# Testing Computability by Width-Two OBDDs*

Dana Ron† Gilad Tsur‡

**Abstract**

Property testing is concerned with deciding whether an object (e.g. a graph or a function) has a certain property or is "far" (for a prespecified distance measure) from every object with that property. In this work we consider the property of being computable by a read-once width-2 *Ordered Binary Decision Diagram* (OBDD), also known as a *branching program*, in two settings. In the first setting the order of the variables is fixed and given to the algorithm, while in the second setting it is not fixed. That is, while in the first setting we should accept a function $f$ if it is computable by a width-2 OBDD with a *given* order of the variables, in the second setting we should accept a function $f$ if there *exists* an order of the variables according to which a width-2 OBDD can compute $f$.

Width-2 OBDDs generalize two classes of functions that have been studied in the context of property testing - linear functions (over $GF(2)^n$) and monomials. In both these cases membership can be tested by performing a number of queries that is *independent of the number of variables,* $n$ (and is linear in $1/\epsilon$, where $\epsilon$ is the distance parameter). In contrast, we show that testing computability by width-2 OBDDs when the order of variables is fixed and known requires a number of queries that grows logarithmically with $n$ (for a constant $\epsilon$), and we provide an algorithm that performs $\tilde{O}(\log n/\epsilon)$ queries. For the case where the order is not fixed, we show that there is *no* testing algorithm that performs a number of queries that is sublinear in $n$.

## 1 Introduction

Property testing [28, 14] is concerned with deciding whether an object (e.g. a graph or a function) has a certain property or is "far" (with respect to a prespecified distance measure) from every object with that property. Typical property testing algorithms are randomized, and perform queries regarding local properties of the object (e.g., the value of a function $f$ on the input $x$), returning a correct answer with high probability. That is, the algorithm is given as input a distance parameter $\epsilon$ and is required to accept with high probability objects that have the property and to reject with high probability objects that are $\epsilon$-far from having the property. The goal is to design such algorithms whose query complexity and running time are sublinear in the size of the object and where the dependence on $1/\epsilon$ is as small as possible.

Property testing has been applied in a variety of contexts, with a particular emphasis on testing properties of graphs and on testing properties of functions (for surveys see e.g., [12, 9, 24, 25]).

*Part of the work presented in this paper appeared as an extended abstract in [26] .

†Email: danar@eng.tau.ac.il. This work was supported by the Israel Science Foundation (grant number 246/08).

‡Email: giladt@post.tau.ac.il.

This work belongs to the latter context. Roughly speaking, previous work on testing properties of functions mostly came in two "flavors": testing *algebraic* properties (e.g., linearity [4]), and testing *logical* properties (e.g., monomials [21]). Here we consider two (closely related) properties that are characterized by the *complexity class* to which the functions belong. While logical properties can also be viewed in this manner, there is an aspect in which our results can be viewed as a "new flavor" of results.

### 1.1 Our Results

In this paper we give upper and lower bounds for testing functions for the property of being computable by a read-once width-2 *Ordered Binary Decision Diagram* (OBDD), also known as a read-once *Oblivious Branching Program* of width 2, in two settings. In the first setting we are given an order of the variables $x_1, \ldots, x_n$ and oracle access to a function $f$. Here we must accept if there exists a width-2 OBDD with the given order of the variables that computes $f$, and must reject (with high probability) if $f$ is far from every function computable by a width-2 OBDD with the given order of variables. In the second setting the order of the variables is *not* fixed. That is, the algorithm must accept (with high probability) the function $f$ if there exists a width-2 OBDD with *some* order of variables that computes $f$, and it must reject $f$ (with high probability) if every width-2 OBDD, regardless of order of variables, computes a function that is far from $f$. The query complexity of our algorithm for the first setting is $\tilde{O}(\log(n)/\epsilon)$.[1] For the second setting we give a lower bound that precludes the existence of algorithms whose query complexity is sublinear[2] in $n$.

Width-2 OBDDs generalize two classes of functions that have been studied in the context of property testing – linear functions (over $GF(2)^n$) [28] and monomials [21]. In both these cases membership can be tested by performing a number of queries that is linear in $1/\epsilon$. Interestingly, unlike either of these classes, in which the query complexity of the testing algorithm does not depend on the number of variables in the tested function, we show that testing for computability by width-2 OBDDs requires $\Omega(\log(n))$ queries in the first setting and that $\Omega(n)$ queries are required in the second.

As noted above, the algorithm we present for testing computability by width-2 OBDDs with a fixed given order performs $\tilde{O}(\log(n)/\epsilon)$ queries, which almost matches the lower bound in terms of the dependence on $n$. Observe that the logarithmic dependence on $n$ is still much lower than the linear dependence that is necessary for learning this family of functions (under the uniform distribution and with queries), as the family of functions computable by width-2 OBDDs (over any fixed order) contains all linear functions.

Function classes for which property testing algorithms have been designed are usually characterized as either algebraic (e.g. [4, 28, 1, 16, 15]) or non-algebraic (e.g., [21, 10, 7]), though some results can be viewed as belonging to both categories. We view the family of functions we study as falling naturally into the second category, since it is described by a type of computational device and not by a type of algebraic formula. As opposed to many algorithms for algebraic families, algorithms for non-algebraic families generally rely on the fact that the functions in the family

[1] The notation $\tilde{O}(g(n))$ represents an upper bound that is linear in $g(n)$ up to a polylogarithmic factor.

[2] It is possible to get an almost linear dependence on $n$ (i.e., query complexity of $\tilde{O}(n/\epsilon)$), by an "Occam's razor" type argument. Namely, the algorithm tries to find a width-2 OBDD (with some order over the variables) that is consistent with a sample of uniformly selected inputs. If the function is $\epsilon$-far from any width-2 OBDD, then with high probability there will be no consistent width-2 OBDD.

are close to *juntas*, that is, functions that depend on a small number of variables. This is true, by definition, for singletons [21] and juntas [10], but also for monomials, monotone DNF with a bounded number of terms [21], general DNF, decision lists and many other function classes, studied in [7]. In contrast, our algorithms test for membership in classes of functions in which the function may depend (significantly) on many variables.

## 1.2 Techniques

Variables in functions that are computable by width-2 OBDDs can be divided into two groups. Variables that the function is "linear" in, which we refer to as "XOR" variables, and other, "nonlinear" variables, which we refer to as "AND" and "OR" variables. For reasons that will later become apparent we refer to the "nonlinear" variables also as *blocking variables*. This distinction is made more precise in Section 2. A simple but important observation is that if a width-2 OBDD has more than $\log(1/\epsilon)$ blocking variables, then the variables preceding them can essentially be ignored, since they have very little influence on the function. Another basic observation is that once we find the last $t$ non-linear variables (for some number $t < \log(1/\epsilon)$), we can continue the search for preceding nonlinear variables, by restricting some of the variables to a particular value.

### 1.2.1 The Upper Bound

The algorithm (that works for the fixed and known order case) relies on the known order of the variables. Say this order is $x_1, \ldots, x_n$. The algorithm has a subroutine for determining whether the last blocking variable belongs to $\{x_1, \ldots, x_{n/2}\}$ or to $\{x_{n/2+1}, \ldots, x_n\}$ (or possibly detects that the function is linear or that it cannot be computed by an width-2 OBDD with the given order of variables). This process is used to perform a binary search for the last blocking variable. Once such a variable is detected, the algorithm restricts the function to a particular assignment for this variable and all the variables following it, and continues the search for a new (earlier) blocking variable. Since the algorithm in the fixed order case rejects only when it finds evidence that the tested function is not computable by a width-2 OBDD (over the given order), it immediately follows that it always accepts functions in this family. The core of the proof is in showing that if the function is $\epsilon$-far from the family, then the algorithm rejects with high constant probability. More precisely, we prove the contrapositive statement. Since the algorithm works by verifying that various restrictions of the tested function are close to having certain properties, the difficulty is in proving that we can "glue" together these restrictions and obtain a single width-2 OBDD.

### 1.2.2 The Lower Bounds

We present (two-sided error) lower bounds both for the fixed order case and for the non-fixed order case. The lower bound for the fixed order case builds on two families of functions, one consisting of functions that are computable by width-2 OBDDs (over the given fixed order) and the other consisting of functions that are $\Omega(1)$-far from all those computable by width-2 OBDDs (with the given order). We show that any *non-adaptive* algorithm must perform $\Omega(n)$ queries to distinguish between members of these families, and an $\Omega(\log(n))$ lower bound follows for general (adaptive) algorithms. The lower bound for the case where the order is not fixed uses a reduction from a communication complexity problem, namely Set-Disjointness. This type of reduction was recently

introduced by Blais et al. [3], and was used by Brody et al. [5] to prove bounds for the query complexity of testing computability by several types of OBDDs.

### 1.3 Additional related work

As noted previously, width-2 OBDDs generalize two classes of functions that have been studied in the context of property testing - linear functions (over $GF(2)^n$) [28] and monomials [21]. In both these cases membership can be tested by performing a number of queries that is linear in $1/\epsilon$. Observe that in both cases the functions can be computed by width-2 OBDDs for *any* order of the variables.

We next note that the type of question we ask differs from that studied by Newman [20]. Newman shows that a property testing algorithm exists for any property decidable by a constant width branching program. That is, in [20] the property is defined with respect to a particular branching program, and the algorithm tests membership in a language decidable by that program. In contrast, in our result, the language we test for membership in is one where every word is the truth table of a width-2 branching program.

OBDDs and, in particular, bounded width OBDDs have been studied in the machine learning context rather extensively. It has been shown that width-2 OBDDs are PAC-learnable, while width-3 and wider OBDDs are as hard to learn as DNF formulas [8]. These results were strengthened in [6, 2]. When membership queries are allowed and the underlying distribution is uniform, width-2 branching programs with a single 0 sink are efficiently learnable [2]. When both membership and equivalence queries are allowed then there are several positive results for other restricted types of branching programs [22, 11, 18, 19, 2].

Lower bounds for testing computability by various sub-families of OBDDs have recently been studied by Goldreich [13] and Brody et al. [5]. In particular, Goldreich shows that testing for computability by width-4 OBDDs requires $\Omega(\sqrt{n})$ queries, in addition to giving lower bounds on testing computability by subclasses of linear functions over $GF(3)^n$ and $GF(2)^n$ (which can be computed by width-3 and width-2 OBDDs, respectively). Brody et al. strengthen the first result of Goldreich and prove that for any fixed $w \geq 4$, testing computability by width-$w$ OBDDs requires $\Omega(n)$ queries. Following our $\Omega(\log(n))$ lower bound for testing computability by width-2 OBDDs in the case where the order is fixed, Brody et al. give a similar bound using different techniques. They also give an $\Omega(\log(n))$ lower bound for testing computability by width-2 OBDDs in the case where the order is not fixed, which we strengthen to an $\Omega(n)$ lower bound.

### Errata

In the conference papers on which this work is based there were two errors. In our work on testing computability by width-2 OBDDs where the order of variables is fixed [26] there was an error in the proof of the lower bound (for one-sided error algorithms). The lower bound itself holds and here we prove a similar lower bound that holds also for two-sided error algorithms. In our work on testing computability by width-2 OBDDs where the order of variables is not fixed [27] there was an error in the proof of the upper bound. In this paper we give a lower bound that shows the impossibility of such a result.

### 1.4 Organization

In Section 2 we provide some basic definitions and present general claims regarding OBDDs and width-2 OBDDs in particular. In Section 3 we first give some basic definitions and claims that will only serve for the case where the order of variables is given to us and then introduce a one-sided error testing algorithm for that case. Finally, in Section 4 we first give a lower bound for testing of computability by width-2 OBDDs where the order of variables is fixed, and then give a lower bound for the case where the order of variables is not fixed.

## 2 Preliminaries

### 2.1 Basic Definitions and Notations

**Definition 2.1** *The* distance *between a function $f$ and a function $g$ over the same range $X$, denoted $d(f,g)$, is defined as $\Pr_x[f(x) \neq g(x)]$ where $x$ is drawn uniformly at random from $X$. When $d(f,g) > \epsilon$ we say that $f$ is $\epsilon$-*far *from $g$, otherwise it is $\epsilon$-*close*. For a family of functions $\mathcal{G}$ we let $d(f,\mathcal{G}) = \min_{g\in\mathcal{G}}\{d(f,g)\}$. If $d(f,\mathcal{G}) > \epsilon$, then we say that $f$ is $\epsilon$-*far *from $\mathcal{G}$.*

**Definition 2.2** *A* property testing algorithm *$T$ for membership in a function class $\mathcal{F}$ is given oracle access to a function $f$ and a distance parameter $0 < \epsilon < 1$.*

1. *If $f \in \mathcal{F}$, then $T$ accepts with probability at least $2/3$ (over its internal coin tosses);*
2. *If $f$ is $\epsilon$-far from $\mathcal{F}$, then $T$ rejects with probability at least $2/3$ (over its internal coin tosses).*

*A property testing algorithm is said to have* one-sided error *if it accepts every $f \in \mathcal{F}$ with probability 1.*

Of course, if we wish to increase the success probability from $2/3$ to $1-\delta$ for some given confidence parameter $\delta$, then we can repeat the application of a property testing algorithm $\Theta(\log(1/\delta))$ times and take a majority vote. Later in this work we will routinely "amplify" the probability of success as required.

A property testing algorithm that we use as a basic building block in our algorithms is the linearity tester, proposed by Blum, Luby and Rubinfeld [4]. In [4] it is assumed that for a linear function $f$ it holds that $f(0^n) = 0$. For our purposes, linearity allows for a "free" coefficient, and the BLR algorithm is easily adapted to such a case.

**Definition 2.3** *We say that $f : \{0,1\}^n \to \{0,1\}$ is a* linear *function if there exist coefficients $b_0, b_1, \ldots, b_n \in \{0,1\}$ such that for $x = x_1, \ldots, x_n \in \{0,1\}^n$, $f(x) = b_0 + \Sigma_{i=1}^n b_i x_i$.*

**Theorem 2.1 ([4])** *There exists a one-sided error testing algorithm for linearity. Its query complexity is $O(1/\epsilon)$.*

We now turn to defining (O)BDDs.

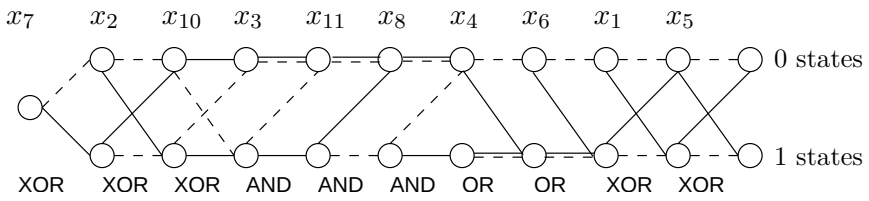


Figure 1: An example of a width-2-OBDD. The dashed lines represent 0 transitions and the solid lines represent 1 transitions. The type of each variable (see Definition 2.6) appears at the bottom of the figure.

**Definition 2.4** *A* Binary Decision Diagram *(BDD), also known as a branching program, is an acyclic directed graph with a single source, where sinks are labeled* 0 *or* 1*, and each other node is labeled by a Boolean variable from* $X = \{x_1, ....x_n\}$*. Every non-sink node has two outgoing edges (potentially multi-edges), labeled* 0 *and* 1*, respectively. The Boolean function associated with a BDD is computed on a particular input* $y = y_1, \ldots, y_n \in \{0,1\}^n$ *by returning the label of the sink reached when this input is used to trace a route through the graph by leaving each node labeled* $x_i$ *along the edge labeled* $y_i$*.*

There are different definitions in the literature for Ordered Binary Decision Diagrams. Our results hold for the definition of a *strict* fixed width binary decision diagram (for an illustration, see Figure 1):

**Definition 2.5** *An* Ordered Binary Decision Diagram *(OBDD) is a BDD that consists of* $n+1$ levels*, according to the distance of the nodes from the source node. The source node is in the first level, the sink nodes are in the last level, and for each level* $i$ *but the last, the nodes in level* $i$ *are labeled by the same variable, where no two levels correspond to the same variable. The* width *of an OBDD is the maximum number of nodes in any level.*

Branching programs can, of course, compute functions from other, non-binary, finite domains and to other finite ranges. We also note that other definitions of OBDDs (which we do not use) allow sinks on any level, or sinks labeled 0 on any level and only one sink labeled 1.

## 2.2 Properties of OBDDs

In this subsection we provide several definitions and claims that set the ground for our algorithm and that are used in the analysis. Since most of the definitions are self-explanatory, and most of the claims are fairly simple, one may choose to continue directly to the description of our algorithm, and refer to this subsection only when needed.

Our first claim follows directly from the definition of OBDDs.

**Claim 2.1** *A function* $f : \{0,1\}^n \to \{0,1\}$ *is computable by a width-*2 *OBDD with an order of the variables* $z_1, \ldots, z_n$ *if and only if there exist functions* $g_n$ *and* $f_{n-1}$ *such that* $f(z_1, \ldots, z_n) = g_n(f_{n-1}(z_1, \ldots, z_{n-1}), z_n)$*, where* $f_{n-1}$ *is a function computable by a width-*2 *OBDD (over* 0 *variables if* $n = 1$*) and* $g_n$ *is a function from* $\{0,1\} \times \{0,1\}$ *to* $\{0,1\}$*.*

**Proof:** We prove this by induction on the number of variables. In the base case we have a function of 1 variable and the claim is trivial. The induction step in each direction is simple too - if we have a function $f$ on $n-1$ variables computable by a width-2 OBDD and a function $g$ that accepts $f$'s output and the $n$'th variable, we can simulate it by placing the correct edges in a width-2 OBDD with $n$ variables. Likewise, if we have an OBDD of length $n$ and width 2, the state on the $n-1$'th level is computable by a width-2 OBDD, and the final state of the OBDD is only a function of the state before last and of the $n$'th bit. ■

In this work we will often find it convenient to discuss a partial traversal of an OBDD and not the full computation performed by it. We will routinely relate to one of the (at most) two vertices in a layer of the width-2 OBDD as 0 and to the other as 1, as done in Claim 2.1. We denote by $f_i : \{0,1\}^i \to \{0,1\}$ the function that is given the string $y_1, \ldots, y_i$ and returns the value of the node in the $(i+1)$'th level reached from the source by traversing the OBDD according to them.

**Definition 2.6** *For a fixed order of the variables $z_1, \ldots, z_n$, and a fixed denotation of different states in an OBDD $M$ as being 0 states or 1 states, we say that the variable $z_i$ is an* AND variable *if $f_{i+1}(z_1, \ldots, z_i) = f_i(z_1, \ldots, z_{i-1}) \wedge \ell_i$, where $\ell_i$ is either $z_i$ or $\overline{z}_i$ (and $f_0$ is either 0 or 1).* OR variables *and* XOR variables *are defined in a similar manner. We call this description the* type *of a variable - thus, the type of a variable is either* AND, OR *or* XOR.

*We will refer to* AND *variables and* OR *variables as* blocking variables *for an OBDD $M$.*

Note that the first variable in an ordering can be considered, according to this definition, as any one of the three types of variables. We generally consider it to be the same type of variable as the one just following it. In the case where there is only a single variable with influence in $f$ we consider it a XOR variable.

**Claim 2.2** *Let $f$ be a function computable by a width-2 OBDD with the order of variables $z_1, \ldots, z_n$. Let $z_k, \ldots, z_{k+\ell}$ be a set of* XOR *variables. It holds that $f_{k+\ell+1}(z_1, \ldots, z_{k+\ell})$ is a linear function of $f_k(z_1, \ldots, z_{k-1})$ and of $z_k, \ldots, z_{k+\ell}$.*

Claim 2.2 follows directly from Definition 2.6.

Let $y^{(i)}$ be the string $y$ with the $i$'th bit flipped. Recall that the bit influence of the $i$'th bit in the function $f$, which we denote by $I_i(f)$ or by $I_{x_i}(f)$, is $\Pr[f(y) \neq f(y^{(i)})]$ when $y$ is drawn from the uniform distribution.

**Claim 2.3** *Let $M$ be a width-2 OBDD, where the order of the variables is $z_1 \ldots z_n$ and let $z_i$ be an* AND *variable or an* OR *variable in $M$. For a variable $z_j$ where $j < i$ it holds that $I_{z_j}(f_i) \leq \frac{1}{2} I_{z_j}(f_{i-1})$.*

**Proof:** Consider, without loss of generality, a width-2 OBDD $M$ where the variable $z_i$ is an AND variable. For $j < i$ we have:

$$\begin{aligned} \mathrm{I}_j(f_i) &= \Pr[f_i(z) \neq f_i(z^{(j)})] && (1) \\ &= \frac{1}{2}\Pr[f_i(z) \neq f_i(z^{(j)}) \mid z_i = 0] + \frac{1}{2}\Pr[f_i(z) \neq f_i(z^{(j)}) \mid z_i = 1] && (2) \\ &= \frac{1}{2}\Pr[f_i(z) \neq f_i(z^{(j)}) \mid z_i = 1] \leq \frac{1}{2} I_j(f_{i-1}) && (3) \end{aligned}$$

and the proof is completed. ■

The next claim follows from the structure of width-2 OBDDs.

**Claim 2.4** *For a function $f$ computable by a width-2 OBDD $M$, the influence of $z_1, \ldots, z_{i-1}$ in $f_i$ is no greater than their influence in $f_{i-1}$.*

**Claim 2.5** *Let $f$ be a function of the form*

$$f(x_1, \ldots, x_n) = f'(g(x_1, \ldots, x_{n-m}), x_{n-m+1}, \ldots, x_n)$$

*where $f'$ is computable by a width-2 OBDD $M$ over the order of the variables $z_1, \ldots, z_{m+1}$, where $z_2 = x_{n-m+1}, \ldots, z_{m+1} = x_n$ . If $M$ has at least $k$ blocking variables, then $f$ is $2^{-k}$-close to the function $f'(0, x_{n-m+1}, \ldots, x_n)$, that is computable by a width-2 OBDD.*

**Proof:** By Claim 2.3 the influence of the first bit of $f'$ is at most $2^{-k}$. The claim follows. ■

# 3 A Testing Algorithm for the Fixed Ordered Case

In this section we assume, unless stated otherwise, that the OBDDs discussed are with the order of the variables $x_1, \ldots, x_n$ and that this order is known to us.

## 3.1 Preliminaries

We will use the following notation for restricting several consecutive variables. Let $f : \{0,1\}^n \to \{0,1\}$ be a function and let $w \in \{0,1\}^m$ be a string. We define $f_{i,j,w}$, where $j = i + m - 1$, to be the function $f$ with the variables $x_i, \ldots, x_j$ assigned the values $w_1, \ldots, w_m$, respectively. This means that

$$f_{i,j,w}(x_1, \ldots, x_n) \equiv f(x_1, \ldots, x_{i-1}, w_1, \ldots, w_m, x_{j+1}, \ldots, x_n).$$

Since the assignment to the variables $x_i, \ldots, x_j$ is fixed, we either view $f_{i,j,w}$ as a function of $n$ variables (where the variables $x_i, \ldots, x_j$ have no influence), or as a function of $n - m$ variables.

**Definition 3.1** *For a given order of the variables $x_1, \ldots, x_n$ and an index $i \in [n]$, a set $S$ is said to be an $i$-*Prefix Equivalence Class *(or just an $i$-equivalence class) for a function $f : \{0,1\}^n \to \{0,1\}$ (where $i \leq n$) when $S$ is a maximal subset of $\{0,1\}^i$ such that for all $y_1, y_2 \in S$ and for all $w \in \{0,1\}^{n-i}$ it holds that $f(y_1 w) = f(y_2 w)$.*

As a corollary of Claim 2.1 we have:

**Corollary 3.1** *A function $f : \{0,1\}^n \to \{0,1\}$ is computable by a width-2 OBDD with the order of the variables $x_1, \ldots, x_n$ if and only if $\forall i \in [n]$ there are at most 2 distinct $i$-prefix equivalence classes for $f$.*

**Proof:** If a function is computable by a width-2 OBDD, then it has, for all $i \in [n]$, at most two nodes on its $i$'th level (denoted arbitrarily 0 and 1 if two such nodes exist, otherwise denoted as 0). Thus, each prefix $x_1, \ldots, x_i$ either belongs to the equivalence class of all prefixes reaching the node 0, or to that of all prefixes reaching the node 1.

In the other direction, let $f$ be a function of $n$ variables where $\forall i \in [n]$ there are at most two distinct $i$-prefix equivalence classes for $f$. we prove by induction on $n$ that $f$ is computable by a

width-2 OBDD. The base case is where $f$ is a function of 0 variables, which can clearly be computed by a width-2 OBDD. For the induction step, we consider $f_{n-1}$, the function telling us to which equivalence class in $f$ each prefix of length $n-1$ belongs. As $f$ is a function of $f_{n-1}$ (which is computable by a width-2 OBDD by the induction hypothesis) and of $x_n$, we have from Claim 2.1 that $f_n$ is computable by a width-2 OBDD. ■

**Definition 3.2** *Let $S_1, S_2$ be two $i$-prefix equivalence classes for a function $f$. A string $w \in \{0,1\}^{n-i}$ is a* distinguishing assignment *for $S_1$ and $S_2$ if for every $y_1 \in S_1, y_2 \in S_2$ it holds that $f(y_1 w) \neq f(y_2 w)$.*

### 3.2 The Testing Algorithm

To gain some intuition for the way the testing algorithm works, we first consider the following scenarios.

Imagine that we are given query access to a function $f$ where we have a promise that $f$ is either computable by a width-2 OBDD that has no blocking variables, or $f$ is far from any function computable by a width-2 OBDD. We could check which of the above is the case using BLR's linearity test on $f$, as a function computable by a width-2 OBDD that has no blocking variables is a linear function.

Now, imagine we are promised that $f$ is either far from any function computable by a width-2 OBDD or that it is computable by a width-2 OBDD that has exactly one blocking variable in the $i$'th level, where $i$ is known. We could check to see which of the cases above holds by going through the following procedure. First we would like to see if $f$ has at most two $i$-equivalence classes. We cannot know this exactly, but we are able to tell if $f$ is close to a function with 1, 2, or more $i$-equivalence classes using an algorithm we will describe below. If we only find one $i$-equivalence class for $f$ it remains to check if $f$ is a linear function of $x_{i+1}, \ldots, x_n$. If it is, then $f$ is computable by a width-2 OBDD with one blocking variable, $x_i$ (and we can accept). If $f$ has more than two $i$-equivalence classes then it is clearly not computable by a width-2 OBDD (of any kind), and we can reject. Finally, if $f$ has exactly two $i$-equivalence classes, then we must check that the function $f_{i-1}$ (the function that maps the variables $x_1, \ldots, x_{i-1}$ to $(i-1)$-equivalence classes) is linear, and that the function which maps the $i$-equivalence class and the variables $x_{i+1}, \ldots, x_n$ to $f(x_1, \ldots, x_n)$ is linear, as well.

As a final hypothetical scenario, consider the following promise: $f$ is either far from every function computable by a width-2 OBDD, or it can be computed using a width-2 OBDD with a single *unknown* blocking variable. If we could locate the blocking level, then we could tell which of the two cases holds, as done in the previous paragraph. We note that as a consequence of Claim 2.3, any function that is computable by a width-2 OBDD with a single blocking variable, is far from linear, so we would like to check $f$ and see what parts of it are linear. We can do this by performing a binary search for a linear section. Begin by restricting the first $n/2$ variables to 0, and checking if the function computed on all the rest of the variables is (close to) linear. If it is, repeat the process with fewer variables restricted. If it isn't, repeat the process with more variables restricted. If we are, indeed, given a function that is computable by a width-2 OBDD that has only a single blocking variable, this process will allow us to detect the blocking variable with high probability.

The property testing algorithm we suggest for computability by a width-2 OBDD is based on the observations made above and on those made in the previous sections. In particular, we note

that, as a consequence of Claim 2.5, any function $f$ computable by a width-2 OBDD is $\epsilon$-close to a function $g$ computable by a width-2 OBDD that has $O(\log(1/\epsilon))$ blocking variables, where the blocking variables of $g$ are all blocking levels of $f$. When our algorithm is given as input a function computable by a width-2 OBDD, it will (with high probability) locate the last $O(\log(1/\epsilon))$ blocking variables (if such blocking variables exist, of course). Locating these variables will be done using a binary search technique reminiscent of the one suggested above. We will restrict the function on some of its bits $(x_1, \ldots, x_j)$ and test whether the restricted function is linear, using a version of the BLR linearity test. For any function $f$ computable by a width-2 OBDD our algorithm will find the structure of a function $g$ that is close to it, and for every function that passes our test, we will show that it is likely to be close to a function computable by a width-2 OBDD.

A notion that is used repeatedly is that of a function $f$ that can be computed by a width-2 OBDD that accepts as input the value of a function $g(x_1, \ldots, x_t)$ and the bits $x_{t+1}, \ldots, x_n$ (in that order) and outputs the value $f(x_1, \ldots, x_n)$. We define this formally:

**Definition 3.3** *A function $f : \{0,1\}^n \to \{0,1\}$ is said to be a* W2-function *of $g : \{0,1\}^t \to \{0,1\}$ and of $x_{t+1}, \ldots, x_n$ if there exists a width-2 OBDD that accepts as input the value $g(x_1, \ldots, x_t)$ and the bits $x_{t+1}, \ldots, x_n$ (in that order) and outputs the value $f(x_1, \ldots, x_n)$.*

In Figure 2 we present the testing algorithm for computability by a width-2 OBDD when the order of variables is fixed and known. In the algorithm we use a parameter $\epsilon'$, which intuitively stands for the amount of error we are willing to accumulate during each round of the algorithm. We set $\epsilon' = \epsilon/(4\log(1/\epsilon))$. The algorithm uses two sub-procedures, **Get-linear-level** and **Count-equiv-classes**, both described after the algorithm.

**Theorem 3.2** *Algorithm* **Test-width-**2 *is a one-sided error testing algorithm for the property of being computable by a width-*2 *OBDD for a fixed given order of the variables. The algorithm performs $\tilde{O}(\log(n)/\epsilon)$ queries.*

### 3.3 Building Blocks

We now proceed to discuss the probabilistic procedures used in **Test-width-**2 . We later return to proving Theorem 3.2. We have already mentioned the BLR linearity test, one procedure that we will use as an internal building block in our own sub-procedures. We now turn to describe an additional building block - a procedure that is given access to a function $f$ and a number $i$, and attempts to check whether $f$ has $1, 2$ or more $i$-equivalence classes. Despite the fact it only counts up to 2, or perhaps up to "many", we dub this sub-procedure **Count-equiv-classes**. A precise description of the algorithm appears in Figure 3. The straightforward approach to performing this task may be to take a set of prefixes of length $i$ and compare each two (or all of them) on a set of suffixes, trying to find prefixes that belong to different equivalence classes. A simple analysis implies a procedure that performs $\Theta(1/\epsilon^2)$ queries. The approach we take is slightly different. We start with the arbitrary string $0^i$, which belongs to some equivalence class. To identify a second equivalence class we simply test the equality of $f(x_1, \ldots, x_n)$ with $f_{1,i,0^i}(x_{i+1}, \ldots, x_n)$. If a second equivalence class is detected then we use a similar technique to try and find a third equivalence class (with a small adjustment). This approach leads to a $\Theta(1/\epsilon)$ algorithm.

**Test-width-2**
Input: Oracle access to a function $f$; Precision parameter $\epsilon$.

1. Let $f^0 = f$. Let $r = 1$ and $t^0 = n$.
   Here $t$ will be the number of variables of $f$ that we haven't restricted, and $r$ will be the number of the current round. The indexes on $t$ and $f$ will indicate the round $r$ and help us keep track of different values for the analysis.
2. While $t^{r-1} \neq 1$ and $r \leq \log(1/\epsilon) + 2$
   (a) **Locate linear section:** Run **Get-linear-level**($f^{r-1}, \epsilon'/3, \frac{1}{6(\log(1/\epsilon)+2)}$).
   This locates the last index $j$ such that $f^{r-1}$ is $(\epsilon'/3)$-close to a linear function of $f^{r-1}_{j+1,t^{r-1},w}(x_1, \ldots, x_j)$ and of $x_{j+1}, \ldots, x_{t^{r-1}}$ for a distinguishing sequence $w$.
   (b) If **Get-linear-level** indicated the existence of more than 2 different $i$-equivalence classes on some level $i$, reject.
   (c) Otherwise, let $j$ be the level returned by **Get-linear-level** and let $w$ be the distinguishing sequence returned by it.
   (d) Let $g^r = f^{r-1}_{j+1,t^{r-1},w}$ and let $\tilde{t}^r = j$.
   (e) If $j \neq 1$
      i. Run **Count-equiv-classes**($g^r$, $\tilde{t}^r - 1$, $\epsilon'/3$, $\frac{1}{12\log(1/2\epsilon)}$).
      This tells us whether the number of $(\tilde{t}^r - 1)$-equivalence classes in $g^r$ is $1, 2$ or more with precision $\epsilon'/3$.
      ii. If a single equivalence class is found, accept.
      iii. If more than 2 equivalence classes are found, reject.
      iv. Let $w'$ denote the distinguishing assignment (of size 1) between the 2 equivalence classes found (returned by **Count-equiv-classes**). Let $f^r = g^r_{j,j,w'}$ and let $t^r = j - 1$.
   (f) Else, let $f^r = g^r$ and let $t^r = \tilde{t}^r$.
   (g) $r = r + 1$.
3. return accept.

Figure 2: Algorithm Test-width-2 (fixed order case).

**Claim 3.1** *The algorithm* **Count-equiv-classes**, *given oracle access to a function $f$ acts as follows:*

1. *If $f$ is $\epsilon$-far from every function with one $i$-equivalence class, then with probability at least $1 - \delta$* **Count-equiv-classes** *will return representatives of at least two equivalence classes.*
2. *If $f$ is $\epsilon$-far from every function with at most two $i$-equivalence classes, then with probability at least $1 - \delta$* **Count-equiv-classes** *will return representatives of three equivalence classes.*
3. *In any case* **Count-equiv-classes** *does not indicate the existence of more than the number of $i$-equivalence classes of $f$.*
4. *Conditioned on Count-equiv-classes($f$) returning the representatives of 2 different $i$-equivalence classes and a distinguishing assignment, with probability at least $1 - \delta$ it holds that $f$ is $\epsilon$-close to a function of $f_{i+1,n,w}(x_1, \ldots, x_i)$ (where $w$ is the distinguishing assignment) and of the variables $x_{i+1}, \ldots, x_n$.*

**Count-equiv-classes**
Input: Oracle access to a function $f$; Integer value $0 < i \leq n$; Precision parameter $\epsilon$; Confidence parameter $\delta$.

1. Select $m = \Theta(\log(1/\delta)/\epsilon)$ strings $x^1, \ldots, x^m$ from $\{0,1\}^n$.
2. If $f(x^j) = f_{1,i,0^i}(x^j_{i+1}, \ldots, x^j_n)$ for all $x^j \in \{x^1, \ldots, x^m\}$, then output that 1 equivalence class was found. Otherwise, let $y \in \{0,1\}^i, w \in \{0,1\}^{n-i}$ be such that $f(yw) \neq f(0^i w)$.
3. Select $m = \Theta(\log(1/\delta)/\epsilon)$ new strings $z^1, \ldots, z^m$ from $\{0,1\}^n$.
4. Define $g(z^j)$ as 0 if $f(0^i, w) = f(z^j_1, \ldots, z^j_i, w)$, and as 1 otherwise. Compute $g(z^j)$ for all $j$.
5. For each $j$, if $g(z^j) = 0$ and $f(z^j) \neq f(0^i, z^j_{i+1}, \ldots, z^j_n)$, then output representatives of 3 different $i$-equivalence classes *($0^i$, $y$ and $z^j_1, \ldots, z^j_i$)* and distinguishing assignments for them *($w$ and $z^j_{i+1}, \ldots, z^j_n$)*. Do the same if $g(z^j) = 1$ and $f(z^j) \neq f(y, z^j_{i+1}, z^j_n)$.
6. Output the representatives of 2 equivalence classes *($0^i$ and $y$)* and a distinguishing assignment for them *($w$)*.

Figure 3: Algorithm Count-equiv-classes.

*The algorithm performs* $O\left(\frac{\log(1/\delta)}{\epsilon}\right)$ *queries.*

**Proof:** If $f$ is $\epsilon$-far from any function with one $i$-equivalence class, then it is, in particular, $\epsilon$-far from $f_{1,i,0^i}(x_{i+1}, \ldots, x_n)$. We will therefore encounter a string $x^j$ such that $f(x^j) \neq f_{1,i,0^i}(x^j_{i+1}, \ldots, x^j_n)$ with probability at least $1-(\delta/2)$ if we set $m$ appropriately, and will thus return the representatives of at least two $i$-equivalence classes. Likewise, if $f$ is $\epsilon$-far from any function with at most two $i$-equivalence classes, it is in particular $\epsilon$-far from the function that it is compared to in Step 5. Namely, this is the function that takes the value $f_{1,i,0^i}$ for all $z$ such that $g(z) = 0$, and takes the value $f_{1,i,y}$ for all $z$ such that $g(z) = 1$. Hence, with probability at least $1 - (\delta/2)$ we will obtain a string $z^j$ such that $g(z^j) = 0$ and $f(z^j) \neq f(0^i, z^j_{i+1}, z^j_n)$ or such that $g(z^j) = 1$ and $f(z^j) \neq f(y, z^j_{i+1}, z^j_n)$. As a consequence, we shall output the representatives of 2 different $i$-equivalence classes. Under no circumstances does the algorithm indicate the existence of equivalence classes for which no witness was found.

It remains to prove the last item. To this end we define the function $h_w(x)$ as follows: If $g(x) = 0$ then $h_w(x) = f(0^i, x_{i+1}, \ldots, x_n)$, and otherwise $h_w(x) = f(y_1, \ldots, y_i, x_{i+1}, \ldots, x_n)$. For the string $w$ determined in Step 2 it holds that if $f$ is $\epsilon$-far from $h_w$ then we reject with probability at least $1 - \delta$ in Step 5 (which can be seen as testing identity between $f$ and $h_w$). As $h_w$ is a function of $f_{i+1,n,w}(x_1, \ldots, x_i)$ and of the variables $x_{i+1}, \ldots, x_n$ the conclusion follows.
The query complexity is linear in $m$, which is $O(\log(1/\delta)/\epsilon)$. ■

Before describing the algorithm **Get-linear-level** (in Figure 5) and proving its correctness, we describe the algorithm **Test-level-linearity** (see Figure 4) that it uses as a building block.

**Claim 3.2** *When given oracle access to a function $f$ and a value $i$,* **Test-level-linearity** *acts as follows:*

**Test-level-linearity**
Input: Oracle access to a function $f$; Integer value $0 < i \leq n$; Precision parameter $\epsilon$; Confidence parameter $\delta$.

1. Run **Count-equiv-classes**($i$, $\epsilon' = \epsilon/4$, $\delta' = \delta/4$).
2. If **Count-equiv-classes** returns representatives of 3 different $i$-equivalence classes, output reject.
3. If **Count-equiv-classes** returns 1 equivalence class, run the BLR linearity test on $f_{1,i,0^i}$ with precision parameter $\epsilon' = \epsilon/3$ and with confidence $\delta' = \delta/4$. If the test accepts - output accept and $0^i$ as a distinguishing assignment, otherwise, output reject.
4. If **Count-equiv-classes** returns representatives of 2 different $i$-equivalence classes, $x$ and $y$, and a distinguishing assignment $w$, do the following:
   (a) Run the BLR linearity test on $f_{1,i,x}$ and on $f_{1,i,y}$ with precision $\epsilon' = \epsilon/4$ and with confidence $\delta' = \delta/4$. If either test rejects, reject.
   (b) Select $m = \Theta(\log(1/\delta)/\epsilon)$ strings in $\{0,1\}^{n-i}$, denoted $w^1, \ldots, w^m$.
   (c) If $f_{1,i,x}(w^j) = f_{1,i,y}(w^j)$ for all $j$, or $f_{1,i,x}(w^j) \neq f_{1,i,y}(w^j)$ for all $j$, output accept and the distinguishing assignment $w$. Otherwise output reject.

Figure 4: Algorithm Test-level-linearity.

1. *If $f$ is a linear function of the output of a Boolean function $g(x_1, \ldots, x_i)$ and of the variables $x_{i+1}, \ldots, x_n$, then* **Test-level-linearity** *accepts, and outputs a sequence $w$ such that $f_{i+1,n,w}$ equals $g$.*
2. *If $f$ is $\epsilon$-far from any linear function of any function $g(x_1, \ldots, x_i)$ and of the variables $x_{i+1}, \ldots, x_n$, then with probability at least $1 - \delta$* **Test-level-linearity** *rejects, possibly identifying 3 different equivalence classes.*

*The algorithm performs $\Theta(\log(1/\delta)/\epsilon)$ queries.*

**Proof:** We start with Item 1. **Test-level-linearity** rejects in only 4 cases, none of which happen if $f$ is a linear function of a function $g(x_1, \ldots, x_i)$ and of the variables $x_{i+1}, \ldots, x_n$:

1. The algorithm encounters 3 different $i$-equivalence classes. This is clearly impossible as $g(x_1, \ldots, x_i)$ can only take (at most) 2 different values.
2. The algorithm encounters a single equivalence class, and restricted to that class, $f$ is not a linear function. As any restriction on the Boolean variables of a linear function gives a linear function, if $f'$ is linear, so is, e.g., $f'_{1,1,0}$.
3. The algorithm encounters two equivalence classes, and restricted to one of them $f$ is not a linear function. Again, as any restriction on the Boolean variables of a linear function gives a linear function, if $f'$ is linear, so is, e.g., $f'_{1,1,0}$.

4. We have two strings $x, y$ such that $f_{1,i,x}(w^j) = f_{1,i,y}(w^j)$ on some assignment $w^j$, and $f_{1,i,x}(w^k) \neq f_{1,i,y}(w^k)$ on some assignment $w^k$. This clearly doesn't happen in a linear function $f'(g(x_1,\ldots,x_i),x_{i+1},\ldots,x_n)$ where the bit $g(x_1,\ldots,x_i)$ has influence 1, and likewise doesn't happen in a linear function $f'(g(x_1,\ldots,x_i),x_{i+1},\ldots,x_n)$ where the bit $g(x_1,\ldots,x_i)$ has *no* influence. As every bit in a linear function has either influence 1 or 0, it holds that $f'$ is not a linear function.

We show the correctness of Item 2 as follows. We assume none of the sub-procedures used by **Test-level-linearity** fails, e.g., when a call is made to the BLR linearity test it accepts only if the function it is invoked upon is indeed $(\epsilon/4)$-close to linear, and (despite this not being an explicit sub-procedure) we assume that in Step 4c of **Test-level-linearity** we accepted a function $f$ where $f_{1,i,x}(w^j) = f_{1,i,y}(w^j)$ on all but $\epsilon/4$ of the values or that $f_{1,i,x}(w^j) \neq f_{1,i,y}(w^j)$ on all but $\epsilon/4$ of the values. The cumulative probability of one of these sub-procedures failing is at most $\delta$, and thus the conclusion will follow once we show that such success and $f$ passing the test implies that $f$ is $(\epsilon)$-close to a linear function of some function $g(x_1,\ldots,x_i)$ and of the variables $x_{i+1},\ldots,x_n$.

Assuming all sub-procedures succeed as described above, if $f$ is accepted in Step 3, then on all but at most $\epsilon/4$ of the inputs, $f$ is close to a function that does not depend on the variables $x_1,\ldots,x_i$ (by the correctness of Count-equiv-classes), and on all but at most $\epsilon/3$ of the inputs it equals a linear function of $x_{i+1},\ldots,x_n$. It follows that $f$ is $\epsilon$-close to a linear function of the variables $x_{i+1},\ldots,x_n$ and of a constant function (of the variables $x_1,\ldots,x_i$). If $f$ is accepted in Step 4, then (by the correctness of Count-equiv-classes) it is $(\epsilon/3)$-close to a function of the variables $x_{i+1},\ldots,x_n$ and of $f_{i+1,n,w}(x_1,\ldots,x_i)$ where $w$ is the distinguishing assignment returned by Count-equiv-classes. Step 4a ensures that $f$ is $(\epsilon/4)$-close to a linear function when $f_{i+1,n,w}(x_1,\ldots,x_i) = 0$, and Steps 4b and 4c ensure that when $f_{i+1,n,w}(x_1,\ldots,x_i) = 0$ and when $f_{i+1,n,w}(x_1,\ldots,x_i) = 1$ the function $f$ is either close to being the same linear function of $x_{i+1},\ldots,x_n$ (making $f$ close to a linear function of $x_{i+1},\ldots,x_n$ that disregards the values $x_1,\ldots,x_i$) or is close to being an opposite function (making $f$ close to a linear function of $x_{i+1},\ldots,x_n$ and of $f_{i+1,n,w}(x_1,\ldots,x_i)$), as required. ■

**Claim 3.3** *When* **Get-linear-level** *is given oracle access to a function $f$ (of $n$ variables), a precision parameter $\epsilon$ and a confidence parameter $\delta$ it acts as follows.* **Get-linear-level** *rejects only if more than 2 different $i$-equivalence classes were located for some $i$. Otherwise, with probability greater or equal to $1-\delta$ it returns a value $1 \leq i \leq n$ and a string $w$ so that the following hold:*

1. *The function $f$ is $\epsilon$-close to a linear function of $f_{j+1,n,w}(x_1,\ldots,x_j)$ and of the variables $x_{j+1},\ldots,x_n$.*
2. *If $j \neq 1$, then the function $f$ is not a linear function of $f_{j,n,w}(x_1,\ldots,x_{j-1})$ and of the variables $x_j,\ldots,x_n$ for any[3] $w$.*

*The algorithm performs $O\left(\frac{\log(n)\log(\log(n)/\delta)}{\epsilon}\right)$ queries.*

**Proof:** We prove that both items hold conditioned on **Test-level-linearity** never failing (i.e., never accepting when in fact $f$ is far from every linear function of some $g(x_1,\ldots,x_i)$ and of the

[3]Note that this is true with probability 1 due to the one-sided rejection criteria of **Test-level-linearity**, but the claim as is suffices

**Get-linear-level**
Input: Oracle access to a function $f$; Precision parameter $\epsilon$; Confidence parameter $\delta$.

1. Let $min = 1$ and $max = n$.
2. Let $w$ be the empty string.
3. While $min < max$
   (a) Let $mid = \lfloor (max + min)/2 \rfloor$
   (b) Run **Test-level-linearity**($f$, $mid$, $\epsilon$, $\delta' = \delta/\log(n)$). If **Test-level-linearity** finds 3 different $mid$-equivalence classes, reject.
   (c) If **Test-level-linearity** returns accept set $max = mid$ and set $w$ to be the distinguishing sequence.
   (d) Otherwise, set $min = mid + 1$
4. return $mid$ and $w$.

Figure 5: Algorithm Get-linear-level.

variables $x_{i+1}, \ldots, x_n$). The probability of such a failure on each round is at most $\delta/\log(n)$, which cumulatively never exceeds $\delta$ in the (at most) $\log(n)$ rounds that the algorithm performs.

The correctness of Item 1 when $i \neq n$ follows from the correctness of **Test-level-linearity** according to Claim 3.2 - $f$ passed the subroutine **Test-level-linearity** on the value $i$ and is thus $\epsilon$-close to a linear function of $f_{i+1,n,w}(x_1, \ldots, x_i)$ and of the variables $x_{i+1}, \ldots, x_n$. When $i = n$ the result is trivial as the linear function is of one variable: $f(x_1, \ldots, x_n)$.

The correctness of Item 2 again follows from the correctness of **Test-level-linearity**. Unless $j = 1$ the function **Test-level-linearity** rejected when invoked on the $j - 1$'th level of $f$ (by the structure of binary search), and thus $f$ is not a linear function of $f_{i,n,w}$ and of the variables $x_i, \ldots, x_n$ for any string $w$.

The query complexity follows directly from the query complexity of **Test-level-linearity**, which is invoked at most $\log(n)$ times with the parameters $\epsilon$ and $\delta' = \delta/\log(n)$. ■

### 3.4 Wrapping it all up

Before proving Theorem 3.2 we prove a small claim that will assist us in the proof. In both the claim and the proof of the claim we describe a situation where none of the (probabilistic) sub-procedures used by **Test-width-2** fail. By "Procedures not failing" we mean, e.g., that if the BLR test accepts, then the function is indeed $\epsilon$-close to a linear function.

**Claim 3.4** *Assuming none of the sub-procedures used by it fail, at the end of the r'th round of* **Test-width-2** *, the function $f^{r-1}$ is $\epsilon'$-close to a W2 function of $f^r(x_1, \ldots, x_{t^r})$ and the variables $x_{t^r+1}, \ldots, x_{t^{r-1}}$.*

The relationship between $f^{r-1}$ and $f^r$ is demonstrated in Figure 6.

**Proof:** By Claim 3.3 we have that at the end of Step 2d of **Test-width-2** in the $r$'th round, the function $f^{r-1}$ is $(\epsilon'/3)$-close to a W2 function of $g^r(x_1, \ldots, x_j)$ and the variables $x_{j+1}, \ldots, x_{\tilde{t}^r}$. When $j = 1$ this suffices (as we set $f^r = g^r$). When $j > 1$ we have by Claim 3.1 that at Step 2(e)iv $f^r$ is set to a function $\epsilon'/3$-close to a W2 function of $g^r$ and of the variable $x_{\tilde{t}^r}$. This function is surely a W2 function and thus $f^{r-1}$ is $2\epsilon'/3$-close to a W2 function of $f^r(x_1, \ldots, x_{t^r})$ and the variables $x_{t^r+1}, \ldots, x_{t^{r-1}}$, as required. ■

**Proof of Theorem 3.2:** We prove Theorem 3.2 in three stages. We first show the correctness of the algorithm assuming that none of the probabilistic procedures it performed erred in any way. We follow this by bounding the probability of error for the different probabilistic procedures, and finally, we analyze the query complexity, concluding the proof.

Correctness assuming the success of sub-tests involves proving the following:

1. **Completeness**: **Test-width-2** , given oracle access to a function computable by a width-2 OBDD, accepts.
2. **Soundness**: **Test-width-2** , given oracle access to a function $\epsilon$-far from any function computable by a width-2 OBDD, rejects with probability at least 2/3.

Proof of the completeness condition is straightforward: Rejection by **Test-width-2** occurs only when 3 different $i$-equivalence classes are detected. By Corollary 3.1 this never happens in a function computable by a width-2 OBDD. As **Test-width-2** always terminates either by accepting a function or rejecting it, the completeness condition holds.

We prove the soundness condition by proving the contrapositive - any function that passes the tester with probability greater than 1/3 is $\epsilon$-close to a function computable by a width-2 OBDD. To this end we assume that all the sub-procedures performed by **Test-width-2** succeed and show that in such a case any function passing the test is, indeed, $\epsilon$-close to a function computable by a width-2 OBDD. We later prove that the cumulative probability of the "sub-procedures" failing is at most 1/3, thus ensuring that **Test-width-2** is indeed a one-sided error property testing algorithm.

We define the function $\alpha^0$ to be $f$. We next construct for every round $r$ of the algorithm a function $\alpha^r$ that has the following properties:

1. The function $\alpha^r$ is close to the function $\alpha^{r-1}$. In particular $d(\alpha^r, \alpha^{r-1}) \leq \epsilon' = \epsilon/(4\log(1/\epsilon))$.
2. The function $\alpha^r$ is a W2 function of $f^r(x_1, \ldots, x_{t^r})$ and of $x_{t^r+1}, \ldots, x_n$, and has at least $r-1$ blocking variables. The W2 function that accepts as input the values $f^r(x_1, \ldots, x_{t^r})$ and $x_{t^r+1}, \ldots, x_n$ is denoted $\beta^r$.

We construct $\alpha^r$ based on $\alpha^{r-1}$ as follows: By Claim 3.4, at the end of the $r$'th round the function $f^{r-1}$ is $\epsilon'$-close to a W2 function, which we denote $\psi^r$, of $f^r(x_1, \ldots, x_{t^r})$ and the variables $x_{t^r+1}, \ldots, x_{t^{r-1}}$. Let

$$\alpha^r = \beta^{r-1}(\psi^r(f^r(x_1, \ldots, x_{t^r}), x_{t^r+1}, \ldots, x_{t^{r-1}}), x_{t^{r-1}+1}, \ldots, x_n) \ .$$

As we wish to view $\alpha^r$ as equivalent to $\beta^r(f^r(x_1, \ldots, x_{t^r}), x_{t^r+1})$, we define $\beta^r$ accordingly (see Figure 6). We have that $d(\alpha^r, \alpha^{r-1}) \leq \epsilon'$, since $\alpha^r$ and $\alpha^{r-1}$ can only differ when

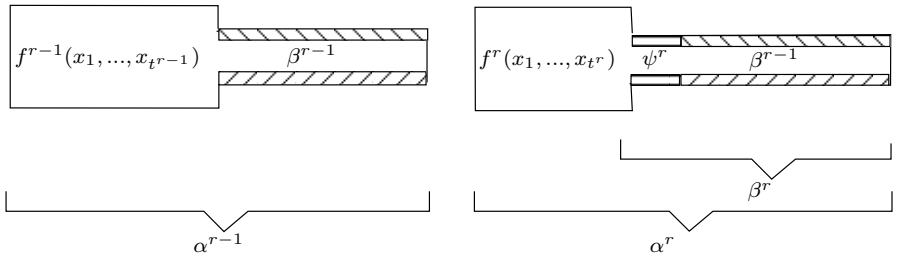


Figure 6: An illustration for the construction of $\alpha^r$ and $\beta^r$

$\psi^r(f^r(x_1,\ldots,x_{t^r}),x_{t^r+1},\ldots,x_{t^{r-1}}) \neq f^{r-1}(x_1,\ldots,x_{t^{r-1}})$. We note that $\beta^r$ is computable by a width-2 OBDD by a straightforward construction, and that unless $j=1$ on the $r$'th round, the new width-2 OBDD constructed by this procedure (that computes $\beta^r$) has one more blocking variable than the one on the $r-1$'th round.

Denoting the last round of **Test-width-2** as $s$ we now note that $\alpha^s$ is $(\epsilon/4)$-close to a function computable by a width-2 OBDD (assuming $f$ passed the test). There are three ways the test can terminate successfully:

1. The test reaches the $(\log(1/\epsilon)+2)$'th round. In such a case $\alpha^s$ is a W2 function (that accepts $f^s(x_1,\ldots,x_{t^s})$ and $x_{t^s+1},\ldots,x_n$ as input) with $\log(1/\epsilon)+2$ blocking variables in the OBDD computing it, and by Claim 2.5 is $(\epsilon/4)$-close to a function computable by a width-2 OBDD.
2. The test terminates because $t^s=1$. In such a case, by Claim 3.4, at the end of the $s$'th round the function $f^s$ is a function of 0 variables (a constant function), surely computable by a width-2 OBDD.
3. The test terminates because a single equivalence class was found in Step 2e. In such a case $f^s$ is $(\epsilon'/3)$-close to a constant function, as above.

Let $h$ be a function computable by a width-2 OBDD that's $(\epsilon/2)$-close to $\alpha^s$, and let $W2$ be the set of functions computable by width 2 OBDDs. We have

$$\begin{aligned} d(f,W2) &\leq d(f,h) && (4)\\ &\leq d(f,\alpha^1)+d(\alpha^1,\alpha^2)+\cdots+d(\alpha^{s-1},\alpha^s)+\epsilon/2 && (5)\\ &\leq \epsilon/2+(\log(1/\epsilon)+2)(\epsilon/(4\log(1/\epsilon))) && (6)\\ &\leq \epsilon && (7)\end{aligned}$$

as required.

The probability of any sub-test failing. The cumulative probability of any sub-procedure used by **Test-width-2** of failing during **Test-width-2**'s execution is at most 1/3. This is due to the fact that in each of at most $\log(1/\epsilon)+2$ rounds the algorithm performs two probabilistic sub-procedures, each with a probability of failure of at most $\frac{1}{6(\log(1/\epsilon)+2)}$. Using a simple union bound we get a total probability of failure of at most $2(\log(1/\epsilon)+2)\cdot\frac{1}{6(\log(1/\epsilon)+2)}=1/3$.

THE QUERY COMPLEXITY. It remains to analyze the query complexity of the algorithm. The tester repeats the outer loop at most $\log(1/\epsilon) + 2$ times, and performs queries in two Steps - 2a and 2e, where the number of queries in Step 2a is by far the larger and sums up to $\Theta(\log(n)(\log(\log(n)/\delta')/\epsilon''))$ where $\epsilon'' = \epsilon/(12\log(1/\epsilon))$ and $\delta' = \frac{1}{6(\log(1/\epsilon)+2)}$, giving us a total number of queries of

$$\Theta\left(\frac{\log(n)\log(\log(n)\log(1/\epsilon))\log(1/\epsilon)}{\epsilon}\right)$$

and the proof is complete. ■

# 4 Lower Bounds

In this section we give two lower bounds. An $\Omega(\log(n))$ lower bound on the query complexity in the fixed order case (where $\epsilon$ is a constant) and an $\Omega(n)$ lower bound in the case where the order of variables is not fixed. Both lower bounds hold for general, two-sided-error adaptive testers, where the first lower bound follows from an $\Omega(n)$ lower bound for non-adaptive testers (when the order is fixed). A lower bound for the fixed order case, which is based on the same construction as ours but is proved using a different technique, was recently given by Brody et.al. [5].

## 4.1 A Lower Bound For the Fixed Order Case

**Theorem 4.1** *Any two-sided error non-adaptive tester for computability by a width-2 OBDDs with the order of variables $x_1, \ldots, x_n$ must perform $\Omega(n)$ queries.*

As a corollary we get:

**Corollary 4.2** *Any two-sided error (possibly adaptive) tester for computability by a width-2 OBDD with the order of variables $x_1, \ldots, x_n$ must perform $\Omega(\log n)$ queries.*

In order to prove Theorem 4.1 we define two families of functions, $\mathcal{F}_1$ and $\mathcal{F}_2$. Each function in $\mathcal{F}_1$ can be computed by a width-2 OBDD, while each function in $\mathcal{F}_2$ is $\Omega(1)$-far from any function computable by a width-2 OBDD. However, it is not possible to distinguish with constant success probability between a uniformly selected function in $\mathcal{F}_1$ (which should be accepted with probability at least 2/3) and a uniformly selected function in $\mathcal{F}_2$ (which should be rejected with probability at least 2/3) by performing $o(n)$ non-adaptive queries.

In both families each function is defined by the choice of a coordinate $i^*$ such that $1 \leq i^* \leq n-3$ and $i^* = 1 \pmod 3$ (which we refer to as the *special* coordinate), and by a subset of coordinates $I \subseteq \{i^*+3, \ldots, n\}$. Given a choice of $i^*$ and $I$, in one family the corresponding function $f_1^{i^*,I}$ is

$$f_1^{i^*,I}(x) \;=\; (x_{i^*+1} \wedge x_{i^*+2}) + \sum_{i \in I} x_i \;, \tag{8}$$

where summation is in $GF(2)$, and in the second family the corresponding function is

$$f_2^{i^*,I}(x) \;=\; x_{i^*} + (x_{i^*+1} \wedge x_{i^*+2}) + \sum_{i \in I} x_i \;. \tag{9}$$

Clearly, each function $f_1^{i^*,I}(x)$ is computable by a width-2 OBDD. We next show that each function $f_2^{i^*,I}(x)$ is far from every width-2 OBDD.

**Claim 4.1** *For every $1 \leq i^* \leq n-3$ such that $i^* = 1 \pmod 3$ and for every $I^* \subseteq \{i^*+3, \ldots, n\}$, the function $f_2^{i^*,I}(x)$ is $\Omega(1)$-far from every function that is computable by a width-2 OBDD with the order of variables $x_1, \ldots, x_n$.*

We prove this claim using the fact that $x_{i^*}$ has influence 1 in $f_2^{i^*,I}$, and that all the variables have influence greater than or equal to 1/2 (all the variables have influence 1 except $x_{i^*+1}$ and $x_{i^*+2}$, that have influence 1/2).

**Proof:** Consider towards a contradiction a function $f$ that is computable by a width-2 OBDD with the order of variables $x_1, \ldots, x_n$ such that there exist $i^*$ and $I$ where $d(f, f_2^{i^*,I}) \leq 1/4$. We consider several cases:

1. If the influence of $x_{i^*}$ in $f$ is less than 1, then, since $f$ is computed by a width-2 OBDD, we get by Claim 2.3 and Claim 2.4 that the influence of $x_{i^*}$ in $f$ must be at most 1/2. This is the case because non-blocking variables do not reduce influence, and other variables reduce influence by at least half. If the influence of $x_{i^*}$ is indeed less than 1/2, then $f$ is at least 1/4-far from $f_2^{i^*,I}$ and we reach a contradiction.

2. Otherwise, as the influence of $x_{i^*}$ is 1, by Claim 2.3 and Claim 2.4, all of the variables in an OBDD computing $f$ are non-blocking (recall that $x_1, \ldots, x_{i^*-1}$ have no influence in $f$), and thus $f$ is a linear function. Again, $f$ must be at least 1/4-far from $f_2^{i^*,I}$, because the influence of $x_{i^*+1}$ in $f$ is 1 and in $f_2^{i^*,I}$ it is 1/2.

Thus, a contradiction is reached and the proof is complete. ■

We next show that for every (non-adaptive) choice of at most $q = n/c$ queries (for some sufficiently large constant $c$) the statistical difference between the distributions on answers that are induced by the two distributions on functions (uniform over $\mathcal{F}_1$ and uniform over $\mathcal{F}_2$, respectively), is a small constant. The claim will actually be a bit stronger: for every choice of $q$ queries we have that for all but at most $q$ settings of the index $i^*$, once we fix $i^*$ and select $I$ uniformly at random, the distribution on answers is identical for both families. This implies that for every choice of $q = n/c$ queries, the statistical difference between the two distributions on answers is at most $q/(n/3) = 3/c$.

We shall make use of the following definition.

**Definition 4.1** *Given a set $Y$ of $q$ queries, $Y = \{y^1, \ldots, y^q\}$ where $y^j \in \{0,1\}^n$, we say that a coordinate $1 \leq i^* \leq n-3$ is* suspicious *with respect to $Y$, if the following holds. There exists a subset $Y'$ of the queries such that when we take the sum of all queries in $Y'$ (mod 2), then the resulting vector $v = \bigoplus_{j \in Y} y^j$ satisfies: (1) $v_i = 0$ for every $i \geq i^*+3$; and (2) $v_i \neq 0$ for at least one $i \in \{i^*, i^*+1, i^*+2\}$.*

**Claim 4.2** *For any choice $Y$ of $q$ queries, the number of coordinates that are suspicious with respect to $Y$ is at most $q$.*

**Proof:** For each $i^*$ that is suspicious with respect to $Y$, let $z^{i^*}$ be the vector that results from summing the subset $Y'$ of queries that give evidence to the fact that $i^*$ is suspicious (if there is more than one such subset, then we take one arbitrarily). Thus $z^{i^*}$ consists of some arbitrary prefix of $i^*-1$ coordinates, it is then non-0 in one of the following 3 coordinates, and then it is all 0.

The main observation is that the different $z^{i^*}$'s (for the suspicious $i^*$'s) are linearly independent (recall that $i^* = 1 \pmod 3$). However, each is a linear combination of some subset of the $q$ queries, and the dimension of the subspace spanned by any $q$ queries is at most $q$. Therefore the number of such $z^{i^*}$'s cannot be more than $q$, implying the same upper bound on the number of suspicious coordinates. ■

Having upper-bounded the number of suspicious coordinates, we turn to the non-suspicious ones. The next notation will be useful. Let $Y' = \{y^1, \ldots, y^t\}$ be a fixed set of queries and let $L' = \ell^1, \ldots, \ell^t$ be a corresponding vector of answers. For any query $y^{t+1}$ and a given choice of $i^*$, let $p_1(Y', L', y^{t+1}, i^*)$, be the probability that the answer to $y^{t+1}$ is 1 if we select $f_1^{i^*,I}$ uniformly among all functions in $\mathcal{F}_1$ that are consistent with $L' = \ell^1, \ldots, \ell^t$ on $Y'$ and in which $i^*$ is their special coordinate. Define $p_2(Y', L', y^{t+1}, i^*)$ analogously, that is, when the function is drawn from $\mathcal{F}_2$.

**Claim 4.3** *Let $Y = \{y^1, \ldots, y^q\}$ be a fixed set of queries and, let $i^*$ be a fixed coordinate that is not suspicious with respect to $Y$. For any subset $Y' = \{y^1, \ldots, y^t\}$, $t < q$, and any setting of answers to the queries in $Y'$, denoted $L = \ell^1, \ldots, \ell^t$ we have that $p_1(Y', L', y^{t+1}, i^*) = p_2(Y', L', y^{t+1}, i^*)$.*

**Proof:** Each query $y^i$, $1 \leq i \leq t+1$ is of the form $x^i a^i b^i c^i v^i$, where $x^i \in \{0,1\}^{i^*-1}$, $a^i, b^i, c^i \in \{0,1\}$ and $v^i \in \{0,1\}^{n-(i^*+2)}$. We consider two cases: if $v^{t+1}$ is not linearly dependent on $v^1, \ldots, v^t$ then, since in both distributions the subset $I$ is selected uniformly, the new answer is equally probable to be 0 or 1. That is, in this case $p_1(Y', L', y^{t+1}, i^*) = p_2(Y', L', y^{t+1}, i^*) = 1/2$. On the other hand, if $v^{t+1}$ is linearly dependent on the previous $v^i$'s, then we'll show that in both distributions the answer to $y^{t+1}$ is determined by the previous queries and answers to be the same value (that is, either $p_1(Y', L', y^{t+1}, i^*) = p_2(Y', L', y^{t+1}, i^*) = 1$ or $p_1(Y', L', y^{t+1}, i^*) = p_2(Y', L', y^{t+1}, i^*) = 0$).

For the sake of notational simplicity, assume that $v^{t+1} = \sum_{i=1}^t v^i$ (though it can be the sum of any subset of the $v^i$'s). Since $i^*$ is not not suspicious, we know that it also holds that $a^{t+1} = \sum_{i=1}^t a^i$ (an analogous statement holds for $b^{t+1}$ and $c^{t+1}$, but we won't need to use it). By definition of the two function classes, we know that for every $1 \leq i \leq t$, $\ell^i = (b^i \wedge c^i) + w^1 v^i$ for some vector $w^1 \in \{0,1\}^{n-(i^*+2)}$, and that $\ell^i$ also equals $a^i + (b^i \wedge c^i) + w^2 v^i$ (for some vector $w^2 \in \{0,1\}^{n-(i^*+2)}$). Since $v^{t+1} = \sum_{i=1}^t v^i$, we get that for the first family, $\ell^{t+1}$ should be

$$
\begin{aligned}
(b^{t+1} \wedge c^{t+1}) + w^1 \sum_{i=1}^t v^i &= (b^{t+1} \wedge c^{t+1}) + \sum_{i=1}^t w^1 v^i && (10)\\
&= (b^{t+1} \wedge c^{t+1}) + \sum_{i=1}^t (\ell^i - (b^i \wedge c^i)) \,. && (11)
\end{aligned}
$$

For the second family it should be

$$
\begin{aligned}
a^{t+1} + (b^{t+1} \wedge c^{t+1}) + w^1 \sum_{i=1}^t v^i &= a^{t+1} + (b^{t+1} \wedge c^{t+1}) + \sum_{i=1}^t w^1 v^i && (12)\\
&= a^{t+1} + (b^{t+1} \wedge c^{t+1}) + \sum_{i=1}^t (\ell^i - ((b^i \wedge c^i) + a^i)) && (13)\\
&= \sum_{i=1}^t a^i + (b^{t+1} \wedge c^{t+1}) + \sum_{i=1}^t ((\ell^i - ((b^i \wedge c^i) + a^i)) \,, && (14)
\end{aligned}
$$

and we get the same value (recall that we are working over $GF(2)$ so that $+$ and $-$ are the same). ■

The proof of Theorem 4.1 follows by combining Claims 4.1–4.3.

## 4.2 A Lower Bound for the Non-Fixed Order Case

In this subsection we prove the following theorem.

**Theorem 4.3** *Any two-sided error (possibly adaptive) tester for computability by a width-2 OBDDs when the order of the variables is not fixed must perform $\Omega(n)$ queries.*

We begin by establishing several facts about width-2 OBDDs when the order of variables is not fixed.

**Definition 4.2** *For a given order of the variables $z_1, \ldots, z_n$, a set of consecutive variables $z_i, \ldots, z_j$ is said to be a* series *of variables if they are all of the same type.*

We now give an alternative definition of a blocking variable, a definition that relates to a function $f$ and not to a particular OBDD computing it. We show this definition is equivalent to our original one, and relate it to the notion of series. We denote by $f_{x_i=\sigma}$ the function $f$ with the variable $x_i$ restricted to the value $\sigma$. In a similar manner, when restricting a set of variables $S$ to a value $\sigma$ we will use the notation $f_{S=\sigma}$.

**Definition 4.3** *A variable $x_i$ is* blocking *a variable $x_j$ with respect to a function $f$ if $x_j$ has no influence in $f_{x_i=\sigma}$ (for some $\sigma \in \{0,1\}$), but has influence in $f$. A variable $x_i$ that blocks some $x_j$ will be called a* blocking variable.

Note that the property of being a blocking variable relates to the function $f$ and not to a specific representation of this function (e.g., as a width-2 OBDD). However, blocking variables with respect to a function $f$ are AND variables or OR variables in any width-2 OBDD that computes $f$. The next claim follows from the definition of blocking variables (as well as AND and OR variables).

**Claim 4.4** *If $x_i$ is an* AND *variable or an* OR *variable in a width-2 OBDD that computes a function $f$, then $x_i$ is a blocking variable with respect to $f$ and blocks all variables in series preceding it, as well as the variables in the same series it belongs to.*

The other direction holds as well.

**Claim 4.5** *Let $x_i$ be a variable blocking $x_j$ in $f$. In any width-2 OBDD computing $f$ it holds that $x_i$ is either an* AND *variable or an* OR *variable. Furthermore, $x_i$ either appears in the same series as $x_j$ or in a series following it.*

**Proof:** We first show that $x_i$ either appears in the same series as $x_j$ or in a series following it: Let $x_j$ appear as the $m$'th variable in an OBDD computing $f$. Assume towards a contradiction that $x_i$ precedes $x_j$ in the variable order. There are two cases:

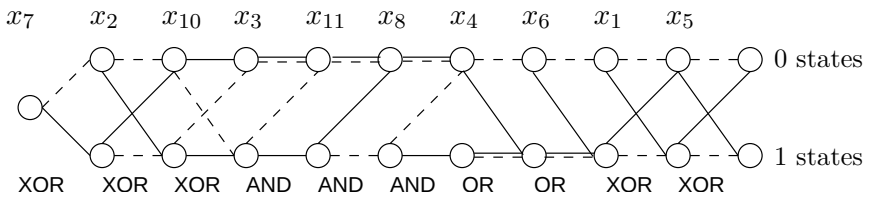


Figure 7: An example of a width-2-OBDD and its partition into series and sections. The dashed lines represent 0 transitions and the solid lines represent 1 transitions.

1. If $x_j$ is a XOR variable it will clearly have influence in $f_{m+1}$ for any assignment $x_i = \sigma$, and is hence not blocked by $x_i$.

2. Otherwise, let the section that $x_j$ belongs to begin with the variable $z_\ell$. Let $f' = f_\ell$ and note that if $f'_{x_i=\sigma}(z_1, \ldots, z_{\ell-1})$ is not constant (for some $\sigma \in \{0,1\}$), then $x_i$ cannot block $x_j$. Thus, there can be no XOR variables separating the series containing $x_i$ and $x_j$. Assume without loss of generality that $x_j$ is an AND variable. This means that the series immediately preceding $x_j$'s is an OR section. If $x_i$ is not in the OR section immediately preceding $x_j$ than, when a literal in that section is true, $x_j$ can certainly have influence and thus $x_i$ cannot block it. If $x_i$ is in the series preceding the one containing $x_j$, it's easy to see that $x_j$ can have influence whether $x_i$ is restricted to a true or false value.

Thus, $x_i$ cannot precede $x_j$. It remains to note that given that $x_i$ blocks $x_j$, it cannot be a XOR variable in the same series as $x_j$ or following it. ■

To prove Theorem 4.3 we use a reduction from a communication complexity problem.[4] This approach for proving property testing lower bounds was pioneered by Blais et al. [3]. They show how communication complexity problems can be reduced to property testing problems, implying that the number of bits that two parties must communicate in order to solve certain communication complexity problems gives a lower bound on the number of queries required to solve corresponding testing problems.

Our reduction actually combines several reductions. Specifically, we give a reduction from the Set-Disjointness communication complexity problem, which we define shortly, to the problem of deciding whether two linear functions share some of their influential variables. This reduction is inspired by a similar reduction introduced by Blais et al. [3]. Having done this we show that the problem of deciding whether two linear functions share any influential variables is essentially equivalent to deciding whether the influential variables in one linear function are a subset of those in another[5]. Finally, we show that a property testing algorithm that distinguishes between functions computable by width-2 OBDDs (for some order of the variables) and functions far from all functions in this family, can be used to decide whether the influential variables in one linear function are a subset of those in another.

We start with a few definitions.

[4]Here we give only the definitions necessary for our purposes. For an excellent textbook, see [17].

[5]In fact we only show this in one direction, which we require. The other direction follows similar lines.

**Definition 4.4** *For a linear function $f$ we denote the set of variables that have influence of 1 in $f$ by $S(f)$.*

**Definition 4.5** *Two linear functions $f$ and $g$ are said to* intersect *if $|S(f) \cap S(g)| \neq 0$. Deciding whether two linear functions intersect is referred to as the* Linear Intersection *problem.*

**Definition 4.6** *A linear function $g$ is said to* contain *a linear function $f$ if $S(f) \subseteq S(g)$. We denote this by $f \subseteq g$. Deciding whether one linear function contains another is referred to as the* Linear Containment *problem.*

**Definition 4.7** *For $x \in \{0,1\}^n$ we define the linear function $\chi_x : \{0,1\}^n \to \{0,1\}$ as: $\chi_x(y) = \bigoplus_{i=1}^{n}(x_i \wedge y_i)$.*

**Claim 4.6** *Let $T$ be an algorithm that receives oracle access to two linear functions $f, g$, performs $q(n)$ queries, and solves the Linear Containment problem with probability of success at least $2/3$. There exists an algorithm that solves the Linear Intersection problem and performs $O(q(n))$ queries.*

**Proof:** Consider the variables $S(f \oplus g)$. These variables are exactly those that are either in $S(f)$ or in $S(g)$ but not in both. Thus, to determine whether $|S(f) \cap S(g)| \neq 0$ we can emulate $T$ on the functions $f$ and $f \oplus g$, returning *true* where $T$ returns *false* and vice versa. This is correct because $f \subseteq f \oplus g$, if and only if $S(f)$ and $S(g)$ do not intersect. ■

**Definition 4.8** *Let Alice and Bob be two parties where Alice is given an input $x = x_1, \ldots, x_n$ and Bob is given an input $y = y_1, \ldots, y_n$. Both parties have unlimited computation time, and each party may access a shared string of random bits. They are interested in computing some function $h(x, y)$, and to this end they may send each other messages (possibly in several rounds). The (randomized)* communication complexity *of computing $h$ is the minimum number of bits that Alice and Bob need to communicate so as to ensure that for every $x, y$, they compute $h(x, y)$ with probability at least $2/3$.*

*In the* Set Disjointness *problem Alice and Bob should compute $h(x, y) \stackrel{\text{def}}{=} \bigvee_{i=1}^{n}(x_i \wedge y_i)$. That is, they should determine (with success probability at least $2/3$) whether there exists an index $i$ such that both $x_i = 1$ and $y_i = 1$.*

**Theorem 4.4 ([23])** *The randomized communication complexity of the Set Disjointness problem is $\Omega(n)$.*

**Claim 4.7** *Any algorithm that receives oracle access to two linear functions $f$ and $g$, and solves the Linear Intersection problem with success probability at least $2/3$ must perform $\Omega(n)$ queries.*

**Proof:** Let $T$ be an algorithm for the Linear Intersection problem. We show that $T$ can be emulated by two communicating parties to solve the Set Disjointness problem, where the number of bits communicated is equal to the number of queries performed by $T$. for a similar problem. Alice is given the input $x$ and Bob is given the input $y$. They emulate $T$ on the input $(\chi_x, \chi_y)$ (with coin tosses determined according to the shared random string), where the results of queries to $\chi_x$ are calculated by Alice and communicated to Bob, and results of queries to $\chi_y$ are calculated by Bob and

communicated to Alice. This returns the correct result with probability 2/3 as $|S(\chi_x) \cap S(\chi_y)| > 0$ only if $\bigvee_{i=1}^{n}(x_i \wedge y_i) = 1$. The number of bits communicated is the same as the number of queries performed, as a bit is sent to communicate the result of each query and no other bits are sent. The claim follows. ■

We now describe two families of functions, $\mathcal{F}$ and $\mathcal{G}$, such that functions in $\mathcal{F}$ are far from all functions computable by width-2 OBDDs, and functions in $\mathcal{G}$ are computable by width-2 OBDDs. We shall use the following notation: for a set of variables $U$, we let $\bigoplus U$ be a shorthand for $\bigoplus_{x_i \in U} x_i$. The family $\mathcal{F}$ is composed of functions that are computed as follows. Every function $f \in \mathcal{F}$ has three disjoint and non-empty sets of variables, $U, V, W$ where $x_1 \notin U \cup V \cup W$. We compute $f$ as follows: If $x_1 = 0$ then $f = \bigoplus U \oplus \bigoplus V$. Otherwise, $f = \bigoplus U \oplus \bigoplus W$. For $g \in \mathcal{G}$ we have only two (disjoint, non-empty) sets of variables, $U$ and $V$. If $x_1 = 1$ then $g = \bigoplus U \oplus \bigoplus V$. Otherwise, $g = \bigoplus U$.

**Claim 4.8** *Every function $g \in \mathcal{G}$ can be computed by a width-2 OBDD.*

**Proof:** The OBDD that computes $g$ is constructed as follows. We first have a XOR-series containing the variables in $V$, followed by $x_1$ as an AND variable. Finally, we have a XOR-series with the variables in $U$. Verifying this computes $g$ is straightforward. ■

**Claim 4.9** *Every $f \in \mathcal{F}$ is 1/8-far from every function computable by a width-2 OBDD, regardless of the order of variables.*

**Proof:** Let us fix a function $f \in \mathcal{F}$ and consider a particular width-2 OBDD $M$. We show that the function computed by $M$, which we denote $f^M$, is 1/8-far from $f$. To do this we first consider the influence of different variables in $f$:

1. All the variables $u \in U$ have influence 1.
2. All the variables $v \in V$ and $w \in W$ have influence 1/2, as they have influence 1 in $f_{x_1=\sigma}$ and influence 0 in $f_{x_1=\bar{\sigma}}$ (where the identity of $\sigma$ depends on whether we are discussing $v$ or $w$).
3. The variable $x_1$ has influence 1/2, as it effects the value of the function on those inputs where $\bigoplus V \neq \bigoplus W$.

Now consider toward a contradiction a function $f^M$ computable by a width-2 OBDD $M$ that is 1/8-close to $f$. We first establish several facts about the variables and their order in $M$:

1. No variable blocks the variable $x_1$. Otherwise, for such a variable $v$ there would be an assignment $v = \sigma$ where $x_1$ has no influence. But in $f$ we have an influence of 1/2 for $x_1$ for any assignment $v = \sigma$, and this would mean $f^M$ would be more than 1/8-far from $f$. It follows that $x_1$ appears in $M$ after all blocking variables (aside, perhaps, of itself).
2. The variable $x_1$ is not a XOR variable in $M$. Otherwise, as it is not blocked by any other variable (which we know by Fact 1), it has influence 1 in $F^M$. But in $f$ we have an influence of 1/2 for $x_1$, and this would mean $f^M$ would be more than 1/8-far from $f$. It follows (by Claim 4.4) that $x_1$ is a blocking variable.

3. For the order of variables in $M$ all the variables $u \in U$ are in the last XOR series. If this were not the case, by Fact 1 there would exist $u \in U$ with influence at most 1/2 (because it would come before $x_1$), and $f^M$ would be more than 1/8-far from $f$.

4. The variables $v \in V$ appear before $x_1$ in the OBDD $M$. If this were not the case, as no variable blocks $x_1$, by Claim 4.5 they would be XOR variables that are not blocked by any variable, and would have influence 1. As they have influence 1/2 in $f$ this would mean $f^M$ would be more than 1/8-far from $f$. The same holds for variables $w \in W$.

Given the facts above it remains to consider the case where the variables in $V$ and $W$ all come before the variable $x_1$ in $M$'s order of variables. As $x_1$ is a blocking variable in $M$, it has a blocking value $\sigma$. It follows that in $f^M_{x_1=\sigma}$ all the variables $u \in U$ and $w \in W$ have influence 0. However, by the definition of $f$, either $v \in V$ or $w \in W$ have an influence of 1 in $f_{x_1=\sigma}$, and thus a contradiction is reached and the claim holds. ■

We now give a reduction $R$ that maps pairs of linear functions $(f, g)$ of $n$ variables, to a function $h$ as follows. Each pair is mapped to a function $h : \{0,1\}^{n+1} \to \{0,1\}$, such that $h(0, x_2, \ldots, x_{n+1}) = f(x_2, \ldots, x_{n+1})$ and $h(1, x_2, \ldots, x_{n+1}) = g(x_2, \ldots, x_{n+1})$. Note that if the variables in $f$ are a subset of the variables in $g$, then $h \in \mathcal{G}$. Otherwise, $h \in \mathcal{F}$.

**Proof of Theorem 4.3:** Theorem 4.3 follows from Claims 4.6, 4.7, 4.8 and 4.9, as well as Theorem 4.4. In particular, let $T$ be a property testing algorithm for computability by width-2 OBDDs where the order of the variables is not fixed. By Claim 4.6 it holds that $\chi_x \subseteq \chi_x \oplus \chi_y$ if and only if $\chi_x$ and $\chi_y$ intersect. Thus, by Claims 4.8 and 4.9 the function $R(\chi_x, \chi_x \oplus \chi_y)$ is computable by a width-2 OBDD if $\chi_x$ and $\chi_y$ intersect, and far from every function computable by a width-2 OBDD otherwise. Running $T$ on $R(\chi_x, \chi_x \oplus \chi_y)$ solves the Linear Intersection problem for any $\chi_x, \chi_y$, and thus by Claim 4.7 it holds that $T$ must perform $\Omega(n)$ queries. ■

**Acknowledgements**

We would like to thank the anonymous reviewers of this paper for their helpful comments. We would also like to thank Joshua Brody, Kevin Matulef, and Chenggang Wu, for noting an error in a previous version of one of our lower bounds, and for bringing to our attention their paper [5], as well as [3].

# References

[1] N. Alon, M. Krivelevich, T. Kaufman, S. Litsyn, and D. Ron. Testing Reed-Muller codes. *IEEE Transactions on Information Theory*, 51(11):4032–4038, 2005.

[2] F. Bergadano, N. Bshouty, C. Tamon, and S. Varricchio. On learning branching programs and small depth circuits. In *Proceedings of the Tenth Annual ACM Conference on Computational Learning Theorey (COLT)*, pages 150–161, 1997.

[3] E. Blais, J. Brody, and K. Matulef. Property testing lower bounds via communication complexity. To appear in the 26th Conference on Computational Complexity (CCC), 2011.

[4] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of the ACM*, 47:549–595, 1993.

[5] J. Brody, K. Matulef, and C. Wu. Lower bounds for testing computability by small width OBDDs. Manuscript, 2011.

[6] N. Bshouty, C. Tamon, and D. Wilson. On learning width two branching programs. *Information Processing Letters*, 65:217–222, 1998.

[7] I. Diakonikolas, H. K. Lee, K. Matulef, K. Onak, R. Rubinfeld, R. A. Servedio, and A. Wan. Testing for concise representations. In *Proceedings of the Forty-Eighth Annual Symposium on Foundations of Computer Science (FOCS)*, pages 549–557, 2007.

[8] F. Ergün, R. S. Kumar, and R. Rubinfeld. On learning bounded-width branching programs. In *Proceedings of the Eighth Annual ACM Conference on Computational Learning Theorey (COLT)*, pages 361–368, 1995.

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

[10] E. Fischer, G. Kindler, D. Ron, S. Safra, and S. Samorodnitsky. Testing juntas. *Journal of Computer and System Sciences*, 68(4):753–787, 2004.

[11] R. Gavalda and D. Guijarro. Learning ordered binary decision diagrams. In *Proceedings of the Sixth Interntional Conference on Algorithmic Learning Theory (ALT)*, pages 228–238, 1995.

[12] O. Goldreich. Combinatorial property testing - a survey. In *Randomization Methods in Algorithm Design*, pages 45–60, 1998.

[13] O. Goldreich. On testing computability by small width OBDDs. In *Proceedings of the Fourteenth International Workshop on Randomization and Computation (RANDOM)*, pages 574–587, 2010.

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

[15] C. S. Jutla, A. C. Patthak, A. Rudra, and D. Zuckerman. Testing low-degree polynomials over prime fields. In *Proceedings of the Forty-Fifth Annual Symposium on Foundations of Computer Science (FOCS)*, 2004.

[16] M. Kearns and D. Ron. Testing problems with sub-learning sample complexity. *Journal of Computer and System Sciences*, 61(3):428–456, 2000.

[17] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[18] A. Nakamura. Query learning of bounded-width OBDDs. *Theoretical Computer Science*, 241:83–114, 2000.

[19] A. Nakamura. An efficient query learning algorithm for OBDDs. *Information and Computation*, 201:178–198, 2005.

[20] I. Newman. Testing membership in languages that have small width branching programs. *SIAM Journal on Computing*, 31(5):1557–1570, 2002.

[21] M. Parnas, D. Ron, and A. Samorodnitsky. Testing basic boolean formulae. *SIAM Journal on Discrete Math*, 16(1):20–46, 2002.

[22] V. RagHavan and D. Wilkins. Learning branching programs with queries. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theorey (COLT)*, pages 27–36, 1993.

[23] A. A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106(2):385–390, 1992.

[24] D. Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008.

[25] D. Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2010.

[26] D. Ron and G. Tsur. Testing computability by width two OBDDs. In *Proceedings of the Thirteenth International Workshop on Randomization and Computation (RANDOM)*, pages 686–699, 2009.

[27] D. Ron and G. Tsur. Testing computability by width-two obdds where the variable order is unknown. In *Proceedings of the Seventh International Conference on Algorithms and Complexity (CIAC)*, 2010.

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