MIT 18.434: Seminar in Theoretical Computer Science Spring 2016

Fields and Polynomials

Rishad Rahman

1 Fields and the significance of $\mathbb{F}_p$

Definition 1.1. *A field $\mathbb{F}$ is a set where the following hold on its elements under two compatible operators $(+, \times)$.*

- *Closure under $+$ and $\times$*
- *Associativity under $+$ and $\times$*
- *Commutativity under $+$ and $\times$*
- *Existence of identities 0 under $+$ and 1 under $\times$*
  - *Remark: You can show $1 \neq 0$ for non-trivial fields*
- *Existence of additive inverses for all elements and multiplicative inverses for all elements except 0*
- *Distributivity of multiplication over addition*

Proposition 1.1. *If p is a prime, then $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ under addition $\bmod p$ and multiplication $\bmod p$ is a field, denoted by $\mathbb{F}_p$.*

Proof. Most of the conditions are easy to verify. Existence of an identity under multiplication $\bmod p$ is not so obvious though so we'll verify that here. Consider q such that $q \neq 0 \bmod p$ i.e. $\gcd(p, q) = 1$. This leads us to consider the following lemma:

Lemma 1.1. $\gcd(p, q) = 1 \iff \exists a, b \in \mathbb{Z}$ *s.t.* $ap + bq = 1$

Proof. The backwards direction is trivial, for the forward direction we relay Euclidean Algorithm. The algorithm is based on the fact that if $p > q$ then $\gcd(p, q) = \gcd(q, p \mod q)$. If $\gcd(p, q) = 1$, the algorithm converges to 1 once we reach r such that $\gcd(r, 1) = 1$. Note that the running time of the algorithm is $O(\log p)$ since the total size of the inputs halves at every step. It is easy to verify that if p', q' are of the form $ap + bq$, $a, b \in \mathbb{Z}$ then $r = p' \bmod q'$ is also of that form. Hence $1 = ap + bq$ for some a, b. □

Suppose $ap + bq = 1$, then $bq = 1 \bmod p \Rightarrow b = q^{-1} \bmod p$ □

Great, we have a field! But what's so special about $\mathbb{F}_p$? First of all $\mathbb{F}_p$ is much smaller than $\mathbb{Z}$ and smaller sets are usually easier to work with it. This is really useful when we have more complex spaces such as that of polynomials with integer coefficients which we'll analyze later. The existence

of multiplicative inverses, which we just showed, are nice since then most linear transforms on the set are automorphims so algebraic structure is preserved. In computer science, we usually look at a mapping from the "universe" of objects we are concerned with, to $\mathbb{F}_p$ which is mathematically easier to analyze. One thing to note is that this works well with randomization, which we may attribute, at least intuitively, to not having some sort of bias when choosing an integer between 0 and $p-1$.

2 Finding a prime

Before we can start working with $\mathbb{F}_p$, we must determine it by finding a relevant p. Even though an infinite number exists, (un)fortunately[1] finding a large prime is not an easy matter. However primality testing is in **P** via AKS, although the algorithm is very intricate so we won't discuss it here. A couple of basic ways to find an m-bit prime include

- Picking m-bit numbers at random and running primality testing until we discover a prime
- Running primality testing in sequential order on 2^m+1 to $2^{m+1}-1$ until we discover a prime

Note that one is deterministic while one is random, so use whichever is more fitting to the problem at hand. Note that the running time of both methods rely on the density of primes. Thankfully the prime number theorem states that the fraction of m- bit numbers which are prime is $\approx (2\ln 2 \cdot m)^{-1}$ so we expect to run $O(m)$ trials before finding a prime. Another theorem useful in theory is the following:

Theorem 2.1. *(Bertrand's Postulate)* $\forall n > 1\ \exists p$ *prime such that* $n < p < 2n$

An example of where this is used is in Universal Hashing. Consider the following set of hash functions:

$$\mathcal{H} = \{h_{a,b} = (ax+b \bmod p) \bmod n \mid a \in 1,2,\ldots,p-1, b \in 0,1,2,\ldots,p-1\}$$

where $h_a, b : U \mapsto \mathbb{Z}_n$ and $|U| \leq p \leq 2|U|$. We then have the following claim:

Claim 2.1. *For* $x \neq y$*, if* $h_{a,b}(x) = h_{a,b}(y)$*, then* $s \neq t$ *but* $s = t \bmod n$ *where* $s = ax+b \bmod p$ *and* $t = ay+b \bmod p$*.*

Proof. Suppose WLOG $x > y$. If $s = t$ then p divides $(ax+b)-(ay+b) = a(x-y)$ however $a < p$ and $x - y < |U| \leq p \Rightarrow\Leftarrow$. □

This is useful since then you have a bijection between (s,t) pairs and (a,b) pairs for fixed $x \neq y$. In particular $a = (s-t)(x-y)^{-1} \bmod p$ and $b = s - ax \bmod p$. When we fix s there are at msot $(p-1)/n$ values for t such that $s \neq t$ but $s = t \bmod n$. Therefore the total number of hash functions in this family which results in collision for fixed x, y is at most $p(p-1)/n$.

[1] Although it seems like we're using primes in order to make a problem easier, let's not be deterred from cases where the difficulty of problems involving primes is important e.g. RSA

3 Polynomials

We start with some basic review on polynomials. Note that we will be interested in polynomials where the coefficients are taken from a field $\mathbb{F}$ rather than the usual $\mathbb{R}$. We denote this set of polynomials by $\mathbb{F}[x_1, \ldots]$.

Definition 3.1. *Degree of a polynomial: max degree of any monomial/term denoted by* deg

Definition 3.2. *Degree of a variable x_i: max degree of x_i*

Observation 3.1. *Univariate division: Given $p(x)$ and $q(x)$, we can write $p(x) = q(x)Q(x) + r(x)$ where $\deg r < \deg p$.*

As a result we can extend the Euclidean Algorithm to polynomials. This leads to the following definition of prime extended to polynomials:

Definition 3.3. *A polynomial $p(x)$ with $deg(p) > 1$ is irreducible (can't be factored) if and only if $\gcd(p, q) = e$, where $e \in \mathbb{F}$, for any $q(x)$ where $deg(q) < deg(p)$. Note that $x^2 + 1$ is reducible in $\mathbb{F}_2$ while $x^2 + x + 1$ is not.*

4 Polynomial Fields

We can also generalize $\mathbb{F}_p$ to polynomials. If $p(x)$ is an irreducible polynomial then

$$\left\{ \mathbb{F}_p[x] \bmod p(x) \right\} \text{ is a field of size } |\mathbb{F}|^{\deg p(x)}$$

We can do all arithmetic in this field in time polynomial in the number of bits needed to describe the coefficients (each polynomial in the field corresponds to a vector in $\mathbb{F}_p^l$ where $l = \deg p(x)$). In general this field is isomorphic to $\mathbb{F}_{p^l}$. This construction is particularly useful because it allows us to construct a fields of size p^l which are particularly useful when l tends to be large.

We already showed x^2+x+1 is irreducible with in $\mathbb{F}_2$. Let us look at the powers of x mod x^2+x+1:

$$x^1 = x$$
$$x^2 = x + 1$$
$$x^3 = 1$$

Note that this generates the field minus 0, so we call x a generator or a primitive root. In general if $q = p^r$ then $\mathbb{F}_q^*$ has $\phi(q-1)$ primitive roots. Finding these primitive roots are useful because we can generate a lookup table according to the power and then to multiply two elements of the field, we can just add their indices modulo $q - 1$ and lookup the result. The indices are also called the discrete logs. Note that given X it is easy to compute $Y = g^X$ where g is a primitive root by repeated squaring. However given Y, computing X is computationally difficult when q is large, this is called the discrete log problem and has applications in cryptography.

Naturally we extend the question of finding a prime p to finding an irreducible polynomial $p(x)$. We can randomly pick polynomials and test for irreducibility. Similarly to the case for primes we must require the check to be efficient and also have irreducible polynomials to be somewhat dense. The former can be done in polynomial time and for the latter we have the following:

Observation 4.1. *If $p(x) = x^l + c_{l-1}x^{l-1} + \ldots + c_0$ where c_i is chosen randomly from $\mathbb{F}_p$ then*

$$\frac{1}{2l} < \mathbb{P}[p(x) \text{ irreducible}] < \frac{1}{l}$$

Observation 4.2. *An irreducible polynomial over $\mathbb{F}_p$ of degree n remains irreducible over $\mathbb{F}_{p^k}$ if and only if $\gcd(n, k) = 1$.*

We also have the following two theorems

Theorem 4.3. *There exists a deterministic, poly(p, l) time algorithm to find a degree l irreducible polynomial modulo p.*

Theorem 4.4. *$x^{2\cdot(3k)} + x^{3k} + 1$ is irreducible modulo 2 for all $k \geq 0$.*

A lot of nice results regarding polynomial fields rely on heavy duty number theory and algebra but it is nice to note these results since they allow for efficient arithmetic.

5 Roots of polynomials

We first note the following result which we'll call the "Degree Mantra" and covers the trivial number of roots bound opposite to that in the Fundamental Theorem of Algebra.

Theorem 5.1. *A non-zero, univariate polynomial with degree d has at most d roots.*

Proof. Factor the polynomial into a reducible and irreducible parts. The reducible part has at max $d' \leq d$ roots while the irreducible part has no roots (via Observation 3.1). □

This is really useful when we have a finite field since there are only a finite number of elements available to possibly be roots.

6 Schwartz-Zippel Lemma

The Degree Mantra in its most general form is stated below

Theorem 6.1. *(Schwartz-Zippel Lemma) Let $p \in \mathbb{F}_q[x_1, \ldots, x_n]$ be a reduced polynomial (that is not 0) with total degree $\leq d$. Then*

$$\mathbb{P}_{\alpha_1,\ldots,\alpha_n}[p(\alpha_1, \ldots, \alpha_n) \neq 0] \geq \frac{1}{q^{\lfloor \frac{d}{q-1} \rfloor}} \cdot \left(1 - \frac{d \bmod (q-1)}{q}\right)$$

Although it looks monstrous, it can be proved via induction but we will not go through with such troublesome affairs. Instead we look at the common cases of the lemma.

Corollary 6.1. *If $q = 2$, $\mathbb{P}[p \neq 0] \geq 2^{-d}$*

Corollary 6.2. *If $d < q - 1$, $\mathbb{P}[p = 0] \leq d/q$*

We see that the probability bound depends heavily on the relation between d and q. However with a good bound, the Schwartz-Zippel Lemma is extremely useful in saving time since multiplying out multivariate polynomials can get very difficult.

7 Communication complexity

Suppose Alice and Bob are the proud owners of bit strings a and b respectively where $a, b \in \{0,1\}^n$. They only know their respective bit strings at the start but a channel is open between them via which they can send messages. We eventually want to perform a computation involving both a and b. We can bound the size of messages sent per computation by $n+1$ bits since Bob can send all of his bits to Alice who can perform the computation and then send the result to Bob. Let us analyze the following problem: does $a = b$?

We note that deterministic algorithms require at least n bits of communication so we will try a randomized approach. Specifically we want to output a correct answer with probability $1 - 1/n$ using $O(\log n)$ bits of communication. Note that an increase in 1 bit in the communication complexity has a strong influence on the algorithm's accuracy.

Randomized algorithm to decide if $a = b$:

1. Alice chooses $q \in [n^2, 2n^2]$ fixing $\mathbb{F}_q$. She tells Bob, which requires $O(\log q) = O(\log n)$ bits.
2. Alice formulates the polynomial $p_A(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x$
3. Bob formulates his own polynomial $p_B(x) = b_n x^n + b_{n-1}x^{n-1} + \ldots + b_1 x$
4. Alice chooses at random $\alpha \in \mathbb{F}_q$, computes $p_A(\alpha)$ and sends both results to Bob requiring $O(\log n)$ bits.
5. Bob says $a = b$ iff $p_A(\alpha) = p_B(\alpha)$.

Proof. If $a = b$ then we have the algorithm produces a correct result with probability 1. If $a \neq b$ and $q(x) = p_A(x) - p_B(x)$ then

$$\mathbb{P}_{\alpha \in \mathbb{F}_q}[p_A(\alpha) = p_B(\alpha)] = \mathbb{P}_\alpha[q(\alpha) = 0] \leq \frac{n}{n^2} = \frac{1}{n}$$

□

Remark: Repeating the procedure k times gives us a bound of $1 - 1/n^k$.

8 Perfect matching in bipartite graphs

We now look at applications of the Schwartz-Zippel Lemma.

Definition 8.1. *A perfect matching of a bipartite graph G is a subset S of its edges such that every vertex of G appears in S exactly once.*

The problem of finding a perfect matching can be solved in $O(VE)$ time via max-flow algorithms. We will formulate a much simpler algorithm by allowing for randomization.

If the two sets of nodes of a bipartite graph G are S and T. Note we can assume $|S| = |T| = V/2$

otherwise there can't be a perfect matching. Define the $S \times T$ Karp Matrix M as having entry $e_{ij} = X_{ij}$ if $(i,j) \in E$ and 0 otherwise. Note that the X_{ij} is a variable, not a value.

An interesting quantity to look at is the determinant of M. Recall

$$\det(M) = \sum_{\pi \sim S_n} \text{sign}(\pi) \cdot \prod_{i=1}^{n} e_{i,\pi(i)}$$

Each term of the summation is the product of one element from each row i of M, determined by π, scaled by $\text{sign}(\pi)$. We have the following relation

$$\det(M) \begin{cases} \equiv 0 & \text{if there is no perfect matching} \\ \not\equiv 0 & \text{if there is a perfect matching} \end{cases}$$

The permutation π in the determinant formula, corresponds to selecting edges $e_{i,\pi(i)}$ for all i and since $\pi(i) \neq \pi(j)$ when $i \neq j$, a perfect matching results in the existence of a permutation such that $\prod e_{i,\pi(i)}$ is not 0 hence the determinant will not be exactly 0, but it can evaluate to 0 for certain X_{ij} so we will use Schwartz-Zippel. Choose random $X_{ij} \in \{1, \ldots, n^2\}$ and evaluate the determinant (which takes time $O(V^3)$).

$$\nexists \text{ a perfect matching} \implies \mathbb{P}[\det(M) = 0] = 1$$
$$\exists \text{ a perfect matching} \implies \mathbb{P}[\det(M) = 0] \leq \frac{n}{n^2} = \frac{1}{n}$$

We note that we can extend the idea of finding a perfect matching to general graphs by defining the $V \times V$ Tutte matrix by $e_{i,j} = X_{ij}$ if $(i,j) \in E$ for $i > j$, $e_{i,j} = -X_{ij}$ if $(i,j) \in E$ for $i < j$, and 0 otherwise.

9 Polynomials as functions

Since the properties of polynomial fields are very nice, we are usually interested in simulating an arbitrary function via a polynomial within a specified field.

Observation 9.1. *Every $p \in \mathbb{F}[x_1, \ldots, x_n]$ computes some function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ and vice versa*

We can find the polynomial which matches the function via interpolation. Note that q^{q^n} such functions exist while there are infinitely many polynomials. If p and q are polynomials which map to the same function, $p \neq q$, then $p - q$ is a non-zero polynomial that computes the 0 function.

Definition 9.1. *A reduced monomial is such that the exponent of each variable is $\leq q - 1$. A reduced polynomial is a polynomial such that every term is reduced.*

We note that there exists q^n reduced monomials and hence q^{q^n} reduced polynomials so we have the following corollary.

Corollary 9.1. *Every function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ is computed by a unique reduced polynomial.*

10 Arithmetization of Boolean Formulas

Boolean formulas appear in many algorithms and protocols involving randomization so it is of particular interest to us to analyze them. However to do that with a raw formula can be difficult so we transform it into an element of a polynomial field via arithmetization. That is to say formulate a $p(x_1, \ldots, x_m)$ is associated with $\phi(x_1, \ldots, x_m)$ where we associate true with the value 1 and false with the 0, i.e. satisfying assignments correspond to $p = 1$ and unsatisfying assignments correspond to $p = 0$.

Let's look at some simple mappings into $\mathbb{F}_2[x, y]$.

$$\begin{aligned} x &\longrightarrow x \\ \overline{x} &\longrightarrow x \\ x \wedge y &\longrightarrow xy \\ x \vee y &\longrightarrow 1 - (1-x)(1-y) \end{aligned}$$

A simple way to see verify last one is to write $x \vee y$ as $\neg(\overline{x} \wedge \overline{y})$. If we replace x and y with subformulas α, β we can construct polynomials for higher order formulas. Although $\mathbb{F}_2$ seems to be the natural field to work with regarding boolean formulas, the above actually works for any field $\mathbb{F}_q$ and the choice for q will be important if we wish to apply Schwarz-Zippel on these polynomials. Since $\deg(pq) \leq \deg p + \deg(q)$ we have that the degree of the polynomial associated with ϕ is at most n, the length of ϕ.

Define the $\#SAT$ problem as given an input boolean formula ϕ and a number k, to verify if ϕ has exactly k satisfying assignments. Note that since there are 2^m possible possible assignments, given an $f(n)$ black-box to $\#SAT$ we can find the number of satisfying assignments of a boolean formula in $O(mf(n))$ time via binary search. Suppose $f_i(a_1, \ldots, a_i)$ is a function which counts the number of satisfying assignments of ϕ given $x_j = a_j$ for $j \leq i$. Then we have the following observations

Observation 10.1.

- $f_0()$ *counts the number of satisfying assignments of* ϕ
- $f_m(a_1, \ldots, a_m)$ *is* 1 *if the* a_i*'s satisfy* ϕ *and* 0 *otherwise*
- $f_i(a_1, \ldots, a_i) = f_{i+1}(a_1, \ldots, a_i, 0) + f_{i+1}(a_1, \ldots, a_i, 1)$
- *If* $p(x_1, \ldots, x_m)$ *is the arithmetization of* ϕ *then*

$$f_i(a_1, \ldots, a_i) = \sum_{a_{i+1}, \ldots, a_m \in \{0,1\}} p(a_1, \ldots, a_m)$$

We now present a problem similar to that of the communication problem presented earlier. Suppose we have a prover P and a verifier V which are allowed to send messages to each other. The prover is trying to convince the verifier ϕ indeed has k satisfying assignments but the verifier wants to make a correct verification with high probability. In our protocol we will have P send polynomials related to f_i to V who checks if they are correct. If so V will send back a random number to extend

the evaluations to f_{i+1}. This continues until m numbers are sent and V accepts if and only if he doesn't make an incorrect evaluation. The protocol is written below where $X \to Y : m$ means X sends message m to Y. Assume the polynomial's coefficients belong to $\mathbb{F}_q$ where $q > 2^n$ and all r_i's are chosen randomly from $\mathbb{F}_q$.

Step 0. $P \to V : f_0()$
V rejects if $k \neq f_0()$

Step 1. $P \to V : f_1(z)$
V rejects if $f_0() \neq f_1(0) + f_1(1)$
$V \to P : r_1$
$\vdots$

Step i. $P \to V : f_i(r_1, \ldots, r_{i-1}, z)$
V rejects if $f_{i-1}(r_1, \ldots, r_{i-1}) \neq f_i(r_1, \ldots, r_{i-1}, 0) + f_i(r_1, \ldots, r_{i-1}, 1)$
$V \to P : r_i$
$\vdots$

Step $m+1$. V rejects if $f_m(r_1, \ldots, r_m) \neq p_m(r_1, \ldots, r_m)$, otherwise accepts

We see that if ϕ indeed has k satisfying assignments, then the P can make V accept as long as he reports the right polynomials. If ϕ does not have k satisfying assignments, then P is forced to lie to make V accept. For example at Step 0, since the actual value $f_0() \neq k$, P will report $\widetilde{f_0} = k$. Let the notion of incorrectness mean that the reported $\widetilde{f_i}$ does not match the true value f_i. Note that if $\widetilde{f_{i-1}}(r_1, \cdots, r_{i-1})$ is incorrect then one of $\widetilde{f_i}(r_1, \cdots, r_{i-1}, \{0,1\})$ must be incorrect hence the polynomial $\widetilde{f_i}(r_1, \ldots, r_{i-1}, z)$ must be incorrect. As a result P is always sending incorrect polynomials unless by chance $\widetilde{f_i}(r_1, \ldots, r_i) = f_i(r_1, \ldots, r_i)$. If this happens, then P can just start sending correct polynomials and will make V accept at the end. If this never happens then the error propagates and since V compares $f_m(r_1, \ldots, r_m)$ to $\widetilde{f_m}(r_1, \ldots, r_m)$ directly at the end, V will reject. The probability that P gets "lucky" at Step i by Schwarz-Zippel is $\leq \frac{n}{2^n} \leq \frac{1}{n^2}$ for $n \geq 10$. Hence the probability this happens at any step is $\leq m/n^2 < 1/n$ by the Union Bound which completes the analysis.

The class of problems that can be solved using this system in polynomial time is called IP and so we just showed $\#SAT \in IP$. This can be extended to show that $IP = PSPACE$ or the class of problems that can be solved in polynomial space.