1 Groups, Fields and Polynomials

1.1 Groups

Definition 1.1 (Group)**.**
A group G is a tuple $(A, \circ)$ satisfying 3 conditions (listed below), where

- A is a non-empty set and denotes the elements of the group;
- $\circ$ is a function of the form $\circ : A \times A \to A$, and is called the *group operation.*

For $a, b \in A$, we use the shorthand $a \circ b$ or ab to denote $\circ(a, b)$. Given $G = (A, \circ)$, we often identify A with G. For example, we write $a \in G$ to denote $a \in A$. The 3 conditions that a group must satisfy are as follows.

(i) (*Associativity*) For all $a, b, c \in G$, $a \circ (b \circ c) = (a \circ b) \circ c$.

(ii) (*Identity*) There is an element $e \in G$ such that for all $a \in G$, we have $e \circ a = a \circ e = a$. This element e is called the *identity* of the group. Often the identity is denoted by 1.

(iii) (*Inverse*) For every $a \in G$, there is an element $b \in G$ such that $a \circ b = b \circ a = e$. This element b is called the *inverse* of a and is often denoted by a^{-1}.

■

Remark. The following are groups.

- $\mathbb{Z}$ with the addition operation $+$.
- Even integers with the addition operation $+$.
- $\mathbb{Q}\backslash\{0\}$ with the multiplication operation $\cdot$.
- $\mathbb{R}^+$ with the multiplication operation $\cdot$.
- $\mathbb{Z}_N$ with the addition operation $+_N$, where $\mathbb{Z}_N = \{0, 1, 2, \ldots, N-1\}$ and $a +_N b \overset{\text{def}}{=} (a + b) \mod N$.
- $\mathbb{Z}_N^*$ with the multiplication operation $\cdot_N$, where $\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$ and $a \cdot_N b \overset{\text{def}}{=} ab \mod N$.

Remark. The following are not groups.

- $\mathbb{N}$ with the addition operation $+$.
- $\mathbb{Z}\backslash\{0\}$ with the multiplication operation $\cdot$.
- $\mathbb{Z}$ with the subtraction operation $-$.

- Odd integers with the addition operation $+$.
- $\mathbb{Z}_N$ with the multiplication operation $\cdot_N$.

Theorem 1.2 (Uniqueness of the identity)**.** *If G is a group, the identity element is unique.*

Proof. Suppose there are two identity elements e_1 and e_2. Then we can show that it must be the case that $e_1 = e_2$. Indeed, since e_1 is the identity, by definition, $e_1e_2 = e_2$. And since e_2 is the identity, by definition, $e_1e_2 = e_1$. Therefore both e_1 and e_2 are equal to e_1e_2, i.e., they are equal to each other. □

Theorem 1.3 (Uniqueness of the inverses)**.** *If G is a group and $a \in G$, then its inverse a^{-1} is unique.*

Proof. Suppose b and c are both inverses of a. We will show that they have to be the same element. Let e be the identity element (which we know is unique from the previous theorem). Then we know $ab = e$ and $ca = e$. Using these equalities and the associativity of the group operation, we have

$$c = ce = c(ab) = (ca)b = eb = b,$$

i.e., $c = b$. □

Theorem 1.4 (Cancellation rule)**.** *Suppose G is a group and $a, b, c \in G$. If $ab = ac$, then $b = c$. Similarly, if $ba = ca$, then $b = c$.*

Proof. If $ab = ac$, we can multiply both sides (from the left) by a^{-1}. The LHS of the equality becomes $a^{-1}(ab) = (a^{-1}a)b = eb = b$, and the RHS becomes $a^{-1}(ac) = (a^{-1}a)c = ec = c$. An analogous argument (multiplying from the right by a^{-1}) proves $ba = ca \implies b = c$. □

Exercise 1.5. Show that if G is a group and $a \in G$, then the inverse of a^{-1} is a.

Exercise 1.6. Show that if G is a group and $a, b \in G$, then $(ab)^{-1} = b^{-1}a^{-1}$.

Exercise 1.7. Show that in a group product $a_1 \circ a_2 \circ \cdots \circ a_k$, every possible legal parenthesization of the expression results in the same group element when computed. Therefore parentheses are often omitted from such expressions.

Definition 1.8 (Abelian/commutative group)**.**
We say that two elements $a, b \in G$ of a group *commute* if $ab = ba$. If all pairs of elements of a group commute, then we call the group commutative or Abelian. ■

Exercise 1.9. Give two examples of non-Abelian groups.

Notation 1.10. If G is an Abelian group, we often denote the group operation with a plus $+$ and the identity element with 0. The inverse of an element a is denoted by $-a$.

Notation 1.11. For $n \in \mathbb{N}$, we write a^n as a shorthand for

$$\underbrace{aa\cdots a}_{n \text{ times}},$$

where a^0 is defined to be the identity 1. Similarly, a^{-n} is a shorthand for $a^{-1}a^{-1}\cdots a^{-1}$ (n times). In the case of Abelian groups, we use na to denote $a + a + \cdots + a$ (n times).

1.2 Fields

Definition 1.12 (Field)**.**
A *field* $\mathbb{F}$ is a set together with two binary operations $\cdot$ (called multiplication) and $+$ (called addition) such that:

- $\mathbb{F}$ with the $+$ operation is an Abelian group with $0 \in \mathbb{F}$ being the additive identity element;
- $\mathbb{F}\backslash\{0\}$ with the $\cdot$ operation is an Abelian group with $1 \in \mathbb{F}$ being the multiplicative identity element;
- $\mathbb{F}$ satisfies the *distributive law*, that is, for any $a, b, c \in \mathbb{F}$, we have $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

■

Notation 1.13. In a field $\mathbb{F}$, we often drop $\cdot$ from expressions and write ab instead of $a \cdot b$. Furthermore, given an expression with $+$ and $\cdot$ operations, the $\cdot$ operation has precedence, so an expression like $(a \cdot b) + (a \cdot c)$ is often simply written as $ab + ac$.

Notation 1.14. For $a, b \in \mathbb{F}$, $a + (-b)$ is often abbreviated to $a - b$ and ab^{-1} is often abbreviated to a/b.

Exercise 1.15. Show that $\mathbb{R}, \mathbb{Q}, \mathbb{C}$, and $\mathbb{Z}_p$ for a prime p are all fields. Show that $\mathbb{Z}_6$ is not a field.

Exercise 1.16. Let $\mathbb{F}$ be a field and $a_1, a_2, \ldots, a_n \in \mathbb{F}$ such that $a_1a_2\cdots a_n = 0$. Prove that there exists an i such that $a_i = 0$.

Theorem 1.17. *There is a finite field with q elements if and only if $q = p^k$ for some $k \in \mathbb{N}^+$ and prime p (in this case we say q is a prime power). The field with q elements is unique (up to relabeling the field elements).*[1]

Notation 1.18. Let q be a prime power. The finite field with q elements is denoted by $\mathbb{F}_q$ or $\mathrm{GF}(q)$.

Corollary 1.19. *For a prime p, $\mathbb{F}_p = \mathbb{Z}_p$.*

[1]This means that two fields with q elements must have identical multiplication and addition tables after renaming the elements of the second field with the ones in the first field.

1.3 Polynomials

Definition 1.20 (Polynomial)**.**
Let $\mathbb{F}$ be a field and x be a variable symbol. A *polynomial* over a field $\mathbb{F}$ is defined to be an expression of the form

$$c_d x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0,$$

where each c_i belongs to $\mathbb{F}$, $c_d \neq 0$, and $d \in \mathbb{N}$. The expression 0 is also a polynomial (which does not quite fit the definition in the previous sentence). We call d the *degree* of the polynomial and the c_i's are called *coefficients*. By convention, the degree of 0 is defined to be $-\infty$. ■

Notation 1.21. We often use the notation $P(x)$ to denote a polynomial. The set of all polynomials over a field $\mathbb{F}$ is denoted by $\mathbb{F}[x]$, and we write $P(x) \in \mathbb{F}[x]$ to specify the field $P(x)$ is over. The degree of $P(x)$ is denoted by $\deg(P(x))$.

Remark. A polynomial over a field $\mathbb{F}$ is uniquely identified by its coefficients. So we could have defined a polynomial as a sequence of coefficients $(c_d, c_{d-1}, \ldots, c_1, c_0)$, where each $c_i \in \mathbb{F}$ and $c_d \neq 0$. The reason for defining a polynomial as an expression involving the variable symbol x is because we want to view a polynomial as a function $P : \mathbb{F} \to \mathbb{F}$ that maps a field element to another field element. This is called the *evaluation* of the polynomial and is defined below in Definition 1.28.

Definition 1.22 (Addition and subtraction of polynomials)**.**
Let $P(x) \in \mathbb{F}[x]$ be a polynomial of degree d,

$$P(x) = c_d x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0,$$

and let $Q(x) \in \mathbb{F}[x]$ be a polynomial of degree d',

$$Q(x) = c'_{d'} x^{d'} + c'_{d'-1}x^{d'-1} + \cdots + c'_1 x + c'_0.$$

Assume $d > d'$. Then we define the sum of these polynomials, denoted $P(x) + Q(x)$ (or $Q(x) + P(x)$), to be the polynomial such that for $k = 0, 1, \ldots, d$, the coefficient of x^k is $c_k + c'_k$, where $c'_k = 0$ for $k > d'$.

We define $P(x) - Q(x)$ as $P(x) + (-Q(x))$, where

$$-Q(x) = -c'_{d'} x^{d'} - c'_{d'-1}x^{d'-1} - \cdots - c'_1 x - c'_0.$$

■

Exercise 1.23. Show that $\mathbb{F}[x]$ forms an Abelian group under addition.

Remark. Below we give the definition of multiplication of two polynomials. Even though the formal definition may seem complicated, it does correspond exactly to our knowledge about how two polynomials should be multiplied.

Definition 1.24 (Multiplication of polynomials)**.**
Let $P(x) \in \mathbb{F}[x]$ be a polynomial of degree d,

$$P(x) = c_d x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0,$$

and let $Q(x) \in \mathbb{F}[x]$ be a polynomial of degree d',

$$Q(x) = c'_{d'} x^{d'} + c'_{d'-1}x^{d'-1} + \cdots + c'_1 x + c'_0.$$

Then we define the multiplication/product of these polynomials, denoted $P(x)Q(x)$ (or $Q(x)P(x)$), to be the polynomial such that for $k = 0, 1, \ldots, d + d'$, the coefficient of x^k is

$$\sum_{\substack{i \in \{1,\ldots,d\} \\ j \in \{1,\ldots,d'\} \\ i+j=k}} c_i c'_j.$$

■

Exercise 1.25. Let $P(x) = x^2 + 5x + 10 \in \mathbb{F}_{11}[x]$ and $Q(x) = 3x^3 + 10x \in \mathbb{F}_{11}[x]$. Verify that $P(x)Q(x) = 3x^5 + 4x^4 + 7x^3 + 6x^2 + x$.

Exercise 1.26. Prove the following.

- Let $P(x), Q(x) \in \mathbb{F}[x]$ be two polynomials and $a, b \in \mathbb{F}$ be two elements. Then

$$\deg(aP(x) + bQ(x)) \leq \max\{\deg(P(x)), \deg(Q(x))\}.$$

- Let $P(x), Q(x) \in \mathbb{F}[x]$ be two polynomials. Then

$$\deg(P(x)Q(x)) = \deg(P(x)) + \deg(Q(x)).$$

Remark. Even though $\mathbb{F}[x]$ forms a group under addition, it does not form a group under multiplication because the multiplicative inverse of a polynomial may not exist. This means that given $P(x), Q(x) \in \mathbb{F}[x]$, the division $P(x)/Q(x)$ may not be well-defined. On the other hand, we can divide polynomials with a remainder just like we can divide integers with a remainder.

Proposition 1.27 (Division in polynomials)**.** *Given $A(x), B(x) \in \mathbb{F}[x]$ with $B(x) \neq 0$, there is a way to write*

$$A(x) = Q(x)B(x) + R(x),$$

where $\deg(R(x)) < \deg(B(x))$.

Definition 1.28 (Evaluation of a polynomial)**.**
Let $P(x) \in \mathbb{F}[x]$, i.e.

$$P(x) = c_d x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0.$$

Let $a \in \mathbb{F}$. Then $P(a) \in \mathbb{F}$ denotes the *evaluation* of polynomial $P(x)$ at element a, and is defined as the element obtained by evaluating the following expression in the field $\mathbb{F}$:

$$c_d a^d + c_{d-1} a^{d-1} + \cdots + c_1 a + c_0.$$

■

Definition 1.29 (Root of a polynomial)**.**
Let $P(x) \in \mathbb{F}[x]$. Then $a \in \mathbb{F}$ is called a *root* of $P(x)$ if $P(a) = 0$. ■

Theorem 1.30 (Few-Roots Theorem)**.** *Let $\mathbb{F}$ be a field and $d \in \mathbb{N}$. A degree-d polynomial $P(x) \in \mathbb{F}[x]$ has at most d roots.*

Proof. The proof is by induction on the degree $d \in \mathbb{N}$. The base case is $d = 0$. In this case, we know $P(x) = a$ for some $a \neq 0$. Such a $P(x)$ has 0 roots.

Assume the statement is true for all polynomials of degree up to and including d. Our goal is to show that the statement is true for all polynomials of degree $d+1$. Let $P(x)$ be an arbitrary polynomial of degree $d+1$. If $P(x)$ has no roots, then we are done, so assume b is a root of $P(x)$. If we divide $P(x)$ by $(x-b)$, we can write

$$P(x) = Q(x)(x-b) + R(x),$$

where $\deg(R(x)) < 1$ since $\deg(R(x)) < \deg((x-b))$. This means $R(x) = r$ for some field element r. In the above equation, if we substitute b for x, we get

$$0 = P(b) = Q(b)(b-b) + R(b) = r,$$

which means the remainder was 0. In other words, $P(x) = Q(x)(x-b)$. This implies $\deg(Q(x)) = d$, and by induction hypothesis, $Q(x)$ has at most d roots. It then follows that $P(x)$ has at most $d+1$ roots since if $P(a) = 0$, then it must be the case that either $Q(a) = 0$ or $(a-b) = 0$. □

Corollary 1.31. *Suppose a polynomial $P(x) \in \mathbb{F}[x]$ is such that $\deg(P(x)) \leq d$ and $P(x)$ has more than d roots. Then $P(x) = 0$.*

Exercise 1.32. Explain where the above proof breaks down if we consider polynomials over $\mathbb{Z}_6$, which is not a field.

Theorem 1.33 (Unique Interpolating Polynomial Theorem)**.** *Suppose we are given $(a_1, b_1), (a_2, b_2), \ldots, (a_{d+1}, b_{d+1})$, where each a_i and b_i are elements of $\mathbb{F}$ and the a_i's are distinct. Then there is exactly one polynomial $P(x) \in \mathbb{F}[x]$ of degree at most d such that $P(a_i) = b_i$ for all $i \in \{1, 2, \ldots, d+1\}$.*

Proof. The result follows from the following two claims.

(i) There is at *least* one polynomial of degree at most d satisfying $P(a_i) = b_i$ for all i.

(ii) There is at *most* one polynomial of degree at most d satisfying $P(a_i) = b_i$ for all i.

We start by proving (ii). Assume there are two polynomials $P(x)$ and $Q(x)$ such that $P(a_i) = b_i$ and $Q(a_i) = b_i$ for all i. Define $R(x) = P(x) - Q(x)$. Then notice that $R(a_i) = 0$ for all i, which means $R(x)$ has more than d roots. On the other hand, since $\deg(P) \leq d$ and $\deg(Q) \leq d$, we must have $\deg(R) \leq d$. By Corollary 1.31, this implies $R(x) = 0$, i.e., $P(x) = Q(x)$.

We now prove (i) by explicitly constructing the polynomial $P(x)$ with the desired properties. The method we use for constructing the polynomial is called Lagrange Interpolation. For each $i \in \{1, 2, \ldots, d+1\}$, define the polynomial[2]

$$S_i(x) = \frac{\prod_{j \neq i}(x - a_j)}{\prod_{j \neq i}(a_i - a_j)}.$$

For example,

$$S_1(x) = \frac{(x - a_2)(x - a_3) \cdots (x - a_{d+1})}{(a_1 - a_2)(a_1 - a_3) \cdots (a_1 - a_{d+1})}.$$

Then we define $P(x)$ as follows:

$$P(x) = b_1 S_1(x) + b_2 S_2(x) + \cdots + b_{d+1} S_{d+1}(x).$$

We need to check that $P(x)$ satisfies the desired properties. First, note that $\deg(S_i) = d$ for each i and therefore $\deg(P(x)) \leq d$ (by the first part of Exercise 1.26). Next, observe that by the definitions of $S_i(x)$, we have

$$S_i(x) = \begin{cases} 1 & \text{if } x = a_i, \\ 0 & \text{if } x = a_j \text{ for } j \neq i. \end{cases}$$

This implies that when we plug in a_i to $P(x)$, the only term that survives (i.e. that is non-zero) is $b_i S_i(a_i) = b_i$. So $P(a_i) = b_i$ for all $i \in \{1, 2, \ldots, d+1\}$, as desired. $\square$

Corollary 1.34. *Let $P(x)$ and $Q(x)$ be two polynomials of degree at most d in $\mathbb{F}[x]$ such that $P(x)$ and $Q(x)$ agree on $d+1$ distinct points, i.e. for distinct $a_1, a_2, \ldots, a_{d+1} \in \mathbb{F}$, $P(a_i) = Q(a_i)$ for all i. Then $P(x) = Q(x)$.*

Exercise 1.35. Find a polynomial $P(x) \in \mathbb{F}_7[x]$ of degree at most 2 such that $P(2) = 1$, $P(3) = 3$, and $P(4) = 5$.

Remark. The previous theorem implies that given $d+1$ points with their evaluations, there is a unique polynomial of degree at most d consistent with this given data. This means we can have two different ways of representing a polynomial $P(x)$ of degree at most d:

[2] Note we are not dividing by 0 since the a_i's are distinct.

(i) *Coefficients representation:* We can represent $P(x)$ by the sequence of its coefficients: $(c_d, c_{d-1}, \ldots, c_1, c_0)$.

(ii) *Values representation:* We can represent $P(x)$ by $(a_1, b_1), (a_2, b_2), \ldots, (a_{d+1}, b_{d+1})$, where the a_i's are distinct and $b_i = P(a_i)$ for all i. If we fix the elements $a_1, a_2, \ldots, a_{d+1}$, then we can simply represent the polynomial by the sequence $(b_1, b_2, \ldots, b_{d+1})$.

One can easily go from representation (i) to representation (ii) by evaluating the polynomial at the points $a_1, \ldots, a_{d+1}$. One can also go from representation (ii) to representation (i) by using Lagrange Interpolation (proof of Theorem 1.33). Switching between these two representations of polynomials is often a very useful trick in applications (e.g. in error-correcting codes).

2 Error-Correcting Codes

In this section, we are interested in the following setting. There are two parties Alice and Bob. Alice would like to send a message to Bob over some channel. The channel, however, is noisy. This means that what Bob receives is potentially different from what Alice sends. We would still like for Bob to successfully receive Alice's original message. How can we accomplish this?

Since the channel is noisy, Alice will have to add some redundancy into her message so that even if some of the information is corrupted, there is still enough uncorrupted information left that would allow Bob to recover the original intended message. More specifically, if Alice's original message is M, she *encodes* it to produce the *codeword* M' (which will be longer than M). Bob receives a noisy/corrupted version of M', but is still able to decode it and recover M.

Let's view each symbol/character of M (and M') as an element from a finite field $\mathbb{F}$. Let's in fact fix our field to be $\mathbb{F}_{257}$, which would allow us to represent all letters and common punctuation symbols. If the channel is able to corrupt all the symbols of the codeword M', then we have no hope of recovering the original message M. So we do need to make some assumption on how many symbols can be corrupted. We put an absolute bound on the number of symbols that can be corrupted and denote this number by k.

There are various ways a message can be corrupted by the channel. We will look at two types:

(i) *Erasures*: The channel replaces up to k symbols with a ? mark. For example, if $k = 2$, a message $(118, 114, 120, 85, 66, 78)$ could be received as $(118, 114, ?, 85, ?, 78)$.

(ii) *Errors:* The channel replaces up to k symbols with different elements of the field. For example, if $k = 2$, a message $(118, 114, 120, 85, 66, 78)$ could be received as $(118, 114, 104, 85, 35, 78)$.

Note that type (i) corruptions are easier to handle since we have information about which symbols are corrupted. With type (ii), we have no idea which symbols are correct and which are wrong.

Our goal is to come up with *coding schemes* that specifies the following:

- *Encoding:* How does Alice convert M to M'?
- *Decoding:* How does Bob recover M even if up to k symbols of the codeword M' are corrupted?

Of course, we would like the encoding and decoding to be done efficiently. Furthermore, we would like to minimize $|M'|/|M|$. In other words, we do not want the length of the codeword to be too large compared to the length of the original message.

2.1 Repetition code

A simple way of handling corruption is to build M' by repeating each symbol of M a certain number of times. If we have up to k erasures, then we can repeat every symbol $k+1$ many times. In other words, for each symbol we see in M, we create a block with $k+1$ symbols. The i'th block consists of a repetition of the i'th symbol in M $k+1$ times. For example, if $k = 2$ and $M = (118, 114, 120, 85, 66, 78)$, then we define

$$M' = (118, 118, 118, 114, 114, 114, 120, 120, 120, 85, 85, 85, 66, 66, 66, 78, 78, 78).$$

Now even if 2 symbols are erased above, each block contains at least one good copy of the original symbol. So Bob can easily recover the original message M. Even though this is a simple method, it feels like a wasteful code. Can we have a coding scheme with smaller length codewords?

Before we answer that question, let's also note that a repetition code can be used to handle type (ii) errors. For this, we need to repeat every symbol $2k+1$ many times. Even if k symbols are changed, in each block, the majority of the symbols must be the original symbol. So Bob can recover the original message by looking at each block and selecting the symbol that appears at least $k+1$ many times.

2.2 Reed-Solomon code

The repetition code, although simple, is too wasteful. Polynomials allow us to do much better! Before presenting the coding scheme based on polynomials, let's look at what is the best result we can hope for. That is, if we have k type (i) errors or k type (ii) errors, what is the minimum length for M' that would allow Bob to successfully recover M? For type (i) errors, the best we can hope for is $|M'| = |M| + k$ because otherwise (i.e., if $|M'| < |M| + k$), the corrupted message would have less than $|M|$ symbols, which is not enough information to reliably recover M. One can also argue that for type (ii) errors, the best we can hope for is $|M'| = |M| + 2k$. Reed-Solomon codes that we describe below achieve these bounds.

Handling type (i) errors: Let's say that $|M| = d+1$, so Alice is trying to send $d+1$ elements of the field $\mathbb{F}_{257}$. Let's represent these elements as $(c_d, c_{d-1}, \dots, c_1, c_0)$. Alice views M as a degree d polynomial $P(x) \in \mathbb{F}_{257}[x]$ in a natural way:

$$P(x) = c_d x^d + c_{d-1} x^{d-1} + \ldots + c_1 x + c_0.$$

So the task of sending M to Bob is the same as the task of sending $P(x)$ to him. As we have mentioned at the end of Section 1, one can represent a polynomial in two ways: coefficients representation and values representation. The main trick is to use the values representation to construct the codeword. In particular, Alice sends Bob the evaluation of $P(x)$ at $d+1+k$ points:

$$(P(1), P(2), P(3), \dots, P(d+1+k)).$$

Now even if there are k erasures, we still have the evaluation of $P(x)$ on at least $d+1$ points. This means we can use Lagrange interpolation to find the unique polynomial $P(x)$ that is consistent with the data.

Handling type (ii) errors: To handle type (ii) errors, we basically follow the same scheme as described above, but instead of sending the evaluation of $P(x)$ at $d+1+k$ points, we send the evaluation of $P(x)$ at $d+1+2k$ points:

$$(P(1), P(2), P(3), \dots, P(d+1+2k)).$$

Suppose k of the above elements get changed by the channel. So Bob receives

$$(r_1, r_2, \dots, r_{d+1+2k}),$$

where $r_i = P(i)$ for all but k i's.

Proposition 2.1. *$P(x)$ is the unique polynomial of degree at most d such that $r_i = P(i)$ for all but k i's.*

Proof. Clearly the original polynomial $P(x)$ satisfies the condition. Suppose there is another polynomial $Q(x)$ with the property that $r_i = Q(i)$ for all but k i's. Then note that $P(i) \neq Q(i)$ for at most $2k$ values among $\{1, 2, \dots, d+1+2k\}$.[3] This implies $P(i) = Q(i)$ for at least $(d+1+2k) - 2k = d+1$ values. So by Corollary 1.34, we know $P(x) = Q(x)$. $\square$

By the above proposition, we know that Bob has enough information to figure out the polynomial $P(x)$ and therefore the original message of Alice. In particular, he can compute what $P(x)$ is by a brute-force algorithm as follows. Given $(r_1, r_2, \dots, r_{d+1+2k})$, consider every possible subset S of size k. We want to check if S corresponds to the k error points. Use Lagrange interpolation to see if there is a polynomial of degree at most d consistent with all the points except for the ones in S. If there is, that is the

[3]We are not giving a detailed argument here, but please verify it by yourself.

polynomial we are looking for. Note that by the proposition above, only $P(x)$ can pass this test.

The problem with this scheme is that the decoding process is not efficient. There are $\binom{d+1+2k}{k}$ possible subsets S of size k, and this is a large number for large k (e.g. if $k = d/2$, the number of subsets grows exponentially in d). It turns out there are much better algorithms for recovering $P(x)$. The first polynomial-time algorithm was given by Peterson in 1960. Later, Berlekamp and Massey improved the running time by giving practical algorithms for recovering $P(x)$. This led to the adoption of polynomial-based error-correcting codes in real-world applications.