Math 332 Winter 2023, Lecture 26: Polynomials

website: `https://www.cip.ifi.lmu.de/~grinberg/t/23wa`

3. Monoid algebras and polynomials

Recall: For this entire chapter, we fix a **commutative** ring R.

3.3. Univariate polynomials (cont'd)

3.3.3. Roots

We shall now discuss roots of polynomials. Our definition of roots is rather liberal:

Definition 3.3.10. Let A be an R-algebra. Let $f \in R[x]$ be a polynomial. An element $a \in A$ is called a **root** of f if $f(a) = 0$ (that is, $f[a] = 0$).

This allows, e.g., a matrix or an element of a quotient ring to be a root of a polynomial. For example, the matrix $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \in \mathbb{Q}^{2\times 2}$ is a root of the polynomial $x^2 - 1 \in \mathbb{Q}[x]$, since

$$\left(x^2-1\right)\left[\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\right] = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}^2 - \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}_{\substack{\text{this is the unity} \\ \text{of the ring } \mathbb{Q}^{2\times 2}}} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = 0_{\mathbb{Q}^{2\times 2}}.$$

For another example, the polynomial $x^2 + 1 \in \mathbb{Z}[x]$ has infinitely many roots in the ring $\mathbb{H}$ of quaternions. (Indeed, it is not hard to check that $(ai + bj + ck)^2 = -\left(a^2 + b^2 + c^2\right)$ in $\mathbb{H}$ for any $a, b, c \in \mathbb{R}$. But there are infinitely many triples $(a, b, c) \in \mathbb{R}^3$ satisfying $a^2 + b^2 + c^2 = 1$.)

Let us first say a few things about roots in R:

Proposition 3.3.11. Let f be a polynomial in $R[x]$. Let $a \in R$. Then, a is a root of f if and only if $x - a \mid f$ in $R[x]$.

Proof. See §4.3.3 in the text. □

The following theorem is often known as the **easy half of the FTA (Fundamental Theorem of Algebra)**:

Theorem 3.3.12 (easy half of the FTA)**.** Let R be an integral domain. Let $n \in \mathbb{N}$. Then, any nonzero polynomial $f \in R[x]$ of degree $\leq n$ has at most n roots in R. (We are not counting the roots with multiplicity here.)

Proof. See §4.3.3 in the text. □

Theorem 3.3.12 can fail if R is not an integral domain. For example, the polynomial $x^2 - \overline{1}$ has 4 roots in $\mathbb{Z}/8$. (Find them!)

Remark 3.3.13. The famous **fundamental theorem of algebra** (short: **FTA**) says that any polynomial of degree n in $\mathbb{C}[x]$ has exactly n roots in $\mathbb{C}$, if we count the roots with multiplicity. Despite its name, this theorem is not actually a theorem of algebra, as it relies on the analytic structure of the complex numbers (and the underlying real numbers), and does not hold (e.g.) for the Gaussian rationals $\mathbb{Q}[i]$. Accordingly, each proof of the FTA requires at least a little bit of real analysis (and sometimes a not-so-little bit). Various proofs can be found in [LaNaSc16, Chapter 3], [Aluffi16, Theorem 7.1], [Knapp16, Chapter IX, §10], [Warner90, Theorem 44.8], [Steinb06, Theorem 11.6.7] and many other places (some of which prove weaker-sounding but equivalent versions of the result); more exotic proofs are listed in `https://mathoverflow.net/questions/10535` .

However, one "half" of the FTA – namely, the claim that a polynomial of degree n in $\mathbb{C}[x]$ always has **at most** n roots in $\mathbb{C}$ – actually can be proved algebraically, and holds not just for $\mathbb{C}$ but also for any integral domain R in its stead. If we drop the notion of multiplicities, then this "half" is precisely Theorem 3.3.12. Thus, Theorem 3.3.12 is called the "easy half of the FTA".

As far as abstract algebra is concerned, this easy half is by far the most useful; the next subsections will attest to some of its uses. In comparison, the "hard half of the FTA" (the part that really requires $\mathbb{C}$) is rarely used in abstract algebra (since algebraists prefer to work in settings more general than $\mathbb{C}$), but it is important (e.g.) in complex linear algebra, where it is responsible (e.g.) for the fact that each $n \times n$-matrix over $\mathbb{C}$ has n eigenvalues (counted with multiplicities). Thus, the name "FTA" should be regarded as somewhat of a historical artefact.

3.3.4. Application to $\mathbb{Z}/p$: Wilson revisited

The easy half of the FTA has a surprising number of applications, some fairly unexpected. We begin with one of the simplest.

Let p be a prime number. Consider the polynomial

$$x^p - x \in (\mathbb{Z}/p)[x].$$

This polynomial has degree p, and has p roots in $\mathbb{Z}/p$: Indeed, any element $a \in \mathbb{Z}/p$ satisfies $a^p = a$ (by Fermat's Little Theorem[1]) and thus is a root of the polynomial $x^p - x$. No surprises here; just one example of Theorem 3.3.12 being right.

[1] Proposition 2.6.4 in the text

However, we can do something more interesting. Namely, consider the more sophisticated polynomial

$$\begin{aligned} f &:= (x^p - x) - \underbrace{\prod_{u \in \mathbb{Z}/p} (x-u)}_{= \left(x-\overline{0}\right)\left(x-\overline{1}\right)\cdots\left(x-\overline{p-1}\right)} \\ &= (x^p - x) - \left(x-\overline{0}\right)\left(x-\overline{1}\right)\cdots\left(x-\overline{p-1}\right) \in (\mathbb{Z}/p)[x]. \end{aligned}$$

This polynomial f again has p roots in $\mathbb{Z}/p$ (since every $a \in \mathbb{Z}/p$ satisfies[2]

$$f[a] = \underbrace{(a^p - a)}_{\substack{=0 \\ \text{(by Fermat's Little Theorem)}}} - \underbrace{\left(a-\overline{0}\right)\left(a-\overline{1}\right)\cdots\left(a-\overline{p-1}\right)}_{\substack{=0 \\ \text{(since } a-a \text{ is a factor of this product)}}} = 0 - 0 = 0,$$

thus is a root of f), but its degree is $\leq p-1$ (since the polynomials $x^p - x$ and $\prod_{u \in \mathbb{Z}/p} (x-u)$ both have leading term x^p, and thus their leading terms cancel out when you take the difference). This contradicts the easy half of the FTA, unless f is zero. So f must be 0. In other words,

$$x^p - x = \prod_{u \in \mathbb{Z}/p} (x-u).$$

So we have proved the following:

Proposition 3.3.14. Let p be a prime number. Then,

$$x^p - x = \prod_{u \in \mathbb{Z}/p} (x-u) \qquad \text{in the polynomial ring } (\mathbb{Z}/p)[x].$$

We can derive more concrete conclusions from this equality by comparing coefficients. For example, comparing the coefficients of x^{p-1} on both sides, we obtain (for $p > 2$) that

$$0 = -\left(\overline{0} + \overline{1} + \cdots + \overline{p-1}\right)$$

(since the coefficient of x^{p-1} in $x^p - x$ is 0 when $p > 2$, whereas the coefficient of x^{p-1} in $\prod_{u \in \mathbb{Z}/p} (x-u)$ is $-\left(\overline{0} + \overline{1} + \cdots + \overline{p-1}\right)$ because of the way how x^{p-1} terms are generated when the product $\prod_{u \in \mathbb{Z}/p} (x-u)$ is expanded). This can be rewritten as

$$\overline{0} + \overline{1} + \cdots + \overline{p-1} = 0,$$

[2] We are tacitly using Theorem 3.2.7 from Lecture 25 here. Indeed, thanks to this theorem, we know that substituting a for x in the product $\left(x-\overline{0}\right)\left(x-\overline{1}\right)\cdots\left(x-\overline{p-1}\right)$ yields the product $\left(a-\overline{0}\right)\left(a-\overline{1}\right)\cdots\left(a-\overline{p-1}\right)$.

i.e., as

$$0+1+\cdots+(p-1) \equiv 0 \bmod p.$$

This is a nice congruence, although it can be easily proved directly as well (since $0+1+\cdots+(p-1) = \dfrac{(p-1)\,p}{2} = \underbrace{\dfrac{p-1}{2}}_{\substack{\in\mathbb{Z} \\ \text{when } p>2}} \cdot p \equiv 0 \bmod p$ for $p > 2$).

But comparing other coefficients results in less obvious consequences. If two polynomials f and g are equal, then each monomial has equal coefficients in them (i.e., each $n \in \mathbb{N}$ satisfies $[x^n] f = [x^n] g$), and thus a single equality between two polynomials gives rise to an infinite sequence of equalities between coefficients (although only finitely many of them are nontrivial).

There is also another way to obtain consequences from Proposition 3.3.14: Namely, we can rewrite the claim of Proposition 3.3.14 as follows:

$$\begin{aligned} x^p - x &= \prod_{u \in \mathbb{Z}/p} (x-u) = \underbrace{\left(x-\overline{0}\right)}_{=x} \left(x-\overline{1}\right) \cdots \left(x-\overline{p-1}\right) \\ &= x\left(x-\overline{1}\right)\left(x-\overline{2}\right)\cdots\left(x-\overline{p-1}\right). \end{aligned}$$

Dividing both sides of this equality by x (with remainder if you will, but the remainder will be 0), we obtain

$$x^{p-1} - \overline{1} = \left(x-\overline{1}\right)\left(x-\overline{2}\right)\cdots\left(x-\overline{p-1}\right).$$

Now, substituting $\overline{0}$ for x in this equality, we obtain

$$\begin{aligned} \overline{0}^{p-1} - \overline{1} &= \left(\overline{0}-\overline{1}\right)\left(\overline{0}-\overline{2}\right)\cdots\left(\overline{0}-\overline{p-1}\right) = \left(-\overline{1}\right)\left(-\overline{2}\right)\cdots\left(-\overline{p-1}\right) \\ &= (-1)^{p-1}\,\overline{1\cdot 2\cdot\cdots\cdot(p-1)} = (-1)^{p-1}\,\overline{(p-1)!}. \end{aligned}$$

Thus,

$$(-1)^{p-1}\,\overline{(p-1)!} = \underbrace{\overline{0}^{p-1}}_{\substack{=\overline{0} \\ \text{(since } p>1\text{)}}} - \overline{1} = \overline{-1}.$$

Therefore,

$$\overline{(p-1)!} = (-1)^{p-1} \cdot \overline{-1} = \overline{(-1)^p} \qquad \text{in } \mathbb{Z}/p.$$

Back in $\mathbb{Z}$, this means that

$$(p-1)! \equiv (-1)^p \bmod p.$$

Since it is easy to see that $(-1)^p \equiv -1 \bmod p$ (by Fermat's Little Theorem, or by separating the cases of p even and p odd), we can simplify this to

$$(p-1)! \equiv -1 \bmod p.$$

This is exactly Wilson's theorem that we proved a while ago (Theorem 1.15.2 in Lecture 17). Thus, we have found a new proof of Wilson's theorem.

Here is yet another unexpected application of the easy half of the FTA:

Proposition 3.3.15. Let p be a prime. Let $k \in \{0, 1, \ldots, p-2\}$. Then, the sum

$$0^k + 1^k + \cdots + (p-1)^k = \sum_{j=0}^{p-1} j^k$$

is divisible by p.

Proof. There is a beautiful proof of this, but the lecture is too short. See Proposition 4.3.20 in §4.3.5 in the text. □

3.3.5. $F[x]$ is a Euclidean domain

The division-with-remainder theorem for polynomials (Theorem 3.3.8 **(a)** in Lecture 25, to be specific) looks suspiciously like the definition of a Euclidean norm in the definition of a Euclidean ring (Definition 1.13.2 **(b)** in Lecture 15). Unfortunately, the condition "the leading coefficient of b is a unit" in the former theorem prevents it from literally fitting the bill. However, when the ring R is a field, every leading coefficient is a unit, and so this condition is no longer standing in our way. Thus, we obtain the following:

Theorem 3.3.16. Let F be a field. Then, the polynomial ring $F[x]$ is a Euclidean domain. The map

$$N : F[x] \to \mathbb{N},$$
$$p \mapsto \max\{\deg p, 0\} = \begin{cases} \deg p, & \text{if } p \neq 0; \\ 0, & \text{if } p = 0 \end{cases}$$

is a Euclidean norm on $F[x]$.

Proof. Follows from the division-with-remainder theorem for polynomials, as we just explained. □

Theorem 3.3.16 allows us to apply the whole machinery of Euclidean domains to $F[x]$. In particular, it yields that polynomials in $F[x]$ (where F is a field) have gcds and lcms, and the Euclidean algorithm can be used to compute them, and Bezout's theorem allows a gcd of two polynomials a and b to be written as $ua + vb$ where $u, v \in F[x]$. Also, it entails that $F[x]$ is a PID (since any Euclidean domain is a PID). See §4.3.6 in the text for details.

3.3.6. Lagrange interpolation

The easy half of the FTA has yet another consequence:

Corollary 3.3.17 (uniqueness of the interpolating polynomial)**.** Let R be an integral domain. Let $a_0, a_1, \ldots, a_n$ be $n+1$ distinct elements of R. Let $f, g \in R[x]$ be two polynomials of degree $\leq n$. Assume that

$$f[a_i] = g[a_i] \qquad \text{for all } i \in \{0, 1, \ldots, n\}.$$

Then, $f = g$.

Proof. The difference $f - g$ is a polynomial of degree $\leq n$, but has at least $n+1$ roots (namely, $a_0, a_1, \ldots, a_n$). This contradicts the easy half of the FTA unless $f - g = 0$. Thus, we must have $f - g = 0$; but this means that $f = g$. □

Corollary 3.3.17 says that a polynomial of degree $\leq n$ over an integral domain is uniquely determined by its values at $n+1$ distinct elements $a_0, a_1, \ldots, a_n$ of R. So, if you specify the values $f[a_0]$, $f[a_1]$, $f[a_2]$, $\ldots$, $f[a_n]$ of a polynomial f of degree $\leq n$ at $n+1$ given distinct elements $a_0, a_1, \ldots, a_n$ of R, the polynomial f is uniquely determined. But does such a polynomial f always exist (for any choice of these values)?

If R is a field, the answer is "yes":

Theorem 3.3.18 (Lagrange interpolation)**.** Let F be a field. Let $n \in \mathbb{N}$.

Let $a_0, a_1, \ldots, a_n$ be $n+1$ distinct elements of F. Let $b_0, b_1, \ldots, b_n$ be $n+1$ elements of F. Then:

(a) There is a **unique** polynomial $p \in F[x]$ satisfying $\deg p \leq n$ and

$$p[a_i] = b_i \qquad \text{for all } i \in \{0, 1, \ldots, n\}. \tag{1}$$

(b) This polynomial p is explicitly given by

$$p = \sum_{j=0}^{n} b_j \frac{\prod\limits_{k \neq j} (x - a_k)}{\prod\limits_{k \neq j} (a_j - a_k)}. \tag{2}$$

Proof. See Theorem 4.3.26 in the text for a detailed proof.

In a nutshell: If p is the polynomial defined by (2), then an easy computation shows that p does satisfy $\deg p \leq n$ and (1) (this is a computation worth making at least once in your life; it greatly demystifies the right hand side of (2)). Thus, the existence part of part **(a)** follows. The uniqueness follows easily from Corollary 3.3.17, and thus part **(b)** follows as well. □

Theorem 3.3.18 allows us to construct (or reconstruct) a polynomial of degree $\leq n$ over a field F from knowing $n+1$ of its values (at distinct inputs). This is called **Lagrange interpolation**.

This kind of interpolation is particularly useful when F is a finite field such as $\mathbb{Z}/p$ for a prime p. Here are two applications:

- **Shamir's Secret Sharing Scheme:** Assume that you have a secret **a** (a piece of information, encoded e.g. as a bitstring[3]), which you want to distribute among n people ("keepers"), by giving each keeper a "piece" of the secret (e.g., another bitstring), in such a way that
  - any k keepers (working together) can piece together the secret **a** from their "pieces",
  - but any $k-1$ keepers are left completely clueless about **a** (that is, they cannot infer anything about **a** from their "pieces").

  How would you do that? Shamir's Secret Sharing Scheme does it as follows:

  Fix a prime p such that $p > n$ and $p > 2^N$, where N is the size of **a** (in bits).

  Label the n keepers $1, 2, \ldots, n$.

  Encode the secret **a** as a residue class $\alpha \in \mathbb{Z}/p$. (This can be done, since $p > 2^N$. Of course, you have to agree with the keepers on an encoding; this encoding needs not be kept secret.)

  Pick $k-1$ uniformly random elements $\beta_1, \beta_2, \ldots, \beta_{k-1}$ of $\mathbb{Z}/p$. (This requires a good random number generator, but such things can be assumed to exist.)

  Let f be the polynomial

  $$\beta_{k-1}x^{k-1} + \beta_{k-2}x^{k-2} + \cdots + \beta_1 x + \alpha \in (\mathbb{Z}/p)[x]$$

  (which has degree $\leq k-1$).

  Give one value of f to each keeper. Specifically, give the value $f\left[\overline{i}\right] \in \mathbb{Z}/p$ to keeper i. Then, throw the polynomial f away (and certainly do not share it with the keepers!).

  Hence, any k keepers (working together) know k different values of f, and thus can use Lagrange interpolation (specifically, the explicit formula (2)) to piece together the polynomial f from these values. Hence, they can compute α (as the constant term of f) and thus recover the original secret **a**.

  But $k-1$ keepers $i_1, i_2, \ldots, i_{k-1}$ cannot infer anything about α from their values $f\left[\overline{i_1}\right]$, $f\left[\overline{i_2}\right]$, $\ldots$, $f\left[\overline{i_{k-1}}\right]$. (This is not completely obvious. The easiest way to see this is as follows: The element α is the constant term of f, so that $\alpha = f\left[\overline{0}\right]$. For any $\gamma \in \mathbb{Z}/p$, Theorem 3.3.18 **(a)** yields a unique polynomial g_γ of degree $\leq k-1$ that takes this value γ at $\overline{0}$ while taking the values $f\left[\overline{i_1}\right]$, $f\left[\overline{i_2}\right]$, $\ldots$, $f\left[\overline{i_{k-1}}\right]$ at $\overline{i_1}, \overline{i_2}, \ldots, \overline{i_{k-1}}$. From the viewpoint

[3] A **bitstring** is a tuple of bits (i.e., elements of $\{0,1\}$). This is how information is usually stored on digital media.

of our $k-1$ keepers $i_1, i_2, \ldots, i_{k-1}$, the polynomial f can be any of these p polynomials $g_{\overline{0}}, g_{\overline{1}}, \ldots, g_{\overline{p-1}}$, and none of them is more likely than any other (if $\beta_1, \beta_2, \ldots, \beta_{k-1}$ were really chosen uniformly at random). Thus, α can be any of the values $\overline{0}, \overline{1}, \ldots, \overline{p-1}$ with equal probabilities. I.e., our $k-1$ keepers know nothing about α.)

See [Smart11, §23.5] for more about this secret-sharing scheme. Note that we can use any finite field F instead of $\mathbb{Z}/p$, but in that case we may have to number our n keepers with some of the nonzero elements of F (as opposed to $1, 2, \ldots, n$).

- **Error-correcting codes:** Coding theory is the study of how to encode information in such a way that minor errors (i.e., changes to just a few bits) can be corrected or at least detected. This has myriad applications in communication (e.g., sending digital data over radio or fiber) and data storage (e.g., hard drives and RAM). Texts written about coding theory easily fill bookshelves, and many universities have courses dedicated to it. Much (but not all) of coding theory relies on finite fields (and occasionally more general rings). Here is just a little taste of the subject:

  Imagine that you want to send a message to a recipient via messenger pigeons (more realistically, IP packets). Each pigeon can carry an element of $\mathbb{Z}/p$ for a given prime p (more realistically, a bitstring of a given size, but this can easily be reduced to $\mathbb{Z}/p$ by picking an appropriate prime p). Your message is a tuple of n elements of $\mathbb{Z}/p$, thus would fit on n of these pigeons. However, you expect that a few pigeons will be lost on the way, so you have to build some redundancy into your communication. (We assume that the pigeons are numbered $1, 2, 3, \ldots$, so that the recipient will know what he is missing.)

  The easiest way to build redundancy into the system is to send each message twice, thus using $2n$ instead of n pigeons. This can "correct 1 error", i.e., your message will still be received if 1 pigeon goes missing. Likewise, with $3n$ pigeons, you can afford losing 2 pigeons. But this is clearly wasteful (and more pigeons will likely lead to more loss). Can you do better?

  A huge improvement can be obtained using "check-sums": If your message is the tuple $(a_1, a_2, \ldots, a_n)$, then you can send n pigeons carrying the elements $a_1, a_2, \ldots, a_n$, respectively, and an extra "checksum" pigeon carrying the element $a_1 + a_2 + \cdots + a_n \in \mathbb{Z}/p$. If at most 1 pigeon is lost, your recipient will be able to infer the missing element from the others, so you end up correcting 1 error at the expense of merely 1 extra pigeon.

  Similarly, you can correct 2 errors by sending two "checksum" pigeons, one carrying $a_1 + a_2 + \cdots + a_n$ and another carrying $1a_1 + 2a_2 + \cdots + na_n$ for example. Things get more complicated for more errors, however.

The situation becomes even more interesting if we allow not just for lost pigeons, but for pigeons that corrupt (i.e., change) their messages (arguably not a very likely problem with pigeons, but a pretty common one with data over the internet or on digital storage). The single checksum $a_1 + a_2 + \cdots + a_n$ will discover the corruption of a single pigeon, although it will not help correct it (as it is not clear which of the $n+1$ pigeons messed up).

However, polynomials and Lagrange interpolation save our day. We encode our intended message into a polynomial

$$f := a_1 x^0 + a_2 x^1 + a_3 x^2 + \cdots + a_n x^{n-1} \in (\mathbb{Z}/p)[x],$$

and give each pigeon a value of this polynomial (e.g., giving the i-th pigeon the value $f\left(\overline{i}\right)$). Now, any n pigeons are sufficient to reconstruct the polynomial f (and thus the original message) using Lagrange interpolation. Thus, if we send $n+k$ pigeons on their way, then k missing pigeons will not destroy the message. Better yet, using $n+2k$ pigeons, the recipient will be able to reconstruct the message correctly even if k (or fewer) pigeons corrupt their data. (Theoretically, this can be done by applying Lagrange interpolation to every possible $(n+k)$-element set of pigeons, yielding a polynomial of degree $\leq n+k-1$ each time. If this polynomial has degree $\geq n$, then it cannot be f. Using Corollary 3.3.17, it is not hard to see that only one polynomial of degree $\leq n-1$ will come out, and that will be f, provided that no more than k pigeons have corrupted their data or gotten lost. Finding an actually efficient algorithm is trickier.)

This is called a **Reed–Solomon code** (going back to Reed and Solomon in 1960, [ReeSol60]), and (just like secret sharing) it can be used with any finite field instead of $\mathbb{Z}/p$ (and some other choices are more popular for technical reasons).

We have just touched the surface of coding theory here (and even of just the theory of Reed–Solomon codes). See [MulMum07, Chapter 3] for an introduction to error-correcting codes, and [Garret03] for a more thorough textbook.

References

[Aluffi16] Paolo Aluffi, *Algebra: Chapter 0*, Graduate Studies in Mathematics **104**, 2nd printing, AMS 2016.

[Garret03] Paul Garrett, *The Mathematics of Coding Theory*, Prentice Hall 2003. `https://www-users.cse.umn.edu/~garrett/coding/CodingNotes.pdf`

[Knapp16] Anthony W. Knapp, *Basic Algebra*, Digital 2nd edition 2016.
`http://www.math.stonybrook.edu/~aknapp/download.html`

[LaNaSc16] Isaiah Lankham, Bruno Nachtergaele, Anne Schilling, *Linear Algebra As an Introduction to Abstract Mathematics*, 2016.
`https://www.math.ucdavis.edu/~anne/linear_algebra/mat67_course_notes.pdf`

[MulMum07] Gary L. Mullen, Carl Mummert, *Finite Fields and Applications*, Student Mathematical Library **41**, AMS 2007.

[ReeSol60] I. S. Reed, G. Solomon, *Polynomial codes over certain finite fields*, J Soc. Indust. Appl. Math. **8**, No. 2, June, 1960.

[Smart11] Nigel Smart, *Cryptography: An Introduction*, 3rd Edition 2011.
`https://www.cs.umd.edu/~waa/414-F11/IntroToCrypto.pdf` or `https://homes.esat.kuleuven.be/~nsmart/Crypto_Book/` (with errata).

[Steinb06] Mark Steinberger, *Algebra*, 31 August 2006.
`https://math.hawaii.edu/~tom/algebra.pdf`

[Warner90] Seth Warner, *Modern Algebra: two volumes bound as one*, Dover 1990.