Checking Polynomial Identities over any Field: Towards a Derandomization?

Daniel Lewin*
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
danl@theory.lcs.mit.edu

Salil Vadhan†
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
salil@math.mit.edu

Abstract

We present a Monte Carlo algorithm for testing multivariate polynomial identities over any field using fewer random bits than other methods. To test if a polynomial $P(x_1, \ldots, x_n)$ is zero, our method uses $\sum_{i=1}^{n} \lceil \log(d_i + 1) \rceil$ random bits , where d_i is the degree of x_i in P, to obtain *any* inverse polynomial error in polynomial time. The algorithm applies to polynomials given as a black box or in some implicit representation such as a straight-line program. Our method works by evaluating P at truncated formal power series representing square roots of irreducible polynomials over the field. This approach is similar to that of Chen and Kao [CK97], but with the advantage that the techniques are purely algebraic and apply to any field. We also prove a lower bound showing that the number of random bits used by our algorithm is essentially optimal in the black-box model.

1 Introduction

Checking multivariate polynomial identities of the form $P_1(x_1, \ldots, x_n) = P_2(x_1, \ldots, x_n)$ is a problem central to both algorithm design and complexity theory. Algorithms such as the RNC algorithm for perfect matching [Lov79, MVV87, CRS95], the BPP algorithm for testing equivalence of read-once branching programs [BCW80], and one of the randomized algorithms for testing multiset equality [BK95] rely on efficiently checking if a multivariate polynomial is identically zero. Results in complexity theory such as IP = PSPACE [LFKN90, Sha90], MIP = NEXPTIME [BFL90], and NP = PCP$(\log n, 1)$ [AS92, ALM+92] all fundamentally rely on viewing a boolean assignment not as a group of bits, but as the values of a multivariate polynomial. Testing if such a polynomial is identically zero is a procedure used frequently in this context. In addition, many results in learning theory, and sparse multivariate polynomial interpolation also rely on checking polynomial identities [Zip79, GKS90, CDGK91, RB91].

Clearly, the problem is easy if the input polynomials are given as lists of coefficients (known as standard reduced form). However, in many cases the polynomials are given in some implicit representation such as a symbolic determinant or as a product of multiple polynomials. Reducing a polynomial in such a succinct representation to its standard form can take exponential time in the length of the description since there could be an exponential number of non-zero coefficients that need to be determined. A property of many succinct representations is that despite the fact that the reduced standard form of the polynomial may have exponential size, it is possible to evaluate the polynomial at a given point in only polynomial time. For example, the determinant can be evaluated in polynomial time, as can a polynomial-sized product of polynomials.

Many randomized methods for checking polynomial identities have been discovered based on the assumption that the polynomials can be evaluated efficiently. The basic scheme is to use randomization to select a number of sample points on which the identity is checked by evaluation. The test accepts if the identity is found to hold at all the sample points and rejects otherwise. Schwartz and Zippel discovered in [Sch80] and [Zip79] that the probability that a non-zero multivariate polynomial evaluates to zero is small as long as the point is selected at random from a large enough domain. In a recent development, Chen and Kao [CK97] showed how to check if a polynomial with integer coefficients is zero using fewer random bits than the Schwartz-Zippel method. Their method is to evaluate the polynomial at approximations of easily computable irrational points. An innovative feature of Chen and Kao's algorithm is that the error probability of the test can be decreased by doing more computation instead of increasing the number of random bits used. The main drawback of Chen and Kao's algorithm is that it only applies to polynomials with *integer* coefficients.

In this paper we extend Chen and Kao's work by showing how to achieve the same result in *any* field. Our result is obtained by uncovering the essential ingredients of Chen and Kao's algorithm and abstracting them. We obtain a purely

*URL: `http://theory.lcs.mit.edu/~danl`. Supported by the love of his wife and kids.

†URL: `http://www-math.mit.edu/~salil`. Supported by an NDSEG/DOD Graduate Fellowship and partially by DARPA grant DABT63-96-C-0018.

algebraic formulation of the algorithm while Chen and Kao's description relies on the structure of the real numbers. We view uncovering this algebraic structure as a step towards the derandomization of polynomial identity checking.

Using the Schwartz-Zippel lemma and a simple counting argument, one can show that there exists a set S of $\mathrm{poly}(s, d)$ points, so that any nonzero multivariate polynomial of 'description size' at most s and degree at most d evaluates to non-zero on at least one of the points of S. Finding such a set of points *deterministically* would be a major breakthrough, as it would imply the derandomization of all polynomial identity testing, a long standing open problem. Even when P is restricted to symbolic determinants with entries that are linear forms in the input variables, it is not known how to construct such a set explicitly.

We view our work (as well as that of [CK97]) as restricting the domain in which one has to search for a set of "good points." Our purely algebraic approach, in contrast to that of [CK97], results in a highly structured domain, whose algebraic properties might give insight into the search for good evaluation points.

Any derandomization of polynomial identity testing must take advantage of the polynomials' succinct representation. Indeed, we show in Section 7 that the number of random bits used by our algorithm is essentially optimal in the *black-box* model, where description size is not used.

1.1 Formal Setting

Let F be a field. For most of the paper, we assume that a multivariate polynomial $P(x_1, \ldots, x_n)$ with coefficients in F is described by an efficient procedure for evaluating P given values for $x_1, \ldots, x_n$. Such a procedure can, for example, be described by a *straight-line program* doing computations in F. For example, P could be a symbolic determinant over F, and the procedure would be any efficient method for computing the determinant. We also consider the *black-box model*, in which P is represented by a "black box" which, given a point $(x_1, \ldots, x_n) \in F^n$, evaluates P at that point. In Appendix A, we define and discuss both straight-line programs and the black-box model,

We concentrate on algorithms for checking if the polynomial $P(x_1, \ldots, x_n)$ is zero since any polynomial identity can be transformed into this form.

1.2 Previous Algorithms

1.2.1 Schwartz-Zippel

The first randomized test was discovered by both Schwartz and Zippel. The method is based on the following famous lemma.

Lemma 1.1 ([Sch80, Zip79]) *Let d be the (total) degree of $P(x_1, \ldots, x_n)$. Let S be a set of size at least Cd. If P is not identically zero, then $P(s_1, \ldots, s_n) = 0$ with probability at most $\frac{1}{C}$, where $s_1, \ldots, s_n$ are chosen uniformly and at random from S.*

This lemma immediately implies the following test:

1. Choose a random point $(s_1, \ldots, s_n)$ from S^n, where $S \subseteq F$, and $|S| = 2d$.
2. Evaluate $P(s_1, \ldots, s_n)$ using the procedure supplied for P.
3. Output `'nonzero'` if $P(s_1, \ldots, s_n) \neq 0$, else output `'probably zero'`.

One technicality is that if the field F has fewer than $2d$ elements in it, then there is no set S large enough to be used in the algorithm. In this case, S can be selected from an extension field of F and P is evaluated over the extension field.

Clearly if P is the zero polynomial, the test always outputs `'probably zero'` which is the correct answer. On the other hand, Lemma 1.1 implies that if $P \neq 0$, then the test is wrong with probability no more than $\frac{1}{2}$. That is, the error probability is at most $\frac{1}{2}$. The algorithm clearly uses $n \log 2d$ random bits.

As discussed in [CK97], there are three basic methods to reduce the error probability of the Schwartz-Zippel algorithm to $1/t$ for an arbitrary t. The first is to perform $\log t$ independent repetitions of the above test, using $(\log t)(n \log 2d)$ random bits. The second is to enlarge the size of S to be td (possibly moving to an extension field of F) thus using $n \log td$ random bits. The third, which works for $t \leq 2^{n \log 2d}$ is to perform t pairwise independent repetitions of the algorithm, thus using $2n \log 2d$ random bits.

1.2.2 Chen-Kao

Recently, Chen and Kao [CK97] discovered a new algorithm for testing if a multivariate polynomial is identically zero. Their algorithm uses fewer random bits than the algorithm of Schwartz-Zippel in order to obtain a given error probability. Chen and Kao's algorithm only works for polynomials with *integer* coefficients.

Chen and Kao's basic strategy is to evaluate the polynomial $P(x_1, \ldots, x_n)$ at a set of irrational points $\pi_1, \ldots \pi_n \in \mathbb{R}$. In their algorithm, each π_i is a sum of a small number of square roots of primes: $\pi_i = \sum_j \sqrt{p_{ij}}$. They show that $P(\pi_1, \ldots, \pi_n) = 0$ if and only if P is identically zero. That is, if you can evaluate the polynomial P at this single point, then you can check if P is identically zero!

Unfortunately, this does not immediately imply a testing algorithm since P needs to be evaluated at infinite precision irrational numbers. To get around this problem, Chen and Kao approximate each $\sqrt{p_{ij}}$ by r_{ij} which is obtained by truncating the binary expansion of $\sqrt{p_{ij}}$ at the ℓ'th position. They then show that if P is evaluated at the points $\overline{\pi}_i = \sum_j \sigma_{ij} r_{ij}$ where σ_{ij} is *randomly chosen* to be $+1$ or -1, then the error probability drops proportionally to $1/\ell$. This implies the surprising result that any inverse polynomial error can be achieved in polynomial time while using the same number of random bits!

For reference, we roughly describe the Chen-Kao algorithm below:

Let d_i be the degree of x_i in P.

1. **Find Primes:** Find the first $\sum_i \log(d_i + 1)$ primes p_{ij}, $1 \le i \le n, 1 \le j \le \log(d_i + 1)$.
2. **Approximate Square Roots:** Compute the r_{ij}'s by computing the first ℓ bits of $\sqrt{p_{ij}}$.
3. **Add Randomization:** Set $\overline{\pi}_i = \sum_j \sigma_{ij} r_{ij}$ where σ_{ij} is randomly chosen to be $+1$ or -1.
4. **Evaluate Polynomial:** Output `'nonzero'` if $P(\overline{\pi}_1, \dots, \overline{\pi}_n) \neq 0$, else output `'probably zero'`.

From this description, we see that the Chen-Kao algorithm uses $\sum_i \log_2(d_i + 1)$ random bits to achieve any inverse polynomial error probability in polynomial time. This can be substantially lower than the number of random bits used by Schwartz-Zippel, which is at least $n \log_2(2d)$ to achieve an error probability of $1/2$. In the simple case that P is a multilinear polynomial of degree n, Chen-Kao use n random bits to achieve any inverse polynomial error while Schwartz-Zippel use $n \log n$ random bits to achieve error $1/2$.

1.3 Our Contribution

At first glance, it seems that the techniques of Chen and Kao cannot be extended to finite fields, since there are no clear notions of primes or approximations in finite fields. This seems to imply that testing polynomial identities over the integers is somehow easier than over an arbitrary field.

In this paper we show that this is not the case. We obtain results comparable to those of Chen and Kao that hold for polynomials with coefficients from any field F. More specifically, we show that over any field F it is possible to test if a multivariate polynomial $P(x_1, \dots, x_n)$ is zero with *any* inverse polynomial error probability in polynomial time, using only $\sum \log_2(d_i + 1)$ random bits (d_i is the degree of x_i in P).

The first obstacle in extending Chen and Kao's approach is the lack of "primes" in arbitrary fields (or even in finite fields). We overcome this by extending our view from the field F to the ring of polynomials $F[x]$. Now, it seems natural that irreducible polynomials over F take the place of the primes in Chen and Kao's algorithm. But what is a square root of an irreducible polynomial? Clearly, irreducible polynomials do not have square roots that are polynomials, but it turns out that they may have roots which are *infinite power series*![1]

For example, consider the polynomial $x+1$ over the field with three elements. The square root of this polynomial as an infinite power series is:

$$1 + 2x + x^2 + x^3 + 2x^4 + \dots$$

This notion of a root implies a natural extension of the notion of approximation. Namely, approximations are obtained by truncating infinite power series at some power x^ℓ (which can be viewed as taking the series *modulo* x^ℓ). For example, the approximation of the square root of $x+1$ modulo x^2 in the field of three elements is the *polynomial* $1+2x$.

Thus, the intuition behind our algorithm can be summed up in the following table:

Primes	$\rightarrow$	Irreducible Polynomials in $F[x]$
Square Roots	$\rightarrow$	Infinite Power Series over F
Approximation	$\rightarrow$	Square Roots mod x^ℓ

Using this analogy, a rough description of our algorithm reads much the same as Chen and Kao's algorithm:

1. **Find Irreducible Polynomials:** Find $\sum_i \log(d_i + 1)$ distinct irreducible polynomials p_{ij} ($1 \le i \le n$, $1 \le j \le \log(d_i + 1)$ that have square roots as infinite power series.
2. **Approximate Square Roots:** Compute approximations r_{ij} to the square roots $\sqrt{p_{ij}}$ modulo x^ℓ. Note that r_{ij} is a polynomial of degree $\ell - 1$.
3. **Add Randomization:** Set $\overline{\pi}_i = \sum_j \sigma_{ij} r_{ij}$ where σ_{ij} is randomly chosen to be $+1$ or -1. Note that the $\overline{\pi}_i$ are univariate polynomials!
4. **Evaluate Polynomial:** Output `'nonzero'` if $P(\overline{\pi}_1, \dots, \overline{\pi}_n) \neq 0 \pmod{x^\ell}$. Note that we evaluate P after a *univariate* polynomial has been substituted in place of each of its variables.

We show that the error probability of this test can be reduced, in polynomial time, to *any* inverse polynomial quantity by using approximations modulo larger powers of x.

1.4 Layout of the Paper

Section 3 describes some standard algebraic tools which our algorithm uses. Section 4 gives a more detailed description of our algorithm, along with an example. In Section 5, we prove the correctness of our algorithm. The analysis of the algorithm makes use of techniques that may be useful in other applications involving multivariate polynomials.

Section 6 describes the extension of our algorithm to fields of characteristic 2. In Section 7, we show that in the black-box model our algorithm uses an essentially optimal number of random bits. Section 8 shows how the ideas in this paper can be used to obtain a purely algebraic alternative to Chen and Kao's algorithm.

2 Definitions

Let F be a field of characteristic $\neq 2$. We denote by $F[x]$ the ring of polynomials over the field F, and by $F(x)$ the field of fractions of $F[x]$; in other words, $F(x)$ is the field of rational functions over F. The ring of formal power series over F is denoted $F[[x]]$. We denote by $F[\alpha]$ the field extension of F obtained by adjoining to F an algebraic element α. For most of the paper, we assume that the field we are working over is not of characteristic 2. The case of characteristic 2 is dealt with in Section 6.

Throughout the paper, we count arithmetic operations in F as single steps. In places, we also need to efficiently

[1] This is assuming that the field is not of characteristic 2. This case is treated in Section 6.

enumerate some number m of distinct elements of F, and we count this as taking m steps. In applications using finite fields, for example, these conventions are reasonable for standard representations, as these procedures can be implemented with only a $\mathrm{poly}(n)$ factor slowdown, where n is the number of bits needed to represent elements of the field.

3 Algebraic Tools

As stated in the Introduction, our algorithm requires finding power series approximations to square roots of irreducible polynomials. Thus we must:

1. Characterize which irreducible polynomials have square roots.

2. Describe how to find such polynomials.

3. Explain how to find approximations to their square roots.

In this section, we handle Items 1 and 2. Item 3 is a standard technique, so it is deferred to Appendix B. Our goal is to exhibit efficient algorithms for Items 2 and 3. However, in the interest of clarity, we do not always describe the most efficient algorithms that are known.

3.1 Which Irreducible Polynomials Have Square Roots

Not every irreducible polynomial has a square root as a formal power series. For example, over the rationals, the polynomial $x - 3$ is irreducible, but does *not* have a square root as a formal power series since the constant term of the series would have to be $\sqrt{3}$, which is irrational. This example shows that for an irreducible polynomial to have a square root, its constant term must be a quadratic residue. Surprisingly, in fields of characteristic $\neq 2$, this condition is also sufficient! This is a special case of a very useful construction called *Hensel Lifting*. Hensel lifting has been used for other algorithmic purposes, such as factoring sparse multivariate polynomials [Zip79, Zip81, Kal82, vzGK85].

Hensel Lifting is described in two parts. First we state Hensel's Lemma which characterizes when a polynomial equation with coefficients in $F[x]$ has a root in $F[[x]]$. For example, finding a square root of a polynomial $f(x) \in F[x]$ can be viewed as finding a root in $F[[x]]$ of $Z^2 - f = 0$. In Appendix B, we describe a standard technique for finding approximations to these roots, given that they exist.

Lemma 3.1 (Hensel's Lemma [Eis95, Cor. 7.4]) *Let $S(Z)$ be a polynomial with coefficients in $F[x]$. S can be viewed as a bivariate polynomial $S(Z, x)$ over F. If there is a $g \in F$ such that:*

1. $S(g, 0) = 0$.

2. $S_Z(g, 0) \neq 0$ where $S_Z(Z, x) = \frac{\partial S(Z,x)}{\partial Z}$.

Then, there exists a $\tilde{g}(x) \in F[[x]]$ such that $S(\tilde{g}(x), x) = 0$.

Say we have an irreducible polynomial $f(x) \in F[x]$. We can use Lemma 3.1 to find the conditions under which $f(x)$ has a square root in $F[[x]]$. Let $S(Z, x)$ be the polynomial $S(Z, x) = Z^2 - f(x)$. The two conditions of Lemma 3.1 are:

1. $S(g, 0) = g^2 - f_0 = 0$ where f_0 is the constant term of f. So, g is a square root of f_0 in F.

2. $S_Z(g, 0) = 2g \neq 0$. This is true as long as $f_0 \neq 0$, and F is not of characteristic 2.

So, from Lemma 3.1 and our previous discussion it follows that $f(x)$ has a square root as a formal power series if and only if f_0 is a quadratic residue over F.

3.2 Finding Irreducible Polynomials With Square Roots

Lemma 3.1 tells us that any irreducible polynomial with a constant term that is a quadratic residue in F, has a square root in $F[[x]]$ (assuming that F is not of characteristic 2). As a subroutine of our algorithm we need to be able to find k such polynomials in $F[x]$. Clearly, if we can find k distinct, monic, irreducible polynomials in $F[x]$, then by multiplying each by its constant term we obtain a set of k irreducible polynomials that have square roots in $F[[x]]$.

Luckily, finding k monic, irreducible polynomials in F is not hard. In fact, if we don't care about being as efficient as possible we can just hunt for them by brute force. That is, go through the monic elements of $F[x]$ one by one in order of degree, and check if any of the irreducible polynomials found so far divides them. If none do, we have another irreducible polynomial, otherwise we move on to the next element of $F[x]$. The following lemma says that if we want to find k polynomials this way, we don't have to search very far.

Lemma 3.2 [2] *Let F be a finite field, and $F[x]$ the ring of polynomials over F. Then, the number of irreducible polynomials of degree at most n in $F[x]$ is at least $(|F|^n - 1)/n$.*

The proof of Lemma 3.2 can be found in the full version of the paper [LV98].

If the number k of irreducible polynomials we are looking for is less than $|F|$, we only need to use degree 1 polynomials: $x - e_1, x - e_2, \ldots, x - e_k$ where $e_i \in F$. However, if $k > |F|$ then Lemma 3.2 says that we do not have to go over more than polynomial in k elements of $F[x]$ until we find k monic, irreducible polynomials.

We have seen how to find a set of irreducible polynomials that have square roots as power series, but how can we find approximations to the square roots efficiently? Luckily, there is a well known method for finding square roots modulo x^ℓ using $\mathrm{poly}(\ell)$ algebraic operations in F. This procedure is described in Appendix B.

[2] Actually, in analogy to the famous Prime Number Theorem over $\mathbb{Z}$, it is known that the number of irreducible polynomials of degree n over F is asymptotic to $|F|^n/n$.

4 The Algorithm

In this section, we give a formal description of our algorithm for testing if a multivariate polynomial is zero. The algorithm is described and then a simple example of how the algorithm runs is presented. The proof of correctness of the algorithm is in Section 5.

Inputs to the algorithm:

1. A multivariate polynomial $P(x_1, \dots, x_n) \in F[x_1, \dots, x_n]$ described by a straight-line program or given as a black box.

2. Upper bounds d_i, $1 \leq i \leq n$ on the maximum degree of x_i in the polynomial P, and an upper bound d on the total degree of P.

3. The desired probability of error, ϵ, $0 < \epsilon \leq 1$.

Both straight-line programs and the black-box model are defined and discussed in Appendix A. In most applications, the structure of P can be used to obtain the degree bounds. For example, if P is a symbolic determinant, then the degree of any variable is not more that the number if times it appears in the matrix (with multiplicity).

The algorithm: On input: P, d, d_i, $1 \leq i \leq n$, and ϵ:

Find Irreducible Polynomials: Find $\sum_i \lceil \log(d_i + 1) \rceil$ irreducible polynomials p_{ij} ($1 \leq i \leq n$, $1 \leq j \leq \lceil \log(d_i + 1) \rceil$) that have square roots as infinite power series. Do this by computing by brute force the first $\sum_i \lceil \log(d_i + 1) \rceil$ monic, irreducible polynomials and multiplying them by their constant term.

Approximate Square Roots: Set $\ell = \frac{1}{\epsilon} \left(\frac{d \max(\deg(p_{ij}))}{2} \right)$. Compute approximations r_{ij} to the square roots of the p_{ij}'s modulo x^ℓ, using the Hensel Lifting algorithm described in Appendix B. That is, compute $r_{ij} = \sqrt{p_{ij}}$ modulo x^ℓ.

Add Randomization: Set $\overline{\pi}_i = \sum_j \sigma_{ij} r_{ij}$ where σ_{ij} is randomly chosen to be $+1$ or -1.

Evaluate Polynomial: Output `'nonzero'` if $P(\overline{\pi}_1, \dots, \overline{\pi}_n) \neq 0 \pmod{x^\ell}$, else output `'probably zero'`. Note that we evaluate P after a univariate polynomial has been substituted in place of each of its variables. Appendix A explains how to accomplish this in both the straight-line model and the black-box model.

Our main theorem follows:

Theorem 4.1 *Given a straight-line program for a polynomial $P(x_1, \dots, x_n)$ over a field F of characteristic $\neq 2$, the above algorithm has the following properties:*

1. *If $P(x_1, \dots, x_n)$ is the zero polynomial, then the algorithm always outputs* `'probably zero'`.

2. *If $P(x_1, \dots, x_n)$ is* not *zero, then the probability that the algorithm outputs* `'probably zero'` *is no more than ϵ.*

3. *The number of random bits used is $\sum_i \lceil \log(d_i + 1) \rceil$.*

4. *The running time is polynomial in d, n, $1/\epsilon$, and the length of the straight-line program, counting arithmetic operations in F as one step.*

When P is given as a black-box and $|F| > \ell d$, the same properties hold, counting evaluations of P as a single step in the running time.

It is clear that the number of random bits used is as stated. Running time is also straightforward to verify; more detail is given in the full version of the paper [LV98]. The main task is to prove the correctness of the algorithm; this is done in Section 5.

Note that as in Chen and Kao's algorithm, we can decrease the error probability without using a single additional random bit!

4.1 An Example

Say you are given a straight-line program over the field with three elements for the polynomial:

$$x_1 x_2 + 2x_1 + 2x_2 + 1$$

The checking procedure uses two random bits. The polynomial is multilinear, so only two irreducible polynomials are needed. Searching by brute force gives: $x + 1$ and $x^2 + 1$. The power series roots of these polynomials are:

$$\sqrt{x+1} = 1 + 2x + x^2 + x^3 + 2x^4 + \cdots$$
$$\sqrt{x^2+1} = 1 + 2x^2 + x^4 + x^6 + 2x^8 + \cdots$$

Now, set $\pi_1 = \sigma_1 \sqrt{x+1} \pmod{x^\ell}$, and $\pi_2 = \sigma_2 \sqrt{x^2+1} \pmod{x^\ell}$ for $\sigma_i = \pm 1$. The algorithm outputs `'nonzero'` if $\pi_1 \pi_2 + 2\pi_1 + 2\pi_2 + 1 \pmod{x^\ell} \neq 0$. To see how the algorithm works, we try this for $\ell = 1, 2, 3, 4$, and get the following table:

σ_1	σ_2	mod x	mod x^2	mod x^3	mod x^4
1	1	$= 0$	$= 0$	$= 0$	$\neq 0$
1	-1	$= 0$	$\neq 0$	$\neq 0$	$\neq 0$
-1	1	$= 0$	$= 0$	$\neq 0$	$\neq 0$
-1	-1	$= 0$	$\neq 0$	$\neq 0$	$\neq 0$

Note that as we use better approximations of the square roots and compute modulo larger powers of x, the probability of error (taken over the choice of σ_1, and σ_2), goes down. The number of random bits stays the same!

5 Analysis

In this section, we prove the correctness of the algorithm presented in the previous section. That is, we show that if the input polynomial is the zero polynomial, then the algorithm always outputs `'probably zero'`. On the other hand, if the polynomial is not identically zero, then we show that the algorithm makes a mistake with probability less than ϵ.

If P is the zero polynomial in $F[x_1, \ldots, x_n]$, then substituting the π_i's in place of the x_i's produces the zero polynomial in $F[x]$, which is zero modulo x^ℓ. Therefore, no matter what ℓ and the σ_i are, the algorithm outputs `'probably zero'`. Showing that the algorithm has error probability ϵ when $P \neq 0$ is more involved.

The first basic idea is that we extend our view from the field F to $F[x]$, and then to the field of fractions $F(x)$ (elements of $F(x)$ can be viewed as rational functions in x). Now, the polynomials $Z^2 - p_{ij}$ are irreducible in the ring $F(x)[Z]$, because the p_{ij} are irreducible in $F[x]$. Hence, we can look at the field extension of $F(x)$ obtained by adjoining to $F(x)$ all the elements $\sqrt{p_{ij}}$ which are the roots of the polynomials $Z^2 - p_{ij}$. This extension is denoted $K = F(x)[\sqrt{p_{ij}}]$.

The proof relies on the following lemma. Roughly, the lemma states that if we evaluate the polynomial P over *infinite* power series, instead of truncated ones, then the algorithm always correctly identifies polynomials that are non-zero.

Throughout the proof, we write e_i for $\lceil \log(d_i + 1) \rceil$ and denote by M the value $\sum_{i=1}^n e_i$.

Lemma 5.1 *Let σ_{ij} be $+1$ or -1, for $1 \le i \le n$, $1 \le j \le e_i$. For $1 \le i \le n$, let $\pi_i = \sum_{j=1}^{e_i} \sigma_{ij}\sqrt{p_{ij}}$.*

Then, if $P(x_1, \ldots, x_n)$ is a non zero polynomial in $F[x_1, \ldots, x_n]$, then $P(\pi_1, \ldots, \pi_n) \neq 0$ in K.

For example, the polynomial $x_1x_2 + 2x_1 + 2x_2 + 1$ over the field with three elements (reusing the example of Section 4.1) is not zero in $F[x_1, x_2]$. Lemma 5.1 states that if we evaluate $\sqrt{x+1}\sqrt{x^2+1} + 2\sqrt{x+1} + 2\sqrt{x^2+1} + 1$ in the field extension $F(x)[\sqrt{x+1}, \sqrt{x^2+1}]$, then we get a non-zero value.

Proof: We prove Lemma 5.1 by induction on n (the number of variables in the polynomial). For $n = 0$, the result is trivial, so we consider $n \ge 1$. Given $P(x_1, \ldots, x_n)$, we rewrite the polynomial as:

$$P(x_1, \ldots, x_n) = \sum_{i=0}^{d_n} x_n^i P_i(x_1, \ldots, x_{n-1})$$

Now, since we assume that P is not the zero polynomial, at least one of the $P_i(x_1, \ldots, x_{n-1})$ must be a non-zero polynomial. Hence, by the induction hypothesis we have that:

$$P(\pi_1, \ldots, \pi_{n-1}, x_n) = \sum_{i=0}^{d_n} x_n^i P_i(\pi_1, \ldots, \pi_{n-1})$$

is a non-zero univariate polynomial in x_n of degree no more than d_n (with coefficients in $F(x)[\pi_1, \ldots, \pi_{n-1}]$). The following claim, whose proof is in Appendix C, demonstrates that π_n cannot be a root of this polynomial.

Claim 5.2 *π_n is of degree $\ge d_n + 1$ over $F(x)[\pi_1, \ldots, \pi_{n-1}]$.*

This concludes the proof of lemma 5.1. $\square$

For each possible selection of the signs σ_{ij}, we call $P(\pi_1, \pi_2, \ldots, \pi_n)$ a *conjugate* of P. Using this terminology, choosing random σ_{ij}'s can be viewed as choosing a *random conjugate* from the 2^M possible conjugates.

Lemma 5.1 says that if we could compute efficiently with infinite power series, then, no matter which conjugate we choose (by choosing the σ_{ij}), $P(\pi_1, \ldots, \pi_n)$ is non-zero as long as P is non-zero. However, since we cannot compute using *infinite* power series we truncate the π_i by doing all operations modulo x^ℓ. Thus, the algorithm can viewed as evaluating a *random conjugate* modulo x^ℓ.

So, our goal is to show that not more than $\epsilon 2^M$ conjugates vanish modulo x^ℓ. One way to prove this is to show that the *product* of all the conjugates does not vanish modulo some larger power of x. Luckily, the product of the 2^M conjugates is a well studied object, and is called the *norm* of these conjugates.[3]

$$\begin{aligned}
\mathrm{norm} &= \prod_{\sigma \in \{\pm 1\}^M} P(\pi_1, \ldots, \pi_n) \\
&= \prod_{\sigma \in \{\pm 1\}^M} P\left(\sum_{j=1}^{e_1} \sigma_{1j}\sqrt{p_{1j}}, \ldots, \sum_{j=1}^{e_n} \sigma_{nj}\sqrt{p_{nj}}\right)
\end{aligned}$$

For example, say that our polynomial is $P(x_1, x_2) = x_1 + x_2$ over the field of three elements, and the irreducible polynomials are $x + 1$ and $x^2 + 1$. Then norm is:

$$\prod_{\sigma_1, \sigma_2 \in \{\pm 1\}} (\sigma_1\sqrt{x+1} + \sigma_2\sqrt{x^2+1}) = x^4 + x^3 + x^2$$

Note that the norm is a *polynomial* over F; all of the square roots cancel out. This is in fact a general phenomenon captured by the following claim which is proved in Appendix C:

Claim 5.3 $\mathrm{norm} \in F[x]$

Lemma 5.1 shows that $\mathrm{norm} \neq 0$ since each element of the product is non-zero. Recall, that our goal is to show that the norm does not vanish modulo some power of x, and since the claim states that the norm of P is in fact a nonzero polynomial over F, all we need to do is to upper bound its degree! We would like to show that the degree of the polynomial can't build up very much over the product of the 2^M conjugates. The problem is that the elements inside the product are *not* polynomials, and it is unclear what their "degree" is.

We solve this problem by defining a degree function $\deg : F[x, \sqrt{p_{11}}, \sqrt{p_{12}}, \ldots, \sqrt{p_{ne_n}}] \to \mathbb{N}$ with the following three properties:

1. $\deg(fg) = \deg(f) + \deg(g)$
2. $\deg(f+g) \le \max(\deg(f), \deg(g))$

[3] This norm is the usual Galois Theory norm over the field extension $F(x)[\sqrt{p_{11}}, \sqrt{p_{12}}, \ldots, \sqrt{p_{ne_n}}]$. Note that we are making implicit use of the fact that the Galois group is $(\mathbb{Z}/2\mathbb{Z})^M$, which follows from Kummer Theory. See Appendix C.

3. If $f \in F[x]$ then $\deg(f)$ is *equal* to the degree of f as a polynomial in x.

In order to define the degree function, we need the following claim, whose proof can be found in the full version of the paper [LV98].

Claim 5.4 *Every $f \in F[x, \sqrt{p_{11}}, \sqrt{p_{12}}, \ldots, \sqrt{p_{ne_n}}]$ can be* uniquely *represented in the form:*

$$f = \sum_{\alpha} f_{\alpha}(x) \prod_{i,j} \sqrt{p_{ij}}^{\alpha_{ij}}$$

where we sum over all vectors $\alpha \in \{0,1\}^M$ assigning 0 or 1 to each pair (i,j) with $1 \leq i \leq n$, $1 \leq j \leq e_n$, and where $f_{\alpha}(x)$ is an element of $F[x]$.

The degree function for f is defined using this unique representation:

$$\deg(f) = \max_{\alpha} \left(\mathrm{DEG}(f_{\alpha}) + \sum_{i,j:\ \alpha_{ij}=1} \frac{\mathrm{DEG}(p_{ij})}{2} \right),$$

where the $\max$ is taken over all non-zero summands in the unique representation of f, and DEG is the regular degree function on $F[x]$.

It is a simple matter to verify that this function has the three properties we want from the degree function. We remark that this definition of degree is actually *determined* by the three properties above, because they imply that the degree of $\sqrt{p_{ij}}$ must be half the degree of p_{ij}.

Since $\mathrm{norm} \in F[x]$, we know, by the last property of the degree function, that $\deg(\mathrm{norm})$ is the degree of the norm as a polynomial in $F[x]$. Now, using the other properties of the degree function we have:

$$\begin{aligned}
&\deg(\mathrm{norm}) \\
&= \deg\left(\prod_{\sigma \in \{\pm 1\}^M} P\left(\sum_{j=1}^{e_1} \sigma_{1j}\sqrt{p_{1j}}, \ldots, \sum_{j=1}^{e_n} \sigma_{nj}\sqrt{p_{nj}} \right) \right) \\
&= \sum_{\sigma \in \{\pm 1\}^M} \deg\left(P\left(\sum_{j=1}^{e_1} \sigma_{1j}\sqrt{p_{1j}}, \ldots, \sum_{j=1}^{e_n} \sigma_{nj}\sqrt{p_{nj}} \right) \right) \\
&\leq 2^M d \left(\frac{\max(\deg(p_{ij}))}{2} \right),
\end{aligned}$$

where d is the total degree of P and the $\max$ is taken over all p_{ij}'s.

Now, suppose that T conjugates vanish modulo x^{ℓ}. This means that norm must vanish modulo $x^{\ell T}$, so it must be true that:

$$\ell T \leq \deg(\mathrm{norm}) \leq 2^M d \left(\frac{\max(\deg(p_{ij}))}{2} \right)$$

Therefore we have:

$$\frac{T}{2^M} \leq \frac{d \max(\deg(p_{ij}))}{2\ell}$$

The left hand side of the above inequality is just the probability of choosing a "bad" conjugate; that is, one that vanishes modulo x^{ℓ}. Setting $\ell = \frac{1}{\epsilon}\left(\frac{d\max(\deg(p_{ij}))}{2}\right)$ bounds the probability of error by ϵ. This concludes the proof of Theorem 4.1.

6 Characteristic 2

In this section, we sketch an extension of our algorithm to fields of characteristic 2. The essential problem when F is of characteristic 2 is that *no* irreducible polynomials have square roots in $F[[x]]$. Instead, we have to work with *cube roots*. By Hensel's Lemma (Lemma 3.1), a polynomial in $F[x]$ has a cube root in $F[[x]]$ iff its constant term is a cube in F. Also, to choose a random conjugate of a cube root, one needs to multiply by a random cube root of unity, rather than ± 1. Thus, for now, we suppose that F contains a primitive cube root of unity ζ. (For finite F of characteristic 2, this is the case iff F is of order 2^k for k even.) Then the algorithm proceeds as follows:

1. **Find Irreducible Polynomials:** Find $\sum_i \log_3(d_i+1)$ irreducible polynomials p_{ij} ($1 \leq i \leq n$, $1 \leq j \leq \log_3(d_i+1)$) whose constant terms are cubes in F.
2. **Approximate Cube Roots:** Compute approximations r_{ij} to the cube roots $\sqrt[3]{p_{ij}}$ modulo x^{ℓ}. This can be done using a method similar to the one for finding approximations to square roots.
3. **Add Randomization:** Set $\overline{\pi}_i = \sum_{ij} \sigma_{ij} r_{ij}$ where σ_{ij} is randomly chosen in $\{1, \zeta, \zeta^2\}$.
4. **Evaluate Polynomial:** Output `'nonzero'` if $P(\overline{\pi}_1, \ldots, \overline{\pi}_n) \neq 0 \pmod{x^{\ell}}$.

The analysis of this algorithm proceeds much as in the characteristic $\neq 2$ case, and shows that $\ell = \epsilon^{-1}\mathrm{poly}(n,d)$ is sufficient to obtain error probability ϵ. The number of random bits used by this algorithm is essentially $(\log_2 3) \sum \log_3(d_i+1) = \sum \log_2(d_i+1)$, as before.

The only question that remains is what to do when F does not have a cube root of unity. In the straight-line model, this is easily dealt with: treat ζ as a formally adjoined cube root of 1, reducing ζ^2's to $-\zeta - 1$ when they arise in the computation.

In the black-box model, we treat $P(\overline{\pi}_1, \ldots, \overline{\pi}_n)$ as a bivariate polynomial $g(x, \zeta)$ of degree less than ℓd in x and at most $2d$ in ζ. Analogous to the argument in Appendix A.2, it suffices to substitute $(\ell d) \cdot (2d+1)$ values for (x, ζ) to distinguish between the cases that P is the zero-polynomial and the case that the real value for $P(\overline{\pi}_1, \ldots, \overline{\pi}_n)$ does not modulo x^{ℓ}. This requires the field to be of size at least $d\ell(2d+1)$.

7 A Lower Bound

In this section we show that in the black box model our algorithm uses essentially an optimal number of random bits. The lower bound implies that description size *must* be taken into account in order to fully derandomize polynomial identity checking. In particular, the lower bound implies that the degrees of the variables, d_i, are *not* a good description of size for use in derandomization.

Theorem 7.1 *Let P be a polynomial over a field F that has n variables and let d_i be the degree of the variable x_i. Let A be any randomized algorithm that has only black-box access to P and has the following properties:*

- *Makes $T(n, d_1, \dots, d_n)$ queries to the black box.*
- *Outputs* `'probably zero'` *with prob. 1 if $P = 0$.*
- *Outputs* `'nonzero'` *with positive prob. if $P \neq 0$.*

Then A must use at least $\sum_{i=1}^{n} \log_2(d_i + 1) - \log_2(T(n, d_1, \dots, d_n))$ random bits. In particular, if $T(n, d_1, \dots, d_n) \leq \mathrm{poly}(n)$, then A must use

$$(1 - o(1)) \sum_{i=1}^{n} \log_2(d_i + 1)$$

random bits.

Proof: There are $\prod_{i=1}^{n}(d_i + 1)$ possible monomials in the polynomial P (choosing the degree of each variable which can be between 0 and d_i). The job of the algorithm can be viewed as checking if all the coefficients of these monomials are zero or not. Each query to the black box at a given point of F^n gives a linear equation on the coefficients of the polynomial.

We first show that any *deterministic* algorithm that always gives the right answer (i.e. outputs `'probably zero'` if and only if $P = 0$), must make at least $\prod_{i=1}^{n}(d_i + 1)$ queries to the black box. Consider the situation after the algorithm makes k queries to the black box. If the answers on all these k queries is zero, then we have a system of k linear, homogeneous equations on the coefficients C_i. If $k < \prod_{i=1}^{n}(d_i + 1)$, then there is a non-zero solution to the system — which represents a non-zero polynomial that is indistinguishable from the zero polynomial to the algorithm. Thus, any deterministic algorithm must make at least $\prod_{i=1}^{n}(d_i + 1)$ queries to the black box before it is able to output a correct answer.

Now consider a randomized algorithm that uses r random bits and has the properties in the statement of the theorem. We "derandomize" the algorithm and get a deterministic algorithm for the problem by trying all of the 2^r possible random coin tosses. Now, by the above argument, this deterministic algorithm must make at least $\prod_{i=1}^{n}(d_i + 1)$ queries to the black box. Thus:

$$2^r T(n, d_1, \dots, d_n) \geq \prod_{i=1}^{n}(d_i + 1)$$

and therefore:

$$r \geq \sum_{i=1}^{n} \log_2(d_i + 1) - \log_2(T(n, d_1, \dots, d_n))$$

$\square$

8 Another Algorithm over the Integers

In this section, we mention how the ideas in this paper yield a purely algebraic alternative to Chen and Kao's algorithm over the integers. The main observation, following from a more general form of Hensel's Lemma, is that any prime p that is congruent to 1 modulo 8 has a square root in the 2-adic integers [Eis95, Sec. 7.2]. Moreover, there is a natural notion of approximate solutions in the 2-adics, namely solutions modulo 2^ℓ. Thus our algorithm over $\mathbb{Z}$ and its analysis proceed much as in the finite field case, using the following analogy:

Irreducible polynomials	$\rightarrow$	Prime numbers
F[[x]]	$\rightarrow$	2-adics
Square roots mod x^ℓ	$\rightarrow$	Square roots mod 2^ℓ

The use of the 2-adics is inessential and can be replaced with the q-adics for any fixed prime q.

Acknowledgments

We are grateful to Madhu Sudan for his comments on this manuscript. We thank Amit Sahai for collaboration at an early stage of this research, and Dan Spielman for useful conversations on this topic. We also thank W. Russell Mann for numerous conversations about algebraic concepts relevant to this work. We also thank Amos Beimel for useful conversations, especially on the lower bound, and for commenting on a version of this paper.

References

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the Thirty Third Annual Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs. In *Proceedings of the Thirty Third Annual Symposium on Foundations of Computer Science*, pages 2–13, 1992.

[BCW80] M. Blum, A.K. Chandra, and M.N. Wegman. Equivalence of free Boolean graphs can be tested in polynomial time. *Information Processing Letters*, 10:80–82, 1980.

[BFL90] László Babai, Lance Fortnow, and Carsten Lund. Nondeterministic exponential time has two-prover interactive protocols. In *31st Annual Symposium on Foundations of Computer Science*, volume I, pages 16–25, St. Louis, Missouri, 22–24 October 1990. IEEE.

[BK95] M. Blum and S. Khanna. Designing programs that check their work. *Journal of the Association for Computing Machinery*, 42:269–291, 1995.

[CDGK91] Michael Clausen, Andreas Dress, Johannes Grabmeier, and Marek Karpinski. On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. *Theoretical Computer Science*, 84(2):151–164, 29 July 1991.

[CK97] Zhi-Zhong Chen and Ming-Yang Kao. Reducing randomness via irrational numbers. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 200–209, El Paso, Texas, 4–6 May 1997.

[CRS95] Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. Randomness-optimal unique element isolation with applications to perfect matching and related problems. *SIAM Journal on Computing*, 24(5):1036–1050, October 1995.

[Eis95] David Eisenbud. *Commutative Algebra with a View Toward Algebraic Geometry*. Graduate Texts in Mathematics 150. Springer-Verlag, 1995.

[GKS90] Dima Yu. Grigoriev, Marek Karpinski, and Michael F. Singer. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM Journal on Computing*, 19(6):1059–1063, December 1990.

[Kal82] Erich Kaltofen. A polynomial reduction from multivariate to bivariate integral polynomial factorization. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 261–266, San Francisco, California, 5–7 May 1982.

[Kal88] Erich Kaltofen. Greatest common divisors of polynomials given by straight-line programs. *Journal of the Association for Computing Machinery*, 35(1):231–264, January 1988.

[Lan93] Serge Lang. *Algebra*. Addison-Wesley, 3 edition, 1993.

[LFKN90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proofs. In *Proceedings of the Thirty First Annual Symposium on Foundations of Computer Science*, pages 1–10, 1990.

[Lov79] L. Lovasz. On determinants, matchings, and random algorithms. In L. Budach, editor, *Fundamentals of Computing Theory*. Akademia-Verlag, 1979.

[LV98] Daniel Lewin and Salil Vadhan. Checking polynomial identities over any field: Towards a derandomization? *Available from* `http://theory.lcs.mit.edu/~danl` *or* `http://www-math.mit.edu/~salil`, 1998.

[MVV87] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[RB91] Ron M. Roth and Gyora M. Benedek. Interpolation and approximation of sparse multivariate polynomials over $GF(2)$. *SIAM Journal on Computing*, 20(2):291–314, April 1991.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the Association for Computing Machinery*, 27(4):701–717, October 1980.

[Sha90] Adi Shamir. IP=PSPACE. In *Proceedings of the Thirty First Annual Symposium on Foundations of Computer Science*, pages 11–15, 1990.

[Str72] V. Strassen. Berechnung und programm I. *Acta Informatica*, 1:320–335, 1972. In German.

[vzGK85] Joachim von zur Gathen and Erich Kaltofen. Factoring sparse multivariate polynomials. *Journal of Computer and System Sciences*, 31(2):265–287, October 1985.

[Zip79] R. E. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of EUROSAM '79*, pages 216–226. Springer-Verlag, 1979. Lecture Notes in Computer Science, vol. 72.

[Zip81] R. Zippel. Newton's iteration and the sparse Hensel algorithm. In *Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation*, pages 68–72, Utah, 1981.

A Two Models for Implicitly Given Polynomials

A.1 Straight-Line Programs

Informally, a *straight-line program* [Str72, Kal88] describes a polynomial by a sequence of algebraic operations. More precisely, let D be a ring, $S \subset D$ be a finite set of constants, and $x_1, \ldots, x_n$ be a set of input variables. Then a straight-line program P is a sequence of m statements, where the i'th statement has one of the following forms:

$$y_i \leftarrow x_j \quad \text{for } 1 \le j \le n$$
$$y_i \leftarrow s \quad \text{for some } s \in S$$
$$y_i \leftarrow \begin{cases} y_j & + & y_k \\ y_j & - & y_k \\ y_j & \times & y_k \\ y_j & \div & y_k \end{cases} \quad \text{for some } j, k < i$$

The *output* of P is defined to be y_m. It is clear that every such program defines an easily-computable rational function $P(x_1, \ldots, x_n)$ on D^n, assuming arithmetic in D is feasible. We say that a straight-line program *defines a polynomial*, if the formal expression in the input variables resulting from following the steps of the straight-line program is in fact a polynomial in $x_1, \ldots, x_n$ and if, for every $\alpha \in D^n$ all divisions occurring in the steps of P on input α are actually divisions by invertible elements of D. Many polynomial functions of interest, such as the determinant, can be expressed as straight-line programs.

Recall that our algorithm requires evaluating a multivariate polynomial $P(x_1, \ldots, x_n)$ defined by a straight-line program at univariate polynomials $\overline{\pi}_1(x), \ldots, \overline{\pi}_n(x)$ modulo x^ℓ. We can do this by simply interpreting the straight-line program for P (over a field F) as a straight-line program over the larger ring $R = F[x]/(x^\ell)$. We need to check two things: evaluating P at points of R^n only results in results in divisions by invertible elements of R, and that these divisions can be done efficiently. To see this, consider the evaluation of P on $(h_1(x), \ldots, h_n(x)) \in R^n$. Taking every step of this evaluation *modulo* x, it is easy to see that we obtain the evaluation of P on $(a_1, \ldots, a_n) \in F^n$, where $a_1, \ldots, a_n$ are the constant terms of $h_1, \ldots, h_n$. We know that evaluating P on elements of F^n never results in division by 0, so whenever P attempts to invert an element of R, it must be an element with nonzero constant term. The technique in Appendix B for inverting $g_\ell(x)$ during Hensel lifting shows that every element of R with nonzero constant term is invertible and that this inverse can be computed with $\mathrm{poly}(\ell)$ operations in F.

Remark. When discussing straight-line programs over finite fields, there is some ambiguity in the statement $P(x_1, \ldots, x_n) = 0$. It could mean that the polynomial obtained by applying the steps of P to the indeterminates $x_1, \ldots, x_n$ is the *zero element* of the ring $F[x_1, \ldots, x_n]$. Or it could mean that P defines the *zero function* on F^n; that is, $P(a_1, \ldots, a_n) = 0$ for all $a_1, \ldots, a_n \in F$. Although these two conditions are equivalent over infinite fields, they are not in finite fields. For example, the polynomial $x^q - x$ vanishes at all points of $\mathrm{GF}(q)$ but is not the zero element of $\mathrm{GF}(q)[x]$. The two notions are equivalent, however, whenever $|F|$ is greater than the degree d_i of P in each variable x_i. When this condition does not hold, *our algorithms test whether $P(x_1, \ldots, x_n)$ is the zero element of the ring* $F[x_1, \ldots, x_n]$. We note that the Schwartz-Zippel approach requires that the the field is larger than the *total degree* to work at all, whereas our algorithm is meaningful even when the field is $\mathrm{GF}(2)$.

A.2 The Black-Box Model

The definition of this model is as one would expect — instead being given a description of P, our algorithm is given oracle access to a "black-box" that will evaluate P at any point of F^n. In this case, we cannot directly evaluate P at univariate polynomials $\overline{\pi}_1(x), \ldots, \overline{\pi}_n(x)$. Instead, we observe that the univariate polynomial $g(x) = P(\overline{\pi}_1(x), \ldots, \overline{\pi}_n(x))$ has degree less than ℓd, where d is the total degree of P, because each $\overline{\pi}_i(x)$ has degree less than ℓ. Moreover, we can evaluate g at any point of F using the black-box for P. Suppose we evaluate g at ℓd distinct points of F. If all the values obtained are zero, then g must be the zero polynomial so it certainly vanishes modulo x^ℓ and our algorithm should output `'probably zero'`. However, if at least one of the values is nonzero, then P must be a nonzero polynomial and our algorithm should output `'nonzero'`. Note that this approach works whenever $|F| > \ell d$. This type of restriction on degree is typical of identity-testing algorithms in the black-box model (cf., [CDGK91])

B Finding Approximations to Square Roots

In this section, we describe how to find approximations to square roots of a polynomial. The method we describe constructs an approximation modulo $x^{2\ell}$ given an approximation modulo x^ℓ. This is similar to what can be done for Newton approximation.

Say we are trying to approximate the square root of the irreducible polynomial $f(x) \in F[x]$. Let $g_\ell(x)$, $\ell = 1, 2, 3, \ldots$ be successive approximations of $\sqrt{f(x)}$. That is,

$$g_\ell(x)^2 = f(x) \pmod{x^\ell}$$

The first approximation, $g_1(x)$, is simply the square root of f_0 in F: $g_1(x) = \sqrt{f_0}$. (Notice that, in our algorithm, we always construct the polynomial f so that we know the square root of the constant term .)

Now, assume that we have found the ℓ'th approximation, $g_\ell(x)$, such that $g_\ell(x)^2 = f(x) \pmod{x^\ell}$. The 2ℓ'th approximation has the form:

$$g_{2\ell}(x) = x^\ell p(x) + g_\ell(x),$$

where $p(x)$ is a polynomial of degree $\ell - 1$. We want to find a $p(x)$ so that $g_{2\ell}(x)^2 = f(x) \pmod{x^{2\ell}}$. Substituting for $g_{2\ell}$, this is equivalent to

$$2x^\ell p(x) g_\ell(x) + g_\ell(x)^2 = f(x) \pmod{x^{2\ell}}$$

Since $g_\ell(x)^2 = f(x) \pmod{x^\ell}$, we know that $f(x) - g_\ell(x)^2$ is divisible by x^ℓ and we obtain:

$$\frac{f(x) - g_\ell(x)^2}{2x^\ell} = p(x) g_\ell(x) \pmod{x^\ell}$$

The polynomial $g_\ell(x)$ has an inverse in $F[[x]]$ which can be found by the following trick. Write $g_\ell(x) = g_0 + x g_\ell^*(x)$, and then note that:

$$\begin{aligned} \frac{1}{g_\ell(x)} &= \frac{1}{g_0\left(1 + \frac{x g_\ell^*(x)}{g_0}\right)} \\ &= \frac{1}{g_0}\left(1 - \frac{x g_\ell^*(x)}{g_0} + \frac{x^2 g_\ell^*(x)^2}{g_0^2} - \frac{x^3 g_\ell^*(x)^3}{g_0^3} + \cdots\right) \end{aligned}$$

Since $p(x)$ has degree $\ell - 1$, we have:

$$p(x) = \left(\frac{f(x) - g_\ell(x)^2}{2x^\ell}\right)\left(\frac{1}{g_\ell(x)}\right) \pmod{x^\ell}$$

So, we can find $p(x)$ by computing $\frac{1}{g_\ell(x)} \pmod{x^\ell}$ using the trick, and plugging into the above equation.

C Algebraic Lemmas

We lead up to the proofs of Claims 5.2 and 5.3 with a few intermediate facts. For notational convenience, let $\{r_1, \ldots, r_M\} = \{p_{ij} \colon 1 \le i \le n, 1 \le j \le e_i\}$. Recall that $r_1, \ldots, r_m \in F[x]$ are irreducible polynomials, and we were studying the field extension $K = F(x)[\sqrt{r_1}, \ldots, \sqrt{r_n}]$ over $F(x)$. First we obtain the degree and Galois group of this extension using Kummer theory [Lan93, VI, Thm. 8.1]. The full derivation of the Galois group can be found in the full version of this paper [LV98]. The result is:

Lemma C.1 *The Galois group of $K/F(x)$ consists exactly of automorphisms σ of the form $\sigma(\sqrt{r_i}) = \sigma_i \sqrt{r_i}$ for any $(\sigma_1, \ldots, \sigma_M) \in \{\pm 1\}^M$.*

We now proceed to the proofs of Claims 5.2 and 5.3.

Proof (of Claim 5.2): Clearly, it suffices to show that for $i \le j \le M$, $\alpha = \sqrt{r_{i+1}} + \cdots + \sqrt{r_j} \in K$ has degree at least 2^{j-i} over L, where $L = F(x)[\sqrt{r_1}, \ldots, \sqrt{r_i}]$. Let $f \in L[x]$ be the irreducible polynomial for α over L. For any $\sigma_{i+1}, \ldots, \sigma_j \in \{\pm 1\}$, there is an automorphism σ of K fixing L and taking $\sqrt{r_k}$ to $\sigma_k \sqrt{r_k}$ for $i < k \le j$. (By our description of the Galois group of $K/F(x)$.) Notice that α has 2^{j-i} distinct images under such automorphisms. For any such σ, we have $f(\sigma(\alpha)) = \sigma(f(\alpha)) = 0$, since σ is an automorphism fixing L. Thus, f has at least 2^{j-i} roots, and α is of degree at least 2^{j-i} over L. □

Proof (of Claim 5.3): First observe that by the characterization of the Galois group of $K/F(x)$ above, norm is in fact the usual Galois-theoretic norm of $P(\pi_1, \ldots, \pi_n)$ from K to $F(x)$. And, since norms always lie in the base field (see [Lan93, VI,Thm 5.1]), $\mathrm{norm} \in F(x)$. But we need to prove that norm is in $F[x]$, not $F(x)$. This follows from the fact that norm is "integral" over $F[x]$ using standard theorems, namely [Lan93, VII, Cor. 1.6] and [Lan93, VII, Prop. 1.7]. The complete proof can be found in the full version of this paper [LV98]. □