Chapter 1

Self-Testing Polynomial Functions Efficiently and over Rational Domains

Ronitt Rubinfeld [*] Madhu Sudan [†]

Abstract

In this paper we give the first self-testers and checkers for polynomials over rational and integer domains. We also show significantly stronger bounds on the efficiency of a simple modification of the algorithm for self-testing polynomials over finite fields given in [8].

1 Introduction

Suppose someone gives us an extremely fast program P that we can call as a black box to compute a function f. Rather than trust that P works correctly, a *self-testing program* for f ([5]) verifies that program P is correct on most inputs (*without* assuming the correctness of another program that is as difficult as one that computes the function), and a *self-correcting program* ([5] [9]) for f takes a program P, that is correct on most inputs, and uses it to compute f correctly on every input (with high probability). Both access P only as a black-box and in some precise way are not allowed to compute the function f. Self-testing/correcting is an extension of program result checking as defined in [3],[4]. If f has a self-tester and a self-corrector, then f has a program result checker.

Our first result concerns checking and self-testing over rational and integer domains. To date, most self-testing/correcting pairs and checkers that have been found for numerical functions (cf. [5][2][9][8]) have been for functions over finite fields (for example polynomial functions over finite fields [2][9] [8]) or for domains for which a group structure has been imposed on a finite subset of the domain (for example integer multiplication and division functions [5], and floating point logarithm and exponentiation functions [8]). In [7] there is a checker that can also be made into a self-tester for matrix multiplication over any field, but self-correctors for matrix multiplication are only known over finite fields [5].

Many of the self-testers, for example those for linear and polynomial functions [5][8], have been based on showing that a program which satisfies certain easy to verify conditions on randomly chosen inputs must be computing the correct function on most inputs.

[*] Hebrew University. This work was done while the author was at Princeton University. Supported by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), NSF-STC88-09648. Part of this research was also done while the author was visiting Bellcore.

[†] U.C. Berkeley. Research supported by NSF PYI Grant CCR 8896202.

Finite fields are much easier to work with to this end because they are conducive to clean simple arguments about distributions, i.e. if y is chosen uniformly at random from finite field F, then for fixed $a, b \in F$, $a+y, by, a+by$ are all uniformly distributed in F (shifting and scaling the distribution gives back the same distribution). This is not at all the case for nontrivial distributions over fields of characteristic 0. In fact, if D is a finite subset of F, and y is chosen uniformly at random from D, then for most $a, b \in D$, $a+y, by, a+by$ are not even likely to be in D. However, functions over such fields, for example rational domains, are of great importance for application programming. A first step in the direction of finding self-testing/correcting pairs for polynomial functions is given in [8], where a self-corrector for polynomial functions over fixed point rational domains of the form $D_{s,r} = \{\frac{y}{s} | \text{ integers } |y| \leq r\}, r \in \mathcal{Z}, s \in \mathcal{Z}\backslash\{0\}$ is given (note that this includes integer domains and fixed point arithmetic domains). The self-corrector needs a program that is known to be correct for a large fraction of inputs over a larger and finer precision domain than the one for which the self-corrector will be able to compute the function correctly.

We show that there is a very simple way to actually verify that the program is good enough for the polynomial function self-corrector over fixed point rational and integer domains, without assuming the correctness of another program that computes the polynomial: we give a self-tester for any (multivariate) polynomial function over fixed point rational and integer domains. Thus we also give the first checkers for polynomials over rational and integer domains.

A first approach to testing the program might be to attempt to interpolate the polynomial being computed by the program and to verify that the program's output at random points is equal to the value of the interpolated polynomial. This approach is not desirable because both the interpolation procedure and the evaluation of the polynomial at random points make the tester at least as difficult as the program, and not different in the sense defined by [3] [4]. However, similar approaches have been useful in the area of interactive proofs [1] [6] [10], which only requires that the procedure be in polynomial time.

The self-tester for polynomials over finite fields in [8] is composed of two parts: One is a *degree test* - a test which verifies that a program P is computing a function which is usually equal to some (total) degree d polynomial g. The other is an *equality test* - a test which verifies that polynomial g is identically equal to polynomial f. The equality test uses the fact that g can be computed easily using P (this follows from the fact that P is usually equal to g and because there is a self-corrector for polynomial functions over finite fields [2][9]) and verifies that $g = f$ on at least $(d+1)^n$ points (where n is the number of variables). Then since two different degree d polynomials can agree on fewer than $(d+1)^n$ points, if the program passes the test, it must be computing the correct degree d polynomial. The second part can be easily extended to work on rational domains since there is a self-corrector over such domains.

We concentrate on showing that there is a degree test which verifies that the total degree of the polynomial is at most d over the rationals. The ideas used to get such testers and checkers over the new domains seem quite general and it is hoped that they will be applicable to many other problems, for example matrix determinant, matrix rank and polynomial multiplication over the rationals and reals.

Our second result concerns the self-tester for polynomials over finite fields in [8], which has been shown to work for multivariate polynomials as well by Shen [11]. In particular, we are interested in the efficiency of the degree test, since often testing is done online (i.e. when the user decides to use the program) [5]. Efficient

degree tests are interesting for another reason in addition to program testing: they have been used as a tool in order to get multi-prover interactive proofs for non-deterministic exponential time complete functions in [1]. [6] have in turn used the interactive proof in [1] in order to obtain results about the difficulty of approximating the size of the clique, in which the strength of the result depends on the efficiency of the interactive proof. Although their results are not improved by our test, our test is more efficient in situations where the total degree is smaller than maximum degree of any variable multiplied by the number of variables.

The degree test given in [8][11] makes $O(d^2)$ tests, for a total of $O(d^3)$ calls, to the program in order to verify that the total degree of the polynomial is at most d. On the other hand, it is easy to see that at least d calls to the program are needed. We narrow this gap by showing that a slight modification of the self-tester needs only $O(d)$ tests and $O(d^2)$ total calls. We conjecture that our algorithm (or a slight modification of it) requires only $O(1)$ tests and $O(d)$ calls. The degree tests given in [1],[6] both require a number of calls depending on the number of variables: the test in [1] requires $O(n^6 d_{max})$ calls to the program where n is the number of variables, and d_{max} is the maximum degree of any variable, and the test in [6] requires $O(nd_{max})$ calls to the program.

All of the tests given in this paper are very simple to compute, and require only a small number of additions, subtractions and comparisons.

2 Definitions

Consider a function $P : X \to Y$ that attempts to compute f. We consider three domains, the "query" domain $D_q \subset X$, the "test" domain $D_t \subset X$ and the "safe" domain $D_s \subset X$. We say that program P ϵ-*computes* f on D_t if $Pr_{x \in D_t}[P(x) = f(x)] > 1 - \epsilon$. An (ϵ_1, ϵ_2)-*self-tester* $(0 \leq \epsilon_1 < \epsilon_2)$ for f on (D_q, D_t) must fail any program that does not ϵ_2-compute f on D_t, and must pass any program that ϵ_1-computes f on D_q (note that the behavior of the tester is not specified for all programs). If $D_q = D_t = D$, then we say we have an (ϵ_1, ϵ_2) self-tester for f on D. The tester should satisfy these conditions with error probability at most β, where β is a confidence parameter input by the user. An ϵ-*self-corrector* for f on (D_t, D_s) is a program C that uses P as a black box, such that for every $x \in D_s$, $Pr[C^P(x) = f(x)] \geq 2/3$, [1] for *every* P which ϵ-computes f on D_t. Furthermore, all require only a small multiplicative overhead over the running time of P and are different, simpler and faster than any correct program for f in a precise sense defined in [4].

We use $x \in_R D$ to mean that x is chosen uniformly at random in D. We use $[k]$ to denote the set of integers $\{1, 2, \cdots, k\}$.

The self-testers and self-correctors for polynomials are based on the existence of the following interpolation identity relating the function values between points: for all multivariate polynomials f of total degree at most d, $\forall x, t \in F^n$, $\sum_{i=0}^{d+1} \alpha_i f(x + a_i \cdot t) = 0$ where the a_i's are distinct elements of F, $\alpha_0 = -1$ and α_i depends only on F, d and not on x, t or even f. In particular, if $F = Z_p^n$, a_i can be chosen to be i, and then $\alpha_i = (-1)^{i+1}\binom{d+1}{i}$. It is also known that if a function satisfies the interpolation identities at a set of $d + 1$

[1] this can be amplified to $1 - \beta$ by $O(\log 1/\beta)$ independent repetitions and and a majority vote.

points, then its values at those $d+1$ points lie on a degree d polynomial. Furthermore $\sum_{i=0}^{d+1} \alpha_i f(x+i\cdot t)$ can be computed in $O(d^2)$ time using only additions and comparisons by the method of finite differences described in the appendix [12].

3 Testers for Polynomials over Integer and Rational Domains

In this section we present a tester for polynomial functions over rational domains. In [8] a self-corrector for such functions is presented which can compute the correct value of a function f, from a "safe" domain D_s, provided the function has been tested over a finite number of "test" domains D_t^i. Typically the test domains were much larger and of finer precision than the safe domain, but were all rational domains. In this section we complement this result by presenting a tester for any rational domain.

Notation : We use $\mathcal{D}_{n,s}$ to denote the set $\{\frac{i}{s} | i \in \mathcal{Z},\ |i| \leq n\}$.

Note that all rational domains are of the form $\mathcal{D}_{n,s}$, where $n, s \in \mathcal{Z}$.

The tester we construct in this section explicitly tests that certain properties hold over various domains in order to infer that the desired property holds for the domain that we are interested in. We define the domains that our test considers.

3.1 Domains and their properties.

Throughout this section x and t will denote n dimensional vectors chosen from an appropriate domain.

Let the domain we wish to certify the program over be

$$\mathcal{X}_0 \equiv \mathcal{D}_{p,s}^n.$$

We will use the following domains :

- $\mathcal{X}_1 \equiv \mathcal{D}_{(d+2)p,s}^n$: This will ensure that for all $x, t \in \mathcal{X}_0$, and for $i \in [d+1]$, $x + i * t \in \mathcal{X}_1$.
- $\mathcal{T} \equiv \mathcal{D}_{L_p,L_s}^n$, where $L_p = p(d+2)(n(d+1)!)^3$ and $L_s = s((d+1)!)^3$: $\mathcal{T}$ contains $\mathcal{X}_1$ and is a considerably larger and finer domain than $\mathcal{X}_1$.
- For all $j \in [d+1]$, $\mathcal{T}_j \equiv \{jx | x \in \mathcal{T}\}$.
- For all $i, j \in [d+1]$, $\mathcal{X}_{ij} \equiv \{ix | x \in \mathcal{T}_j\}$: The fineness of $\mathcal{T}$ allows us to conclude that all the domains $\mathcal{T}_j$ and $\mathcal{X}_{ij}$ contain $\mathcal{X}_1$, a fact which will be useful later.

Frequently, in our proof, we will need to establish that a certain property holds over a new domain which is not quite the same as any of the domains above but is "close" to one of them. We first define the notion of "closeness" that we use in this paper and then we establish some relations among domains. (*The definition as well as its properties are initially given in [8].*)

DEFINITION 3.1 ([8]) $\delta(p_1, p_2) = \begin{cases} p_1 & \text{if } p_1 = p_2 \\ 0 & \text{otherwise} \end{cases}$

DEFINITION 3.2 ([8]) *For domains $\mathcal{A}$ and $\mathcal{B}$, both subsets of a universe $\mathcal{X}$,*

$$\delta(\mathcal{A}, \mathcal{B}) = \sum_{s \in \mathcal{X}} \delta(\Pr_{x \in \mathcal{A}}[x = s], \Pr_{y \in \mathcal{B}}[y = s])$$

(Note that $0 \leq \delta(\mathcal{A}, \mathcal{B}) \leq 1$.)

DEFINITION 3.3 ([8]) *Domains $\mathcal{A}$ and $\mathcal{B}$ are ϵ-close if $\delta(\mathcal{A}, \mathcal{B}) \geq 1 - \epsilon$.*

Lemma 1 ([8]) *If domains $\mathcal{A}$ and $\mathcal{B}$ are ϵ-close and δ is the probability of x lying in a bad set, S, when x is picked uniformly from $\mathcal{A}$, then the probability of y belonging to S when y is picked uniformly from $\mathcal{B}$ is at most $\epsilon + \delta$.*

Lemma 2 *For a fixed $x \in \mathcal{X}_1$, the domains $\mathcal{T}_j$ and $\{x + t | t \in \mathcal{T}_j\}$ are ϵ_1-close, where $\epsilon_1 = O(\frac{1}{n^2})$.*

`Proof:` Let $\mathcal{T}_j^i$ and $\mathcal{T}_j^{x,i}$ denote the domains from which the ith coordinate of the elements t and $x + t$ come from. We have

$\mathcal{T}_j^i = \{jz | z \in \mathcal{D}_{L_p, L_s}\}$
where $L_p = p(d+2)(n(d+1)!)^3$
and $L_s = s((d+1)!)^3$
and $\mathcal{T}_j^{x,i} = \{jz + x_i | z \in \mathcal{D}_{L_p, L_s}\}$
where x_i is the ith coordinate of x

Let q represent the probability that a randomly picked element of $\mathcal{T}_j^i$ belongs to $\mathcal{T}_j^{x,i}$. We have

$$q \geq 1 - \frac{(d+2)p/s}{L_p/L_s} \geq 1 - \frac{1}{n^3}$$

Since the size of the domains $\mathcal{T}_j$ and $\{x + t | t \in \mathcal{T}_j\}$ are equal, the quantity $\delta(\mathcal{T}_j, \{x + t | t \in \mathcal{T}_j\})$ is really equal to $\frac{|\mathcal{T}_j \cap \{x+t | t \in \mathcal{T}_j\}|}{|\mathcal{T}_j|}$. We have (for an appropriately chosen constant c)

$$\delta(\mathcal{T}_j, \{x + t | t \in \mathcal{T}_j\}) \geq q^n \geq 1 - \frac{c}{n^2} = 1 - \epsilon_1$$

□

Lemma 3 *For a fixed x, the domains $\{x + t | t \in \mathcal{X}_{ij}\}$ and $\mathcal{X}_{ij}$ are ϵ_2-close, where $\epsilon_2 = O(1/n^2)$.*

`Proof:` Similar to proof of Lemma 2. □

3.2 Tests.

We perform the following tests :

Test$_0$:
Repeat $O(\frac{1}{\delta}\log(\frac{1}{\beta}))$ times
Pick $k \in_R [d+1]$, $x \in_R \mathcal{X}_0$, $t \in_R \mathcal{T}_k$
Verify $\sum_{i=0}^{d+1} \alpha_i P(x+i*t) = 0$
Reject if the test fails more than $\delta/2$ fraction of the time

Test$_j$:
(done separately for each $j \in [d+1]$)
Repeat $O(\frac{1}{\delta}\log(\frac{1}{\beta}))$ times
Pick $k, l \in_R [d+1]$, $x \in_R \mathcal{X}_{kj}$, $t \in_R \mathcal{T}_l$
Verify $\sum_{i=0}^{d+1} \alpha_i P(x+i*t) = 0$
Reject if the test fails more than $\delta/2$ fraction of the time

Test$_{i,j}$:
(done for all pairs $i, j \in [d+1]$)
Repeat $O(\frac{1}{\delta}\log(\frac{1}{\beta}))$ times
Pick $k \in_R [d+1]$, $x \in_R \mathcal{X}_{ij}$, $t \in_R \mathcal{T}_k$
Verify $\sum_{l=0}^{d+1} \alpha_l P(x+l*t) = 0$
Reject if the test fails more than $\delta/2$ fraction of the time

Next we list a set of properties, such that if a program P does not have these properties, it is very unlikely to pass the corresponding tests.

Property P_0:

$$\Pr_{k \in_R [d+1], x \in_R \mathcal{X}_0, t \in_R \mathcal{T}_k} [\sum_{i=0}^{d+1} \alpha_i P(x+i*t) = 0] \geq 1-\delta$$

Property P_j, $j \in [d+1]$:

$$\Pr_{k,l \in_R [d+1], x \in_R \mathcal{X}_{kj}, t \in_R \mathcal{T}_l} [\sum_{i=0}^{d+1} \alpha_i P(x+i*t) = 0] \geq 1-\delta$$

Property P_{ij}, $i, j \in [d+1]$:

$$\Pr_{k \in_R [d+1], x \in_R \mathcal{X}_{ij}, t \in_R \mathcal{T}_k} [\sum_{l=0}^{d+1} \alpha_l P(x+l*t) = 0] \geq 1-\delta$$

From hereon we assume that the program P has all the above mentioned properties and show that such a program is essentially computing a degree d polynomial.

3.3 Proof of Correctness.

We define the function g to be

$$g(x) \equiv \text{majority}_{k\in[d+1],t\in\mathcal{T}_k}\{\sum_{i=1}^{d+1}\alpha_i P(x+i*t)\}$$

Lemma 4

$$\forall i,j \quad \Pr_{x\in\mathcal{X}_{ij}}[P(x)=g(x)] \geq 1-2\delta$$

`Proof:` Fix i and j and consider all $x \in \mathcal{X}_{ij}$ such that

$$P(x) = \text{majority}_{k\in[d+1],t\in\mathcal{T}_k}\{\sum_{i=1}^{d+1}\alpha_i P(x+i*t)\}$$

For such x's we have $g(x) = P(x)$. But by property P_{ij} and a straightforward counting argument the fraction of such x's is at least 2δ. □

Lemma 4′

$$\Pr_{x\in\mathcal{X}_0}[P(x)=g(x)] \geq 1-2\delta$$

`Proof:` Similar to proof of Lemma 4 above (using property P_0 instead of property P_{ij}). □

Lemma 5

$$\forall x \in \mathcal{X}_1, \quad \Pr_{k\in_R[d+1],t\in_R\mathcal{T}_k}[g(x) = \sum_{j=1}^{d+1} P(x+j*t)] \geq 1-\delta_1$$

where $\delta_1 = 2(d+1)(\delta+\epsilon_2)$

`Proof:` Consider $k,l \in_R [d+1]$ and $t_1 \in_R \mathcal{T}_k$ and $t_2 \in_R \mathcal{T}_l$. For a fixed $j \in [d+1]$, by property P_j and the fact that the domains $\{x+j*t_1 | t_1 \in \mathcal{T}_k\}$ and $\mathcal{X}_{jk}$ are ϵ_2-close we get that

$$\Pr[P(x+j*t_1) = \sum_{i=1}^{d+1} P(x+j*t_1+i*t_2)] \geq 1-\delta-\epsilon_2.$$

Similarly we get

$$\Pr[P(x+i*t_2) = \sum_{j=1}^{d+1} P(x+j*t_1+i*t_2)] \geq 1-\delta-\epsilon_2.$$

Summing up over $i \in [d+1]$ and $j \in [d+1]$ we get

$$\Pr[\sum_{j=1}^{d+1}\alpha_j P(x+j*t_1) = \sum_{i=1}^{d+1}\alpha_i P(x+i*t_2)]$$

$$\geq 1-2(d+1)(\delta+\epsilon_2) = 1-\delta_1.$$

(The probabilities in all the expressions above are for $k, l \in_R [d+1],\ t_1 \in_R \mathcal{T}_k$ and $t_2 \in_R \mathcal{T}_l$.) We have shown that with high $(1-\delta_1)$ probability, we get the same answer if we evaluate $\sum_{j=1}^{d+1} \alpha_j P(x+j*t)$ for a randomly chosen k, t two times. It is well known that this is a lower bound on the probability of the answer that is most likely to appear. Thus we have

$$\Pr_{k \in_R [d+1], t \in_R \mathcal{T}_k} [g(x) = \sum_{j=1}^{d+1} P(x+j*t)] \geq 1 - \delta_1.$$

□

Lemma 6

$$\forall x \in \mathcal{X}_1,\ \forall i, \quad \Pr_{t \in_R \mathcal{T}_i} [g(x) = \sum_{j=1}^{d+1} \alpha_j P(x+j*t)] \geq 1 - \delta_2$$

where $\delta_2 = (d+1)\delta_1$.

`Proof:` Lemma 5 guarantees that

$$\Pr_{k \in_R [d+1], t \in_R \mathcal{T}_k} [g(x) = \sum_{j=1}^{d+1} P(x+j*t)] \geq 1 - \delta_1$$

Thus the probability that this happens for a fixed $k = i$ must be at least $1 - (d+1)\delta_1$. □

Lemma 7

$$\forall x \in \mathcal{X}_1,\ \forall i, \quad \Pr_{t \in_R \mathcal{T}_i} [g(x) = \sum_{j=1}^{d+1} \alpha_j g(x+j*t)] \geq 1 - \delta_3$$

where $\delta_3 = \delta_2 + (d+1)(2\delta + \epsilon_2)$.

`Proof:` Lemma 6 says

$$\forall x \in \mathcal{X}_1,\ \forall i, \quad \Pr_{t \in_R \mathcal{T}_i} [g(x) = \sum_{j=1}^{d+1} \alpha_j P(x+j*t)] \geq 1 - \delta_2$$

Lemma 4 and the fact that the distributions $\{x+t | t \in \mathcal{X}_{ij}\}$ and $\mathcal{X}_{ij}$ are ϵ_2-close implies that

$$\Pr_{t \in_R \mathcal{T}_i} [g(x+j*t) = P(x+j*t)] \geq 1 - 2\delta - \epsilon_2$$

Putting them together we get

$$\forall x \in \mathcal{X}_1,\ \forall i,$$

$$\Pr_{t \in_R \mathcal{T}_i} [g(x) = \sum_{j=1}^{d+1} \alpha_j g(x+j*t)]$$

$$\geq 1 - \delta_2 - (d+1)(2\delta + \epsilon_2)$$

□

Lemma 8

$$\forall x, t \in \mathcal{X}_0, \quad \sum_{i=0}^{d+1} \alpha_i g(x + i * t) = 0$$

`Proof:` $t_1 \in_R \mathcal{T}$ implies $it_1 \in_R \mathcal{T}_i$. Lemma 7 implies

$$\forall x, t \in \mathcal{X}_0, \ \forall i,$$

$$\Pr_{t_1 \in_R \mathcal{T}}[g(x + i * t) = \sum_{j=0}^{d+1} \alpha_j g(x + i * t + j(it_1))]$$

$$\geq 1 - \delta_3$$

Also Lemma 7 and the fact that the distribution of $\{t + jt_1 | t_1 \in_R \mathcal{T}\}$ and $\mathcal{T}_j$ are ϵ_1-close implies

$$\forall x, t \in \mathcal{X}_0, \ \forall j,$$

$$\Pr_{t_1 \in_R \mathcal{T}}[g(x) = \sum_{i=0}^{d+1} \alpha_i g(x + i(t + j * t_1))]$$

$$\geq 1 - \delta_3 - \epsilon_1$$

Summing up we get

$$\Pr_{t_1 \in_R \mathcal{T}}[\sum_{i=0}^{d+1} \alpha_i g(x + i * t) = 0]$$

$$\geq 1 - (d+1)(2\delta_3 + \epsilon_1)$$

□

Theorem 1 *For $\delta = o(\frac{1}{d^3})$, there exists an $(0, \delta)$ self-tester for n-variate polynomials of total degree d over $(D_q = D^n_{L_p, L_s}, D_t = D^n_{p,s})$ (where $p, s \in \mathcal{Z}^+$ and $L_p = p(d+2)(n(d+1)!)^3$ and $L_s = s((d+1)!)^3$) which makes $O(d^3(\frac{1}{\delta}\log(\frac{1}{\beta})))$ calls to the program.*

`Proof:` The tester performs the tests given above. With probability at least $1 - \beta$ a program P which does not have any of the properties listed above will not pass the test. On the other hand if a program P passes the tests above, then there exists a function g, which is a polynomial on the domain $\mathcal{X}_0$ (Lemma 8), which agrees with P on the domain $\mathcal{X}_0$ (Lemma 4′). Thus P is a "good" program. □

4 Efficient Testers for Polynomials

We present an improved total degree test for polynomials in this section. The algorithm is a simple modification of the algorithm given in [8]. The algorithm given in [8] picks a random point x and a random offset t, and verifies that the values of the program at the $d+2$ points $x + it, \ \ i \in [0, ..., d+1]$ satisfy the interpolation equations, or equivalently, lie on the same polynomial. The modification in this algorithm is to look at $10d$ points to make sure that they all lie on the same polynomial. This can be done almost exactly as before using the interpolation equations, and with only a constant factor more running time.

```
program Improved-Total-Degree-Test(P, ε, β)

Membership Test
  l ← 10d
  Repeat O((1/ε) log(1/β)) times
    Pick x, t ∈_R Z_p^n and test that
      ∃ a poly h, deg h ≤ d such that
        ∀j ∈ {0, ···, l},    h(x + jt) = P(x + jt)
  Reject P if the test fails more than
      an ε fraction of the time.
```

Using the method of finite differences described in the appendix, it is very easy to determine whether or not there exists a polynomial that agrees with the function at the queried places. This can be done using only $O(d^2)$ *subtractions and no multiplications.*

Theorem 2 *If* $\epsilon < \frac{4}{300(d+1)}$*, and* P *does not* ϵ*-compute some polynomial* g *of total degree at most* d *on* Z_p^n*, then Program* Improved Total Degree Test *rejects* P *with probability* $1 - \beta$*.*

If a program P passes the test in [8] for $\epsilon < 1/O(d^2)$ then one can infer that the program is essentially computing a degree d polynomial. We show that if P passes the above test for $\epsilon < 4/(300d + 1)$ then one can infer that the program is essentially computing a degree d polynomial. The rest of this section is devoted to the presentation of the proof for $n = 1$. Some minor modifications are needed to prove the theorem for $n > 1$, which we omit in this version.

Before we prove the theorem, we first mention a technical lemma concerning a property of polynomials that will play a central role in our proof. This is a slight generalization of the techniques used in [8]. Essentially what the lemma shows is that suppose we have a $(d + 2) \times (d + 2)$ matrix M, of points (x, y), such that y is a function of x and:

1. For all columns and all but one row a degree d polynomial can be associated with the column or row such that the function values of the points in that column/row lie on the associated polynomial. The associated polynomials are not assumed to be the same.

2. Matrix M is a submatrix (obtained by deleting rows and columns) of a larger matrix A where A has the property that the x coordinates of the points from any of its rows (or any of its columns) are in an arithmetic progression.

Then the function values of the points on the remaining row also lie on some degree d polynomial.

Lemma 9 (Matrix Transposition Lemma) *Let* $\bar{a} =< a_0, ..., a_{d+1} >$ *and* $\bar{b} =< b_0, ..., b_{d+1} >$*. Let* $M_{\bar{a},\bar{b}} = \{m_{ij} | i, j \in \{0, \cdots, d+1\}\}$ *be a matrix such that* $m_{ij} \in Z_p$ *is of the form* $m_{ij} = x + a_i.t_1 + b_j.t_2 + a_i.b_j.t_3$*, and let* ϕ *be a function from* Z_p *to* Z_p *such that*

1. $\forall i \in \{1, \cdots, d+1\}$, $\exists$ *a polynomial* r_i *of degree at most* d *such that* $\forall j \in \{0, \cdots, d+1\}$, $\phi(m_{ij}) = r_i(m_{ij})$. *(*i.e. all of the function values of the points in row r_i are on the same degree d polynomial for rows 1 through $d+1$.*)*

2. $\forall j \in \{0, \cdots, d+1\}$, $\exists$ *a polynomial* c_j *of degree at most* d *such that* $\forall i \in \{0, \cdots, d+1\}$, $\phi(m_{ij}) = c_j(m_{ij})$. *(*i.e. all of the function values of the points in column c_j are on the same degree d polynomial for columns 0 through $d+1$.*)*

Then there also exists a polynomial r_0 *such that* $\forall j \in \{0, \cdots, d+1\}$, $r_0(m_{0j}) = \phi(m_{0j})$. *(*i.e. all of the points in row 0 are *also* on the same degree d polynomial*).*

The proof of this lemma is in the appendix.

We now turn our attention to showing that the algorithm satisfies the claims of the theorem:

`Proof:`

DEFINITION 4.1 *We define* δ *to be*

$$\delta \equiv \Pr_{x,t}[\nexists \text{ polynomial } h \text{ of degree at most } d \text{ s.t.}$$
$$\forall j \in \{0, \cdots, 10d\} \;\; P(x+jt) = h(x+jt)]$$

It is easy to see that if δ is at least 2ϵ, then the program is unlikely to pass the test. From here on we assume that we have a program P with $\delta \leq \frac{8}{300(d+1)}$.

The proof follows the same basic outline as the one in [8], but in order to achieve the better efficiency, we use ideas that can be thought of in terms of error-correction. Thus many of the steps that were quite simple in [8] require more work here. We define a self-corrector for f which uses P as an oracle. We show that this self-corrector has the nice property that for every x, self-correction using two different random strings h_1 and h_2 yields the same result (with high probability). We define a function g in terms of the self-corrector function, and we show that g has the following properties:

1. $g(x) = P(x)$ with probability at least $1 - 2\delta$ if x is picked randomly from Z_p.
2. For all x, $g(x)$ lies on the same polynomial (of degree at most d) as at least two-thirds of the points $g(x+1), g(x+2), \cdots, g(x+3d+3)$.

We then show that any function that has the second property mentioned above is a polynomial of degree d.

In [8], the function g was defined to be the value that occurs most often (for most t) when one looks at the evaluation at x of the unique polynomial which agrees with the values of P at $x+t, ..., x+(d+1)t$. Here we view the values of a polynomial f at $x+t, ..., x+10dt$ as a code word. Intuitively, the values of P at $x+t, ..., x+10dt$ will often have enough good information in it to allow us to get back to a correct codeword. The function g defined below can be thought of as the value that occurs most often (for most t) when one looks at the polynomial defined by the *error correction* of the values of P at $x+t, ..., x+10dt$ evaluated at x.

We introduce notation which we will be using in the rest of this section.

DEFINITION 4.2 *Given a set of points $S \subset Z_p$ and a function ϕ from Z_p to Z_p, the* **most likely** *polynomial fitting ϕ at these points is a polynomial h of degree at most d, with the property that it maximizes $|\{x \in S | h(x) = \phi(x)\}|$.* In case there is more than one candidate for the most likely polynomial, one of them is chosen arbitrarily.

We now define a self-corrector for the function f using P as an oracle.

DEFINITION 4.3 *Let h be the most likely polynomial fitting P on the set $\{x + jt | j \in \{1, \cdots, 10d\}$. Then $SC^P(x,t)$ is defined to be $h(x)$, if $h(x + jt) = P(x + jt)$ for all but 10δ fraction of the values of j. $SC^P(x,t)$ is defined to be* `error` *otherwise.*

We define g as follows

DEFINITION 4.4

$$g(x) \equiv \text{majority}_{t \in Z_p}\{SC^P(x,t)\}$$

In case there is more than one candidate for $g(x)$, one of them can be chosen arbitrarily. However, it will be shown that this is never the case.

Our first lemma shows that P and g agree at most places.

Lemma 10 $\Pr_{x \in_R Z_p}[g(x) = P(x)] \geq 1 - 2\delta$

The proof of Lemma 10 is the similar to the proof of Lemma 7 in Section 3 of [8] and is omitted here.

The next two lemmas show that the self-corrector function has the property that for all $x \in Z_p$, it is well-defined - most t's in Z_p give the same answer when used to evaluate $SC^P(x,t)$:

Lemma 11 *For all $x \in Z_p$,*

$$\Pr_{t_1, t_2 \in_R Z_p}\left[SC^P(x,t_1) = SC^P(x,t_2) \neq \texttt{error}\right] \geq 4/5$$

`Proof:` In [8], it is shown that if one computes the value of a polynomial function at x by interpolating from the values of the function along offset t_1 which in turn are computed by interpolating from the values of the function along offset t_2, then one would get the same answer as if one had computed the value of the function at x by interpolating from the values of the function along offset t_2 which in turn are computed by interpolating from the values of the function along offset t_1. This is not hard to see because it turns out that an interpolation can be thought of as a weighted sum, and so it amounts to changing the order of a double summation. Here the self-correction function is actually an interpolation of the *error-correction* of the values of the function, which is no longer a simple algebraic function of the observed values. We avoid analyzing this function by reducing the problem to the non-error-correcting version.

Consider the matrix $A = \{a_{ij} | i, j \in \{0, \cdots, 10d\}\}$, where $a_{ij} = x + it_1 + jt_2$. *(These are all the values that would affect $SC^P(x,t_1)$ if instead of using $P(x + it_1)$, wherever its value is needed, we used the value $SC^P(x + it_1, t_2)$.)* Let r_i denote the most likely polynomial fitting P on the elements in row i of A and let c_j denote the most likely polynomial fitting P on the elements in column j. Call a row i, $i \geq 1$,

(column j, $j \geq 1$) *good* if *all* entries in the ith row (jth column) have the property that $P(a_{ij}) = r_i(a_{ij})$ ($P(a_{ij}) = c_j(a_{ij})$). Otherwise, call it *bad*. By our definition of δ and since for $i \in \{1, ..., d+1\}$, $x + it_1$ and $x + it_2$ are uniformly distributed in Z_p, we get that

$$\Pr_{t_1, t_2 \in_R Z_p} [\text{row } i \text{ is } \textit{good}\,] \geq 1 - \delta$$

By applying Markov's inequality we get that

$$\Pr_{t_1, t_2 \in_R Z_p} [\text{At most } 10\delta \text{ fraction of the rows are } \textit{bad}\,] \quad \geq \quad 9/10$$

Similarly we get

$$\Pr_{t_1, t_2 \in_R Z_p} [\text{At most } 10\delta \text{ fraction of the columns are } \textit{bad}\,] \quad \geq \quad 9/10$$

Now, consider a $(d+2) \times (d+2)$ submatrix consisting of the row 0 and the $d+1$ *good* rows $e_1, ..., e_{d+1}$, and any $d+2$ *good* columns $f_1, ..., f_{d+1}$. The Matrix Transposition Lemma with $\phi \leftarrow P; (a_1, ..., a_{d+1}) \leftarrow (e_1, ..., e_{d+1}); (b_1, ..., b_{d+1}), \leftarrow (f_1, ..., f_{d+1}); t_3 \leftarrow 0$ guarantees that the $d+2$ points on the row 0 lie on some polynomial r_0. Repeated application by choosing other *good* columns show that all points in the 0th row from *good* columns lie on the same polynomial r_0. Thus $SC^P(x, t_1) = r_0(x)$. A similar argument based on *good* rows shows that there exists a polynomial c_0 such that all points from column 0 and *good* rows lie on this polynomial, implying $SC^P(x, t_2) = c_0(x)$. Showing that $r_0(x) = c_0(x)$ will conclude the proof.

Consider a $d+2 \times d+2$ submatrix of A containing the row 0, $d+1$ *good* rows and the column 0 and $d+1$ *good* columns. Apply the Matrix Transposition Lemma with $\phi = P$ everywhere except when the argument is x where $\phi(x) = r_0(x)$. The Matrix Transposition Lemma guarantees that all points from the 0th column lie on c_0, implying that $c_0(x) = r_0(x)$. □

The following lemma follows immediately and we state it without proof.

Lemma 12 $\forall x$, $\Pr_{t \in Z_p} \left[g(x) = SC^P(x, t) \right] \geq 4/5$.

Lemma 13 $\forall x$, $g(x) = h(x)$, *where h is the unique most likely polynomial fitting g at the points* $\{x+1, x+2, \cdots, x+3d+3\}$.

`Proof:` As in the proof of Lemma 11 we construct a matrix $A = \{a_{ij} | i \in \{0, ..., 3d+3\}, j \in \{0, ..., 10d\}\}$, where $a_{ij} = (x+i) + j(t_1 + it_2) = (x + jt_1) + i(1 + jt_2)$. The rows of this matrix are the elements that affect $SC^P(x + i, t_1 + it_2)$. The columns are a subset of the elements that affect $SC^P(x + jt_1, 1 + jt_2)$. Call a row i, of A *good* if $g(x+i) = SC^P(x+i, it_1 + t_2)$. By Lemma 11 we know that

$$\Pr_{t_1, t_2 \in_R Z_p} [\text{row } i \text{ is } \textit{good}] \geq 4/5$$

Using Markov's inequality we get

$$\Pr_{t_1, t_2 \in_R Z_p} [\text{at least two-thirds of the rows are } \textit{good}] \quad \geq \quad 2/5$$

Thus we find that with probability at least 1/5, two-thirds of the rows are good and row 0 is good. From now on we will work with a submatrix of A containing the 0th row and $2d+2$ other *good* rows. Let r_i be the most likely polynomial fitting P on points from the ith row. Call an element m_{ij} *bad* if $P(m_{ij}) \neq r_i(m_{ij})$. Delete all columns of M which contain *bad* elements. Since we are working with only the *good* rows of A, any row can have at most $10d \cdot 10\delta$ bad elements. Thus we delete at most $300(d^2+d)\delta$ columns. Hence we will still be left with at least $2d$ columns (since $\delta \leq \frac{8}{300(d+1)}$). Let c_j be the most likely polynomial fitting P on elements from column j. Delete all columns containing elements such that $c_j(m_{ij}) \neq P(m_{ij})$. By the definition of δ and application of Markov's inequality we find that with probability at least $1/10$, no more than $d-1$ columns have such elements, and so $d+1$ of the columns remain (this assumes that $d \geq 4$, if $d < 4$ then the algorithm in [8] is better). Applying the Matrix Transposition Lemma to all $(d+2) \times (d+2)$ submatrices shows that all the elements $g(x+i)$, i is a good row, lie on the same polynomial. Thus we get that at least 2/3rds of the points $g(x+i)$ and the point $g(x)$ lie on the same polynomial. In other words, $g(x) = h(x)$ where $h(x)$ is the most likely polynomial fitting g on the points $\{x+i | i \in \{1, \cdots, 3d+3\}$. Also since h agrees with at least two-thirds of the points of the set, it must be the unique most likely polynomial fitting these points. □

Our next lemma shows that the condition guaranteed above suffices to show that g is a degree d polynomial.

Lemma 14 *If ϕ is a function from Z_p to Z_p such that for all x, $\phi(x) = h(x)$ where h is the unique most likely polynomial, of degree at most d, fitting ϕ on the points $\{x+i | i \in \{1, \cdots, 3d+3\}\}$. Then ϕ is a degree d polynomial.*

`Proof:` We prove this by claiming that the unique most likely polynomial fitting ϕ on any set of the form $\{x+i | i \in \{1, \cdots, 3d+3\}\}$ is the same. Assume otherwise. Then there must exist an x such that the majority polynomial fitting ϕ on the sets $\{x+i | i \in \{1, \cdots, 3d+3\}\}$ and $\{x+i | i \in \{0, \cdots, 3d+2\}\}$ are different. But this cannot be since $\phi(x) = h(x)$ where h is the most likely polynomial fitting ϕ on the set $\{x+i | i \in \{1, \cdots, 3d+3\}\}$. Hence the most likely polynomial fitting ϕ in any set of the form $\{x+i | i \in \{1, \cdots, 3d+3\}\}$ is the same, say h and hence for all $x \in Z_p,\ \phi(x) = h(x)$ □

□

5 Open Questions

We have shown that programs for polynomial functions can be self-tested over rational domains. A very important question along these lines is to determine whether there is an approximate self-tester [8] for programs which approximate polynomial functions over rational domains, as would occur in floating point and fixed precision computation (an approximate self-tester verifies that a program gives a good approximation to the function on a large fraction of the inputs).

Secondly, we have shown stronger bounds on the efficiency of a simple modification of the algorithm for self-testing polynomials in [8]. This algorithm makes $O(d)$ tests, for a total of $O(d^2)$ calls to the program. It would be interesting to show whether or not $O(1)$ tests of this algorithm or a slight modification of it are sufficient.

6 Acknowledgements

We are very grateful to Avi Wigderson for suggesting that we look for more efficient testers for polynomials, as well as his technical help in proving the theorems in Section 4. We thank Mike Luby, Oded Goldreich and Joan Feigenbaum for their comments on the writeup of this paper.

References

[1] Babai, L., Fortnow, L., Lund, C., "Non-Deterministic Exponential Time has Two-Prover Interactive Protocols", *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*,, 1990.

[2] Beaver, D., Feigenbaum, J., "Hiding Instance in Multioracle Queries", *Proceedings of Symposium on Theoretical Aspects of Computer Science 1990*.

[3] Blum, M., "Designing programs to check their work", Submitted to *CACM*.

[4] Blum, M., Kannan, S., "Program correctness checking ... and the design of programs that check their work", *Proc. 21st ACM Symposium on Theory of Computing*, 1989.

[5] Blum, M., Luby, M., Rubinfeld, R., "Self-Testing/Correcting with Applications to Numerical Problems," *Proc. 22th ACM Symposium on Theory of Computing*, 1990.

[6] Feige, U., Goldwasser, S., Lovasz, L., Safra, M., Szegedy, M., "Approximating Clique is Almost NP-Complete", *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*,, 1991.

[7] Freivalds, R., "Fast Probabilistic Algorithms", Springer Verlag Lecture Notes in CS No. 74, Mathematical Foundations of CS, 57-69 (1979).

[8] Gemmell, P., Lipton, R., Rubinfeld, R., Sudan, M., Wigderson, A., "Self-Testing/Correcting for Polynomials and for Approximate Functions", *Proc. 23th ACM Symposium on Theory of Computing*, 1991.

[9] Lipton, R.J., "New directions in Testing," in Distributed Computing and Cryptography, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 2, 1991, pp. 191–202.

[10] Lund, C., "The Power of Interaction", University of Chicago Technical Report 91-01, January 14, 1991.

[11] Shen, Sasha, personal communication, May 1991.

[12] Van Der Waerden, B.L., *Algebra*, Vol. 1, Frederick Ungar Publishing Co., Inc., pp. 86-91, 1970.

A Appendix

I. Proof of Lemma 9

We give the proof of Lemma 9.

Lemma 9 (Matrix Transposition Lemma) Let $\bar{a} =< a_0, ..., a_{d+1} >$ and $\bar{b} =< b_0, ..., b_{d+1} >$. Let $M_{\bar{a},\bar{b}} = \{m_{ij} | i, j \in \{0, \cdots, d+1\}\}$ be a matrix such that $m_{ij} \in Z_p$ is of the form $m_{ij} = x + a_i.t_1 + b_j.t_2 + a_i.b_j.t_3$, and let ϕ be a function from Z_p to Z_p such that

1. $\forall i \in \{1, \cdots, d+1\}$, $\exists$ a polynomial r_i of degree at most d such that $\forall j \in \{0, \cdots, d+1\}$, $\phi(m_{ij}) = r_i(m_{ij})$. (*i.e. all of the function values of the points in row r_i are on the same degree d polynomial for rows 1 through $d+1$.*)

2. $\forall j \in \{0, \cdots, d+1\}$, $\exists$ a polynomial c_j of degree at most d such that $\forall i \in \{0, \cdots, d+1\}$, $\phi(m_{ij}) = c_j(m_{ij})$. (*i.e. all of the function values of the points in column c_j are on the same degree d polynomial for columns 0 through $d+1$.*)

Then there also exists a polynomial r_0 such that $\forall j \in \{0, \cdots, d+1\}$, $r_0(m_{0j}) = \phi(m_{0j})$. (*i.e. all of the points in row 0 are* also *on the same degree d polynomial*).

`Proof:` By standard arguments (cf. [9]), the existence of polynomials $r_1, \cdots, r_{d+1}$ implies there exist constants $e_0, e_1, \cdots, e_{d+1}$ such that $\forall i \in \{1, \cdots, d+1\}$, $\sum_{j=0}^{d+1} e_j r_i(m_{ij}) = 0$. *The constants depend only on $\bar{b}$ and d.* Similarly there exist constants $f_1, \cdots, f_{d+1}$, depending only on $\bar{a}$ and d, such that $\forall j \in \{0, \cdots, d+1\}$, $c_j(m_{0j}) = \sum_{i=1}^{d+1} f_i c_j(m_{ij})$. Thus we find that

$$\begin{aligned}
\sum_{j=0}^{d+1} e_j \phi(m_{0j}) &= \sum_{j=0}^{d+1} e_j c_j(m_{0j}) \\
&= \sum_{j=0}^{d+1} e_j \sum_{i=1}^{d+1} f_i c_j(m_{ij}) \\
&= \sum_{i=1}^{d+1} f_i \sum_{j=0}^{d+1} e_j c_j(m_{ij}) \\
&= \sum_{i=1}^{d+1} f_i \sum_{j=0}^{d+1} e_j r_i(m_{ij}) \\
&= 0
\end{aligned}$$

Since the function values at these points satisfy the interpolation identities, we know that there must exist a polynomial r_0 such that $\forall j \in \{0, \cdots, d+1\}$, $r_0(m_{0j}) = \phi(m_{0j})$. □

II. Method of Finite Differences.

Here we consider the following problem:

Given k evenly-spaced points $x_1, x_2, \cdots, x_k \in Z_p^n$, $x_{i+1} - x_i = x_i - x_{i-1} = h$, and values of some function f at these points:

Does there exist a polynomial h, of degree at most d, such that $\forall i \in [k],\ h(x_i) = f(x_i)$?

We describe how to use the method of finite differences to test for this property. The test performs $O(kd)$ subtractions and no multiplications.

Define the function $f^1(x) = f(x) - f(x-t)$. In general, define f^i by $f^i(x) = f^{i-1}(x) - f^{i-1}(x-t)$. The method of finite differences implies the following:

Lemma 15 ([12]) *f agrees with some degree d polynomial on the points $x_1, \cdots, x_k$ if and only if the $d+1^{st}$ difference $f^{d+1}(x_j) = 0$ for $j \in \{d+2, \cdots, k\}$.*

Using this Lemma we find that we only have to evaluate the f^i's, for $i \in [d+1]$, to check if the above property holds, and this can be done using $O(kd)$ subtractions.