Testing Read-Once Formula Satisfaction*

Eldar Fischer† Yonatan Goldhirsh‡ Oded Lachish§

May 9, 2016

Abstract

We study the query complexity of testing for properties defined by read once formulas, as instances of *massively parametrized properties*, and prove several testability and non-testability results. First we prove the testability of any property accepted by a Boolean read-once formula involving any bounded arity gates, with a number of queries exponential in ϵ, doubly exponential in the arity, and independent of all other parameters. When the gates are limited to being monotone, we prove that there is an *estimation* algorithm, that outputs an approximation of the distance of the input from satisfying the property. For formulas only involving And/Or gates, we provide a more efficient test whose query complexity is only quasipolynomial in ϵ. On the other hand, we show that such testability results do not hold in general for formulas over non-Boolean alphabets; specifically we construct a property defined by a read-once arity 2 (non-Boolean) formula over an alphabet of size 4, such that any 1/4-test for it requires a number of queries depending on the formula size. We also present such a formula over an alphabet of size 5 that additionally satisfies a strong monotonicity condition.

1 Introduction

Property Testing deals with randomized approximation algorithms that operate under low information situations. The definition of a property testing algorithm uses the following components: A set of *objects*, usually the set of strings Σ^* over some alphabet Σ; a notion of a single *query* to the input object $w = (w_1, \ldots, w_n) \in \Sigma^*$, which in our case would consist of either retrieving the length $|w|$ or the i'th letter w_i for any i specified by the algorithm; and finally a notion of *farness*, a normalized distance, which in our case will be the Hamming distance — $\texttt{farness}(w, v)$ is defined to be ∞ if $|w| \neq |v|$ and otherwise it is $|\{i : w_i \neq v_i\}|/|v|$.

Given a *property* P, that is a set of objects $P \subseteq \Sigma^*$, an integer q, and a farness parameter $\epsilon > 0$, an *ϵ-test for P with query complexity q* is an algorithm that is allowed access to an input object only through queries, and distinguishes between inputs that satisfy P and inputs that are ϵ-far from satisfying P (that is, inputs whose farness from any object of P is more than ϵ), while using at most q queries. By their nature the only possible testing algorithms are probabilistic, with either 1-sided or 2-sided error (1-sided error algorithms must accept objects from P with probability 1). Traditionally the query "what is $|w|$" is not counted towards the q query limit.

The ultimate goal of Property-Testing research is to classify properties according to their optimal ϵ-test query-complexity. In particular, a property whose optimal query complexity depends on ϵ alone and not on the length $|w|$ is called *testable*. In many (but not all) cases a "query-efficient" property test will also be efficient in other computational resources, such as running time (usually it will be the time it takes to retrieve a query multiplied by some function of the number of queries) and space complexity (outside the space used to store the input itself).

*Research supported in part by an ERC-2007-StG grant number 202405. A preliminary version of this work appeared in the Proceedings of 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2012).

†Technion – Israel Institute of Technology

‡Technion – Israel Institute of Technology

§Birkbeck, University of London

Property-Testing was first addressed by Blum, Luby and Rubinfeld [4], and most of its general notions were first formulated by Rubinfeld and Sudan [18], where the investigated properties are mostly of an algebraic nature, such as the property of a Boolean function being linear. The first excursion to combinatorial properties and the formal definition of testability were by Goldreich, Goldwasser and Ron [11]. Since then Property-Testing has attracted significant attention leading to many results. For surveys see [6], [10], [16], [17].

Many times *families* of properties are investigated rather than individual properties, and one way to express such families is through the use of parameters. For example, k-colorability (as investigated in [11]) has an integer parameter, and the more general partition properties investigated there have the sequence of density constraints as parameters. In early investigations the parameters were considered "constant" with regards to the query complexity bounds, which were allowed to depend on them arbitrarily. However, later investigations involved properties whose "parameter" has in fact a description size comparable to the input itself. Probably the earliest example of this is [14], where properties accepted by a general read-once oblivious branching program are investigated. In such a setting a general dependency of the query complexity on the parameter is inadmissible, and indeed in [14] the dependency is only on the maximum width of the branching program, which may be thought of as a complexity parameter of the stated problem.

A fitting name for such families of properties is *massively parametrized properties*. A good way to formalize this setting is to consider an input to be divided to two parts. One part is the *parameter*, the branching program in the example above, to which the testing algorithm is allowed full access without counting queries. The other part is the *tested input*, to which the algorithm is allowed only a limited number of queries as above. Also, in the definition of farness only changes to the tested input are allowed, and not to the parameter. In other words, two "inputs" that differ on the parameter part are considered to be ∞-far from each other. In this setting also other computational measures commonly come into play, such as the running time it takes to plan which queries will be made to the tested input.

Recently, a number of results concerning a massively parametrized setting (though at first not under this name) have appeared. See for example [12, 5, 7, 9] and the survey [15], as well as [2], where such an ϵ-test was used as part of a larger mechanism.

A central area of research in Property-Testing in general and Massively-Parametrized Testing in particular is to associate the query complexity of problems to their other measures of complexity. There are a number of results in this direction, to name some examples see [1, 14, 8]. In [3] the study of formula satisfiability was initiated. There it was shown that there exists a property that is defined by a 3-CNF formula and yet has a query complexity that is linear in the size of the input. This implies that knowing that a specific property is accepted by a 3-CNF formula does not give any information about its query complexity. In [13] it was shown that if a property is accepted by a read-twice CNF formula, then the property is testable. Here we continue this line of research.

In this paper we study the query complexity of properties that are accepted by read once formulas. These can be described as computational trees, with the tested input values at the leaves and logic gates at the other nodes, where for an input to be in the property a certain value must result when the calculation is concluded at the root.

Section 2 contains preliminaries. First we define the properties that we test, and then we introduce numerous definitions and lemmas about bringing the formulas whose satisfaction is tested into a normalized "basic form". These are important and in fact implicitly form a preprocessing part of our algorithms. Once the formula is put in a basic form, testing an assignment to the formula becomes manageable.

In Section 3 we show the testability of properties defined by formulas involving arbitrary Boolean gates of bounded arity. For such formulas involving only monotone gates, we provide an *estimation* algorithm in Section 4, that is an algorithm that not only tests for the property, but with high probability outputs a real number η such that the true farness of the tested input from the property is between $\eta - \epsilon$ and $\eta + \epsilon$. In Section 5 we show that when restricted to And/Or gates, we can provide a test whose query complexity is quasipolynomial in ϵ. We supply a brief analysis of the running times of the algorithms in Section 6.

On the other hand, we prove in Section 7 that these results can not be generalized to alphabets that have at least four different letters. We construct a formula utilizing only one (symmetric and binary) gate

type over an alphabet of size 4, such that the resulting property requires a number of queries depending on the formula (and input) size for a 1/4-test. We also prove that for the cost of one additional alphabet symbol, we can construct a non-testable explicitly monotone property (both the gate used and the acceptance condition are monotone).

Results such as these might have interesting applications in computational complexity. One interesting implication of the testability results here is that any read-once formula accepting an untestable Boolean property must use unbounded arity gates other than And/Or. By proving that properties defined by formulas of a simple form admit efficient property testers, one also paves a path for proving that certain properties cannot be defined by formulas of a simple form — just show that these properties cannot be efficiently testable. Since property testing lower bounds are in general easier to prove than computational complexity lower bounds, we hope that this can be a useful approach.

Acknowledgment

We thank Prajakta Nimbhorkar for the helpful discussion during the early stages of this work.

2 Preliminaries

We use $[k]$ to denote the set $\{1, \ldots, k\}$. A *digraph* G is a pair (V, E) such that $E \subseteq V \times V$. For every $v \in V$ we set $\texttt{out-deg}(v) = |\{u \in V \mid (v, u) \in E\}|$. A *path* is a tuple $(u_1, \ldots, u_k) \in |V|^k$ such that $(u_i, u_{i+1}) \in E$ for every $i \in [k-1]$. We say that a path $(u_1, \ldots, u_k)$ is *simple* if $u_1, \ldots, u_k$ are all distinct. The *length* of a path $(u_1, \ldots, u_k) \in |V|^k$ is $k - 1$. We say that there is a path from u to v if there exists a path $(u_1, \ldots, u_k)$ in G such that $u_1 = u$, and $u_k = v$. The *distance* from $u \in V$ to $v \in V$, denoted $\texttt{dist}(u, v)$, is the length of the shortest path from u to v if one exists and infinity otherwise.

We use the standard terminology for outward-directed rooted trees. A *rooted directed tree* is a tuple (V, E, r), where (V, E) is a digraph, $r \in V$ and for every $v \in V$ there is a unique path, simple or otherwise, from r to v. Let $u, v \in V$. If $\texttt{out-deg}(v) = 0$ then we call v a leaf. We say that u is an *ancestor* of v and v is a *descendant* of u if there is a path from u to v. We say that u is a *child* of v and v is a *parent* of u if $(v, u) \in E$, and set $\texttt{Children}(v) = \{w \in V \mid w \text{ is a child of } v\}$.

2.1 Formulas, evaluations and testing

With the terminology of rooted trees we now define our properties; first we define what is a formula and then we define what it means to satisfy one.

Definition 2.1 (Formula) *A* Formula *is a tuple $\Phi = (V, E, r, X, \kappa, B, \Sigma)$, where (V, E, r) is a rooted directed tree, Σ is an alphabet, X is a set of variables (later on they will take values in Σ), $B \subseteq \bigcup_{k<\infty}\{\Sigma^k \mapsto \Sigma\}$ a set of functions over Σ, and $\kappa : V \to B \cup X \cup \Sigma$ satisfies the following (we abuse notation somewhat by writing κ_v for $\kappa(v)$).*

- *For every leaf $v \in V$ we have that $\kappa_v \in X \cup \Sigma$.*
- *For every v that is not a leaf $\kappa_v \in B$ is a function whose arity is $|\texttt{Children}(v)|$.*

In the case where B contains functions that are not symmetric, we additionally assume that for every $v \in V$ there is an ordering of $\texttt{Children}(v) = (u_1, \ldots, u_k)$.

In the special case where Σ is the binary alphabet $\{0, 1\}$, we say that Φ is *Boolean*. Unless stated otherwise $\Sigma = \{0, 1\}$, in which case we shall omit Σ from the definition of formulas. A formula $\Phi = (V, E, r, X, \kappa, B, \Sigma)$ is called *read k-times* if for every $x \in X$ there are at most k vertices $v \in V$, where $\kappa_v \equiv x$. We call Φ a *read-once-formula* if it is read 1-times. A formula $\Phi = (V, E, r, X, \kappa, B, \Sigma)$ is called *k-ary* if the arity (number of children) of all its vertices is at most k. If a formula is 2-ary we then call it *binary*. A function $f : \{0, 1\}^n \to \{0, 1\}$ is *monotone* if whenever $x \in \{0, 1\}^n$ is such that $f(x) = 1$, then for every $y \in \{0, 1\}^n$ such that $x \leq y$ (coordinate-wise) we have $f(y) = 1$ as well. If all the functions in B are monotone then we say that Φ is (explicitly) *monotone*. We denote $|\Phi| = |X|$ and call it the

formula size (this makes sense for read-once formulas). Note that this is different from another notion of formula size that refers to the number of operators. In our case, the formula size is the size of its input.

Definition 2.2 (Sub-Formula) *Let $\Phi = (V, E, r, X, \kappa, B)$ be a formula and $u \in V$. The formula $\Phi_u = (V_u, E_u, u, X_u, \kappa, B)$, is such that $V_u \subseteq V$, with $v \in V_u$ if and only if $\texttt{dist}(u, v)$ is finite, and $(v, w) \in E_u$ if and only if $v, w \in V_u$ and $(v, w) \in E$. X_u is the set of all $\kappa_v \in X$ such that $v \in V_u$. If $u \neq r$ then we call Φ_u a* strict sub-formula. *We define $|\Phi_u|$ to be the number of variables in V_u, that is $|\Phi_u| = |X_u|$, and the* weight *of u with respect to its parent v is defined as $|\Phi_u|/|\Phi_v|$.*

Definition 2.3 (assignment to and evaluation of a formula) *An* assignment *σ to a formula $\Phi = (V, E, r, X, \kappa, B, \Sigma)$ is a mapping from X to Σ. The* evaluation *of Φ given σ, denoted (abusing notation somewhat) by $\sigma(\Phi)$, is defined as $\sigma(r)$ where $\sigma : V \to \Sigma$ is recursively defined as follows.*

- *If $\kappa_v \in \Sigma$ then $\sigma(v) = \kappa_v$.*
- *If $\kappa_v \in X$ then $\sigma(v) = \sigma(\kappa_v)$.*
- *Otherwise ($\kappa_v \in B$) denote the members of the set $\texttt{Children}(v)$ by $(u_1, \ldots, u_k)$ and set $\sigma(v) = \kappa_v(\sigma(u_1), \ldots, \sigma(u_k))$.*

Given an assignment $\sigma : X \to \Sigma$ and $u \in V$, we let σ_u denote its restriction to X_u, but whenever there is no confusion we just use σ also for the restriction (as an assignment to Φ_u).

For Boolean formulas, we set $SAT(\Phi = b)$ to be all the assignments σ to Φ such that $\sigma(\Phi) = b$. When $b = 1$ and we do not consider the case $b = 0$ in that context, we simply denote these assignments by $SAT(\Phi)$. If $\sigma \in SAT(\Phi)$ then we say that σ *satisfies* Φ. Let σ_1, σ_2 be assignments to Φ. We define $\texttt{farness}_\Phi(\sigma_1, \sigma_2)$ to be the relative Hamming distance between the two assignments. That is, $\texttt{farness}_\Phi(\sigma_1, \sigma_2) = |\{x \in X \mid \sigma_1(x) \neq \sigma_2(x)\}|/|\Phi|$. For every assignment σ to Φ and every subset S of assignments to Φ we define $\texttt{farness}_\Phi(\sigma, S) = \min\{\texttt{farness}_\Phi(\sigma, \sigma') \mid \sigma' \in S\}$. If $\texttt{farness}_\Phi(\sigma, S) > \epsilon$ then σ is *ϵ-far* from S and otherwise it is *ϵ-close* to S.

We now have the ingredients to define testing of assignments to formulas in a massively parametrized model. Namely, the formula Φ is the parameter that is known to the algorithm in advance and may not change, while the assignment $\sigma : X \to \Sigma$ must be queried using as few queries as possible, and farness is measured with respect to the fraction of alterations it requires.

Definition 2.4 [(ϵ, q)-test] *An* (ϵ, q)-test *for $SAT(\Phi)$ is a randomized algorithm $\mathcal{A}$ with free access to Φ, that given oracle access to an assignment σ to Φ operates as follows.*

- *$\mathcal{A}$ makes at most q queries to σ (where on a query $x \in X$ it receives σ_x as the answer).*
- *If $\sigma \in SAT(\Phi)$, then $\mathcal{A}$ accepts (returns 1) with probability at least $2/3$.*
- *If σ is ϵ-far from $SAT(\Phi)$, then $\mathcal{A}$ rejects (returns 0) with probability at least $2/3$. Recall that σ is ϵ-far from $SAT(\Phi)$ if its relative Hamming distance from every assignment in $SAT(\Phi)$ is at least ϵ.*

We say that $\mathcal{A}$ is non-adaptive *if its choice of queries is independent of their values (and may depend only on Φ). We say that $\mathcal{A}$ has* 1-sided error *if given oracle access to $\sigma \in SAT(\Phi)$, it accepts (returns 1) with probability 1. We say that $\mathcal{A}$ is an* (ϵ, q)-estimator *if it returns a value η such that with probability at least $2/3$, σ is both $(\eta + \epsilon)$-close and $(\eta - \epsilon)$-far from $SAT(\Phi)$.*

We can now summarize the contributions of the paper in the following theorem:

Theorem 2.5 (Main Theorem) *The following statements all hold for all constant k:*

- *For any read-once formula Φ where B is the set of all functions of arity at most k there exists a 1-sided (ϵ, q)-test for $SAT(\Phi)$ with $q = \exp(\text{poly}(\epsilon^{-1}))$ (Theorem 3.1).*
- *For any read-once formula Φ where B is the set of all monotone functions of arity at most k there exists an (ϵ, q)-estimator for $SAT(\Phi)$ with $q = \exp(\text{poly}(\epsilon^{-1}))$ (Theorem 4.1).*

- *For any read-once formula Φ where B is the set of all conjunctions and disjunctions of any arity there exists an (ϵ, q)-test for SAT(Φ) with $q = \epsilon^{O(\log \epsilon)}$ (Corollary 5.9 of Theorem 5.8).*
- *There exists an infinite family of 4-valued read-once formulas Φ, where B contains one binary function, and an appropriate $b \in \Sigma$, such that there is no non-adaptive (ϵ, q)-test for SAT($\Phi = b$) with $q = O(\texttt{depth}(\Phi))$, and no adaptive (ϵ, q)-test for SAT(Φ) with $q = O(\log(\texttt{depth}(\Phi)))$; there also exists such a family of 5-valued read-once formulas whose gates and acceptance condition are monotone with respect to a fixed order of the alphabet. (Theorem 7.8 and Theorem 7.14 respectively).*

Note that for the first two items, the degree of the polynomial is linear in k.

2.2 Basic formula simplification and handling

In the following, unless stated otherwise, our formulas will all be read-once and Boolean. For our algorithms to work, we will need a somewhat "canonical" form of such formulas. We say that two formulas Φ and Φ' are *equivalent* if $\sigma(\Phi) = \sigma(\Phi')$ for every assignment $\sigma : X \to \Sigma$.

Definition 2.6 *A 1-witness for a boolean function $f : \{0,1\}^n \to \{0,1\}$ is a subset of coordinates $W \subseteq [n]$ for which there exists an assignment $\sigma : W \to \{0,1\}$ such that for every $x \in \{0,1\}^n$ which agrees with σ (that is, where for all $i \in W$, we have that $x_i = \sigma(i)$) we have that $f(x) = 1$.*

Note that a function can have several 1-witnesses and that a 1-witness for a monotone function can always use the assignment σ that maps all coordinates to 1.

Definition 2.7 *The* mDNF *(monotone disjunctive normal form) of a monotone boolean function $f : \{0,1\}^n \to \{0,1\}$ is a set of terms T where each term in T is a 1-witness for f and for every $x \in \{0,1\}^n$, $f(x) = 1$ if and only if there exists a term $T_j \in T$ such that for all $i \in T_j$, we have that $x_i = 1$.*

Observation 2.8 *Any monotone boolean function $f : \{0,1\}^n \to \{0,1\}$ has a unique* mDNF *T.*

This is true, since this mDNF is the disjunction of f's minimal 1-witnesses.

Definition 2.9 *For $u \in V$, $v \in \texttt{Children}(u)$ is called* (a,b)-forceful *if $\sigma(v) = a$ implies $\sigma(u) = b$. v is* forceful *if it is (a,b)-forceful for some $a, b \in \{0,1\}$.*

For example, for $\wedge$ all children are $(0,0)$-forceful, and for $\vee$ all children are $(1,1)$-forceful. Forceful variables are variables that cause an "Or-like" or "And-like" behavior in the gate.

Definition 2.10 *A vertex $v \in V$ in a formula Φ is called* unforceable *if no child of v is forceful.*

Definition 2.11 *A vertex $v \in V$ in a formula Φ is called* trivial *if there exists a constant $c \in \{0,1\}$ such that for every assignment σ, $\sigma(v) = c$.*

Definition 2.12 (k-x-Basic formula) *A read-once formula Φ is k-x-*basic *if it is Boolean and all the functions in B are either:*

- *Negations,*
- *unforceable and of arity at least 2 and at most k,*
- *an $\wedge$ gate or an $\vee$ gate of arity at least 2.*

Additionally, Φ must satisfy the following:

- *Except for the leaves, there are no trivial vertices,*
- *negations may only have leaves as children,*

- *there is no leaf $v \in V$ such that $\kappa_v \in \{0,1\}$,*
- *no $\wedge$ is a child of a $\wedge$ and no $\vee$ is a child of a $\vee$,*
- *every variable may appear at most once in a leaf.*

The set of variables that appear negated will be denoted by $\neg X$.

Definition 2.13 (k-Basic formula) *A read-once formula Φ is a* k-basic *formula if it is k-x-basic, and furthermore all functions in B are also monotone. If B contains only conjunctions and disjunctions then we abbreviate and call the formula* basic.

Note that a k-Basic formula can obviously only be monotone.

Lemma 2.14 *Every read-once formula Φ with gates of arity at most k has an equivalent k-x-basic formula Φ', possibly over a different set of functions B.*

Proof. Suppose for some u that $v \in \texttt{Children}(u)$ is (a,b)-forceful. If $b = 1$ then κ_u can be replaced with an $\vee$ gate, where one input of the $\vee$ gate is v if $a = 1$ or the negation of v if $a = 0$, and the other input is the result of u when fixing $\sigma(\kappa_v) = 1 - a$. If $b = 0$ then κ_u can be replaced with an $\wedge$ gate, where one input of the $\wedge$ gate is v if $a = 0$ or the negation of v if $a = 1$, and the other input is the gate u when fixing $\sigma(\kappa_v) = 1 - a$. After performing this transformation sufficiently many times we have no forceable gates left except for $\wedge$ and $\vee$.

We will now eliminate $\neg$ gates. Any $\neg$ gate in the input or output of a gate which is not $\wedge$ or $\vee$ can be assimilated into the gate. Otherwise, a $\neg$ on the output of an $\vee$ gate can be replaced with an $\wedge$ gate with $\neg$'s on all of its inputs, according to De-Morgan's laws. Also by De-Morgan's laws, a $\neg$ on the output of an $\wedge$ gate can be replaced with an $\vee$ gate with $\neg$'s on all of its inputs.

Finally, any $\vee$ gates that have $\vee$ children can be merged with them, and the same goes for $\wedge$ gates. Now we have achieved an equivalent k-x-basic formula. ■

Observation 2.15 *Any formula Φ which is comprised of only monotone k-arity gates has an equivalent k-basic formula Φ'.*

This observation follows by inspecting the above proof, and noticing that monotone gates will never produce negations in the process described.

2.3 Observations about subformulas and farness

Definition 2.16 (heaviest child $h(v)$) *Let $\Phi = (V, E, r, X, \kappa, B)$ be a formula. For every $v \in V$ we define $h(v)$ to be v if $\texttt{Children}(v) = \emptyset$, and otherwise to be an arbitrarily selected vertex $u \in \texttt{Children}(v)$, such that $|\Phi_u| = \max\{|\Phi_w| \mid w \in \texttt{Children}(v)\}$.*

Definition 2.17 (vertex depth $\texttt{depth}_\Phi(v)$) *Let $\Phi = (V, E, r, X, \kappa, B)$ be a formula. For every $v \in V$ we define $\texttt{depth}_\Phi(v) = \texttt{dist}(r, v)$ and $\texttt{depth}(\Phi) = \max\{\texttt{depth}_\Phi(u) \mid u \in V\}$.*

Our first observation is that in "and" gates and similar situations, farness implies farness in subformulas, in a Markov's inequality-like fashion.

Observation 2.18 *Let $v \in V$ be a vertex with no trivial children, such that either $\kappa_v \equiv \vee$ and its output $b = 0$ or $\kappa_v \equiv \wedge$ and $b = 1$, and $\texttt{farness}(\sigma, SAT(\Phi_v = 1 - b)) \geq \epsilon$. For every $1 > \alpha > 0$ there exists $S \subseteq \texttt{Children}(v)$ such that $\sum_{s \in S} |\Phi_s| \geq \epsilon\alpha|\Phi|$ and $\texttt{farness}(\sigma, SAT(\Phi_w = 1 - b)) \geq \epsilon(1 - \alpha)$ for every $w \in S$. Furthermore, there exists a child $u \in \texttt{Children}(v)$ such that $\texttt{farness}(\sigma, SAT(\Phi_u = 1 - b)) \geq \epsilon$.*

Proof. Let T be the maximum subset of $\texttt{Children}(v)$ such that Φ_w is $\epsilon(1-\alpha)$-far from being evaluated to b for every $w \in T$. If $\sum_{t \in T} |\Phi_t| < \epsilon\alpha|\Phi|$ then the distance from having Φ_v evaluate to b is at most $\epsilon\alpha + \epsilon(1 - \alpha) < \epsilon$, since we only need to change the $\epsilon\alpha|\Phi_v|$ leaves that descend from the children in S and for the rest, we know that each of them is $\epsilon(1 - \alpha)$-close to satisfaction, and therefore only that

fraction of inputs in leaves that descend from children outside of S need to be changed. This contradicts the assumption.

For the second statement, note that if no such child exists then Φ_v is ϵ-close to being evaluated to b. ■

Observation 2.19 *Let $v \in V$ be a vertex with no trivial children, such that either $\kappa_v \equiv \vee$ and $b = 1$ or $\kappa_v \equiv \wedge$ and $b = 0$, and $\texttt{farness}(\sigma, SAT(\Phi_v = b)) \geq \epsilon$. For every child $u \in \texttt{Children}(v)$, $|\Phi_u| \geq |\Phi|\epsilon$ and $\texttt{farness}(\sigma, SAT(\Phi_u = b)) \geq \epsilon(1+\epsilon)$. Furthermore, $\epsilon \leq 1/2$, and for any $u \in \texttt{Children}(v) \setminus \{h(v)\}$, $\texttt{farness}(\sigma, SAT(\Phi_u = b)) \geq 2\epsilon$.*

Proof. First suppose that the weight of some child u is less than ϵ. In this case, setting u to b makes the formula Φ_v evaluate to b by changing less than an ϵ fraction of inputs, a contradiction.

Since there are at least two children, every child u is of weight at most $1 - \epsilon$, and since setting it to b would make Φ_v evaluate to b, it is at least $\epsilon(1+\epsilon)$-far from being evaluated to b.

For the last part, note that since $|\texttt{Children}(v)| > 1$, there exists $u \in \texttt{Children}(v)$ such that $|\Phi_u| \leq |\Phi_v|/2$. Thus every assignment to Φ_v is $1/2$-close to an assignment σ' by which Φ_v evaluates to b. Also note that any $u \in \texttt{Children}(v) \setminus \{h(v)\}$ satisfies $|\Phi_u| \leq |\Phi_v|/2$, and therefore if Φ_u were 2ϵ-close to being evaluated to b, Φ_v would be ϵ-close to being evaluated to b. ■

2.4 Heavy and Light Children in General Gates

We would like to pick the heaviest child of a general gate, same as we did above. The problem is that since we will use this for unforceable gates, we will simultaneously want the heaviest child or children not to be "too heavy". This brings us to the following definition.

Definition 2.20 *Given a k-x-basic formula Φ, a parameter ϵ and a vertex u, we let $\ell = \ell(u, \epsilon)$ be the smallest integer such that the size of the ℓ'th largest child of u is less than $|\Phi|(4k/\epsilon)^{-\ell}$ if such an integer exists, and set $\ell = k + 1$ otherwise. The* heavy *children of u are the $\ell - 1$ largest children of u, and the rest of the children of u are its* light *children.*

Note that if there is a really big child, then σ is close to both $SAT(\Phi_v = 1)$ and $SAT(\Phi_v = 0)$. More formally:

Lemma 2.21 *If an unforceable vertex v with no trivial children has a child u such that $|\Phi_v|(1-\epsilon) \leq |\Phi_u|$, then σ is both ϵ-close to $SAT(\Phi_v = 1)$ and ϵ-close to $SAT(\Phi_v = 0)$.*

Proof. The child is unforceful, and therefore it is possible to change the remaining children to obtain any output value. ■

Observation 2.22 *If for a vertex u with no trivial children, $\kappa_u \not\equiv \wedge$, $\kappa_u \not\equiv \vee$, $\kappa_u \notin X$ and σ is ϵ-far from $SAT(\Phi_u = b)$, then it must have at least two heavy children.*

Proof. By the definition of ℓ, if there is just one heavy child, then $\ell = 2$ and the total weight of the light children is strictly smaller than ϵ. Therefore by Lemma 2.21 there must be more than one heavy child, as otherwise the gate is ϵ-close to both 0 and 1. ■

3 Upper Bound for General Bounded Arity Formula

Algorithm 1 tests whether the input is ϵ-close to having output b with 1-sided error, and also receives a confidence parameter δ. The explicit confidence parameter makes the inductive arguments easier and clearer. The algorithm operates by recursively checking the conditions in Observations 2.18 and 2.19.

Theorem 3.1 *Algorithm 1$(\Phi, \epsilon, \delta, \sigma)$ always accepts any input that satisfies the read-once formula Φ, and rejects any input far from satisfying Φ with probability at least $1 - \delta$. Its query complexity (treating k and δ as constant) is always $O(\exp(poly(\epsilon^{-1})))$.*

Proof. Follows from Lemma 3.4, Lemma 3.5 and Lemma 3.3 (in that order) below. ■

Algorithm 1 Test satisfiability of read-once formula

Input: read-once k-x-basic formula $\Phi = (V, E, r, X, \kappa)$, parameters $\epsilon, \delta > 0, b \in \{0, 1\}$, oracle to σ.
Output: "true" or "false".

1: **if** $\epsilon > 1$ **then return** "true"
2: **if** $\kappa_r \in X$ **then return** the truth value of $\sigma(r) = b$
3: **if** $\kappa_r \in \neg X$ **then return** the truth value of $\sigma(r) = 1 - b$
4: **if** ($\kappa_r \equiv \wedge$ and $b = 1$) or ($\kappa_r \equiv \vee$ and $b = 0$) **then**
5: $y \longleftarrow$ "true"
6: **for** $i = 1$ to $l = 32(8k/\epsilon)^k \epsilon^{-1} \log(\delta^{-1})$ **do**
7: $u \longleftarrow$ a vertex in $\texttt{Children}(r)$ selected independently at random, where the probability that $w \in \texttt{Children}(r)$ is selected is $|\Phi_w|/|\Phi|$
8: $y \longleftarrow y \wedge$ Algorithm $1(\Phi_u, (\epsilon(1 - (8k/\epsilon)^{-k}/16)), \sigma, \delta/2, b)$
9: **end for**
10: **return** y
11: **end if**
12: **if** ($\kappa_r \equiv \wedge$ and $b = 0$) or ($\kappa_r \equiv \vee$ and $b = 1$) **then**
13: **if** there exists a child of weight less than ϵ **then return** "true"
14: $y \longleftarrow$ "false"
15: **for all** $u \in \texttt{Children}(r)$ **do** $y \longleftarrow y \vee$ Algorithm $1(\Phi_u, (\epsilon(1 + \epsilon)), \sigma, \epsilon\delta/2, b)$
16: **return** y
17: **end if**
18: **if** there is a child of weight at least $1 - \epsilon$ **then return** "true"
19: **for all** $u \in \texttt{Children}(r)$ **do**
20: $y_u^0 \longleftarrow$ Algorithm $1(\Phi_u, (\epsilon(1 + (4k/\epsilon)^{-k})), \sigma, \delta/2k, 0)$
21: $y_u^1 \longleftarrow$ Algorithm $1(\Phi_u, (\epsilon(1 + (4k/\epsilon)^{-k})), \sigma, \delta/2k, 1)$
22: **end for**
23: **if** there exists $x \in \{0, 1\}^k$ such that κ_r on x evaluates to b and for all $u \in \texttt{Children}(r)$ we have $y_u^{x_u}$ equal to "true" **then**
24: **return** "true"
25: **else**
26: **return** "false"
27: **end if**

Lemma 3.2 *The depth of recursion in Algorithm 1 is at most* $16(8k/\epsilon)^k \log(\epsilon^{-1})$.

Proof. If $\epsilon > 1$ then the condition in Line 1 is satisfied and the algorithm returns without any recursion.

All recursive calls occur in Lines 8, 15, 20 and 21. Since Φ is k-x-basic, any call with a subformula whose root is labeled by $\wedge$ results in calls to subformulas, each with a root labeled either by $\vee$ or an unforceable gate, and with the same b value (this is crucial since the b value for which $\wedge$ recurses with a smaller ϵ is the b value for which $\vee$ recurses with a bigger ϵ, and vice-versa). Similarly, any call with a subformula whose root is labeled by $\vee$ results in calls to subformulas, each with a root labeled either by $\wedge$ or an unforceable gate, and with the same b value.

Therefore, in two consecutive recursive calls, there are three options:

1. The first call is made with farness parameter $\epsilon(1 + \epsilon) \geq \epsilon' \geq \epsilon(1 + (4k/\epsilon)^{-k})$ and the second with $\epsilon'' = \epsilon'(1 - (8k/\epsilon')^{-k}/16)$. In this case in two consecutive calls the farness parameter increases by at least $(1 + (4k/\epsilon)^{-k})(1 - (8k/\epsilon(1 + \epsilon))^{-k}/16) \geq (1 + (4k/\epsilon)^{-k}/8)$.

2. The first call is made with farness parameter $\epsilon' = \epsilon(1 - (8k/\epsilon)^{-k}/16)$ and the second with $\epsilon'' \geq \epsilon'(1 + (4k/\epsilon')^{-k})$. In this case in two consecutive calls the farness parameter increases by at least $\epsilon(1 - (8k/\epsilon)^{-k}/16)(1 + (4k/\epsilon(1 - (8k/\epsilon)^{-k}/16))^{-k}) \geq (1 + (8k/\epsilon)^{-k}/8)$.

3. The first call is made with farness parameter $\epsilon' \geq \epsilon(1+(4k/\epsilon)^{-k})$ and the second with $\epsilon'' \geq \epsilon'(1+(4k/\epsilon')^{-k})$. In this case two consecutive in calls the farness parameter increases by at least $(1+(4k/\epsilon')^{-k})^2$

Therefore, either way, an increase of two in the depth results in an increase of the farness parameter from ϵ to at least $\epsilon(1+(8k/\epsilon)^{-k}/8)$. Thus in recursive calls of depth $16(8k/\epsilon)^k \log(\epsilon^{-1})$ the farness parameter exceeds 1 and the call returns without making any further calls. ■

Lemma 3.3 *Algorithm 1 uses at most* $\epsilon^{-480(8k/\epsilon)^{k+4}\log\log(\delta^{-1})}$ *queries.*

Proof. If $\epsilon > 1$ then the condition in Line 1 is satisfied and no queries are made. Therefore assume $\epsilon \leq 1$. Observe that in a specific instantiation at most one query is used, either in Line 2 or Line 3. Therefore the number of queries is upper bounded by the number of instantiations of Algorithm 1.

In a specific instantiation at most $32(8k/\epsilon)^k\epsilon^{-1}\log(\delta^{-1})$ recursive calls are made in total (note that by Line 13 there are at most $1/\epsilon$ children in the case of the condition in Line 12, and in the case of an unforceable gate there are at most $2k$ recursive calls). Recall that by Lemma 3.2 the depth of the recursion is at most $16(8k/\epsilon)^k\log(\epsilon^{-1})$.

To conclude, we note that the value of the confidence parameter in all these calls is lower bounded by $\delta\cdot(\epsilon/2k)^{16(8k/\epsilon)^k\log(\epsilon^{-1})} \geq \delta\cdot\epsilon^{32(8k/\epsilon)^k\log(k\epsilon^{-1})}$.

Therefore at most $(32(8k/\epsilon)^{2k}\epsilon^{-1}\log(\delta^{-1}\cdot\epsilon^{-32(8k/\epsilon)^k\log(k\epsilon^{-1})}))^{16(8k/\epsilon)^k\log(\epsilon^{-1})} \leq \epsilon^{-480(8k/\epsilon)^{k+4}\log\log(\delta^{-1})}$ queries are used. ■

Lemma 3.4 *If* Φ *on* σ *evaluates to* b *then Algorithm 1 returns "true" with probability* 1*.*

Proof. If $\epsilon > 1$ then the condition of Line 1 is satisfied and "true" is returned correctly. We proceed with induction over the depth of the formula. If $\texttt{depth}(\Phi) = 0$ then $\kappa_r \in X \cup \neg X$. If $\kappa_r \in X$ then since Φ evaluates to b, $\sigma(r) = b$, and if $\kappa_r \in \neg X$ then $\sigma(r) = 1-b$, and the algorithm returns "true" correctly.

Now assume that $\texttt{depth}(\Phi) > 0$. Obviously, for all $u \in \texttt{Children}(r)$, we have that $\texttt{depth}(\Phi) > \texttt{depth}(\Phi_u)$ and therefore from the induction hypothesis any recursive call with parameter $b' \in \{0,1\}$ on a subformula that evaluates to b' returns "true" with probability 1.

If $\kappa_r \equiv \wedge$ and $b = 1$ or $\kappa_r \equiv \vee$ and $b = 0$, then it must be the case that for all $u \in \texttt{Children}(r)$, Φ_u evaluates to b. By the induction hypothesis all recursive calls will return "true" and y will get the value "true", which will be returned by the algorithm.

Now assume that $\kappa_r \equiv \wedge$ and $b = 0$ or $\kappa_r \equiv \vee$ and $b = 1$. Since Φ evaluates to b then it must be the case that at least for one $u \in \texttt{Children}(r)$, Φ_u evaluates to b. By the induction hypothesis, the recursive call on that u will return "true", and y will get the value "true" which will be returned by the algorithm (unless the algorithm already returned "true" for another reason, e.g. in line 13).

Lastly, assume that r is an unforceable gate. Since Φ evaluates to b, the children of r evaluate to the assignment σ which evaluates to b. By the induction hypothesis, for every $u \in \texttt{Children}(r)$ the recursive call on Φ_u with $\sigma(u)$ will return "true", and thus the assignment σ will, in particular, fill the condition in Line 23 and the algorithm will return "true". ■

Lemma 3.5 *If* σ *is* ϵ*-far from getting* Φ *to output* b *then Algorithm 1 returns "false" with probability at least* $1-\delta$*.*

Proof. The proof is by induction over the tree structure, where we partition to cases according to κ_r and b. Note that $\epsilon \leq 1$.

If $\kappa_r \in X$ or $\kappa_r \in \neg X$ then by Lines 2 or 3 the algorithm returns "false" whenever σ does not make Φ output b.

If $\kappa_r \equiv \wedge$ and $b = 1$ or $\kappa_r \equiv \vee$ and $b = 0$, since σ is ϵ-far from getting Φ to output b then by Observation 2.18 we get that there exists $T \subseteq \texttt{Children}(r)$ for which it holds that $\sum_{t\in T}|\Phi_t| \geq |\Phi|\epsilon((8k/\epsilon)^{-k}/16)$ and each Φ_t is $\epsilon(1-(8k/\epsilon)^{-k}/16)$-far from being evaluated to b. Let S be the set of all vertices selected in Line 7. The probability of a vertex from T being selected is at least $\epsilon((8k/\epsilon)^{-k}/16)$. Since this happens at least $32(8k/\epsilon)^k\epsilon^{-1}\log(\delta^{-1})$ times independently, with probability at least $1-\delta/2$ we have that $S \cap T \neq \emptyset$. Letting $w \in T \cap S$, the recursive call on it with parameter $\epsilon(1-(8k/\epsilon)^{-k}/16)$

will return "false" with probability at least $1-\delta/2$, which will eventually cause the returned value to be "false" as required. Thus the algorithm succeeds with probability at least $1-\delta$.

Now assume that $\kappa_r \equiv \wedge$ and $b = 0$ or $\kappa_r \equiv \vee$ and $b = 1$. Since Φ is ϵ-far from being evaluated to b, Observation 2.19 implies that all children are of weight at least ϵ and are $\epsilon(\epsilon+1)$-far from b, and therefore the conditions of Line 13 would not be triggered. Every recursive call on a vertex $v \in \texttt{Children}(r)$ is made with distance parameter $\epsilon(1+\epsilon)$ and so it returns "true" with probability at most $\epsilon\delta/2$. Since there are at most ϵ^{-1} children of r, the probability that none returns "true" is at least $1-\delta/2$ and in that case the algorithm returns "false" successfully.

Now assume that κ_r is some unforceable gate. By Lemma 2.21, since Φ is ϵ-far from being satisfied the condition in Line 18 is not triggered. If the algorithm returned "true" then it must be that the condition in Line 23 is satisfied. If there exists some heavy child $u \in \texttt{Children}(r)$ such that y_u^b is "true" and y_u^{1-b} is "false", then by Lemma 3.4 the formula Φ_u does evaluate to b and the assignment σ must be such that $\sigma(u) = b$. For the rest of the children of r, assuming the calls succeeded, the subformula rooted in each v is $(\epsilon(1+(4k/\epsilon)^{-k}))$-close to evaluate to $\sigma(v)$. Since u is heavy, the total weight of $\texttt{Children}(r) \setminus \{u\}$ is at most $1-(4k/\epsilon)^{-k}$, and thus by changing at most a $(\epsilon(1+(4k/\epsilon)^{-k}))(1-(4k/\epsilon)^{-k}) \leq \epsilon$ fraction of inputs we can get to an assignment where Φ evaluates to b.

If all heavy children u are such that both y_u^b and y_u^{1-b} are "true", then pick some heavy child u arbitrarily. Since r is unforceable, there is an assignment that evaluates to b no matter what the value of Φ_u is. Take such an assignment $\tilde{\sigma}$ that fits the real value of Φ_u. Note that for every heavy child v we have that $y_v^{x_v}$ is "true", and therefore by changing at most an $(\epsilon(1+(4k/\epsilon)^{-k}))$-fraction of the variables in Φ_v we can get it to evaluate to x_v. The weight of u is at least $(4k/\epsilon)^{-\ell+1}$ (recall the definition of ℓ in definition 2.20), thus the total weight of the other heavy children is at most $1-(4k/\epsilon)^{-\ell+1}$ and the total weight of the light children is at most $\frac{\epsilon}{4}(4k/\epsilon)^{-\ell}$. So by changing all subformulas to evaluate to the value implied by $\tilde{\sigma}$ we change at most an $(\epsilon(1+(4k/\epsilon)^{-k}))(1-(4k/\epsilon)^{-\ell+1}) + \frac{\epsilon}{4}(4k/\epsilon)^{-\ell} \leq \epsilon$ fraction of inputs and get an assignment where Φ evaluates to b. Note that this $\tilde{\sigma}$ does not necessarily correspond to the x found in Line 23.

Thus we have found that finding an assignment x in Line 23, assuming the calls are correct, implies that Φ is ϵ-close to evaluate to b. The probability that all relevant calls to an assignment return "true" incorrectly is at most the probability that any of the $2k$ recursive calls errs, which by the union bound is at most δ, and the algorithm will return "false" correctly with probability at least $1-\delta$. ■

4 Estimator for monotone formula of bounded arity

Algorithm 2 below operates in a recursive manner, and estimates the distance to satisfying the formula rooted in r according to estimates for the subformula rooted in every child of r. The algorithm receives a confidence parameter δ as well as the approximation parameter ϵ, and should with probability at least $1-\delta$ return a number η such that the input is both $(\eta+\epsilon)$-close and $(\eta-\epsilon)$-far from satisfying the given formula.

The following states that Algorithm 2 indeed gives an estimation of the distance. While estimation algorithms cannot have 1-sided error, there is an additional feature of this algorithm that makes it also useful as a 1-sided test (by running it and accepting if it returns $\eta = 0$).

Theorem 4.1 *With probability at least $1-\delta$, the output of Algorithm 2$(\Phi, \epsilon, \delta, \sigma)$ is an η such that the assignment σ is both $(\eta+\epsilon)$-close to satisfying Φ and $(\eta-\epsilon)$-far from satisfying it. Additionally, if the assignment σ satisfies Φ then $\eta = 0$ with probability 1. Its query complexity (treating k and δ as constant) is $O(\exp(poly(\epsilon^{-1})))$.*

Proof. The bound on the number of queries is a direct result of Lemma 4.3 below. Given that, the correctness proof is done by induction on the height over the formula. The base case (for any ϵ and δ) is the observation that an instantiation of the algorithm that makes no recursive calls (i.e. triggers the condition in Line 1, 2 or 3) always gives a value that satisfies the assertion.

The induction step uses Lemma 4.4 and Lemma 4.5 below. Given that the algorithm performs correctly (for any ϵ and δ) for every formula Φ' of height smaller than Φ, the assertions of the lemma corresponding to κ_r (out of the two) are satisfied, and so the correctness for Φ itself follows. ■

Algorithm 2 Approximate distance to satisfiability of monotone formula

Input: read-once k-basic formula $\Phi = (V, E, r, X, \kappa)$, parameters $\epsilon, \delta > 0$, oracle to σ .
Output: $\eta \in [0, 1]$.

1: **if** $\kappa_r \in X$ **then return** $1 - \sigma(\kappa_r)$
2: **if** $\epsilon > 1$ **then return** 0
3: **if** $\kappa_r \equiv \vee$ and there exists $u \in \texttt{Children}(r)$ with $|\Phi_u| < \epsilon|\Phi|$ **then return** 0
4: **if** $\kappa_r \equiv \wedge$ **then**
5: **for** $i = 1$ to $l = \lceil 1000\epsilon^{-2k-2}(8k)^{2k} \cdot \log(1/\delta) \rceil$ **do**
6: $u \longleftarrow$ a vertex in $\texttt{Children}(r)$ selected independently at random, where the probability that $w \in \texttt{Children}(r)$ is selected is $|\Phi_w|/|\Phi|$
7: $\alpha_i \longleftarrow$ Algorithm 2$(\Phi_u, \epsilon(1 - (8k/\epsilon)^{-k}/8), \delta\epsilon(8k/\epsilon)^{-k}/32, \sigma)$
8: **end for**
9: **return** $\sum_{i=1}^{l} \alpha_i/l$
10: **else**
11: **for** every light child u of r set $\alpha_u \longleftarrow 0$
12: **for** every heavy child u of r perform a recursive call and use return value to set $\alpha_u \longleftarrow$ Algorithm 2$(\Phi_u, \epsilon(1 + (4k/\epsilon)^{-k}), \delta/\max\{k, 1/\epsilon\}, \sigma)$
13: **for** every term C in the mDNF of κ_r set $\alpha_C \longleftarrow \sum_{u \in C} \alpha_u \cdot \frac{|\Phi_u|}{|\Phi|}$
14: **return** $\min\{\alpha_C : C \in \text{mDNF}(\kappa_r)\}$
15: **end if**

The dependence on δ can be made into a simple logarithm by a standard amplification technique: Algorithm 2 is run $O(1/\delta)$ independent times, each time with a confidence parameter of 2/3, and then the median of the outputs is taken.

Lemma 4.2 *When called with Φ, ϵ, δ, and oracle access to σ, Algorithm 2 goes at most $3(8k/\epsilon)^k \log(1/\epsilon) = \text{poly}(\epsilon)$ recursion levels down.*

Proof. Recursion can only happen on Line 7 and Line 12. Moreover, because of the formula being k-basic, recursion cannot follow through Line 7 two recursion levels in a row. In every two consecutive recursive calls there are three options:

1. The first call is made with farness parameter $\epsilon' = \epsilon(1 + (4k/\epsilon)^{-k})$ and the second with $\epsilon'' = \epsilon'(1 - (8k/\epsilon')^{-k}/8)$. In this case the farness parameter increases by a factor of $(1 + (4k/\epsilon)^{-k})(1 - (8k/(\epsilon(1 + (4k/\epsilon)^{-k})))^{-k}/8) \geq (1 + \frac{7}{8}(4k/\epsilon)^{-k})$.

2. The first call is made with farness parameter $\epsilon' = \epsilon(1 - (8k/\epsilon)^{-k}/8)$ and the second with $\epsilon'' = \epsilon'(1 + (4k/\epsilon')^{-k})$. In this case the farness parameter increases by a factor of $(1 - (8k/\epsilon)^{-k}/8)(1 + (4k/(\epsilon(1 - (8k/\epsilon)^{-k}/8)))^{-k}) \geq (1 + \frac{7}{8}(8k/\epsilon)^{-k})$

3. The first call is made with farness parameter $\epsilon' = \epsilon(1 + (4k/\epsilon)^{-k})$ and the second with $\epsilon'' = \epsilon'(1+(4k/\epsilon')^{-k})$. In this case the farness parameter increases by a factor of at least $(1+(4k/\epsilon)^{-k})^2$.

Therefore, either way, in every two consecutive levels of the recursion ϵ is increased by a factor of at least $(1 + \frac{7}{8}(8k/\epsilon)^{-k})$. After $3(8k/\epsilon)^k \log(1/\epsilon)$ recursive steps, such an increase has occurred at least $\frac{3}{2}(8k/\epsilon)^k \log(1/\epsilon)$ times, and therefore the farness parameter is at least $\epsilon \cdot (1+\frac{7}{8}(8k/\epsilon)^{-k})^{\frac{3}{2}(8k/\epsilon)^k \log(1/\epsilon)} > 1$. In such a case the algorithm immediately returns 0 and the recursion stops. ■

Lemma 4.3 *When called with Φ, ϵ, δ, and oracle access to σ, Algorithm 2 uses a total of at most $\exp(\text{poly}(1/\epsilon))$ queries for any constant k.*

Proof. Denote by ϵ' the smallest value of the farness parameter in any recursive call. Denoting by δ' the smallest value of δ in any recursive call, it holds that $\delta' \geq \delta(\epsilon'(8k/\epsilon')^{-k}/32)^{3(8k/\epsilon)^k \log(1/\epsilon)}$ by Lemma 4.2. The number of recursive calls per instantiation of the algorithm is thus at most $l' =$

$\lceil 1000\epsilon'^{-2k-2}(8k)^{2k} \cdot \log(1/\delta') \rceil = \text{poly}(1/\epsilon')$. Now, by the proof of Lemma 4.2, every two consecutive recursive calls increase the value of the farness parameter, since it only decreases in line 7, it holds that $\epsilon' \geq \epsilon(1-(8k/\epsilon)^{-k}/8))$. This means that $l' = \text{poly}(1/\epsilon)$.

Since the algorithm may make at most one query per instantiation, and this only in the case where a recursive call is not performed, the total number of queries is (bounding the recursion depth through Lemma 4.2) at most $(l')^{3(8k/\epsilon)^k \log(1/\epsilon)} = \exp(\text{poly}(1/\epsilon))$. ■

Lemma 4.4 *If $\kappa_r \not\equiv \wedge$ and all recursive calls satisfy the assertion of Theorem 4.1, then with probability at least $1-\delta$ the current instantiation of Algorithm 2 provides a value η such that σ is both $(\eta+\epsilon)$-close to satisfying Φ and $(\eta-\epsilon)$-far from satisfying it. Furthermore, if σ satisfies Φ then with probability 1 the output is $\eta = 0$.*

Proof. First we note that Step 3, if triggered, gives a correct value for η (as the σ can be made into a satisfying assignment by changing possibly all variables of the smallest child of r). We also note that if $\kappa_r \equiv \vee$ and Step 3 was not triggered, then by definition all of r's children are heavy, and there are no more than $1/\epsilon$ of them.

Let us consider the cost of fixing input bits in order to make σ satisfy Φ. Note that any such fix must make all of the children in some term C in the mDNF evaluate to 1, since these terms are all of the 1-witnesses. Additionally, making all of the children of one term evaluate to 1 is sufficient. Therefore, the farness of σ from Φ is the minimum over all terms C in κ_r of the adjusted cost of making all children of C evaluate to 1, which is $\sum_{u \in C} \texttt{farness}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|}$. Now in this case there are clearly no more than $\max\{k, \epsilon^{-1}\}$ children, so by the union bound, with probability at least $1-\delta$, every call done through Line 7 gave a value η_u so that indeed σ is $(\eta_u + \epsilon(1+(4k/\epsilon)^{-k}))$-close and $(\eta_u - \epsilon(1+(4k/\epsilon)^{-k}))$-far from Φ_u.

Now let D_i denote C_i minus any light children that it may contain, since the approximation ignores these. It may be that some D_i's contain all heavy children of C_i, where "heavy children" refers to the children of r. Since there are no forcing children (and there exist heavy children) it must be the case that some D_i's do not contain all heavy children, since if a heavy child appears in all D_is, then it appears in all C_is and therefore by setting it to 0 we force a 0 in the output. The D_is that do not contain all heavy children will dominate the expression in Line 14. Note that $\sum_{u \in D_i} |\Phi_u| \leq (1-(4k/\epsilon)^{k-\ell})|\Phi|$ for any D_i not containing a heavy child. This implies by bounding $(1+(4k/\epsilon)^{-k})) \cdot (1-(4k/\epsilon)^{k-\ell})$:

$$\sum_{u \in D_i} \texttt{farness}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} - \epsilon < \frac{\sum_{u \in D_i} \eta_u |\Phi_u|}{|\Phi|}$$

$$< \sum_{u \in D_i} \texttt{farness}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} + \epsilon - 2k(4k/\epsilon)^{-\ell}$$

Now the true farness of C_i not containing all heavy children is at least that of D_i, and at most that of D_i plus the added farness of making all light children evaluate to 1, which is bounded by $k(4k/\epsilon)^{-\ell}$. This means that for such a C_i we have:

$$\sum_{u \in C_i} \texttt{farness}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} - \epsilon < \frac{\sum_{u \in D_i} \eta_u |\Phi_u|}{|\Phi|}$$

$$< \sum_{u \in C_i} \texttt{farness}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} + \epsilon - k(4k/\epsilon)^{-\ell}$$

The value returned as η is the minimum over terms C_i in κ_r of $\eta_u \cdot \frac{\sum_{u \in D_i} |\Phi_u|}{|\Phi|}$. We also know that this minimum is reached by some C_j which does not contain all heavy children, but it may be that in fact $\texttt{farness}(\sigma, SAT(\Phi)) = \sum_{u \in C_i} \texttt{farness}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|}$ for some $i \neq j$ (the true farness is the minimum of the total farness of each clause, but it may be reached by a different clause).

By our assumptions

$$\texttt{farness}(\sigma, SAT(\Phi)) - \epsilon = \sum_{u \in C_i} \texttt{farness}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} - \epsilon$$

$$\leq \sum_{u \in C_j} \texttt{farness}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} - \epsilon < \eta$$

so we have one side of the required bound. For the other side, we split into cases. If C_i also does not contain all heavy children then we use the way we calculated η as the minimum over the corresponding sums:

$$\eta = \frac{\sum_{u \in D_j} \eta_u |\Phi_u|}{|\Phi|} \leq \frac{\sum_{u \in D_i} \eta_u |\Phi_u|}{|\Phi|} < \texttt{farness}(\sigma, SAT(\Phi)) + \epsilon$$

In the final case, we note that by the assumptions on the light children we will always have (recalling that C_i will in particular have all heavy children of C_j):

$$\eta = \frac{\sum_{u \in D_j} \eta_u |\Phi_u|}{|\Phi|} < \sum_{u \in C_j} \texttt{farness}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} + \epsilon - k(4k/\epsilon)^{-\ell}$$

$$\leq \sum_{u \in C_i} \texttt{farness}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} + \epsilon$$

where the rightmost term equals $\texttt{farness}(\sigma, SAT(\Phi)) + \epsilon$ as required.

For the last part of the claim, note that if σ satisfies Φ, then in particular, one of the terms C of κ_r must be satisfied. By the induction hypothesis, for all $u \in C$ we would have $\alpha_u = 0$ and therefore $\alpha_C = 0$, and since α is taken as a minimum over all terms we would have $\alpha = 0$. ■

Lemma 4.5 *If $\kappa_r \equiv \wedge$ and all recursive calls satisfy the assertion of Theorem 4.1, then with probability at least $1 - \delta$ the current instantiation of Algorithm 2 provides a value η such that σ is both $(\eta + \epsilon)$-close to satisfying Φ and $(\eta - \epsilon)$-far from satisfying it. If σ satisfies Φ then with probability 1 the output is $\eta = 0$.*

Proof. First note that if we sample a vertex w according to the distribution of Line 5 and then take the true farness $\texttt{farness}(\sigma, SAT(\Phi_w))$, then the expectation (but not the value) of this equals $\texttt{farness}(\sigma, SAT(\Phi))$. This is because to make σ evaluate to 1 at the root, we need to make all its children evaluate to 1, an operation whose adjusted cost is given by the weighted sum of farnesses that corresponds to the expectation above.

Thus, denoting by X_i the random variable whose value is $\texttt{farness}(\sigma, SAT(\Phi_{w_i}))$ where w_i is the vertex picked in the ith iteration, we have $\mathrm{E}[X_i] = \texttt{farness}(\sigma, SAT(\Phi))$. By a Chernoff type bound, with probability at least $1 - \delta/2$, the average X of $X_1, \ldots, X_l$ is no more than $\epsilon^{k+1}(4k)^{-k}/16$ away from $\mathrm{E}[X_i]$ and hence satisfies:

$$\texttt{farness}(\sigma, SAT(\Phi)) - \epsilon^{k+1}(4k)^{-k}/16 < X < \texttt{farness}(\sigma, SAT(\Phi)) + \epsilon^{k+1}(4k)^{-k}/16$$

Then note that by the Markov inequality, the assertion of the lemma means that with probability at least $1 - \delta/2$, all calls done in Line 12 but at most $\epsilon(4k/\epsilon)^{-k}/16$ of them return a value η_w so that σ is $(\eta_w + \epsilon(1 - (4k/\epsilon)^{-k}/16))$-close and $(\eta_w - \epsilon(1 - (4k/\epsilon)^{-k}/16))$-far from Φ_w.

When this happens, at least $(1 - \epsilon(4k/\epsilon)^{-k}/16)$ of the answers α_i of the calls are up to $\epsilon(1 - (4k/\epsilon)^{-k}/16))$ away from each corresponding X_i, and at most $\epsilon(4k/\epsilon)^{-k}/16$ of the answers α_i are up to 1 away from each corresponding X_i. Summing up these deviations, the final answer average η satisfies

$$X - \epsilon(1 - (4k/\epsilon)^{-k}/8) - \epsilon(4k/\epsilon)^{-k}/16 < \eta < X + \epsilon(1 - (4k/\epsilon)^{-k}/8) + \epsilon(4k/\epsilon)^{-k}/16$$

With probability at least $1 - \delta$ both of the above events occur, and summing up the two inequalities we obtain the required bound

$$\texttt{farness}(\sigma, SAT(\Phi)) - \epsilon \leq \eta < \texttt{farness}(\sigma, SAT(\Phi)) + \epsilon$$

■

5 Quasi-polynomial Upper Bound for Basic-Formulas

Let $\Phi = (V, E, r, X, \kappa, B)$ be a basic formula and σ be an assignment to Φ.

The main idea of the algorithm is to randomly choose a full root to leaf path, and recurs over all the children of "$\vee$" vertices on this path that go outside of it, if they are not too many. The main technical part is in proving that if σ is indeed ϵ-far from satisfying Φ, then many of these paths have few such children (few enough to recurs over all of them), where additionally the distance of σ from satisfying the corresponding sub-formulas is significantly larger. An interesting combinatorial corollary of this is that formulas, for which there are not a lot of leaves whose corresponding paths have few such children, do not admit ϵ-far assignments at all.

5.1 Critical and Important

To understand the intuition behind the following definitions, it is useful to first consider what happens if we could locate a vertex that is "(ϵ, σ)-critical" in the sense that is defined next.

Definition 5.1 [(ϵ, σ)-***important,*** (ϵ, σ)-***critical***] *A vertex $v \in V$ is* (ϵ, σ)-important *if $\sigma \notin SAT(\Phi)$, and for every u that is either v or an ancestor of v, we have that*

- $\texttt{farness}(\sigma, SAT(\Phi_u)) \geq (2\epsilon/3)(1 + 2\epsilon/3)^{\lfloor \texttt{depth}_\Phi(u)/3 \rfloor}$
- *If $\kappa_u \equiv \vee$ and $u \neq v$ then the heaviest child of u, $h(u)$ is either v or an ancestor of v.*

An (ϵ, σ)-critical *vertex v is an (ϵ, σ)-important vertex v for which $\kappa_v \in X$.*

Note that such a vertex is never too deep, since $\texttt{farness}(\sigma, SAT(\Phi_u))$ is always at most 1. Hence the following observation follows from Definition 5.1.

Observation 5.2 *If v is (ϵ, σ)-important, then $\texttt{depth}_\Phi(v) \leq 4\epsilon^{-1} \log(2\epsilon^{-1})$.*

A hypothetical oracle that provides a critical vertex can be used as follows. If v is the vertex returned by such an oracle, then for every ancestor u of v such that $\kappa_u = \vee$, and every $w \in \texttt{Children}(u)$ that is not an ancestor of v, a number of recursive calls with Φ_w and distance parameter significantly larger than ϵ are used. The following lemma implies that if for each of these vertices one of the recursive calls returned 0, then we know that $\sigma \notin SAT(\Phi)$.

Definition 5.3 (Special relatives) *The set of special relatives of $v \in V$ is the set T of every u that is not an ancestor of v or v itself but is a child of an ancestor w of v, where $\kappa_w \equiv \vee$.*

Lemma 5.4 *If $\sigma \notin SAT(\Phi_u)$ for every $u \in T \cup \{v\}$, then $\sigma \notin SAT(\Phi)$.*

Proof. If $\texttt{depth}_\Phi(v) = 0$ then $\sigma \notin SAT(\Phi_v)$ implies $\sigma \notin SAT(\Phi)$. Assume by induction that the lemma holds for any formula $\Phi' = (V', E', r', X', \kappa')$, assignment σ' to Φ' and vertex $u \in V'$ such that $0 \leq \texttt{depth}_{\Phi'}(u) < \texttt{depth}_\Phi(v)$. Let w be the parent of v. Observe that the special relatives of w are a subset of the special relatives of v and hence by the induction assumption we only need to prove that $\sigma \notin SAT(\Phi_w)$ in order to infer that $\sigma \notin SAT(\Phi)$.

If $\kappa_w \equiv \wedge$, then $\sigma \notin SAT(\Phi_v)$ implies that $\sigma \notin SAT(\Phi_w)$. If $\kappa_w \equiv \vee$, then $\sigma \notin SAT(\Phi_v)$ and $\sigma \notin SAT(\Phi_u)$ for every $u \in T$ implies that $\sigma \notin SAT(\Phi_w)$, since we have that $\texttt{Children}(w) \setminus \{v\} \subseteq T$. ■

The following lemma states that if σ is ϵ-far from $SAT(\Phi)$, then (ϵ, σ)-critical vertices are abundant, and so we can locate one of them by merely sampling a sufficiently large (linear in $1/\epsilon$) number of vertices.

The main part of the proof that this holds is in showing that if σ is only $2\epsilon/3$-far from $SAT(\Phi)$, then there exists an (ϵ, σ)-critical vertex for σ. We first show that this is sufficient to show the claimed abundance of (ϵ, σ)-critical vertices, and then state and prove the required lemma.

Lemma 5.5 *If σ is ϵ-far from $SAT(\Phi)$, then $|\{v | v \text{ is } (\epsilon, \sigma)\text{-critical}\}| \geq \epsilon|\Phi|/4$.*

Proof. Set $\texttt{Critical}_{\epsilon,\sigma} = \{v | v \text{ is } (\epsilon,\sigma)\text{-critical}\}$ and assume on the contrary that $|\texttt{Critical}_{\epsilon,\sigma}| < \epsilon|\Phi|/4$. Set σ' to be an assignment to X so that for every $s \in V$ where $\kappa_s \in X$, we have that $\sigma'(\kappa_s) = 1$ if $\kappa_s \in \texttt{Critical}_{\epsilon,\sigma}$ and otherwise $\sigma'(x) = \sigma(x)$. Thus $\texttt{Critical}_{\epsilon,\sigma'} = \emptyset$. By the triangle inequality we have that

$$\texttt{farness}(\sigma, SAT(\Phi)) - \texttt{farness}(\sigma', SAT(\Phi)) \leq \texttt{farness}(\sigma', \sigma).$$

Finally, by $\texttt{Critical}_{\epsilon,\sigma'} = \emptyset$, Lemma 5.6, which we prove below, asserts that $\texttt{farness}(\sigma', SAT(\Phi)) < 2\epsilon/3$ and we reach a contradiction. ■

Lemma 5.6 *If there is no (ϵ,σ)-critical vertex, then σ is $2\epsilon/3$-close to $SAT(\Phi)$.*

Proof. We shall show that if σ is $2\epsilon/3$-far from $SAT(\Phi)$, then there exists an (ϵ,σ)-critical vertex. Assume that σ is $2\epsilon/3$-far from $SAT(\Phi)$. This implies that r is an (ϵ,σ)-important vertex. Hence an (ϵ,σ)-important vertex exists. Let v be an (ϵ,σ)-important vertex such that $\texttt{depth}_\Phi(v)$ is maximal. Consequently, none of the vertices in $\texttt{Children}(v)$ are (ϵ,σ)-important. We next prove that v is (ϵ,σ)-critical.

Assume on the contrary that v is not (ϵ,σ)-critical. Consequently $\kappa_v \notin X$ and hence to get a contradiction it is sufficient to show that there exists an (ϵ,σ)-important vertex in $\texttt{Children}(v)$. If $\kappa_v \equiv \vee$, then by Observation 2.19 we get that

$$\texttt{farness}(\sigma, SAT(\Phi_{h(v)})) \geq (2\epsilon/3)(1+2\epsilon/3)^{\lfloor \texttt{depth}_\Phi(h(v))/3 \rfloor},$$

and hence $h(v)$ is (ϵ,σ)-important.

Assume that $\kappa_v \equiv \wedge$. Let u be such that $\texttt{farness}(\sigma, SAT(\Phi_u)) \geq \texttt{farness}(\sigma, SAT(\Phi_v))$. Observation 2.18 asserts that such a vertex exists. We assume that $\texttt{depth}_\Phi(u) > 2$, since otherwise it cannot be the case that $\texttt{farness}(\sigma, SAT(\Phi_u)) < (2\epsilon/3)(1+2\epsilon/3)^0$. Let $w \in V$ be the parent of v. Since w is an ancestor of v it is (ϵ,σ)-important, and hence $\texttt{farness}(\sigma, SAT(\Phi_w)) \geq (2\epsilon/3)(1+2\epsilon/3)^{\lfloor \texttt{depth}_\Phi(w)/3 \rfloor}$. Since Φ is basic we have that $\kappa_w \equiv \vee$. Thus by Observation 2.19 we get that

$$\texttt{farness}(\sigma, SAT(\Phi_v)) \geq (2\epsilon/3)(1+2\epsilon/3)^{1+\lfloor \texttt{depth}_\Phi(w)/3 \rfloor}.$$

Finally since $\texttt{farness}(\sigma, SAT(\Phi_u)) \geq \texttt{farness}(\sigma, SAT(\Phi_v))$ and additionally we have $\texttt{depth}_\Phi(u) = \texttt{depth}_\Phi(w) + 2$ we get that

$$\texttt{farness}(\sigma, SAT(\Phi_u)) \geq (2\epsilon/3)(1+2\epsilon/3)^{\lfloor \texttt{depth}_\Phi(u)/3 \rfloor}.$$

■

5.2 Algorithm

This algorithm detects far inputs with probability $\Omega(\epsilon)$, but this can be amplified to 2/3 using iterated applications.

Algorithm 3 Test satisfiability of basic read-once formula

Input: read-once basic formula $\Phi = (V, E, r, X, \kappa)$, a parameter $\epsilon > 0$, oracle to σ .
Output: $z \in \{0, 1\}$.

1: **if** $\epsilon > 1$ **then return** 1
2: **if** $\kappa_r \in X$ **then return** $\sigma(\kappa_r)$
3: Pick s uniformly at random from all v such that $\kappa_v \in X$
4: $A \longleftarrow$ all ancestors v of s such that $\kappa_v \equiv \vee$
5: $R \longleftarrow \left(\bigcup_{v \in A} \texttt{Children}(v)\right) \setminus \{w \mid w \text{ is an ancestor of } s\}$
6: **if** $|R| > 3\epsilon^{-2} \log(2\epsilon^{-1})$ **then return** 1
7: **for all** $u \in R$ **do**
8: $\quad y_u \longleftarrow 1$
9: $\quad$ **for** $i = 1$ to $\lceil 20\epsilon^{-1} \log \epsilon^{-1} \rceil$ **do** $y_u \longleftarrow y_u \wedge$ Algorithm 3$(\Phi_u, \sigma, 4\epsilon/3)$
10: **end for**
11: **return** $\sigma(\kappa_s) \vee \bigvee_{u \in R} y_u$

We now proceed to prove the correctness of Algorithm 3. Algorithm 3 is clearly non-adaptive. We first bound its number of queries and next prove that it always returns "1" for an assignment that satisfies the formula, and returns "0" with probability linear in ϵ for an assignment that is ϵ-far from satisfying the formula. Using $O(1/\epsilon)$ independent iterations amplifies the later probability to 2/3.

Lemma 5.7 *For $\epsilon > 0$, Algorithm 3 halts after using at most $\epsilon^{-16+16\log\epsilon}$ queries, when called with Φ, ϵ and oracle access to σ.*

Proof. The proof is formulated as an inductive argument over the value of the (real) farness parameter ϵ. However, it is formulated in a way that it can be viewed as an inductive argument over the integer valued $\lceil \log(\alpha\epsilon^{-1}) \rceil$, for an appropriate global constant α. This is since the value of the distance parameter increases multiplicatively with every recursive call.

If $\epsilon > 1$, then the condition in Line 1 is satisfied, and there are no queries or recursive calls. Hence we assume that $\epsilon \leq 1$. Observe that in a specific instantiation at most one query is used, since a query is only made on Line 2 or on Line 11, and always as part of a "return" command. Hence the number of queries is upper bounded by the number of calls to Algorithm 3 (initial and recursive). We shall show that the number of these calls is at most $\epsilon^{-16+16\log\epsilon}$.

Assume by induction that for some $\eta \leq 1$, for every $\eta \leq \eta' \leq 1$, every formula Φ' and assignment σ' to Φ', on call to Algorithm 3 with Φ', η' and an oracle to σ', at most $\eta'^{-16+16\log\eta'}$ calls to Algorithm 3 are made (including recursive ones).

Assume that $\epsilon > 3\eta/4$. If $\kappa_r \in X$, then the condition on Line 2 is satisfied and hence there are no recursive calls. Thus Algorithm 3 is called only once and $1 \leq \epsilon^{-16+16\log\epsilon}$.

Assume that $\kappa_r \notin X$. Note that every recursive call is done by Line 9. By Line 7 and Line 9 at most $|R| \cdot \lceil 20\epsilon^{-1}\log\epsilon^{-1} \rceil$ recursive calls are done. The condition on Line 6 ensures that $|R| \cdot \lceil 20\epsilon^{-1}\log\epsilon^{-1} \rceil \leq 3\epsilon^{-2}\log(2\epsilon^{-1}) \cdot \lceil 20\epsilon^{-1}\log\epsilon^{-1} \rceil$. According to Line 9 each one of these recursive calls is done with distance parameter $4\epsilon/3 > \eta$. Thus by the induction assumption the number of calls to Algorithm 3 is at most

$$3\epsilon^{-2}\log(2\epsilon^{-1}) \cdot \lceil 20\epsilon^{-1}\log\epsilon^{-1} \rceil \cdot (4\epsilon/3)^{-16+16\log(4\epsilon/3)}.$$

This is less than $\epsilon^{-16+16\log\epsilon}$. ■

The following theorem will be immediate from Lemma 5.7 above when coupled with Lemma 5.10 and Lemma 5.12 below.

Theorem 5.8 *Let $\epsilon > 0$. When Algorithm 3 is called with Φ, ϵ and an oracle to σ, it uses at most $\epsilon^{-16+16\log\epsilon}$ queries; if $\sigma \in SAT(\Phi)$ then it always returns 1, and if σ is ϵ-far from $SAT(\Phi)$ then it returns 0 with probability at least $\epsilon/8$.*

Theorem 5.8 does not imply that Algorithm 3 is an ϵ-test for $SAT(\Phi)$. However it does imply that in order to get an ϵ-test for $SAT(\Phi)$ it is sufficient to do the following. Call Algorithm 3 repeatedly $\lceil 20\epsilon^{-1}\log\epsilon^{-1} \rceil$ times, return 0 if any of the calls returned 0, and otherwise return 1. This only increases the query complexity to the value in the following corollary.

Corollary 5.9 *There exists an ϵ-test for Φ, that uses at most $\epsilon^{-20+16\log\epsilon}$ queries.*

Lemma 5.10 *Let $\epsilon > 0$ and $\sigma \in SAT(\Phi)$. Algorithm 3 returns 1 when called with Φ, ϵ and an oracle to σ.*

Proof. To prove the lemma we shall show that if Algorithm 3 returns 0, when called with Φ, ϵ and oracle access to σ, then $\sigma \notin SAT(\Phi)$. We will show this by induction on $\texttt{depth}(\Phi)$. If $\texttt{depth}(\Phi) = 0$ then the condition in Line 1 is satisfied and $\sigma(\kappa_r)$ is returned. Hence $\sigma(\kappa_r) = 0$ and therefore $\sigma \notin SAT(\Phi)$. Assume that for every $\epsilon' > 0$, Φ' where $\texttt{depth}(\Phi') < \texttt{depth}(\Phi)$, and assignment σ' to Φ', if Algorithm 3 returns 0, when called with Φ', ϵ' and oracle access to σ', then $\sigma' \notin SAT(\Phi)$.

Observe that the only other way a 0 can be returned is through Line 11, if it is reached. Let R be the set of vertices on which there was a recursive call in Line 9 and κ_s the variable whose value is queried on Line 11. According to Line 11 a 0 is returned if and only if $\sigma(\kappa_s) = 0$ and for every $u \in R$ there

was at least one recursive call with Φ_u and distance parameter $4\epsilon/3$ that returned a 0. By the induction assumption this implies that $\sigma \notin SAT(\Phi_u)$ for every $u \in R$. Note that the set R satisfies the exact same conditions that the set T of special relatives satisfies in Lemma 5.4. Hence, Lemma 5.4 asserts that $\sigma \notin SAT(\Phi)$. ■

We now turn to proving soundness. This depends on first noting that the algorithm will indeed check the paths leading to critical vertices.

Observation 5.11 *If the vertex s picked in Line 3 is (ϵ, σ)-critical, then it will not trigger the condition of Line 6.*

Proof. Definition 5.1 in particular implies (see observation 2.19) that for every $u \in A$ (as per Line 4) we have $|\texttt{Children}(u)| \leq (3/2\epsilon)(1 + 2\epsilon/3)^{-\lfloor \texttt{depth}_\Phi(u)/3 \rfloor} \leq 3/2\epsilon$, as otherwise σ will be too close to satisfying Φ_u. Also, from Observation 5.2 we know that $\texttt{depth}_\Phi(s) \leq 4\epsilon^{-1}\log(2\epsilon^{-1})$ and so $|A| \leq 2\epsilon^{-1}\log(2\epsilon^{-1}) + 1$.

The two together give us the bound $|R| \leq (3/2\epsilon - 1)(2\epsilon^{-1}\log(2\epsilon^{-1}) + 1) \leq 3\epsilon^{-2}\log(2\epsilon^{-1})$, and so the condition in Line 3 is not triggered. ■

Lemma 5.12 *Let σ be ϵ-far from $SAT(\Phi)$. If Algorithm 3 is called with ϵ, Φ and an oracle to σ, then it returns 0 with probability at least $\epsilon/8$.*

Proof. We will prove this by induction on $\texttt{depth}(\Phi)$.

The base case, $\kappa_r \in X$, is handled correctly by Line 1. Assume next that $\epsilon > 3/4$. Assume first that the vertex s selected in Line 3 is (ϵ, σ)-critical. By Lemma 5.5, with probability at least 3/16 the vertex s selected in Line 3 is indeed (ϵ, σ)-critical. Hence by definition σ is more than 1/2-far from $SAT(\Phi_u)$ for every ancestor u of s. Thus by Observation 2.19 we have that $\kappa_u \equiv \wedge$ for every ancestor u of s. Consequently, by Line 2 and Line 11 the value returned will be $\sigma(\kappa_s)$, and $\sigma(\kappa_s) = 0$ because s is (ϵ, σ)-critical.

Thus, 0 is returned with probability at least 3/16, which is greater than $\epsilon/8$ when $3/4 < \epsilon \leq 1$.

For all other ϵ we proceed with the induction step. Assume that for any formula Φ' such that $\texttt{depth}(\Phi') < \texttt{depth}(\Phi)$ and any assignment σ' to Φ' that is η-far from $SAT(\Phi')$ (for any η), Algorithm 3 returns 0 with probability at least $\eta/8$. Given this we prove that 0 is returned with probability at least $\epsilon/8$ for Φ and σ.

Assume first that the vertex s selected in Line 3 is (ϵ, σ)-critical. Let A, R be the sets from Line 4 and Line 5. Since s is (ϵ, σ)-critical, by definition for every $u \in A$ we have that σ is $2\epsilon/3$-far from $SAT(\Phi_u)$. Also, because s is (ϵ, σ)-critical, by definition for every $u \in A$ and $w \in \texttt{Children}(u) \cap R$ we have that $w \neq h(u)$, and therefore by Observation 2.19 we have that σ is $4\epsilon/3$-far from $SAT(\Phi_w)$ for every $w \in R$.

By the induction assumption, for every $w \in R$, with probability at least $1 - (4\epsilon/3)/8$ Algorithm 3 returns 0 when called with $4\epsilon/3$, Φ_w and an oracle to σ. Hence, for every $w \in R$, the probability that on $\lceil 20\epsilon^{-1}\log\epsilon^{-1} \rceil$ such independent calls to Algorithm 3 the value 0 was never returned is at most $(1 - (4\epsilon/3)/8)^{\lceil 20\epsilon^{-1}\log\epsilon^{-1} \rceil}$. This is less than $(\epsilon^{-2}\log(2\epsilon^{-1}))/6$.

Observation 5.11 ensures that $|R| \leq 3\epsilon^{-2}\log(2\epsilon^{-1})$, and in particular the condition in Line 6 is not invoked and the calls in Line 9 indeed take place. By the union bound over the vertices of R, with probability at least 1/2, for every $u \in R$ at least one of calls to Algorithm 3 with $4\epsilon/3$, Φ_u and an oracle to σ returned the value 0. This means that for every $u \in R$, y_u in Line 9 was set to 0. Consequently this is the value returned in Line 11

Finally, since σ is ϵ-far from $SAT(\Phi)$, by Lemma 5.5 the vertex s selected in Line 3 is (ϵ, σ)-critical with probability at least $\epsilon/4$. Therefore 0 is returned with probability at least $\epsilon/8$. ■

6 The Computational Complexity of the Testers and Estimator

There are two parts to analyzing the computational complexity (as opposed to query complexity) of a test for a massively parametrized property. The first part is the running time of the preprocessing phase, which reads the entire parameter part of the input, in our case the formula, but has no access yet to the tested part, in our case the assignment. This part is subject to traditional running time and working

space definitions, and ideally should have a running time that is quasi-linear or at least polynomial in the size of its input (the "massive parameter"). The second part is the testing part, which ideally should take a time that is logarithmic in the input size for every query it makes (as a very basic example, even a tester that just makes independent uniformly random queries over the input would require such a time to draw the necessary $\log(n)$ random coins for each query).

In our case, the preprocessing part would need to take a k-ary formula and convert it to the basic form corresponding to the algorithm that we run. We may assume that the formula is represented as a graph with additional information stored in the vertices.

Constructing the basic form by itself can be done very efficiently (and also have an output size linear in its input size). For example, if the input formula has only "$\wedge$" and "$\vee$" gates, then a Depth First Search over the input would do nicely, where the output would follow this traversal, but create a new child gate in the output only when it is different than its parent (otherwise it would continue traversing the input while remaining in the same output node). With more general monotone gates, a first pass would convert them to unforceable gates by "splitting off" forceful children as in the proof of Lemma 2.14. It is not hard to efficiently handle "$\neg$" gates using De-Morgan's law too.

Aside from the basic form of the formula, the preprocessing part should construct several additional data structures to make the second part (the test itself) as efficient as possible.

For Algorithm 1, we would need to quickly pick a child of a vertex with probability proportional to its sub-formula size, and know who are the light children as well as what is the relative size of the smallest child. This mainly requires storing the size of every sub-formula for every vertex of the tree, as well as sorting the children of each vertex by their sizes and storing the value of the corresponding "ℓ". Algorithm 2 requires very much the same additional data as Algorithm 1. This information can be stored in the vertices of the graph while performing a depth-first traversal of it, starting at the root, requiring a time linear in the size of the basic formula.

For Algorithm 3, we would need to navigate the tree both downwards and upwards (for finding the ancestors of a vertex), as well as the ability to pick a vertex corresponding to a variable at random, which in itself does not require special preprocessing but does require generating a list of all such vertices. Constructing the set of ancestors is simply following the path from the vertex to the root, requiring time linear in the depth of the vertex in the tree.

The only part in the algorithms above that depends on ϵ is designating the light children, but this can also be done "for all ϵ" at a low cost by storing the range of ϵ for every positive ℓ. Since ℓ is always an integer no larger than $k+1$, this requires an array of such size in every vertex.

Let us turn to analyzing the running time complexity of the second part, namely the testing algorithm. Once the above preprocessing is performed, the time per instantiation (and thus per query) of the algorithm will be very small (where we charge the time it takes to calculate a recursive call to the recursive instantiation). In Algorithm 1, the cost in every instantiation is at most the cost of selecting a child vertex at random for each iteration of the loop in line 6, a cost linear in k for performing the calls in Lines 20 and 21 and a cost of $O(2^k)$ for searching the space of possible x's in Line 23. This would make it a cost logarithmic in the input size per query (multiplied by the time it takes to write and read an address) – where the log incurrence is in fact only when we need to randomly choose a child according to its weight. The case of Algorithm 2 is similar, except that while we don't have the cost of iterating over possible assignments to the root, there is an additional constant cost for every term in the mDNF, of which there are at most 2^k.

For Algorithm 3, every instantiation requires iterating over all the ancestors of one vertex picked at random. This requires time linear in the depth of the formula and logarithmic in the input size per query, where the depth only depends on the farness parameter (see observation 5.2).

7 The Untestable Formulas

We describe here a read-once formula over an alphabet with 4 values, defining a property that cannot be 1/4-tested using a constant number of queries. The formula will have a very simple structure, with only one gate type. Then, building on this construction, we describe a read-once formula over an alphabet

with 5-values that cannot be 1/12-tested, which satisfies an additional monotonicity condition: All gates as well as the acceptance condition are monotone with respect to a fixed ordering of the alphabet.

7.1 The 4-valued formula

For convenience we denote our alphabet by $\Sigma = \{0, 1, P, F\}$. An input is said to be *accepted* by the formula if, after performing the calculations in the gates, the value received at the root of the tree is not "F". We restrict the input variables to $\{0, 1\}$, although it is easy to see that the following argument holds also if we allow other values to the input variables (and also if we change the acceptance condition to the value at the root having to be "P").

Definition 7.1 *The* balancing gate *is the gate that receives two inputs from Σ and outputs the following.*

- *For $(0, 0)$ the output is 0 and for $(1, 1)$ the output is 1.*
- *For $(1, 0)$ and $(0, 1)$ the output is P.*
- *For (P, P) the output is P,*
- *For anything else the output is F.*

For a fixed $h > 0$, the balancing formula *of height h is the formula defined by the following.*

- *The tree is the full balanced binary tree of height h with variables at the leaves, and hence there are 2^h variables.*
- *All gates are set to the balancing gate.*
- *The formula accepts if the value output at the root is not "F".*

We denote the variables of the formula in their order by $x_0, \ldots, x_{2^h-1}$. The following is easy.

Lemma 7.2 *An assignment $a_0 \in \{0, 1\}, \ldots, a_{2^h-1} \in \{0, 1\}$ to $x_0, \ldots, x_{2^h-1}$ is accepted by the formula if and only if for every $0 < k \leq h$ and every $0 \leq i < 2^{h-k}$, the number of 1 values in $a_{i2^k}, \ldots, a_{(i+1)2^k-1}$ is either 0, 2^k or 2^{k-1}.*

Proof. Denote the number of 1 values in variables descending from a gate u by $\mathrm{num}_1(u)$. Note that $a_{i2^k}, \ldots, a_{(i+1)2^k-1}$ are the set of descendant leaves of a single vertex, denote it by v. We prove by induction on k that:

- $\mathrm{num}_1(v) = 0$ if and only if the value of v is 0.
- $\mathrm{num}_1(v) = 2^k$ if and only if the value of v is 1.
- If the value of v is P then $\mathrm{num}_1(v) = 2^{k-1}$.
- If $\mathrm{num}_1(v) \notin \{0, 2^{k-1}, 2^k\}$ then the value of v is F.

For $k = 1$ we have the two inputs of v, and by the definition of the balancing gate the claim follows.

For $k > 1$, if at least one of the children of v evaluates to F then so is v (and so does the entire formula) and by the induction hypothesis one of the descendants of its children doesn't have the correct number of 1 values. If neither of them evaluates to F then by the induction hypothesis for both children of v, denoted u, w, we have that $\mathrm{num}_1(u), \mathrm{num}_1(w) \in \{0, 2^{k-2}, 2^{k-1}\}$ and that this determines their value. If $\mathrm{num}_1(w) = \mathrm{num}_1(u) = 0$ then they both evaluate to 0 and so does v. Similarly, if $\mathrm{num}_1(w) = \mathrm{num}_1(u) = 2^{k-1}$ then both evaluate to 1 and so does v. If $\mathrm{num}_1(u) = 2^{k-1}$ and $\mathrm{num}_1(w) = 0$, then u evaluates to 1 and w to 0, and indeed v evaluates to P (and similarly for the symmetric case). If $\mathrm{num}_1(u) = \mathrm{num}_1(w) = 2^{k-2}$, then both evaluate to P and so does v. The remaining case is $\mathrm{num}_1(u) \in \{0, 2^{k-1}\}$ and $\mathrm{num}_1(w) = 2^{k-2}$ (and the symmetric case). Here the induction hypothesis and the definition of the balancing gate implies that v evaluates to F and the formula is unsatisfied, while the interval of all descendant variables of v does not have the correct number of 1 values. ■

In other words, for every satisfying assignment every "binary search interval" is either all 0, or all 1, or has the same number of 0 and 1. This will allow us to easily prove that certain inputs are far from satisfying the property.

7.2 Two distributions

We now define two distributions, one over satisfying inputs and the other over far inputs.

Definition 7.3 *The distribution D_Y is defined by the following process.*

- *Uniformly pick $2 \leq k \leq h$.*
- *For every $0 \leq i < 2^{h-k}$, independently pick either $(y_{i,0}, y_{i,1}) = (0,1)$ or $(y_{i,0}, y_{i,1}) = (1,0)$ (each with probability $1/2$).*
- *For every $0 \leq i < 2^{h-k}$, set*

$$x_{i2^k} = \cdots = x_{i2^k+2^{k-1}-1} = y_{i,0}; \qquad x_{i2^k+2^{k-1}} = \cdots = x_{(i+1)2^k-1} = y_{i,1}.$$

Definition 7.4 *The distribution D_N is defined by the following process.*

- *Uniformly pick $2 \leq k \leq h$.*
- *For every $0 \leq i < 2^{h-k}$, independently choose $(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$ to have either one 1 and three 0 or one 0 and three 1 (each of the 8 possibilities with probability $1/8$).*
- *For every $0 \leq i < 2^{h-k}$, set*

$$x_{i2^k} = \cdots = x_{i2^k+2^{k-2}-1} = z_{i,0}; \qquad x_{i2^k+2^{k-2}} = \cdots = x_{i2^k+2^{k-1}-1} = z_{i,1};$$

$$x_{i2^k+2^{k-1}} = \cdots = x_{i2^k+2^{k-1}+2^{k-2}-1} = z_{i,2}; \qquad x_{i2^k+2^{k-1}+2^{k-2}} = \cdots = x_{(i+1)2^k-1} = z_{i,3}.$$

It is easier to illustrate this by considering the calculation that results from the distributions. In both distributions we can think of a randomly selected level k (counted from the bottom, where the leaf level 0 and the level above it, 1, are never selected). In D_Y, the output of all gates at or above level k is "P", while the inputs to every gate at level k will be either $(0,1)$ or $(1,0)$, chosen uniformly at random.

In D_N all gates at level k will output "F" (note however that we cannot query a gate output directly); looking two levels below, every gate as above holds the result from a quadruple chosen uniformly from the 8 choices described in the definition of D_N (the quadruple $(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$). At level $k-2$ or lower the gate outputs are 0 and 1 and their distribution resembles very much the distribution as in the case for D_Y, as long as we cannot "focus" on the transition level k. This is formalized in terms of lowest common ancestors below.

Lemma 7.5 *Let $Q \subset \{1, \ldots, 2^h\}$ be a set of queries, and let $H \subset \{0, \ldots, h\}$ be the set of levels containing lowest common ancestors of subsets of Q. Conditioned on neither k nor $k-1$ being in H, both D_Y and D_N induce exactly the same distribution over the outcome of querying Q.*

Proof. Let us condition the two distributions on a specific value of k satisfying the above. For two queries $q, q' \in Q$ whose lowest common ancestor is on a level below $k-1$, with probability 1 they will receive the exact same value (this holds for both D_N and D_Y). The reason is clear from the construction – their values will come from the same $y_{i,j}$ or $z_{i,j}$.

Now let Q' contain one representative from every set of queries in Q that must receive the same value by the above argument. For any $q, q' \in Q'$, their lowest common ancestor is on a level above k. For D_Y it means that x_q takes its value from some $y_{i,j}$ and $x_{q'}$ takes its value from some $y_{i',j'}$ where $i \neq i'$. Because each pair $(y_{i,0}, y_{i,1})$ is chosen independently from all other pairs, this means that the outcome of the queries in Q' is uniformly distributed among all $2^{|Q'|}$ possibilities. The same argument (with $z_{i,j}$ and $z_{i',j'}$ instead of $y_{i,j}$ and $y_{i',j'}$) holds for D_N. Hence the distribution of outcomes over Q' is the same for both distributions, and by extension this holds over Q. ■

On the other hand, the two distributions are very different with respect to satisfying the formula.

Lemma 7.6 *An input chosen according to D_Y always satisfies the balancing formula, while an input chosen according to D_N is always $1/4$-far from satisfying it.*

Proof. By Lemma 7.2, the assignment constructed in D_Y will always be satisfied. This is since for every vertex in a level lower than k, all of its descendant variables will be of the same value, while for every vertex in level k or above, exactly half of the variables will have each value.

Note that in an input constructed according to D_N, every vertex at level k has one quarter of its descendant variables of one value, while the rest are of the other value. By averaging, if one were to change less than $1/4$ of the input values, we would have a vertex v at level k for which less than $1/4$ of the values of its descendant variables were changed. This means that v cannot satisfy the requirements in Lemma 7.2, and therefor it and hence the entire formula evaluate to F. ■

7.3 Proving non-testability

We use here the following common application of Yao's method (see e.g. [6]).

Lemma 7.7 *If D_Y is a distribution over satisfying inputs and D_N is a distribution over ϵ-far inputs, such that for any fixed set of queries Q with $|Q| \leq l$ the probability distributions over the outcomes differ by less than $\frac{1}{3}$ (in the variation distance norm) for D_Y and D_N, then there is no non-adaptive ϵ-test for the property that makes at most l queries (1-sided or 2-sided).*

This allows us to conclude the proof.

Theorem 7.8 *Testing for being a satisfying assignment of the balancing formula of height h requires at least $\Omega(h)$ queries for a non-adaptive test and $\Omega(\log h)$ queries for a possibly adaptive one.*

Proof. We note that for any set of queries Q, the size of the set of lowest common ancestors (outside Q itself) is less than $|Q|$, and hence (in the notation of Lemma 7.5) we have $|H| \leq |Q|$. If $|Q| = o(h)$, then the event of Lemma 7.5 happens with probability $1 - o(1)$, and hence the variation distance between the two (unconditional) distributions over outcomes is $o(1)$. Together with Lemma 7.6 this fulfills the conditions for Lemma 7.7 for concluding the proof for non-adaptive algorithms.

For adaptive algorithms the bound follows by the standard procedure that makes an adaptive algorithm into a non-adaptive one at an exponential cost, by querying in advance the algorithm's entire decision tree given its internal coin tosses. ■

7.4 An untestable 5-valued monotone formula

While the lower bound given above uses a gate which is highly non-monotone, we can also give a similar construction where the alphabet is of size 5 and the gates are monotone (that is, where increasing any input of the gate according to the order of the alphabet does not decrease its input).

Instead of just "$\{1, \ldots, 5\}$" we denote our alphabet by $\Sigma = \{0, F_0, P, F_1, 1\}$ in that order. We will restrict the input variables to $\{0, 1\}$, although it is not hard to generalize to the case where the input variables may take any value in the alphabet. At first we analyze a formula that has a non-monotone satisfying condition.

Definition 7.9 *The* monotone balancing gate *is the gate that receives two inputs from Σ and outputs the following.*

- *For $(0, 0)$ the output is 0 and for $(1, 1)$ the output is 1.*
- *For $(1, 0)$ and $(0, 1)$ the output is P.*
- *For (P, P) the output is P.*
- *For $(0, P)$ and $(P, 0)$ the output is F_0.*
- *For $(1, P)$ and $(P, 1)$ the output is F_1.*
- *For (P, F_0), (F_0, P), $(F_0, 0)$, $(0, F_0)$ and (F_0, F_0) the output is F_0.*
- *For $(F_0, 1)$ and $(1, F_0)$ the output is F_1.*

- *For any pair of inputs containing F_1, the output is F_1.*

For a fixed $h > 0$, the almost-monotone balancing formula *of height h is the formula defined by the following.*

- *The tree is the full balanced binary tree of height h with variables at the leaves, and hence there are 2^h variables.*
- *All gates are set to the monotone balancing gate.*
- *The formula accepts if the value output at the root is not "F_0" or "F_1".*

The following observation is easy by just running over all possible outcomes of the gate.

Observation 7.10 *The monotone balancing gate is monotone. Additionally, if the values F_0 and F_1 are unified then the gate is still well-defined, and is isomorphic to the 4-valued balancing gate.*

In particular, the above observation implies that the almost-monotone balancing formula has the same property testing lower bound as that of the balancing formula, using the same proof with the same distributions D_Y and D_N. However, we would like a completely monotone formula. For that we use a monotone decreasing acceptance condition; we note that a formula with a monotone increasing acceptance condition can be obtained from it by just "reversing" the order over the alphabet.

Definition 7.11 *The* monotone sub-balancing formula *is defined the same as the almost-monotone balancing formula, with the exception that the formula accepts if and only if the value output at the root is not F_1 or 1.*

By Observation 7.10, the distribution D_Y is also supported by inputs satisfying the monotone sub-balancing formula. To analyze D_N, note the following.

Lemma 7.12 *An assignment $a_0 \in \{0,1\}, \ldots, a_{2^h-1} \in \{0,1\}$ to $x_0, \ldots, x_{2^h-1}$, for which for some $0 < k \leq h$ and some $0 \leq i < 2^{h-k}$ the number of 1 values in $a_{i2^k}, \ldots, a_{(i+1)2^k-1}$ is more than 2^{k-1} and less than 2^k, cannot be accepted by the formula.*

Proof. We set u to be the gate whose descendant variables are exactly $a_{i2^k}, \ldots, a_{(i+1)2^k-1}$. We first note that it is enough to prove that u evaluates to F_1, because then by the definition of the gates the root will also evaluate to F_1. We then use induction over k, while referring to Observation 7.10 and the proof of Lemma 7.2. The base case $k = 1$ is true because then no assignment satisfies the conditions of the lemma.

If any of the two children of u evaluates to F_1 then we are also done by the definition of the gate. The only other possible scenario (using induction) is when one of the children v of u must evaluate to 1, and hence all of its 2^{k-1} descendant variables are 1, while for the other child w of u some of the descendant variables are 0 and some are 1. But this means that w does not evaluate to either 0 or 1, which again means that u evaluates to F_1. ■

This yields the following.

Lemma 7.13 *With probability $1-o(1)$, an input chosen according to D_N will be $1/12$-far from satisfying the monotone sub-balancing formula.*

Proof. This is almost immediate from Lemma 7.12, as a large deviation inequality implies that with probability $1 - o(1)$, more than $1/3$ of the quadruples $(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$ as per the definition of D_N will have three 1's and one 0. ■

Now we can prove a final lower bound.

Theorem 7.14 *Testing for being a satisfying assignment of the monotone sub-balancing formula of height h requires at least $\Omega(h)$ queries for a non-adaptive test and $\Omega(\log h)$ queries for a possibly adaptive one.*

Proof. This follows exactly the proof of the lower bound for the balancing formula. Due to Observation 7.10 and Lemma 7.13 we can use the same D_Y and D_N, since the $o(1)$ probability of D_N not producing a far input makes no essential difference for the use of Yao's method. ■

References

[1] Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM J. Comput.*, 30(6):1842–1862, 2000.

[2] Eli Ben-Sasson, Prahladh Harsha, Oded Lachish, and Arie Matsliah. Sound 3-query pcpps are long. *ACM Trans. Comput. Theory*, 1:7:1–7:49, September 2009.

[3] Eli Ben-Sasson, Prahladh Harsha, and Sofya Raskhodnikova. Some 3CNF properties are hard to test. *SIAM J. Comput.*, 35(1):1–21, 2005.

[4] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[5] Sourav Chakraborty, Eldar Fischer, Oded Lachish, Arie Matsliah, and Ilan Newman. Testing *st* -connectivity. In *APPROX-RANDOM*, pages 380–394, 2007.

[6] Eldar Fischer. The art of uninformed decisions: A primer to property testing. *Current Trends in Theoretical Computer Science: The Challenge of the New Century*, I:229–264, 2004.

[7] Eldar Fischer, Oded Lachish, Ilan Newman, Arie Matsliah, and Orly Yahalom. On the query complexity of testing orientations for being eulerian. *Transactions on Algorithms*, to appear.

[8] Eldar Fischer, Ilan Newman, and Jiri Sgall. Functions that have read-twice constant width branching programs are not necessarily testable. *Random Struct. Algorithms*, 24(2):175–193, 2004.

[9] Eldar Fischer and Orly Yahalom. Testing convexity properties of tree colorings. *Algorithmica*, 60(4):766–805, 2011.

[10] Oded Goldreich. A brief introduction to property testing. In Oded Goldreich, editor, *Property Testing*, pages 1–5. Springer-Verlag, 2010.

[11] Oded Goldreich, Shaffi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45:653–750, July 1998.

[12] Shirley Halevy, Oded Lachish, Ilan Newman, and Dekel Tsur. Testing orientation properties. *Electronic Colloquium on Computational Complexity (ECCC)*, (153), 2005.

[13] Shirley Halevy, Oded Lachish, Ilan Newman, and Dekel Tsur. Testing properties of constraint-graphs. In *IEEE Conference on Computational Complexity*, pages 264–277, 2007.

[14] Ilan Newman. Testing membership in languages that have small width branching programs. *SIAM J. Comput.*, 31(5):1557–1570, 2002.

[15] Ilan Newman. Property testing of massively parametrized problems - a survey. In Oded Goldreich, editor, *Property Testing*, pages 142–157. Springer-Verlag, 2010.

[16] Dana Ron. Property testing: A learning theory perspective. *Found. Trends Mach. Learn.*, 1:307–402, March 2008.

[17] Dana Ron. *Algorithmic and Analysis Techniques in Property Testing.* 2010.

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