On Self-Testing without the Generator Bottleneck

S. Ravikumar* D. Sivakumar†

Abstract

Suppose P is a program designed to compute a linear function f on a group G. The task of *self-testing* f, that is, testing if P computes f correctly on most inputs, usually involves checking if P computes f correctly on all the generators of G. Recently, F. Ergün presented self-testers that avoid this *generator bottleneck* for specific functions. In this paper, we generalize Ergün's results, and extend them to a much larger class of functions. Our results give efficient self-testers for polynomial differentiation, integration, arithmetic in large finite field extensions, and constant-degree polynomials over large rings.

1 Introduction

The notions of program result-checking, self-testing, and self-correcting [BK89, Lip91, BLR93] have proved to be very useful concepts. The theoretical developments in this area are at the heart of the recent breakthrough results on probabilistically checkable proofs and the subsequent results that show non-approximability of hard combinatorial problems. From a practical viewpoint, these methods offer realistic and efficient tools for program verification. See the survey articles by Blum and Wasserman [BW94] and by Madhu Sudan [Sud94] for interesting expositions and pointers.

Suppose we are given a program P designed to compute a function f. Informally, a *self-tester* for f enables us to estimate the fraction of the inputs on which P computes f correctly. Alternatively, the role of a self-tester is to either certify that P computes f correctly on most inputs, or reject P as largely erroneous. A *result-checker* for a function f takes a program P and an input q and outputs "CORRECT" if P always computes f correctly, and outputs "WRONG" if $P(q) \neq f(q)$. Given a program P that computes f correctly on most inputs, a *self-corrector* for f is a program that uses P as an oracle and computes f correctly on every input with high probability. The existence of a self-tester/corrector pair implies the existence of a result-checker. Blum and Kannan [BK89] and Blum, Luby, and Rubinfeld [BLR93] have laid out the minimal requirements for self-testers and result-checkers. These programs should not compute f directly or require the value of f at too many locations, that is, they should be "different" from the program being tested, and should be at least as efficient as the fastest algorithm for f.

*Department of Computer Science, Cornell University, Ithaca, NY 14853. Email: ravi@cs.cornell.edu.
Research supported in part by ONR Young Investigator Award N00014-93-1-0590.

†Department of Computer Science, State Univ. of New York at Buffalo, Buffalo, NY 14260. Email: sivak-d@cs.buffalo.edu.
Research supported in part by NSF grant CCR-9409104.

Self-testers are usually built for a *class* $\mathcal{F}$ of functions by identifying certain properties of the class $\mathcal{F}$ that are *robust*. Informally, property Π is said to be a robust characterization of a function class $\mathcal{F}$ if the following two conditions hold: (1) every $f \in \mathcal{F}$ satisfies Π, and (2) if P is a function (program) that satisfies Π on most inputs, then P must agree with some $g \in \mathcal{F}$ on most inputs. For example, consider the family $\mathcal{F}$ of linear functions on a group $(G, \circ)$. Here the linearity property $\Pi(f) \equiv (\forall p, q \in G)[f(p \circ q) = f(p) \circ f(q)]$ is robust: every $f \in \mathcal{F}$ satisfies Π, and moreover, if some program P satisfies Π for most $p, q \in G$, then there is a linear function $g \in \mathcal{F}$ with which P agrees most of the time. Moreover, it is often the case that given the faulty program P, a "self-corrected" version P_{sc} of P can be built that correctly computes (with high probability) the "ghost function" g on every input.

Various linear and low-degree polynomials, and functions described by functional equations have been shown to have self-testers and/or self-correctors [BLR93, Lip91, GLR$^+$91, RS92, RS93, Rub94]. Using the notions of robust characterizations and self-correcting, the self-testing scheme for a program P that purports to compute a linear function f can be described as follows. First check if the program P satisfies the linearity property on most inputs $p, q \in G$. By the robustness of linearity, this implies that there is a linear function g on G that can be computed by the self-corrector P_{sc} of P. Using P_{sc}, verify that g agrees with f on the generators of the group G. If P passes these tests, it is then shown by induction that g agrees with f on all inputs. Since P agrees with g on most inputs, it follows that P computes f correctly on most inputs.

If the group G has many generators, for example, if the group is a vector space of large dimension, then testing if P_{sc} agrees with f on the generators adds to the overhead in the testing process. This is called the *generator bottleneck*. It is desirable to build self-testers that make only $O(1)$ calls to the program P being tested. Such self-testers are called *efficient* in [BLR93]. The problem of generator bottleneck was first addressed by Ergün [Erg95], who shows how to obtain efficient self-testers for specific functions like Fourier transforms, polynomial multiplication, etc.

We generalize and extend Ergün's idea to a larger class of functions. First we show that the technique can be applied in a more natural and general setting, namely that of vector spaces. We prove a general theorem that gives a condition for a linear function f to be efficiently self-testable on a large vector space. This approach has an extra benefit: it allows us to give conditions for efficient self-testing in terms of the matrices of linear operators. Applying this result, we obtain self-testers for many functions that include polynomial differentiation, integration, the polynomial "mod" function, etc., besides Fourier transforms. We then extend this to the case of multilinear functions, and derive the result for polynomial multiplication as a consequence. Another application we give is for large finite fields: we show that multilinear functions over finite field extensions of dimension n can be efficiently self-tested with $O(1)$ calls, independent of the dimension n.

The next step we take is to extend the result to the case of nonlinear functions. We give the first self-testers for exponentiation functions that avoid the generator bottleneck. Consider the function that computes the square of a polynomial over a finite field: $f(p) = p^2$. Here we do not have the nice property of linearity that was crucial in the proof for the linear functions. Instead, we use the fact that the interpolation identity for polynomials gives a robust characterization. We exhibit a self-tester for the function $f(p) = p^d$ that makes $O(d)$

calls to the program being tested.

The passage from constant degree exponentiation to constant degree polynomials is much harder. First we show a reduction from multiplication to the computation of low-degree polynomials. Using this reduction and the notion of result-checker, we construct a self-tester for degree d polynomials over finite field extensions of dimension n that make $O(2^d)$ calls to the program being tested.

2 Linear Functions over Vector Spaces

In this section, we generalize the idea of [Erg95] to the case of linear functions on a vector space. We begin with the basic definitions.

Definition 1 *Let f be a function on a domain D. A (γ, ϵ)-**self-tester** for f is a probabilistic program T that, given a program P for f, and a parameter δ, satisfies the following conditions.*

- $\Pr_{X \in D}[P(X) \neq f(X)] \leq \gamma \Rightarrow \Pr[T^P$ *outputs "PASS"* $] \geq 1 - \delta$.
- $\Pr_{X \in D}[P(X) \neq f(X)] \geq \epsilon \Rightarrow \Pr[T^P$ *outputs "FAIL"* $] \geq 1 - \delta$.

*A ϵ-**self-corrector** for P is a probabilistic program R that takes an input Y and a parameter δ, and satisfies the following condition:*

- $\Pr_{X \in D}[P(X) \neq f(X)] \geq \epsilon \Rightarrow \Pr[R^P(Y) = f(Y)] \geq 1 - \delta$.

*A **checker** (or result-checker) for f is a probabilistic program C that, given a program P for f, an input Y, and a parameter δ, satisfies the following conditions:*

- $\Pr_{X \in D}[P(X) = f(X)] = 1 \Rightarrow \Pr[C^P(Y)$ *outputs "CORRECT"* $] = 1$.
- $P(Y) \neq f(Y) \Rightarrow \Pr[C^P(Y)$ *outputs "WRONG"* $] \geq 1 - \delta$.

Let V be a vector space of finite dimension n over a field $\mathbb{K}$, and let f be a function from V into a ring R. We are interested in building a self-tester for the case where f is a *linear* function, that is, $f(c\alpha + \beta) = cf(\alpha) + f(\beta)$ for all $\alpha, \beta \in V$ and $c \in \mathbb{K}$. For simplicity, we will assume that the field $\mathbb{K}$ is finite; our testers work for infinite fields as well, but additional complications arise. For example, it is not clear how to choose a random element from the field of real numbers.

The self-tester we build follows the framework of [BLR93]. For $1 \leq i \leq n$, let e_i denote the vector that has a 1 in the i-th position and 0's in the other positions. The vectors $e_1, e_2, \ldots, e_n$ form a collection of basis vectors that span V. Viewed as an abelian group under vector addition, V is generated by $e_1, \ldots, e_n$. Suppose P is a program that purports to compute f. The framework of [BLR93] comprises the following two tests:

(1) *Linearity Test*: By checking if $P(\alpha + \beta) = P(\alpha) + P(\beta)$ on many randomly chosen inputs $\alpha, \beta \in V$, ensure that P satisfies the linearity property.

It is shown in [BLR93] that the linearity property is robust, that is, if the program P passes Test (1) sufficiently often, then there is a unique linear function g that agrees with P on most inputs. Moreover, as mentioned in the Introduction, there is a self-corrector $P_{\rm sc}$ that uses P as an oracle and computes g correctly on every input (with high probability).

The second test ensures that g agrees with f on the generators.

(2) *Basis Tests*: For $1 \leq i \leq n$, verify that $g(e_i) = f(e_i)$.

Here $g(e_i)$ will be computed using the program $P_{\rm sc}$. If $P_{\rm sc}$, the self-corrector for P, passes Test (2), then one can show by induction, using the linearity of g, that g is identical to the function f.

There are two problems with this: one is that the self-tester is inefficient—if the inputs are n-element vectors, the self-tester makes $O(n)$ calls to the program, which is not desirable. The requirement in [BLR93] for an *efficient* self-tester is that it make $O(1)$ calls to the program being tested. Secondly, the self-tester needs to know the correct value of f on n different points, which is also undesirable. Our primary interest is to avoid this *generator bottleneck*. Ergün [Erg95] introduced an elegant trick that accomplishes this for specific functions. The key idea is to find an easy uniform way that "converts" one generator into the next generator. We illustrate this idea through the following example.

Let V denote the group of all degree n polynomials under addition. The elements $1, x, x^2, \ldots, x^n$ generate V. Given any generator x^k, multiplying by x gives the next generator x^{k+1}. For a polynomial $p \in V$ and a scalar $c \in \mathbb{K}$, let $E_c(p)$ denote the function that evaluates $p(c)$. Clearly E_c is linear and satisfies the simple relation $E_c(xp) = cE_c(p)$. Suppose P is a program that purports to compute E_c, and assume that P has passed Test (1) given above. Then we know that there is a "ghost linear function" g that agrees with P on most inputs. Now, rather than verify that $g(e) = f(e)$ for all generators e, we may instead verify that g satisfies the property $g(xp) = cg(p)$ everywhere. By an easy induction, this implies that g agrees with f at all the generators. By the linearity of g, it would follow that g agrees with f on all inputs.

We are now faced with the task of verifying $g(xp) = cg(p)$ for all $p \in V$, which is hopeless. Ergün [Erg95] shows that in order to verify this, it suffices to check if $g(xp) = cg(p)$ holds *almost* everywhere. That is, pick many random $p \in V$, ask the program $P_{\rm sc}$ to compute the values of $g(p)$ and $g(xp)$ in two calls, and cross-check that $g(xp) = cg(p)$ holds. In other words, the property "$(\forall p)[g(xp) = cg(p)]$" satisfies a *robustness* condition. Notice that the number of points on which the self-tester needs to know the value of f is just one, in contrast to knowing the values of f on all the n generators (as in the original approach of [BLR93]).

We note that this idea has a natural generalization to vector spaces. Let θ denote the linear operator on V that "rotates" the coordinate axes that span V; that is,

$$\theta = \left(e_2^T e_3^T \cdots e_n^T e_1^T\right)^T$$

The matrix θ defines a one-to-one correspondence from the set of basis vectors to itself: for every i, $\theta(e_i) = e_{(i+1) \bmod n}$. The computational payoff is achieved when there is a simple relation between $f(\alpha)$ and $f(\theta(\alpha))$ for all vectors $\alpha \in V$. More specifically, we show that the generator bottleneck can be avoided if there is an easily computable function h such that $f(\theta(\alpha)) = h(\theta, \alpha, f(\alpha))$ for all $\alpha \in V$. If the function f is linear, the linearity of the operator θ implies that h is always linear in its second argument in the following sense: $h(\theta, \alpha+\beta, f(\alpha+\beta)) = h(\theta, \alpha, f(\alpha)) + h(\theta, \beta, f(\beta))$. What is more important is that h be easy to compute, given just α and $f(\alpha)$.

Using this scheme, we show that besides the functions self-tested in [Erg95], many other natural functions f have a suitable candidate for h. Recall that if P passes Test (1) there is a unique linear function g that agrees with P on most inputs, and that can be computed by P_{sc} (using P as an oracle). The basis tests of [BLR93] can now be replaced by:

(2) Verify that $g(e_1) = f(e_1)$.

(3) For many randomly chosen α's, verify that $g(\theta(\alpha)) = h(\theta, \alpha, g(\alpha))$.

Theorem 1 *Suppose f is a linear function from the vector space V into a ring R, and suppose P is a program for f. Let $\epsilon < 2/9$, and suppose P satisfies the following condition:*

(1) $\Pr_{\alpha,\beta \in V}[P(\alpha+\beta) \neq P(\alpha) + P(\beta)] \leq \epsilon$.

Then the function g defined by $g(\alpha) = \text{majority}_{\beta \in V}\{P(\alpha+\beta) - P(\beta)\}$ is a linear function on V, and g agrees with P on at least $1-2\epsilon$ fraction of the inputs. Furthermore, suppose g satisfies the following conditions:

(2) $g(e_1) = f(e_1)$.

(3) $\Pr_{\alpha \in V}[g(\theta(\alpha)) \neq h(\theta, \alpha, g(\alpha))] \leq \epsilon$.

Then $g(\alpha) = f(\alpha)$ for all $\alpha \in V$.

Remarks. The above theorem only brings out two robust properties of linear functions. The fact that this yields a self-tester is presented in Theorem 3. Note that hypotheses (1), (2), and (3) above are *conditions* on P and g, not *tests* performed by a self-tester.

Proof. The proof that the function g is linear and self-corrects P is due to [BLR93] (the "2/9" in the theorem is the bound obtained there). For the rest of this proof, we will assume that g is linear and that it satisfies conditions (2) and (3) above.

We first argue that it suffices to prove that for every $\alpha \in V$, $g(\theta(\alpha)) = h(\theta, \alpha, g(\alpha))$. By condition (2), g agrees with f on the first basis vector. For $i > 1$, the basis vector e_i can be obtained by $\theta(e_{i-1})$. If g satisfies $g(\theta(\alpha)) = h(\theta, \alpha, g(\alpha))$ everywhere, it would follow that g computes f correctly on all the basis vectors. Finally, since g is linear, it computes f correctly on all of V, since the vectors in V are just linear combinations of the basis vectors.

Fix an arbitrary element $\alpha \in V$. We will show that the probability over a random $\beta \in V$ that $g(\theta(\alpha)) = h(\theta, \alpha, g(\alpha))$ is positive. Since the equality is independent of β and holds with non-zero probability, it must be true with probability 1. Now

$$
\begin{aligned}
\Pr_{\beta \in V}[g(\theta(\alpha)) &= g(\theta(\beta + \alpha - \beta)) \\
&= g(\theta(\beta) + \theta(\alpha - \beta)) \\
&= g(\theta(\beta)) + g(\theta(\alpha - \beta)) \\
&= h(\theta, \beta, g(\beta)) + h(\theta, \alpha - \beta, g(\alpha - \beta)) \\
&= h(\theta, \beta + \alpha - \beta, g(\beta) + g(\alpha - \beta)) \\
&= h(\theta, \alpha, g(\alpha))] \geq 1 - 2\epsilon > 0.
\end{aligned}
$$

The first equality in the above is just rewriting. The second equality follows from the linearity of θ. The third equality follows from the fact that g is linear. If the r.v. β is distributed uniformly in V, the r.v.'s β and $\alpha - \beta$ are distributed identically and uniformly in V. Therefore, by the assumption that g satisfies condition (3), the fourth equality fails with probability at most 2ϵ. The fifth equality uses the fact that h is linear, and the last equality uses the fact that g is linear. □

The foregoing theorem shows that if P (and g) satisfy certain conditions, then g, which can be computed using P, is identically equal to the function f. To build a self-tester, we will verify that each of these conditions holds. Since the vector space V could be very large, the self-tester cannot check these conditions exhaustively. However, by random sampling, the self-tester can *estimate* the required probabilities to a high degree of accuracy.

The self-tester takes two parameters as input: an *accuracy parameter* ϵ that specifies the conditions P is expected to meet, and a *confidence parameter* δ, which is an upper bound on the probability that the self-tester fails, and which also specifies the allowed error on the various estimates made by the self-tester. An efficient self-tester should only make $k^{O(1)}$ calls to P, where $k = \frac{1}{\epsilon} + \frac{1}{\delta}$. The self-tester we describe performs three tests, and we will apportion the error allowance δ equally among the three tests, that is, each test will fail with probability at most $\delta/3$. To this end, we use the following form of the Chernoff bounds:

Theorem 2 ((Chernoff Bounds)) *Let $X_1, \ldots, X_\ell$ be independent Bernoulli random variables with $\Pr[X_i = 1] = p$ for all i, and let $X = \sum_i X_i$. Then*

$$\Pr\left[\left|\frac{X}{\ell} - p\right| \geq \lambda\right] \leq 2e^{-2\lambda^2 \ell}.$$

The self-tester comprises the following tests:

(1) *Linearity Test*: Let $\ell = (1/2\delta^2)\ln(6/\delta)$. Pick uniformly and independently at random $\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_\ell, \beta_\ell$ from V. For $1 \leq i \leq \ell$, let X_i be a Bernoulli random variable that is 1 iff $P(\alpha_i) \neq P(\alpha_i + \beta_i) - P(\beta_i)$. Let $X = \sum_i X_i$, and let $\tau = X/\ell$. If $\tau > \epsilon + \delta$, then output "FAIL."

(2) *Basis Test*: Let $\ell = (1/2\delta^2)\ln(6/\delta)$. Pick uniformly and independently at random $\beta_1, \beta_2, \ldots, \beta_\ell$ from V, and let $u = \text{majority}_{\beta_i}\{P(e_1 + \beta_i) - P(\beta_i)\}$. If $u \neq f(e_1)$ then output "FAIL."

(3) *Inductive Test*: Let $\ell = (1/2\delta^2)\ln(12/\delta)$. Pick uniformly and independently at random $\alpha_1, \alpha_2, \ldots, \alpha_\ell$ from V. Let $k = (1/2\delta^2)\ln(12\ell/\delta)$. For each α_i, pick uniformly and independently at random $\beta_{i1}, \beta_{i2}, \ldots, \beta_{ik}$ from V, and let $u_i = \text{majority}_{\beta_{ij}}\{P(\alpha_i+\beta_{ij})-P(\beta_{ij})\}$. Also for each α_i, pick uniformly and independently $\gamma_{i1}, \gamma_{i2}, \ldots, \gamma_{ik}$ from V, and let $v_i = \text{majority}_{\gamma_{ij}}\{P(\theta(\alpha_i)+\gamma_{ij}) - P(\gamma_{ij})\}$. For each i, let X_i be a Bernoulli random variable that is 1 iff $v_i \neq h(\theta, \alpha_i, u_i)$. Let $X = \sum_i X_i$, and let $\tau = X/\ell$. If $\tau > \epsilon + \delta$, then output "FAIL."

If all the above tests were completed successfully, output "PASS."

Theorem 3 *For any $\delta < 1$ and $\epsilon < 2/9$, the above three tests comprise a $(0, 2\epsilon)$-self-tester for f. That is, if a program P computes f correctly on all inputs, then the self-tester outputs "PASS" with probability 1, and if P computes f incorrectly on more than ϵ fraction of the inputs, then the self-tester outputs "FAIL" with probability at least $1-\delta$.*

Proof. Clearly if P always computes f correctly, the tester always outputs "PASS." For the other direction, we will show that if P passes the tests, then P computes f correctly on at least $1-2\epsilon$ fraction of the inputs. By Theorem 1, we know that the function g equals P on at least $1-2\epsilon$ fraction of the inputs, and that g is identical to f. It suffices to verify that conditions (1), (2) and (3) of the hypothesis of Theorem 1 are verified by the self-tester with high probability. By our choice of parameters and by the Chernoff bounds, with probability at least $1-\delta$, all the probability estimates made by the self-tester are accurate to within δ, and the result follows. The only point worth mentioning is that in the Inductive test, the error allowance $\delta/3$ was further apportioned into two equal halves. With error at most $\delta/6$, the tester computes $u_i = g(\alpha_i)$ correctly for all the α_i's, and with error at most $\delta/6$, the tester estimates the probability that g fails the inductive test to within δ. $\square$

2.1 Applications

We present some applications of Theorem 3. We remind the reader that a linear function f on a vector space V is efficiently self-testable without the generator bottleneck if there is a (linear) function h that is easily computable and that satisfies $f(\theta(\alpha)) = h(\theta, \alpha, f(\alpha))$ for all $\alpha \in V$. In each of our applications f, we show that a suitable function h exists that satisfies the above conditions. Recall the example of the function $E_c(p) = p(c)$, where the identity $E_c(xp) = cE_c(p)$ holds; in the applications below, we will only establish similar "recurrences." Also, for the sake of simplicity, we do not give all the technical parameters required; these can be computed by routine calculations, following the proofs of the theorems in the last subsection. Details will be given in the full version of the paper.

Our first application concerns linear functions of polynomials. Besides obtaining self-testers for polynomial evaluation and Fourier Transforms as in [Erg95], we obtain new self-testers for polynomial differentiation, integration, and the "mod" function of polynomials. Moreover, the vector space setting lets us generalize some of these results in terms of the matrices that compute linear transforms of vector spaces.

Let $P_n \subseteq \mathbb{K}[x]$ denote the group of polynomials in x of degree $\leq n$ over a field $\mathbb{K}$. The group P_n forms a vector space under usual polynomial addition and scalar multiplication by elements from $\mathbb{K}$. The polynomials $1, x, x^2, \ldots, x^n$ span P_n, and a polynomial $p(x) = \sum_i p_i x^i$ has vector representation $(p_n, p_{n-1}, \ldots, p_1, p_0)$. The linear operator θ in this case is just multiplication by x, thus $\theta(p) = xp$. To handle the case that multiplying p by x results in a polynomial of degree $n+1$, our testers will pad the polynomials with an extra zero coefficient (for x^{n+1}). For simplicity of exposition, we will suppress this detail; this does not cause any technical complication.

Polynomial Evaluation. For any $c \in \mathbb{K}$, let $E_c(p)$ denote, as described before, the function that returns the value $p(c)$. This function is linear. Moreover, the relation between $E_c(xp)$ and $E_c(p)$ is simple and linear: $E_c(xp) = cE_c(p)$. To self-test a program that claims to compute E_c, the inductive test (test (3)) is simply to choose many random p's, and verify that $E_c(xp) = cE_c(p)$ holds.

Vandermonde Operators and the Discrete Fourier Transform. If $u_1, u_2, \ldots, u_{n+1}$ are $n+1$ distinct elements of $\mathbb{K}$, then one may wish to evaluate a polynomial $p \in P_n$ simultaneously on all $n+1$ points. The ideas for E_c extend easily to this case, for $E_u(xp) = uE_u(p)$ for any $u \in \mathbb{K}$, and these relations hold simultaneously.

Let ω be a principal $(n+1)$-st root of unity in $\mathbb{K}$. The operation of converting a polynomial from its coefficient representation to pointwise evaluation at the powers of ω is known as the Discrete Fourier Transform (DFT). DFT has many fundamental applications that include fast multiplication of integers and polynomials. With our notation, the DFT of a polynomial $p \in P_n$ is simply $F(p) = (E_{\omega^0}(p), E_{\omega^1}(p), \ldots, E_{\omega^n}(p))$. The DFT F is linear, and $F(xp) = (\omega^0 E_{\omega^0}(p), \ldots, \omega^n E_{\omega^n}(p))$. Notice that here the function "h" is really n functions h_{ω^i} for $0 \le i \le n$. The self-tester will simply choose p's randomly, request the program to compute $F(p)$ and $F(xp)$, and verify for each i, $0 \le i \le n$, that $(F(xp))[i] = \omega^i (F(p))[i]$ holds.

This suggests the following generalization (for the case of arbitrary vector spaces). Simultaneous evaluation of a polynomial at $d+1$ points $u_1, \ldots, u_{n+1}$ corresponds to multiplying the vector p by a Vandermonde matrix M, where $M_{ij} = u_i^{j-1}$. The ideas used to test simultaneous evaluation of polynomials and the DFT extend to give a self-tester for any linear transform that is represented by a Vandermonde matrix.

The matrix for the Discrete Fourier Transform can be written as a Vandermonde matrix F, where $F_{ij} = \omega^{ij}$. The inverse of the DFT, that is, converting a polynomial from pointwise representation to coefficient form, also has a Vandermonde matrix whose entries are given by $\widetilde{F}_{ij} = (1/\det F)\omega^{-ij}$. It follows that the inverse Fourier Transform can be self-tested efficiently. Another point worth mentioning here is that in carrying out the inductive test (test (3)), the self-tester does not have to compute $\det F$. All it needs to do is verify that for many randomly chosen p's, the identity $(\widetilde{F}(xp))[i] = \omega^{-i}(\widetilde{F}(p))[i]$ holds.

Operators in elementary Jordan canonical form. A linear operator M is said to be in elementary Jordan canonical form if for some $c \in \mathbb{K}$, all the diagonal entries of M are c's, and all the elements to the left of the main diagonal (the first non-principal diagonal in the lower triangle of M) are 1's. It is easy to verify that $M\theta = \theta M + M'$, where M' is a matrix that has a -1 in the top left corner and a 1 in the bottom right corner and zeroes elsewhere. Therefore, for every $v = (v_n, v_{n-1}, \ldots, v_2, v_1)^T$ in the vector space, $M(\theta(v)) = \theta(M(v)) + (-v_n, 0, \ldots, 0, v_1)^T$. This gives an easy way to implement the inductive test in the self-tester.

An attempt to extend this to matrices in Jordan canonical form, or even to diagonal matrices, seems not to work. However, if a diagonal or "shifted" diagonal matrix has a special structure, then we can obtain self-testers that avoid the generator problem. For example, the matrix corresponding to the differentiation of polynomials has a special structure: it contains the entries $n, n-1, \ldots, 1$ on the diagonal below the main diagonal. We exploit this to obtain the following:

Differentiation. Differentiation of polynomials is a linear function $D : P_n \to P_{n-1}$. We have the explicit form for h: $D(xp) = p + xD(p)$. Integration of polynomials is a linear function $I : P_n \to P_{n+1}$. The explicit form for h is $I(xp) = xI(p) - I(I(p))$. Hence we can avoid the generator problem for these functions.

Higher Order Differentiation. Let D^k denote the k-th differential operator. It is easy to write a recurrence-like identity for D^k in terms of $D^j, j < k$. This gives us a self-tester in the "library setting" described in [BLR93, Rub90], where one assumes that there are programs to compute all these differential operators. However, if we wish to self-test a program that only computes D^k, this assumption is not valid. To remedy this, we will use the following lemma.

Lemma 4 *If p is a polynomial in x that is differentiable k times, then*

$$\sum_{i=0}^{k} \binom{k}{i} (-x)^{k-i} D^k(x^i p) = k!p$$

Proof. By induction on k. The base case ($k = 0$) is obviously true. Let $k > 0$, we have $\mathcal{S}_k = \sum_{i=0}^{k+1} \binom{k+1}{i}(-x)^{k+1-i} D^{k+1}(x^i p)$. Since $D^{k+1}(x^i p) = D^k(ix^{i-1}p + x^i D(p))$ and since differentiation is linear, we have

$$\mathcal{S}_k = \left[\sum_{i=0}^{k+1} \binom{k+1}{i} (-x)^{k+1-i} D^k(ix^{i-1}p)\right] + \left[\sum_{i=0}^{k+1} \binom{k+1}{i} (-x)^{k+1-i} D^k(x^i D(p))\right].$$

Since the first term ($i = 0$) in the first sum vanishes, and since $i\binom{k+1}{i} = (k+1)\binom{k}{i-1}$, the first sum evaluates to $(k+1)\sum_{i=1}^{k} \binom{k}{i-1}(-x)^{k+1-i} D^k(ix^{i-1}p)$, which equals $(k+1)!p$ by the inductive hypothesis. Hence it suffices to show that the second sum evaluates to 0. This summation can be written as $\sum_{i=0}^{k} \binom{k+1}{i}(-x)^{k-i+1} D^k(x^i D(p)) + D^k(x^{k+1}D(p))$. By Pascal's identity, this sum can be split into the terms $\sum_{i=0}^{k} \binom{k}{i-1}(-x)^{k-i+1} D^k(x^i D(p))$, $\sum_{i=0}^{k} \binom{k}{i}(-x)^{k-i+1} D^k(x^i D(p))$, and $D^k(x^k \cdot xD(p))$. Regrouping terms and applying the inductive hypothesis, this equals $(-x)k!D(p) + k!(xD(p)) = 0$. $\square$

By the "robustness" of the inductive test, testing if a program P satisfies this identity for most p suffices to ensure that it satisfies this identity everywhere. This implies that if P computes $D^k(p)$ correctly, then it computes $D^k(x^k p)$ correctly. To carry the induction through, we need to test k base cases: $P(1), P(x), P(x^2), \ldots, P(x^{k-1})$.

Mod Function. Let $q \in \mathbb{K}[x]$ be a monic irreducible polynomial. Let $M_q(p)$ denote the mod function with respect to q, that is, $M_q(p) = p \bmod q$. This is a linear function when the addition is interpreted as mod q addition. Since q is monic, the degree of $M_q(p)$ is always less than $\deg(q)$. If $c \in \mathbb{K}$ is the coefficient of the highest degree term in p, we have

$$M_q(xp) = \begin{cases} xM_q(p) & \text{if } \deg(xM_q(p)) < \deg(q) \\ xM_q(p) - cq & \text{if } \deg(xM_q(p)) = \deg(q) \end{cases}$$

As before, in testing if a program P computes the function M_q, step (3) of the self-tester is to choose many p's at random, compute $P(p)$ and $P(xp)$, and verify that one of the identities $P(xp) = xP(p)$ or $P(xp) = xP(p) - cq$ holds (depending on the degree of p).

3 Multilinear Functions

A function $f(x_1, \ldots, x_k)$ is *multilinear* if it is linear in each of its variables when the other variables are fixed. Following [Erg95], our main motivating example is polynomial multiplication: suppose we wish to test a program that purports to multiply polynomials of degree $\leq n$ over a finite field $\mathbb{K}$. The naive approach would require testing the program at n^2 pairs of generators. Blum, Luby, and Rubinfeld [BLR93] give a "bootstrap" self-tester that makes $O(\log n)$ calls to the program. Ergün's work presents an efficient self-tester that avoids the generator bottleneck. As in the previous section, we generalize this to arbitrary multilinear functions over large groups. For k-variate functions, we need to test the program at n^k different k-tuples of generators, where n is the number of generators of the group. In this section, we give a scheme to efficiently self-test multilinear functions, where the self-tester is required to know the correct value of f at only one point. For easy visualization, we request the reader to consider polynomial multiplication as an archetypal example.

We proceed in the same manner as before. Let $\langle\alpha\rangle$ denote $(\alpha_1, \ldots, \alpha_k) \in V^k$, and let $\langle\alpha, i\rangle$ denote $(\alpha_1, \ldots, \alpha_{i-1}, \theta(\alpha_i), \alpha_{i+1}, \ldots, \alpha_k) \in V^k$. If $f(\langle\alpha, i\rangle) = h_i(\theta, \langle\alpha\rangle, g(\langle\alpha\rangle))$, then h_i is multilinear in $\langle\alpha\rangle$. Moreover if f is symmetric, then all the h_i's are identical. The self-tester for f consists of the following tests:

(1) *Multilinearity Test*: Use the self-tester for multilinearity from [BFL91, FGL$^+$91], which also works in our vector space setting, to verify that P agrees with some multilinear function g on all but an ϵ fraction of the inputs. (Note, however, that these tests impose a limit $\epsilon \leq \epsilon(k) = (1/k)^{O(1)}$. We will focus, for the moment, on the case where $1/2^k \ll \epsilon(k)$; more details will be given in the full paper.) If the program P passes this test, perform the next two tests using $P_{\rm sc}$, the self-corrector for P, to compute g.

(2) *Basis Test*: Verify that $g(\langle e_1\rangle) = f(\langle e_1\rangle)$.

(3) *Inductive Test*: For $1 \leq i \leq k$, verify that $\Pr_{\alpha\in V^k}[g(\langle\alpha, i\rangle) \neq h_i(\theta, \langle\alpha\rangle, g(\langle\alpha\rangle))] \leq \epsilon+\delta$.

With a total of $(1/\delta)^{O(1)}$ calls to P, we can arrange to have the probability that all three tests succeed to be at least $1-\delta$. Output "PASS" iff P passes all these tests.

It is not hard to see that multilinear functions over groups (for example, multiplication of polynomials over a finite field $\mathbb{K}$) are randomly self-reducible, and hence self-correctable. For the rest of the proof, we assume that g is multilinear. Since Test (2) ensures that

g agrees with f on $\langle e_1 \rangle$, it suffices to prove that for every $i = 1, \ldots, k$ and every $\langle \alpha \rangle \in V^k, g(\theta(\langle \alpha, i \rangle)) = h_i(\langle \alpha \rangle, \theta, g(\langle \alpha, \rangle))$. By induction, we can then conclude that g computes f at all generators and hence everywhere.

For any $\langle \alpha \rangle \in V^k$, we prove that $g(\theta(\langle \alpha, i \rangle)) = h_i(\theta, \langle \alpha \rangle, g(\langle \theta \rangle))$. For simplicity, we give the proof for $k = 2$ and $i = 2$.

$$
\begin{aligned}
& \Pr_{\beta_1, \beta_2 \in V^2} [g(\alpha_1, \theta(\alpha_2)) \\
= \;& g(\beta_1 + \alpha_1 - \beta_1, \theta(\beta_2 + \alpha_2 - \beta_2)) \\
= \;& g(\beta_1, \theta(\beta_2 + \alpha_2 - \beta_2)) + g(\alpha_1 - \beta_1, \theta(\beta_2 + \alpha_2 - \beta_2)) \\
= \;& g(\beta_1, \theta(\beta_2)) + g(\beta_1, \theta(\alpha_2 - \beta_2)) + g(\alpha_1 - \beta_1, \theta(\beta_2)) + g(\alpha_1 - \beta_1, \theta(\alpha_2 - \beta_2)) \\
= \;& h_2(\theta, \beta_1, \beta_2, g(\beta_1, \beta_2)) + h_2(\theta, \beta_1, \alpha_2 - \beta_2, g(\beta_1, \alpha_2 - \beta_2)) + \\
& h_2(\theta, \alpha_1 - \beta_1, \beta_2, g(\alpha_1 - \beta_1, \beta_2)) + h_2(\theta, \alpha_1 - \beta_1, \alpha_2 - \beta_2, g(\alpha_1 - \beta_1, \alpha_2 - \beta_2)) \\
= \;& h_2(\theta, \beta_1, \alpha_2, g(\beta_1, \alpha_2)) + h_2(\theta, \alpha_1 - \beta_1, \alpha_2, g(\alpha_1 - \beta_1, \alpha_2)) \\
= \;& h_2(\theta, \alpha_1, \alpha_2, g(\alpha_1, \alpha_2))] \geq 1 - 4\epsilon > 0.
\end{aligned}
$$

The first equality is rewriting. Multilinearity of g implies the second and third equalities. If the true probability $\Pr[g(\langle \alpha, i \rangle) \neq h_i(\theta, \langle \alpha \rangle, g(\langle \alpha \rangle))] < \epsilon$, then the fourth equality fails with probability less than 4ϵ. The rest of the equality follows from the multilinearity of h_2 and g. If $\epsilon < 1/4$, this probability is non-zero. The fact that the probability is independent of β_1, β_2 yields the desired result.

It is easy to see that the above proof extends to an arbitrary k so long as $\epsilon < 1/2^k$. Thus, we obtain the following theorem whose proof mirrors the proof of Theorem 3.

Theorem 5 *If f is a k-variate linear function, then for any $\delta < 1$ and $\epsilon < 1/2^k$, the above three tests comprise a $(0, 2^k\epsilon)$-self-tester for f that succeeds with probability at least $1 - \delta$.*

3.1 Applications

The function $M(p_1, p_2)$ that multiplies two polynomials is symmetric and linear in each variable. Moreover, since $M(xp_1, p_2) = xM(p_1, p_2)$, polynomial multiplication has an efficient self-tester.

An interesting application of polynomial multiplication, together with the "mod" function described in Section 2.1, is the following. It is well-known that a degree n (finite) extension $\mathbb{K}$ of a finite field $\mathbb{F}$ is isomorphic to the field $\mathbb{F}[x]/(\alpha)$, where α is an irreducible polynomial of degree n over $\mathbb{F}$. Under this isomorphism, each element of $\mathbb{K}$ is viewed as a polynomial of degree $\leq n$ over $\mathbb{F}$, addition of two elements $p, q \in \mathbb{K}$ is just their sum $p + q$ as polynomials, and multiplication of $p, q \in \mathbb{K}$ is given by $pq \bmod \alpha$. It follows that linear functions over a finite extension of a finite field can be self-tested without the generator bottleneck, that is, the number of calls made to the program being tested is independent of the degree of field extension!

4 Nonlinear Functions

In this section, we consider nonlinear functions. Specifically, we deal with exponentiation and constant degree polynomials over groups with n generators, and show how to self-test them without the generator bottleneck. Since a clear notion of multiplication is required, and since we need the Lagrange interpolation formula (for random self-reducibility of polynomials), we will restrict ourselves to the case of rings. An interesting example is the ring of polynomials over finite fields, and for simplicity, we restrict our discussion below to this ring. It is obvious that exponentiation and constant degree polynomials are clearly defined over this ring.

We first consider the function $f(p) = p^d$ for some d (that is, raising a polynomial to the d-th power). Suppose a program P claims to perform this exponentiation for all degree n polynomials $p \in P_n \subseteq \mathbb{K}[x]$, the ring of all polynomials over the field $\mathbb{K}$. Using the "low-degree" test of Rubinfeld and Sudan [RS93] we can first test if the function computed by P is a degree d polynomial. As before, we can also verify that g, the self-corrected version of P, satisfies $g(e_1) = f(e_1)$.

The induction identity $f(xp) = x^d f(p)$ also applies, so it remains to show how to verify that $g(xp) = x^d g(p)$ for all $p \in P_n$. This is where the difficulty comes. The obvious idea would be to emulate the case of linear functions and verify that $g(xp) = x^d g(p)$ for most randomly chosen p. It turns out, however, that proving that this property is robust, that is, proving that random verification suffices to ensure that this identity holds everywhere, is not so easy as before. Recall that in the proof for the linear case, we expressed p as $p - q + q$ for some random q, and proved using the linearity of g and θ that $g(\theta p) = h(\theta, p, g(p))$ holds with positive probability, and hence with probability one. There we made essential use of the fact that $g(x(p-q) + xq) = g(x(p-q)) + g(xq)$. When g is nonlinear, this identity is no longer true.

To circumvent this problem, we follow the ideas of Rubinfeld and Sudan [RS93], who extended the [BLR93] result from linear functions to low-degree polynomials. They used the fact that the Lagrange interpolation identity for polynomials is robust. Here our task is to verify that $g(xp) = x^d g(p)$ for all p. We will show that if this equality holds on a large fraction of the p's, then it holds for all p. Before proceeding to do so, we pause to state the following elementary fact concerning the Lagrange interpolation identity.

Fact 6 *Let g be a degree d polynomial. For any $p \in P_n$, if $q_1, q_2, \ldots, q_{d+1}$ are distinct elements of P_n,*

$$g(p) = \sum_{i=1}^{d+1} g(q_i) \prod_{j \neq i} \frac{p - q_j}{q_i - q_j}; \qquad \textit{and also} \qquad g(xp) = \sum_{i=1}^{d+1} g(xq_i) \prod_{j \neq i} \frac{p - q_j}{q_i - q_j}.$$

The self-tester for $f(x) = x^d$ comprises the following tests:

(1) *Degree Test*: Use the low-degree test of [RS93] to verify that P computes a degree d polynomial.

(2) *Basis Test*: Using P_{sc} to compute g, verify that $g(e_1) = f(e_1)$.

(3) *Inductive Test*: Using $P_{\rm sc}$ to compute g, verify that $\Pr_\alpha[g(x\alpha) \neq x^d(g(\alpha))] \leq \epsilon + \delta$.

As before, the all the above tests can be made to succeed with probability $1 - \delta$, using $(1/\delta)^{O(1)}$ calls to P. Let β denote the probability that the $d+1$ random choices from the domain produce distinct elements. Usually, the domain is large enough so that β is close to 1. Below we sketch the proof that if $\epsilon < \beta/(d+1)$ and $P_{\rm sc}$ passes Tests (1) and (3), then g satisfies $g(xp) = x^d(g(p))$ everywhere.

$$\begin{aligned}
\Pr_{q_1,\ldots,q_{d+1}}[g(xp) &= \sum_{i=1}^{d+1} a_i g(xq_i) \\
&= \sum_{i=1}^{d+1} a_i x^d g(q_i) \\
&= x^d \sum_{i=1}^{d+1} a_i g(q_i) \\
&= x^d g(p)] \geq \beta - (d+1)\epsilon > 0.
\end{aligned}$$

Here $a_i = \prod_{j \neq i}(p - q_j)/(q_i - q_j)$. The first equality is Fact 6, and applies since g has been verified to be a degree d polynomial. Since the q_i's are uniformly and identically distributed, by Test (3), the second equality fails with probability $< (d+1)\epsilon$. The third equality is just rewriting, and the fourth equality is due to Fact 6. Since the equality $g(xp) = x^d g(p)$ holds independent of q_i's, if $\epsilon < \beta/(d+1)$, it holds always.

Theorem 7 *The function $f(p) = p^d$ has an efficient self-tester that avoids the generator bottleneck.*

Next we consider extending the above for arbitrary degree d polynomials. Clearly the low-degree test and the basis test work as before. The interpolation identity is valid, too. The missing ingredient is the availability of nice identities like "$f(xp) = x^d f(p)$." We get around this difficulty using the notion of *program result-checkers* [BK89]. Recall that a checker for a function f is a probabilistic program that, given a program P purporting to compute f and an input p, outputs "CORRECT" if $P(q) = f(q)$ for all q, outputs "WRONG" if $P(p) \neq f(p)$. The checker may call the program P on inputs other than p, but must access P only as a black-box; it does not have access to the code of P. If a function has a self-tester and a self-corrector, then it has a checker.

Suppose $f(p)$ is a degree d polynomial in p, and suppose P purports to compute f. The most obvious way to perform self-testing would be to randomly choose many inputs p, compute $P(p)$ and check if $P(p) = f(p)$. This way, one can easily estimate the probability that $P(p) = f(p)$. The difficulty, however, is that the self-tester needs to know the correct value of $f(p)$ on many randomly chosen inputs. In other words, the self-tester is not "different" from the function being self-tested. To get around this, we employ a suggestion due to Ronitt Rubinfeld. We show how to use the (possibly faulty) program P to compute the correct value of $f(p)$ in a "certifiable" manner—this is where the checker comes in.

We first prove a technical lemma that helps us express d-ary multiplication in terms of f—that is, we establish a reduction from the multilinear function $\prod_{i=1}^d p_i$ to the nonlinear function f. This reduction is a generalization to degree d of the elementary identity $pq = ((p+q)^2 - (p-q)^2)/4$, but stronger in that it works for arbitrary polynomials of degree d, not just degree-d exponentiation. Given this reduction, we can build a program P' that performs c-ary multiplication for any $c \le d$ (if $c < d$, we can simply multiply by extra 1's). The program P' can then be self-tested without the generator bottleneck, using the result of Section 3. We can then build a checker for multiplication, and using this we can package P' into a multiplication program Q that *certifies* its output as "CORRECT" or "WRONG." The checker itself, and hence Q, may fail with probability β for some $\beta > 0$. We can then build a program Q' that computes f using the reliable multiplication program Q. Finally, we pick random p's, and check if $P(p) = Q'(p)$, and estimate the probability that P computes f correctly.

Lemma 8 *Let $p_1, \ldots, p_d$ be distinct variables. Let $0 \le m < 2^d$. Denote by m_i the i-th least significant bit of m, that is, $m = m_d m_{d-1} \ldots m_1$. Then*

$$\sum_{m=0}^{2^d-1} \left(\prod_{i=1}^{d} (-1)^{m_i}\right) \left(\sum_i (-1)^{m_i} p_i\right)^c = \begin{cases} 0 & \textit{if } c < d \\ 2^d d! \prod_{i=1}^d p_i & \textit{if } c = d \end{cases}$$

Proof.

$$\begin{aligned}
&\sum_{m=0}^{2^d-1} \left(\prod_{i=1}^{d} (-1)^{m_i}\right) \left(\sum_i (-1)^{m_i} p_i\right)^c \\
&= \sum_{m=0}^{2^d-1} \left(\prod_{i=1}^{d} (-1)^{m_i}\right) \sum_{\substack{n_1+\cdots+n_d=c \\ 0\le n_1,\ldots,n_d \le c}} \binom{c}{n_1,\ldots,n_d} \prod_{i=1}^{d} (-1)^{m_i n_i} p_i^{n_i} \\
&= \sum_{\substack{n_1+\cdots+n_d=c \\ 0\le n_1,\ldots,n_d \le c}} \binom{c}{n_1,\ldots,n_d} \prod_{i=1}^{d} p_i^{n_i} \sum_{m=0}^{2^d-1} \prod_{i=1}^{d} (-1)^{m_i + m_i n_i} \\
&= \sum_{\substack{n_1+\cdots+n_d=c \\ 0\le n_1,\ldots,n_d \le c}} \alpha \left[\sum_{m=0}^{2^d-1} \prod_{i=1}^{d} (-1)^{(m_i + m_i n_i) \bmod 2}\right],
\end{aligned}$$

where α depends only on the p_i's and the n_i's. Fix $n_1, \ldots, n_d$. We claim that the inner summation is non-zero if and only if $c = d$ and $n_1 = \cdots = n_d = 1$. When $c = d$ and $n_1 = \cdots = n_d = 1$,

$$\sum_{m=0}^{2^d-1} \prod_{i=1}^{d} (-1)^{(m_i + m_i n_i) \bmod 2} = 2^d; \qquad \text{and} \qquad \alpha = d! \prod_{i=1}^{d} p_i.$$

For every i such that n_i is odd, $(-1)^{(m_i + m_i n_i) \bmod 2} = 1$. If $c < d$ or if all the n_i's are not 1, there must be at least one n_i which is zero (since they sum to c). Therefore, there must be at least one n_i which is even. Arbitrarily fix some j such that n_j is even; now,

$$(-1)^{(m_j + m_j n_j) \bmod 2} = (-1)^{m_j}.$$

For any setting of the bits of m other than m_j, the two terms that correspond to $m_j = 0$ and to $m_j = 1$ cancel each other, and the sum is zero. $\square$

Corollary 9 *For any polynomial $G(x) = \sum_i p_i x^i$ of degree d,*

$$\sum_{m=0}^{2^d-1} \left(\prod_{i=1}^{d} (-1)^{m_i} \right) G \left(\sum_{i=1}^{d} (-1)^{m_i} x_i \right) = p_d 2^d d! \prod_{i=1}^{d} x_i.$$

By Corollary 9, it is easy to see that if P computes f correctly on all inputs, the self-tester will output "PASS" with probability one. If $\epsilon < 1/2^d$, the self-corrector for multiplication will work correctly, since a random call to d-ary multiplication results in 2^d uniformly and identically distributed calls to P. Therefore, the certified program Q computes multiplication with probability arbitrarily close to $1-\beta$, and the certified program Q' computes $f(p)$ correctly for every p with probability arbitrarily close to $1 - d\beta$. If $P(p) \neq f(p)$ with probability $> \epsilon$, then the probability that the self-tester will catch an error in one trial is $\epsilon(1-d\beta)$; by repeated trials, this can be boosted to any required success probability $1 - \delta$. The dependence on the running time of our self-tester on d is rather poor. However, if the degree n of the polynomials is large and if d is constant, this scheme is attractive. In particular, it helps us efficiently self-test constant degree polynomials on finite field extensions of large dimension.

Acknowledgments

We are grateful to Professor Ronitt Rubinfeld for valuable suggestions and guidance. We thank Professor Manuel Blum, Funda Ergün, and Mandar Mitra for useful discussions, and the anonymous referees for helpful comments.

References

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BK89] M. Blum and S. Kannan. Designing programs that check their work. In *Proc. 21st Annual ACM Symposium on the Theory of Computing*, pages 86–97, 1989.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comp. Sys. Sci.*, 47(3):549–595, 1993. An earlier version appeared in STOC 1990.

[BW94] M. Blum and H. Wasserman. Program result-checking: A theory of testing meets a test of theory. In *Proc. 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 382–392, 1994.

[Erg95] F. Ergün. Testing multivariate linear functions: Overcoming the generator bottleneck. In *Proc. 27th Annual ACM Symposium on the Theory of Computing*, pages 407–416, 1995.

[FGL+91] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 2–12, 1991.

[GLR+91] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/ correcting for polynomials and for approximate functions. In *Proc. 23rd Annual ACM Symposium on the Theory of Computing*, pages 32–42, 1991.

[Lip91] R. Lipton. New directions in testing. In *Proc. of DIMACS Workshop on Distributed Computing and Cryptography*, pages 191–202, 1991.

[RS92] R. Rubinfeld and M. Sudan. Testing polynomial functions efficiently and over rational domains. In *Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 23–43, 1992.

[RS93] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials and their applications to program testing. TR 93-1387, Dept. of Computer Science, Cornell University, 1993. To appear in *SIAM Journal of Computing.*

[Rub90] R. Rubinfeld. *A Mathematical Theory of Self-Checking, Self-Testing, and Self-Correcting Programs.* PhD thesis, University of California at Berkeley, 1990.

[Rub94] R. Rubinfeld. Robust functional equations with applications to self-testing/ correcting. In *Proc. 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 288–299, 1994.

[Sud94] Madhu Sudan. On the role of algebra in the efficient verification of proofs. In M. Agrawal, V. Arvind, and M. Mahajan, editors, *Proc. Workshop on Algebraic Methods in Complexity Theory (AMCOT)*, pages 58–68, 1994. Technical Report IMSc 94/51, The Institute of Mathematical Sciences, Madras, India.