CS 880: Pseudorandomness and Derandomization 1/28/2013

Lecture 2: Pseudorandom Generators and Extractors

Instructors: Holger Dell and Dieter van Melkebeek Scribe: Nick Pappas

In the previous lecture we described the two main topics of the course: derandomization and randomness extraction. In this lecture we introduce two key constructs in the pursuit of these topics, namely pseudorandom generators and extractors, respectively. We also review some background on finite fields that will be needed in future lectures.

1 Pseudorandom Generators

1.1 Definition

Intuitively, a pseudorandom generator (PRG) is a procedure that generates a pseudorandom distribution. A PRG shares the parameters of the underlying pseudorandom distribution: the class $\mathcal{A}$ of algorithms to which the generated distribution looks random, and a bound ϵ on how different the pseudorandom distribution can look from the uniform one for algorithms from $\mathcal{A}$.

Definition 1 (Pseudorandom generator). *An $\epsilon(r)$-pseudorandom generator for $\mathcal{A}$ is a sequence of mappings $G = (G_r)_{r\in\mathbb{N}}$ where $G_r : \{0,1\}^{\ell(r)} \mapsto \{0,1\}^r$ such that $(G_r(U_{\ell(r)}))_{r\in\mathbb{N}}$ is $\epsilon(r)$-pseudorandom for $\mathcal{A}$.*

Thus, a PRG generates a pseudorandom distribution by taking a sample σ (called the seed) from the uniform distribution on $\{0,1\}^{\ell(r)}$, and outputting $G_r(\sigma)$. In addition to the parameters of the underlying pseudorandom distribution, there are two more that interest us:

- the seed length $\ell(r)$, which we want to be as small as possible as a function of r, and
- the computational complexity of the transformations G_r, which we would like to be low as a function of its input length $\ell(r)$.

1.2 Use for derandomization

At the cost of the evaluation of G_r and a deviation of at most $\epsilon(r)$ in the output distribution, the use of an $\epsilon(r)$-PRG for $\mathcal{A}$ allows us to effectively reduce the need for truly random bits from r down to $\ell(r)$, the seed length of the PRG. If the seed length is sufficiently small, we can then efficiently compute the output distribution of any algorithm $A \in \mathcal{A}$ on a given input x to within $\epsilon(r)$ in 1-norm: cycle over all seeds σ and keep track of the statistics of $A(x, G_r(\sigma))$.

In particular, this allows us to fully derandomize decision procedures with error less than $\frac{1}{2}-\epsilon(r)$. Recall that a decision problem L can be represented as the set $L \subseteq \Sigma^*$ of all "yes"-instances, or equivalently, as a mapping $L : \Sigma^* \mapsto \{0,1\}$, where 0 represents "no", and 1 represents "yes". A randomized decision procedure A for L has error at most η if for every $x \in \Sigma^*$

$$\Pr[A(x, U_r) \neq L(x)] \leq \eta.$$

If $\eta < \frac{1}{2} - \epsilon(r)$, we can deterministically decide L as follows using the $\epsilon(r)$-PRG G for A. On input x:

1. Cycle over all $\sigma \in \{0,1\}^{\ell(r)}$ and compute $A(x, G_r(\sigma))$.
2. Output "yes" if more than half of the σ's lead to acceptance, and "no" otherwise.

The correctness of the above procedure for deciding L follows because

$$\begin{aligned}
\Pr[A(x, G_r(U_{\ell(r)})) \neq L(x)] &\leq \Pr[A(x, U_r) \neq L(x)] \\
&\quad + \Big| \Pr[A(x, G_r(U_{\ell(r)})) \neq L(x)] - \Pr[A(x, U_r) \neq L(x)] \Big| \\
&< \left(\frac{1}{2} - \epsilon(r)\right) + \epsilon(r) = \frac{1}{2}.
\end{aligned}$$

What about the efficiency of the deterministic simulation? Suppose A runs in time t and uses at most s bits of work space, and that evaluating G takes time t_G and work space s_G. Then the above procedure runs in time $O(2^\ell \cdot (t_G + t))$ and space $O(\ell + s_G + s)$. Note that, due to the factor of 2^ℓ in the running time, there is not much point for the running time t_G of the PRG to be substantially less than 2^ℓ, at least not when the goal is full derandomization. In contrast, when the goal is to reduce the amount of randomness from r to $\ell(r)$, then we typically want the time complexity t_G of the PRG to be lower, e.g., polynomial in ℓ or in r. The latter situation occurs in cryptographic applications of PRGs, among others.

What seed length $\ell(r)$ can we hope for? Naïvely, one may think that $\ell = 0$ may be within reach. However, for the classes $\mathcal{A}$ that we will consider, this it too much to hope for. The reason is that the *same* pseudorandom bit sequences $G_r(\sigma)$ have to work well irrespective of which input x and which algorithm A from our class $\mathcal{A}$ we are considering. This obliviousness usually imposes a lower bound of $\Omega(\log r)$ on the seed length $\ell(r)$. Thus, the best we can hope for are PRGs with seed length $\ell(r) = \Theta(\log r)$. Such PRGs yield the following deterministic simulations for the class $\mathcal{A}_d$ of all randomized decision procedures with error less than, say, 1/3 that run in time n^d, and the class $\mathcal{A}'_d$ of all randomized decision procedures with error less than 1/3 that use $d \cdot \log n$ bits of work space and always halt. Recall that we use n to denote the length of the input x.

Proposition 1.

(i) (Time-bounded setting) If there exists a $\frac{1}{6}$-PRG for $\mathcal{A}_d$ that has seed length $\ell(r) = O(\log r)$ and is computable in time $2^{O(\ell)}$, then $\mathcal{A}_d$ can be simulated deterministically in time $n^{O(1)}$.

(ii) (Space-bounded setting) If there exists a $\frac{1}{6}$-PRG for $\mathcal{A}'_d$ that has seed length $\ell(r) = O(\log r)$ and is computable using $O(\ell)$ bits of work space, then $\mathcal{A}'_d$ can be simulated deterministically using $O(\log n)$ bits of work space.

This proposition follows from the above analysis because algorithms in $\mathcal{A}_d$ and in $\mathcal{A}'_d$ need no more than $r \leq n^d$ random bits in total. This is because the algorithm needs at most one random bit per computation step and because the algorithm runs for at most n^d steps in either of the two settings.

1.3 Construction

In the rest of this course, we will construct PRGs for various interesting classes $\mathcal{A}$ of algorithms. The name of the game is to exhibit some limitation in the way algorithms from $\mathcal{A}$ use their random bits, and then exploit that limitation to construct a pseudorandom generator for $\mathcal{A}$ with short seed length. For some more restrictive classes $\mathcal{A}$, the limitation is not too difficult to point out; for broader classes like $\mathcal{A}_d$ above, the limitation is not that clear, and we will resort to the mantra from the previous lecture:

Whether something looks random to you, depends on your computational power.

The mantra has two sides to it. Let us first make one side more concrete – that the mere existence of a nontrivial PRG G for a class $\mathcal{A}$ implies some computational limitation on the class $\mathcal{A}$.

Proposition 2. *If G is an $\epsilon(r)$-PRG for $\mathcal{A}$ with $\epsilon(r) < 1/2$ and seed length $\ell(r) < r$, then there does not exist $A \in \mathcal{A}$ such that $A(x, \rho)$ accepts if and only if there exists ρ' in the range of G_r such that ρ and ρ' have the same prefix of length $\ell(r) + 1$.*

Proof. Consider the predicate $A(x, \rho)$ defined in the statement. By construction we have that

$$\Pr[A(x, G_r(U_{\ell(r)})) \text{ accepts}] = 1\,, \tag{1}$$

whereas the fact that the range of G_r has size at most 2^ℓ implies that

$$\Pr[A(x, U_r) \text{ accepts}] \leq \frac{2^\ell}{2^{\ell+1}} = \frac{1}{2}\,, \tag{2}$$

so the probabilities on the left-hand sides of (1) and (2) differ by at least 1/2. On the other hand, the pseudorandomness property of G would require those probabilities to differ by no more than $\epsilon(r) < 1/2$ if A were in $\mathcal{A}$. $\square$

We refer to Proposition 2 as yielding a *hard problem* for $\mathcal{A}$. The hard problem given in Proposition 2 may seem somewhat unnatural in that it ignores its actual input x, but it does point at a limitation in the way algorithms from $\mathcal{A}$ can use their random bits when a nontrivial PRG for $\mathcal{A}$ exists. This is one side of the mantra.

The other side of the mantra – the one that is more interesting to us – is how to use such a limitation to construct a good PRG for $\mathcal{A}$, that is, how to use a hard problem to build a PRG. This is known as the *hardness-based approach to the construction of PRGs.* It represents an important avenue towards deterministic simulations of very broad classes of algorithms like $\mathcal{A}_d$ and $\mathcal{A}'_d$ in Proposition 1, and a substantial part of this course will be devoted to developing that approach.

Time-bounded setting. In order to obtain PRGs for the classes $\mathcal{A}_d$, we usually construct PRGs for their nonuniform counterparts. In fact, it suffices to construct a PRG G for the class $\tilde{\mathcal{A}}$ of predicates $A(\cdot, \rho)$ that ignore their first input and can be decided by circuits of size at most $r \doteq |\rho|$. This is because for every $A' \in \mathcal{A}_d$ and every input $x' \in \Sigma^n$ on which A' needs, say, r' random bits, there exists a predicate $A \in \tilde{\mathcal{A}}$ such that $A(\cdot, \rho) = A'(x', \rho')$, where $|\rho| \doteq r = n^{O(d)}$ and ρ' denotes the prefix of length r' of ρ. Thus, taking the prefixes of length r' of the output of G_r yields a PRG for A.

Proposition 2 yields the following for $\tilde{\mathcal{A}}$. Let G be an ϵ-PRG for $\tilde{\mathcal{A}}$ with $\epsilon < 1/2$ and $\ell(r) < r$, and consider the problem of deciding whether a given string of length $\ell(r) + 1$ is the prefix of some string in the range of G_r. Proposition 2 implies that this problem cannot be decided by a Boolean circuit of size r. Note that, if the PRG is computable in time $2^{O(\ell)}$, then the problem can be decided by a uniform algorithm in time $2^{O(\ell)}$. Thus, for $\ell(r) = O(\log r)$ we obtain the following implication:

$$\begin{array}{c}\text{the existence of a PRG for } \tilde{\mathcal{A}} \text{ that is computable in time} \\ 2^{O(\ell)} \text{ and has seed length } \ell(r) = O(\log r)\end{array} \tag{3}$$

$$\Downarrow$$

$$\begin{array}{c}\text{the existence of a decision problem that is computable in time } 2^{O(\ell)} \\ \text{and requires circuits of size } 2^{\Omega(\ell)} \text{ on inputs of length } \ell.\end{array} \tag{4}$$

As for the reverse implication $\Uparrow$, we will see how to use a decision problem of type (4) to build a PRG of type (3) [IW97], which by the above discussion and Proposition 1 implies that $\mathcal{A}_d$ can be simulated deterministically in polynomial time. Thus, polynomial-time derandomizations of $\mathcal{A}_d$ through PRGs for $\tilde{\mathcal{A}}$ are equivalent to the hardness condition (4).

The latter seems plausible, which is why we conjecture that every randomized decision procedure can be simulated deterministically with only a polynomial overhead in running time. However, in spite of half a century of research in circuit complexity, the validity of (4) remains open. In fact, the same holds for weaker circuit lower bounds that are equivalent to PRGs for $\tilde{\mathcal{A}}$ computable in time $2^{O(\ell)}$ and with seed length between $\Theta(\log r)$ and $r^{\Theta(1)}$.

Space-bounded setting. In the model where algorithms have *two-way* access to a stream of random bits, a similar equivalence holds between plausible-but-open hardness conditions and space-efficient derandomization via PRGs. The equivalence supports the common belief that every randomized decision procedure can be simulated deterministically with only a constant-factor overhead in work space.

Moreover – in contrast to the time-bounded setting – there are unconditional results in the model with *one-way* access to the random bit stream. Results in that more restricted model are most relevant because coin flips cannot be reread unless they are stored in memory, which is limited in size. In particular, we will develop a PRG for $\mathcal{A}'_d$ that has seed length $O(\log^2 r)$ and is computable using $O(\log r)$ work space [Nis92], thus yielding a full derandomization of $\mathcal{A}'_d$ using space $O(\log^2 n)$ via PRGs. Currently no PRG for $\mathcal{A}'_d$ is known that achieves a shorter seed length. Nevertheless, there exists a full derandomization of $\mathcal{A}'_d$ using work space $O(\log^{3/2} n)$ [SZ99]. In contrast to the derandomizations obtained via PRGs as in Proposition 1, the latter deterministic simulation needs access to the input and to the code of the underlying randomized algorithm. This discrepancy points at the possibility that PRGs, though the canonical tool for derandomization, may not be omnipotent.

2 Extractors

Just like PRGs are the canonical tool for derandomization, extractors are the canonical tool for randomness extraction. Intuitively, an extractor is a procedure that takes a sample from a distribution that contains randomness in a crude form and massages the sample such that the resulting output distribution is close to a "pure" random source, that is, close to uniform.

The extractor can have some limited knowledge about the distribution, that is, the extractor is only supposed to work for a certain class $\mathcal{D}$ of distributions $D = (D_n)_{n \in \mathbb{N}}$ (where D_n is a distribution on $\{0,1\}^n$). An additional parameter is a bound ϵ on the allowed deviation of the output distribution from uniform.

Definition 2 (Extractor). *An $\epsilon(n)$-extractor for $\mathcal{D}$ is a sequence of mappings $E = (E_n)_{n\in\mathbb{N}}$ where $E_n : \{0,1\}^n \mapsto \{0,1\}^{r(n)}$ such that $(\forall D \in \mathcal{D})(\forall^\infty n \in \mathbb{N})$*

$$d_{\text{stat}}\Big(E_n(D_n), U_{r(n)}\Big) \leq \epsilon(n)\,.$$

We would like $r(n)$ to be as large a function of n as possible. It can never be larger than n, but what values we can realize depends on the amount of crude randomness that is present in the distributions D from $\mathcal{D}$. For example, if the support of D is a singleton, then we cannot hope to extract even a single bit of true randomness because D corresponds to some deterministic process.

We will discuss how to measure the amount of crude randomness in a distribution D and various natural choices for $\mathcal{D}$ later. For now, let us contrast the notion of an extractor with that of a PRG. The notions seem quite different: whereas PRGs stretch their inputs, extractors shrink them. Nevertheless, there exists a close connection between extractors and hardness-based constructions of pseudorandom generators [Tre01].

3 Finite Fields

We now present some basic properties of finite fields that will be needed in the course and may not be familiar to all readers. We first show the existence of finite fields $\mathbb{F}_q$ with $q = p^k$ elements, where p is a prime and k a positive integer. We then consider the efficiency of arithmetic within finite fields, and conclude with a few remarks regarding uses of finite fields.

3.1 Preliminaries

We first recap the terminology we use. A *group* is a structure consisting of a universe G and a binary operation $+$ that is associative, has a neutral element (a.k.a. a unit), and such that every element has an inverse. If $+$ is commutative, the group is called commutative. A *ring* is a structure consisting of a universe R and two binary operations $+$ and $\cdot$ where: $(R,+)$ forms a commutative group, $\cdot$ is associative on R, and $\cdot$ distributes over $+$. If $\cdot$ is commutative, the ring is called commutative. A *field* is a commutative ring with a multiplicative unit such that each element other than the additive unit 0 has an inverse for $\cdot$. We often refer to $(F,+)$ as the additive group of the field and to $(F\setminus\{0\},\cdot)$ as the multiplicative group of the field.

The following proposition gives a useful sufficient condition for a finite ring to be a field.

Proposition 3. *Consider a finite commutative ring R with a multiplicative unit that is different from 0. Then R is a field if and only if for all $a,b \in R$, $ab = 0$ implies that $a = 0$ or $b = 0$.*

Proof. Let the multiplicative unit of R be denoted by 1. We must only show that each element a of R has a multiplicative inverse. We show this by showing that the mapping $x \mapsto a\cdot x$ is a bijection. If this is a bijection, then there is some element a' such that $a \cdot a' = 1$. This a' is the multiplicative inverse of a.

Now suppose for the purpose of contradiction that $x \to a \cdot x$ is not a bijection. Then we have $a\cdot x_1 = a\cdot x_2$ for distinct $x_1, x_2 \in R$. Rearranging terms, we have $a\cdot(x_1 - x_2) = 0$ which contradicts the hypothesis. □

Given a field F, we consider polynomials over a single variable with coefficients from F.

Exercise 1. *The set of polynomials $F[x]$ over F forms a commutative ring with a multiplicative unit.*

We will be interested in polynomials over F that do not factor over F, as defined presently.

Definition 3. *Let F be a field. A polynomial $g(x)$ with coefficients from F is called* irreducible over F *if there are no two polynomials $g_1(x)$ and $g_2(x)$ with coefficients over F and of degree less than $g(x)$ such that $g(x) = g_1(x) \cdot g_2(x)$.*

Example 4. *Consider the polynomial $g(x) = x^3 + x + 1$ over $\mathbb{Z}_2$. We claim that $g(x)$ is irreducible over $\mathbb{Z}_2$. In principle, we must verify that each pair of polynomials of degree less than three over $\mathbb{Z}_2$ multiplies to yield something other than $g(x)$. The set of polynomials that must be checked is: $\{1, x, 1+x, x^2, 1+x^2, x+x^2, 1+x+x^2\}$. In this case we can also argue as follows. Since $g(x)$ has degree 3, at least one of the factors of any factorization as $g(x) = g_1(x)g_2(x)$ with $g_1(x)$ and $g_2(x)$ of degree less than three, has to have degree exactly one. This implies that $g(x)$ would have a zero over $\mathbb{Z}_2$. However, $g(0) = 1 = g(1)$.*

An irreducible polynomial plays the role in the polynomial ring that prime numbers play in the integers. The following is a property of irreducible polynomials that also holds for prime numbers in the integers.

Proposition 5. *Let $g(x)$, $g_1(x)$, and $g_2(x)$ be polynomials over a field F. If $g(x)$ is irreducible over F, then $g(x)$ divides $g_1(x) \cdot g_2(x)$ if and only if $g(x)$ divides $g_1(x)$ or $g(x)$ divides $g_2(x)$.*

Proof (sketch). Just as an integer can be factored uniquely into its prime factors, a polynomial over a field can be factored uniquely into irreducible polynomials. If $g(x)$ divides $g_1(x) \cdot g_2(x)$, then $g(x) \cdot h(x) = g_1(x) \cdot g_2(x)$ for some polynomial $h(x)$. If we view this equation in terms of the unique factorization of each polynomial into irreducible polynomials, it becomes evident that $g(x)$ must divide either $g_1(x)$ or $g_2(x)$. $\square$

The final building block we need is that of modular arithmetic. We assume the reader is familiar with $\mathbb{Z}_n$, the integers modulo n. We can also use modular arithmetic over the ring of polynomials over a field F.

Definition 4. *Let $F[x]$ be the ring of polynomials over a field F, and let $g(x)$ be a polynomial with coefficients from F. Then $F[x]/g(x)$ is the ring of polynomials over F modulo $g(x)$. Formally, $F[x]/g(x)$ contains an equivalence class for each polynomial that can result as a remainder upon dividing by $g(x)$, and arithmetic among the equivalence classes is performed modulo $g(x)$.*

Exercise 2. *If F is a field and $g(x)$ is a polynomial with coefficients from F, then $F[x]/g(x)$ is a finite commutative ring with a multiplicative unit. Further, if $g(x)$ has degree d, the elements of $F[x]/g(x)$ are in one-to-one and onto correspondence with the polynomial of degree less than d over F.*

3.2 Existence

We have now set up the appropriate background to prove the existence of finite fields. We first mention the finite fields that we are most familiar with.

Theorem 6. *For all $n \geq 2$, $\mathbb{Z}_n$ is a commutative ring with a multiplicative unit. $\mathbb{Z}_n$ is a field if and only if n is prime.*

The second part of Theorem 6 follows from Proposition 3 and demonstrates finite fields that are suitable for many of our purposes. However, there are other finite fields we will need to make use of. The following theorem is the main purpose of this section, demonstrating a finite field for all prime powers.

Theorem 7. *For prime p, and $k \geq 1$ there is a field with p^k elements.*

Proof. Let $\mathbb{Z}_p[x]$ be the ring of polynomials with coefficients from $\mathbb{Z}_p$, and $\mathbb{Z}_p[x]/g(x)$ be the quotient ring of polynomials modulo the polynomial $g(x)$. By Exercise 2, $\mathbb{Z}_p[x]/g(x)$ is a finite commutative ring with a multiplicative unit. The theorem follows from the following two lemmas.

Lemma 8. *Let $k \geq 1$ be an integer and p a prime. There exists an irreducible polynomial of degree k over $\mathbb{Z}_p$.*

Proof (sketch). This can be proved by a careful counting argument showing that the number of irreducible polynomials is positive. We do not present further details here. □

Lemma 9. *$\mathbb{Z}_p[x]/g(x)$ is a field if and only if $g(x)$ is irreducible over $\mathbb{Z}_p$.*

Proof. The elements of $\mathbb{Z}_p[x]/g(x)$ are in one-to-one and onto correspondence with the polynomials over $\mathbb{Z}_p$ of degree less than the degree of $g(x)$. A product of two elements is zero if and only if the product of the corresponding polynomials is a multiple of $g(x)$.

If $g(x)$ is not irreducible, then $g(x) = g_1(x) \cdot g_2(x)$ for some polynomials $g_1(x)$ and $g_2(x)$ of degree less than the degree of $g(x)$, meaning that for non-zero ring elements $g_1(x)$ and $g_2(x)$ their product is zero. Then $\mathbb{Z}_p[x]/g(x)$ is not a field by Proposition 3.

Let $g(x)$ be irreducible. Suppose there are $g_1(x)$ and $g_2(x)$ whose product is a multiple of $g(x)$ (that is, whose product is zero in the ring). Then Proposition 5 tells us that $g(x)$ must divide at least one of $g_1(x)$ or $g_2(x)$, meaning at least one of $g_1(x)$ or $g_2(x)$ is zero in the ring. By Proposition 3, $\mathbb{Z}_p[x]/g(x)$ is a field. □

In fact, the construction given in Theorem 7 is enough to generate all possible finite fields, stated formally in the following theorem whose proof we omit.

Theorem 10. *Let F be a finite field. Then F has p^k elements for some prime p and integer $k \geq 1$. Further, each finite field with p^k elements is isomorphic.*

We use $\mathbb{F}_q$ to denote a generic finite field with $q = p^k$ elements.

Example 11. *Let us construct $\mathbb{F}_{2^3}$ using the irreducible polynomial of degree 3 from the example in the first section. Therefore, $\mathbb{F}_{2^3}$ can be constructed as $\mathbb{Z}_2[x]/(x^3+x+1)$. Each element of the field is viewed as a degree at most two polynomial, and can thus be specified with three bits. As an example of multiplication in the field, $(x^2+1)\cdot(x+1) = (x^3+x+x^2+1) = (x^3+x+1)+x^2 = 0+x^2 = x^2$. As an example of addition in the field, $(x^2+1)+(x+1) = (x^2+x+1+1) = x^2+x$.*

3.3 Complexity

Theorem 7 only shows that finite fields of order p^k exist. For a finite field to be of practical use, it should be efficiently constructible, and arithmetic in the field should be efficient. We leave it as an exercise to verify that arithmetic can be performed in $\mathbb{F}_{p^k}$ in polynomial time once an irreducible polynomial of degree k over $\mathbb{Z}_p$ is found.

To efficiently construct $\mathbb{F}_{p^k}$, all that needs to be done is to find an irreducible polynomial of degree k over $\mathbb{Z}_p$. We would like to be able to find such a polynomial in polynomial time, where the input length is the number of bits needed to specify a degree k polynomial with coefficients in $\mathbb{Z}_p$, that is, $O(k \cdot \log p)$. It is unknown whether there is an algorithm running in time $\mathrm{poly}(k, \log p)$ to do this. The following is essentially the best known algorithm.

Theorem 12 ([Sho90]). *There is a deterministic algorithm running in time* $\mathrm{poly}(k, p)$ *to find an irreducible polynomial of degree k over $\mathbb{Z}_p$.*

Notice that for small values of p this is in fact a polynomial time algorithm. In particular, this shows that we can in polynomial time construct a suitable irreducible polynomial to construct $\mathbb{F}_{2^n}$. For most purposes, this is sufficient. If our requirements are even more lenient, we can do even better. The following gives an explicit formula for irreducible polynomials for certain values of n.

Theorem 13 ([vL98], Thm 1.1.28). *Let $n = 2 \cdot 3^{\ell-1}$. Then $x^n + x^{n/2} + 1$ is irreducible over $\mathbb{Z}_2$.*

3.4 Remarks

One common use of finite fields is to view data as elements of the finite field and take advantage of the nice properties of polynomials over fields, e.g., that a degree d polynomial can have at most d roots.

Finally, we remark on the distinction between formal polynomials and polynomials as functions. A formal polynomial refers to the particular coefficients that specify it. Two formal polynomials over a finite field F induce the same function if and only if they are equal modulo $\prod_{a \in F}(x-a)$. In particular, all polynomials of degree less than q over $\mathbb{F}_q$ induce different functions.

References

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: derandomizing the XOR lemma. In *Proceedings of the 29th Annual Symposium on Theory of Computing, STOC 1997*, pages 220–229, 1997.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[Sho90] V. Shoup. New algorithms for finding irreducible polynomials over finite fields. *Mathematics of Computation*, 54:435–447, 1990.

[SZ99] Michael Saks and Shiyu Zhou. $\mathrm{BP_HSPACE(S)} \subseteq \mathrm{DSPACE(S^{3/2})}$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

[Tre01] Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001.

[vL98] J.J. van Lint. *Introduction to Coding Theory.* Springer-Verlag New York Inc., third edition, 1998.