Testing Formula Satisfaction*

Eldar Fischer[1], Yonatan Goldhirsh[1], and Oded Lachish[2]

[1] Department of Computer Science, Technion, Haifa 32000, Israel.
{eldar,jongold}@cs.technion.ac.il
[2] Birkbeck, University of London, London, UK. oded@dcs.bbk.ac.uk

Abstract. We study the query complexity of testing for properties defined by read once formulae, 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 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 formulae 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 formulae over non-Boolean alphabets; specifically we construct a property defined by a read-once arity 2 (non-Boolean) formula over alphabets of size 4, such that any 1/4-test for it requires a number of queries depending on the formula size.

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 from 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

* Research supported in part by an ERC-2007-StG grant number 202405.

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).

Property-Testing was first addressed by Blum, Luby and Rubinfeld [4], and most of its general notions were first formulated by Rubinfeld and Sudan [19], 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 [12]. Since then Property-Testing has attracted significant attention leading to many results. For surveys see [6], [11], [17], [18].

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 [12]) 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 [15], where properties accepted by a general read-once oblivious branching program are investigated. In such a setting a general dependency on the parameter is inadmissible, and indeed in [15] 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. 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 [13, 5, 8, 10] and the survey [16], 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, 15, 9]. In [3] the study of formulae 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 [14] 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 formulae. 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.

We prove a number of results. Due to space considerations we provide in this extended abstract only the proofs of the most general of them, while the rest can be found in the full version [7]. Section 2 contains preliminaries. First we define the properties 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 formulae involving arbitrary gates of bounded arity. We suppply a brief analysis of the running times of the algorithms in Section 4. One interesting implication of the testability results presented here, is that any read-once formula accepting an untestable Boolean property must use unbounded arity gates other than And/Or.

More results are available in the full version [7]: For such formula involving only monotone gates, we provide an *estimation* algorithm, 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 $\eta + \epsilon$. We also show that when restricted to And/Or gates, we can provide a test whose query complexity is quasipolynomial in ϵ. On the other hand, we prove 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.

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 $\texttt{out-deg}(v) = \{u \in V \mid (u, v) \in E\}$. A *path* is a tuple $(u_1, \ldots, u_k) \in |V|^k$ such that $u_1, \ldots, u_k$ are all distinct and $(u_i, u_{i+1}) \in E$ for every $i \in [k-1]$. 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 an unique path 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 Formulae, 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 1 (Formula). *A* Read-Once 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 formulae. 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 is the *formula size*.

Definition 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 $|\Phi_u| = |X_u|$.*

Definition 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 ($\kappa_v \in B$) we set $\sigma(v) = \kappa_v(\sigma(u_1), \ldots, \sigma(u_k))$, where $\texttt{Children}(v) = (u_1, \ldots, u_k)$.*

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 formulae, 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, then 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 subset S of assignments to Φ we set $\texttt{farness}_\Phi(\sigma, S) = \min\{\texttt{farness}_\Phi(\sigma, \sigma') \mid \sigma' \in S\}$. 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 formulae 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 with as few queries as possible, and farness is measured with respect to the fraction of alterations it requires.

Definition 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. 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*(Φ)*.*

We can now summarize the contributions of the paper in the following theorem. The first item is proved in this extended abstract, while the rest can be found in the full version [7]:

Theorem 5 (Main Theorem). *The following theorems all hold:*

- *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*(Φ) *with* $q = \exp(\text{poly}(\epsilon^{-1}))$*.*
- *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*(Φ) *with* $q = \exp(\text{poly}(\epsilon^{-1}))$*.*
- *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* $q = \epsilon^{O(\log \epsilon)}$*.*
- *There exists an infinite family of* 4 *valued read-once formulae* Φ *such that there is no non-adaptive* (ϵ, q)*-test for SAT*(Φ) *with* $q = O(\texttt{depth}(\Phi))$*, and no adaptive* (ϵ, q)*-test for SAT*(Φ) *with* $q = O(\log(\texttt{depth}(\Phi)))$*.*

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

Definition 6. *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* $T_i \in T$ *is a subset* $T_i \subseteq [n]$*, there exists no two different terms* $T_i, T_j \in T$ *such that* $T_i \subset T_j$*, 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 7 *Any monotone boolean function* $f : \{0,1\}^n \to \{0,1\}$ *has a unique* mDNF T*.*

Definition 8. *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* $a, b \in \{0,1\}$*.*

Forceful variables are variables that cause "Or-like" or "And-like" behavior in the gate.

Definition 9. *A vertex* $v \in V$ *is called* unforceable *if no child of* v *is forceful.*

Definition 10 (k-x-Basic formula). *A read-once formula Φ is* k-x-basic *if it is Boolean, all the functions in B have arity at least 2, and are either of arity at most k and unforceable, or $\wedge$ or $\vee$ of arity at least 2, and Φ satisfies the following. There is no $v \in V$ such that $\kappa_v \in \{0,1\}$. No $\wedge$ is a child of a $\wedge$ and no $\vee$ is a child of a $\vee$. Any variable may appear at most once in a leaf, either positively or negated.*

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

Lemma 11. *Every read-once formula Φ with gates of arity at most k has an equivalent k-x-basic formula Φ'.*

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 negation of the gate u when it is assumed that $\sigma(\kappa_v) = a$. After performing this transformation sufficiently many times we have no forceable gates left.

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 $\vee$ can be replaced with an $\wedge$ 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 a $\wedge$ can be replaced with an $\vee$ 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.

Note that $\vee$ and $\wedge$ gates are very much forceable.

2.3 Observations about subformulae and farness

Definition 12 (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 13 (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 $\texttt{depth}(\Phi) = \max\{\texttt{depth}_\Phi(u) \mid u \in V\}$.*

Observation 14 *Let $v \in V$ be such that either $\kappa_v \equiv \vee$ and $b = 0$ or $\kappa_v \equiv \wedge$ and $b = 1$, and $\texttt{farness}(\sigma, SAT(\Phi_v = b)) \geq \epsilon$. For every $1 > \alpha > 0$ there exists $S \subseteq \texttt{Children}(v)$ such that $\sum_{s \in S} |\Phi_s| \geq \epsilon\alpha^2|\Phi|$ and $\texttt{farness}(\sigma, SAT(\Phi_w = b)) \geq \epsilon(1-\alpha)$ for every $w \in S$. Furthermore, the exists a child $u \in \texttt{Children}(v)$ such that $\texttt{farness}(\sigma, SAT(\Phi_u = b)) \geq \epsilon$.*

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 $\sum_{t\in T} |\Phi_t| < \epsilon\alpha^2|\Phi|$ then the distance from having Φ_v evaluate to b is at most $\epsilon\alpha^2 + \epsilon(1-\alpha)(1-\alpha) < \epsilon$, which contradicts the assumption.

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

Observation 15 *Let $v \in V$ be 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 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 $u \in \texttt{Children}(v) \setminus \{h(v)\}$ is of weight at most $1/2$, and therefore if Φ_u were 2ϵ-close to being evaluated to b, Φ_v was ϵ-close to being evaluated to b.

2.4 Heavy and Light Children in General Gates

Definition 16. *Given a 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 it 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.*

Lemma 17. *If an unforceable vertex v 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 $SAT(\Phi_v = 0)$.*

Observation 18 *If $\kappa_u \not\equiv \wedge$ and $\kappa_u \notin X$ and σ is ϵ-far from $SAT(\Phi_u = b)$, then it must have at least two heavy children.*

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.

Lemma 19. *The depth of recursion in Algorithm 1 is at most* $16(4k/\epsilon)^k \log(\epsilon^{-1})$.

Algorithm 1

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: $\quad y \longleftarrow$ "true"
6: $\quad$ **for** $i = 1$ to $l = 32(2k/\epsilon)^{2k} \log(\delta^{-1})$ **do**
7: $\quad\quad u \longleftarrow$ 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: $\quad\quad y \longleftarrow y \wedge$ Algorithm $1(\Phi_u, (\epsilon(1 - (2k/\epsilon)^{-k}/16)), \sigma, \delta/2, b)$
9: $\quad$ **return** y
10: **if** $(\kappa_r \equiv \wedge$ and $b = 0)$ or $(\kappa_r \equiv \vee$ and $b = 1)$ **then**
11: $\quad$ **if** there exists a child of weight less than ϵ **then return** "true"
12: $\quad y \longleftarrow$ "false"
13: $\quad$ **for all** $u \in \texttt{Children}(r)$ **do** $y \longleftarrow y \vee$ Algorithm $1(\Phi_u, (\epsilon(1 + \epsilon)), \sigma, \epsilon\delta/2, b)$
14: $\quad$ **return** y
15: **if** there is a child of weight at least $1 - \epsilon$ **then return** "true"
16: **for all** $u \in \texttt{Children}(r)$ **do**
17: $\quad y_u^0 \longleftarrow$ Algorithm $1(\Phi_u, (\epsilon(1 + (4k/\epsilon)^{-k})), \sigma, \delta/2k, 0)$
18: $\quad y_u^1 \longleftarrow$ Algorithm $1(\Phi_u, (\epsilon(1 + (4k/\epsilon)^{-k})), \sigma, \delta/2k, 1)$
19: **if** There exists a string $x \in \{0, 1\}^k$ such that κ_r on x would evaluate to b and for all $u \in \texttt{Children}(r)$ we have $y_u^{x_u}$ equal to "true" **then return** "true" **else return** "false"

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

All recursive calls occur in Lines 8, 13, 17 and 18.

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, an increase of two in the depth results in an increase of the farness parameter from ϵ to at least $(\epsilon(1 - (2k/\epsilon)^{-k}/16))(\epsilon(1 + (4k/\epsilon)^{-k})) \geq \epsilon(1 + (4k/\epsilon)^{-k}/16)$. Thus in recursive calls of depth $16(4k/\epsilon)^k \log(\epsilon^{-1})$ the farness parameter exceeds 1 and the call returns without making any further calls.

Lemma 20. *Algorithm 1 uses at most* $\epsilon^{-480(4k/\epsilon)^{k+3} \log\log(\delta^{-1})}$ *queries.*

The proof is standard and thus deferred to the full version. It follows from Lemma 19.

Lemma 21. *If Φ on σ evaluates to b then Algorithm 1 returns "true" with probability* 1.

The proof follows by an induction on formula depth and is deferred to the full version.

Lemma 22. *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" if σ is 0-far from getting Φ to 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 14 we get that there exists $T \subseteq \texttt{Children}(r)$ such that $\sum_{t\in T} |\Phi_t| \geq |\Phi|\epsilon((2k/\epsilon)^{-2k}/16)$ and each Φ_t is $\epsilon(1-(2k/\epsilon)^{-k}/16)$-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 $\epsilon((2k/\epsilon)^{-2k}/16)$. Since this happens at least $32(2k/\epsilon)^{2k}\log(\delta^{-1})$ 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 $\epsilon(1-(2k/\epsilon)^{-k}/16)$ will return "false" with probability at least $1-\delta/2$, which will evetually 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 15 implies that all children are of weight at least ϵ, and therefore the conditions of Line 11 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 Observation 17, since Φ is ϵ-far from being satisfied the condition in Line 15 is not triggered. If the algorithm returned "true" then it must be that the condition in Line 19 is satisfied. If there exists some heavy $u \in \texttt{Children}(r)$ such that y_u^b is "true" and y_u^{1-b} is "false", then by Lemma 21 the formula Φ_u does evaluate to b and the string in x must be such that $x_u = b$. For the rest of the children of r, assuming the calls succeeded, each the subformula rooted in v is $(\epsilon(1+(4k/\epsilon)^{-k}))$-close to evaluate to x_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 x 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 $(4k/\epsilon)^{-\ell+1}$, 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 x 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 x is not necessarily the one found in Line 19.

Thus we have found that finding an assignment x in Line 19, 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 the $2k$ recursive calls err, which by the union bound is at most δ, and the algorithm will return "false" correctly with probability at least $1-\delta$.

4 The Computational Complexity of the Testers and Estimator

There are two parts to analyzing the computational 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 parameter size.

In our case, the preprocessing part would need to take a k-ary formula and convert it to the k-x-basic form corresponding to the algorithm that we run. We would also need to put the formula in a data structure that allows the following operations to be performed as quickly 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".

It is not hard to see that both constructing the normal form and calculating the above additional data can be done very efficiently. Furthermore, the only part that depends on epsilon is designating the light children, but this can also be done "for all epsilon" at a low cost (by storing the range of ϵ for every positive ℓ).

The second part is analyzing the running time complexity of the 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). 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.

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, Yonatan Goldhirsh, and Oded Lachish. Testing formula satisfaction. arXiv:1204.3413v1 [cs.DS].
8. Eldar Fischer, Oded Lachish, Ilan Newman, Arie Matsliah, and Orly Yahalom. On the query complexity of testing orientations for being eulerian. *TALG*, to appear.
9. 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.
10. Eldar Fischer and Orly Yahalom. Testing convexity properties of tree colorings. *Algorithmica*, 60(4):766–805, 2011.
11. Oded Goldreich. A brief introduction to property testing. In Oded Goldreich, editor, *Property Testing*, pages 1–5. Springer-Verlag, 2010.
12. Oded Goldreich, Shaffi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45:653–750, July 1998.
13. Shirley Halevy, Oded Lachish, Ilan Newman, and Dekel Tsur. Testing orientation properties. *(ECCC)*, (153), 2005.
14. Shirley Halevy, Oded Lachish, Ilan Newman, and Dekel Tsur. Testing properties of constraint-graphs. In *IEEE Conference on Computational Complexity*, 2007.
15. Ilan Newman. Testing membership in languages that have small width branching programs. *SIAM J. Comput.*, 31(5):1557–1570, 2002.
16. Ilan Newman. Property testing of massively parametrized problems - a survey. In Oded Goldreich, editor, *Property Testing*, pages 142–157. Springer-Verlag, 2010.
17. Dana Ron. Property testing: A learning theory perspective. *Found. Trends Mach. Learn.*, 1:307–402, March 2008.
18. Dana Ron. *Algorithmic and Analysis Techniques in Property Testing.* 2010.
19. Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.