20

Algorithms for finite fields

This chapter discusses efficient algorithms for factoring polynomials over finite fields, and related problems, such as testing if a given polynomial is irreducible, and generating an irreducible polynomial of given degree.

> *Throughout this chapter, F denotes a finite field of characteristic p and cardinality $q = p^w$.*

In addition to performing the usual arithmetic and comparison operations in F, we assume that our algorithms have access to the numbers p, w, and q, and have the ability to generate random elements of F. Generating such a random field element will count as one "operation in F," along with the usual arithmetic operations. Of course, the "standard" ways of representing F as either $\mathbb{Z}_p$ (if $w = 1$), or as the ring of polynomials modulo an irreducible polynomial over $\mathbb{Z}_p$ of degree w (if $w > 1$), satisfy the above requirements, and also allow for the implementation of arithmetic operations in F that take time $O(\text{len}(q)^2)$ on a RAM (using simple, quadratic-time arithmetic for polynomials and integers).

20.1 Tests for and constructing irreducible polynomials

Let $f \in F[X]$ be a monic polynomial of degree $\ell > 0$. We develop here an efficient algorithm that determines if f is irreducible.

The idea is a simple application of Theorem 19.10. That theorem says that for every integer $k \geq 1$, the polynomial $X^{q^k} - X$ is the product of all monic irreducibles whose degree divides k. Thus, $\gcd(X^q - X, f)$ is the product of all the distinct linear factors of f. If f has no linear factors, then $\gcd(X^{q^2} - X, f)$ is the product of all the distinct quadratic irreducible factors of f. And so on. Now, if f is not irreducible, it must be divisible by some irreducible polynomial of degree at most $\ell/2$, and if g is an irreducible factor of f of minimal degree, say k, then we have $k \leq \ell/2$ and $\gcd(X^{q^k} - X, f) \neq 1$. Conversely, if f is irreducible, then $\gcd(X^{q^k} - X, f) = 1$ for

all positive integers k up to $\ell/2$. So to test if f is irreducible, it suffices to check if $\gcd(X^{q^k} - X, f) = 1$ for all positive integers k up to $\ell/2$—if so, we may conclude that f is irreducible, and otherwise, we may conclude that f is not irreducible. To carry out the computation efficiently, we note that if $h \equiv X^{q^k} \pmod{f}$, then $\gcd(h - X, f) = \gcd(X^{q^k} - X, f)$.

The above observations suggest the following algorithm.

Algorithm IPT**.** On input f, where $f \in F[X]$ is a monic polynomial of degree $\ell > 0$, determine if f is irreducible as follows:

$h \leftarrow X \bmod f$
for $k \leftarrow 1$ to $\lfloor \ell/2 \rfloor$ do
    $h \leftarrow h^q \bmod f$
    if $\gcd(h - X, f) \neq 1$ then return *false*
return *true*

The correctness of Algorithm IPT follows immediately from the above discussion. As for the running time, we have:

Theorem 20.1. *Algorithm* IPT *uses $O(\ell^3 \operatorname{len}(q))$ operations in F.*

Proof. Consider an execution of a single iteration of the main loop. The cost of the qth-powering step (using a standard repeated-squaring algorithm) is $O(\operatorname{len}(q))$ multiplications modulo f, and so $O(\ell^2 \operatorname{len}(q))$ operations in F. The cost of the gcd computation is $O(\ell^2)$ operations in F. Thus, the cost of a single loop iteration is $O(\ell^2 \operatorname{len}(q))$ operations in F, from which it follows that the cost of the entire algorithm is $O(\ell^3 \operatorname{len}(q))$ operations in F. □

Using a standard representation for F, each operation in F takes time $O(\operatorname{len}(q)^2)$ on a RAM, and so the running time of Algorithm IPT on a RAM is $O(\ell^3 \operatorname{len}(q)^3)$, which means that it is a polynomial-time algorithm.

Let us now consider the related problem of constructing an irreducible polynomial of specified degree $\ell > 0$. To do this, we can simply use the result of Theorem 19.12, which has the following probabilistic interpretation: if we choose a random, monic polynomial f of degree ℓ over F, then the probability that f is irreducible is at least $1/2\ell$. This suggests the following probabilistic algorithm:

Algorithm RIP**.** On input ℓ, where ℓ is a positive integer, generate a monic irreducible polynomial $f \in F[X]$ of degree ℓ as follows:

repeat
    choose $c_0, \ldots, c_{\ell-1} \in F$ at random
    set $f \leftarrow X^\ell + \sum_{i=0}^{\ell-1} c_i X^i$
    test if f is irreducible using Algorithm IPT
until f is irreducible
output f

Theorem 20.2. *Algorithm* RIP *uses an expected number of* $O(\ell^4 \operatorname{len}(q))$ *operations in* F*, and its output is uniformly distributed over all monic irreducibles of degree* ℓ*.*

Proof. This is a simple application of the generate-and-test paradigm (see Theorem 9.3, and Example 9.10 in particular). Because of Theorem 19.12, the expected number of loop iterations of the above algorithm is $O(\ell)$. Since Algorithm IPT uses $O(\ell^3 \operatorname{len}(q))$ operations in F, the statement about the running time of Algorithm RIP is immediate. The statement about its output distribution is clear. □

The expected running-time bound in Theorem 20.2 is actually a bit of an overestimate. The reason is that if we generate a random polynomial of degree ℓ, it is likely to have a small irreducible factor, which will be discovered very quickly by Algorithm IPT. In fact, it is known (see §20.7) that the expected value of the degree of the least degree irreducible factor of a random monic polynomial of degree ℓ over F is $O(\operatorname{len}(\ell))$, from which it follows that the expected number of operations in F performed by Algorithm RIP is actually $O(\ell^3 \operatorname{len}(\ell) \operatorname{len}(q))$.

EXERCISE 20.1. Let $f \in F[X]$ be a monic polynomial of degree $\ell > 0$. Also, let $\xi := [X]_f \in E$, where E is the F-algebra $E := F[X]/(f)$.

(a) Given as input $\alpha \in E$ and $\xi^{q^m} \in E$ (for some integer $m > 0$), show how to compute the value $\alpha^{q^m} \in E$, using just $O(\ell^{2.5})$ operations in F, and space for $O(\ell^{1.5})$ elements of F. Hint: see Theorems 16.7 and 19.7, as well as Exercise 17.3.

(b) Given as input $\xi^{q^m} \in E$ and $\xi^{q^{m'}} \in E$, where m and m' are positive integers, show how to compute the value $\xi^{q^{m+m'}} \in E$, using $O(\ell^{2.5})$ operations in F, and space for $O(\ell^{1.5})$ elements of F.

(c) Given as input $\xi^q \in E$ and a positive integer m, show how to compute the value $\xi^{q^m} \in E$, using $O(\ell^{2.5} \operatorname{len}(m))$ operations in F, and space for $O(\ell^{1.5})$ elements of F. Hint: use a repeated-squaring-like algorithm.

EXERCISE 20.2. This exercise develops an alternative irreducibility test.

(a) Show that a monic polynomial $f \in F[X]$ of degree $\ell > 0$ is irreducible if and only if $X^{q^\ell} \equiv X \pmod{f}$ and $\gcd(X^{q^{\ell/s}} - X, f) = 1$ for all primes $s \mid \ell$.

(b) Using part (a) and the result of the previous exercise, show how to determine if f is irreducible using $O(\ell^{2.5}\operatorname{len}(\ell)\omega(\ell) + \ell^2\operatorname{len}(q))$ operations in F, where $\omega(\ell)$ is the number of distinct prime factors of ℓ.

(c) Show that the operation count in part (b) can be reduced to

$$O(\ell^{2.5}\operatorname{len}(\ell)\operatorname{len}(\omega(\ell)) + \ell^2\operatorname{len}(q)).$$

Hint: see Exercise 3.39.

EXERCISE 20.3. Design and analyze a *deterministic* algorithm that takes as input a list of irreducible polynomials $f_1, \ldots, f_r \in F[X]$, where $\ell_i := \deg(f_i)$ for $i = 1, \ldots, r$, and assume that $\{\ell_i\}_{i=1}^r$ is pairwise relatively prime. Your algorithm should output an irreducible polynomial $f \in F[X]$ of degree $\ell := \prod_{i=1}^r \ell_i$ using $O(\ell^3)$ operations in F. Hint: use Exercise 19.5.

EXERCISE 20.4. Design and analyze a probabilistic algorithm that, given a monic irreducible polynomial $f \in F[X]$ of degree ℓ as input, generates as output a random monic irreducible polynomial $g \in F[X]$ of degree ℓ (i.e., g should be uniformly distributed over all such polynomials), using an expected number of $O(\ell^{2.5})$ operations in F. Hint: use Exercise 18.9 (or alternatively, Exercise 18.10).

EXERCISE 20.5. Let $f \in F[X]$ be a monic irreducible polynomial of degree ℓ, let $E := F[X]/(f)$, and let $\xi := [X]_f \in E$. Design and analyze a deterministic algorithm that takes as input the polynomial f defining the extension E, and outputs the values

$$s_j := \mathbf{Tr}_{E/F}(\xi^j) \in F \quad (j = 0, \ldots, \ell - 1),$$

using $O(\ell^2)$ operations in F. Here, $\mathbf{Tr}_{E/F}$ is the trace from E to F (see §19.4). Show that given an arbitrary $\alpha \in E$, along with the values $s_0, \ldots, s_{\ell-1}$, one can compute $\mathbf{Tr}_{E/F}(\alpha)$ using just $O(\ell)$ operations in F.

20.2 Computing minimal polynomials in $F[X]/(f)$ (III)

We consider, for the third and final time, the problem considered in §17.2 and §18.5: $f \in F[X]$ is a monic polynomial of degree $\ell > 0$, and $E := F[X]/(f) = F[\xi]$, where $\xi := [X]_f$; we are given an element $\alpha \in E$, and want to compute the minimal polynomial $\phi \in F[X]$ of α over F. We develop an alternative algorithm, based on the theory of finite fields. Unlike the algorithms in §17.2 and §18.5, this algorithm only works when F is finite and the polynomial f is irreducible, so that E is also a finite field.

From Theorem 19.15, we know that the degree of α over F is the smallest positive integer k such that $\alpha^{q^k} = \alpha$. By successive qth powering, we can determine

the degree k and compute the conjugates $\alpha, \alpha^q, \ldots, \alpha^{q^{k-1}}$ of α, using $O(k \operatorname{len}(q))$ operations in E, and hence $O(k\ell^2 \operatorname{len}(q))$ operations in F.

Now, we could simply compute the minimal polynomial ϕ by directly using the formula

$$\phi(Y) = \prod_{i=0}^{k-1} (Y - \alpha^{q^i}). \tag{20.1}$$

This would involve computations with polynomials in the variable Y whose coefficients lie in the extension field E, although at the end of the computation, we would end up with a polynomial all of whose coefficients lie in F. The cost of this approach would be $O(k^2)$ operations in E, and hence $O(k^2\ell^2)$ operations in F.

A more efficient approach is the following. Substituting ξ for Y in the identity (20.1), we have

$$\phi(\xi) = \prod_{i=0}^{k-1} (\xi - \alpha^{q^i}).$$

Using this formula, we can compute (given the conjugates of α) the value $\phi(\xi) \in E$ using $O(k)$ operations in E, and hence $O(k\ell^2)$ operations in F. Now, $\phi(\xi)$ is an element of E, and for computational purposes, it is represented as $[g]_f$ for some polynomial $g \in F[X]$ of degree less than ℓ. Moreover, $\phi(\xi) = [\phi]_f$, and hence $\phi \equiv g \pmod{f}$. In particular, if $k < \ell$, then $g = \phi$; otherwise, if $k = \ell$, then $g = \phi - f$. In either case, we can recover ϕ from g with an additional $O(\ell)$ operations in F.

Thus, given the conjugates of α, we can compute ϕ using $O(k\ell^2)$ operations in F. Adding in the cost of computing the conjugates, this gives rise to an algorithm that computes the minimal polynomial of α using $O(k\ell^2 \operatorname{len}(q))$ operations in F.

In the worst case, then, this algorithm uses $O(\ell^3 \operatorname{len}(q))$ operations in F. A reasonably careful implementation needs space for storing a constant number of elements of E, and hence $O(\ell)$ elements of F. For very small values of q, the efficiency of this algorithm will be comparable to that of the algorithm in §18.5, but for large q, it will be much less efficient. Thus, this approach does not really yield a better algorithm, but it does serve to illustrate some of the ideas of the theory of finite fields.

20.3 Factoring polynomials: square-free decomposition

In the remaining sections of this chapter, we develop efficient algorithms for factoring polynomials over the finite field F. We begin in this section with a simple and efficient preprocessing step. Recall that a polynomial is called square-free if it is not divisible by the square of any polynomial of degree greater than zero. This

preprocessing algorithm takes the polynomial to be factored, and partially factors it into a product of square-free polynomials. Given this algorithm, we can focus our attention on the problem of factoring square-free polynomials.

Let $f \in F[X]$ be a monic polynomial of degree $\ell > 0$. Suppose that f is not square-free. According to Theorem 19.1, $d := \gcd(f, \mathbf{D}(f)) \neq 1$, where $\mathbf{D}(f)$ is the formal derivative of f; thus, we might hope to get a non-trivial factorization of f by computing d. However, we have to consider the possibility that $d = f$. Can this happen? The answer is "yes," but if it does happen that $d = f$, we can still get a non-trivial factorization of f by other means:

Theorem 20.3. *Suppose that $f \in F[X]$ is a monic polynomial of degree $\ell > 0$, and that $\gcd(f, \mathbf{D}(f)) = f$. Then $f = g(X^p)$ for some $g \in F[X]$. Moreover, if $g = \sum_i a_i X^i$, then $f = h^p$, where*

$$h = \sum_i a_i^{p^{(w-1)}} X^i. \tag{20.2}$$

Proof. Since $\deg(\mathbf{D}(f)) < \deg(f)$ and $\gcd(f, \mathbf{D}(f)) = f$, we must have $\mathbf{D}(f) = 0$. If $f = \sum_i c_i X^i$, then $\mathbf{D}(f) = \sum_i i c_i X^{i-1}$. Since this derivative must be zero, it follows that all the coefficients c_i with $i \not\equiv 0 \pmod{p}$ must be zero to begin with. That proves that $f = g(X^p)$ for some $g \in F[X]$. Furthermore, if h is defined as above, then

$$h^p = \Big(\sum_i a_i^{p^{(w-1)}} X^i \Big)^p = \sum_i a_i^{p^w} X^{ip} = \sum_i a_i (X^p)^i = g(X^p) = f. \quad \square$$

Our goal now is to design an efficient algorithm that takes as input a monic polynomial $f \in F[X]$ of degree $\ell > 0$, and outputs a list of pairs $((g_1, s_1), \ldots, (g_t, s_t))$, where

- each $g_i \in F[X]$ is monic, non-constant, and square-free,
- each s_i is a positive integer,
- the family of polynomials $\{g_i\}_{i=1}^t$ is pairwise relatively prime, and
- $f = \prod_{i=1}^t g_i^{s_i}$.

We call such a list a **square-free decomposition of** f. There are a number of ways to do this. The algorithm we present is based on the following theorem, which itself is a simple consequence of Theorem 20.3.

Theorem 20.4. *Let $f \in F[X]$ be a monic polynomial of degree $\ell > 0$. Suppose that the factorization of f into irreducibles is $f = f_1^{e_1} \cdots f_r^{e_r}$. Then*

$$\frac{f}{\gcd(f, \mathbf{D}(f))} = \prod_{\substack{1 \leq i \leq r \\ e_i \not\equiv 0 \pmod{p}}} f_i.$$

Proof. The theorem can be restated in terms of the following claim: for each $i = 1, \ldots, r$, we have

- $f_i^{e_i} \mid \mathbf{D}(f)$ if $e_i \equiv 0 \pmod{p}$, and
- $f_i^{e_i - 1} \mid \mathbf{D}(f)$ but $f_i^{e_i} \nmid \mathbf{D}(f)$ if $e_i \not\equiv 0 \pmod{p}$.

To prove the claim, we take formal derivatives using the usual rule for products, obtaining

$$\mathbf{D}(f) = \sum_j e_j f_j^{e_j - 1} \mathbf{D}(f_j) \prod_{k \neq j} f_k^{e_k}. \tag{20.3}$$

Consider a fixed index i. Clearly, $f_i^{e_i}$ divides every term in the sum on the right-hand side of (20.3), with the possible exception of the term with $j = i$. In the case where $e_i \equiv 0 \pmod{p}$, the term with $j = i$ vanishes, and that proves the claim in this case. So assume that $e_i \not\equiv 0 \pmod{p}$. By the previous theorem, and the fact that f_i is irreducible, and in particular, not the pth power of any polynomial, we see that $\mathbf{D}(f_i)$ is non-zero, and (of course) has degree strictly less than that of f_i. From this, and (again) the fact that f_i is irreducible, it follows that the term with $j = i$ is divisible by $f_i^{e_i - 1}$, but not by $f_i^{e_i}$, from which the claim follows. □

This theorem provides the justification for the following square-free decomposition algorithm.

Algorithm SFD**.** On input f, where $f \in F[X]$ is a monic polynomial of degree $\ell > 0$, compute a square-free decomposition of f as follows:

```
initialize an empty list L
s ← 1
repeat
    j ← 1, g ← f / gcd(f, D(f))
    while g ≠ 1 do
        f ← f/g, h ← gcd(f, g), m ← g/h
        if m ≠ 1 then append (m, js) to L
        g ← h, j ← j + 1
    if f ≠ 1 then // f is a pth power
        // compute a pth root as in (20.2)
        f ← f^{1/p}, s ← ps
until f = 1
output L
```

Theorem 20.5. *Algorithm* SFD *correctly computes a square-free decomposition of f using $O(\ell^2 + \ell(w-1)\operatorname{len}(p)/p)$ operations in F.*

Proof. Let $f = \prod_i f_i^{e_i}$ be the factorization of the input f into irreducibles. Let S

be the set of indices i such that $e_i \not\equiv 0 \pmod{p}$, and let S' be the set of indices i such that $e_i \equiv 0 \pmod{p}$. Also, for $j \geq 1$, let $S_{\geq j} := \{i \in S : e_i \geq j\}$ and $S_{=j} := \{i \in S : e_i = j\}$.

Consider the first iteration of the main loop. By Theorem 20.4, the value first assigned to g is $\prod_{i \in S} f_i$. It is straightforward to prove by induction on j that at the beginning of the jth iteration of the inner while loop, the value assigned to g is $\prod_{i \in S_{\geq j}} f_i$, and the value assigned to f is $\prod_{i \in S_{\geq j}} f_i^{e_i - j + 1} \cdot \prod_{i \in S'} f_i^{e_i}$. Moreover, in the jth loop iteration, the value assigned to m is $\prod_{i \in S_{=j}} f_i$. It follows that when the while loop terminates, the value assigned to f is $\prod_{i \in S'} f_i^{e_i}$, and the value assigned to L is a square-free decomposition of $\prod_{i \in S} f_i^{e_i}$; if f does not equal 1 at this point, then subsequent iterations of the main loop will append to L a square-free decomposition of $\prod_{i \in S'} f_i^{e_i}$.

That proves the correctness of the algorithm. Now consider its running time. Again, consider just the first iteration of the main loop. The cost of computing $f / \gcd(f, \mathbf{D}(f))$ is at most $C_1 \ell^2$ operations in F, for some constant C_1. Now consider the cost of the inner while loop. It is not hard to see that the cost of the jth iteration of the inner while loop is at most

$$C_2 \ell \sum_{i \in S_{\geq j}} \deg(f_i)$$

operations in F, for some constant C_2. This follows from the observation in the previous paragraph that the value assigned to g is $\prod_{i \in S_{\geq j}} f_i$, along with our usual cost estimates for division and Euclid's algorithm. Therefore, the total cost of all iterations of the inner while loop is at most

$$C_2 \ell \sum_{j \geq 1} \sum_{i \in S_{\geq j}} \deg(f_i)$$

operations in F. In this double summation, for each $i \in S$, the term $\deg(f_i)$ is counted exactly e_i times, and so we can write this cost estimate as

$$C_2 \ell \sum_{i \in S} e_i \deg(f_i) \leq C_2 \ell^2.$$

Finally, it is easy to see that in the if-then statement at the end of the main loop body, if the algorithm does in fact compute a pth root, then this takes at most

$$C_3 \ell (w - 1) \operatorname{len}(p)/p$$

operations in F, for some constant C_3. Thus, we have shown that the total cost of the first iteration of the main loop is at most

$$(C_1 + C_2)\ell^2 + C_3 \ell (w - 1) \operatorname{len}(p)/p$$

operations in F. If the main loop is executed a second time, the degree of f at the start of the second iteration is at most ℓ/p, and hence the cost of the second loop iteration is at most

$$(C_1 + C_2)(\ell/p)^2 + C_3(\ell/p)(w-1)\operatorname{len}(p)/p$$

operations in F. More generally, for $t = 1, 2, \ldots$, the cost of loop iteration t is at most

$$(C_1 + C_2)(\ell/p^{t-1})^2 + C_3(\ell/p^{t-1})(w-1)\operatorname{len}(p)/p,$$

operations in F, and summing over all $t \geq 1$ yields the stated bound. □

20.4 Factoring polynomials: the Cantor–Zassenhaus algorithm

In this section, we present an algorithm due to Cantor and Zassenhaus for factoring a given polynomial over the finite field F into irreducibles. We shall assume that the input polynomial is square-free, using Algorithm SFD in §20.3 as a preprocessing step, if necessary. The algorithm has two stages:

Distinct Degree Factorization: The input polynomial is decomposed into factors so that each factor is a product of distinct irreducibles of the same degree (and the degree of those irreducibles is also determined).

Equal Degree Factorization: Each of the factors produced in the distinct degree factorization stage are further factored into their irreducible factors.

The algorithm we present for distinct degree factorization is a deterministic, polynomial-time algorithm. The algorithm we present for equal degree factorization is a *probabilistic* algorithm that runs in expected polynomial time (and whose output is always correct).

20.4.1 Distinct degree factorization

The problem, more precisely stated, is this: given a monic, square-free polynomial $f \in F[X]$ of degree $\ell > 0$, produce a list of pairs $((g_1, k_1), \ldots, (g_t, k_t))$ where

- each g_i is the product of monic irreducible polynomials of degree k_i, and
- $f = \prod_{i=1}^{t} g_i$.

This problem can be easily solved using Theorem 19.10, using a simple variation of the algorithm we discussed in §20.1 for irreducibility testing. The basic idea is this. We can compute $g := \gcd(X^q - X, f)$, so that g is the product of all the linear factors of f. After removing all linear factors from f, we next compute $\gcd(X^{q^2} - X, f)$, which will be the product of all the quadratic irreducibles dividing f, and we can remove these from f—although $X^{q^2} - X$ is the product of all linear

and quadratic irreducibles, since we have already removed the linear factors from f, the gcd will give us just the quadratic factors of f. In general, for $k = 1, \ldots, \ell$, having removed all the irreducible factors of degree less than k from f, we compute $\gcd(X^{q^k} - X, f)$ to obtain the product of all the irreducible factors of f of degree k, and then remove these from f.

The above discussion leads to the following algorithm for distinct degree factorization.

Algorithm DDF**.** On input f, where $f \in F[X]$ is a monic square-free polynomial of degree $\ell > 0$, do the following:

```
initialize an empty list L
h ← X mod f
k ← 0
while f ≠ 1 do
    h ← h^q mod f, k ← k + 1
    g ← gcd(h − X, f)
    if g ≠ 1 then
        append (g, k) to L
        f ← f/g
        h ← h mod f
output L
```

The correctness of Algorithm DDF follows from the discussion above. As for the running time:

Theorem 20.6. *Algorithm* DDF *uses* $O(\ell^3 \operatorname{len}(q))$ *operations in* F.

Proof. Note that the body of the main loop is executed at most ℓ times, since after ℓ iterations, we will have removed all the factors of f. Thus, we perform at most ℓ qth-powering steps, each of which takes $O(\ell^2 \operatorname{len}(q))$ operations in F, and so the total contribution to the running time of these is $O(\ell^3 \operatorname{len}(q))$ operations in F. We also have to take into account the cost of the gcd and division computations. The cost per loop iteration of these is $O(\ell^2)$ operations in F, contributing a term of $O(\ell^3)$ to the total operation count. This term is dominated by the cost of the qth-powering steps, and so the total cost of Algorithm DDF is $O(\ell^3 \operatorname{len}(q))$ operations in F. □

20.4.2 Equal degree factorization

The problem, more precisely stated, is this: given a monic polynomial $f \in F[X]$ of degree $\ell > 0$, and an integer $k > 0$, such that f is of the form

$$f = f_1 \cdots f_r$$

for distinct monic irreducible polynomials $f_1, \ldots, f_r$, each of degree k, compute these irreducible factors of f. Note that given f and k, the value of r is easily determined, since $r = \ell/k$.

We begin by discussing the basic mathematical ideas that will allow us to efficiently split f into two non-trivial factors, and then we present a somewhat more elaborate algorithm that completely factors f.

By the Chinese remainder theorem, we have an F-algebra isomorphism

$$\begin{aligned} \theta : \quad & E \to E_1 \times \cdots \times E_r \\ & [g]_f \mapsto ([g]_{f_1}, \ldots, [g]_{f_r}), \end{aligned}$$

where E is the F-algebra $F[X]/(f)$, and for $i = 1, \ldots, r$, E_i is the extension field $F[X]/(f_i)$ of degree k over F.

Recall that $q = p^w$. We have to treat the cases $p = 2$ and $p > 2$ separately. We first treat the case $p = 2$. Let us define the polynomial

$$M_k := \sum_{j=0}^{wk-1} X^{2^j} \in F[X]. \tag{20.4}$$

(The algorithm in the case $p > 2$ will only differ in the definition of M_k.)

For $\alpha \in E$, if $\theta(\alpha) = (\alpha_1, \ldots, \alpha_r)$, then we have

$$\theta(M_k(\alpha)) = M_k(\theta(\alpha)) = (M_k(\alpha_1), \ldots, M_k(\alpha_r)).$$

Note that each E_i is an extension of $\mathbb{Z}_2$ of degree wk, and that

$$M_k(\alpha_i) = \sum_{j=0}^{wk-1} \alpha_i^{2^j} = \mathbf{Tr}_{E_i/\mathbb{Z}_2}(\alpha_i),$$

where $\mathbf{Tr}_{E_i/\mathbb{Z}_2} : E_i \to \mathbb{Z}_2$ is the trace from E_i to $\mathbb{Z}_2$, which is a surjective, $\mathbb{Z}_2$-linear map (see §19.4).

Now, suppose we choose $\alpha \in E$ at random. Then if $\theta(\alpha) = (\alpha_1, \ldots, \alpha_r)$, the family of random variables $\{\alpha_i\}_{i=1}^r$ is mutually independent, with each α_i uniformly distributed over E_i. It follows that the family of random variables $\{M_k(\alpha_i)\}_{i=1}^r$ is mutually independent, with each $M_k(\alpha_i)$ uniformly distributed over $\mathbb{Z}_2$. Thus, if $g := \mathrm{rep}(M_k(\alpha))$ (i.e., $g \in F[X]$ is the polynomial of degree less than ℓ such that $M_k(\alpha) = [g]_f$), then $\gcd(g, f)$ will be the product of those factors f_i of f such that $M_k(\alpha_i) = 0$. We will fail to get a non-trivial factorization only if the $M_k(\alpha_i)$

are either all 0 or all 1, which for $r \geq 2$ happens with probability at most $1/2$ (the worst case being when $r = 2$).

That is our basic splitting strategy. The algorithm for completely factoring f works as follows. The algorithm proceeds in stages. At any stage, we have a partial factorization $f = \prod_{h \in H} h$, where H is a set of non-constant, monic polynomials. Initially, $H = \{f\}$. With each stage, we attempt to get a finer factorization of f by trying to split each $h \in H$ using the above splitting strategy—if we succeed in splitting h into two non-trivial factors, then we replace h by these two factors. We continue in this way until $|H| = r$.

Here is the full equal degree factorization algorithm.

Algorithm EDF**.** On input f, k, where $f \in F[X]$ is a monic polynomial of degree $\ell > 0$, and k is a positive integer, such that f is the product of $r := \ell/k$ distinct monic irreducible polynomials, each of degree k, do the following, with M_k as defined in (20.4):

$H \leftarrow \{f\}$
while $|H| < r$ do
  $H' \leftarrow \emptyset$
  for each $h \in H$ do
    choose $\alpha \in F[X]/(h)$ at random
    $d \leftarrow \gcd(\mathrm{rep}(M_k(\alpha)), h)$
    if $d = 1$ or $d = h$
      then $H' \leftarrow H' \cup \{h\}$
      else $H' \leftarrow H' \cup \{d, h/d\}$
  $H \leftarrow H'$
output H

The correctness of the algorithm is clear from the above discussion. As for its expected running time, we can get a quick-and-dirty upper bound as follows:

- For a given h and $\alpha \in F[X]/(h)$, the value $M_k(\alpha)$ can be computed using $O(k \deg(h)^2 \operatorname{len}(q))$ operations in F, and so the number of operations in F performed in each iteration of the main loop is at most a constant times

$$k \operatorname{len}(q) \sum_{h \in H} \deg(h)^2 \leq k \operatorname{len}(q) \Big(\sum_{h \in H} \deg(h) \Big)^2 = k\ell^2 \operatorname{len}(q).$$

- The expected number of iterations of the main loop until we get some non-trivial split is $O(1)$.
- The algorithm finishes after getting $r - 1$ non-trivial splits.

- Therefore, the total expected cost is $O(rk\ell^2 \operatorname{len}(q))$, or $O(\ell^3 \operatorname{len}(q))$, operations in F.

This analysis gives a bit of an over-estimate — it does not take into account the fact that we expect to get fairly "balanced" splits. For the purposes of analyzing the overall running time of the Cantor–Zassenhaus algorithm, this bound suffices; however, the following analysis gives a tight bound on the complexity of Algorithm EDF.

Theorem 20.7. *In the case $p = 2$, Algorithm* EDF *uses an expected number of $O(k\ell^2 \operatorname{len}(q))$ operations in F.*

Proof. We may assume $r \geq 2$. Let L be the random variable that represents the number of iterations of the main loop of the algorithm. For $n \geq 1$, let H_n be the random variable that represents the value of H at the beginning of the nth loop iteration. For $i, j = 1, \ldots, r$, we define L_{ij} to be the largest value of n (with $1 \leq n \leq L$) such that $f_i \mid h$ and $f_j \mid h$ for some $h \in H_n$.

We first claim that $\mathsf{E}[L] = O(\operatorname{len}(r))$. To prove this claim, we make use of the fact (see Theorem 8.17) that

$$\mathsf{E}[L] = \sum_{n \geq 1} \mathsf{P}[L \geq n].$$

Now, $L \geq n$ if and only if for some i, j with $1 \leq i < j \leq r$, we have $L_{ij} \geq n$. Moreover, if f_i and f_j have not been separated at the beginning of one loop iteration, then they will be separated at the beginning of the next with probability $1/2$. It follows that

$$\mathsf{P}[L_{ij} \geq n] = 2^{-(n-1)}.$$

So we have

$$\mathsf{P}[L \geq n] \leq \sum_{i<j} \mathsf{P}[L_{ij} \geq n] \leq r^2 2^{-n}.$$

Therefore,

$$\begin{aligned} \mathsf{E}[L] = \sum_{n \geq 1} \mathsf{P}[L \geq n] &= \sum_{n \leq 2\log_2 r} \mathsf{P}[L \geq n] + \sum_{n > 2\log_2 r} \mathsf{P}[L \geq n] \\ &\leq 2\log_2 r + \sum_{n > 2\log_2 r} r^2 2^{-n} \leq 2\log_2 r + \sum_{n \geq 0} 2^{-n} = 2\log_2 r + 2, \end{aligned}$$

which proves the claim.

As discussed in the paragraph above this theorem, the cost of each iteration of the main loop is $O(k\ell^2 \operatorname{len}(q))$ operations in F. Combining this with the fact that $\mathsf{E}[L] = O(\operatorname{len}(r))$, it follows that the expected number of operations in F for the

entire algorithm is $O(\text{len}(r)k\ell^2 \text{len}(q))$. This is significantly better than the above quick-and-dirty estimate, but is not quite the result we are after. For this, we have to work a little harder.

For each polynomial h dividing f, define $\omega(h)$ to be the number of irreducible factors of h. Let us also define the random variable

$$S := \sum_{n=1}^{L} \sum_{h \in H_n} \omega(h)^2.$$

It is easy to see that the total number of operations performed by the algorithm is $O(Sk^3 \text{len}(q))$, and so it will suffice to show that $\mathsf{E}[S] = O(r^2)$.

We claim that

$$S = \sum_{i,j} L_{ij},$$

where the sum is over all $i, j = 1, \ldots, r$. To see this, define $\delta_{ij}(h)$ to be 1 if both f_i and f_j divide h, and 0 otherwise. Then we have

$$S = \sum_{n} \sum_{h \in H_n} \sum_{i,j} \delta_{ij}(h) = \sum_{i,j} \sum_{n} \sum_{h \in H_n} \delta_{ij}(h) = \sum_{i,j} L_{ij},$$

which proves the claim.

We can write

$$S = \sum_{i \neq j} L_{ij} + \sum_{i} L_{ii} = \sum_{i \neq j} L_{ij} + rL.$$

For $i \neq j$, we have

$$\mathsf{E}[L_{ij}] = \sum_{n \geq 1} \mathsf{P}[L_{ij} \geq n] = \sum_{i \geq 1} 2^{-(n-1)} = 2,$$

and so

$$\mathsf{E}[S] = \sum_{i \neq j} \mathsf{E}[L_{ij}] + r\,\mathsf{E}[L] = 2r(r-1) + O(r \text{len}(r)) = O(r^2).$$

That proves the theorem. □

That completes the discussion of Algorithm EDF in the case $p = 2$. Now assume that $p > 2$, so that p, and hence also q, is odd. Algorithm EDF in this case is exactly the same as above, except that in this case, we define the polynomial M_k as

$$M_k := X^{(q^k-1)/2} - 1 \in F[X]. \tag{20.5}$$

Just as before, for $\alpha \in E$ with $\theta(\alpha) = (\alpha_1, \ldots, \alpha_r)$, we have

$$\theta(M_k(\alpha)) = M_k(\theta(\alpha)) = (M_k(\alpha_1), \ldots, M_k(\alpha_r)).$$

Note that each group E_i^* is a cyclic group of order $q^k - 1$, and therefore, the image of the $(q^k - 1)/2$-power map on E_i^* is $\{\pm 1\}$.

Now, suppose we choose $\alpha \in E$ at random. Then if $\theta(\alpha) = (\alpha_1, \ldots, \alpha_r)$, the family of random variables $\{\alpha_i\}_{i=1}^r$ is mutually independent, with each α_i uniformly distributed over E_i. It follows that the family of random variables $\{M_k(\alpha_i)\}_{i=1}^r$ is mutually independent. If $\alpha_i = 0$, which happens with probability $1/q^k$, then $M_k(\alpha_i) = -1$; otherwise, $\alpha_i^{(q^k-1)/2}$ is uniformly distributed over $\{\pm 1\}$, and so $M_k(\alpha_i)$ is uniformly distributed over $\{0, -2\}$. That is to say,

$$M_k(\alpha_i) = \begin{cases} 0 & \text{with probability } (q^k - 1)/2q^k, \\ -1 & \text{with probability } 1/q^k, \\ -2 & \text{with probability } (q^k - 1)/2q^k. \end{cases}$$

Thus, if $g := \mathrm{rep}(M_k(\alpha))$, then $\gcd(g, f)$ will be the product of those factors f_i of f such that $M_k(\alpha_i) = 0$. We will fail to get a non-trivial factorization only if the $M_k(\alpha_i)$ are either all zero or all non-zero. Assume $r \geq 2$. Consider the worst case, namely, when $r = 2$. In this case, a simple calculation shows that the probability that we fail to split these two factors is

$$\left(\frac{q^k - 1}{2q^k}\right)^2 + \left(\frac{q^k + 1}{2q^k}\right)^2 = \frac{1}{2}(1 + 1/q^{2k}).$$

The (very) worst case is when $q^k = 3$, in which case the probability of failure is at most $5/9$.

The same quick-and-dirty analysis given just above Theorem 20.7 applies here as well, but just as before, we can do better:

Theorem 20.8. *In the case $p > 2$, Algorithm* EDF *uses an expected number of $O(k\ell^2 \,\mathrm{len}(q))$ operations in F.*

Proof. The analysis is essentially the same as in the case $p = 2$, except that now the probability that we fail to split a given pair of irreducible factors is at most $5/9$, rather than equal to $1/2$. The details are left as an exercise for the reader. □

20.4.3 Analysis of the whole algorithm

Given an arbitrary monic square-free polynomial $f \in F[X]$ of degree $\ell > 0$, the distinct degree factorization step takes $O(\ell^3 \,\mathrm{len}(q))$ operations in F. This step produces a number of polynomials that must be further subjected to equal degree factorization. If there are t such polynomials, where the ith polynomial has degree ℓ_i, for $i = 1, \ldots, t$, then $\sum_{i=1}^t \ell_i = \ell$. Now, the equal degree factorization step for the ith polynomial takes an expected number of $O(\ell_i^3 \,\mathrm{len}(q))$ operations in F (actually, our initial, "quick and dirty" estimate is good enough here), and so it

follows that the total expected cost of all the equal degree factorization steps is $O(\sum_i \ell_i^3 \operatorname{len}(q))$, which is $O(\ell^3 \operatorname{len}(q))$, operations in F. Putting this all together, we conclude:

Theorem 20.9. *The Cantor–Zassenhaus factoring algorithm uses an expected number of $O(\ell^3 \operatorname{len}(q))$ operations in F.*

This bound is tight, since in the worst case, when the input is irreducible, the algorithm really does do this much work. Also, we have assumed the input to the Cantor–Zassenhaus is a square-free polynomial. However, we may use Algorithm SFD as a preprocessing step to ensure that this is the case. Even if we include the cost of this preprocessing step, the running time estimate in Theorem 20.9 remains valid.

EXERCISE 20.6. Show how to modify Algorithm DDF so that the main loop halts as soon as $2k > \deg(f)$.

EXERCISE 20.7. Suppose that in Algorithm EDF, we replace the two lines

```
for each h ∈ H do
    choose α ∈ F[X]/(h) at random
```

by the following:

```
choose a_0, ..., a_{2k-1} ∈ F at random
g ← Σ_{j=0}^{2k-1} a_j X^j ∈ F[X]
for each h ∈ H do
    α ← [g]_h ∈ F[X]/(h)
```

Show that the expected running time bound of Theorem 20.6 still holds (you may assume $p = 2$ for simplicity).

EXERCISE 20.8. This exercise extends the techniques developed in Exercise 20.1. Let $f \in F[X]$ be a monic polynomial of degree $\ell > 0$, and let $\xi := [X]_f \in E$, where $E := F[X]/(f)$. For each integer $m > 0$, define polynomials

$$T_m := X + X^q + \cdots + X^{q^{m-1}} \in F[X] \quad \text{and} \quad N_m := X \cdot X^q \cdot \cdots \cdot X^{q^{m-1}} \in F[X].$$

(a) Given as input $\xi^{q^m} \in E$ and $\xi^{q^{m'}} \in E$, where m and m' are positive integers, along with $T_m(\alpha)$ and $T_{m'}(\alpha)$, for some $\alpha \in E$, show how to compute the values $\xi^{q^{m+m'}}$ and $T_{m+m'}(\alpha)$, using $O(\ell^{2.5})$ operations in F, and space for $O(\ell^{1.5})$ elements of F.

(b) Given as input $\xi^q \in E$, $\alpha \in E$, and a positive integer m, show how to

compute (using part (a)) the value $T_m(\alpha)$, using $O(\ell^{2.5}\operatorname{len}(m))$ operations in F, and space for $O(\ell^{1.5})$ elements of F.

(c) Repeat parts (a) and (b), except with "N" in place of "T."

EXERCISE 20.9. Using the result of the previous exercise, show how to implement Algorithm EDF so that it uses an expected number of

$$O(\operatorname{len}(k)\ell^{2.5} + \ell^2 \operatorname{len}(q))$$

operations in F, and space for $O(\ell^{1.5})$ elements of F.

EXERCISE 20.10. This exercise depends on the concepts and results in §18.6. Let E be an extension field of degree ℓ over F, specified by an irreducible polynomial of degree ℓ over F. Design and analyze an efficient probabilistic algorithm that finds a normal basis for E over F (see Exercise 19.14). Hint: there are a number of approaches to solving this problem; one way is to start by factoring $X^\ell - 1$ over F, and then turn the construction in Theorem 18.12 into an efficient probabilistic procedure; if you mimic Exercise 11.2, your entire algorithm should use $O(\ell^3 \operatorname{len}(\ell) \operatorname{len}(q))$ operations in F (or $O(\operatorname{len}(r)\ell^3 \operatorname{len}(q))$ operations, where r is the number of distinct irreducible factors of $X^\ell - 1$ over F).

20.5 Factoring polynomials: Berlekamp's algorithm

We now develop an alternative algorithm, due to Berlekamp, for factoring a polynomial over the finite field F into irreducibles. We shall assume that the input polynomial is square-free, using Algorithm SFD in §20.3 as a preprocessing step, if necessary.

Let us now assume we have a monic square-free polynomial $f \in F[X]$ of degree $\ell > 0$ that we want to factor into irreducibles. We first present the mathematical ideas underpinning the algorithm.

Let E be the F-algebra $F[X]/(f)$. Let σ be the Frobenius map on E over F, which maps $\alpha \in E$ to $\alpha^q \in E$. We know that σ is an F-algebra homomorphism (see Theorem 19.7). Consider the subalgebra B of E fixed by σ (see Theorem 16.6). Thus,

$$B = \{\alpha \in E : \alpha^q = \alpha\}.$$

The subalgebra B is called the **Berlekamp subalgebra of** E. Let us take a closer look at it. Suppose that f factors into irreducibles as

$$f = f_1 \cdots f_r,$$

and let

$$\begin{aligned} \theta : \quad & E \to E_1 \times \cdots \times E_r \\ & [g]_f \mapsto ([g]_{f_1}, \ldots, [g]_{f_r}) \end{aligned}$$

be the F-algebra isomorphism from the Chinese remainder theorem, where $E_i := F[X]/(f_i)$ is an extension field of F of finite degree for $i = 1, \ldots, r$. Now, for $\alpha \in E$, if $\theta(\alpha) = (\alpha_1, \ldots, \alpha_r)$, then we have $\alpha^q = \alpha$ if and only if $\alpha_i^q = \alpha_i$ for $i = 1, \ldots, r$; moreover, by Theorem 19.8, we know that for all $\alpha_i \in E_i$, we have $\alpha_i^q = \alpha_i$ if and only if $\alpha_i \in F$. Thus, we may characterize B as follows:

$$B = \{\theta^{-1}(c_1, \ldots, c_r) : c_1, \ldots, c_r \in F\}.$$

Since B is a subalgebra of E, then as F-vector spaces, B is a subspace of E. Of course, E has dimension ℓ over F, with the natural basis $\{\xi^{i-1}\}_{i=1}^{\ell}$, where $\xi := [X]_f$. As for the Berlekamp subalgebra, from the above characterization of B, it is evident that the elements

$$\theta^{-1}(1, 0, \ldots, 0),\ \theta^{-1}(0, 1, 0, \ldots, 0),\ \ldots,\ \theta^{-1}(0, \ldots, 0, 1)$$

form a basis for B over F, and hence, B has dimension r over F.

Now we come to the actual factoring algorithm.

Stage 1: Construct a basis for B

The first stage of Berlekamp's factoring algorithm constructs a basis for B over F. We can easily do this using Gaussian elimination, as follows. Let $\rho : E \to E$ be the map that sends $\alpha \in E$ to $\sigma(\alpha) - \alpha = \alpha^q - \alpha$. Since σ is an F-linear map, the map ρ is also F-linear. Moreover, the kernel of ρ is none other than the Berlekamp subalgebra B. So to find a basis for B, we simply need to find a basis for the kernel of ρ using Gaussian elimination over F, as in §14.4.

To perform the Gaussian elimination, we need to choose a basis S for E over F, and construct the matrix $Q := \mathrm{Mat}_{S,S}(\rho) \in F^{\ell \times \ell}$, that is, the matrix of ρ with respect to this basis, as in §14.2, so that evaluation of ρ corresponds to multiplying a row vector on the right by Q. We are free to choose a basis in any convenient way, and the most convenient basis, of course, is $S := \{\xi^{i-1}\}_{i=1}^{\ell}$, since for computational purposes, we already represent an element $\alpha \in E$ by its coordinate vector $\mathrm{Vec}_S(\alpha)$. The matrix Q, then, is the $\ell \times \ell$ matrix whose ith row, for $i = 1, \ldots, \ell$, is $\mathrm{Vec}_S(\rho(\xi^{i-1}))$. Note that if $\alpha = \xi^q$, then $\rho(\xi^{i-1}) = (\xi^{i-1})^q - \xi^{i-1} = (\xi^q)^{i-1} - \xi^{i-1} = \alpha^{i-1} - \xi^{i-1}$. This observation allows us to construct the rows of Q by first computing ξ^q via repeated squaring, and then just computing successive powers of ξ^q.

After we construct the matrix Q, we apply Gaussian elimination to get row vectors $v_1, \ldots, v_r$ that form a basis for the row null space of Q. It is at this point that

our algorithm actually discovers the number r of irreducible factors of f. Our basis for B is $\{\beta_i\}_{i=1}^r$, where $\mathrm{Vec}_S(\beta_i) = v_i$ for $i = 1, \ldots, r$.

Putting this all together, we have the following algorithm to compute a basis for the Berlekamp subalgebra.

Algorithm B1**.** On input f, where $f \in F[X]$ is a monic square-free polynomial of degree $\ell > 0$, do the following, where $E := F[X]/(f)$, $\xi := [X]_f \in E$, and $S := \{\xi^{i-1}\}_{i=1}^{\ell}$:

```
let Q be an ℓ × ℓ matrix over F (initially with undefined entries)
compute α ← ξ^q using repeated squaring
β ← 1_E
for i ← 1 to ℓ do  // invariant: β = α^{i−1} = (ξ^{i−1})^q
    Row_i(Q) ← Vec_S(β),  Q(i, i) ← Q(i, i) − 1,  β ← βα
compute a basis {v_i}_{i=1}^r of the row null space of Q using
    Gaussian elimination
for i = 1, ..., r do β_i ← Vec_S^{−1}(v_i)
output {β_i}_{i=1}^r
```

The correctness of Algorithm B1 is clear from the above discussion. As for the running time:

Theorem 20.10. *Algorithm* B1 *uses* $O(\ell^3 + \ell^2 \operatorname{len}(q))$ *operations in* F.

Proof. This is just a matter of counting. The computation of α takes $O(\operatorname{len}(q))$ operations in E using repeated squaring, and hence $O(\ell^2 \operatorname{len}(q))$ operations in F. To build the matrix Q, we have to perform an additional $O(\ell)$ operations in E to compute the successive powers of α, which translates into $O(\ell^3)$ operations in F. Finally, the cost of Gaussian elimination is an additional $O(\ell^3)$ operations in F. □

Stage 2: Splitting with a basis for B

The second stage of Berlekamp's factoring algorithm is a probabilistic procedure that factors f using a basis $\{\beta_i\}_{i=1}^r$ for B. As we did with Algorithm EDF in §20.4.2, we begin by discussing how to efficiently split f into two non-trivial factors, and then we present a somewhat more elaborate algorithm that completely factors f.

Let $M_1 \in F[X]$ be the polynomial defined by (20.4) and (20.5); that is,

$$M_1 := \begin{cases} \sum_{j=0}^{w-1} X^{2^j} & \text{if } p = 2, \\ X^{(q-1)/2} - 1 & \text{if } p > 2. \end{cases}$$

Using our basis for B, we can easily generate a random element β of B by simply

choosing $c_1, \ldots, c_r$ at random, and computing $\beta := \sum_i c_i \beta_i$. If $\theta(\beta) = (b_1, \ldots, b_r)$, then the family of random variables $\{b_i\}_{i=1}^r$ is mutually independent, with each b_i uniformly distributed over F. Just as in Algorithm EDF, $\gcd(\mathrm{rep}(M_1(\beta)), f)$ will be a non-trivial factor of f with probability at least $1/2$, if $p = 2$, and probability at least $4/9$, if $p > 2$.

That is the basic splitting strategy. We turn this into an algorithm to completely factor f using the same technique of iterative refinement that was used in Algorithm EDF. That is, at any stage of the algorithm, we have a partial factorization $f = \prod_{h \in H} h$, which we try to refine by attempting to split each $h \in H$ using the strategy outlined above. One technical difficulty is that to split such a polynomial h, we need to efficiently generate a random element of the Berlekamp subalgebra of $F[X]/(h)$. A particularly efficient way to do this is to use our basis for the Berlekamp subalgebra of $F[X]/(f)$ to generate a random element of the Berlekamp subalgebra of $F[X]/(h)$ for all $h \in H$ simultaneously. Let $g_i := \mathrm{rep}(\beta_i)$ for $i = 1, \ldots, r$. If we choose $c_1, \ldots, c_r \in F$ at random, and set $g := c_1 g_1 + \cdots + c_r g_r$, then $[g]_f$ is a random element of the Berlekamp subalgebra of $F[X]/(f)$, and by the Chinese remainder theorem, it follows that the family of random variables $\{[g]_h\}_{h \in H}$ is mutually independent, with each $[g]_h$ uniformly distributed over the Berlekamp subalgebra of $F[X]/(h)$.

Here is the algorithm for completely factoring a polynomial, given a basis for the corresponding Berlekamp subalgebra.

Algorithm B2**.** On input $f, \{\beta_i\}_{i=1}^r$, where $f \in F[X]$ is a monic square-free polynomial of degree $\ell > 0$, and $\{\beta_i\}_{i=1}^r$ is a basis for the Berlekamp subalgebra of $F[X]/(f)$, do the following, where $g_i := \mathrm{rep}(\beta_i)$ for $i = 1, \ldots, r$:

$H \leftarrow \{f\}$
while $|H| < r$ do
    choose $c_1, \ldots, c_r \in F$ at random
    $g \leftarrow c_1 g_1 + \cdots + c_r g_r \in F[X]$
    $H' \leftarrow \emptyset$
    for each $h \in H$ do
        $\beta \leftarrow [g]_h \in F[X]/(h)$
        $d \leftarrow \gcd(\mathrm{rep}(M_1(\beta)), h)$
        if $d = 1$ or $d = h$
            then $H' \leftarrow H' \cup \{h\}$
            else $H' \leftarrow H' \cup \{d, h/d\}$
    $H \leftarrow H'$
output H

The correctness of the algorithm is clear. As for its expected running time, we can get a quick-and-dirty upper bound as follows:

- The cost of generating g in each loop iteration is $O(r\ell)$ operations in F. For a given h, the cost of computing $\beta := [g]_h \in F[X]/(h)$ is $O(\ell \deg(h))$ operations in F, and the cost of computing $M_1(\beta)$ is $O(\deg(h)^2 \operatorname{len}(q))$ operations in F. Therefore, the number of operations in F performed in each iteration of the main loop is at most a constant times

$$r\ell + \ell \sum_{h \in H} \deg(h) + \operatorname{len}(q) \sum_{h \in H} \deg(h)^2$$
$$\leq 2\ell^2 + \operatorname{len}(q)\Big(\sum_{h \in H} \deg(h)\Big)^2 = O(\ell^2 \operatorname{len}(q)).$$

- The expected number of iterations of the main loop until we get some non-trivial split is $O(1)$.
- The algorithm finishes after getting $r - 1$ non-trivial splits.
- Therefore, the total expected cost is $O(r\ell^2 \operatorname{len}(q))$ operations in F.

A more careful analysis reveals:

Theorem 20.11. *Algorithm* B2 *uses an expected number of*

$$O(\operatorname{len}(r)\ell^2 \operatorname{len}(q))$$

operations in F.

Proof. The proof follows the same line of reasoning as the analysis of Algorithm EDF. Indeed, using the same argument as was used there, the expected number of iterations of the main loop is $O(\operatorname{len}(r))$. As discussed in the paragraph above this theorem, the cost per loop iteration is $O(\ell^2 \operatorname{len}(q))$ operations in F. The theorem follows. □

The bound in the above theorem is tight (see Exercise 20.11 below): unlike Algorithm EDF, we cannot make the multiplicative factor of $\operatorname{len}(r)$ go away.

Putting together Algorithms B1 and B2, we get Berlekamp's complete factoring algorithm. The running time bound is easily estimated from the results already proved:

Theorem 20.12. *Berlekamp's factoring algorithm uses an expected number of* $O(\ell^3 + \ell^2 \operatorname{len}(\ell) \operatorname{len}(q))$ *operations in* F.

We have assumed the input to Berlekamp's algorithm is a square-free polynomial. However, we may use Algorithm SFD as a preprocessing step to ensure that

this is the case. Even if we include the cost of this preprocessing step, the running time estimate in Theorem 20.12 remains valid.

So we see that Berlekamp's algorithm is faster than the Cantor–Zassenhaus algorithm, whose expected operation count is $O(\ell^3 \operatorname{len}(q))$. The speed advantage of Berlekamp's algorithm grows as q gets large. The one disadvantage of Berlekamp's algorithm is space: it requires space for $\Theta(\ell^2)$ elements of F, while the Cantor–Zassenhaus algorithm requires space for only $O(\ell)$ elements of F. One can in fact implement the Cantor–Zassenhaus algorithm so that it uses $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F, while using space for only $O(\ell^{1.5})$ elements of F — see Exercise 20.13 below.

EXERCISE 20.11. Give an example of a family of input polynomials that cause Algorithm B2 to use an expected number of at least $\Omega(\ell^2 \operatorname{len}(\ell) \operatorname{len}(q))$ operations in F. Assume that computing $M_1(\beta)$ for $\beta \in F[X]/(h)$ takes $\Omega(\deg(h)^2 \operatorname{len}(q))$ operations in F.

EXERCISE 20.12. Using the ideas behind Berlekamp's factoring algorithm, devise a deterministic irreducibility test that, given a monic polynomial of degree ℓ over F, uses $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F.

EXERCISE 20.13. This exercise develops a variant of the Cantor–Zassenhaus algorithm that uses $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F, while using space for only $O(\ell^{1.5})$ elements of F. By making use the variant of Algorithm EDF discussed in Exercise 20.9, our problem is reduced to that of implementing Algorithm DDF within the stated time and space bounds, assuming that the input polynomial is square-free.

(a) Show that for all non-negative integers i, j, with $i \neq j$, the irreducible polynomials in $F[X]$ that divide $X^{q^i} - X^{q^j}$ are precisely those whose degree divides $i - j$.

(b) Let $f \in F[X]$ be a monic polynomial of degree $\ell > 0$, and let $m = O(\ell^{1/2})$. Let $\xi := [X]_f \in E$, where $E := F[X]/(f)$. Show how to compute

$$\xi^q, \xi^{q^2}, \ldots, \xi^{q^{m-1}} \in E \text{ and } \xi^{q^m}, \xi^{q^{2m}}, \ldots, \xi^{q^{(m-1)m}} \in E$$

using $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F, and space for $O(\ell^{1.5})$ elements of F.

(c) Combine the results of parts (a) and (b) to implement Algorithm DDF on square-free inputs of degree ℓ, so that it uses $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F, and space for $O(\ell^{1.5})$ elements of F.

20.6 Deterministic factorization algorithms (∗)

The algorithms of Cantor and Zassenhaus and of Berlekamp are probabilistic. The exercises below develop a deterministic variant of the Cantor–Zassenhaus algorithm. (One can also develop deterministic variants of Berlekamp's algorithm, with similar complexity.)

This algorithm is only practical for finite fields of small characteristic, and is anyway mainly of theoretical interest, since from a practical perspective, there is nothing wrong with the above probabilistic method. In all of these exercises, we assume that we have access to a basis $\{\varepsilon_i\}_{i=1}^{w}$ for F as a vector space over $\mathbb{Z}_p$.

To make the Cantor–Zassenhaus algorithm deterministic, we only need to develop a deterministic variant of Algorithm EDF, as Algorithm DDF is already deterministic.

EXERCISE 20.14. Let $f = f_1 \cdots f_r$, where the f_i's are distinct monic irreducible polynomials in $F[X]$. Assume that $r > 1$, and let $\ell := \deg(f)$. For this exercise, the degrees of the f_i's need not be the same. For an intermediate field F', with $\mathbb{Z}_p \subseteq F' \subseteq F$, let us call a set $S = \{\lambda_1, \ldots, \lambda_s\}$, where each $\lambda_u \in F[X]$ with $\deg(\lambda_u) < \ell$, a **separating set for f over F'** if the following conditions hold:

- for $i = 1, \ldots, r$ and $u = 1, \ldots, s$, there exists $c_{ui} \in F'$ such that $\lambda_u \equiv c_{ui} \pmod{f_i}$, and
- for every pair of distinct indices i, j, with $1 \leq i < j \leq r$, there exists $u = 1, \ldots, s$ such that $c_{ui} \neq c_{uj}$.

Show that if S is a separating set for f over $\mathbb{Z}_p$, then the following algorithm completely factors f using $O(p|S|\ell^2)$ operations in F.

```
H ← {f}
for each λ ∈ S do
    for each a ∈ Z_p do
        H' ← ∅
        for each h ∈ H do
            d ← gcd(λ − a, h)
            if d = 1 or d = h
                then H' ← H' ∪ {h}
                else H' ← H' ∪ {d, h/d}
        H ← H'
output H
```

EXERCISE 20.15. Let f be as in the previous exercise. Show that if S is a

separating set for f over F, then the set

$$S' := \Big\{ \sum_{i=0}^{w-1} (\varepsilon_j \lambda)^{p^i} \bmod f : 1 \le j \le w,\ \lambda \in S \Big\}$$

is a separating set for f over $\mathbb{Z}_p$. Show how to compute this set using $O(|S|\ell^2 \operatorname{len}(p) w(w-1))$ operations in F.

EXERCISE 20.16. Let f be as in the previous two exercises, but further suppose that each irreducible factor of f is of the same degree, say k. Let $E := F[X]/(f)$ and $\xi := [X]_f \in E$. Define the polynomial $\phi \in E[Y]$ as follows:

$$\phi := \prod_{i=0}^{k-1} (Y - \xi^{q^i}).$$

If

$$\phi = Y^k + \alpha_{k-1} Y^{k-1} + \cdots + \alpha_0,$$

with $\alpha_0, \ldots, \alpha_{k-1} \in E$, show that the set

$$S := \{\operatorname{rep}(\alpha_i) : 0 \le i \le k-1\}$$

is a separating set for f over F, and can be computed deterministically using $O(k^2 + k \operatorname{len}(q))$ operations in E, and hence $O(k^2\ell^2 + k\ell^2 \operatorname{len}(q))$ operations in F.

EXERCISE 20.17. Put together all of the above pieces, together with Algorithms SFD and DDF, so as to obtain a deterministic algorithm for factoring polynomials over F that runs in time at most p times a polynomial in the size of the input, and make a careful estimate of the running time of your algorithm.

EXERCISE 20.18. It is a fact that when our prime p is odd, then for all integers a, b, with $a \not\equiv b \pmod{p}$, there exists a non-negative integer $i \le p^{1/2} \log_2 p$ such that $(a + i \mid p) \neq (b + i \mid p)$ (here, "$(\cdot \mid \cdot)$" is the Legendre symbol). Using this fact, design and analyze a deterministic algorithm for factoring polynomials over F that runs in time at most $p^{1/2}$ times a polynomial in the size of the input.

The following two exercises show that the problem of factoring polynomials over F reduces in deterministic polynomial time to the problem of finding roots of polynomials over $\mathbb{Z}_p$.

EXERCISE 20.19. Let f be as in Exercise 20.14. Suppose that $S = \{\lambda_1, \ldots, \lambda_s\}$ is a separating set for f over $\mathbb{Z}_p$, and $\phi_u \in F[X]$ is the minimal polynomial over F of $[\lambda_u]_f \in F[X]/(f)$ for $u = 1, \ldots, s$. Show that each ϕ_u is the product of linear factors over $\mathbb{Z}_p$, and that given S, along with the roots of all the ϕ_u's, we can deterministically factor f using $(|S| + \ell)^{O(1)}$ operations in F. Hint: see Exercise 16.9.

EXERCISE 20.20. Using the previous exercise, show that the problem of factoring a polynomial over F reduces in deterministic polynomial time to the problem of finding roots of polynomials over $\mathbb{Z}_p$.

20.7 Notes

The average-case analysis of Algorithm IPT, assuming its input is random, and the application to the analysis of Algorithm RIP, is essentially due to Ben-Or [14]. If one implements Algorithm RIP using fast polynomial arithmetic, one gets an expected cost of $O(\ell^{2+o(1)}\operatorname{len}(q))$ operations in F. Note that Ben-Or's analysis is a bit incomplete — see Exercise 32 in Chapter 7 of Bach and Shallit [11] for a complete analysis of Ben-Or's claims.

The asymptotically fastest probabilistic algorithm for constructing an irreducible polynomial over F of given degree ℓ is due to Shoup [96]. That algorithm uses an expected number of $O(\ell^{2+o(1)} + \ell^{1+o(1)}\operatorname{len}(q))$ operations in F, and in fact does not follow the "generate and test" paradigm of Algorithm RIP, but uses a completely different approach.

As far as *deterministic* algorithms for constructing irreducible polynomials of given degree over F, the only known methods are efficient when the characteristic p of F is small (see Chistov [26], Semaev [88], and Shoup [94]), or under a generalization of the Riemann hypothesis (see Adleman and Lenstra [4]). Shoup [94] in fact shows that the problem of constructing an irreducible polynomial of given degree over F is deterministic, polynomial-time reducible to the problem of factoring polynomials over F.

The algorithm in §20.2 for computing minimal polynomials over finite fields is due to Gordon [43].

The square-free decomposition of a polynomial over a field of characteristic zero can be computed using an algorithm of Yun [111] using $O(\ell^{1+o(1)})$ field operations. Yun's algorithm can be adapted to work over finite fields as well (see Exercise 14.30 in von zur Gathen and Gerhard [39]).

The Cantor–Zassenhaus algorithm was initially developed by Cantor and Zassenhaus [24], although many of the basic ideas can be traced back quite a ways. A straightforward implementation of this algorithm using fast polynomial arithmetic uses an expected number of $O(\ell^{2+o(1)}\operatorname{len}(q))$ operations in F.

Berlekamp's algorithm was initially developed by Berlekamp [15, 16], but again, the basic ideas go back a long way. A straightforward implementation using fast polynomial arithmetic uses an expected number of $O(\ell^3 + \ell^{1+o(1)}\operatorname{len}(q))$ operations in F; the term ℓ^3 may be replaced by ℓ^ω, where ω is the exponent of matrix multiplication (see §14.6).

There are no known efficient, deterministic algorithms for factoring polynomials

over F when the characteristic p of F is large (even under a generalization of the Riemann hypothesis, except in certain special cases).

The asymptotically fastest algorithms for factoring polynomials over F are due to von zur Gathen, Kaltofen, and Shoup:† the algorithm of von zur Gathen and Shoup [40] uses an expected number of $O(\ell^{2+o(1)} + \ell^{1+o(1)} \operatorname{len}(q))$ operations in F; the algorithm of Kaltofen and Shoup [53] has a cost that is subquadratic in the degree—it uses an expected number of $O(\ell^{1.815} \operatorname{len}(q)^{0.407})$ operations in F when $\operatorname{len}(q) = O(\ell^{1.375})$. Exercises 20.1, 20.8, and 20.9 are based on [40]. Although the "fast" algorithms in [40] and [53] are mainly of theoretical interest, a variant in [53], which uses $O(\ell^{2.5} + \ell^{1+o(1)} \operatorname{len}(q))$ operations in F, and space for $O(\ell^{1.5})$ elements of F, has proven to be quite practical (Exercise 20.13 develops some of these ideas; see also Shoup [97]).

† The running times of these algorithms can be improved using faster algorithms for modular composition—see footnote on p. 485.