Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 156 (2019)

Almost Optimal Testers for Concise Representations

Nader H. Bshouty
Dept. of Computer Science
Technion, Haifa, 32000

June 21, 2023

Abstract

We give improved and almost optimal testers for several classes of Boolean functions on n inputs that have concise representation in the uniform and distribution-free model. Classes, such as k-Junta, k-Linear Function, s-Term DNF, s-Term Monotone DNF, r-DNF, Decision List, r-Decision List, size-s Decision Tree, size-s Boolean Formula, size-s Branching Program, s-Sparse Polynomial over the binary field and functions with Fourier Degree at most d.

The approach is new and combines ideas from Diakonikolas et al. [30], Bshouty [15], Goldreich et al. [39], and learning theory. The method can be extended to several other classes of functions over any domain that can be approximated by functions that have a small number of relevant variables.

ISSN 1433-8092

Contents

1 Inroduction

Property testing of Boolean function was first considered in the seminal works of Blum, Luby and Rubinfeld [13] and Rubinfeld and Sudan [55] and has recently become a very active research area. See for example, [2, 4, 5, 6, 9, 10, 15, 19, 20, 21, 22, 23, 25, 26, 30, 33, 38, 40, 43, 44, 48, 47, 51, 56] and other works referenced in the surveys [36, 53, 54].

A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is said to be k-junta if it depends on at most k coordinates. The class k-Junta is the class of all k-juntas. The class k-Junta has been of particular interest to the computational learning theory community [11, 12, 17, 29, 41, 45, 49]. A problem closely related to learning k-Junta is the problem of learning and testing subclasses C of k-Junta and classes C of Boolean functions that can be approximated by k-juntas [10, 12, 31, 21, 30, 39, 40, 51]: Given black-box query access to a Boolean function f. In learning, for $f \in C$, we need to learn, with high probability, a hypothesis h that is ϵ-close to f. In testing, for any Boolean function f, we need to distinguish, with high probability, the case that f is in C versus the case that f is ϵ-far from every function in C.

In the *uniform-distribution property testing* (and learning) the distance between Boolean functions is measured with respect to the uniform distribution. In the *distribution-free property testing*, [39], (and learning [57]) the distance between Boolean functions is measured with respect to an arbitrary and unknown distribution $\mathcal{D}$ over $\{0,1\}^n$. In the distribution-free model, the testing (and learning) algorithm is allowed (in addition to making black-box queries) to draw random $x \in \{0,1\}^n$ according to the distribution $\mathcal{D}$. This model is studied in [15, 27, 32, 34, 42, 46].

1.1 Results

We give improved and almost optimal testers for several classes of Boolean functions on n inputs that have concise representation in the uniform and distribution-free models. The classes studied here are k-Junta, k-Linear Functions, k-Term, s-Term DNF, s-Term Monotone DNF, s-Term Monotone r-DNF, r-DNF, Decision List, Length-k Decision List, r-Decision List, size-s Decision Tree, size-s Branching Programs, size-s Boolean Formula, size-s-Boolean Circuit, s-Sparse Polynomials over the binary field, s-Sparse Polynomials of Degree d and functions with Fourier Degree at most d.

In Table 1, we list all the previous results and our results in this paper. In the table, $\tilde{O}(T)$ stands for $O(T \cdot poly(\log T))$, U and D stand for uniform and distribution-free model, and Exp and Poly stand for exponential and polynomial time.

It follows from the lower bounds of Saglam, [56], that our query complexity is almost optimal (with log-factor) for the classes k-Junta, k-Linear, k-Term, s-Term DNF, s-Term Monotone DNF, r-DNF (r constant), Decision List, r-Decision List (r constant), size-s Decision Tree, size-s Branching Programs and size-s Boolean Formula. For more details on the previous results and the results in this paper see Table 1 and Sections 5, 7 and 8.

1.2 Notations

In this subsection, we give some notations that we use throughout the paper.

Denote $[n] = \{1, 2, \ldots, n\}$. For $S \subseteq [n]$ and $x = (x_1, \ldots, x_n)$ we denote $x(S) = \{x_i | i \in S\}$. For $X \subset [n]$ we denote by $\{0,1\}^X$ the set of all binary strings of length $|X|$ with coordinates indexed by $i \in X$. For $x \in \{0,1\}^n$ and $X \subseteq [n]$ we write $x_X \in \{0,1\}^X$ to denote the projection of x over coordinates in X. We denote by 1_X and 0_X the all-one and all-zero strings in $\{0,1\}^X$,

Class of Functions	**Model**	**#Queries**	**Time**	**Reference**
s-Term Monotone DNF s-Term Unate DNF	U	$\tilde{O}(s^2/\epsilon)$	Poly.	[51]
	U	$\tilde{O}(s/\epsilon^2)$	Exp.	[21]
	U	$\tilde{O}(s/\epsilon)$	Poly.	This Paper
s-Term Monotone r-DNF s-Term Unate r-DNF	U	$\tilde{O}(s/\epsilon^2)$	Exp.	[21]
	U	$\tilde{O}(s/\epsilon)$	Poly.	This Paper
	D	$\tilde{O}(s^2r/\epsilon)$	Poly.	This Paper
s-Term DNF	U	$\tilde{O}(s^2/\epsilon)$	Exp.	[30]
	U	$\tilde{O}(s/\epsilon^2)$	Exp.	[21]
	U	$\tilde{O}(s/\epsilon)$	Exp.	This Paper
r-DNF (Constant r)	U	$\tilde{O}(1/\epsilon)$	Poly.	This Paper
Decision List	U	$\tilde{O}(1/\epsilon^2)$	Poly.	[30]
	U	$\tilde{O}(1/\epsilon)$	Poly.	This Paper
Length-k Decision List	D	$\tilde{O}(k^2/\epsilon)$	Poly.	This Paper
r-DL (Constant r)	U	$\tilde{O}(1/\epsilon)$	Poly.	This Paper
k-Linear	U	$\tilde{O}(k/\epsilon)$	Poly.	[8, 13]
	D	$\tilde{O}(k/\epsilon)$	Poly.	This Paper
k-Term	U	$O(1/\epsilon)$	Poly.	[51]
	U	$\tilde{O}(1/\epsilon)$	Poly.	This Paper
	D	$\tilde{O}(k/\epsilon)$	Poly.	This Paper
size-s Decision Trees and size-s Branching Programs	U	$\tilde{O}(s/\epsilon^2)$	Exp.	[21]
	U	$\tilde{O}(s/\epsilon)$	Exp.	This Paper
	D	$\tilde{O}(s^2/\epsilon)$	Exp.	This Paper
size-s Boolean Formulas	U	$\tilde{O}(s/\epsilon^2)$	Exp.	[21]
	U	$\tilde{O}(s/\epsilon)$	Exp.	This Paper
size-s Boolean Circuit	U	$\tilde{O}(s^2/\epsilon^2)$	Exp.	[21]
	U	$\tilde{O}(s^2/\epsilon)$	Exp.	This Paper
Functions with Fourier Degree $\le d$	U	$\tilde{O}(2^{2d}/\epsilon^2)$	Exp.	[21]
	D	$\tilde{O}(2^d/\epsilon + 2^{2d})$	Poly.	This Paper
s-Sparse Polynomial over F_2 of Degree d	U	$poly(s/\epsilon) + \tilde{O}(2^{2d})$	Poly.	[1, 31]
	U	$\tilde{O}(s^2/\epsilon + 2^{2d})$	Poly.	This Paper+[1]
	U	$\tilde{O}(s/\epsilon + s2^d)$	Poly.	This Paper
	D	$\tilde{O}(s^2/\epsilon + s2^d)$	Poly.	This Paper
s-Sparse Polynomial over F_2	U	$\tilde{O}(s/\epsilon^2)$	Exp.	[21]
	U	$Poly(s/\epsilon)$	Poly.	[31]
	U	$\tilde{O}(s^2/\epsilon)$	Poly.	This Paper

Figure 1: A table of the results. In the table, $\tilde{O}(T)$ stands for $O(T \cdot poly(\log T))$, U and D stand for uniform and distribution-free model, and Exp and Poly stand for exponential and polynomial time.

respectively. When we write $x_I = 0$ we mean $x_I = 0_I$. For $X_1, X_2 \subseteq [n]$ where $X_1 \cap X_2 = \emptyset$ and

$x \in \{0,1\}^{X_1}, y \in \{0,1\}^{X_2}$ we write $x \circ y$ to denote their concatenation, i.e., the string in $\{0,1\}^{X_1 \cup X_2}$ that agrees with x over coordinates in X_1 and agrees with y over coordinates in X_2. Notice that $x \circ y = y \circ x$. When we write $u = \circ_{w \in W} w$ we mean that u is the concatenation of all the strings in W. For $X \subseteq [n]$ we denote $\overline{X} = [n] \backslash X = \{x \in [n] | x \notin X\}$. We say that two strings x and y are *equal on I* if $x_I = y_I$.

Given $f, g : \{0,1\}^n \to \{0,1\}$ and a probability distribution $\mathcal{D}$ over $\{0,1\}^n$, we say that f is *ϵ-close to g with respect to $\mathcal{D}$* if $\mathbf{Pr}_{x \in \mathcal{D}}[f(x) \neq g(x)] \leq \epsilon$, where $x \in \mathcal{D}$ means x is chosen from $\{0,1\}^n$ according to the distribution $\mathcal{D}$. We say that f is *ϵ-far from g with respect to $\mathcal{D}$* if $\mathbf{Pr}_{x \in \mathcal{D}}[f(x) \neq g(x)] \geq \epsilon$. For a class of Boolean functions C, we say that f is *ϵ-far from every function in C with respect to $\mathcal{D}$* if for every $g \in C$, f is ϵ-far from g with respect to $\mathcal{D}$. We will use U to denote the uniform distribution over $\{0,1\}^n$ or over $\{0,1\}^X$ when X in clear from the context.

For a Boolean function f and $X \subset [n]$, we say that X is a *relevant set* of f if there are $a, b \in \{0,1\}^n$ such that $f(a) \neq f(b_X \circ a_{\overline{X}})$. We call the pair (a, b) (or just a when $b = 0$) a *witness* of f for the relevant set X. When $X = \{i\}$ then we say that x_i is a *relevant variable* of f and a is *a witness* of f for x_i. Obviously, if X is relevant set of f then $x(X)$ contains at least one relevant variable of f.

We say that the Boolean function $f : \{0,1\}^n \to \{0,1\}$ is a literal if $f \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ where $\overline{x}$ is the negation of x.

Let C be a class of Boolean functions $f : \{0,1\}^n \to \{0,1\}$. We say that C is *closed under variable projection* if for every *projection* $\pi : [n] \to [n]$ and every $f \in C$, we have $f(x(\pi)) \in C$ where $x(\pi) := (x_{\pi(1)}, \cdots, x_{\pi(n)})$. We say that C is *closed under zero projection* (resp. *closed under one projection*) if for every $f \in C$ and every $i \in [n]$, $f(0_{\{i\}} \circ x_{\overline{\{i\}}})$ (resp. $f(1_{\{i\}} \circ x_{\overline{\{i\}}}) \in C$). We say it is closed under zero-one projection if is closed under zero and one projection.

Throughout the paper, we assume that the class C is closed under variable and zero projection. After section 3, we assume that it is also closed under one projection.

1.3 The Model

In this subsection, we define the testing and learning models.

In the testing model, we consider the problem of testing a class of Boolean function C in the uniform and distribution-free testing models. In the distribution-free testing model (resp. uniform model), the algorithm has access to a Boolean function f via a black-box that returns $f(x)$ when a string x is queried. We call this query *membership query* (MQ$_f$ or just MQ). The algorithm also has access to unknown distribution $\mathcal{D}$ (resp. uniform distribution) via an oracle that returns $x \in \{0,1\}^n$ chosen randomly according to the distribution $\mathcal{D}$ (resp. according to the uniform distribution). We call this query *example query* (ExQ$_{\mathcal{D}}$ (resp. ExQ)).

A *distribution-free testing algorithm*, [39], (resp. *testing algorithm*) $\mathcal{A}$ for C is an algorithm that, given as input a distance parameter ϵ and the above two oracles to a Boolean function f,

1. if $f \in C$ then $\mathcal{A}$ outputs "accept" with probability at least $2/3$.

2. if f is ϵ-far from every $g \in C$ with respect to the distribution $\mathcal{D}$ (resp. uniform distribution) then $\mathcal{A}$ outputs "reject" with probability at least $2/3$.

We will also call $\mathcal{A}$ *a tester (or ϵ-tester) for the class C* and an algorithm for *ϵ-testing C*.

We say that $\mathcal{A}$ is *one-sided* if it always accepts when $f \in C$; otherwise, it is called *two-sided* algorithm. The *query complexity of $\mathcal{A}$* is the maximum number of queries $\mathcal{A}$ makes on any Boolean function f.

In the learning models, C is a class of representations of Boolean functions rather than a class of Boolean functions. Therefore, we may have two different representations in C that are logically equivalent. In this paper, we assume that this representation is verifiable, that is, given a representation g, one can decide in polynomial time on the length of this representation if $g \in C$.

A *distribution-free proper learning algorithm* (resp. proper learning algorithm under the uniform distribution) $\mathcal{A}$ for C is an algorithm that, given as input an accuracy parameter ϵ, a confidence parameter δ and an access to both MQ_f for the *target function* $f \in C$ and $\mathrm{ExQ}_{\mathcal{D}}$, with unknown $\mathcal{D}$, (resp. ExQ or ExQ_U), with probability at least $1-\delta$, $\mathcal{A}$ returns $h \in C$ that is ϵ-close to f with respect to $\mathcal{D}$ (resp. with respect to the uniform distribution). This model is also called *proper PAC-learning with membership queries* under any distribution (resp. under the uniform distribution) [3, 57]. A *proper exact learning algorithm* [3] for C is an algorithm that given as input a confidence parameter δ and an access to MQ_f for $f \in C$, with probability at least $1-\delta$, returns $h \in C$ that is equivalent to f. The *query complexity of* $\mathcal{A}$ is the maximum number of queries $\mathcal{A}$ makes on any Boolean function $f \in C$.

2 Overview of the Distribution-Free Tester

2.1 Preface

Our approach refers to testing properties that are (symmetric) sub-classes C of k-juntas; that is, $f : \{0,1\}^n \to \{0,1\}$ has the property if there exists a function $f' : \{0,1\}^k \to \{0,1\}$ that belongs to a predetermined class C' of functions (over k-bit strings) such that $f(x) = f'(x_\Gamma)$ for some k-subset Γ. Our new approach builds upon the "testing by implicit sampling" approach of Diakonikolas *et al.* [30], while extending it from the case of uniform distribution to the case of arbitrary unknown distributions (i.e., the distribution-free model).

This allows us to present (almost optimal) *distribution-free* testers for classes of properties that are sub-classes of k-juntas, which correspond to classes of k-bit long Boolean functions.

While we follow Diakonikolas *et al.* [30] in considering learning algorithms for the underlying classes, our approach is also applicable to testing algorithms (see [37, Sec. 6.2]).

Let us again spell out our task. For a class C of n-bit long Boolean functions and a proximity parameter ϵ, given samples from an unknown distribution $\mathcal{D}$ and oracle access to a function $f : \{0,1\}^n \to \{0,1\}$, we wish to distinguish the case that $f \in C$ from the case that f is ϵ-far from C. Recall that C is a (symmetric) class consisting of a symmetric subclass of k-juntas C'; that is, $f \in C$ if and only if there exists a k-subset $\Gamma \subset [n]$ and $f' \in C'$ such that $f(x) = f'(x_\Gamma)$, where $x_{\{i_1,\ldots,i_k\}} = (x_{i_1}, \ldots, x_{i_k})$. Actually, we also assume that C' is closed under zero projection.

2.2 A Bird's Eye View

The basic strategy is to consider a random partition of $[n]$ to $r = O(k^2)$ parts, denoted $(X_1, \ldots, X_r)$, while relying on the fact that, whp, each X_i contains at most one influential variable. Assuming that $f \in C$, first we determine a set I of at most k indices such that $\cup_{i\in[n]\setminus I} X_i$ contains no "significantly influential" variables of f. Suppose that $f' : \{0,1\}^k \to \{0,1\}$, $f' \in C'$, is a function that corresponds to the tested function $f : \{0,1\}^n \to \{0,1\}$, and that $I \subset [n]$ is indeed the collection of all sets that contain influential variables. The crucial ingredient is devising a method that allows to generate samples of the form $(x', f'(x'))$, when given samples of the form $(x, f(x))$ (for $x \in \mathcal{D}$). We stress that we cannot afford to find the influential variables, and so this method works without determining these locations. Using this method, we can test whether f' belongs to the underlying class C'; hence, we test f by implicitly sampling the projection of $\mathcal{D}$ on the (unknown) influential variables.

The method employed by Diakonikolas *et al.* [30] only handles the uniform distribution (i.e., the case that $\mathcal{D}$ is uniform over $\{0,1\}^n$), and so it only yields testers for the standard testing model (rather than for the distribution-free testing model). Furthermore, their method as well as the identification of the set I rely heavily on the notion of influence of sets, where the influence of a set S of locations on the value of a function is defined as $\Pr_{x',x''\in\{0,1\}^n:x'_S=x''_S}[f(x')\neq f(x'')]$. However, this notion refers to the uniform distribution (over $\{0,1\}^n$) and does not seem adequate for the distribution-free context (e.g., for[1] $f(x)=x_1+x_2$ we may get $\Pr_{x',x''\in\mathcal{D}:x'_1=x''_1}[f(x')\neq f(x'')]=0$).

We use a different way of identifying the set I and for generating samples for the underlying function f'. Loosely speaking, we identifies I as the set of indices i for which $f(1_{X_i}\circ 0_{\overline{X_i}})\neq f(0^n)$, where (recall that) $1_S\circ 0_{\overline{S}}$ is a string that is 1 on the locations in S and is 0 on other locations. (*Be warned that this description is an over-simplification*!) This means that for every $i\in I$ and $x\in\{0,1\}^n$, the value of x at the influential variable in the set X_i (a variable whose location is unknown to us!), equals $f(x')+f(0^n)$ where $x'=x_{X_i}\circ 0_{\overline{X_i}}$, i.e., $x'_j=x_j$ if $j\in X_i$ and $x'_j=0$ otherwise.[2] Note that the foregoing holds when $f\in C$; in general, we can test whether $x\mapsto f(x')+f(0^n)$ is close to a dictatorship (under the uniform distribution) and reject otherwise, whereas if the mapping is close to a dictatorship, we can self-correct it.

To sample the distribution $\mathcal{D}_\Gamma$, where Γ is the influential variables in $X_I=\cup_{i\in I}X_i$, we sample $\mathcal{D}$ and determine the value of the influential variable in each set X_i, for $i\in I$. Queries to the function f' are answered by querying f such that the query $y=y_1\cdots y_k$ is mapped to the query $\mathtt{ext}(y)$ such that[3] $\mathtt{ext}(y)_j=y_i$ if j belongs to the i^{th} set in the collection I (and $\mathtt{ext}(y)_j=0$ if $j\in[n]\setminus X_I$). Effectively, we query the function $F:\{0,1\}^n\to\{0,1\}$ defined as $F(x)=f(\mathtt{ext}(x_\Gamma))$, and this makes sense provided that F is close to f (under the distribution $\mathcal{D}$). To test the latter hypothesis condition, we sample $\mathcal{D}$ and for each sample point x we compare $f(x)$ to $F(x)$, where here we again use the ability to determine the value of the influential variable in each set. Specifically, $\mathtt{ext}(x_\Gamma)$ is computed by determining the value of x_Γ (without knowing Γ), and using our knowledge of $(X_i)_{i\in I}$.

We warn that the foregoing description presumes that we have correctly identified the collection I of all sets containing an influential variable. This leaves us with two questions: The first question is, how do we identify the set I. (Note that the influence of a variable may be as low as 2^{-k}, whereas we seek algorithms of poly(k)-complexity.) The solution (to be presented in Section 2.3.1) will be randomized, and will have one-sided error; specifically, we may fail to identify some sets that contain influential variables, but will never include in our collection sets that have no influential variables. Consequently, $f(1_{X_i}\circ 0_{\overline{X_i}})\neq f(0^n)$ may not hold for some $i\in I$, and (over-simplifying again) we shall seek instead some $v^{(i)}\in\{0,1\}^n$ such that $f(v^{(i)})\neq f(w^{(i)})$, where $w^{(i)}=v^{(i)}_{\overline{X_i}}\circ 0_{X_i}$ (i.e., $w^{(i)}_j=v^{(i)}_j$ if $j\in[n]\setminus X_i$ and $w^{(i)}_j=0$ otherwise). Second, as before, for every $i\in I$ and $x\in\{0,1\}^n$, we wish to determine the value in x of the influential variable in the set X_i (a variable

[1] The addition operation in this paper is over the binary field F_2

[2] Indeed, if $\tau(i)\in X_i$ is the index of the (unique) influential variable that resides in the set X_i, then

$$f(x')=x_{\tau(i)}\cdot f(1_{X_i}\circ 0_{\overline{X_i}})+(x_{\tau(i)}+1)\cdot f(0^n)=x_{\tau(i)}+f(0^n)$$

since $f(1_{X_i}\circ 0_{\overline{X_i}})+f(0^n)=1$.

[3] Notice that $\mathtt{ext}(y)=0_{\overline{X_I}}\circ\left(\underset{i\in I}{\circ}(y_i)_{X_i}\right)$ - Here $(y)_X=1_X$ if $y=1$ and 0_X if $y=0$.

whose location is unknown to us!). This is done by observing that if $f \in C$ then this value equals $f(x') + f(v^{(i)}) + 1$ where $x' = x_{X_j} \circ v^{(i)}_{\overline{X_j}}$ (i.e., $x'_j = x_j$ if $j \in X_i$ and $x'_j = v^{(i)}_j$ otherwise).[4] Again, we need to test whether $x \mapsto f(x') + f(v^{(i)}) + 1$ is a dictatorship, and use self-correction.

2.3 The Actual Tester

As warned, the above description is an over-simplification, and the actual way in which the set I is identified and used is more complex.

We fix a random partition of $[n]$ to $r = O(k^2)$ parts, denoted $(X_1, \ldots, X_r)$. If $f \in C$, then, with high probability, each X_i contains at most one influential variable, denoted $\tau(i)$. We assume that this is the case when providing intuition throughout this section.

2.3.1 Stage 1: Finding I and corresponding $v^{(i)}$

Our goal is to find a collection I of at most k sets such that the function h_I is $\epsilon/3$-close to f (w.r.t distribution $\mathcal{D}$), where h_I is defined as $h_I(x) = f(x_{X_I} \circ 0_{\overline{X_I}})$ and $X_I = \cup_{i \in I} X_i$. In addition, for each $i \in I$, we seek a witness $v^{(i)}$ for the fact that f depends on some variable in X_i; that is, $f(v^{(i)}) \neq f(w^{(i)})$ for some $v^{(i)}$ that differ from $w^{(i)}$ only on X_i.

The procedure.

We proceed in iterations, starting with $I = \emptyset$.

1. We sample $\mathcal{D}$ for $O(1/\epsilon)$ times, trying to find $u \in \mathcal{D}$ such that $f(u) \neq h_I(u)$.

   (Note that if $I = \emptyset$, then $h_I(u) = f(0^n)$. In general, we seek u such that $f(u) \neq f(u_{X_I} \circ 0_{\overline{X_I}})$.

   If no such u is found, then we set $h = h_I$ and proceed to Stage 2. In this case, we may assume that h_I is $\epsilon/3$-close to f (w.r.t $\mathcal{D}$).

2. Otherwise (i.e., $f(u) \neq h_I(u)$), we find an $i \in [m] \setminus I$ and $v^{(i)}$ such that $h_I(v^{(i)}) \neq h_{I \cup \{i\}}(v^{(i)})$, which means that X_i contains an influential variable and $v^{(i)}$ is the witness for the sensitivity that we seek. We set $I \leftarrow I \cup \{i\}$ and proceed to the next iteration.

   (We find this i by binary search that seeks i and S such that $h_{I \cup S \cup \{i\}}(u) \neq h_{I \cup S}(u)$, which means that $v^{(i)}$ equals u in locations outside S and is zero on S.)[5]

Once the iterations are suspended (due to not finding u), we reject if $|I| > k$, and continue to the Stage 2 otherwise. Recall that in the latter case $h = h_I$ is $\epsilon/3$-close to f (w.r.t $\mathcal{D}$).

Note that if $f \in C$, then I contains only sets that contain variables of the k-junta, and so we never reject in this stage. In general, if $i \in I$, then $h_{I \setminus \{i\}}(v^{(i)}) \neq h_I(v^{(i)})$, which implies that $f(x') \neq f(x'')$, where x' and x'' differ only on X_i (e.g., $x''_{X_I} = v^{(i)}_{X_I}$ and $x''_j = 0$ if $j \notin X_I$).

[4] Indeed, if $\tau(i) \in X_i$ is the index of the (unique) influential variable that resides in the set X_i, then

$$f(x') = x_{\tau(i)} \cdot f(v^{(i)}) + (x_{\tau(i)} + 1) \cdot f(w^{(i)}) = x_{\tau(i)} + f(v^{(i)}) + 1$$

since $f(v^{(i)}) + f(w^{(i)}) = 1$.

[5] By Step 1, we have $h_{S' \cup I}(u) \neq h_{S'' \cup I}(u)$, for $S' = [n] \setminus I$ and $S'' = \emptyset$, and in each iteration we cut $S' \setminus S''$ by half while maintaining $h_{S' \cup I}(u) \neq h_{S'' \cup I}(u)$.

2.3.2 Stage 2: Extracting the value of an influential variable

Given a collection I as found in Stage 1 (and a sensitivity witness $v^{(i)}$ for each $i \in I$), let $h = h_I$ and recall that h is close to f w.r.t $\mathcal{D}$. For each $i \in I$, given $x \in \{0,1\}^n$, we wish to determine the value of x at the influential variable that resides in X_i.

For each $i \in I$, we define $\nu_i : \{0,1\}^{|X_i|} \to \{0,1\}$ such that $\nu_i(z) = h_I(y)$, where $y_{X_i} = z$ and $y_{\overline{X_i}} = v^{(i)}_{\overline{X_i}}$. Suppose that $f \in C$, and recall that $\tau(i) \in X_i$ denotes the location of the influential variable in X_i. Let $\sigma(i)$ denote the index of $\tau(i)$ in X_i (i.e., the $\sigma(i)^{\text{th}}$ element of X_i is $\tau(i)$). Then, in this case, ν_i is either a dictatorship or an anti-dictatorship. In particular, if ν_i is a dictatorship, then $\nu_i(z) = z_{\sigma(i)}$ (and otherwise $\nu_i(z) = z_{\sigma(i)} + 1$).

For each $i \in I$, we test whether ν_i is a dictatorship or anti-dictatorship, where testing is w.r.t the uniform distribution over $\{0,1\}^{|X_i|}$. Note that we also check whether ν_i is a dictatorship or anti-dictatorship. If the tester (run with proximity parameter 0.1) fails, we reject. Otherwise (i.e., if we did not reject), we can compute ν_i via self-correction on h_I; that is, to compute ν_i at z, we select $u \in \{0,1\}^{|X_i|}$ at random, and return $\nu_i(z+u)+\nu_i(u)$, which (w.h.p.) equals $(z+u)_{\sigma(i)} + u_{\sigma(i)} = z_{\sigma(i)}$.

Hence, we always continue to Stage 3 if $f \in C$, and whenever we continue to Stage 3 we can compute all ν_i (for $i \in I$) via self-correction.

2.3.3 Stage 3: Emulating a tester of C'

Recall that when reaching this stage, we may assume that $h = h_I$ is $\epsilon/3$-close to f (w.r.t $\mathcal{D}$). Also recall that $h_I(x)$ depends only on x_{X_I}, where $X_I = \cup_{i \in I} X_i$, and that by Stage 2 we may assume that $\nu_i(z) = z_{\sigma(i)}$ (for every $i \in I$ and almost all z). In light of the forgoing, we define $F : \{0,1\}^n \to \{0,1\}$ such that $F(x) = h(x')$ where $x'_{X_i} = (x_{\sigma(i)}, \ldots, x_{\sigma(i)})$ (i.e., $x'_j = (x_{X_i})_{\sigma(i)} = x_{\tau(i)}$ if $j \in X_i$)[6] and $x'_j = 0$ otherwise. (Indeed, if $f \in C$, then $F(x) = h(x)$, since $h(y)$ depends only on $(y_{\tau(i)})_{i \in I}$. Using hypothesis that C' (and so C) is closed under zero projection, it follows that $F \in C$.)

We observe that if F is $\epsilon/3$-close (w.r.t $\mathcal{D}$) to both h and C, then f must be ϵ-close to C (since f is $\epsilon/3$-close to h). Hence, we test both these conditions. Specifically, using our ability to sample $\mathcal{D}$, query f, and determine the value of the influential variables in X_I, we proceed as follows:

1. Test whether $F = h$, where testing is w.r.t the distribution $\mathcal{D}$ and proximity parameter $\epsilon/3$.

   This is done by taking $O(1/\epsilon)$ samples of $\mathcal{D}$, and comparing the values of F and h on these sample points. Recall that $h(u) = h_I(u) = f(u_{X_I} \circ 0_{\overline{X_I}})$.

   The value of F on u is determined as follows.

   (a) For every $i \in I$, if ν_i is a dictatorship, then set v_i to equal the self-corrected value of $\nu_i(u_{X_i})$, where ν_i is as defined in Stage 2. Otherwise (i.e., when ν_i is an anti-dictatorship), we set v_i to equal the self-corrected value of $\nu_i(u_{X_i}) + 1$.

   (b) Return the value $h(u')$, where $u'_j = v_i$ if $j \in X_i$ and $u'_j = 0$ otherwise.

   Indeed, $F = h$ always passes this test, whereas F that is $\epsilon/3$-far from h (w.r.t $\mathcal{D}$) is rejected w.h.p.

2. Test whether F is in C, where testing is w.r.t the distribution $\mathcal{D}$ and proximity parameter $\epsilon/3$. This is done by testing whether F' is in C, where $F'(z) = F(x)$ such that $x_j = z_i$ if j is

[6] In general, $\tau(i)$ denotes the location in $[n]$ of the $\sigma(i)^{\text{th}}$ element of X_i.

in the i^{th} set in the collection I, and $x_j = 0$ otherwise. Here we use a distribution-free tester, and analyze it w.r.t the distribution $\mathcal{D}_I$. Toward this end, we need to samples $\mathcal{D}_I$ as well as answer queries to F', where both tasks can be performed as in the prior step.

Recall that if $f \in C$, then $F \in C$, and this test will accept (w.h.p.), whereas if F is $\epsilon/3$-far from C the test will reject (w.h.p.).

We conclude that if we reached Stage 3 and $f \in C$ (resp., f is ϵ-far from C), then we accept (resp., reject) w.h.p.

2.4 Digest: Our approach vs the original one [30]

Our new approach differs from the original approach of Diakonikolas *et al.* [30] in two main aspects:

1. In [30], sets that contain influential variables are identified according to their influence, which is defined with respect to the uniform distribution. This definition seems inadequate when dealing with arbitrary distributions. Instead, we identify such a set by searching for two assignments that differ only on this set and yield different function values. The actual process is iterative and places additional constraints on these assignments (as detailed in Section 2.3.1).

2. In [30], given an assignment to the function, the value of the unique influential variable that resides in a given set S is determined by approximating the influence of two subsets of S (i.e., the subsets of locations assigned the value 0 and 1, respectively). In contrast, we determines this value by defining an auxiliary function, which depends on the unknown influential variable, and evaluating this function (via self-correction w.r.t the uniform distribution; see Section 2.3.2).

2.5 More on Our Techniques

In this section, we give a detailed overview of our techniques.

2.5.1 Testing Subclasses of k-Junta

For testing a subclass C of k-Junta that is closed under variable and zero projections, we use **Tester**C in Figure 8. We first note that **Tester**C rejects if any procedure that it calls rejects.

First, **Tester**C calls the procedure **ApproxTarget**, in Figure 2. **ApproxTarget** partitions the (indices of the) variables $[n]$ into $r = O(k^2)$ disjoint sets $X_1, \ldots, X_r$. Since $C \subseteq k-$Junta it follows that, with high probability (whp), if $f \in C$ then different relevant variables of f fall into different sets. Therefore, if $f \in C$, whp, every X_i contains at most one relevant variable of f. The procedure then binary searches for enough relevant sets $\{X_i\}_{i \in I}$ such that, whp, for $X = \cup_{i \in I} X_i$, $h = f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-close to f with respect to $\mathcal{D}$. If the procedure finds more than k relevant sets of f then there are more than k relevant variables in f and it rejects. If $f \in C$ then the procedure does not reject and, since C is closed under zero projection, $h \in C$. Since, whp, h is $(\epsilon/3)$-close to f with respect to $\mathcal{D}$, it is enough to distinguish whether h is in C or $(2\epsilon/3)$-far from every function in C with respect to $\mathcal{D}$. **ApproxTarget** also finds, for each relevant set X_i, $i \in I$, a witness $v^{(i)} \in \{0,1\}^n$ of h for X_i. That is, for every $i \in I$, $h(v^{(i)}) \neq h(0_{X_i} \circ v^{(i)}_{\overline{X_i}})$. If $f \in C$, then $h \in C$ and, whp, for each $i \in I$, $h(x_{X_i} \circ v^{(i)}_{\overline{X_i}})$ is a literal. **ApproxTarget** makes $\tilde{O}(k/\epsilon)$ queries.

In the second stage, the tester calls the procedure **TestSets**, in Figure 4. **TestSets** verifies, whp, that for every $i \in I$, $h(x_{X_i} \circ v^{(i)}_{\overline{X_i}})$ is (1/30)-close to some literal in $\{x_{\tau(i)}, \overline{x_{\tau(i)}}\}$ for some $\tau(i) \in X_i$, with respect to the uniform distribution. If $f \in C$, then $h \in C$ and, whp, for each $i \in I$, $h(x_{X_i} \circ v^{(i)}_{\overline{X_i}})$ is a literal and therefore **TestSets** does not reject. Notice that if $f \in C$, then, whp, $\Gamma := \{x_{\tau(i)}\}_{i \in I}$ are the relevant variables of h. This test does not give $\tau(i)$ but the fact that $h(x_{X_i} \circ v^{(i)}_{\overline{X_i}})$ is close to $x_{\tau(i)}$ or $\overline{x_{\tau(i)}}$ can be used to find the value of $u_{\tau(i)}$ in every assignment $u \in \{0,1\}^n$ without knowing $\tau(i)$. The latter is done, whp, by the procedure **RelVarValues**. See Figure 5. Both procedures make $\tilde{O}(k)$ queries.

Recall that for $\xi \in \{0,1\}$, ξ_X is the all ξ vector in $\{0,1\}^X$. Then the tester defines the Boolean function $F = h(0_{\overline{X}} \circ \circ_{i \in I}(x_{\tau(i)})_{X_i})$ on the variables $\{x_{\tau(j)}\}_{j \in I}$, that is, the function F is obtained by substituting in h for every $i \in I$ and every $x_j \in x(X_i)$ the variable $x_{\tau(i)}$. Since $C \subseteq k$-Junta and C is closed under variable and zero projections, $\tau(i) \in X_i$ and, whp, $\Gamma = \{x_{\tau(i)}\}_{i \in I}$ are the relevant variables of h we have:

- If the function f is in C then, whp, $F = h \in C$ and F depends on all the variables in $\Gamma = \{x_{\tau(j)}\}_{j \in I}$.

If h is $(2\epsilon/3)$-far from every function in C with respect to $\mathcal{D}$ then either h is $(\epsilon/3)$-far from F with respect to $\mathcal{D}$ or F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$ where $C(\Gamma)$ is the set of all functions in C that depends on all the variables in Γ. Therefore,

- If the function f is ϵ-far from every function in C then, whp, either
  1. h is $(\epsilon/3)$-far from F with respect to $\mathcal{D}$ or
  2. F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$.

Therefore, it remains to do two tests. The first is testing whether $h = F$ given that h is either $(\epsilon/3)$-far from F with respect to $\mathcal{D}$ or $h = F$. The second is testing whether $F \in C$ given that F is either $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$ or $f \in C(\Gamma)$.

The former test, $h = F$, can be done, whp, by choosing $O(1/\epsilon)$ strings $u \in \{0,1\}^n$ according to the distribution $\mathcal{D}$ and testing whether $F(u) = h(u)$. To compute $F(u)$ we need to find $\{u_{\tau(i)}\}_{i \in I}$, which can be done by the procedure **RelVarValues**. Therefore, each query to F requires one call to the procedure **RelVarValues** that uses $\tilde{O}(k)$ queries to f. Thus, the first test can be done using $\tilde{O}(k/\epsilon)$ queries. This is done in the procedure **Close**fF in Figure 6.

Notice that, thus far, all the above procedures run in polynomial time and make $\tilde{O}(k/\epsilon)$ queries.

Testing whether $F \in C$ can be done, whp, by choosing $O((\log |C(\Gamma)|)/\epsilon)$ strings $u \in \{0,1\}^n$ according to the distribution $\mathcal{D}$ and testing whether $F(u) = g(u)$ for every $g \in C(\Gamma)$. Notice here that the time complexity is $poly(|C(\Gamma)|)$ which is polynomial only when $C(\Gamma)$ contains polynomial number of functions.

If the distribution is uniform, we do not need to use **RelVarValues** to find $\{u_{\tau(i)}\}_{i \in I}$ because when the distribution of u is uniform the distribution of $\{u_{\tau(i)}\}_{i \in I}$ is also uniform. Therefore we can just test whether $F(u) = g(u)$ for every $g \in C(\Gamma)$ for uniform $\{u_{\tau(i)}\}_{i \in I}$. Then computing $F(u)$ for random uniform string u can be done in one query to h. Thus, for the uniform distribution, the algorithm makes $\tilde{O}((\log |C(\Gamma)|)/\epsilon)$ queries to f. This is the procedure **Close**FCU in Figure 7.

If the distribution is unknown then each computation of $F(u)$ for a random string u according to the distribution $\mathcal{D}$ requires choosing u according to the distribution $\mathcal{D}$, then extracting $\{u_{\tau(i)}\}_{i \in I}$

from u and then substituting the values $\{u_{\tau(i)}\}_{i\in I}$ in F. This can be done by the procedure **RelVarValues** using $\tilde{O}(k)$ queries to h. Therefore, for unknown distribution the algorithm makes $\tilde{O}((k\log|C(\Gamma)|)/\epsilon)$ queries to f. This is the procedures **Close**FCD in Figure 7.

As we mentioned before the time complexity of **Close**FCU and **Close**FCD is polynomial only if $|C(\Gamma)|$ is polynomial. When $|C(\Gamma)|$ is exponential, we solve the problem via learning theory. We find a proper learning algorithm $\mathcal{A}$ for $C(\Gamma)$. We run $\mathcal{A}$ to learn F. If the algorithm fails, runs more time than it should, asks more queries than it should or outputs a hypothesis $g\not\in C$ then we know that, whp, $F\not\in C(\Gamma)$. Otherwise, it outputs a function $g\in C(\Gamma)$ and then, as above, we test whether $g=F$ given that g is $(\epsilon/3)$-far from F or $g=F$.

Therefore, for the uniform distribution, if the proper learning algorithm for C makes m MQs and q ExQs then the tester makes $m+q+O(1/\epsilon)$ queries. If the distribution is unknown, then the tester makes $m+\tilde{O}(kq+k/\epsilon)$ queries.

2.5.2 Testing Classes that are Close to k-Junta

To understand the intuition behind the second technique, we demonstrate it for testing s-term DNF.

The tester first runs the procedure **Approx**C in Figure 11. This procedure is similar to the procedure **ApproxTarget**. It randomly uniformly partitions the variables to $r=4c^2(c+1)s\log(s/\epsilon)$ disjoint sets $X_1,\ldots,X_r$ and finds relevant sets $\{X_i\}_{i\in I}$. Here c is a large constant. To find a new relevant set, it chooses two random uniform strings $u,v\in\{0,1\}^n$ and verifies if $f(u_X\circ v_{\overline{X}})\neq f(u)$ where X is the union of the relevant sets that it has found thus far. If $f(u_X\circ v_{\overline{X}})\neq f(u)$ then the binary search finds a new relevant set.

In the binary search for a new relevant set, the procedure defines a set X' that is equal to the union of half of the sets in $\{X_i\}_{i\not\in I}$. Then either $f(u_{X\cup X'}\circ v_{\overline{X'}})\neq f(u)$ or $f(u_{X\cup X'}\circ v_{\overline{X'}})\neq f(u_X\circ v_{\overline{X}})$. Then it recursively does the above until it finds a new relevant set X_ℓ.

It is easy to show that if f is s-term DNF then, whp, for all the terms T in f of size at least $c^2\log(s/\epsilon)$, for all the random uniform strings u,v chosen in the algorithm and for all the strings w generated in the binary search, $T(u_X\circ v_{\overline{X}})=T(u)=T(w)=0$. Therefore, when f is s-term DNF, the procedure, whp, runs as if there are no terms of size greater than $c^2\log(s/\epsilon)$ in f. This shows that, whp, each relevant set that the procedure finds contains at least one variable that belongs to a term of size at most $c^2\log(s/\epsilon)$ in f. Therefore, if f is s-term DNF, the procedure, whp, does not generate more than $c^2s\log(s/\epsilon)$ relevant sets. If the procedure finds more than $c^2s\log(s/\epsilon)$ relevant sets then, whp, f is not s-term DNF and therefore it rejects.

Let R be the set of all the variables that belong to the terms in f of size at most $c^2\log(s/\epsilon)$. The procedure returns $h=f(x_X\circ w_{\overline{X}})$ for random uniform w where X is the union of the relevant sets $X=\cup_{i\in I}X_i$ that is found by the procedure. If f is s-term DNF then since $r=4c^2(c+1)s\log(s/\epsilon)$ and the number of relevant sets is at most $c^2s\log(s/\epsilon)$, whp, at least $(1/2)c\log(s/\epsilon)$ variables in each term of f that contains at least $c\log(s/\epsilon)$ variables not in R falls outside X in the partition of $[n]$. Therefore, for random uniform w, whp, terms T in f that contains at least $c\log(s/\epsilon)$ variables not in R satisfies $T(x_X\circ w_{\overline{X}})=0$ and therefore, whp, are vanished in $h=f(x_X\circ w_{\overline{X}})$. Thus, whp, h contains all the terms that contains variables in R and at most $cs\log(s/\epsilon)$ variables not in R. Therefore, whp, h contains at most $c(c+1)s\log(s/\epsilon)$ relevant variables. From this, and using similar arguments as for the procedure **ApproxTarget** in the previous subsection, we prove that, **Approx**C makes at most $\tilde{O}(s/\epsilon)$ queries and

1. If f is s-term DNF then, whp, the procedure outputs X and w such that
   - $h = f(x_X \circ w_{\overline{X}})$ is s-term DNF.
   - The number of relevant variables in $h = f(x_X \circ w_{\overline{X}})$ is at most $O(s\log(s/\epsilon))$.
2. If f is ϵ-far from every s-term DNF then the procedure either rejects or outputs X and w such that, whp, $h = f(x_X \circ w_{\overline{X}})$ is $(3\epsilon/4)$-far from every s-term DNF.

We can now run **Tester**C (with $3\epsilon/4$) on h from the previous subsection for testing C^* where C^* is the set of s-term DNF with $k = O(s\log(s/\epsilon))$ relevant variables. All the procedures makes $\tilde{O}(s/\epsilon)$ queries except **Close**FCU that makes $\tilde{O}(s^2/\epsilon)$ queries. This is because that the size of the class $C^*(\Gamma)$ is $2^{\tilde{O}(s^2)}$ and therefore **Close**FCU makes $\tilde{O}(s^2/\epsilon)$ queries. This gives a tester that makes $\tilde{O}(s^2/\epsilon)$ queries which is not optimal.

Instead, we consider the class C' of s-term DNF with $O(s\log(s/\epsilon))$ variables and terms of size at most $c\log(s/\epsilon)$ and show that, in **Close**FCU, whp, all the terms T of size greater than $c\log(s/\epsilon)$ and all the random strings u chosen in the procedure satisfies $T(u) = 0$ and therefore it runs as if the target function h has only terms of size at most $c\log(s/\epsilon)$. This gives a tester that makes $\tilde{O}(s/\epsilon)$ queries.

As in the previous section, all the procedures run in polynomial time except **Close**FCU. For some classes, we replace **Close**FCU with polynomial time learning algorithms and obtains polynomial time testers.

3 Preparing the Target for Accessing the Relevant Variables

In this Section we give the three procedures **ApproxTarget**, **TestSets** and **RelVarValues**.

3.1 Preliminaries

In this subsection, we give some known results that will be used in the sequel.

The following lemma is straightforward

Lemma 1. *If $\{X_i\}_{i\in[r]}$ is a partition of $[n]$ then for any Boolean function f the number of relevant sets X_i of f is at most the number of relevant variables of f.*

We will use the following folklore result that is formally proved in [46].

Lemma 2. *Let $\{X_i\}_{i\in[r]}$ be a partition of $[n]$. Let f be a Boolean function and $u, w \in \{0,1\}^n$. If $f(u) \neq f(w)$ then a relevant set X_ℓ of f with a string $v \in \{0,1\}^n$ that satisfies $f(v) \neq f(w_{X_\ell} \circ v_{\overline{X_\ell}})$ can be found using $\lceil \log_2 r \rceil$ queries.*

The following is from [8]

Lemma 3. *There exists a one-sided adaptive algorithm,* **UniformJunta**(f, k, ϵ, δ)*, for ϵ-testing k-junta that makes $O(((k/\epsilon) + k\log k)\log(1/\delta))$ queries and rejects f with probability at least $1-\delta$ when it is ϵ-far from every k-junta with respect to the uniform distribution.*

Moreover, it rejects only when it has found $k+1$ pairwise disjoint relevant sets and a witness of f for each one.

3.2 Approximating the Target

In this subsection we give the procedure **ApproxTarget** that returns $(X = \cup_{i\in I} X_i, V = \{v^{(i)}\}_{i\in I}, I)$, $X \subseteq [n]$, $V \subseteq \{0,1\}^n$ and $I \subseteq [r]$ where, whp, each $x(X_i)$, $i \in I$, contains at least one relevant variable of $h := f(x_X \circ 0_{\overline{X}})$ and exactly one if f is k-junta. Each $v^{(i)}$, $i \in I$, is a witness of $f(x_X \circ 0_{\overline{X}})$ for the relevant set X_i. Also, whp, $f(x_X \circ 0_{\overline{X}})$ is (ϵ/c)-close to the target with respect to the distribution $\mathcal{D}$.

```
ApproxTarget($f, \mathcal{D}, \epsilon, c$)
Input: Oracle that accesses a Boolean function $f$ and
          an oracle that draws $x \in \{0,1\}^n$ according to the distribution $\mathcal{D}$.
Output: Either “reject” or $(X, V, I)$

Partition $[n]$ into $r$ sets
1.  Set $r = 2k^2$.
2.  Choose uniformly at random a partition $X_1, X_2, \ldots, X_r$ of $[n]$

Find a close function and relevant sets
3.  Set $X = \emptyset$; $I = \emptyset$; $V = \emptyset$; $t(X) = 0$.
4.  Repeat $M = ck\ln(15k)/\epsilon$ times
5.      Choose $u \in \mathcal{D}$.
6.      $t(X) \leftarrow t(X) + 1$
7.      If $f(u_X \circ 0_{\overline{X}}) \neq f(u)$ then
8.          $W \leftarrow \emptyset$.
9.          Binary Search to find a new relevant set from $(u, u_X \circ 0_{\overline{X}}) \to \ell$;
10.             and a string $w^{(\ell)} \in \{0,1\}^n$ such that $f(w^{(\ell)}) \neq f(w^{(\ell)}_{X_\ell} \circ 0_{X_\ell})$;
11.         $X \leftarrow X \cup X_\ell$; $I \leftarrow I \cup \{\ell\}$.
12.         If $|I| > k$ then Output(“reject”).
13.         $W = W \cup \{w^{(\ell)}\}$.
14.         Choose $w^{(r)} \in W$.
15.         If $f(w^{(r)}_X \circ 0_{\overline{X}}) \neq f(w^{(r)}_{X\backslash X_r} \circ 0_{\overline{X}\cup X_r})$ then
                $W \leftarrow W \backslash \{w^{(r)}\}$; $v^{(r)} \leftarrow w^{(r)}_X \circ 0_{\overline{X}}$; $V \leftarrow V \cup \{v^{(r)}\}$;
                If $W \neq \emptyset$ then Goto 14
16.          Else If $f(w^{(r)}_X \circ 0_{\overline{X}}) \neq f(w^{(r)})$ then $u \leftarrow w^{(r)}$; Goto 9
17.          Else $u \leftarrow w^{(r)}_{X_r} \circ 0_{X_r}$; Goto 9
18.         $t(X) = 0$.
19.     If $t(X) = c\ln(15k)/\epsilon$ then Output$(X, V, I)$.
```

Figure 2: A procedure that finds relevant sets $\{X_i\}_{i\in I}$ of f and a witness $v^{(i)}$ for each relevant set X_i for $h := f(x_X \circ 0_{\overline{X}})$ where $X = \cup_{i\in I} X_i$. Also, whp, h is (ϵ/c)-close to the target.

Consider the procedure **ApproxTarget** in Figure 2. In steps 1-2 the procedure partitions the set $[n]$ into $r = 2k^2$ disjoint sets $X_1, X_2, \ldots, X_r$. In step 3 it defines the variables X, I, V and $t(X)$.

At each iteration of the procedure, I contains the indices of some relevant sets of $f(x_X \circ 0_{\overline{X}})$ where $X = \cup_{i\in I} X_i$, i.e., each X_i, $i \in I$ is relevant set of $f(x_X \circ 0_{\overline{X}})$. The set V contains, for each $i \in I$, a string $v^{(i)} \in \{0,1\}^n$ that satisfies $f(v^{(i)}_X \circ 0_{\overline{X}}) \neq f(v^{(i)}_{X\backslash X_i} \circ 0_{X_i} \circ 0_{\overline{X}})$. That is, a witness of $f(x_X \circ 0_{\overline{X}})$ for the relevant set X_i, $i \in I$.

The procedure in steps 4-19 tests if $f(u_X \circ 0_{\overline{X}}) = f(u)$ for at least $c\ln(15/k)/\epsilon$, independently and at random, chosen u according to the distribution $\mathcal{D}$. The variable $t(X)$ counts the number of such u. If this happens then, whp, $f(x_X \circ 0_{\overline{X}})$ is (ϵ/c)-close to f with respect to $\mathcal{D}$ and the procedure returns (X, V, I). If not then $f(u_X \circ 0_{\overline{X}}) \neq f(u)$ for some u and then a new relevant set is found. If the number of relevant sets is greater than k, it rejects. This is done in steps 8-18.

In steps 9-10, the procedure uses Lemma 2 to (binary) searches for a new relevant set. The search gives an index ℓ of the new relevant set X_ℓ and a witness $w^{(\ell)}$ that satisfies $f(w^{(\ell)}) \neq f(0_{X_\ell} \circ w^{(\ell)}_{\overline{X_\ell}})$. Then ℓ is added to I and X is extended to $X \cup X_\ell$. The binary search gives a witness that X_ℓ is relevant set of f, but not a witness that it is relevant set of $f(x_X \circ 0_{\overline{X}})$. This is why we need steps 14-17. In those steps the procedure adds $w^{(\ell)}$ to W. Then for each $w^{(r)} \in W$ (at the beginning $r = \ell$) it checks if $w^{(r)}$ is a witness of $f(x_X \circ 0_{\overline{X}})$ for X_r. If it is then it adds it to V. If it isn't then we show in the discussion below that a new relevant set can be found. The procedure rejects when it finds more than k relevant sets.

If the procedure does not reject then it outputs (X, V, I) where I contains the indices of some relevant sets of $f(x_X \circ 0_{\overline{X}})$, $X = \cup_{i\in I} X_i$ and the set V contains for each $i \in I$ a string $v^{(i)} \in \{0,1\}^n$ that is a witness of $f(x_X \circ 0_{\overline{X}})$ for X_i, i.e., $f(v^{(i)}_X \circ 0_{\overline{X}}) \neq f(v^{(i)}_{X\backslash X_i} \circ 0_{X_i} \circ 0_{\overline{X}})$. We will also show in Lemma 9 that, whp, $\mathbf{Pr}_{\mathcal{D}}[f(x_X \circ 0_{\overline{X}}) \neq f(x)] \leq \epsilon/c$.

We first prove

Lemma 4. *Consider steps 1-2 in the* **ApproxTarget**. *If f is a k-junta then, with probability at least $2/3$, for each $i \in [r]$, the set $x(X_i) = \{x_j | j \in X_i\}$ contains at most one relevant variable of f.*

Proof. Let x_{i_1} and x_{i_2} be two relevant variables in f. The probability that x_{i_1} and x_{i_2} are in the same set is equal to $1/r$. By the union bound, it follows that the probability that some relevant variables x_{i_1} and x_{i_2}, $i_1 \neq i_2$, in f are in the same set is at most $\binom{k}{2}/r \leq 1/3$. □

	$X^{(j)} = \bigcup_{i=1}^{j} X_{\ell_i}$		$\overline{X^{(j)}}$	
	$X^{(j)}\backslash X_r$	X_r	$\bigcup_{i=j+1}^{q} X_{\ell_i}$	$\overline{\overline{X^{(q)}}}$
$v^{(r)}$	***********	*****	00000000000	000000
$w^{(r)}$	***********	*****	***********	******
$w^{(r)}_{X^{(j)}} \circ 0_{\overline{X^{(j)}}} = v^{(r)}$	***********	*****	00000000000	000000
$w^{(r)}_{\overline{X_r}} \circ 0_{X_r}$	***********	00000	***********	******
$w^{(r)}_{X^{(j)}\backslash X_r} \circ 0_{\overline{X^{(j)} \cup X_r}}$	***********	00000	00000000000	000000

Figure 3: The value of $v^{(r)}$, $w^{(r)}$, $w^{(r)}_{X^{(j)}} \circ 0_{\overline{X^{(j)}}}$, $w^{(r)}_{\overline{X_r}} \circ 0_{X_r}$ and $w^{(r)}_{X^{(j)}\backslash X_r} \circ 0_{\overline{X^{(j)} \cup X_r}}$ where * indicates any value.

Recall that after the binary search in step 9 the procedure has a witness $w^{(\ell)}$ that satisfies $f(w^{(\ell)}) \neq f(w^{(\ell)}_{\overline{X_\ell}} \circ 0_{X_\ell})$ that is not necessarily a witness of $f(x_X \circ 0_{\overline{X}})$ for X_ℓ, i.e., does not necessarily satisfies $f(w^{(\ell)}_X\circ 0_{\overline{X}})\not= f(w^{(\ell)}_{X\backslash X_\ell}\circ 0_{X_\ell}\circ 0_{\overline{X}})$. This is why we first add $w^{(\ell)}$ to W and not to V. We next will show that an element $w^{(r)}$ in W is either a witness of $f(x_X\circ 0_{\overline{X}})$ for X_ℓ, in which case we add it to V and remove it from W, or, this element generates another new relevant set and then another witness of f is added to W.

Suppose the variable ℓ in the procedure takes the values $\ell_1,\ldots,\ell_q$. Then X_ℓ takes the values $X_{\ell_1},\ldots,X_{\ell_q}$ and X takes the values $X^{(i)}$ where $X^{(i)}=X^{(i-1)}\cup X_{\ell_i}$ and $X^{(0)}=\emptyset$. Notice that $X^{(0)}\subset X^{(1)}\subset\cdots\subset X^{(q)}$.

Suppose, at some iteration, the procedure chooses, in step 14, $w^{(r)}\in W$ where $r=\ell_i$. By step 10, $f(w^{(r)})\not=f(w^{(r)}_{\overline{X_r}}\circ 0_{X_r})$. Suppose at this iteration $X=X^{(j)}$. Then $r\le j$, $X_{\ell_1},\ldots,X_{\ell_j}$ are the relevant sets that are discovered so far and $X_{\ell_{j+1}},\ldots,X_{\ell_q}\subseteq\overline{X^{(j)}}$. Since $w^{(r)}\in W$, by step 11, $X_r\subseteq X^{(j)}$. See the table in Figure 3. If in step 15, $f(w^{(r)}_{X^{(j)}}\circ 0_{\overline{X^{(j)}}})\not= f(w^{(r)}_{X^{(j)}\backslash X_r}\circ 0_{\overline{X^{(j)}}\cup X_r})$ then $v^{(r)}=w^{(r)}_{X^{(j)}}\circ 0_{\overline{X^{(j)}}}$ is added to the set V. This is the only step that adds an element to V. Since $v^{(r)}=w^{(r)}_{X^{(j)}}\circ 0_{\overline{X^{(j)}}}$ and $X^{(j)}\subseteq X^{(q)}$ we have $v^{(r)}_{\overline{X^{(q)}}}=0$ and $f(v^{(r)})=f(w^{(r)}_{X^{(j)}}\circ 0_{\overline{X^{(j)}}})\not= f(w^{(r)}_{X^{(j)}\backslash X_r}\circ 0_{\overline{X^{(j)}}\cup X_r})=f(v^{(r)}_{\overline{X_r}}\circ 0_{X_r})$.

Therefore

Lemma 5. *If the procedure outputs $(X^{(q)},V,I)$ then for every $v^{(\ell)}\in V$ we have $v^{(\ell)}_{\overline{X^{(q)}}}=0$ and $f(v^{(\ell)})\not=f(v^{(\ell)}_{\overline{X_\ell}}\circ 0_{X_\ell})$. That is, $v^{(\ell)}\in V$ is a witness of $f(x_{X^{(q)}}\circ 0_{\overline{X^{(q)}}})$ for X_ℓ.*

We now show that if, in step 15, $f(w^{(r)}_X\circ 0_{\overline{X}})=f(w^{(r)}_{X\backslash X_r}\circ 0_{\overline{X}\cup X_r})$ then the procedure finds a new relevant set.

Lemma 6. *Consider step 15 in the procedure in the iteration where $X=X^{(j)}$. If $w^{(r)}$ is not a witness of $f(x_{X^{(j)}}\circ 0_{\overline{X^{(j)}}})$ for X_r, i.e., $f(w^{(r)}_{X^{(j)}}\circ 0_{\overline{X^{(j)}}})=f(w^{(r)}_{X^{(j)}\backslash X_r}\circ 0_{\overline{X^{(j)}}\cup X_r})$, then a new relevant set is found.*

Proof. See the table in Figure 3 throughout the proof. Since by step 10, $f(w^{(r)})\not=f(w^{(r)}_{\overline{X_r}}\circ 0_{X_r})$, then either $f(w^{(r)})\not= f(w^{(r)}_{X^{(j)}}\circ 0_{\overline{X^{(j)}}})$ or $f(w^{(r)}_{X^{(j)}\backslash X_r}\circ 0_{\overline{X^{(j)}}\cup X_r})\not= f(w^{(r)}_{\overline{X_r}}\circ 0_{X_r})$. If $f(w^{(r)})\not= f(w^{(r)}_{X^{(j)}}\circ 0_{\overline{X^{(j)}}})$ then the procedure in step 16 assign $u=w^{(r)}$ and goes to step 9 to find a relevant set in $\overline{X^{(j)}}$. Step 9 finds a new relevant set because $w^{(r)}$ and $w^{(r)}_{X^{(j)}}\circ 0_{\overline{X^{(j)}}}$ are equal on $X^{(j)}$. If $f(w^{(r)}_{X^{(j)}\backslash X_r}\circ 0_{\overline{X^{(j)}}\cup X_r})\not= f(w^{(r)}_{\overline{X_r}}\circ 0_{X_r})$ then the procedure in step 17 assign $u=w^{(r)}_{\overline{X_r}}\circ 0_{X_r}$ and goes to step 9 to find a relevant set in $\overline{X^{(j)}}$. Step 9 finds a new relevant set because $w^{(r)}_{X^{(j)}\backslash X_r}\circ 0_{\overline{X^{(j)}}\cup X_r}$ and $w^{(r)}_{\overline{X_r}}\circ 0_{X_r}$ are equal on $X^{(j)}$. □

Therefore, for every $w^{(r)}\in W$ the procedure either finds $v^{(r)}$ that satisfies the condition in Lemma 5 or finds a new relevant set. If the number of relevant sets is greater than k, then the procedure rejects. This is because each relevant set contains a relevant variable, and the relevant sets are disjoint. So the function, in this case, is not k-junta and therefore not in C. If the number of relevant sets is less than or equal to k, then the algorithm eventually finds, for each $\ell\in I$, a witness $v^{(\ell)}$ of $f(x_X\circ 0_{\overline{X}})$ for X_{i_ℓ}. This implies

Lemma 7. *If* **ApproxTarget** *does not reject then it outputs* $(X = X^{(q)}, V = \{v^{(\ell_1)}, \ldots, v^{(\ell_q)}\}, I = \{\ell_1, \ldots, \ell_q\})$ *that satisfies*

1. $q = |I| \le k$.
2. *For every* $\ell \in I$, $v^{(\ell)}_{\overline{X}} = 0$ *and* $f(v^{(\ell)}) \ne f(0_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$*. That is,* $v^{(\ell)} \in V$ *is a witness of* $f(x_X \circ 0_{\overline{X}})$ *for* X_ℓ *.*
3. *Each* $x(X_\ell)$*,* $\ell \in I$*, contains at least one relevant variable of* $f(x_X \circ 0_{\overline{X}})$*.*

Lemma 8. *If* f *is* k*-junta and each* $x(X_i)$ *contains at most one relevant variable of* f *then*

1. **ApproxTarget** *outputs* (X, V, I)*.*
2. *Each* $x(X_\ell)$*,* $\ell \in I$*, contains exactly one relevant variable in* $f(x_X \circ 0_{\overline{X}})$*.*
3. *For every* $\ell \in I$*,* $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ *is a literal.*

Proof. By *3* in Lemma 7, $x(X_\ell)$, $\ell \in I$, contains exactly one relevant variable. Thus, for every $\ell \in I$, $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is a literal.

Since f contains at most k relevant variables, by Lemma 1, the number of relevant sets $|I|$ is at most k. Therefore, **ApproxTarget** does not halt in step 12. □

The following lemma shows that

Lemma 9. *If* **ApproxTarget** *outputs* (X, V, I) *then* $|I| \le k$ *and with probability at least* $14/15$

$$\mathbf{Pr}_{u \in \mathcal{D}}[f(u_X \circ 0_{\overline{X}}) \ne f(u)] \le \epsilon/c.$$

Proof. If $|I| > k$ then, from step 12, **ApproxTarget** outputs "reject". Therefore, the probability that **ApproxTarget** fails to output (X, V, I) with $\mathbf{Pr}_{u \in \mathcal{D}}[f(u_X \circ 0_{\overline{X}}) \ne f(u)] \le \epsilon/c$ is the probability that for some $X^{(\ell)}$, $\mathbf{Pr}_{u \in \mathcal{D}}[f(x_{X^{(\ell)}} \circ 0_{\overline{X^{(\ell)}}}) \ne f(x)] > \epsilon/c$ and $f(u_{X^{(\ell)}} \circ 0_{\overline{X^{(\ell)}}}) = f(u)$ for $c\ln(15k)/\epsilon$ strings u chosen independently at random according to the distribution $\mathcal{D}$. This probability is at most

$$k\left(1 - \frac{c}{\epsilon}\right)^{c\ln(15k)/\epsilon} \le \frac{1}{15}.$$

□

We now give the query complexity

Lemma 10. *The procedure* **ApproxTarget** *makes* $O((k\log k)/\epsilon)$ *queries.*

Proof. The condition in step 7 requires two queries and is executed at most $M = ck\ln(15k)/\epsilon$ times. This is $2M = O((k\log k)/\epsilon)$ queries. Steps 9-17 are executed at most $k+1$ times. This is because each time it is executed, the value of $|I|$ is increased by one, and when $|I| = k+1$ the procedure rejects. By Lemma 2, to find a new relevant set the procedure makes $O(\log r) = O(\log k)$ queries. This gives another $O(k\log k)$ queries. Therefore, the query complexity is $O((k\log k)/\epsilon)$. □

```
**TestSets**$(X,V,I)$
*Input*: Oracle that accesses a Boolean function $f$ and $(X,V,I)$.
*Output*: Either "reject" or "OK"

1.  For every $\ell \in I$ do
2.      If **UniformJunta**$(f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}}), 1, 1/30, 1/15)$="reject"
3.              then Output("reject")
4.      Choose $b \in U$
5.      If $f(b_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}}) = f(\overline{b_{X_\ell}} \circ v^{(\ell)}_{\overline{X_\ell}})$ then Output("reject")
6.  Return "OK"
```

Figure 4: A procedure that tests if for all $\ell \in I$, $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is (1/30)-close to some literal with respect to the uniform distribution.

3.3 Testing the Relevant Sets

In this subsection we give the procedure **TestSets** that takes as an input $(X, V = \{v^{(\ell_1)}, \ldots, v^{(\ell_q)}\}, I = \{\ell_1, \ldots, \ell_q\})$ and tests if for all $\ell \in I$, $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is (1/30)-close to some literal with respect to the uniform distribution.

We first prove

Lemma 11. *If f is k-junta and each $x(X_i)$ contains at most one relevant variable of f then* **TestSets** *returns "OK".*

Proof. By Lemma 8, for every $\ell \in I$, $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is a literal.

If **TestSets** rejects in step 3 then, by Lemma 3, for some X_ℓ, $\ell \in I$, $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is not 1-Junta (literal or constant function) and therefore $x(X_\ell)$ contains at least two relevant variables. If it rejects in step 5, then $f(b_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}}) = f(\overline{b_{X_\ell}} \circ v^{(\ell)}_{\overline{X_\ell}})$ and then $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is not a literal. In all cases we get a contradiction. □

In the following lemma we show that if **TestSets** returns "OK" then, whp, each $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is close to a literal with respect to the uniform distribution.

Lemma 12. *If for some $\ell \in I$, $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is* (1/30)*-far from every literal with respect to the uniform distribution then, with probability at least* $1 - (1/15)$, **TestSets** *rejects.*

Proof. If $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is (1/30)-far from every literal with respect to the uniform distribution then it is either (case 1) (1/30)-far from every 1-Junta (literal or constant) or (case 2) (1/30)-far from every literal and (1/30)-close to 0-Junta. In case 1, by Lemma 3, with probability at least $1 - (1/15)$, **UniformJunta** $(f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}}), 1, 1/30, 1/15)$ = "reject" and then the procedure rejects. In case 2, if $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is (1/30)-close to some 0-Junta then it is either (1/30)-close to 0 or (1/30)-close to 1. Suppose it is (1/30)-close to 0. Let b be a random uniform string chosen in steps 4. Then $\bar{b}$ is random uniform and for $g(x) = f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ we have

$$\begin{aligned}
\mathbf{Pr}[\text{The procedure does not reject}] &= \mathbf{Pr}\left[g(b) \neq g(\bar{b})\right] \\
&= \mathbf{Pr}[g(b) = 1 \wedge g(\bar{b}) = 0] + \mathbf{Pr}[g(b) = 0 \wedge g(\bar{b}) = 1] \\
&\leq \mathbf{Pr}[g(b) = 1] + \mathbf{Pr}[g(\bar{b}) = 1] \\
&\leq \frac{1}{15}.
\end{aligned}$$

□

Lemma 13. *The procedure* **TestSets** *makes* $O(k)$ *queries.*

Proof. Steps 2 and 5 are executed $|I| \leq k$ times, and by Lemma 3, the total number of queries made is $O(1/(1/30)\log(15))k + 2k = O(k)$. □

3.4 Determining the Values of the Relevant Variables

```
RelVarValues$(w, X, V, I, \delta)$
Input: Oracle that accesses a Boolean function $f$, $(X, V, I)$ and $w \in \{0,1\}^n$.
Output: Either "reject" or for every $\ell \in I$, the value, $z_\ell = w_{\tau(\ell)}$ where $x_{\tau(\ell)}$ is one of the
        relevant variables of $f(x_X \circ 0_{\overline{X}})$ in $x(X_\ell)$

1.  For every $\ell \in I$ do
2.          For $\xi \in \{0,1\}$ set $Y_{\ell,\xi} = \{j \in X_\ell | w_j = \xi\}$.
3.          Set $G_{\ell,0} = G_{\ell,1} = 0$;
4.          Repeat $h = \ln(k/\delta)/\ln(4/3)$ times
5.              Choose $b \in U$;
6.                  If $f(b_{Y_{\ell,0}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}}) \neq f(\overline{b_{Y_{\ell,0}}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}})$ then $G_{\ell,0} \leftarrow G_{\ell,0} + 1$
7.                  If $f(b_{Y_{\ell,1}} \circ b_{Y_{\ell,0}} \circ v^{(\ell)}_{\overline{X_\ell}}) \neq f(\overline{b_{Y_{\ell,1}}} \circ b_{Y_{\ell,0}} \circ v^{(\ell)}_{\overline{X_\ell}})$ then $G_{\ell,1} \leftarrow G_{\ell,1} + 1$
8.          If $(\{G_{\ell,0}, G_{\ell,1}\} \neq \{0, h\})$ then Output("reject")
9.          If $G_{\ell,0} = h$ then $z_\ell \leftarrow 0$ else $z_\ell \leftarrow 1$
10. Output("$\{z_\ell\}_{\ell \in I}$")
```

Figure 5: A procedure that takes as input (X, V, I) and a string $w \in \{0,1\}^n$ and, with probability at least $1 - \delta$, returns the values of $w_{\tau(i)}$, $i \in I$, where $f(x_{X_i} \circ v^{(i)}_{\overline{X_i}})$ is (1/30)-close to one of the literals in $\{x_{\tau(i)}, \overline{x_{\tau(i)}}\}$ with respect to the uniform distribution.

In this subsection we give a procedure **RelVarValue** that for an input $(w \in \{0,1\}^n, X, V, I, \delta)$ where (X, V, I) satisfies all the properties in the previous two subsections, the procedure, with probability at least $1 - \delta$, returns the values of $w_{\tau(i)}$, $i \in I$, where $f(x_{X_i} \circ v^{(i)}_{\overline{X_i}})$ is (1/30)-close to one of the literals in $\{x_{\tau(i)}, \overline{x_{\tau(i)}}\}$ with respect to the uniform distribution. When f is k-junta and each $x(X_i)$ contains at most one relevant variable then $\{x_{\tau(i)}\}_{i\in I}$ is the set of the relevant variables of $f(x_X \circ 0_{\overline{X}})$ and $w_{\tau(i)}$, $i \in I$ are the values of the relevant variables. The procedure is in Figure 5.

We first prove

Lemma 14. *If f is k-Junta and each $x(X_i)$ contains at most one relevant variable of f then* **RelVarValues** *outputs z such that $z_\ell = w_{\tau(\ell)}$ where $f(x_{X_\ell} \circ 0_{\overline{X_\ell}}) \in \{x_{\tau(\ell)}, \overline{x_{\tau(\ell)}}\}$.*

Proof. Since $Y_{\ell,0}, Y_{\ell,1}$ is a partition of X_ℓ, $\ell \in I$ and, by Lemma 8, $x(X_\ell)$ contains exactly one relevant variable $x_{\tau(\ell)}$ of $f(x_X \circ 0_{\overline{X}})$, this variable is either in $x(Y_{\ell,0})$ or in $x(Y_{\ell,1})$ but not in both. Suppose w.l.o.g. it is in $x(Y_{\ell,0})$ and not in $x(Y_{\ell,1})$. Then $w_{\tau(\ell)} = 0$, $f(x_{Y_{\ell,0}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}})$ is a literal and $f(x_{Y_{\ell,1}} \circ b_{Y_{\ell,0}} \circ v^{(\ell)}_{\overline{X_\ell}})$ is a constant function. This implies that for any b, $f(b_{Y_{\ell,0}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}}) \neq f(\overline{b_{Y_{\ell,0}}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}})$ and $f(b_{Y_{\ell,1}} \circ b_{Y_{\ell,0}} \circ v^{(\ell)}_{\overline{X_\ell}}) = f(\overline{b_{Y_{\ell,1}}} \circ b_{Y_{\ell,0}} \circ v^{(\ell)}_{\overline{X_\ell}})$. Therefore, by steps 6-7 in the procedure, $G_{\ell,0} = h$ and $G_{\ell,1} = 0$ and the procedure does not output reject in step 8. Thus, by step 9, $z_\ell = w_{\tau(\ell)}$. □

We now prove

Lemma 15. *If for every $\ell \in I$ the function $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is $(1/30)$-close to a literal in $\{x_{\tau(\ell)}, \bar{x}_{\tau(\ell)}\}$ with respect to the uniform distribution, where $\tau(\ell) \in X_\ell$, and in* **RelVarValues***, for every $\ell \in I$, $\{G_{\ell,0}, G_{\ell,1}\} = \{0, h\}$ then, with probability at least $1-\delta$, we have: For every $\ell \in I$, $z_\ell = w_{\tau(\ell)}$.*

Proof. Fix some ℓ. Suppose $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is $(1/30)$-close to $x_{\tau(\ell)}$ with respect to the uniform distribution. The case when it is $(1/30)$-close to $\overline{x_{\tau(\ell)}}$ is similar. Since $X_\ell = Y_{\ell,0} \cup Y_{\ell,1}$ and $Y_{\ell,0} \cap Y_{\ell,1} = \emptyset$ we have that $\tau(\ell) \in Y_{\ell,0}$ or $\tau(\ell) \in Y_{\ell,1}$, but not both. Suppose $\tau(\ell) \in Y_{\ell,0}$. The case where $\tau(\ell) \in Y_{\ell,1}$ is similar. Define the random variable $Z(x_{X_\ell}) = 1$ if $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}}) \neq x_{\tau(\ell)}$ and $Z(x_{X_\ell}) = 0$ otherwise. Then

$$\mathbf{E}_{x_{X_\ell} \in U}[Z(x_{X_\ell})] \le \frac{1}{30}.$$

Therefore

$$\mathbf{E}_{x_{Y_{\ell,1}} \in U} \mathbf{E}_{x_{Y_{\ell,0}} \in U}[Z(x_{Y_{\ell,0}} \circ x_{Y_{\ell,1}})] \le \frac{1}{30}$$

and by Markov's bound

$$\mathbf{Pr}_{x_{Y_{\ell,1}} \in U}\left[\mathbf{E}_{x_{Y_{\ell,0}} \in U}[Z(x_{Y_{\ell,0}} \circ x_{Y_{\ell,1}})] \ge \frac{2}{15}\right] \le \frac{1}{4}.$$

That is, for a random uniform string $b \in \{0,1\}^n$, with probability at least $3/4$, $f(x_{Y_{\ell,0}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}})$ is $(2/15)$-close to $x_{\tau(\ell)}$ with respect to the uniform distribution. Now, given that $f(x_{Y_{\ell,0}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}})$ is $(2/15)$-close to $x_{\tau(\ell)}$ with respect to the uniform distribution the probability that $G_{\ell,0} = 0$ is the probability that $f(b_{Y_{\ell,0}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}}) = f(\overline{b_{Y_{\ell,0}}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}})$ for h random uniform strings $b \in \{0,1\}^n$. Let $b^{(1)}, \ldots, b^{(h)}$ be h random uniform strings in $\{0,1\}^n$, $V(b)$ be the event $f(b_{Y_{\ell,0}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}}) = f(\overline{b_{Y_{\ell,0}}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}})$ and A the event that $f(x_{Y_{\ell,0}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}})$ is $(2/15)$-close to $x_{\tau(\ell)}$ with respect to the uniform distribution. Let $g(x_{Y_{\ell,0}})=f(x_{Y_{\ell,0}}\circ b_{Y_{\ell,1}}\circ v^{(\ell)}_{\overline{X_\ell}})$. Then

$$\begin{array}{rcl}\mathbf{Pr}[V(b)|A] &=& \mathbf{Pr}[g(b_{Y_{\ell,0}})=g(\overline{b_{Y_{\ell,0}}})|A]\\ &=& \mathbf{Pr}[(g(b_{Y_{\ell,0}})=b_{\tau(\ell)}\wedge g(\overline{b_{Y_{\ell,0}}})=b_{\tau(\ell)})\vee(g(b_{Y_{\ell,0}})=\overline{b_{\tau(\ell)}}\wedge g(\overline{b_{Y_{\ell,0}}})=\overline{b_{\tau(\ell)}})|A]\\ &\le& \mathbf{Pr}[g(\overline{b_{Y_{\ell,0}}})\neq\overline{b_{\tau(\ell)}}\vee g(b_{Y_{\ell,0}})\neq b_{\tau(\ell)})|A]\\ &\le& \mathbf{Pr}[g(\overline{b_{Y_{\ell,0}}})\neq\overline{b_{\tau(\ell)}}|A]+\mathbf{Pr}[g(b_{Y_{\ell,0}})\neq b_{\tau(\ell)})|A]\le\dfrac{4}{15}.\end{array}$$

Since $\tau(\ell)\in Y_{\ell,0}$, we have $w_{\tau(\ell)}=0$. Therefore, by step 9 and since $\tau(\ell)\in X_\ell$,

$$\begin{array}{rcl}\mathbf{Pr}[z_\ell\neq w_{\tau(\ell)}] &=& \mathbf{Pr}[z_\ell=1]\\ &=& \mathbf{Pr}[G_{\ell,0}=0\wedge G_{\ell,1}=h]\\ &\le& \mathbf{Pr}[G_{\ell,0}=0]=\mathbf{Pr}[(\forall j\in[h])V(b^{(j)})]\\ &=& (\mathbf{Pr}[V(b)])^h\le\left(\mathbf{Pr}[V(b)|A]+\mathbf{Pr}[\overline{A}]\right)^h\le(4/15+1/4)^h\le(3/4)^h\end{array}$$

Therefore, the probability that $z_\ell\neq w_{\tau(\ell)}$ for some $\ell\in I$ is at most $k(3/4)^h\le\delta$. □

The following is obvious

Lemma 16. *The procedure* **RelVarValues** *makes $O(k\log(k/\delta))$ queries.*

4 Testing Subclasses of k-Junta

In this section, we give testers for subclasses of k-Junta that are closed under variable and zero projections.

Our tester will start by running the two procedures **ApproxTarget** and **TestSets** and therefore, by Lemmas 4, 8 and 11, if $f\in C$ (and therefore is k-junta) then, with probability at least 2/3, both procedures do not reject and item *1* in the following Assumption happens. By Lemmas 7, 9, and 12, if f is ϵ-far from every function in C and both procedures do not reject then, with probability at least 13/15, item *2* in the following Assumption happens. Obviously, the above two probabilities can be changed to $1-\delta$ for any constant δ without changing the asymptotic query complexity.

Assumption 17. *Throughout this section we assume that there are X, $q\le k$, $I=\{\ell_1,\ldots,\ell_q\}$ and $V=\{v^{(\ell_1)},\ldots,v^{(\ell_q)}\}$ such that: For every $\ell\in I$, $v^{(\ell)}_{\overline{X}}=0$ and $f(v^{(\ell)})\neq f(0_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})$. That is, $v^{(\ell)}\in V$ is a witness of $f(x_X\circ 0_{\overline{X}})$ for X_ℓ and*

1. *If $f\in C$ (and therefore is k-junta)*
   - *$f(x_X\circ 0_{\overline{X}})\in C$.*
   - *Each $x(X_\ell)$, $\ell\in I$ contains exactly one relevant variable.*
   - *For every $\ell\in I$, $f(x_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})$ is a literal in $\{x_{\tau(\ell)},\overline{x_{\tau(\ell)}}\}$.*
2. *If f is ϵ-far from every function in C then*

- *$f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-close to f with respect to $\mathcal{D}$ and therefore $f(x_X \circ 0_{\overline{X}})$ is $(2\epsilon/3)$-far from every function in C with respect to $\mathcal{D}$.*
- *For every $\ell \in I$, $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is $(1/30)$-close to a literal in $\{x_{\tau(\ell)}, \bar{x}_{\tau(\ell)}\}$ with respect to the uniform distribution.*

We will also use the set of indices $\Gamma := \{\tau(\ell_1), \dots, \tau(\ell_q)\}$. Notice that if f is k-junta then $x(\Gamma)$ are the relevant variables of f.

We remind the reader that for a projection $\pi : X \to X$ the string $x(\pi)$ is defined as $x(\pi)_j = x_{\pi(j)}$ for every $j \in X$. Define the projection $\pi_{f,I} : X \to X$ that satisfies: For every $\ell \in I$ and every $j \in X_\ell$, $\pi_{f,I}(j) = \tau(\ell)$. Define the function $F(x_\Gamma) = F(x_{\tau(\ell_1)}, \dots, x_{\tau(\ell_q)}) := f(x(\pi_{f,I}) \circ 0_{\overline{X}})$. That is, F is the function that results from substituting in $f(x_X \circ 0_{\overline{X}})$ for every $\ell \in I$ and every x_i, $i \in X_\ell$, the variable $x_{\tau(\ell)}$. Note here that the tester does not know $\tau(\ell_1), \dots, \tau(\ell_q)$.

We now show how to query F by querying f

Lemma 18. *For the function F we have*

1. *Given $(y_1, \dots, y_q)$, computing $F(y_1, \dots, y_q)$ can be done with one query to f.*
2. *Given $x \in \{0,1\}^n$ and δ, there is an algorithm that makes $O(k \log(k/\delta))$ queries and, with probability at least $1 - \delta$, either discovers that some X_i, $i \in I$ contains at least two relevant variables in f (and therefore, whp, f is not k-junta) and then rejects or computes $z = (x_{\tau(\ell_1)}, \dots, x_{\tau(\ell_q)})$ and $F(z)$.*

Proof. 1 is immediate. To prove *2* we use Lemma 15. We run **RelVarValues**(x, X, V, I, δ). If it rejects then $\{G_{\ell,0}, G_{\ell,1}\} \neq \{0, h\}$ for some $\ell \in I$ and therefore $G_{\ell,0}, G_{\ell,1} > 0$. This implies that for some $b, b' \in \{0,1\}^n$, $f(b_{Y_{\ell,0}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}}) \neq f(\overline{b_{Y_{\ell,0}}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}})$ and $f(b'_{Y_{\ell,1}} \circ b'_{Y_{\ell,0}} \circ v^{(\ell)}_{\overline{X_\ell}}) \neq f(\overline{b'_{Y_{\ell,1}}} \circ b'_{Y_{\ell,0}} \circ v^{(\ell)}_{\overline{X_\ell}})$. Since $X_\ell = Y_{\ell,0} \cup Y_{\ell,1}$ and $Y_{\ell,0} \cap Y_{\ell,1} = \emptyset$, the set $x(X_\ell)$ contains at least two relevant variables in f.

If for every ℓ we have $\{G_{\ell,0}, G_{\ell,1}\} = \{0, h\}$ then, by Lemma 15, with probability at least $1 - \delta$, the procedure outputs z where for every ℓ, $z_\ell = x_{\tau(\ell)}$. Then using *1* we compute $F(z)$. Since by Lemma 16, **RelVarValue** makes $O(k \log(k/\delta))$ queries, the result follows. □

We now give the key lemma for the first tester

Lemma 19. *Let $C \subseteq k-$Junta be a class that is closed under variable and zero projections and f be any Boolean function. Let $F(x_\Gamma) = f(x(\pi_{f,I}) \circ 0_{\overline{X}})$ where $\Gamma = \{\tau(\ell) | \ell \in I\}$ and $C(\Gamma)$ be the set of all functions in C that their relevant variables are $x(\Gamma)$. If Assumption 17 is true, then*

1. *If $f \in C$ then $f(x_X \circ 0_{\overline{X}}) = F \in C(\Gamma)$.*
2. *If f is ϵ-far from every function in C with respect to $\mathcal{D}$ then either*
   (a) *$f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-far from F with respect to $\mathcal{D}$,*
   or
   (b) *F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$.*

Proof. We first prove *1.* If $f \in C$, then since C is closed under variable and zero projection $f(x_X \circ 0_{\overline{X}}) \in C$. We have $f(x_X \circ 0_{\overline{X}}) = f(\circ_{\ell\in I} x_{X_\ell} \circ 0_{\overline{X}})$ and, by Assumption 17, every $x(X_\ell)$, $\ell \in I$, contains exactly one relevant variable $x_{\tau(\ell)}$ of $f(x_X \circ 0_{\overline{X}})$. Therefore, $f(x_X \circ 0_{\overline{X}})$ is a function that depends only on the variables $x_{\tau(\ell)}$, $\ell \in I$. By the definition of $x(\pi_{f,I})$ and since $\tau(\ell) \in X_\ell$ we have $x(\pi_{f,I})_{\tau(\ell)} = x_{\tau(\ell)}$ and therefore $f(x_X \circ 0_{\overline{X}}) = f(x(\pi_{f,I}) \circ 0_{\overline{X}}) = F$.

We now prove *2.* Suppose, for the contrary, $f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-close to F with respect to $\mathcal{D}$ and F is $(\epsilon/3)$-close to some function $g \in C(\Gamma)$ with respect to $\mathcal{D}$. Then $f(x_X \circ 0_{\overline{X}})$ is $(2\epsilon/3)$-close to g with respect to $\mathcal{D}$. Since, by Assumption 17, $f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-close to f with respect to $\mathcal{D}$ we get that f is ϵ-close to $g \in C$ with respect to $\mathcal{D}$. A contradiction.

□

In the following two subsections we discuss how to test the closeness of $f(x_X \circ 0_{\overline{X}})$ to F and F to $C(\Gamma)$. We will assume all the procedures in the following subsections have access to X, V, I that satisfies Assumption 17.

4.1 Testing the Closeness of $f(x_X \circ 0_{\overline{X}})$ to F

```
**Close**$fF(f, \mathcal{D}, \epsilon, \delta)$
*Input*: Oracle that accesses a Boolean function $f$ and $\mathcal{D}$.
*Output*: Either "reject" or "OK"

Define $F \equiv f(x(\pi_{f,I}) \circ 0_{\overline{X}})$.
Repeat $t = (3/\epsilon)\ln(2/\delta)$ times
    Choose $u \in \mathcal{D}$.
    $z \leftarrow$ **RelVarValue**$(u, X, V, I, \delta/(2t))$ .
    If $f(u_X \circ 0_{\overline{X}}) \neq F(z)$ then Output("reject")
Return "OK".
```

Figure 6: A procedure that tests whether $f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-far from F with respect to $\mathcal{D}$.

We now give the procedure **Close**fF that tests whether $f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-far from F with respect to $\mathcal{D}$. See Figure 6.

Lemma 20. *For any ϵ, a constant δ, and (X, V, I) that satisfies Assumption 17, procedure* **Close**fF *makes $O((k/\epsilon)\log(k/\epsilon))$ queries and*

1. *If $f \in C$ then* **Close**fF *returns OK.*
2. *If $f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-far from F with respect to $\mathcal{D}$ then, with probability at least $1-\delta$,* **Close**fF *rejects.*

Proof. **Close**fF draws $t = (3/\epsilon)\ln(2/\delta)$ random $u^{(i)} \in \{0,1\}^n$, $i = 1, \ldots, t$ according to the distribution $\mathcal{D}$. It finds $z^{(i)} = u^{(i)}_\Gamma$ and if $F(u^{(i)}_\Gamma) = f(u^{(i)}_X \circ 0_{\overline{X}})$ for all i then it returns "OK". Otherwise it rejects.

If $f \in C$ then, by *1* in Lemma 19, $F(u_\Gamma^{(i)}) = f(u_X^{(i)} \circ 0_{\overline{X}})$ for every i. By Lemma 14 and Assumption 17, $z^{(i)} = u_\Gamma^{(i)}$ for all i, and therefore **Close**fF returns OK.

Suppose now $f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-far from F with respect to $\mathcal{D}$. By *2* in Lemma 18, **RelVarValue** makes $O(k\log((kt)/\delta))$ queries and computes $F(u_\Gamma^{(i)})$, $i = 1, \ldots, t$, with failure probability at most $\delta/2$. Then the probability that it fails to reject is at most $(1-\epsilon/3)^t \le \delta/2$. This gives the result.

Therefore, **Close**fF makes $O((k/\epsilon)\log(k/\epsilon))$ queries and satisfies *1* and *2*. □

4.2 Testing the Closeness of F to $C(\Gamma)$

```
Close$FC\mathcal{D}(f, \mathcal{D}, \epsilon, \delta)$
Input: Oracles that access a Boolean function $f$ and $\mathcal{D}$.
Output: Either "reject" or "OK"

$C^* \leftarrow C(\Gamma)$
Repeat $\tau = (12/\epsilon)\ln(2|C^*|/\delta)$ times
    Choose $u \in \mathcal{D}$.
    $z \leftarrow$RelVarValue$(u, X, V, I, 1/2)$ .
    For every $g \in C^*$
        If $g(z) \neq F(z)$ then $C^* \leftarrow C^* \backslash \{g\}$.
    If $C^* = \emptyset$ then Output("Reject")
Return "OK"

Close$FCU(f, \epsilon, \delta)$
Input: Oracle that accesses a Boolean function $f$.
Output: Either "reject" or "OK"

$C^* \leftarrow C(\Gamma)$
Repeat $\tau = (3/\epsilon)\ln(2|C^*|/\delta))$ times
    Choose $(z_1, \ldots, z_q) \in U$.
    For every $g \in C^*$
        If $g(z) \neq F(z)$ then $C^* \leftarrow C^* \backslash \{g\}$.
    If $C^* = \emptyset$ then Output("Reject")
Return "OK"
```

Figure 7: Two procedures that test whether F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$ and the uniform distribution, respectively.

In this section, we give the procedures **Close**$FC\mathcal{D}$ and **Close**FCU that test whether F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$ and the uniform distribution, respectively. We prove

Lemma 21. *For any ϵ and any constant δ and (X, V, I) that satisfies Assumption 17, the procedures* **Close**$FC\mathcal{D}$ *and* **Close**FCU *make $O((k\log|C(\Gamma)|)/\epsilon)$ and $O((\log|C(\Gamma)|)/\epsilon)$ queries to f, respectively, and*

1. *If $f \in C$ then* **Close***$FC\mathcal{D}$ and* **Close***FCU output OK.*

2. *If F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$ then, with probability at least $1-\delta$,* **Close***$FC\mathcal{D}$ rejects.*

3. *If F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to the uniform distribution, then with probability at least $1-\delta$,* **Close***FCU rejects.*

Both procedures run in time poly$(n, |C(\Gamma)|, 1/\epsilon)$.

Proof. The proof for **Close**FCU is similar to the proof of Lemma 20 with union bound.

For **Close**$FC\mathcal{D}$, notice that it calls **RelVarValue**$(u, X, V, I, 1/2)$, and therefore at each iteration, with probability $1/2$, $z = u_\Gamma$. By Chernoff's bound ((9) in Lemma 60), with probability at least $1-\delta/2$, $(3/\epsilon)\ln(2|C^*|/\delta)$ of the chosen us in the procedures satisfy $z = u_\Gamma$. Then again, by union bound, the result follows. □

4.3 Testing the Closeness of F to $C(\Gamma)$ via Learning $C(\Gamma)$

In this subsection, we show how proper learning implies testing the closeness of F to $C(\Gamma)$. The proofs are similar to the proof of Proposition 3.1.1 in [39].

Let (X, V, I) be as in Assumption 17 and $q = |I| \le k$. Let $Y = \{y_1, \ldots, y_q\}$ be a set of Boolean variables and $C(Y)$ be the set of all functions in C that depend on all the variables of Y. Notice that instead of using $C(Y)$ we could have used $C(\{x_1, \ldots, x_q\})$ but here we use the new Boolean variables y_i to avoid confusion with the variables x_i of f.

Remark 22. *In all the lemmas in this subsection and the following one, in addition to the fact that F depends on all the variables of Y, the learning algorithms can also make use of (X, V, I) that satisfies Assumption 17. This may help for some classes. For example[7], if the target function is a unate monotone function, then from the witnesses in V, we can know if F is positive or negative unate in y_i, for each variable y_i.*

The following is an immediate result that follows from the two procedures **Close**$FC\mathcal{D}$ and **Close**FCU in the previous subsection

Lemma 23. *If there is a polynomial time algorithm that given a set*

$$\mathcal{Y} = \{(y^{(1)}, \xi_1), \ldots, (y^{(t)}, \xi_t)\} \subseteq \{0,1\}^q \times \{0,1\}$$

decides whether there is a function $F \in C(Y)$ that is consistent with $\mathcal{Y}$, i.e., $F(y^{(i)}) = \xi_i$ for all $i = 1, \ldots, t$, then there is a polynomial time algorithm $\mathcal{B}_\mathcal{D}$ (resp. $\mathcal{B}_U$) that makes $O((k \log |C(\Gamma)|)/\epsilon)$ queries (resp. $O((\log |C(\Gamma)|)/\epsilon)$ queries) to f and

1. *If $f \in C$ then $\mathcal{B}_\mathcal{D}$ and $\mathcal{B}_U$ output OK.*

2. *If F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$ then, with probability at least $1-\delta$, $\mathcal{B}_\mathcal{D}$ rejects.*

3. *If F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to the uniform distribution then, with probability at least $1-\delta$, $\mathcal{B}_U$ rejects.*

[7] See the definition of unate in Subsection 5.4

We now give the reduction from exact learning

Lemma 24. *If there is a polynomial time algorithm $\mathcal{A}$ that, given as an input any constant δ, properly exactly learns $C(Y)$ with confidence parameter δ and makes $M(\delta)$ membership queries to F then there is a polynomial time algorithm $\mathcal{B}$ that, given as an input ϵ, any constant δ, makes $M(\delta/3)+O((k/\epsilon)\log(1/\epsilon))$ (resp. $M(\delta/3)+O(1/\epsilon)$) queries to f and*

1. *If $f \in C$ then, with probability at least $1-\delta$, $\mathcal{B}$ outputs OK.*

2. *If F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$ (resp. with respect to the uniform distribution) then, with probability at least $1-\delta$, $\mathcal{B}$ rejects.*

Proof. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ with confidence parameter $\delta/3$ to learn $F(y_1,\ldots,y_q)$. By Lemma 18, each membership query to F can be simulated by one membership query to f. If algorithm $\mathcal{A}$ runs more than it should, asks more than $M(\delta/3)$ membership queries or outputs $h \notin C(Y)$ then $\mathcal{B}$ rejects.

If $\mathcal{A}$ outputs $h \in C(Y)$ then the algorithm needs to distinguish whether $h(x_\Gamma)$ is equal to $F(x_\Gamma)$ or $(\epsilon/3)$-far from $F(x_\Gamma)$ with respect to the distribution $\mathcal{D}$ (resp. uniform distribution). When the distribution is uniform, the algorithm chooses $t=(3/\epsilon)\ln(3/\delta)$ strings $v^{(1)},\ldots,v^{(t)} \in \{0,1\}^q$ and if $F(v^{(i)})=h(v^{(i)})$ for all i then it outputs "OK"; otherwise it rejects.

In the distribution-free model, $\mathcal{B}$ chooses $t=(12/\epsilon)\ln(2/\delta)$ strings $u^{(i)} \in \{0,1\}^n$ according to the distribution $\mathcal{D}$. Then runs $\mathbf{RelValValue}(u^{(i)},X,V,I,1/2)$ to find, with probability $1/2$ the value of $u^{(i)}_\Gamma$, $i=1,\ldots,t$. If $F(u^{(i)}_\Gamma)=h(u^{(i)}_\Gamma)$ for all i, then it outputs "OK"; otherwise it rejects.

The analysis and correctness of the algorithm are the same as in the above proofs and Proposition 3.1.1 in [39]. □

We now give the reduction from learning from MQ and $\mathrm{ExQ}_\mathcal{D}$

Lemma 25. *If there is a polynomial time algorithm $\mathcal{A}$ that, given as an input a constant δ, any ϵ, learns $C(Y)$ with respect to the distribution $\mathcal{D}$ (resp. uniform distribution), with confident δ, accuracy ϵ, makes $M(\epsilon,\delta)$ MQ to F and $Q(\epsilon,\delta)$ $ExQ_\mathcal{D}$ (resp. ExQ_U) queries to F then there is a polynomial time algorithm $\mathcal{B}_\mathcal{D}$ (resp. $\mathcal{B}_U$) that asks*

$$O\left(M(\epsilon/12,\delta/3)+kQ(\epsilon/12,\delta/3)\log(kQ(\epsilon/12,\delta/3))+\frac{k}{\epsilon}\log\frac{1}{\epsilon}\right)$$

queries (resp.

$$O\left(M(\epsilon/12,\delta/3)+Q(\epsilon/12,\delta/3)+\frac{1}{\epsilon}\right)$$

queries) to f and

1. *If $f \in C$ then with probability at least $1-\delta$, $\mathcal{B}_\mathcal{D}$ and $\mathcal{B}_U$ output OK.*

2. *If F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$ then, with probability at least $1-\delta$, $\mathcal{B}_\mathcal{D}$ rejects.*

3. *If F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to the uniform distribution then, with probability at least $1-\delta$, $\mathcal{B}_U$ rejects.*

Proof. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ with confidence parameter $\delta/3$ and accuracy $\epsilon/12$. By Lemma 18, every membership query to $F(y)$ can be simulated with one membership query to f. Every $\text{ExQ}_{\mathcal{D}'}$ (resp. ExQ) for the induced distribution $\mathcal{D}'$ of $\mathcal{D}$ on the coordinates Γ, can be simulated with one $\text{ExQ}_{\mathcal{D}}$ and $k\log(3kQ(\epsilon/12,\delta/3)/\delta)$ membership queries (resp. one ExQ) with failure probability $\delta/(3Q(\epsilon/12,\delta/3))$, and therefore, with failure probability $\delta/3$ for all the $\text{ExQ}_{\mathcal{D}'}$ queries asked in the learning algorithm.

If algorithm $\mathcal{A}$ runs more than it should, asks more than $Q(\epsilon/12,\delta/3)$ $\text{ExQ}_{\mathcal{D}'}$, asks more than $M(\epsilon/12,\delta/3)$ MQ or outputs $h\not\in C(Y)$ then $\mathcal{B}_{\mathcal{D}}$ rejects. If $\mathcal{A}$ outputs $h\in C(Y)$ then, with probability at least $1-(2\delta/3)$,

1. If $F\in C(\Gamma)$ then F is $(\epsilon/12)$-close to h
2. If F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ then F is $(\epsilon/4)$-far from h.

Now, using Chernoff's bound in Lemma 60, algorithm $\mathcal{B}$, can estimate the distance between F and h with accuracy $\epsilon/24$ and confidence $\delta/6$ using $O((\log(1/\delta))/\epsilon)$ strings chosen according to the distribution $\mathcal{D}'$. This can be done using $O((\log(1/\delta))/\epsilon)$ queries in the uniform model and $O((k/\epsilon)(\log(1/\delta)\log(1/(\epsilon\delta)))$ with confidence $\delta/6$ in the distribution-free model. □

Define the oracle $\text{WExQ}_{\mathcal{D}}$ (Weak $\text{ExQ}_{\mathcal{D}}$) that returns with probability $1/2$ a $x\in\{0,1\}^n$ according to the distribution $\mathcal{D}$ and with probability $1/2$ an arbitrary $x\in\{0,1\}^n$. In some of the learning algorithms given in the sequel, the algorithms still work if we replace the oracle $\text{ExQ}_{\mathcal{D}}$ with $\text{WExQ}_{\mathcal{D}}$. In that case, we can save the factor of $\log(3kQ(\epsilon/12,\delta/3)/\delta)$ in the query complexity of Lemma 25 in the distribution-free setting. We will discuss this in Section **??**.

4.4 The First Tester

We are now ready to give the first tester.

Consider the tester **Tester**C in Figure 8. Note that the tester rejects if any one of the procedures called by the tester rejects. We prove

Theorem 26. *Let $C\subseteq k-Junta$ that is closed under zero and variable projections. Then*

1. *There is a poly$(|C(\Gamma)|,n,1/\epsilon)$ time two-sided adaptive algorithm,* **Tester**C*, for ϵ-testing C that makes $\tilde{O}((1/\epsilon)(k+\log|C(\Gamma)|))$ queries. That is*
   - *(a) If $f\in C$ then, with probability at least $2/3$,* **Tester**C *accepts.*
   - *(b) If f is ϵ-far from every function in C with respect to the uniform distribution then, with probability at least $2/3$,* **Tester**C *rejects.*
2. *There is a poly$(|C(\Gamma)|,n,1/\epsilon)$ time two-sided distribution-free adaptive algorithm,* **Tester**C*, for ϵ-testing C that makes $\tilde{O}((k/\epsilon)\log(2|C(\Gamma)|))$ queries. That is*
   - *(a) If $f\in C$ then, with probability at least $2/3$,* **Tester**C *accepts.*
   - *(b) If f is ϵ-far from every function in C with respect to the distribution $\mathcal{D}$ then, with probability at least $2/3$,* **Tester**C *rejects.*

```
**Tester**$C(f,\mathcal{D},\epsilon)$
*Input*: Oracle that accesses a Boolean function $f$ and $\mathcal{D}$.
*Output*: If any one of the procedures reject
          then "reject" or "accept"

1.  $(X,V,I) \leftarrow$ **ApproxTarget**$(f,\mathcal{D},\epsilon,1/3)$.
2.  **TestSets**$(X,V,I)$.
3.  Define $F \equiv f(x(\pi_{f,I}) \circ 0_{\overline{X}})$
4.  **Close**$fF(f,\mathcal{D},\epsilon,1/15)$

 **For any distribution**
5. **Close**$FC\mathcal{D}(f,\mathcal{D},\epsilon,1/15)$
6.  Return "accept"

 **For the uniform distribution**
5. **Close**$FCU(f,\epsilon,1/15)$
6.  Return "accept"
```

Figure 8: A tester for subclasses C of k-Junta

Proof. We prove *1a* and *2a*. Let $f \in C$. Consider step 1 in **Tester**C. By Lemma 4 and 8, with probability at least 2/3, **ApproxTarget** outputs (X,V,I) that satisfies Assumption 17. Now with this assumption we have: By Lemma 11, **TestSets** in step 2 does not reject. By Lemma 20, **Close**fF in step 4 does not reject. By Lemma 21, **Close**$FC\mathcal{D}$ and **Close**FCU in step 5 do not reject. Therefore, with probability at least 2/3 the tester accepts.

We now prove *1b* and *2b*. Suppose f is ϵ-far from every function in C with respect to $\mathcal{D}$. The proof for the uniform distribution is similar. If in step 1 **ApproxTarget** outputs (X,V,I) then by Lemma 9, with probability at least 14/15, $f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-close to f. If in step 2 **TestSets** does not reject then, by Lemma 12, with probability at least 14/15, for all $\ell \in I$, $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is (1/30)-close to a literal $\{x_{\tau(\ell)}, \overline{x_{\tau(\ell)}}\}$. Therefore with probability at least 13/15, Assumption 17 is true. Then by Lemma 19, either $f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-far from F with respect to $\mathcal{D}$, or F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$. If $f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-far from F with respect to $\mathcal{D}$ then by Lemma 20, with probability at least 14/15, **Close**fF rejects. If F is $(\epsilon/3)$-far from every function in $C(\Gamma)$ with respect to $\mathcal{D}$ then by Lemma 21, with probability at least 14/15, **Close**FC rejects. By the union bound the probability that the tester rejects is at least 2/3.

The query complexity follows from Lemmas 10, 13, 20 and 21. □

If we replace **Close**$FC\mathcal{D}$ and **Close**FCU with the testers in Lemma 23, 24 and 25, we get the following results

Theorem 27. *If there is a polynomial time algorithm that given a set*

$$\mathcal{Y} = \{(y^{(1)},\xi_1),\ldots,(y^{(t)},\xi_t)\} \subseteq \{0,1\}^q \times \{0,1\}$$

decides whether there is a function $F \in C(Y)$ that is consistent with $\mathcal{Y}$, then

1. *There is a polynomial time two-sided adaptive algorithm for ϵ-testing C that makes $\tilde{O}((1/\epsilon)(k+\log|C(\Gamma)|))$ queries.*

2. *There is a polynomial time two-sided distribution-free adaptive algorithm for ϵ-testing C that makes $\tilde{O}((k/\epsilon)\log(2|C(\Gamma)|))$ queries.*

Theorem 28. *If there is a polynomial time algorithm $\mathcal{A}$ that, given as an input any constant δ, properly exactly learns $C(Y)$ with confidence parameter δ and makes $M(\delta)$ membership queries then*

1. *There is a polynomial time two-sided adaptive algorithm for ϵ-testing C that makes $M(1/24)+\tilde{O}(k/\epsilon)$ queries.*

2. *There is a polynomial time two-sided distribution-free adaptive algorithm for ϵ-testing C that makes $M(1/24)+\tilde{O}(k/\epsilon)$ queries.*

Theorem 29. *If there is a polynomial time algorithm $\mathcal{A}$ that, given as an input a constant δ and any ϵ, learns $C(Y)$, with confident δ, accuracy ϵ, makes $M(\epsilon,\delta)$ MQ and $Q(\epsilon,\delta)$ ExQ_U (resp. $ExQ_{\mathcal{D}}$) then*

1. *There is a polynomial time two-sided adaptive algorithm for ϵ-testing C that makes*
$$\tilde{O}\left(M(\epsilon/12,1/24)+Q(\epsilon/12,\delta/3)+\frac{k}{\epsilon}\right)$$
queries.

2. *There is a polynomial time two-sided distribution-free adaptive algorithm for ϵ-testing C that makes*
$$\tilde{O}\left(M(\epsilon/12,1/24)+kQ(\epsilon/12,1/24)+\frac{k}{\epsilon}\right)$$
queries.

Finally, one trivial but useful result is

Theorem 30. *If there is a polynomial time algorithm $\mathcal{A}$ that, given as an input any constant δ and any ϵ makes $M(\epsilon,\delta)$ MQs and $Q(\epsilon,\delta)$ ExQ_U (resp. $ExD_{\mathcal{D}}$) and distinguish between $F\in C(Y)$ and F ϵ-far from every function in $C(Y)$ with respect to the uniform distribution (resp. with respect to the distribution $\mathcal{D}$) then*

1. *There is a polynomial time two-sided adaptive algorithm for ϵ-testing C that makes*
$$\tilde{O}\left(M(\epsilon/12,1/24)+Q(\epsilon/12,1/24)+\frac{k}{\epsilon}\right)$$
queries.

2. *There is a polynomial time two-sided distribution-free adaptive algorithm for ϵ-testing C that makes*
$$\tilde{O}\left(M(\epsilon/12,1/24)+kQ(\epsilon/12,1/24)+\frac{k}{\epsilon}\right)$$
queries.

5 Results

In this section we define the classes and give the results for the classes k-Junta, k-Linear, k-Term, s-Term Monotone r-DNF, size-s Decision Tree, size-s Branching Program, Functions with Fourier Degree at most d, Length-k Decision List and s-Sparse Polynomial of Degree d.

We will use words that are capitalized for classes and non-capitalized words for functions. For example, k-Junta is the class of all k-juntas.

5.1 Testing k-Junta

For k-Junta in uniform distribution framework, Ficher et al. [33] introduced the junta testing problem and gave a non-adaptive algorithm that makes $\tilde{O}(k^2)/\epsilon$ queries. Blais in [7] gave a non-adaptive algorithm that makes $\tilde{O}(k^{3/2})/\epsilon$ queries and in [8] an adaptive algorithm that makes $O(k\log k + k/\epsilon)$ queries. On the lower bounds side, Fisher et al. [33] gave an $\Omega(\sqrt{k})$ lower bound for non-adaptive testing. Chockler and Gutfreund [28] gave an $\Omega(k)$ lower bound for adaptive testing and, recently, Sağlam in [56] improved this lower bound to $\Omega(k\log k)$. For the non-adaptive testing Chen et al. [24] gave the lower bound $\tilde{\Omega}(k^{3/2})/\epsilon$.

For testing k-junta in the distribution-free model, Chen et al. [46] gave a one-sided adaptive algorithm that makes $\tilde{O}(k^2)/\epsilon$ queries and proved a lower bound $\Omega(2^{k/3})$ for any non-adaptive algorithm. The result of Halevy and Kushilevitz in [42] gives a one-sided non-adaptive algorithm that makes $O(2^k/\epsilon)$ queries. The adaptive $\Omega(k\log k)$ uniform-distribution lower bound from [56] trivially extends to the distribution-free model. Bshouty [15] gave a two-sided adaptive algorithm that makes $\tilde{O}(1/\epsilon)k\log k$ queries.

Our algorithm in this paper gives

Theorem 31. *For any $\epsilon > 0$, there is a polynomial time two-sided distribution-free adaptive algorithm for ϵ-testing k-Junta that makes $\tilde{O}(k/\epsilon)$ queries.*

Proof. We use Theorem 30. Since every $F(Y)$ is in k-Junta(Y), the algorithm $\mathcal{A}$ always accepts. Therefore, we have $M = Q = 0$, and the algorithm makes $\tilde{O}(k/\epsilon)$ queries. □

5.2 Testing k-Linear

The function is linear if it is a sum (over the binary field F_2) of variables. The class Linear is the class of all linear functions. The class k-Linear is Linear$\cap k$-Junta. That is, the class of functions that are the sum of at most k variables.

Blum et al. [13] showed that there is an algorithm for testing Linear under the uniform distribution that makes $O(1/\epsilon)$ queries. For testing k-Linear under the uniform distribution, Fisher, et al. [33] gave a tester that makes $O(k^2/\epsilon)$ queries. They also gave the lower bound $\Omega(\sqrt{k})$ for non-adaptive algorithms. Goldreich [35], proved the lower bound $\Omega(k)$ for non-adaptive algorithms and $\Omega(\sqrt{k})$ for adaptive algorithms. Then Blais et al. [9] proved the lower bound $\Omega(k)$ for adaptive algorithms. Blais and Kane, in [10], gave the lower bound $k - o(k)$ for adaptive algorithms and $2k - o(k)$ for non-adaptive algorithms.

Testing k-Linear can be done by first testing if the function is k-Junta and then testing if it is Linear. Therefore, there is an adaptive algorithm for ϵ-testing k-Linear under the uniform distribution that makes $\tilde{O}(k/\epsilon)$ queries.

In this paper we prove

Theorem 32. *For any $\epsilon > 0$, there is a polynomial time two-sided distribution-free adaptive algorithm for ϵ-testing k-Linear that makes $\tilde{O}(k/\epsilon)$ queries.*

Proof. We use Theorem 29. Here $C(Y) = \{y_1 + \cdots + y_q\}$ contains one function and therefore the learning algorithm just outputs $y_1 + \cdots + y_q$. Therefore $M = Q = 0$ and the result follows. □

5.3 Testing k-Term

A *term* (or *monomial*) is a conjunction of literals and *Term* is the class of all terms. A *k-term* is a term with at most k literals and *k-Term* is the class of all k-terms.

In the uniform distribution model, Pernas et al. [51], gave a tester for k-terms that makes $O(1/\epsilon)$ queries in the uniform model. We give the same result in the next section. In this paper we prove the following result for the distribution-free model. When $k = n$, better results can be found in [34, 32].

Theorem 33. *For any $\epsilon > 0$, there is a polynomial time two-sided distribution-free adaptive algorithm for ϵ-testing k-Term that makes $\tilde{O}(k/\epsilon)$ queries.*

Proof. Recall that $x_i^0 = x_i$ and $x_i^1 = \overline{x_i}$. Here $C(Y) = \{y_1^{\xi_1} \wedge \cdots \wedge y_q^{\xi_q} | \xi \in \{0,1\}^q\}$ contains 2^q functions. We use Theorem 29 with Remark 22. Since V contains witnesses for each variable it follows that ξ_i are known. Just take any string a that satisfies $F(a) = 1$ and then $\xi_i = \overline{a_i}$. Therefore $M = Q = 0$ and the result follows. □

5.4 Testing s-Term Monotone r-DNF

A *DNF* is a disjunction of terms. An *r-DNF* is a disjunction of r-terms. The class *s-Term r-DNF* is the class of all r-DNFs with at most s terms. The class *s-Term Monotone r-DNF* is the class of all r-DNFs with at most s terms with no negated variables. A DNF f is called *unate DNF* if there is $\xi \in \{0,1\}^n$ such that $f(x_1^{\xi_1}, \ldots, x_n^{\xi_n})$ is monotone DNF. If $\xi_i = 0$ then we say that f *is positive unate in x_i*; otherwise we say that f *is negative unate in x_i*. Similarly, one can define the classes Unate DNF, Unate s-DNF etc.

We first give a learning algorithm for s-Term Monotone r-DNF. The algorithm is in Figure 9. In the algorithm, we use $P_{1/r}$ for the probability distribution over the strings $b \in \{0,1\}^n$ where each coordinate b_i is chosen randomly and independently to be 1 with probability $1 - 1/r$ and 0 with probability $1/r$. For two strings $x, y \in \{0,1\}^n$ we denote $x * y = (x_1y_1, \ldots, x_ny_n)$ where $x_iy_i = x_i \wedge y_i$. The procedure **FindMinterm**(f, a) flips bits that are one in a to zero as long as $f(a) = 1$.

We now show

Lemma 34. *If the target function f is s-term monotone r-DNF then for any constant δ, algorithm* **LearnMonotone** *asks $O(s/\epsilon)$ ExQ_D and $O(sr\log(ns))$ MQ and, with probability at least $1 - \delta$, learns an s-term monotone r-DNF, h, that satisfies $\mathbf{Pr}_{\mathcal{D}}[h \neq f] \leq \epsilon$.*

Proof. We first show that if in the mth iteration of the algorithm (steps 3-11) the function h contains ℓ terms of f and $f(a) = 1$ and $h(a) = 0$ then, with probability at least $1 - \delta/(2s)$, steps 5 to 11 adds to h a new term of f. This implies that in the $(m+1)$th iteration h contains $\ell + 1$ terms of f. Then, since the number of terms of f is at most s, with probability at least $1 - \delta/2$, all

```
**LearnMonotone**$(f, \mathcal{D}, \epsilon, \delta, s, r)$
*Input*: Oracle that accesses a Boolean function $f$
         that is $s$-term monotone $r$-DNF and $\mathcal{D}$.
*Output*: $h$ that is $s$-term monotone $r$-DNF

$h \leftarrow 0$.
Repeat $4(s/\epsilon)\log(1/\delta)$ times.
    Choose $a \in \mathcal{D}$.
    If $f(a) = 1$ and $h(a) = 0$ then
        $t \leftarrow 0$
        While $t \le \alpha := 4r\ln(2ns/\delta)$ and $wt(a) > r$ do
            $t \leftarrow t + 1$; If $t = \alpha + 1$ Output "fail"
            Choose $y \in P_{1/r}$
            If $f(a * y) = 1$ then $a \leftarrow a * y$
        $a \leftarrow$**FindMinterm**$(f, a)$
        $h \leftarrow h \vee \prod_{a_i=1} x_i$
Output $h$
```

Figure 9: A learning algorithm for s-Term Monotone r-DNF

the terms in h are terms in f. We then show that, with probability at least $1 - \delta/2$, the procedure outputs h that satisfies $\mathbf{Pr}_{\mathcal{D}}[h \ne f] \le \epsilon$.

First notice that if $f(a) = 1$ and $h(a) = 0$ then for every $y \in \{0,1\}^n$, $h(a * y) = 0$. This follows from the fact that h is monotone and $a * y \le a$. Therefore if a receives the values $a^{(1)}, \ldots, a^{(\tau)}$ in the While loop then $f(a^{(i)}) = 1$ and $h(a^{(i)}) = 0$ for all $i = 1, \ldots, \tau$. We also have $a^{(i+1)} = a^{(i)}$ if $f(a^{(i)} * y) = 0$ and $a^{(i+1)} = a^{(i)} * y$ if $f(a^{(i)} * y) = 1$. Consider the random variable $W_i = wt(a^{(i)}) - r$. We will now compute $\mathbf{E}[W_{i+1}|W_i]$. Since $f(a^{(i)}) = 1$ and $h(a^{(i)}) = 0$, there is a term T in f that is not in h that satisfies $T(a^{(i)}) = 1$. Suppose $T = x_{j_1} x_{j_2} \cdots x_{j_{r'}}$, $r' \le r$. Then $a^{(i)}_{j_1} = \cdots = a^{(i)}_{j_{r'}} = 1$. Consider another $r - r'$ entries in $a^{(i)}$ that are equal to 1, $a^{(i)}_{j_{r'+1}} = \cdots = a^{(i)}_{j_r} = 1$. Such entries exist because of the condition $wt(a) > r$ of the While command. Note that $wt(a^{(i)}) = W_i + r$. Let $j_{r+1}, \ldots, j_{r+W_i}$ be the other entries of $a^{(i)}$ that are equal to 1. Let A be the event that, for the $y \in P_{1/r}$ chosen at this stage, $y_{j_1} = \cdots = y_{j_r} = 1$. Notice that if event A happens then $T(a^{(i+1)}) = f(a^{(i+1)}) = 1$ and $a^{(i+1)} = a^{(i)} * y$. Then

$$\begin{aligned}
\mathbf{E}[W_{i+1}|W_i] &= \mathbf{E}[W_{i+1}|W_i, A]\mathbf{Pr}[A] + \mathbf{E}[W_{i+1}|W_i, \bar{A}]\mathbf{Pr}[\bar{A}] \\
&\le \mathbf{E}[W_{i+1}|W_i, A]\left(1 - \frac{1}{r}\right)^r + W_i\left(1 - \left(1 - \frac{1}{r}\right)^r\right) \qquad (1) \\
&= W_i\left(1 - \frac{1}{r}\right)^{r+1} + W_i\left(1 - \left(1 - \frac{1}{r}\right)^r\right) \qquad (2) \\
&= W_i\left(1 - \frac{1}{r}\left(1 - \frac{1}{r}\right)^r\right) \le W_i\left(1 - \frac{1}{4r}\right).
\end{aligned}$$

The inequality in (1) follows from the fact that $W_{i+1} \le W_i$ and (2) follows from the fact that the expected number of ones in $y_{j_{r+1}}a_{j_{r+1}},\ldots,y_{j_{r+W_i}}a_{j_{r+W_i}}$ is $(1-1/r)W_i$.

Therefore $\mathbf{E}[W_i]\le n(1-1/(4r))^i$. The probability that the algorithm fails is the probability that $t=4r\ln(2ns/\delta)$. By Markov's Bound, Lemma 58, this is bounded by

$$\mathbf{Pr}[wt(a^{(t)})>r]=\mathbf{Pr}[W_t>1]\le \mathbf{E}[W_t]\le n\left(1-\frac{1}{4r}\right)^t\le\frac{\delta}{2s}.$$

This completes the first part of the proof.

Now we show that, with probability at least $1-\delta/2$, the procedure outputs h that satisfies $\mathbf{Pr}_{\mathcal{D}}[f\ne h]\le\epsilon$. Let $h^{(i)}$ be the function h at iteration $i=1,2,\ldots,w$. Since $h^{(1)}\implies h^{(2)}\implies\cdots\implies h^{(w)}=h\implies f$, if $\mathbf{Pr}_{\mathcal{D}}[f\ne h]>\epsilon$ then $\mathbf{Pr}_{\mathcal{D}}[f\ne h^{(i)}]>\epsilon$ for all i. Therefore, the probability that $\mathbf{Pr}_{\mathcal{D}}[f\ne h]>\epsilon$ is less than the probability that for $v=4s\log(1/\delta)/\epsilon$ strings $a^{(1)},\ldots,a^{(v)}$ chosen independently at random according to the distribution $\mathcal{D}$, less than s of them satisfies $g_i(a^{(i)})\ne f(a^{(i)})$ for Boolean functions g_i that satisfy $\mathbf{Pr}_{\mathcal{D}}[g_i\ne f]\ge\epsilon$. By Chernoff's bound, Lemma 60, this probability is less than $\delta/2$.

The algorithm asks at most $4s\log(1/\delta)/\epsilon=O(s/\epsilon)$ $\mathrm{ExQ}_{\mathcal{D}}$ and at most $s\cdot 4r\ln(2ns/\delta)=O(sr\log(ns))$ MQ. □

Now we show

Theorem 35. *For any $\epsilon>0$, there is a polynomial time two-sided distribution-free adaptive algorithm for ϵ-testing s-Term Monotone r-DNF that makes $\tilde{O}(rs^2/\epsilon)$ queries.*

Proof. The number of relevant variables in any s-term monotone r-DNF is at most $q\le k=sr$. By Lemma 34, $C(Y)$ can be learned with constant confidence δ and accuracy ϵ in $M=\tilde{O}(sr\log(qs))=\tilde{O}(sr)$ MQ and $O(s/\epsilon)$ $\mathrm{ExQ}_{\mathcal{D}}$. By Theorem 29, there is a distribution-free tester for s-Term Monotone r-DNF that makes $\tilde{O}(s^2r/\epsilon)$ queries. □

Theorem 36. *For any $\epsilon>0$, there is a polynomial time two-sided distribution-free adaptive algorithm for ϵ-testing s-Term Unate r-DNF that makes $\tilde{O}(rs^2/\epsilon)$ queries.*

Proof. The set of witnesses tells us, for every variable x_i, if f is positive unate in x_i or negative unate. If $f(v^{(\ell)})=1$, $f(0_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})=0$ then f is positive unate in $x_{\tau(\ell)}$ and if $f(v^{(\ell)})=0$, $f(0_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})=1$ then f is negative unate in $x_{\tau(\ell)}$. Then the result immediately follows from Theorem 35. □

5.5 Testing Size-s Decision Tree and Size s Branching Program

A *decision tree* is a rooted binary tree in which each internal node is labeled with a variable x_i and has two children. Each leaf is labeled with an output from $\{0,1\}$. A decision tree computes a Boolean function in an obvious way: given an input $x\in\{0,1\}^n$, the value of the function on x is the output in the leaf reached by starting at the root and going left or right at each internal node according to whether the variable's value in x is 0 or 1, respectively. The *size* of a decision tree is the number of leaves of the tree. The class size-s Decision Tree is the class of all decision trees of size s.

A *branching program* is a rooted directed acyclic graph with two sink nodes labeled 0 and 1. As in the decision tree, each internal node is labeled with a variable x_i and has two children. The two edges to the children are labeled with 0 and 1. Given an input x, the value of the branching program on x is the label of the sink node that is reached as described above. The *size* of a branching program is the number of nodes in the graph. The class size-s Branching Program is the class of all Branching Program of size s.

Diakonikolas et al. [30], gave a tester for size-s Decision Tree and size s Branching Program under the uniform distribution that makes $\tilde{O}(s^4/\epsilon^2)$ queries. Chakraborty et al. [21] improved the query complexity to $\tilde{O}(s/\epsilon^2)$. In this paper we prove

Theorem 37. *For any $\epsilon > 0$, there is a two-sided adaptive algorithm for ϵ-testing size-s Decision Tree and size-s Branching Program that makes $\tilde{O}(s/\epsilon)$ queries.*

There is a two-sided distribution-free adaptive algorithm for ϵ-testing size-s Decision Tree and size s Branching Program that makes $\tilde{O}(s^2/\epsilon)$ queries.

Proof. For decision tree, $C(Y)$ contains the decision trees with $q = |Y| \le k = s$ relevant variables. It is shown in [30] that $|C(Y)| \le (8s)^s$. For branching programs $|C(Y)| \le (s+1)^{3s}$. Now by Theorem 26 the result follows. □

5.6 Functions with Fourier Degree at most d

For convenience here we take the Boolean functions to be $f : \{-1,1\}^n \to \{-1,1\}$. Then every Boolean function has a unique Fourier representation $f(x) = \sum_{S\subseteq[n]} \hat{f}_S \prod_{i\in S} x_i$ where $\hat{f}_S$ are the *Fourier coefficients* of f. The *Fourier degree* of f is the largest $d = |S|$ with $\hat{f}_S \neq 0$.

Let C be the class of all Boolean functions over $\{-1,1\}^n$ with Fourier degree at most d. Nisan and Szegedy, [50], proved that any Boolean function with Fourier degree d must have at most $k := d2^d$ relevant variables. Diakinikolas et al. [30], show that every nonzero Fourier coefficient of $f \in C$ is an integer multiple of $1/2^{d-1}$. Since $\sum_{S\subseteq[n]} \hat{f}_S^2 = 1$, there are at most 2^{2d-2} nonzero Fourier coefficients in $f \in C$.

Diakonikolas et al. [30], gave an exponential time tester for Boolean functions with Fourier degree at most d under the uniform distribution that makes $\tilde{O}(2^{6d}/\epsilon^2)$ queries. Chakraborty et al. [21] improved the query complexity to $\tilde{O}(2^{2d}/\epsilon^2)$. In this paper we prove

Theorem 38. *For any $\epsilon > 0$, there is a poly$(2^d, n)$ time two-sided distribution-free adaptive algorithm for ϵ-testing for the class of Boolean functions with Fourier degree at most d that makes $\tilde{O}(2^{2d} + 2^d/\epsilon)$ queries.*

Proof. Bshouty gives in [16] an exact learning algorithm for such class[8] that asks $M = \tilde{O}(2^{2d}\log n)$ membership queries for any constant confidence parameter δ. Now since $q = |Y| \le k = d2^d$, by Theorem 28 the result follows. □

5.7 Testing Length k Decision List

A decision list is a sequence $f = (x_{i_1}, \xi_1, a_1), \ldots, (x_{i_s}, \xi_s, a_s)$ for any s where $\xi_i, a_i \in \{0,1\}$. This sequence represents the following function: $f(x) :=$ If $x_{i_1} = \xi_1$ then output(a_1) else if $x_{i_2} = \xi_2$ then output(a_2) else if $\cdots$ else if $x_{i_s} = \xi_s$ then output(a_s). Length-k decision list is a decision list with $s \le k$. The class Decision List is the class of all decision lists and the class Length-k Decision List is the class of all length-k decision lists.

[8] The class in [16] is the class of decision trees of depth d but the analysis is the same for the class of functions with Fourier degree at most d

It is known that this class is learnable under any distribution with $O((k\log n + \log(1/\delta))/\epsilon)$ ExQ_D, [14, 52]. This implies

Theorem 39. *For any $\epsilon > 0$, there is a polynomial time two-sided distribution-free adaptive algorithm for ϵ-testing Length-k Decision List that makes $\tilde{O}(k^2/\epsilon)$ queries.*

Proof. The result follows from Theorem 29. □

5.8 Testing s-Sparse Polynomial of Degree d

A polynomial (over the field F_2) is a sum (in the binary field F_2) of monotone terms. An s-sparse polynomial is a sum of at most s monotone terms. We say that the polynomial f is of degree d if its terms are monotone d-terms. The class s-Sparse Polynomial of Degree d is the class of all s-sparse polynomials of degree d. The class Polynomial of Degree d is the class of all polynomials of degree d.

In the uniform distribution model, Diakonikolas et al. [30], gave the first testing algorithm for the class s-Sparse Polynomial that runs in exponential time and makes $\tilde{O}(s^4/\epsilon^2)$ queries. Chakraborty et al. [21] improved the query complexity to $\tilde{O}(s/\epsilon^2)$. Diakonikolas et al. gave in [31] the first polynomial time testing algorithm that makes $poly(s, 1/\epsilon)$ queries. In [1], Alon et al. gave a testing algorithm for Polynomial of Degree d that makes $O(1/\epsilon + d2^{2d})$ queries. They also show the lower bound $\Omega(1/\epsilon + 2^d)$ for the number of queries. Combining those results we get a polynomial time testing algorithm for s-Sparse Polynomial of Degree d that makes $poly(s, 1/\epsilon) + \tilde{O}(2^{2d})$ queries. Just run the Alon et al. algorithm in [1] and then run Diakonikolas et al. algorithm in [31] and accept if both algorithms accept.

Here we prove the following Theorem.

Theorem 40. *For any $\epsilon > 0$, there is a two-sided adaptive algorithm for ϵ-testing s-Sparse Polynomial of Degree d that makes $\tilde{O}(s/\epsilon + s2^d)$ queries.*

For any $\epsilon > 0$, there is a two-sided distribution-free adaptive algorithm for ϵ-testing s-Sparse Polynomial of Degree d that makes $\tilde{O}(s^2/\epsilon + s2^d)$ queries.

We first give a learning algorithm **LearnPolynomial** for s-sparse polynomial of degree d. See Figure 10.

Lemma 41. *Let f be an s-sparse polynomial of degree d. For any constant δ, algorithm* **LearnPolynomial** *asks $O((s/\epsilon)\log s)$ $ExQ_{\mathcal{D}}$ and $O(s2^d\log(ns))$ MQ and, with probability at least $1-\delta$, learns an s-sparse polynomial of degree d, h, that satisfies $\mathbf{Pr}_{\mathcal{D}}[h \neq f] \le \epsilon$.*

Proof. Suppose $f = \sum_{M\in F} M$, where F are the set of monotone d-terms of f and $|F| = s' \le s$. Suppose, at some stage of the algorithm $h = \sum_{M\in F'} M$ where $F' \subset F$. Then $f + h = \sum_{M\in F\backslash F'} M$. Notice that F' is the set of terms of f that is found by the algorithm up to this stage and $F\backslash F'$ is the set of terms that is yet to be found. Since the number of terms of f is at most s, all we need to show is that:

1. Each time the algorithm executes steps 6-14, with probability at least $1 - \delta/(2s)$, it finds a term of $f + h$, and therefore, a new term of f.
2. Assuming 1., the algorithm, with probability at least $1-\delta/2$, outputs an s-sparse polynomial of degree d, h that satisfies $\mathbf{Pr}_D[f \neq h] \le \epsilon$.

```
LearnPolynomial($f, \mathcal{D}, \epsilon, \delta, s, d$)
Input: Oracle that accesses an $s$-sparse polynomial $f$
       of degree $d$ and $\mathcal{D}$.
Output: An $s$-sparse polynomial of degree $d$, $h$, or “fail”

1.  $h \leftarrow 0$, $t(h) \leftarrow 0$.
2.  Repeat $(s/\epsilon)\ln(3s/\delta)$ times.
3.      Choose $a \in \mathcal{D}$.
4.      $t(h) \leftarrow t(h) + 1$.
5.      If $(f + h)(a) = 1$ then
6.          $m \leftarrow 0$;
7.          While $m \le \alpha := 16 \cdot 2^d(2\ln(s/\delta) + \ln n)$ and $wt(a) > d$ do
8.              $m \leftarrow m + 1$;
9.              Choose $y \in U$
10.             If $(f + h)(a * y) = 1$ then $a \leftarrow a * y$
11.         If $wt(a) > d$ then “fail”
12.         $M \leftarrow$ Find a monotone term in $(f + h)(a * x)$
13.         $h \leftarrow h + M$
14.         $t(h) \leftarrow 0$.
15.     If $t(h) = (1/\epsilon)\ln(3s/\delta)$ then Output $h$
```

Figure 10: A learning algorithm for s-sparse polynomial of degree d

Then, by the union bound, the success probability of the algorithm is at least $1 - \delta$ and the result follows.

We first prove 1. Let $g = f + h$. Suppose that the algorithm finds a string a such that $g(a) = 1$. Then a satisfies at least one term in g. Let $M' \in F \backslash F'$ be one of them and let $d' \le d$ be the degree of M' . Then $g(a * x) = \sum_{M \in F\backslash F', M(a)=1} M$ contains M' and therefore $g(a * x)$ is not zero.

We first show that the probability that after $\alpha_1 := 16 \cdot 2^d \ln(sn/\delta)$ iterations of steps 7-10, the weight of a does not drop below $24d$ is less than $\delta/(4s)$. Then we show that, if the weight of a is less than or equal to $24d$ then the probability that after $\alpha_2 := \alpha - \alpha_1$ more iterations of steps 7-10 the weight of a does not drop below $d + 1$ is less than $\delta/(4s)$. If these two facts are true then after the algorithm finishes executing the While command, with probability at least $1 - \delta/(2s)$, the weight of a is less than $d + 1$.

It is known that for any non-zero polynomial H of degree at most d, $\mathbf{Pr}_U[H(x) = 1] \ge 1/2^d$, [18]. Since $g(a * x)$ is of degree at most d, for a random uniform string y we get $\mathbf{Pr}[g(a * y) = 1] \ge 1/2^d$. Now suppose $wt(a) \ge 24d$. By Chernoff’s bound, Lemma 60, the probability that $wt(a * y) > (3/4)wt(a)$ is at most $e^{-wt(a)/24} \le 2^{-d-1}$. Therefore, by the union bound,

$$\mathbf{Pr}[g(a * y) = 1 \text{ and } wt(a * y) \le (3/4)wt(a)] \ge 1 - \left(1 - \frac{1}{2^d} + \frac{1}{2^{d+1}}\right) \ge \frac{1}{2^{d+1}}. \quad (3)$$

The probability that after $\alpha_1 = 16 \cdot 2^d \ln(sn/\delta)$ iterations of steps 7-10, the weight of a does not drop below $24d$ is less than the probability that for $\alpha_1 = 16 \cdot 2^d \ln(sn/\delta)$ random uniform strings y, less than $\log(n)/\log(4/3)$ of them satisfies $g(a*y)=1$ and $wt(a*y)\le(3/4)wt(a)$ given that a satisfies $wt(a)\ge 24d$ and $g(a*x)\ne 0$. By (3) and Chernoff's bound this probability is less than $\delta/(4s)$.

We now show that if $wt(a)<24d$, then after $\alpha_2=\alpha-\alpha_1=16\cdot 2^d\ln(s/\delta)$ iterations of steps 7-10, with probability at least $1-\delta/(4s)$, the weight of a drops below $d+1$. Take a that satisfies $d+1\le wt(a)<24d$. Then

$$\begin{aligned}\mathbf{Pr}[g(a*y)=1 \text{ and } wt(a*y)<wt(a)] &\ge \mathbf{Pr}[g(a*y)=1]-\mathbf{Pr}[wt(a*y)=wt(a)]\\ &\ge \frac{1}{2^d}-\frac{1}{2^{d+1}}=\frac{1}{2^{d+1}}.\end{aligned}$$

Then as before, with an additional α_2 iterations of steps 7-10, with probability at least $1-\delta/(4s)$, the weight of a drops below $d+1$.

Once the weight of a is less or equal to d, the algorithm finds in step 12 a monotone term in $g(a*x)$ by building a truth table of $g(a*x)$ using at most 2^d queries and learning one of its terms. This term is in g because all the terms of $g(a*x)$ are terms of $g(x)$.

The proof that the algorithm, with probability at least $1-\delta/2$, outputs h such that $\mathbf{Pr}_D[f\ne h]\le\epsilon$ is identical to the proof in Lemma 9 for the output of **ApproxTarget**. □

We are now ready to prove Theorem 40.

Proof. By Lemma 41, Theorem 29 and since $n=|Y|=sd$, $Q=(s/\epsilon)\log s$ and $M=s2^d\log(sd)$ the result follows. □

6 Testing Classes that are Close to k-Junta

In this section, we show the result for s-term DNF in the uniform distribution model. Then in the following section, we show how to extend it to other classes.

The main idea is the following. We first run the procedure **Approx**C in Figure 11 that finds $X\subset[n]$ and $w\in\{0,1\}^n$ such that, with high probability,

1. The projection $x_X\circ w_{\overline{X}}$ removes variables from f that appear only in terms of f of size at least $c\log(s/\epsilon)$ for some large constant c.
2. $h=f(x_X\circ w_{\overline{X}})$ is $(\epsilon/8)$-close to f.

From (1) we conclude that the terms of size at most $c\log(s/\epsilon)$ in f contain all the variables of h. Since the number of terms in f is at most s the number of variables that remain in h is at most $k:=cs\log(s/\epsilon)$. From (2) we conclude that if f is ϵ-far from every s-term DNF then h is $(7\epsilon/8)$-far from every s-term DNF and therefore h is $(7\epsilon/8)$-far from every s-term DNF with at most k variables. Therefore, it is enough to distinguish whether h is an s-term DNF with at most k variables or $(7\epsilon/8)$-far from every s-term DNF with at most k variables. This can be done by the algorithm **Tester**C in the previous section

Note that removing variables that only appears in large size terms does not necessarily remove large terms in f. Therefore, h may still contain large terms even after running **ApproxTarget** in **Tester**C. To handle large terms, we can use any learning algorithm that learns h with accuracy $\epsilon/12$ and use Theorem 29.

This gives a tester for s-term DNF that makes $\tilde{O}(s^2/\epsilon)$ queries, which is not optimal. This is because the number of s-term DNF with at most k variables is $m := 2^{O(ks)}$ (and therefore the number of queries in **Tester**C is at least $O((\log m)/\epsilon) = \tilde{O}(s^2/\epsilon))$. To get an optimal query tester, we do the following. We build a tester that uses only random uniform queries for the class s-term DNF with at most k variables and terms of size at most $r = c' \log(s/\epsilon)$ where c' is a large constant and show that this tester, with high probability, works well for h. The reason for that is that when the algorithm uses random uniform queries, with high probability, all the terms of h that are of size greater than r are zero for every query. Since the number of s-term DNF with at most k variables and terms of size at most r is at most $m = 2^{O(rs\log k)}$ the number of queries in **Tester**C is at most $O(k/\epsilon + (\log m)/\epsilon) = \tilde{O}(s/\epsilon)$.

In the next subsection, we give the procedure **Approx**C that removes variables that only appear in large size terms, and in Subsection 6.2, we give the tester for s-Term DNF. Then in Section 7, we extend the above to other classes.

6.1 Removing Variables that only Appears in Large Size Terms

```
Algorithm Approx$C(f, \epsilon, \lambda)$
Input: Oracle that accesses a Boolean function $f$ and
Output: Either "$X \subseteq [n], w \in \{0,1\}^n$" or "reject"

Partition $[n]$ into $r$ sets
Set $m = c\log(s/\epsilon)$; $r = 8ms$.
Choose uniformly at random a partition $X_1, X_2, \ldots, X_r$ of $[n]$

Find a close function and relevant sets
Set $X = \emptyset$; $I = \emptyset$; $t(X) = 0$; $k = 3ms$
Repeat $M = 100\lambda k\ln(100k)/\epsilon$ times
    Choose $u, v \in U$.
    $t(X) \leftarrow t(X) + 1$
    If $f(u_X \circ v_{\overline{X}}) \neq f(u)$ then
            Binary Search to find a new relevant set $X_\ell$; $X \leftarrow X \cup X_\ell$; $I \leftarrow I \cup \{\ell\}$.
            If $|I| > k$ then output "reject" and halt.
            $t(X) = 0$.
    If $t(X) = 100\lambda\ln(100k)/\epsilon$ then
              Choose a random uniform $w$;
              Output$(X, w, f(x_X \circ w_{\overline{X}}))$.
```

Figure 11: A procedure that removes variables from f that only appears in large size terms.

We explain our technique by proving the result for s-term DNF.

We remind the reader that for a term T, the size of T is the number of variables that are in it. For a variable x and $\xi \in \{0,1\}$, $x^\xi = x$ if $\xi = 0$ and $x^\xi = \overline{x}$ if $\xi = 1$. For a term $T = x_{i_1}^{c_1} \wedge \cdots \wedge x_{i_v}^{c_v}$ we denote by $\mathrm{Va}(T) = \{x_{i_1}, \ldots, x_{i_v}\}$, the set of variables that appears in T. For a set of terms $\mathcal{T}$

we denote $\mathrm{Va}(\mathcal{T}) = \cup_{T\in\mathcal{T}}\mathrm{Va}(T)$. Here $\lambda > 1$ is any constant and we use $c = O(\log\lambda)$ to denote a large constant.

Consider the procedure **Approx**C in Figure 11. We will prove the following two Lemmas

Lemma 42. *Let f be an s-term DNF.* **Approx**C *makes $\tilde{O}(s/\epsilon)$ queries and, with probability at least $9/10$, outputs X and w such that*

1. *$f(x_X \circ w_{\overline{X}})$ is s-term DNF.*
2. *The number of relevant variables in $f(x_X \circ w_{\overline{X}})$ is at most $3cs\log(s/\epsilon) = O(s\log(s/\epsilon))$.*

Lemma 43. *Let f be ϵ-far from every s-term DNF.* **Approx**C *makes $\tilde{O}(s/\epsilon)$ queries and either rejects, or with probability at least $9/10$, outputs X and w such that $f(x_X \circ w_{\overline{X}})$ is $(1-1/\lambda)\epsilon$-far from every s-term DNF.*

We first prove Lemma 42

Proof. Consider an s-term DNF, $f = T_1 \vee T_2 \vee \cdots \vee T_{s'}$, where $s' \le s$. Let $\mathcal{T} = \{T_1, \ldots, T_{s'}\}$. Let $\mathcal{T}_1 = \{T \in \mathcal{T} : |\mathrm{Va}(T)| \le m\}$ be the set of terms in $\mathcal{T}$ of size at most $m := c\log(s/\epsilon)$ and let $R_1 = \mathrm{Va}(\mathcal{T}_1)$ be the set of variables that appear in the terms in $\mathcal{T}_1$. Let $\mathcal{T}_2 = \{T \in \mathcal{T} : |\mathrm{Va}(T)\backslash R_1| \le m\}$ be the set of terms $T \in \mathcal{T}$ that contain at most m variables not in R_1. Let $R_2 = R_1 \cup \mathrm{Va}(\mathcal{T}_2)$ be the variables in R_1 and of the terms in $\mathcal{T}_2$. Let $\mathcal{T}_3 = \{T \in \mathcal{T} : |\mathrm{Va}(T)\backslash R_1| > m\}$ be the set terms $T \in \mathcal{T}$ that contain more than m variables that are not in R_1. Let $\mathcal{T}_4 = \{T \in \mathcal{T} : |\mathrm{Va}(T)\backslash R_2| \ge m\}$ be the set of terms in $T \in \mathcal{T}$ that contain at least m variables not in R_2. Let $R_3 = R_2 \cup \mathrm{Va}(\mathcal{T}_3\backslash\mathcal{T}_4)$ be the set of variables in R_2 and of the terms in $\mathcal{T}_3\backslash\mathcal{T}_4$. Then

$$|R_1| \le ms, |R_2| \le 2ms, |R_3| \le 3ms \text{ and } \mathcal{T}_4 \subseteq \mathcal{T}_3.$$

In steps 1-2, procedure **Approx**C uniformly at random partitions $[n]$ into $r = 8ms$ sets $X_1, \ldots, X_r$. Suppose that the variables in R_2 are distributed to the sets $X_{j_1}, \ldots, X_{j_q}$, $q \le |R_2| \le 2ms$.

For each $T \in \mathcal{T}_4$, the expected number of variables in $\mathrm{Va}(T)\backslash R_2$ that are not distributed to one of the sets $X_{j_1}, \ldots, X_{j_q}$ is greater or equal to $(1 - q/r)\, m = (3/4)m$. By Hoeffding's bound, Lemma 61, and the union bound, the probability that it is greater than $m/2$ in every term $T \in \mathcal{T}_4$ is at least $1 - s \cdot exp(-m/8) \ge 99/100$.

In steps 5-7 procedure **Approx**C are repeated M times and it finds relevant sets using two random uniform strings u and v. If $f(u_X \circ v_{\overline{X}}) \ne f(u)$ then a new relevant set is found. Consider any $T \in \mathcal{T}_3$. The size of T is at least m. Suppose $T = x_{a_1}^{\xi_1} \wedge \cdots \wedge x_{a_{m'}}^{\xi_{m'}}$, $m' \ge m$. For random uniform $u, v \in \{0,1\}^n$, the probability that there is no $j \in [m']$ such that $u_{a_j} = (u_X \circ v_{\overline{X}})_{a_j} = \xi_j$ is at most $(3/4)^m$. The probability that this happens for at least one $T \in \mathcal{T}_3$ and at least one of the M randomly uniformly chosen u and v in the procedure is at most $(3/4)^m sM \le 1/100$. Notice that if $u_{a_j} = (u_X \circ v_{\overline{X}})_{a_j} = \xi_j$ then $T(u) = T(u_X \circ v_{\overline{X}}) = 0$ and $T(w) = 0$ for every string w in the binary search that is made to find a new relevant set. Therefore, with probability at least 99/100, the procedure runs as if f contains no terms in $\mathcal{T}_3$. Let $f' = \vee_{T\in\mathcal{T}\backslash\mathcal{T}_3}T$. With probability at least 99/100 the procedure runs as if $f = f'$. The number of relevant variables in f' is at most $|R_2| \le 2ms$ and all those variables are distributed to $X_{j_1}, \ldots, X_{j_q}$. Therefore, with probability at least 99/100, the procedure generates at most $2ms < k$ relevant sets and therefore, by step 9, it does not reject and those relevant sets are from $X_{j_1}, \ldots, X_{j_q}$.

The output of the procedure is $X \subseteq X_{j_1} \cup \cdots \cup X_{j_q}$ and a random uniform w. We now show that with high probability $f(x_X \circ w_{\overline{X}})$ contains at most $k = 3ms$ relevant variables.

We have shown above that with probability at least $99/100$ every term $T \in \mathcal{T}_4$ contains at least $m/2$ variables that are not distributed to $X_{j_1}, \ldots, X_{j_q}$. Therefore, for a fixed term $T \in \mathcal{T}_4$ and for a random uniform w, the probability that $T(x_X \circ w_{\overline{X}}) = 0$ is at least $1 - (1/2)^{m/2}$. The probability that $T(x_X \circ w_{\overline{X}}) = 0$ for every $T \in \mathcal{T}_4$ is at least $1 - s(1/2)^{m/2} \geq 99/100$. Therefore, when we randomly uniformly choose $w \in \{0,1\}^n$, with probability at least $99/100$, the function $f(x_X \circ w_{\overline{X}})$ does not contain terms from $\mathcal{T}_4$. Thus, with probability at least $99/100$, $f(x_X \circ w_{\overline{X}})$ contains at most $|R_3| \leq 3ms$ variables.

Now as in the proof of Lemma 10, the query complexity is $2M + k \log r = \tilde{O}(s/\epsilon)$. □

We now prove Lemma 43.

Proof. As in the proof of Lemma 9, the probability that the algorithm fails to output X that satisfies $\mathbf{Pr}_{x,y \in U}[f(x_X \circ y_{\overline{X}}) = f(x)] \leq \epsilon/(100\lambda)$ is at most

$$k\left(1 - \frac{\epsilon}{100\lambda}\right)^{(100\lambda \ln(100/k))/\epsilon} = \frac{1}{100}.$$

If $\mathbf{Pr}_{x,y \in U}[f(x_X \circ y_{\overline{X}}) = f(x)] \leq \epsilon/(100\lambda)$ then, by Markov's inequality, Lemma 58, for a random uniform w, with probability at least $99/100$, $\mathbf{Pr}_{x \in U}[f(x_X \circ w_{\overline{X}}) = f(x)] \leq \epsilon/\lambda$.

Now as in the proof of Lemma 10, the query complexity is $2M + k \log r = \tilde{O}(s/\epsilon)$. □

In the next subsection, we give the result for s-term DNF, and then we show how to use the above technique to other classes.

6.2 Testing s-term DNF

We have shown in Lemma 42 and 43 that, using $\tilde{O}(s/\epsilon)$ queries, the problem of testing s-Term DNF can be reduced to the problem of testing s-Term DNF with $k = O(s \log(s/\epsilon))$ relevant variables. We then can use **Tester**C for the latter problem. This gives a tester for s-term DNF that makes $\tilde{O}(s^2/\epsilon)$. This is because the number of s-term DNF with $k = O(s \log(s/\epsilon))$ relevant variables is $2^{\tilde{O}(s^2)}$. We will now show how to slightly change the tester **Tester**C and get one that makes $\tilde{O}(s/\epsilon)$ queries.

In **Tester**C the procedures **ApproxTarget**, **TestSet** and **Close**fF make $O(k \log k)$, $O(k)$ and $O((k/\epsilon)\log(k/\epsilon))$ queries, respectively. This is $\tilde{O}(s/\epsilon)$ queries. Therefore, the only procedure that makes $\tilde{O}(s^2/\epsilon)$ queries in **Tester**C is **Close**FCU. So we will change this procedure.

Let h be an s-term DNF. Notice that in step 3 in **Close**FCU, for a random uniform $z = (z_1, \ldots, z_q) \in \{0,1\}^q$, the probability that z satisfies a term T in h of size at least $c \log(s/\epsilon)$ (i.e., $T(z) = 1$), is at most $(\epsilon/s)^c$. Therefore, if in **Close**FCU we define C^* to be the class of all s-term DNF with terms of size at most $c \log(s/\epsilon)$, then the probability that for at least one of the $\tau = (3/\epsilon)\log(|C^*|/\delta)$ random uniform $z = (z_1, \ldots, z_q)$ and at least one of the terms T in h of size at least $c \log(s/\epsilon)$, T satisfies z, is at most $s\tau(\epsilon/s)^c \leq 1/100$. Therefore, with probability at least $99/100$ the algorithm runs as if h is s-term DNF with terms of size at most $c \log(s/\epsilon)$ and then accept. Since $\log |C^*| = \tilde{O}(s)$ we get

Theorem 44. *For any $\epsilon > 0$, there is a two-sided adaptive algorithm for ϵ-testing s-Term DNF that makes $\tilde{O}(s/\epsilon)$ queries.*

For completeness we wrote the tester. See **TesterApprox**C in Figure 12. In the tester $C(\{y_1,\dots,y_q\}, c\log(s/\epsilon))$ is the class of all s-term DNFs with terms of size at most $c\log(s/\epsilon)$ over q variables.

7 Results

In this section, we extend the technique used in the previous section to other classes.

```
TesterApprox$C(f,\mathcal{D},\epsilon)$
Input: Oracle that access a Boolean function $f$ and $\mathcal{D}$.
Output: Either “reject” or “accept”

1.  $(X,w) \leftarrow$ApproxC$(f,\epsilon,1/6)$.
2.  $h := f(x_X \circ w_{\overline{X}})$.
3.  $(X,V,I) \leftarrow$ApproxTarget$(h,U,\epsilon,1/6)$.
4.  TestSets$(X,V,I)$.
5.  Close$fF(f,U,\epsilon,1/15)$
6.  $C^* \leftarrow C(\{y_1,\dots,y_q\}, c\log(s/\epsilon))$ where $q=|V|$
Test closeness to $C^*$
7.  Repeat $\tau = (3/\epsilon)\log(30|C^*|)$ times
8.        Choose $(z_1,\dots,z_q) \in U$.
9.        For every $g \in C^*$
10.             If $g(z) \neq F(z)$ then $C^* \leftarrow C^* \backslash \{g\}$.
11.             If $C^* = \emptyset$ then “reject”
12. Return “accept”
```

Figure 12: A procedure for testing s-Term DNF

7.1 Testing s-Term Monotone DNF

We first use the algorithm **LearnMonotone** in Figure 9 to show

Lemma 45. *Let $f:\{0,1\}^n \to \{0,1\}$ be an s-term Monotone DNF. For constant δ, algorithm* **LearnMonotone**$(f,U,\epsilon/2,\delta/2,s,2(\log(s/\epsilon)+\log(1/\delta)))$ *asks $O(s/\epsilon)$ ExQ_U and $O(s(\log n+\log s)\cdot\log(s/\epsilon))$ MQ and, with probability at least $1-\delta$, learns an s-term monotone DNF h that satisfies* $\mathbf{Pr}_U[h\neq f]\le\epsilon$.

Proof. Let $\hat{f}$ be the function f without the terms of size greater than $2(\log(s/\epsilon)+\log(1/\delta))$. Then

$$\mathbf{Pr}_U[f\neq\hat{f}] \le s2^{-2(\log(s/\epsilon)+\log(1/\delta)))} \le \frac{\epsilon}{2}.$$

In the algorithm **LearnMonotone** the probability that one of the assignments in step 3 (that is, a where $a\in U$) satisfies one of the terms in f of size greater than $2(\log(s/\epsilon)+\log(1/\delta))$ is less than

$$(4s/\epsilon)(\log(1/\delta))s2^{-2(\log(s/\epsilon)+\log(1/\delta))} \le \frac{\delta}{2}.$$

Also for a monotone term T, if $T(a)=0$ then for any y, $T(a*y)=0$. Therefore, with probability at least $1-\delta/2$, the algorithm runs as if f is $\hat{f}$ (which is s-term monotone $(2(\log(s/\epsilon)+\log(1/\delta)))$-DNF). By Lemma 34, if the target is $\hat{f}$ then, with probability at least $1-\delta/2$, **LearnMonotone** outputs h that is $(\epsilon/2)$-close to $\hat{f}$. Since $\hat{f}$ is $(\epsilon/2)$-close to f and h is $(\epsilon/2)$-close to $\hat{f}$ we have that h is ϵ-close to f. This happens with probability at least $1-\delta$.

The number of queries follows from Lemma 34. □

We now prove

Theorem 46. *For any $\epsilon>0$, there is a polynomial time two-sided adaptive algorithm for ϵ-testing s-Term Monotone DNF that makes $\tilde{O}(s/\epsilon)$ queries.*

Proof. We first run **Approx**C and get an s-term monotone DNF h with $O(s\log(s/\epsilon))$ variables that is $(\epsilon/6)$-close to f. We then use Theorem 29 with Lemma 45. □

We also have

Theorem 47. *For any $\epsilon>0$, there is a polynomial time two-sided adaptive algorithm for ϵ-testing s-Term Unate DNF that makes $\tilde{O}(s/\epsilon)$ queries.*

Proof. The proof is similar to the proof of Theorem 36. □

7.2 Testing Size-s Boolean Formula and Size-s Boolean Circuit

A Boolean formula is a rooted tree in which each internal node has arbitrarily many children and is labeled with AND or OR. Each leaf is labeled with a Boolean variable x_i or its negation $\bar{x}_i$. The size of a Boolean formula is the number of AND/OR gates it contains. The class size-s Boolean Formula is the class of all Boolean formulas of size at most s.

A Boolean circuit is a rooted directed acyclic graph with internal nodes labeled with an AND, OR or NOT gate. Each AND/OR gate is allowed to have arbitrarily many descendants. Each directed path from the root ends in one of the nodes $x_1,x_2,\ldots,x_n,0,1$.

The same analysis that we did for s-term DNF also applies to size-s Boolean formulas and size-s Boolean circuit. Analogous to the size of terms, we take the number of distinct literals a gate has. If the gate is labeled with AND (respectively, OR) and the number of distinct literals it has is more than $c\log(s/\epsilon)$, then we replace the gate with a node labeled with 0 (respectively, 1) and remove all the edges to its children.

Therefore we have

Lemma 48. *Lemma 42 and 43 are also true for size-s Boolean formulas and size-s Boolean circuit.*

We now prove

Theorem 49. *For any $\epsilon>0$, there is a two-sided adaptive algorithm for ϵ-testing size-s Boolean Formula that makes $\tilde{O}(s/\epsilon)$ queries.*

Proof. Similar to testing s-term DNF, we can ignore gates that have more than $c\log(s/\epsilon)$ distinct literals. Just replace it with 0 if its label is AND and with 1 if it is OR.

The number of Boolean formulas of size s that have at most $c\log(s/\epsilon)$ distinct literal in each gate and at most $k=O(s\log(s/\epsilon))$ variables is $2^{\tilde{O}(s)}$, [30]. The rest of the proof goes along with the proof of testing s-term DNF. □

The number of Boolean circuits of size s that have at most $c\log(s/\epsilon)$ distinct literals in each gate and at most $k = O(s\log(s/\epsilon))$ variables is $2^{\tilde{O}(s^2)}$, [30]. Then similar to the above proof one can show

Theorem 50. *For any $\epsilon > 0$, there is a two-sided adaptive algorithm for ϵ-testing size-s Boolean Circuit that makes $\tilde{O}(s^2/\epsilon)$ queries.*

7.3 Testing s-Sparse Polynomial

In the literature, the first testing algorithm for the class s-Sparse Polynomial runs in exponential time [30] and makes $\tilde{O}(s^4/\epsilon^2)$ queries. Chakraborty et al., [21], then gave another exponential time algorithm that makes $\tilde{O}(s/\epsilon^2)$ queries. Diakonikolas et al. gave in [31] the first polynomial time testing algorithm that makes $poly(s, 1/\epsilon)$ queries. Here we prove

Theorem 51. *For any $\epsilon > 0$, there is a two-sided adaptive algorithm for ϵ-testing s-Sparse Polynomial that makes $\tilde{O}(s^2/\epsilon)$ queries.*

```
**LearnPolyUnif**$(f, \epsilon, \delta, s)$
*Input*: Oracle that accesses a Boolean function $f$ that is $s$-sparse polynomial.
*Output*: $h$ that is $s$-sparse polynomial

1.  $h \leftarrow 0$, $t(h) \leftarrow 0$, $w \leftarrow 0$.
2.  Repeat $(s/\epsilon)\ln(3s/\delta)\log(3s/\delta)$ times.
3.      Choose $a \in U$.
4.      $t(h) \leftarrow t(h) + 1$.
5.      If $(f + h)(a) = 1$ then
6.          $m \leftarrow 0$, $w \leftarrow w + 1$
7.          If $w = \log(3s/\delta)$ then Output $h$
8.          While $m \le \alpha := 16 \cdot (8s/\epsilon)(2\ln(s/\delta) + \ln n)$ and $wt(a) > \log(s/\epsilon) + 3$ do
9.              $m \leftarrow m + 1$;
10.             Choose $y \in U$
11.             If $(f + h)(a * y) = 1$ then $a \leftarrow a * y$
12.         If $wt(a) > \log(s/\epsilon) + 3$ then Goto 16
13.         $w \leftarrow 0$.
14.         $a \leftarrow$ Find a monotone term in $(f + h)(a * x)$
15.         $h \leftarrow h + \prod_{a_i=1} x_i$
16.         $t(h) \leftarrow 0$.
17.     If $t(h) = (1/\epsilon)\ln(3s/\delta)$ then Output $h$
```

Figure 13: A learning algorithm for s-sparse polynomial under the uniform distribution

We have shown in Lemma 42 and 43 that the problem of testing s-Term DNF can be reduced to the problem of testing s-Term DNF with $k = O(s\log(s/\epsilon))$ relevant variables. The same reduction and analysis show that the problem of testing s-sparse polynomials can be reduced to the problem of testing s-sparse polynomials with $k = O(s\log(s/\epsilon))$ relevant variables. The reduction makes $\tilde{O}(s/\epsilon)$ queries. Thus

Lemma 52. *Lemma 42 and 43 are also true for s-Sparse Polynomials.*

We can then use **Tester**C for the latter problem. Therefore, all we need to do in this section is to find a learning algorithm for the class of s-sparse polynomials with $k = O(s \log(s/\epsilon))$ relevant variables.

Consider the algorithm **LearnPolyUnif** in Figure 13. We prove

Lemma 53. *Let f be a s-sparse polynomial with n variables. For constant δ, algorithm* **LearnPolyUnif** *asks $\tilde{O}(s/\epsilon)$ ExQ_U and $\tilde{O}((s^2/\epsilon)\log n)$ MQ and, with probability at least $1-\delta$, learns an s-sparse polynomial h that satisfies $\mathbf{Pr}_U[h \neq f] \leq \epsilon$.*

Proof. Let $f = M_1 + M_2 + \cdots + M_{s'}$, $s' \leq s$, where $deg(M_1) \leq \cdots \leq \deg(M_{s''}) \leq \log(s/\epsilon) + 3 < deg(M_{s''+1}) \leq \cdots \leq deg(M_{s'})$. Let $f_1 = M_1 + \cdots + M_{s''}$ and $f_2 = M_{s''+1} + \cdots + M_{s'}$. Then $f = f_1 + f_2$ and $\mathbf{Pr}_U[f_2 = 1] \leq s2^{-\log(s/\epsilon)-3} = \epsilon/8$. If $f(a) \neq f_1(a)$ then $f_2(a) = 1$ and therefore $\mathbf{Pr}_U[f \neq f_1] \leq \mathbf{Pr}_U[f_2 = 1] \leq \epsilon/8$ and $\mathbf{Pr}_U[f = f_1] \geq 1 - \epsilon/8$. In fact, if $A(x)$ is the event that $T_i(x) = 0$ for all $i > s''$ then $\mathbf{Pr}_U[A] \geq 1 - \epsilon/8$.

The algorithm **LearnPolyUnif** is similar to the algorithm **LearnPolynomial** with $d = \log(s/\epsilon) + 3$ with the changes that is described below.

First notice that in the algorithm the hypothesis h contains only terms from f_1, that is, terms of size at most $\log(s/\epsilon) + 3$. This is because in step 12 the algorithm skips the command that adds a term to h when $wt(a) \geq \log(s/\epsilon) + 3$. Suppose at some stage of the algorithm, $h = \sum_{i \in B \subseteq [s'']} M_i$ contains some terms of f_1 and let $g = f + h = \sum_{i \in [s'] \backslash B} M_i$. Suppose $\mathbf{Pr}_U[f \neq h] \geq \epsilon$. Then $\mathbf{Pr}_U[g = 1] \geq \epsilon$. We want to compute the probability that the algorithm finds a term in $f_1 + h = \sum_{i \in [s''] \backslash B} M_i$ and not in f_2. That is, the probability that it finds a term of f of degree at most $\log(s/\epsilon) + 3$. We first have

$$\begin{aligned}
\mathbf{Pr}_{a \in U}[(f_1 + h)(a) = 1 \wedge A | (f + h)(a) = 1] &= \mathbf{Pr}_{a \in U}[g(a) = 1 \wedge A | g(a) = 1] \\
&= \frac{\mathbf{Pr}_{a \in U}[g(a) = 1 \wedge A]}{\mathbf{Pr}_{a \in U}[g(a) = 1]} \\
&\geq \frac{\mathbf{Pr}_{a \in U}[g(a) = 1] - \mathbf{Pr}_{a \in U}[\overline{A}]}{\mathbf{Pr}_{a \in U}[g(a) = 1]} \\
&\geq 1 - \frac{\epsilon/8}{\epsilon} \geq \frac{7}{8}.
\end{aligned}$$

That is, if $\mathbf{Pr}_U[f \neq h] \geq \epsilon$ and $(f + h)(a) = 1$ then with probability at least 7/8, $(f_1 + h)(a) = 1$ and for every term T in f_2, $T(a) = 0$. If the event $A(a)$ happens then for any y and for every term T in f_2, $T(a * y) = 0$, and therefore, for such a, the algorithm runs as if the target is f_1.

When the algorithm reaches step 5 and finds a string $a \in \{0,1\}^n$ such that $(f + h)(a) = 1$, we have three cases:

1. The event $B \equiv [\ ((f_1 + h)(a) = 1$ and $A(a)]$ happens.

2. $\mathbf{Pr}_U[f \neq h] \geq \epsilon$ and $\overline{B}$.

3. $\mathbf{Pr}_U[f \neq h] < \epsilon$ and $\overline{B}$.

Case 1. Notice that steps 8-11 are identical to steps 7-10 in **LearnPolynomial** in Figure 10 with $d = \log(s/\epsilon) + 3$. Therefore, as in the proof of Lemma 41, with probability at least $1 - \delta/3$ every assignment a that satisfies B gives a term in f_1 that is not in h.

Case 2. This case can happen with probability at most 1/8. So the probability that it happens $\log(3s/\delta)$ consecutive times is at most $\delta/(3s)$. The probability that it does happen for some of the at most s different hypothesis h generated in the algorithm is at most $\delta/3$. Notice that w counts the number of consecutive times that this case happens and step 7 outputs h when it does happen $\log(3s/\delta)$ consecutive times. Therefore, with probability at most $\delta/3$ the algorithm halts in step 7 and output h that satisfies $\mathbf{Pr}[f \neq h] \geq \epsilon$. This is also the reason that the algorithm repeats the search for a in step 2, $(s/\epsilon)\ln(3s/\delta)\log(3s/\delta)$ times which is $\log(3s/\delta)$ times more than in algorithm **LearnPolynomial**.

When this case happens, the algorithm either ends up with a string a of weight that is greater than $\ell := \log(s/\epsilon) + 3$ and then it ignores this string, or, it ends up with a string of weight less than or equal to ℓ and then, Step 14 finds a new term in f_1. This is because that a string of weight less than or equal to ℓ cannot satisfy a term of degree more than ℓ.

Case 3. This case cannot happen more than $\log(3s/\delta)$ consecutive times because if it does step 7 outputs h which is a good hypothesis. □

8 A General Method for Other Testers

In this section, we generalize the method we have used in the previous section and then prove some more results

```
Algorithm ApproxGeneral$C(f, \epsilon, \delta)$
Input: Oracle that accesses a Boolean function $f$
Output: Either "$X \subseteq [n], w \in \{0,1\}^n$" or "reject"

Partition $[n]$ into $r$ sets
Set $r = k^{c+1}$.
Choose uniformly at random a partition $X_1, X_2, \ldots, X_r$ of $[n]$

Find a close function and relevant sets
Set $X = \emptyset$; $I = \emptyset$; $t(X) = 0$
Repeat $M = (4c_1k/(\delta\epsilon))\ln(4k/\delta)$ times
    Choose $u, v \in U$.
    $t(X) \leftarrow t(X) + 1$
    If $f(u_X \circ v_{\overline{X}}) \neq f(u)$ then
        Find a new relevant set $X_\ell$; $X \leftarrow X \cup X_\ell$; $I \leftarrow I \cup \{\ell\}$.
        If $|I| > k$ then Output "reject"
        $t(X) = 0$.
    If $t(X) = (4c_1/(\delta\epsilon))\ln(4k/\delta)$ then
        Choose a random uniform $w$;
        Output$(X, w)$.
```

Figure 14: An algorithm that removes variables from f that have a small influence.

We define the distribution $\mathcal{D}[p]$ to be over $\prod_{i=1}^{n}\{0, 1, x_i\}$ where each coordinate i is chosen to be x_i with probability p, 0 with probability $(1-p)/2$ and 1 with probability $(1-p)/2$. We will denote by $|f|$ the *size* of f in C which is the length of the representation of the function f in C.

Consider the algorithm **ApproxGeneral**C in Figure 14. We start with the following result

Lemma 54. *Let $\delta < 1/2$, $c_1 \geq 1, \lambda > 1$ and $c \geq 1$ be any constants, $k := k(\epsilon, \delta, |f|)$ be an integer and $M = (4c_1k/(\delta\epsilon))\ln(4k/\delta)$. Let C be a class of functions where for every $f \in C$ there is $h \in (C \cap k$-Junta) with relevant variables $x(Y) = \{x_i | x \in Y\}$, $Y \subseteq [n]$, and $h' \in (C \cap (\lambda k)$-Junta) that satisfy the following:*

1. $\mathbf{Pr}_{z \in \mathcal{D}[1/2]}[f(z) \neq h(z)] \leq \delta/(4M)$.

2. $\mathbf{Pr}_{y \in \mathcal{D}[1/k^c]}[f(x_Y \circ y_{\overline{Y}}) \neq h'(y)] \leq \delta/4$.

The algorithm **ApproxGeneral**C *makes $\tilde{O}(k/\epsilon)$ queries and,*

1. *If $f \in C$ then, with probability at least $1-\delta$, the algorithm does not reject and outputs X and w such that $f(x_X \circ w_{\overline{X}}) \in C$ has at most λk relevant variables.*

2. *For any f, if the algorithm does not reject then, with probability at least $1-\delta$, $\mathbf{Pr}[f(x) \neq f(x_X \circ w_{\overline{X}})] \leq \epsilon/c_1$.*

Proof. Let $f \in C$. Let $h \in C \cap k-$Junta and $h' \in (C \cap (\lambda k)$-Junta) be functions that satisfies 1 and 2. The algorithm in step 5 chooses two random uniform strings u and v. Define $z = (z_1, \ldots, z_n)$ such that $z_i = 0$ if $u_i = v_i = 0$, $z_i = 1$ if $u_i = v_i = 1$, and $z_i = x_i$ if $u_i \neq v_i$. Since u and v are chosen uniformly at random we have that $z \in \mathcal{D}[1/2]$ and therefore, with probability at least $1 - \delta/(4M)$, $f(z) = h(z)$. If $f(z) = h(z)$ then $f(u) = h(u)$ and $f(u_X \circ v_{\overline{X}}) = h(u_X \circ v_{\overline{X}})$ for any $X \subseteq [n]$. In Step 8 **ApproxGeneral** does a binary search to find a new relevant set. In the binary search it queries strings a that satisfy $a_i = z_i = u_i = v_i$ for all i that satisfies $z_i \in \{0,1\}$. Therefore, $f(a) = h(a)$ for all the strings a generated in the binary search for finding a relevant set. Therefore, with probability at least $1-\delta/(4M)$, $f(a) = h(a)$ for all the queries used in one iteration and, with probability at least $1-\delta/4$, $f(a) = h(a)$ for all the queries used in the algorithm. That is, with probability at least $1 - \delta/4$, the algorithm runs as if the target is h.

Therefore, if $f \in C$, then with probability at least $1-\delta/4$, each one of the relevant sets discovered in the algorithm contains at least one relevant variable of h. Then since $h \in k$-Junta, the algorithm does not reject in Step 9, that is, $|I| \leq k$.

Now we show that, if $f \in C$ then with probability at least $1-\delta/4$, $f(x_X \circ w_{\overline{X}})$ contains at most λk relevant variables. Consider the partition in steps 1-2 in the algorithm and let $X_{i_1}, \ldots, X_{i_{k'}}$, $k' \leq k$, be the sets where the indices of the relevant variables $x(Y)$ of h are distributed. Let $X' = X_{i_1} \cup \cdots \cup X_{i_{k'}}$. Notice that for a random uniform $w \in \{0,1\}^n$, $(x_{X'} \circ w_{\overline{X'}}) = (x_Y \circ y_{\overline{Y}})$ where $y \in \mathcal{D}[k'/k^{c+1}]$. That is, given that the relevant variables of h are distributed to k' different sets that their union is X', the probability distributions of $(x_{X'} \circ w_{\overline{X'}})$ and of $(x_Y \circ y_{\overline{Y}})$ are identical. Choosing a string in $\mathcal{D}[k'/k^{c+1}]$ can be done by first choosing a string b in $\mathcal{D}[1/k^c]$ and then substitute in each variable x_i, $i \notin Y$, in b, 0, 1 or x_i with probability $1/2 - k'/(2k)$, $1/2 - k'/(2k)$ or k'/k, respectively. Therefore, since $\mathbf{Pr}_{y \in \mathcal{D}[1/k^c]}[f(x_Y \circ y_{\overline{Y}}) \neq h'(y)] \leq \delta/4$, we have $\mathbf{Pr}_{y \in \mathcal{D}[k'/k^{c+1}]}[f(x_Y \circ y_{\overline{Y}}) \neq h'(y)] \leq \delta/4$ and, thus, with probability at least $1-\delta/4$, we have that $f(x_{X'} \circ w_{\overline{X'}}) = h'(x)$. In particular, with probability at least $1-\delta/4$, $f(x_{X'} \circ w_{\overline{X'}})$ has at most λk relevant variables. Since $X \subseteq X'$ we also have with the same probability $f(x_X \circ w_{\overline{X}})$ has at most λk relevant variables. This completes the proof of *1.*

Now let f be any boolean function. If the algorithm does not reject then $|I| \le k$. Since for the final X, $f(u_X \circ v_{\overline{X}}) \neq f(u)$ for $(4c_1/(\delta\epsilon))\ln(4k/\delta)$ random uniform u and v, we have that the probability that the algorithm fails to output X that satisfies $\mathbf{Pr}_{x,y\in U}[f(x_X \circ y_{\overline{X}}) \neq f(x)] \le \delta\epsilon/(4c_1)$ is at most

$$k\left(1-\frac{\delta\epsilon}{4c_1}\right)^{(4c_1/(\delta\epsilon))\ln(4k/\delta)} = \frac{\delta}{4}.$$

If $\mathbf{Pr}_{x,y\in U}[f(x_X \circ y_{\overline{X}}) \neq f(x)] \le \delta\epsilon/(4c_1)$ then, by Markov's inequality, for a random uniform w, with probability at least $1-\delta/4$, $\mathbf{Pr}_{x\in U}[f(x_X \circ w_{\overline{X}}) \neq f(x)] \le \epsilon/c_1$. This completes the proof of *2.* □

In the next subsections, we give some results that follow from Lemma 54.

8.1 Testing Decision List

In [30], Diakonikolas et al. gave a polynomial time tester for Decision List that makes $\tilde{O}(1/\epsilon^2)$ queries. In this paper, we give a polynomial time tester that makes $\tilde{O}(1/\epsilon)$ queries.

We show

Theorem 55. *For any $\epsilon > 0$, there is a two-sided adaptive algorithm for ϵ-testing Decision List that makes $\tilde{O}(1/\epsilon)$ queries.*

Proof. Let $f = (x_{i_1}, \xi_1, a_1), \ldots, (x_{i_s}, \xi_s, a_s)$ be any decision list. We first use Lemma 54.

Define $k = \min(s, c' \log(1/(\epsilon\delta)))$ for some large constant c' and $h = (x_{i_1}, \xi_1, a_1), \ldots, (x_{i_k}, \xi_k, a_k)$. For the distribution $\mathcal{D}[1/2]$, the probability that $f(z) = h(z)$ is 1 when $k = s$ and at least $1 - (3/4)^k \ge 1 - 1/(3M)$ where $M = (4c_1k/(\delta\epsilon))\ln(4k/\delta)$. For the distribution $\mathcal{D}[1/k^2]$ and $Y = \{i_1, \ldots, i_k\}$, the probability that $f(x_Y \circ y_{\overline{Y}})$ has more than $2k$ relevant variables is less than $(3/4)^k \le \delta/3$.

The query complexity of **ApproxGeneral** is $\tilde{O}(k/\epsilon) = \tilde{O}(1/\epsilon)$. Therefore all we need to do to get the result is to give a tester for decision list of size $k = O(\log(1/\epsilon))$ that makes $\tilde{O}(1/\epsilon)$ queries. The learnability of this class with $\tilde{O}(1/\epsilon)$ ExQs follows from [52, 14]. □

8.2 Testing r-DNF and r-Decision List for Constant r

An r-decision list is a sequence $f = (T_1, \xi_1, a_1), \ldots, (T_s, \xi_s, a_s)$ for any s where $\xi_i, a_i \in \{0,1\}$ and T_i are r-terms. This sequence represents the following function: $f(x) :=$"If $T_1 = \xi_1$ then output(a_1) else if $T_2 = \xi_2$ then output(a_2) else if $\cdots$ else if $T_s = \xi_s$ then output(a_s)". The class r-Decision List is the class of all r-decision lists and the class of Length-s r-Decision List is the class of all r-decision lists $f = (T_1, \xi_1, a_1), \ldots, (T_m, \xi_m, a_m)$ with $m \le s$.

In this subsection, we show

Theorem 56. *Let r be any constant. For any $\epsilon > 0$, there is a two-sided adaptive algorithm for ϵ-testing r-Decision List and r-DNF that makes $\tilde{O}(1/\epsilon)$ queries.*

It is known that the class Length-s r-Decision List is learnable under any distribution in time $O(n^r)$ using $O((sr\log n + \log(1/\delta))/\epsilon)$ ExQ$_D$, [14, 52]. Also we may assume that $T_1, \ldots, T_s$ are distinct and therefore s is less than $\sum_{i=1}^r \binom{n}{i} 2^i$, the number of terms of size at most r. Thus, for constant r, it is enough to prove that r-Decision List and r-DNF satisfies 1 and 2 in Lemma 54 with $k = poly(\log(1/\epsilon))$.

We now prove the result for r-Decision List when r is constant. The same analysis shows that the result is also true for r-DNF.

Consider an r-decision list $f=(T_1,\xi_1,a_1)\cdots(T_s,\xi_s,a_s)$. If $\xi_i=0$ then we change $(T_i,0,a_i)$ to the equivalent expression $(x_{i_1}^{1-c_{i_1}},1,a_i)\cdots(x_{i_\ell}^{1-c_{i_\ell}},1,a_i)$ where $T_i=x_{i_1}^{c_1}\cdots x_{i_\ell}^{c_\ell}$. Therefore we may assume that $\xi_j=1$ for all j. In that case we just write $f=(T_1,a_1)\cdots(T_s,a_s)$.

For an r-decision list $f=(T_1,a_1)\cdots(T_s,a_s)$, a *sublist of* f is an r-decision list $g=(T_{i_1},a_{i_1})\cdots(T_{i_\ell},a_{i_\ell})$ such that $1\le i_1<i_2<\cdots<i_\ell\le s$.

We first prove

Lemma 57. *Let r be a constant. For any r-decision list f there is a $k_r:=O(\log^r(1/\epsilon))$-length r-decision list h that is a sublist of f and satisfies 1 and 2 in Lemma 54.*

Proof. We show that it satisfies 1 in Lemma 54. The proof that it also satisfies 2 is similar.

We give a stronger result as long as r is constant. We prove by induction that for any r-decision list f there is a $k_r=O(\log^r(1/\epsilon))$-length r-decision list h that is a sublist of f and satisfies $\mathbf{Pr}_{z\in\mathcal{D}[1/2]}[f(z)\ne h(z)]\le poly(\epsilon)$.

The proof is by induction on r. For $r=1$, the result follows from the proof of Theorem 55 in the previous subsection. Assume the result is true for r-decision list. We now show the result for $(r+1)$-decision list.

Let c be a large constant. Let $f=(T_1,a_1)\cdots(T_s,a_s)$ be $(r+1)$-decision list. Let $s_1=1$ and $T_{s_1},\ldots,T_{s_w}$ be a sequence of terms such that $s_1<s_2<\cdots<s_w\le s$ and for every i, s_i is the minimal integer that is greater than s_{i-1} such that the variables in T_{s_i} do not appear in any one of the terms $T_{s_1},T_{s_2},\ldots,T_{s_{i-1}}$. Define $h_0=(T_1,a_1)(T_2,a_2)\cdots(T_{s_{w'}},a_{s_{w'}})$ if $w':=c2^{r+1}\ln(1/\epsilon)\le w$ and $h_0=f$ if $w'>w$. Then

$$\begin{aligned}\mathbf{Pr}_{z\in\mathcal{D}[1/2]}[f(z)\ne h_0(z)] &\le \mathbf{Pr}[T_1(z)=0\wedge T_2(z)=0\wedge\cdots\wedge T_{s_{w'}}(z)=0]\\ &\le \mathbf{Pr}[T_{s_1}(z)=0\wedge T_{s_2}(z)=0\wedge\cdots\wedge T_{s_{w'}}(z)=0]\\ &\le \left(1-\frac{1}{2^{r+1}}\right)^{w'}\le poly(\epsilon).\end{aligned}$$

Let $S=\{x_{j_1},\ldots,x_{j_t}\}$ be the set of the variables in $T_{s_1},\ldots,T_{s_{w'}}$. Then $t\le(r+1)w'=c(r+1)2^{r+1}\ln(1/\epsilon)$ and every term T_i in h_0 contains at least one variable in S. Consider all the terms that contains the variable x_{j_1}, $T_{i_1}=x_{j_1}T'_{i_1},T_{i_2}=x_{j_1}T'_{i_2},\ldots,T_{i_\ell}=x_{j_1}T'_{i_\ell}$, $i_1<i_2<\cdots<i_\ell$. Consider the r-decision list $g:=(T'_{i_1},a_1)(T'_{i_2},a_2)\cdots(T'_{i_\ell},a_\ell)$. By the induction hypothesis there is an r-decision list g' that is a sublist of g of length at most $k_r=O(\log^r(1/\epsilon))$ such that $\mathbf{Pr}_{z\in\mathcal{D}[1/2]}[g(z)\ne g'(z)]\le poly(\epsilon)$. Let h_1 be h_0 without all the terms (T_{i_w},a_{i_w}) that correspond to the terms (T'_{i_w},a_{i_w}) that do not occur in g'. It is easy to see that $\mathbf{Pr}_{z\in\mathcal{D}[1/2]}[h_0(z)\ne h_1(z)]\le poly(\epsilon)$. We do the same for all the other variables $x_{j_2},\ldots,x_{j_t}$ of S and get a sequence of r-decision lists $h_2,h_3,\ldots,h_t$ that satisfies $\mathbf{Pr}_{z\in\mathcal{D}[1/2]}[h_w(z)\ne h_{w+1}(z)]\le poly(\epsilon)$. Therefore $\mathbf{Pr}_{z\in\mathcal{D}[1/2]}[f(z)\ne h_t(z)]\le t\cdot poly(\epsilon)=poly(\epsilon)$ and the length of h_t is at most $tk_r=O(\log^{r+1}(1/\epsilon))=k_{r+1}$. □

Acknowledgements

The author would like to express heartfelt gratitude to Oded Goldreich for his invaluable contributions to this paper. Goldreich's insightful discussions, thoughtful comments, and generosity

in allowing the inclusion of his overview of the first tester (Section 2) have greatly improved the exposition and presentation of the technique employed in this paper.

References

[1] Noga Alon, Tali Kaufman, Michael Krivelevich, Simon Litsyn, and Dana Ron. Testing low-degree polynomials over GF(2). In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2003 and 7th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2003, Princeton, NJ, USA, August 24-26, 2003, Proceedings*, pages 188–199, 2003. `doi:10.1007/978-3-540-45198-3\_17`.

[2] Noga Alon, Tali Kaufman, Michael Krivelevich, Simon Litsyn, and Dana Ron. Testing reed-muller codes. *IEEE Trans. Information Theory*, 51(11):4032–4039, 2005. `doi:10.1109/TIT.2005.856958`.

[3] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

[4] Roksana Baleshzar, Meiram Murzabulatov, Ramesh Krishnan S. Pallavoor, and Sofya Raskhodnikova. Testing unateness of real-valued functions. *CoRR*, abs/1608.07652, 2016. URL: `http://arxiv.org/abs/1608.07652`, `arXiv:1608.07652`.

[5] Aleksandrs Belovs and Eric Blais. A polynomial lower bound for testing monotonicity. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 1021–1032, 2016. `doi:10.1145/2897518.2897567`.

[6] Arnab Bhattacharyya, Swastik Kopparty, Grant Schoenebeck, Madhu Sudan, and David Zuckerman. Optimal testing of reed-muller codes. In *Property Testing - Current Research and Surveys*, pages 269–275. 2010. `doi:10.1007/978-3-642-16367-8\_19`.

[7] Eric Blais. Improved bounds for testing juntas. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25-27, 2008. Proceedings*, pages 317–330, 2008. `doi:10.1007/978-3-540-85363-3\_26`.

[8] Eric Blais. Testing juntas nearly optimally. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 151–158, 2009. `doi:10.1145/1536414.1536437`.

[9] Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, USA, June 8-10, 2011*, pages 210–220, 2011. `doi:10.1109/CCC.2011.31`.

[10] Eric Blais and Daniel M. Kane. Tight bounds for testing k-linearity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 435–446, 2012. doi:10.1007/978-3-642-32512-0_37.

[11] Avrim Blum. Learning a function of r relevant variables. In *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, pages 731–733, 2003. doi:10.1007/978-3-540-45167-9_54.

[12] Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artif. Intell.*, 97(1-2):245–271, 1997. doi:10.1016/S0004-3702(97)00063-5.

[13] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993. doi:10.1016/0022-0000(93)90044-W.

[14] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Inf. Process. Lett.*, 24(6):377–380, 1987. doi:10.1016/0020-0190(87)90114-1.

[15] Nader H. Bshouty. Almost optimal distribution-free junta testing. *CoRR*, abs/1901.00717, 2018.

[16] Nader H. Bshouty. Exact learning from an honest teacher that answers membership queries. *Theor. Comput. Sci.*, 733:4–43, 2018. doi:10.1016/j.tcs.2018.04.034.

[17] Nader H. Bshouty and Areej Costa. Exact learning of juntas from membership queries. *Theor. Comput. Sci.*, 742:82–97, 2018. doi:10.1016/j.tcs.2017.12.032.

[18] Nader H. Bshouty and Yishay Mansour. Simple learning algorithms for decision trees and multivariate polynomials. *SIAM J. Comput.*, 31(6):1909–1925, 2002. doi:10.1137/S009753979732058X.

[19] Deeparnab Chakrabarty and C. Seshadhri. A $o(n)$ monotonicity tester for boolean functions over the hypercube. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 411–418, 2013. doi:10.1145/2488608.2488660.

[20] Deeparnab Chakrabarty and C. Seshadhri. A $\tilde{O}(n)$ non-adaptive tester for unateness. *CoRR*, abs/1608.06980, 2016. URL: http://arxiv.org/abs/1608.06980, arXiv:1608.06980.

[21] Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Efficient sample extractors for juntas with applications. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pages 545–556, 2011. doi:10.1007/978-3-642-22006-7_46.

[22] Xi Chen, Anindya De, Rocco A. Servedio, and Li-Yang Tan. Boolean function monotonicity testing requires (almost) $n^{1/2}$ non-adaptive queries. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 519–528, 2015. doi:10.1145/2746539.2746570.

[23] Xi Chen, Rocco A. Servedio, and Li-Yang Tan. New algorithms and lower bounds for monotonicity testing. *CoRR*, abs/1412.5655, 2014. URL: `http://arxiv.org/abs/1412.5655`, `arXiv:1412.5655`.

[24] Xi Chen, Rocco A. Servedio, Li-Yang Tan, Erik Waingarten, and Jinyu Xie. Settling the query complexity of non-adaptive junta testing. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, pages 26:1–26:19, 2017. `doi:10.4230/LIPIcs.CCC.2017.26`.

[25] Xi Chen, Erik Waingarten, and Jinyu Xie. Beyond talagrand functions: new lower bounds for testing monotonicity and unateness. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 523–536, 2017. `doi:10.1145/3055399.3055461`.

[26] Xi Chen, Erik Waingarten, and Jinyu Xie. Boolean unateness testing with $\widetilde{O}(n^{3/4})$ adaptive queries. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 868–879, 2017. `doi:10.1109/FOCS.2017.85`.

[27] Xi Chen and Jinyu Xie. Tight bounds for the distribution-free testing of monotone conjunctions. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 54–71, 2016. `doi:10.1137/1.9781611974331.ch5`.

[28] Hana Chockler and Dan Gutfreund. A lower bound for testing juntas. *Inf. Process. Lett.*, 90(6):301–305, 2004. `doi:10.1016/j.ipl.2004.01.023`.

[29] Peter Damaschke. Adaptive versus nonadaptive attribute-efficient learning. *Machine Learning*, 41(2):197–215, 2000. `doi:10.1023/A:1007616604496`.

[30] Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Krzysztof Onak, Ronitt Rubinfeld, Rocco A. Servedio, and Andrew Wan. Testing for concise representations. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 549–558, 2007. `doi:10.1109/FOCS.2007.32`.

[31] Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Rocco A. Servedio, and Andrew Wan. Efficiently testing sparse $GF(2)$ polynomials. *Algorithmica*, 61(3):580–605, 2011. `doi:10.1007/s00453-010-9426-9`.

[32] Elya Dolev and Dana Ron. Distribution-free testing for monomials with a sublinear number of queries. *Theory of Computing*, 7(1):155–176, 2011. `doi:10.4086/toc.2011.v007a011`.

[33] Eldar Fischer, Guy Kindler, Dana Ron, Shmuel Safra, and Alex Samorodnitsky. Testing juntas. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 103–112, 2002. `doi:10.1109/SFCS.2002.1181887`.

[34] Dana Glasner and Rocco A. Servedio. Distribution-free testing lower bound for basic boolean functions. *Theory of Computing*, 5(1):191–216, 2009. `doi:10.4086/toc.2009.v005a010`.

[35] Oded Goldreich. On testing computability by small width obdds. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, pages 574–587, 2010. `doi:10.1007/978-3-642-15369-3\_43`.

[36] Oded Goldreich, editor. *Property Testing - Current Research and Surveys*, volume 6390 of *Lecture Notes in Computer Science*. Springer, 2010. `doi:10.1007/978-3-642-16367-8`.

[37] Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. URL: `http://www.cambridge.org/us/catalogue/catalogue.asp?isbn=9781107194052`, `doi:10.1017/9781108135252`.

[38] Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000. `doi:10.1007/s004930070011`.

[39] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. `doi:10.1145/285055.285060`.

[40] Parikshit Gopalan, Ryan O'Donnell, Rocco A. Servedio, Amir Shpilka, and Karl Wimmer. Testing fourier dimensionality and sparsity. *SIAM J. Comput.*, 40(4):1075–1100, 2011. `doi:10.1137/100785429`.

[41] David Guijarro, Jun Tarui, and Tatsuie Tsukiji. Finding relevant variables in PAC model with membership queries. In *Algorithmic Learning Theory, 10th International Conference, ALT '99, Tokyo, Japan, December 6-8, 1999, Proceedings*, page 313, 1999. `doi:10.1007/3-540-46769-6\_26`.

[42] Shirley Halevy and Eyal Kushilevitz. Distribution-free property-testing. *SIAM J. Comput.*, 37(4):1107–1138, 2007. `doi:10.1137/050645804`.

[43] Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and boolean isoperimetric type theorems. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 52–58, 2015. `doi:10.1109/FOCS.2015.13`.

[44] Subhash Khot and Igor Shinkar. An $\widetilde{O}(n)$ queries adaptive tester for unateness. *CoRR*, abs/1608.02451, 2016. URL: `http://arxiv.org/abs/1608.02451`, `arXiv:1608.02451`.

[45] Richard J. Lipton, Evangelos Markakis, Aranyak Mehta, and Nisheeth K. Vishnoi. On the fourier spectrum of symmetric boolean functions with applications to learning symmetric juntas. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 112–119, 2005. `doi:10.1109/CCC.2005.19`.

[46] Zhengyang Liu, Xi Chen, Rocco A. Servedio, Ying Sheng, and Jinyu Xie. Distribution-free junta testing. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 749–759, 2018. `doi:10.1145/3188745.3188842`.

[47] Kevin Matulef, Ryan O'Donnell, Ronitt Rubinfeld, and Rocco A. Servedio. Testing ±1-weight halfspace. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, pages 646–657, 2009. `doi:10.1007/978-3-642-03685-9\_48`.

[48] Kevin Matulef, Ryan O'Donnell, Ronitt Rubinfeld, and Rocco A. Servedio. Testing halfspaces. *SIAM J. Comput.*, 39(5):2004–2047, 2010. `doi:10.1137/070707890`.

[49] Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning functions of k relevant variables. *J. Comput. Syst. Sci.*, 69(3):421–434, 2004. `doi:10.1016/j.jcss.2004.04.002`.

[50] Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 462–467, 1992. `doi:10.1145/129712.129757`.

[51] Michal Parnas, Dana Ron, and Alex Samorodnitsky. Testing basic boolean formulae. *SIAM J. Discrete Math.*, 16(1):20–46, 2002. URL: `http://epubs.siam.org/sam-bin/dbq/article/40744`.

[52] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987. `doi:10.1007/BF00058680`.

[53] Dana Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008. `doi:10.1561/2200000004`.

[54] Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009. `doi:10.1561/0400000029`.

[55] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996. `doi:10.1137/S0097539793255151`.

[56] Mert Saglam. Near log-convexity of measured heat in (discrete) time and consequences. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 967–978, 2018. `doi:10.1109/FOCS.2018.00095`.

[57] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. `doi:10.1145/1968.1972`.

9 Appendix A

In this Appendix, we give some bounds used in the paper.

Lemma 58. Markov's Bound. *Let $X \geq 0$ be a random variable with a finite expected value $\mu = \mathbf{E}[X]$. Then for any real numbers $\kappa, K > 0$,*

$$\mathbf{Pr}(X \geq \kappa) \leq \frac{\mathbf{E}[X]}{\kappa}. \tag{4}$$

$$\mathbf{Pr}(X \geq K\mathbf{E}[X]) \leq \frac{1}{K}. \tag{5}$$

Lemma 59. Chebyshev's Bound. *Let X be a random variable with a finite expected value $\mu = \mathbf{E}[X]$ and finite non-zero variance $\mathrm{Var}[X] = \mathbf{E}[X^2] - \mathbf{E}[X]^2$. Then for any real numbers $\kappa, K > 0$,*

$$\mathbf{Pr}(|X-\mu| \geq \kappa\sqrt{\mathrm{Var}[X]}) \leq \frac{1}{\kappa^2}. \tag{6}$$

$$\mathbf{Pr}(|X-\mu| \geq K) \leq \frac{\mathrm{Var}[X]}{K^2}. \tag{7}$$

Lemma 60. Chernoff's Bound. *Let $X_1, \ldots, X_m$ be independent random variables taking values in $\{0,1\}$. Let $X = \sum_{i=1}^m X_i$ denotes their sum, and let $\mu = \mathbf{E}[X]$ denotes the sum's expected value. Then*

$$\mathbf{Pr}[X > (1+\lambda)\mu] \leq \left(\frac{e^{\lambda}}{(1+\lambda)^{(1+\lambda)}}\right)^{\mu} \leq e^{-\frac{\lambda^2\mu}{2+\lambda}} \leq \begin{cases} e^{-\frac{\lambda^2\mu}{3}} & \textit{if } 0 < \lambda \leq 1 \\ e^{-\frac{\lambda\mu}{3}} & \textit{if } \lambda > 1 \end{cases}. \tag{8}$$

For $0 \leq \lambda \leq 1$ we have

$$\mathbf{Pr}[X < (1-\lambda)\mu] \leq \left(\frac{e^{-\lambda}}{(1-\lambda)^{(1-\lambda)}}\right)^{\mu} \leq e^{-\frac{\lambda^2\mu}{2}}. \tag{9}$$

Lemma 61. Hoeffding's Bound. *Let $X_1, \ldots, X_m$ are independent random variables taking values in $\{0,1\}$. Let $X = \sum_{i=1}^m X_i$ denote their sum and let $\mu = \mathbf{E}[X]$ denote the sum's expected value. Then for $0 \leq \lambda \leq 1$ we have*

$$\mathbf{Pr}[X > \mu + \lambda m] \leq e^{-2\lambda^2 m} \tag{10}$$

and

$$\mathbf{Pr}[X < \mu - \lambda m] \leq e^{-2\lambda^2 m} \tag{11}$$