expectation for any of the indices is $O(n^{-1})$, which for a sufficiently large n is smaller than $1/8$. ■

4.2 On Distinguishing POS_n from NEG_n

Let $\mathcal{A}$ be a possibly randomized testing algorithm for PAR_2 that asks at most $\alpha n^{1/11}/\log n$ queries for $\alpha \leq e^{-7}$. We define two randomized processes $\mathcal{P}_{\mathrm{POS}}$ and $\mathcal{P}_{\mathrm{NEG}}$ that interact with $\mathcal{A}$. The following claim will follow immediately from their definition.

Claim 16 *The distribution on answers provided by the process $\mathcal{P}_{\mathrm{POS}}$ is equivalent to those obtained from querying a string that is randomly generated according to POS_n. Similarly, the distribution on answers provided by the process $\mathcal{P}_{\mathrm{NEG}}$ is equivalent to those obtained from querying a string that is randomly generated according to NEG_n.*

In particular, at any stage of the interaction, each process considers the set of strings that are consistent with the interaction so far. Given a new query, the probability distribution on the answer is determined by the relative fraction of strings in the set that are consistent with that answer (because both distributions are uniform over their support). While we won't be able to compute these probabilities exactly, we shall be able to bound them, and this will suffice for our proof.

4.2.1 The Process $\mathcal{P}_{\mathrm{POS}}$.

We start by describing $\mathcal{P}_{\mathrm{POS}}$. At each step of the interaction the process maintains the set of positions already queried by the algorithm, and the answers it has provided (that is, what is the symbol in each queried position). In addition, the process $\mathcal{P}_{\mathrm{POS}}$ maintains a subset, denoted MATCHED, of disjoint pairs (j, k) of previously queried positions, where $1 \leq j \leq n/2$, $n/2 + 1 \leq k \leq n$, and both positions were answered by parentheses having the same parenthesis index (as explained next). With each such pair it associates a common parenthesis index $1 \leq \pi \leq n/4$. The final generated string s will be such that for every pair $(j, k) \in$ MATCHED, $\pi_s(j) = \pi_s(k) = \pi$, and for any other two queried positions j', k' such that $(j', k') \notin$ MATCHED, $\pi_s(j') \neq \pi_s(k')$. We stress that the process only "commits" to the parenthesis index of a subset of *pairs* of queried positions, and not to the parenthesis index of every queried position that is answered by a parenthesis.

Before continuing with the description of the process, we introduce two definitions. The first definition is of the query-answer history of an interaction between a testing algorithm and the process $\mathcal{P}_{\mathrm{POS}}$. This history contains the positions queried by the algorithm and the symbols that the process returns as answers. In addition it includes the information concerning queried positions that the process decides to match. Clearly, in an actual execution of the algorithm such information is not provided directly. However, it is also clear that giving this extra information to the algorithm can only help it.

Definition 16 (Query-Answer History) *The* query-answer *history h of length T is a sequence of T triples $(\mathrm{qu}_1, \mathrm{ans}_1, \mathrm{ma}_1), \ldots, (\mathrm{qu}_T, \mathrm{ans}_T, \mathrm{ma}_T)$ such that for every $1 \le i \le T$ the following holds:*

- *The* query qu_i *is an index in $1, \ldots, n$.*
- *The* answer ans_i *is either an 'a' or a parenthesis.*
- *The* matching information ma_i *is either NO-MATCH or a pair $(\mathrm{qu}_{i'}, \pi_i)$ where $i' < i$, and $\pi_i \in \{1, \ldots, n/4\}$. In the latter case $(\mathrm{qu}_{i'}, \mathrm{qu}_i) \in$ MATCHED, with the associated parenthesis index $\pi_s(\mathrm{qu}_{i'}) = \pi_s(\mathrm{qu}_i) = \pi_i$. In the former case there is no $\mathrm{qu}_{i'}$, $i' < i$ such that $(\mathrm{qu}_{i'}, \mathrm{qu}_i) \in$ MATCHED. In particular, if $\mathrm{ans}_i =$ 'a', then necessarily $\mathrm{ma}_i =$ NO-MATCH. Furthermore, the pairs in the subset* MATCHED *are disjoint and the parenthesis indices π_i of pairs in* MATCHED *are distinct.*

We note that if for some i the matching information ma_i is NO-MATCH then it only means that qu_i is not matched to any *previous* query $\mathrm{qu}_{i'}$ where $i' < i$. It is possible that there may be a subsequent query $\mathrm{qu}_{i''}$ where $i'' > i$ such that $(\mathrm{qu}_i, \mathrm{qu}_{i''}) \in$ MATCHED.

Definition 17 (Compatibility) *We say that a string s of length n and a history $h = (\mathrm{qu}_1, \mathrm{ans}_1, \mathrm{ma}_1), \ldots, (\mathrm{qu}_T, \mathrm{ans}_T, \mathrm{ma}_T)$ of length T are* compatible *if the following holds:*

1. *For every $1 \le i \le T$, $s_{\mathrm{qu}_i} = \mathrm{ans}_i$;*
2. *If $\mathrm{ma}_i = (\mathrm{qu}_{i'}, \pi_i)$ for $i' < i$, then $\pi_s(\mathrm{qu}_i) = \pi_s(\mathrm{qu}_{i'}) = \pi_i$.*
3. *If $\mathrm{ma}_i =$ NO-MATCH then for every $i' < i$ such that $\mathrm{ans}_{i'}$ is a parenthesis, $\pi_s(\mathrm{qu}_i) \ne \pi_s(\mathrm{qu}_{i'})$.*

The set of strings in the support of POS_n that are compatible with h is denoted by $S(h)$.

Given a history $h = (\mathrm{qu}_1, \mathrm{ans}_1, \mathrm{ma}_1), \ldots, (\mathrm{qu}_T, \mathrm{ans}_T, \mathrm{ma}_T)$ of length T, let qu_{T+1} be a new query. We would like to determine the distribution on ans_{T+1} and ma_{T+1}, conditioned on the history h. Since POS_n is uniform over its support, the response provided by the process is determined by the relative fraction of strings in $S(h)$ that are consistent with each possible response. We thus partition $S(h)$ into disjoint subsets as follows:

Definition 18 *For a given history h of length T, let $S^a(h, \mathrm{qu}_{T+1})$ denote the subset of all strings $s \in S(h)$ such that $s_{q_{T+1}} =$ 'a', and let $S^{par}(h, \mathrm{qu}_{T+1})$ denote the subset of all strings $s \in S(h)$ such that $s_{q_{T+1}}$ is a parenthesis.*

For every $1 \le \pi \le n/4$ and $1 \le i \le T$, let $S^{\pi, \mathrm{qu}_i}(h, \mathrm{qu}_{T+1})$ denote the subset of all strings $s \in S^{par}(h, \mathrm{qu}_{T+1})$ such that $\pi_s(\mathrm{qu}_{T+1}) = \pi_s(\mathrm{qu}_i) = \pi$, and let $S^{no\text{-}match}(h, \mathrm{qu}_{T+1})$ denote the subset of all strings $s \in S^{par}(h, \mathrm{qu}_{T+1})$ such that $\pi_s(\mathrm{qu}_{T+1}) \ne \pi_s(\mathrm{qu}_i)$ for every $1 \le i \le T$.

Note that there may exist $1 \le \pi \le n/4$ and $1 \le i \le T$, such that the set $S^{\pi, \mathrm{qu}_i}(h, \mathrm{qu}_{T+1})$ is empty due to the compatibility requirement with h.

The Distribution on $\mathcal{P}_{\mathrm{POS}}$'s answers. Given the above definition, the probability that ans_{T+1} is an 'a' is $|S^a(h, \mathrm{qu}_{T+1})|/|S(h)|$, and the probability that it is a parenthesis is $|S^{par}(h, \mathrm{qu}_{T+1})|/|S(h)|$. Conditioned on it being a parenthesis, $\mathcal{P}_{\mathrm{POS}}$ needs to determine its type, and it needs to determine ma_{T+1} (if ans_{T+1} is 'a' then necessarily ma_{T+1} is NO-MATCH).

For each such qu_i, where $1 \le i \le T$, and for each $1 \le \pi \le n/4$, the probability that $\mathrm{ma}_{T+1} = (\mathrm{qu}_i, \pi)$ is $|S^{\pi, \mathrm{qu}_i}(h, \mathrm{qu}_{T+1})|/|S^{par}(h, \mathrm{qu}_{T+1})|$. The probability that $\mathrm{ma}_{T+1} =$ NO-MATCH, conditioned on position qu_{T+1} being a parenthesis, is $|S^{no\text{-}match}(h, \mathrm{qu}_{T+1})|/|S^{par}(h, \mathrm{qu}_{T+1})|$. Finally, after determining ma_{T+1} the process can determine ans_{T+1}: If $\mathrm{ma}_{T+1} = (\mathrm{qu}_i, \pi)$ for some $1 \leq i \leq T$, then ans_{T+1} is a parenthesis of the same type as ans_i. If $\mathrm{ma}_{T+1} =$ NO-MATCH then one of the two types of parentheses is selected with equal probability.

4.2.2 The Process $\mathcal{P}_{\mathrm{NEG}}$.

The process $\mathcal{P}_{\mathrm{NEG}}$ is almost identical to $\mathcal{P}_{\mathrm{POS}}$. Here too, for every history h of length T and a new query qu_{T+1}, $\mathcal{P}_{\mathrm{NEG}}$ considers the set $S(h)$ of strings *in the support of* NEG_n that are compatible with h, and the corresponding subsets $S^a(h, \mathrm{qu}_{T+1})$ and $S^{\pi,\mathrm{qu}_i}(h, \mathrm{qu}_{T+1}) \subset S^{par}(h, \mathrm{qu}_{T+1})$, which are defined analogously to the way that they were defined above. Given these subsets, the probability that the answer ans_{T+1} is set to 'a' or is a parenthesis whose type is yet to be determined, is the same as described for $\mathcal{P}_{\mathrm{POS}}$, and the same holds for the setting of ma_{T+1}. The difference between the two processes is in the choice of the type of parenthesis, in case the process decides that ans_{T+1} is a parenthesis that is matched to a previous query. Suppose that $\mathrm{ma}_{T+1} = (\mathrm{qu}_i, \pi)$ for some $1 \leq i \leq T$. Then the setting of ans_{T+1} depends on π: If π is non-significant, then ans_{T+1} is of the same type as ans_i, and if π is significant, then one of the two types of parentheses is selected with equal probability (as in the case of NO-MATCH).

Thus the two processes differ only in the way they answer queries whose position is matched to a previously answered query, and the common parenthesis index is significant. Therefore, conditioned on the history containing no such match, the two corresponding distributions on query-answer histories are exactly the same.

4.3 Interacting with $\mathcal{P}_{\mathrm{POS}}$ and $\mathcal{P}_{\mathrm{NEG}}$

The next lemma is central to the proof of Theorem 2. In the lemma and in all that follows we assume that the testing algorithm $\mathcal{A}$ receives, for each query qu_i it asks, not only the answer ans_i but also the matching information ma_i. Clearly, any lower bound that holds under this assumption also holds when the algorithm is not provided with this extra information.

Lemma 17 *Let $\mathcal{A}$ be an algorithm that asks at most $\alpha n^{1/11}/\log n$ queries for $\alpha \leq e^{-7}$ and is provided, for each query qu_i, with an answer ans_i and the matching information ma_i, generated by $\mathcal{P}_{\mathrm{POS}}$ (similarly, $\mathcal{P}_{\mathrm{NEG}}$). Consider the distribution on query-answer histories $h = (\mathrm{qu}_1, \mathrm{ans}_1, \mathrm{ma}_1), \ldots, (\mathrm{qu}_T, \mathrm{ans}_T, \mathrm{ma}_T)$ for $T \leq \alpha n^{1/11}/\log n$, that is induced by the random decisions of $\mathcal{A}$ and $\mathcal{P}_{\mathrm{POS}}$ (similarly, $\mathcal{P}_{\mathrm{NEG}}$). Then the probability that there exists an index $1 \leq i \leq T$ such that $ma_i = (\mathrm{qu}_{i'}, \pi)$ where $i' < i$ and π is a significant parenthesis index, is at most $1/4$.*

Proof: We shall refer to a match as described in the lemma, as a *successful match*. Since as long as a successful match does not occur, the two processes $\mathcal{P}_{\mathrm{POS}}$ and $\mathcal{P}_{\mathrm{NEG}}$ behave exactly the same, it suffices to prove the lemma for one of them. Let this process be $\mathcal{P}_{\mathrm{POS}}$.

We shall break the interaction between $\mathcal{A}$ and $\mathcal{P}_{\mathrm{POS}}$ into *phases*. A phase ends whenever the process responds with a match between the newly queried position and a previously queried position. We may assume, without loss of generality, that once the algorithm views a match between positions $1 \leq j \leq n/2$ and $n/2+1 \leq k \leq n$ with parenthesis index $\pi \leq n/16$, then it does not ask any additional queries in the intervals $[j, n/2]$ and $[n/2+1, k]$. Similarly, if the match has parenthesis index $\pi > n/4 - n/16$, then the algorithm does not ask any additional queries in the intervals $[1, j]$ and $[k, n]$.

Hence, as long as a successful match does not occur, at the end of each phase we either have a new match $\pi \leq n/16$ that is greater than any previous match $\pi' \leq n/16$, or we have a new match $\pi > n/4 - n/16$ that is smaller than any previous match $\pi' > n/4 - n/16$. We next define the progress that a new query can make in terms of getting a new match that is closer to the significant range $[n/16+1, n/4-n/16]$.

Definition 19 (Progress) *Let $h = (\mathrm{qu}_1, \mathrm{ans}_1, \mathrm{ma}_1), \ldots, (\mathrm{qu}_T, \mathrm{ans}_T, \mathrm{ma}_T)$ be a given history of length T that does not contain a match in the significant range, and let $\pi_0(h)$ be the maximum over all $\pi_i \leq n/16$ such that $\mathrm{ma}_i = (\mathrm{qu}_{i'}, \pi_i)$ for some $i' < i$. If no such match exists then $\pi_0(h) = 0$. Similarly, let $\pi'_0(h)$ be the minimum over all $\pi_i > n/4 - n/16$ such that $\mathrm{ma}_i = (\mathrm{qu}_{i'}, \pi_i)$, where if no such match exists then $\pi'_0(h) = n/4+1$. We say that a new query qu_{T+1} makes* progress *x, for some integer x, if:*

1. $\mathrm{ans}_{T+1} \in \{(, [,),]\}$ *for some $j \leq T$ (the new query is answered by a parenthesis).*
2. $\mathrm{ma}_{T+1} = (\mathrm{qu}_j, \pi_{T+1})$ *for some $1 \leq j \leq T$ (the new query is matched to a previously queried position), where $\pi_{T+1} \geq \pi_0(h) + x$ and $\pi_{T+1} \leq \pi'_0(h) - x$.*

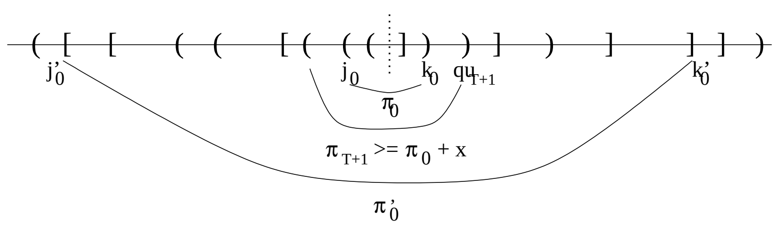

Figure 5: An illustration for Definition 19 and Claim 17.1. The new query, qu_{T+1} is matched to an opening parenthesis on the left side of the string. Here j_0, k_0 and π_0, stand for $j_0(h)$, $k_0(h)$ and $\pi_0(h)$, respectively.

The following claim is central to the proof of Lemma 17, and will be proved subsequently.

Claim 17.1 *Let $h = (\mathrm{qu}_1, \mathrm{ans}_1, \mathrm{ma}_1), \ldots, (\mathrm{qu}_T, \mathrm{ans}_T, \mathrm{ma}_T)$ be a query-answer history h of length $T < \alpha n^{1/11}/\log n$ and let $\pi_0(h)$ and $\pi'_0(h)$ be as in Definition 19. Let $(j_0(h), k_0(h))$ and $(j'_0(h), k'_0(h))$, where $j_0(h), j'_0(h) \leq n/2$ and $k_0(h), k'_0(h) > n/2$, be the corresponding pairs of matched queried positions having parenthesis index $\pi_0(h)$ and $\pi'_0(h)$, respectively. Suppose that $\pi_0(h) \leq n/(4\log n)$, and that $\pi_0(h)$ does not deviate by more than $\sqrt{\log n \cdot \min\{(n/2 - j_0(h)), j_0(h)\}}$ from $(n/2 - j_0(h))/2$ and by more than $\sqrt{\log n \cdot \min\{(k_0(h) - n/2), (n - k_0(h))\}}$ from $(k_0(h) - n/2)/2$. Suppose that an analogous bounds hold for $\pi'_0(h)$. Then for any possible new query $\mathrm{qu}_{T+1} \in \{1, \ldots, n\}$, the probability that the new query makes progress at least $n^{10/11}$, is at most $n^{-1/11}$.*

Completing the proof of Lemma 17. Recall that as stated in Claim 16, for every algorithm $\mathcal{A}$, the distribution on answers provided by $\mathcal{P}_{\mathrm{POS}}$ is identical to those obtained from querying a string that is randomly generated according to POS_n. By Claim 15, the probability that a string generated by POS_n does not obey the inequalities in Claim 15 is at most $1/8$. Hence, for any length of interaction, the probability that there exists a stage at which either $\pi_0(h)$ or $\pi'_0(h)$, as determined in that stage for the current history h, deviate by more than the claim allows from their expected values, is at most $1/8$. Conditioned on such an event not occurring, we can apply Claim 17.1 as long as $\pi_0(h) \leq n/(4\log n)$ and $\pi'_0(h) \geq n/4 - n/(4\log n)$. If the algorithm performs at most $\alpha n^{1/11}/\log n$ queries, and in each it in fact makes progress of at most $n^{10/11}$, then $\pi_0(h)$ and $\pi'_0(h)$ will be as required by Claim 17.1 prior to each query. Hence, by applying Claim 15 and

Claim 17.1, if the algorithm asks at most $\alpha n^{1/11}/\log n$ queries, then the probability that it obtains a successful match is at most

$$1/8 + (\alpha n^{1/11}/\log n) \cdot n^{-1/11} < 1/8 + \alpha < 1/4.$$

Lemma 17 follows. ■

It thus remains to prove Claim 17.1.

4.3.1 An Intuitive Discussion of the Validity of Claim 17.1

Assume first, without loss of generality, that the following conditions hold:

1. $\mathrm{qu}_{T+1} > n/2$, and in particular, $k_0(h) + n^{10/11} \le \mathrm{qu}_{T+1} \le k'_0(h) - n^{10/11}$ (or else clearly the algorithm cannot make sufficient progress).

2. $\mathrm{qu}_{T+1} - k_0(h) \le k'_0(h) - \mathrm{qu}_{T+1}$ so that qu_{T+1} is closer to $k_0(h)$ than to $k'_0(h)$.

The probability that qu_{T+1} makes progress of at least $n^{10/11}$ is the probability, conditioned on a string s that is generated according to POS_n being compatible with h, that for some query-position $\mathrm{qu}_i < j_0(h)$, we have $\pi_s(\mathrm{qu}_i) = \pi_s(\mathrm{qu}_{T+1})$.

In order to bound this probability, suppose that we generate s by first randomly selecting the set L of all $n/4$ parentheses positions on the left half of the string, in a manner consistent with the history h. Each such choice of L determines the parentheses indices of all *queries* on the left half of the string that were answered by parentheses. Denote the set of parentheses indices corresponding to the query positions by $\Pi(L)$. Next we consider the selection of the parentheses positions on the right half of the string, once again in a manner consistent with the history h. In particular, in order to be consistent, the number of parentheses positions selected between $k_0(h)$ and $k'_0(h)$ is exactly $n/4 - (\pi_0(h) + \pi'_0(h))$.

Fixing L, consider each index $\pi \in \Pi(L)$, where there are at most $\alpha n^{1/11}/\log n$ such indices. The probability that $\pi_s(\mathrm{qu}_{T+1}) = \pi$, taken over the selection of parentheses positions on the right half of the string, is the probability that there are *exactly* $\pi - \pi_0(h)$ parentheses between $k_0(h)$ and qu_{T+1} (including qu_{T+1}), and exactly $\pi'_0(h) - \pi$ parentheses between qu_{T+1} and $k'_0(h)$.

For the sake of this discussion, let us now make the following simplifying assumption by which we shall lose generality. Suppose that there is no query qu_i, $1 \le i \le T$ such that $k_0(h) < \mathrm{qu}_i < k'_0(h)$. That is, qu_{T+1} is the first query in this region. Consider in this case the selection of parentheses positions on the right side of the string, and in particular the selection of $n/4 - (\pi_0(h) + \pi'_0(h))$ positions between $k_0(h)$ and $k'_0(h)$. Since there is no conditioning on the way these parentheses are allowed to be distributed (as there are no other queries in this region), it is not very hard to verify that the probability that there are $\pi - \pi_0(h)$ parentheses between $k_0(h)$ and qu_{T+1} and $\pi'_0(h) - \pi$ parentheses between qu_{T+1} and $k'_0(h)$ is relatively small. In particular, it is of the order of $1/\sqrt{(\mathrm{qu}_{T+1} - k_0(h))} \le n^{-5/11}$.

In general there may be up to $\alpha n^{1/11}/\log n$ queried positions between $k_0(h)$ and $k'_0(h)$ that in particular may contain parentheses, and we must select the string s conditioned on these positions *not matching* any queried position on the left hand side. Hence our argument is more complicated.

4.3.2 Proof of Claim 17.1

We need to show that among all strings compatible with the given query-answer history h, the fraction of strings in which the new query qu_{T+1} makes progress of at least $n^{10/11}$ is at most

$n^{-1/11}$. Recall that we assume that the history h does not contain any match in the significant region. We may assume without loss of generality that $\mathrm{qu}_{T+1} > n/2$ and that $\mathcal{P}_{\mathrm{POS}}$ decides that this position should contain a parenthesis (or else clearly no progress is made). In order to simplify our presentation, we also assume that $\pi'_0(h) = n/4 + 1$, that is, the history does not contain any match $\pi_i > n/4 - n/16$. It is not hard to verify that while this simplifies the already cumbersome notation involved, removing the assumption does not change the essence of the argument.

For a given history h and a new query $\mathrm{qu}_{T+1} > n/2$, consider all the strings that are compatible with h and have a parenthesis in position qu_{T+1}. That is, consider the set $S^{par}(h, \mathrm{qu}_{T+1})$ as defined in Definition 18. Then

$$\Pr[\mathrm{qu}_{T+1} \text{ makes progress } n^{10/11} \mid h] \le \frac{\sum_{\pi \ge \pi_0(h)+n^{10/11}} \sum_{1\le i\le T} |S^{\pi,\mathrm{qu}_i}(h, \mathrm{qu}_{T+1})|}{|S^{par}(h, \mathrm{qu}_{T+1})|}, \tag{14}$$

where $S^{\pi,\mathrm{qu}_i}(h, \mathrm{qu}_{T+1})$ is also defined in Definition 18. To this end it will be convenient to use a finer partition of $S^{par}(h, \mathrm{qu}_{T+1})$, since it will be easier for us to relate the sizes of the subsets in this partition. In particular, the strings within each subset have the following in common: The subset L of $n/4$ parentheses positions in the left half of each string is the same for all strings in the subset. Furthermore, the substring $s_{j_0(h),k_0(h)}$, where $j_0(h)$ and $k_0(h)$ are as defined in Claim 17.1, is also the same for all strings in the subset. A formal definition follows.

Definition 20 *Let h, qu_{T+1}, $\pi_0(h)$, $j_0(h)$, and $k_0(h)$ be as defined in Claim 17.1, and assume that $\mathrm{qu}_{T+1} > n/2$ and $\pi'_0(h) = n/4 + 1$. Let w be a fixed substring of length $k_0(h) - j_0(h) + 1$, and let $L' \subset \{1, \ldots, j_0(h) - 1\}$, $|L'| = n/4 - \pi_0(h)$ be a subset of parentheses positions. We define $S^{par}(h, \mathrm{qu}_{T+1}, w, L')$ to be the subset of all strings in $S^{par}(h, \mathrm{qu}_{T+1})$ such that:*

1. *$s_{j_0(h),k_0(h)} = w$;*
2. *For every $j \in L'$, s_j is a parenthesis, and for every $j \in \{1, \ldots, j_0(h) - 1\} \setminus L'$, the symbol s_j is an 'a'.*

Note that there may exist L' and w for which $S^{par}(h, \mathrm{qu}_{T+1}, w, L')$ is empty.

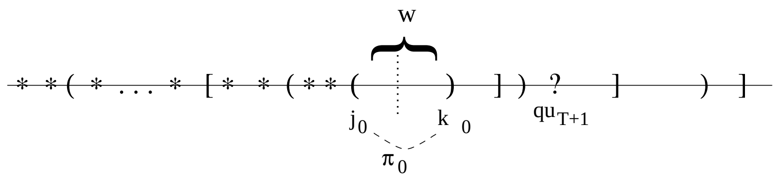

Figure 6: An illustration for Definition 20. The asterisks on the left half of the string represent the selected positions in L', that include in particular all queried positions that were answered by parentheses. The question mark on the right represents the position of the new query, qu_{T+1}. Here j_0, k_0 and π_0, stand for $j_0(h)$, $k_0(h)$ and $\pi_0(h)$, respectively.

Observe that for a fixed w and L' and for every $\mathrm{qu}_i \le n/2$, such that ans_i is a parenthesis, $\pi_s(\mathrm{qu}_i) = \pi_{s'}(\mathrm{qu}_i)$ for every $s, s' \in S^{par}(h, \mathrm{qu}_{T+1}, w, L')$. This is clearly true for every $j_0(h) \le \mathrm{qu}_i \le n/2$, since $s_{j_0(h),n/2}$ is completely determined. As for $\mathrm{qu}_i < j_0(h)$, the parenthesis index of qu_i is simply $\pi_0(h) + |\{j \in L',\ j \ge \mathrm{qu}_i\}|$. In the next definition we consider subsets of $S^{par}(h, \mathrm{qu}_{T+1}, w, L')$ in which for all strings s in the subset, $\pi_s(\mathrm{qu}_{T+1}) = \pi_s(\mathrm{qu}_i)$ for some $\mathrm{qu}_i < j_0(h)$ (as determined by L' and π_0).

Definition 21 *Let h, qu_{T+1}, $\pi_0(h)$, $j_0(h)$, and $k_0(h)$ be as defined in Claim 17.1, and assume that $\mathrm{qu}_{T+1} > n/2$ and $\pi'_0(h) = n/4+1$. Let w and L' be as in Definition 20. We denote by $\Pi(L')$ the set of parentheses indices of query-positions $\mathrm{qu}_i < j_0(h)$ that is induced by L' (and $\pi_0(h)$). For each $\pi \in \Pi(L')$, let $S^{\pi}(h, \mathrm{qu}_{T+1}, w, L') \subset S^{par}(h, \mathrm{qu}_{T+1}, w, L')$ be the subset of strings in $S^{par}(h, \mathrm{qu}_{T+1}, w, L')$ such that position qu_{T+1} has parenthesis index π.*

Note that by definition of $\Pi(L')$, each $\pi \in \Pi(L')$ corresponds to a unique query position $\mathrm{qu}_i < j_0(h)$. Recall that $S^{\pi,\mathrm{qu}_i}(h, \mathrm{qu}_{T+1})$ denotes the set of all strings compatible with h in which qu_{T+1} is matched with qu_i and both are assigned parenthesis index π. Hence, for each $\pi \in \Pi(L')$ there exists a unique qu_i such that $S^{\pi}(h, \mathrm{qu}_{T+1}, w, L') \subset S^{\pi,\mathrm{qu}_i}(h, \mathrm{qu}_{T+1})$.

Claim 17.2 *For every h, qu_{T+1}, w, and L' as in Definition 20, and for every $\pi \in \Pi(L')$, $\pi \geq \pi_0(h) + n^{10/11}$,*

$$\frac{|S^{\pi}(h, \mathrm{qu}_{T+1}, w, L')|}{|S^{par}(h, \mathrm{qu}_{T+1}, w, L')|} < \frac{1}{\alpha} \cdot n^{-2/11}.$$

Before proving Claim 17.2, we apply it to obtain Claim 17.1. By definition, $S^{par}(h, \mathrm{qu}_{T+1}) = \bigcup_{L',w} S^{par}(h, \mathrm{qu}_{T+1}, w, L')$. The subset of strings in $S^{par}(h, \mathrm{qu}_{T+1})$ in which qu_{T+1} makes progress $n^{10/11}$, is the union over $\pi \geq \pi_0(h) + n^{10/11}$ and over all $\mathrm{qu}_i < T$ of the sets $S^{\pi,\mathrm{qu}_i}(h, \mathrm{qu}_{T+1})$. This in turn (by our observation following Definition 21), is equivalent to the union over all w, L' and $\pi \in \Pi(L')$ such that $\pi \geq \pi_0(h) + n^{10/11}$, of the sets $S^{\pi}(h, \mathrm{qu}_{T+1}, w, L')$. Since these sets are disjoint and since $|\Pi(L')| < \alpha n^{1/11}/\log n$, using Equation (14) we get:

$$\begin{aligned}
&\Pr[\mathrm{qu}_{T+1} \text{ makes progress } n^{10/11} \mid h] \\
\leq\; & \frac{\sum_{L',w} \sum_{\pi \in \Pi(L'),\, \pi \geq \pi_0(h)+n^{10/11}} \left|S^{\pi}(h, \mathrm{qu}_{T+1}, w, L')\right|}{\sum_{L',w} \left|S^{par}(h, \mathrm{qu}_{T+1}, w, L')\right|} \\
\leq\; & \max_{L',w} \frac{\sum_{\pi \in \Pi(L'),\, \pi \geq \pi_0(h)+n^{10/11}} \left|S^{\pi}(h, \mathrm{qu}_{T+1}, w, L')\right|}{\left|S^{par}(h, \mathrm{qu}_{T+1}, w, L')\right|} \\
\leq\; & |\Pi(L')| \cdot \frac{1}{\alpha} \cdot n^{-2/11} \quad < \quad n^{-1/11}.
\end{aligned} \tag{15}$$

Claim 17.1 thus follows from Claim 17.2. ■

4.3.3 Proof of Claim 17.2

Since h, qu_{T+1}, w and L' are fixed, we remove them from our notation. Namely we let $\pi_0 = \pi_0(h)$, $j_0 = j_0(h)$, $k_0 = k_0(h)$, $\Pi = \Pi(L')$, $S = S^{par}(h, \mathrm{qu}_{T+1}, w, L')$, and $S^{\pi} = S^{\pi}(h, \mathrm{qu}_{T+1}, w, L')$. Since the claim should hold for every $\pi \in \Pi$, $\pi \geq \pi_0(h) + n^{10/11}$, let us fix such a π. Recall that we have assumed without loss of generality that $\mathrm{qu}_{T+1} > n/2$.

We start with a description of the underlying idea of the proof. One basic approach to proving that S^{π} is relatively small with respect to S, is to define a one-to-many mapping from strings in S^{π} to relatively large subsets of strings in S. The mapping should be such that different strings in S^{π} are mapped to different disjoint subsets of S. Our argument will be in similar vein: Instead of mapping strings to subsets of strings, we do the following. We first partition S into disjoint subsets, such that each subset U in the partition is either contained in S^{π} or is contained in $S \setminus S^{\pi}$. We then map each subset $U \subset S^{\pi}$ in this partition to a relatively large *collection of disjoint subsets* $\{U_i\}$ of S. We shall show that:

1. For every such subset $U \subset S^{\pi}$ and for all but a small fraction of the U_i's in the corresponding collection $\{U_i\}$, the size of each U_i is of the same order as the size of U.

2. There exists a family $\mathcal{U}$ of subsets $U \subset S^{\pi}$ in the partition, such that

   (a) For every two subsets $U, U' \in \mathcal{U}$, the respective collections $\{U_i\}$ and $\{U'_i\}$ are disjoint (that is, for every i, j, $U_i \cap U'_j = \emptyset$).

   (b) $\left|\bigcup_{U \in \mathcal{U}} U\right|$ is relatively large compared to all of S^{π}.

More details follow.

Defining the Partition of S. Let $k_1 < \ldots < k_r$ be all positions of queries in h including qu_{T+1} that are greater than k_0 and were answered by parentheses, where $\mathrm{qu}_{T+1} = k_t$ for some $1 \le t \le r$. We assume without loss of generality that $k_t - k_0 \le n - k_t$ (the case in which k_t is closer to n than to k_0 is symmetric). Recall that k_0 is the queried position on the right half of the string having the largest matched parenthesis index where the matched position is $j_0 \le n/2$. Hence, with the possible exception of k_t, no $k_i > k_0$ is matched to any queried position $\mathrm{qu}_\ell \le n/2$. Since Π is the set of all parentheses indices of queried positions $\mathrm{qu}_\ell < j_0$ that is induced by L', we have that for every string $s \in S$, for every $k_i \ne k_t$ and for every $\pi \in \Pi$, $\pi_s(k_i) \ne \pi$.

We partition S into disjoint subsets according to the number of parentheses between every two positions k_{i-1} and k_i. For $i = 1, \ldots, r+1$, let $b_i = k_i - k_{i-1} - 1$ be the number of positions strictly between k_{i-1} and k_i, and let q_i be the number of queries between k_i and k_{i-1}. That is, $q_i = |\{\mathrm{qu}_j : \; k_{i-1} < \mathrm{qu}_j < k_i\}|$. Since the k_i's were defined to be the query positions that were answered by parentheses, all the query positions strictly between k_{i-1} and k_i were necessarily answered by an 'a'.

Given the above notations b_i and q_i, for every string $s \in S$ and for every $1 \le i \le r+1$, the number of parentheses between positions k_{i-1} and k_i ranges between 0 and $b_i - q_i$. Recall that each such string contains a total of $n/4$ parentheses among positions $n/2+1, \ldots, n$, where there are π_0 parentheses among positions $1, \ldots, k_0$, and r parentheses in positions $k_1, \ldots, k_r$. Hence the total number of parentheses between the k_i's is $n/4 - r - \pi_0$. We shall partition the strings in S according to the number of parentheses they have between every consecutive k_{i-1} and k_i. Specifically, consider any sequence $D = d_1, \ldots, d_{r+1}$ that satisfies the following constraints:

C1. For every $1 \le i \le r+1$, we have $0 \le d_i \le b_i - q_i$;

C2. $\sum_{i=1}^{r+1} d_i = n/4 - r - \pi_0$.

Let $S(D)$ denote the subset of strings in S such that for every $1 \le i \le r+1$, there are exactly d_i parentheses in positions $k_{i-1} < k < k_i$.

Note that given π_0, the sequence D determines the parenthesis index of every k_i, and in particular of k_t. Specifically, the parenthesis index of k_j in every string in $S(D)$ is $\pi_0 + \sum_{i \le j}(d_i + 1)$. The reason we add 1 to each d_i, $i \le j$, is that we need to account for the parentheses in the queried positions $k_1, \ldots, k_j$, where d_i is the number of parentheses strictly between these positions. Thus, if D determines that the parenthesis index of some $k_j \ne k_t$ is in Π, then $S(D)$ is empty, since there is no string compatible with h such that the number of parentheses between the k_i's is as D designates. Otherwise, $S(D)$ is non-empty, since all other compatibility requirements are obeyed. In this case either $S(D) \subseteq S^{\pi}$ if $\pi_0 + \sum_{1 \le i \le T}(d_i + 1) = \pi$, or $S(D) \subset S \setminus S^{\pi}$.

If $S(D)$ is non-empty, then the number of strings in $S(D)$ depends on: (1) The number of ways to select d_i positions for parentheses among the $b_i - q_i$ available positions between k_{i-1} and k_i for every $1 \le i \le r+1$, (2) The number of ways to set the types of the parentheses in the selected positions (that do not correspond to previous queries) on both sides of the string.

Specifically, for any fixed sequence $D = d_1, \ldots, d_{r+1}$ that satisfies conditions C1 and C2, the total number of ways to select d_i positions for parentheses between k_{i-1} and k_i, for every $1 \le i \le r+1$, is simply $\prod_{i=1}^{r+1} \binom{b_i - q_i}{d_i}$. Recall that the set of positions L' of all parentheses positions to the left of j_0 is already fixed for all strings in S, and that assuming $S(D)$ is non-empty, the parenthesis position of each k_i, $i \neq t$ *differs* from the parenthesis position of each of the $|\Pi|$ positions $\mathrm{qu}_i < j_0$ that were answered by a parenthesis. Therefore, the number of ways to set the types of parentheses in the selected positions to the left of j_0 and to the right of k_0 (including k_t) is either $2^{n/4-(\pi_0+|\Pi|+r)}$ or $2^{n/4-(\pi_0+|\Pi|+r-1)}$. The first value corresponds to the case in which the parenthesis index of k_t is not in Π (and so the type of parenthesis in position k_t is not determined), and the second value to the case in which the parenthesis index of k_t belongs to Π. Given the above discussion,

$$\left(\prod_{i=1}^{r+1} \binom{b_i - q_i}{d_i}\right) \cdot 2^{(n/4-\pi_0)-(|\Pi|+r)} \quad \le \quad |S(D)| \quad \le \quad \left(\prod_{i=1}^{r+1} \binom{b_i - q_i}{d_i}\right) \cdot 2^{(n/4-\pi_0)-(|\Pi|+r-1)} \tag{16}$$

Defining the One-to-Many Mapping from each $S(D) \subset S^\pi$ to subsets in S. Consider any fixed sequence $D = d_1, \ldots, d_{r+1}$ that satisfies conditions C1 and C2 and such that $S(D)$ is non-empty and $S(D) \subset S^\pi$. Let $m = n^{4/11}$, and suppose that there exist two indices, $1 \le u \le t$ and $t+1 \le v \le r+1$ such that $d_u \le b_u - q_u - m$ and $d_v \ge m$. Then, for every $1 \le g \le m$, we consider the subset of all strings that result from taking a string in $S(D)$ and "moving" g parentheses from the interval between k_{v-1} and k_v to the interval between k_{u-1} and k_u. More precisely, for each such g we define the subset $S(D_g)$ of strings that correspond to the sequence

$$D_g = d_1, \ldots, d_{u-1}, \underline{d_u + g}, d_{u+1}, \ldots, d_{v-1}, \underline{d_v - g}, d_{v+1}, \ldots, d_{r+1}, \tag{17}$$

where the d_i's on which D and D_g differ are underlined. As we shall show momentarily, all but a relatively small number of these $m = n^{4/11}$ subsets are non-empty. Furthermore, under somewhat stronger conditions on d_u and d_v, each of these subsets is not much smaller than $S(D)$. Finally we show that for every $D' \neq D$ such that $S(D') \subset S^\pi$, the subsets D'_g that are defined analogously to the D_g's in Equation (17) are all disjoint from the D_g's. More details are next provided. Recall that D is fixed, and so the following holds for every D such that $S(D) \subset S^\pi$.

Properties of the Mapping from D to the D_g's.

P1. For all but at most $(r-1) \cdot |\Pi| < \alpha^2 n^{2/9}/\log^2 n$ of the D_g's, $S(D_g) \neq \emptyset$.

This is true since for every $k_i \neq k_t$, the number of indices g such that $\pi_s(k_i) \in \Pi$ for some string $s \in S(D_g)$, is at most $|\Pi| < \alpha n^{1/11}/\log n$, and the number of k_i's is $r - 1 < \alpha n^{1/11}/\log n$.

P2. For every $D' \neq D$ such that $S(D')$ is not empty and $S(D') \subset S^\pi$, the sequences $D'_1, \ldots, D'_m$ all differ from $D_1, \ldots, D_m$.

To verify this, assume in contradiction that for $D' \neq D$, $D' = d'_1, \ldots, d'_r$, we have $D_g = D'_{g'}$. That is, $d_i = d'_i$ for every $i \neq u, v$; $d_u + g = d'_u + g'$; and $d_v - g = d'_v - g'$. But, since $\sum_{i=1}^{t} d_i = \sum_{i=1}^{t} d'_i = \pi - \pi_0 - t$, we have $g = g'$, and so $D = D'$.

P3. For every D_g such that $S(D_g)$ is non-empty, if

$$d_u + m \leq \frac{b_u - q_u}{2} + \sqrt{3(b_u - q_u)} \text{ and } d_v - m \geq \frac{b_v - q_v}{2} - \sqrt{3(b_v - b_v)} \tag{18}$$

then $|S(D_g)| \geq e^{-18}|S(D)|$.

To verify this, consider the ratio $|S(D_g)|/|S(D)|$. By Equation (16) this ratio equals at least

$$\frac{\binom{b_u - q_u}{d_u + g} \cdot \binom{b_v - q_v}{d_v - g}}{\binom{b_u - q_u}{d_u} \cdot \binom{b_v - q_v}{d_v}}$$

Let us lower bound $\binom{b_u - q_u}{d_u + g} / \binom{b_u - q_u}{d_u}$. A lower bound on $\binom{b_v - q_v}{d_v - g} / \binom{b_v - q_v}{d_v}$ is obtained similarly. If $d_u + g \leq (b_u - q_u)/2$ then $\binom{b_u - q_u}{d_u + g} > \binom{b_u - q_u}{d_u}$ and we are done. Otherwise, $d_u + g > (b_u - q_u)/2$, but by our assumption on d_u in Equation (18), we also know that $d_u + g \leq (b_u - q_u)/2 + \sqrt{3(b_u - q_u)}$. On the other hand, $\binom{b_u - q_u}{d_u} \leq \binom{b_u - q_u}{(b_u - q_u)/2}$, and so

$$\frac{\binom{b_u - q_u}{d_u + g}}{\binom{b_u - q_u}{d_u}} \geq \frac{\binom{b_u - q_u}{(b_u - q_u)/2 + \sqrt{3(b_u - q_u)}}}{\binom{b_u - q_u}{(b_u - q_u)/2}}$$

Let us denote $b_u - q_u$ by b. Then the expression we have is:

$$\begin{aligned}
\frac{\binom{b}{b/2 + \sqrt{3b}}}{\binom{b}{b/2}} &= \frac{\prod_{i=0}^{\sqrt{3b}-1}(b/2 - i)}{\prod_{i=1}^{\sqrt{3b}}(b/2 + i)} = \frac{\prod_{i=0}^{\sqrt{3b}-1}(1 - i \cdot (2/b))}{\prod_{i=1}^{\sqrt{3b}}(1 + i \cdot (2/b))} \\
&> \frac{\prod_{i=0}^{\sqrt{3b}-1}\exp(-3i/b))}{\prod_{i=1}^{\sqrt{3b}}\exp(3i/b)} = \exp\left(-(6/b)\sum_{i=1}^{\sqrt{3b}} i\right) > e^{-9}
\end{aligned} \tag{19}$$

Clearly the above can be extended to the case in which the roles of $u \leq t$ and $v > t$ are reversed: that is, D is such that

$$d_u - m \geq \frac{b_u - q_u}{2} - \sqrt{3(b_u - q_u)} \text{ and } d_v + m \leq \frac{b_v - q_v}{2} + \sqrt{3(b_v - b_v)}. \tag{20}$$

In this case the D_g's are defined the same as in Equation (17) except that d_u is *decreased* by g and d_v is *increased* by g.

Defining Families of Subsequences D. If there existed one fixed choice of u and v for which the constraints on d_u and d_v described in Equation (18) or in Equation (20) were valid for *every* D such that $S(D) \subset S^{\pi}$, then we would be essentially done with our proof. While this is not the case, we shall show that there exists a choice of u, v for which the sum of the sizes of the sets $S(D)$, such that D obeys the constraints in one of the two equations, is relatively large. This will suffice for our purposes. (Note that if we allow different choices of pairs (u, v) then the disjointness claim in Property P2 does not necessarily hold.)

Let

$$\mathcal{D}_{u,v}^{\leftarrow} \stackrel{\text{def}}{=} \{D : \text{ Equation (18) holds for } d_u \text{ and } d_v \ \} \tag{21}$$

and

$$\mathcal{D}_{u,v}^{\rightarrow} \stackrel{\text{def}}{=} \{D : \text{ Equation (20) holds for } d_u \text{ and } d_v \ \}. \tag{22}$$

Let

$$\mathcal{D}_{u,v} \stackrel{\text{def}}{=} \mathcal{D}^{\leftarrow}_{u,v} \cup \mathcal{D}^{\rightarrow}_{u,v} \quad \text{and} \quad \tilde{\mathcal{D}}_{u,v} \stackrel{\text{def}}{=} \{D_g : D \in \mathcal{D}_{u,v}\}.$$

Then by Properties P1–P3,

$$\begin{aligned}
\left| \bigcup_{D_g \in \tilde{\mathcal{D}}_{u,v}} S(D_g) \right| &= \sum_{D_g \in \tilde{\mathcal{D}}_{u,v}} |S(D_g)| = \sum_{D_g \in \tilde{\mathcal{D}}_{u,v}:\, S(D_g) \neq \emptyset} |S(D_g)| \\
&\geq \sum_{D \in \mathcal{D}_{u,v}} (m - \alpha^2 n^{2/9} / \log^2 n) \cdot e^{-18} \cdot |S(D)| \\
&\geq e^{-19} \cdot n^{4/11} \cdot \sum_{D \in \mathcal{D}_{u,v}} |S(D)|. \qquad (23)
\end{aligned}$$

Every D Belongs to at Least one Family $\mathcal{D}_{u,v}$. We next show that for every D such that $S(D) \subset S^{\pi}$, there exist $u \leq t$ and $v > t$ such that $D \in \mathcal{D}_{u,v}$. Let us fix D. By our assumption on the query-answer history (i.e., the deviation of π_0 from its expected value),

$$\left| \pi_0 - \frac{k_0 - n/2}{2} \right| < \sqrt{\log n \cdot \min\{(k_0 - n/2), (n - k_0)\}} = \sqrt{\log n (k_0 - n/2)} \qquad (24)$$

where in the equality we have used the assumption that k_0 is closer to $n/2$ than to n. Let

$$x = \frac{1}{2} \sum_{i=1}^{r+1} b_i - (n/4 - r - \pi_0).$$

What does x measure? Recall that $\sum_{i=1}^{r+1} b_i$ is the number of positions between $k_0 + 1$ and n that have not been queried, and amongst which it remains to select $\sum_{i=1}^{r+1} d_i = (n/4 - r - \pi_0)$ positions for parentheses. Let us refer to these positions as *undetermined*. Since the overall number of parentheses in the right half of the string is exactly half the total number of positions in that half, x measures the deviation from the expected value amongst the undetermined positions. By definition, $\sum_{i=1}^{r+1} b_i = n - r - k_0$. Therefore, $x = n/2 - r/2 - k_0/2 - n/4 + r + \pi_0$. Combining this equality with Equation (24) we get

$$-\sqrt{\log n (k_0 - n/2)} + r/2 \;<\; x \;<\; \sqrt{\log n (k_0 - n/2)} + r/2.$$

Let

$$x_1 = \frac{1}{2} \sum_{i=1}^{t} b_i - (\pi - t - \pi_0) \quad \text{and} \quad x_2 = \frac{1}{2} \sum_{i=t+1}^{r+1} b_i - (n/4 - (r - t) - \pi).$$

Recall that we are considering a setting D such that $S(D) \subset S^{\pi}$. That is, for every $s \in S(D)$ we have $\pi_s(k_t) = \pi$. In other words, there are $\pi - \pi_0 - t$ parentheses amongst the undetermined positions between $k_0 + 1$ and $k_t - 1$. Thus x_1 measures the deviation from the expectation of the number of parentheses amongst the undetermined positions between $k_0 + 1$ and $k_t - 1$. Similarly, x_2 measures the deviation from the expectation of the number of parentheses amongst the undetermined positions between $k_t + 1$ and n. By definition, $x_1 + x_2 = x$. We consider the following cases:

1. x_1 and x_2 have an opposite sign (or at least one of them is 0). That is, there is an "extra number" of parentheses between k_0+1 and k_t-1 and a "missing number" of parentheses between k_t+1 and n (amongst the undetermined positions and with respect to the expected numbers). Consider first the case that $x_1 \geq 0$ and $x_2 \leq 0$. Recall that $m = n^{4/11}$ and that $t \leq r \leq \alpha n^{1/11}/\log n$. Also note that by definition of x we have $\pi - t - \pi_0 = \frac{1}{2}\sum_{i=1}^t b_i - x_1$. Therefore,

$$\begin{aligned}
\sum_{i=1}^{t}(d_i+m) &= \left(\sum_{i=1}^{t} d_i\right) + t\cdot m \\
&\leq (\pi - t - \pi_0) + (\alpha n^{1/11}/\log n)\cdot n^{4/11} \\
&= \frac{1}{2}\sum_{i=1}^{t} b_i - x_1 + \alpha n^{5/11}/\log n \\
&\leq \frac{1}{2}\sum_{i=1}^{t} b_i + \alpha n^{5/11}/\log n & (25)\\
&= \frac{1}{2}\left(\sum_{i=1}^{t}(b_i-q_i)\right) + \frac{1}{2}\sum_{i=1}^{t} q_i + \alpha n^{5/11}/\log n \\
&\leq \frac{1}{2}\left(\sum_{i=1}^{t}(b_i-q_i)\right) + \sqrt{\sum_{i=1}^{t}(b_i-q_i)} & (26)
\end{aligned}$$

where Equation (25) follows from our assumption that $x_1 \geq 0$, and the last inequality is due to the fact that $\sum_{i=1}^t b_i \geq \pi - \pi_0 - t$ (or else there cannot be π parentheses up till position k_t), and so

$$\sum_{i=1}^{t}(b_i-q_i) \geq \pi - \pi_0 - \alpha n^{1/11}/\log n \geq n^{10/11} - \alpha n^{1/11}/\log n.$$

Similarly, using our assumption that $x_2 \leq 0$ we can obtain

$$\sum_{i=t+1}^{r+1}(d_i - m) \geq \frac{1}{2}\left(\sum_{i=1}^{t}(b_i-q_i)\right) - \sqrt{\sum_{i=1}^{t}(b_i-q_i)}. \qquad (27)$$

For $1 \leq i \leq t$, let $y_i = (d_i+m) - \frac{1}{2}(b_i-q_i)$. Then Equation (26) states that $\sum_{i=1}^t y_i \leq \sqrt{\sum_{i=1}^t (b_i-q_i)}$. Since $\sum_{i=1}^t y_i^2 \leq (\sum_{i=1}^t y_i)^2$, it follows that there must exist $u \leq t$ such that $y_u \leq \sqrt{b_u - q_u}$, or else $\sum_{i=1}^t y_i^2 > \sum_{i=1}^t (b_i-q_i) \geq (\sum_{i=1}^t y_i)^2$. That is, $d_u + m \leq \frac{1}{2}(d_u - q_u) + \sqrt{b_u-q_u}$. Similarly, it follows from Equation (26) that there exists $v > t$ such that $d_v - m \geq \frac{1}{2}(b_v - q_v) - \sqrt{b_v - q_v}$. Therefore, $D \in \mathcal{D}^{\leftarrow}_{u,v}$.

If $x_1 \leq 0$ and $x_2 \geq 0$, then we can similarly show that $D \in \mathcal{D}^{\rightarrow}_{u,v}$ for some $u \leq t$ and $v > t$.

2. x_1 and x_2 have the same sign. Consider the case that this sign is negative (the positive case is dealt with analogously). By definition of x and using Equation (24),

$$x \geq -\sqrt{\log n \cdot (k_0 - n/2)} \geq -\sqrt{\log n \cdot 2(\pi_0 + \sqrt{\log n \cdot (n-k_0)})}.$$

Recall that by one of the premises of Claim 17.2, $\pi_0 < n/(4/\log n)$, and so for a sufficiently large n we have that $x \geq -\sqrt{(n-k_0)}$. Since $x_1 + x_2 = x$, necessarily, either $x_1 \geq -\sqrt{(k_t - k_0)}$

or $x_2 \geq -\sqrt{(n-k_t)}$ (or both). If $x_1 \geq -\sqrt{(k_t - k_0)} = -\sqrt{t + \sum_{i=1}^{t} b_i}$, then by modifying Equation (26) so as to take into account this bound on x_1, we can obtain that

$$\begin{aligned}\sum_{i=1}^{t}(d_i + m) &\leq \frac{1}{2}\left(\sum_{i=1}^{t}(b_i - q_i)\right) + \sqrt{\sum_{i=1}^{t}(b_i - q_i) + \sqrt{t + \sum_{i=1}^{t} q_i + \sum_{i=1}^{t}(b_i - q_i)}} \\ &\leq \frac{1}{2}\left(\sum_{i=1}^{t}(b_i - q_i)\right) + \sqrt{3\sum_{i=1}^{t}(b_i - q_i)}\end{aligned}$$

and so there exists $u \leq t$ such that $d_u + m \leq \frac{1}{2}(b_u - q_u) + \sqrt{3(b_u - q_u)}$. On the other hand, using $x_2 < 0$ we can apply the same argument as in the previous item to get that there exists $v > t$ such that $d_v - m \geq \frac{1}{2}(b_v - q_v) - \sqrt{b_v - q_v}$, and hence $D \in \mathcal{D}^{\leftarrow}_{u,v}$.

Finishing the Proof of Claim 17.2. Finally, let $u_0 \leq t$ and $v_0 > t$ be such that $\sum_{D \in \mathcal{D}_{u,v}} |S(D)|$ is maximized. The number of pairs $u \leq t$ and $v > t$ is bounded by $\alpha^2 n^{2/11}$, and every D such that $S(D) \subset S^{\pi}$ belongs to some $\mathcal{D}_{u,v}$. Thus by applying Equation (23), we get

$$\frac{|S^{\pi}|}{|S|} \leq \frac{\alpha^2 n^{2/11} \cdot \sum_{D \in \mathcal{D}_{u_0,v_0}} |S(D)|}{\sum_{D_g \in \tilde{\mathcal{D}}_{u_0,v_0}} |S(D_g)|} \leq \alpha^2 n^{2/11} \cdot \exp(19) \cdot n^{-4/11} \leq \frac{1}{\alpha} \cdot n^{-2/11}$$

where the last inequality is by definition of $\alpha = e^{-7}$. We have completed proving Claim 17.2 (and hence Claim 17.1 and Lemma 17). ■

4.4 Wrapping Up the Proof of Theorem 2.

Recall that the *statistical difference* between two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ over a finite domain U is defined as the maximum over all subsets $U' \subseteq U$, of the difference between the probability weight of U' according to $\mathcal{D}_1$ and the probability weight of U' according to $\mathcal{D}_2$. As an immediate corollary of Lemma 17 we thus get:

Corollary 18 *For any algorithm $\mathcal{A}$ that asks at most $\alpha n^{1/11}/\log n$ queries for $\alpha \leq e^{-7}$, consider the distributions on query-answer sequences when it interacts with $\mathcal{P}_{\mathrm{POS}}$ and $\mathcal{P}_{\mathrm{NEG}}$ respectively. Then the statistical difference between the two distributions is at most $1/4$.*

Assume contrary to Theorem 2 that there exists a testing algorithm $\mathcal{A}$ that asks less than $\alpha n^{1/11}/\log n$ queries and accepts with probability at least $2/3$ every string in PAR_2, and rejects with probability at least $2/3$ every string that is 2^{-6}-far from PAR_2.

Let $\mathcal{D}^{\mathcal{A}}_{\mathrm{POS}}$ and $\mathcal{D}^{\mathcal{A}}_{\mathrm{NEG}}$ denote the distributions on query-answer sequences when algorithm $\mathcal{A}$ interacts with $\mathcal{P}_{\mathrm{POS}}$ and $\mathcal{P}_{\mathrm{NEG}}$ respectively. By Claim 16, the distribution $\mathcal{D}^{\mathcal{A}}_{\mathrm{POS}}$ is equivalent to the distribution on query-answer sequences resulting from the execution of $\mathcal{A}$ on a string generated according to POS_n (where every such string belongs to PAR_2). By our assumption on $\mathcal{A}$, we thus have

$$\Pr\left[\mathcal{A}(\mathcal{D}^{\mathcal{A}}_{\mathrm{POS}}) = \text{ accept }\right] \geq 2/3. \tag{28}$$

Since an analogous statement holds for $\mathcal{D}^{\mathcal{A}}_{\mathrm{NEG}}$, then by applying Lemma 14 we obtain

$$\Pr\left[\mathcal{A}(\mathcal{D}^{\mathcal{A}}_{\mathrm{NEG}}) = \text{ accept }\right] < 1 \cdot 1/3 + \exp(-\Omega(n)) \cdot 1. \tag{29}$$

But by Corollary 18, if $\mathcal{A}$ asks $q < \alpha n^{1/11}/\log n$ queries, then the statistical differences between the two distributions is at most 1/4. This implies that

$$|\Pr[\mathcal{A}(\mathcal{D}^{\mathcal{A}}_{\text{POS}}) = \text{ accept }] - \Pr[\mathcal{A}(\mathcal{D}^{\mathcal{A}}_{\text{NEG}}) = \text{ accept }] \leq 1/4$$

But this stands in contradiction to Equations (28) and (29).

4.5 Adapting the Lower-Bound Argument to D_2

Given the distributions $\text{POS}_{n/2}$ and $\text{NEG}_{n/2}$, we define distributions POS'_n and NEG'_n over strings in Σ_2, where now there are two types of parentheses and no additional symbols. For every string s of length $n/2$ generated by $\text{POS}_{n/2}$ (similarly, $\text{NEG}_{n/2}$), consider the string s' where each 'a' in s is replaced by a matching opening and closing parenthesis in s', and each parenthesis in s is replaced by two parentheses of the same type in s'. The resulting string s' is generated by POS'_n (respectively, NEG'_n) with the same probability that s is generated by $\text{POS}_{n/2}$ (respectively, $\text{NEG}_{n/2}$). Then it is not hard to verify that using these two distributions we can obtain the following theorem.

Theorem 3 *Any algorithm for testing D_2 with distance parameter $\epsilon \leq 2^{-6}$ and success probability of at least $2/3$, requires $\Omega(n^{1/11}/\log n)$ queries.*

5 Testing $uu^r vv^r$ in $\tilde{O}(\sqrt{n}/\epsilon)$ time

Let $L_{\text{REV}} = \{uu^r vv^r : u, v \in \Sigma^*\}$, where Σ is any fixed alphabet and u^r denotes the string u in reverse order. In this section, we show that the following algorithm tests whether $w = w_0 \cdots w_{n-1} \in \Sigma^n$ belongs to L_{REV} or is ϵ-far from any word in the language. The query complexity and running time of the algorithm are $\tilde{O}(\sqrt{n}/\epsilon)$. Recall that Alon et. al. [AKNS00] have shown a lower bound of $\Omega(\sqrt{n})$ for a constant ϵ, on the query complexity of testing algorithms for this class.

Algorithm 2 Test for L_{REV}

1. *Let $I = \{0, \ldots, \sqrt{n} - 1\}$ and $J = \{0, \sqrt{n}, 2\sqrt{n}, \ldots, n - \sqrt{n}\}$.*
2. *Pick $m = c_1 \frac{1}{\epsilon} \log n$ indices $p_1, \ldots, p_m$ independently and uniformly from $\{0, \ldots, n-1\}$.*
3. *For each index $i \in I$, let the* backward pattern *of i be the vector $x = w_{(i-p_1) \bmod n}, \ldots, w_{(i-p_m) \bmod n}$. For each index $j \in J$, let the* forward pattern *of j be the vector $y = w_{(j+p_1) \bmod n}, \ldots, w_{(j+p_m) \bmod n}$.*
4. *Output* accept *if there exists a pair $i \in I$ and $j \in J$ (where not both are 0) such that the backward pattern of i and the forward pattern of j are the same. Otherwise output* reject.

In order to implement the last step we simply construct a *trie* that contains both the backward patterns of the indices $i \in I$ and the forward patterns of the indices $j \in J$. That is, we construct a tree whose edges are labeled by alphabet symbols in Σ. Each leaf of the tree is associated with two subsets: the subset of indices in I whose backward pattern corresponds to the path from the root of the tree to the leaf, and the subset of indices in J whose forward pattern corresponds to this path. If for some leaf both subsets are non-empty, then the algorithm accepts. Hence the above algorithm runs in time $(|I| + |J|) \cdot m = O((\sqrt{n} \log n)/\epsilon)$.

Theorem 4 *The above algorithm is a property tester for L_{REV}. Furthermore, the algorithm has a one-sided error.*

Proof: We first show that if $w \in L_{\mathrm{REV}}$ then the test always accepts. Let $w = uu^r vv^r$. We say that $i, j \in [n]$ are *paired with respect to* w if $i+j = (2|u|-1) \bmod n$. In other words, i and j are either in symmetric positions with respect to uu^r, or with respect to vv^r. By definition, if i and j are paired with respect to w, then $w_i = w_j$. Furthermore, for every offset p, $(w_i - p) \bmod n = (w_j + p) \bmod n$ (and vice versa). In particular, for any selection of $p_1, \dots, p_m$, the forward pattern of j and the backward pattern of i are identical. But by our selection of I and J, there must exist $i \in I$ and $j \in J$ that are paired with respect to w. To see why this is true, observe that $(2|u|-1) \bmod n$, which ranges between 1 and $n-1$, can be written as $(a_1 \cdot \sqrt{n} + a_0) \bmod n$, for some $0 \le a_1, a_0 \le \sqrt{n} - 1$. Hence, $a_0 \in I$ and $a_1 \cdot \sqrt{n} \in J$, and the test necessarily accepts w.

Next we show that if w is ϵ-far from L_{REV}, then the test rejects it with probability of at least 2/3. We say that $i, j \in \{0, \dots, n-1\}$ are a *compatible pair with respect to* w if $(j - i) \bmod n$ is odd, and if $w_{(i-\ell) \bmod n} = w_{(j+\ell) \bmod n}$ for at least a $1 - \epsilon$ fraction of the indices $\ell \in [n]$. We claim that if there exists a compatible pair i, j with respect to w, then w is ϵ-close to L_{REV}. To see this, assume that $i < j$, and let $u = w_0, \dots, w_{\lfloor \frac{j+i}{2} \rfloor - 1}$ and $v = w_{j+i+1}, \dots, w_{\lfloor \frac{n+j+i-1}{2} \rfloor}$. It is not hard to verify that w is ϵ-close to $uu^r vv^r$.

Thus, if w is ϵ-far from L_{REV}, then there is no compatible pair with respect to w. It follows that for every fixed pair $i \in I$ and $j \in J$ (that are necessarily not compatible), the probability that the backward pattern of i is identical to the forward pattern of j is at most $(1-\epsilon)^{(c_1 \log n)/\epsilon} < n^{-c_1}$. Applying the union bound, and using the fact that the total number of pairs considered by the algorithm is n, if $c_1 > 2$ then the probability that the test accepts w is smaller than 1/3, as required. ■

References

[AKNS00] N. Alon, M. Krivelevich, I. Newman, and M Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, pages 1842–1862, 2000.

[BR02] M. Bender and D. Ron. Testing properties of directed graphs: Acyclicity and connectivity. *Random Structures and Algorithms*, pages 184–205, 2002.

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

[GR02] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002.

[Har78] M. Harrison. *Introduction to formal language theory.* Addison-Wesley, 1978.

[KMP77] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[Koz97] D. Kozen. *Automata and Computability.* Springer Verlag, 1997.

[New00] I. Newman. Testing of functions that have small width branching programs. In *Proceedings of the Forty-First Annual Symposium on Foundations of Computer Science*, pages 251–258, 2000.

[PR02] M. Parnas and D. Ron. Testing the diameter of graphs. *Random Structures and Algorithms*, 20(2):165–183, 2002.

[Ron01] D. Ron. Property testing. In *Handbook of Randomized Computing, Volume II*, pages 597–649, 2001.

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

[Sch63] N. Chomsky M. P. Schotzenberger. The algebraic theory of context-free languages. In *Computer Programming and Formal Languages, P. Braffort and D. Hirschberg, Eds, North Holland*, pages 118–161, 1963.

[Yao77] A.C. Yao. Probabilistic computation, towards a unified measure of complexity. In *Proceedings of the Eighteenth Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.