Certifying Rings of Integers in Number Fields

Anne Baanen*
anne@anne.mx
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
Lean FRO, USA

Alain Chavarri Villarello
a.chavarri.villarello@vu.nl
Vrije Universiteit Amsterdam
Amsterdam, Netherlands

Sander R. Dahmen
s.r.dahmen@vu.nl
Vrije Universiteit Amsterdam
Amsterdam, Netherlands

Abstract

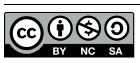
Number fields and their rings of integers, which generalize the rational numbers and the integers, are foundational objects in number theory. There are several computer algebra systems and databases concerned with the computational aspects of these. In particular, computing the ring of integers of a given number field is one of the main tasks of computational algebraic number theory. In this paper, we describe a formalization in Lean 4 for certifying such computations. In order to accomplish this, we developed several data types amenable to computation. Moreover, many other underlying mathematical concepts and results had to be formalized, most of which are also of independent interest. These include resultants and discriminants, as well as methods for proving irreducibility of univariate polynomials over finite fields and over the rational numbers. To illustrate the feasibility of our strategy, we formally verified entries from the *Number fields* section of the *L-functions and modular forms database* (LMFDB). These concern, for several number fields, the explicitly given *integral basis* of the ring of integers and the *discriminant*. To accomplish this, we wrote SageMath code that computes the corresponding certificates and outputs a Lean proof of the statement to be verified.

CCS Concepts: • **Security and privacy → Logic and verification**; • **Mathematics of computing → Mathematical software**.

Keywords: formalized mathematics, algebraic number theory, tactics, Lean, Mathlib

ACM Reference Format:
Anne Baanen, Alain Chavarri Villarello, and Sander R. Dahmen. 2025. Certifying Rings of Integers in Number Fields. In *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '25), January 20–21, 2025, Denver, CO, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3703595.3705874

*Authors listed in alphabetical order



CPP '25, January 20–21, 2025, Denver, CO, USA

ACM ISBN 979-8-4007-1347-7/25/01
https://doi.org/10.1145/3703595.3705874

1 Introduction

There are many fundamental concepts in mathematics that are, in principle, amenable to computation. Focusing on number theory, and more particularly algebraic number theory (i.e. the theory of algebraic numbers), such concepts include the *rings of integers* of a *number field*. Number fields generalize the field of rational numbers $\mathbf{Q}$ (in the sense that they are finite degree field extensions thereof) and each contains a ring of integers, which can be seen as generalizing how the (ordinary) integers $\mathbf{Z}$ are contained in the number field $\mathbf{Q}$; see Section 2 for more details. Rings of integers are thereby key to the arithmetic properties of their number fields, and both are essential to number theory from the 19th century to the modern day. Their basic definitions and properties (amongst other concepts) were formalized in [6], on which we build. From a computational direction, number fields of degree 2 and their rings of integers were considered in [5]. This paper can be seen as a far reaching generalization of such formally verified computations. In order to work with rings of integers inside a proof assistant —Lean 4 in our case— we opted for a *certification* approach. Given a ring of integers, concretely represented by giving an *integral basis* (i.e. a $\mathbf{Z}$-basis) for it, as e.g. computed by an external Computer Algebra System (CAS) or extracted from some database, we let SageMath [42] compute a certificate which is checked in Lean to certify the correctness of the ring of integers. In fact, our SageMath code outputs a complete proof which Lean can readily check.

The isomorphism class of a number field can be represented explicitly, for example by a polynomial with integer coefficients and leading coefficient 1 that is irreducible over $\mathbf{Z}$ (and hence $\mathbf{Q}$). This is also a basic way to define number fields in Computer Algebra Systems such as PARI/GP [41], SageMath [42], and Magma [7] (perhaps relaxing to rational coefficients). Given the prime factorization of the discriminant of the defining polynomial, there exist efficient algorithms to determine the ring of integers, e.g. [17, Algorithm 6.1.8], [24, Chapter III], and implementations are available in various Computer Algebra Systems, including those mentioned before. However, these algorithms are quite involved, which is one reason why we opted for a *certification* approach for rings of integers. This is described in Section 5, the technical heart of this paper.

To show the feasibility of our approach, we formally verified several entries in the well known L-functions and modular forms database (LMFDB) [34]. For various number fields,

we verified in Lean the ring of integers, as given by an explicit *integral basis*, as well as the *discriminant* (an integer valued invariant); see Section 7.

In Section 3, we provide an overview of the different notions of Lean computation used. In order for everything to function, we needed to formalize much underlying theory and computational methods, most of which are interesting independently. Notable topics include irreducibility of polynomials over $\mathbf{Q}$ and finite fields (see Section 4), as well as resultants and discriminants (see Section 6).

We conclude the paper with a brief discussion (in Section 8), including related and future work. Full source code of our formalization and SageMath scripts are available.[1]

2 Preliminaries

In this paper, we assume some familiarity with basic ring and field theory, as can be found e.g. in the undergraduate textbooks [21, 31]. For sake of self-containedness, we will discuss in this section some basics concerning number fields and their rings of integers; the following paragraph essentially follows the exposition (with very little modification) from [5, Section 2].

A number field K is a finite extension of the field $\mathbf{Q}$. It is a finite dimensional vector space over $\mathbf{Q}$, and its dimension is called the degree of K (over $\mathbf{Q}$). Examples of number fields are $\mathbf{Q}$ itself (of degree 1), $\mathbf{Q}(\sqrt{3}) = \{a + b\sqrt{3} : a, b \in \mathbf{Q}\}$ and $\mathbf{Q}(\sqrt{-3}) = \{a + b\sqrt{-3} : a, b \in \mathbf{Q}\}$ (both of degree 2), and $\mathbf{Q}(\alpha) = \{a + b\alpha + c\alpha^2 : a, b, c \in \mathbf{Q}\}$ where α is any (complex) root of the polynomial $x^3 - 3x - 10$. This latter example generalizes as follows. For any polynomial of degree n that is irreducible over $\mathbf{Q}$, adjoining any one of its (complex) roots β to $\mathbf{Q}$ will yield the degree n number field $\mathbf{Q}(\beta) = \{a_0 + a_1\beta + \ldots + a_{n-1}\beta^{n-1} : a_0, a_1, \ldots, a_{n-1} \in \mathbf{Q}\}$ (the n different choices of β will all yield isomorphic fields). Conversely, any number field of degree n will be isomorphic to such a field $\mathbf{Q}(\beta)$. From an algebraic and arithmetic perspective, the (rational) integers $\mathbf{Z}$ constitute a particularly 'nice' subring of its field of fractions, the rational numbers $\mathbf{Q}$. Upon generalizing from $\mathbf{Q}$ to an arbitrary number field K, the analogue of $\mathbf{Z}$ is the ring of integers of K, denoted $\mathcal{O}_K$. It is defined as the integral closure of $\mathbf{Z}$ in K:

$$\mathcal{O}_K := \{x \in K : \exists f \in \mathbf{Z}[x] \text{ monic} : f(x) = 0\}$$

where we recall that a (univariate) polynomial is called *monic* if its leading coefficient is equal to 1. The fact that $\mathcal{O}_K$ is indeed a ring follows for instance from general properties of integral closures ([37, Section I.2]).

The ring of integers for the four explicit examples of number fields above, are $\mathcal{O}_{\mathbf{Q}} = \mathbf{Z}$ (indeed), $\mathcal{O}_{\mathbf{Q}(\sqrt{3})} = \mathbf{Z}[\sqrt{3}] = \{a + b\sqrt{3} : a, b \in \mathbf{Z}\}$, $\mathcal{O}_{\mathbf{Q}(\sqrt{-3})} = \mathbf{Z}[(1 + \sqrt{-3})/2] = \{a + b(1 + \sqrt{-3})/2 : a, b \in \mathbf{Z}\}$, and $\mathcal{O}_{\mathbf{Q}(\alpha)} = \{a + b\alpha + c(\alpha - \alpha^2)/2 : a, b, c \in \mathbf{Z}\}$. For the third example, note that indeed $(1 + \sqrt{-3})/2 \in \mathcal{O}_{\mathbf{Q}(\sqrt{-3})}$ as it is a root of the monic polynomial $x^2 - x + 1$.

The last (cubic) example shows that the ring of integers can become complicated very quickly. In this case $\mathcal{O}_{\mathbf{Q}(\alpha)}$ *cannot* be written in the form $\mathbf{Z}[\gamma]$ for any element $\gamma \in \mathcal{O}_{\mathbf{Q}(\alpha)}$, which is phrased as $\mathcal{O}_{\mathbf{Q}(\alpha)}$ being not *monogenic*. Note that any number field (say of degree n) is monogenic, as it can be generated by a single element β over $\mathbf{Q}$, which yields $\{1, \beta, \beta^2, \ldots, \beta^{n-1}\}$ as a basis for the corresponding $\mathbf{Q}$-vector space, called a *power basis*. Observe that the ring of integers $\mathcal{O}_K$ of a number field K is a $\mathbf{Z}$-module. Recall that in general an R-module is just a vector space if R is a (skew) field, but the same standard defining axioms make sense in the more general case that the *scalars* R are only assumed to form a ring. While the $\mathbf{Z}$-module $\mathcal{O}_K$ might not have a power basis (as it may not be monogenic), it always has some basis, called a $\mathbf{Z}$-basis or *integral basis*. E.g. in the cubic example, a $\mathbf{Z}$-basis is given by $\{1, \alpha, (\alpha - \alpha^2)/2\}$, as any element of $\mathcal{O}_{\mathbf{Q}(\alpha)}$ can be expressed uniquely as a $\mathbf{Z}$-linear combination of these three basis elements. Finally, if we have a subring $\mathcal{O}$ of a number field K, which as a $\mathbf{Z}$-module has a basis that also forms a $\mathbf{Q}$-basis for K, then $\mathcal{O}$ is actually contained in $\mathcal{O}_K$.

3 Forms of Computation in Lean

This section provides an overview of the different notions of computation that we had to deal with.

Lean has a built-in notion of computation through *reduction*: this is a primitive relation between two terms reflecting the computational content of the Calculus of Inductive Constructions [13]. Reduction is invoked as part of definitional equality checking, and allows Lean to verify equalities such as $2 + 2 = 1 + 3$ through computing that either side evaluates to 4. Definitionally equal terms are indistinguishable for the type theory, and therefore a proof of $2 + 2 = 1 + 3$ can be given by the reflexivity principle of equality, rfl. In Lean 4, reduction also applies to literals of primitive types: natural numbers and strings. Reduction is implemented as part of the trusted codebase in the Lean kernel.

Lean as a programming language additionally possesses a notion of computation for definitions which can be compiled into executable code and *evaluated*. Evaluation is somewhat orthogonal to reduction in that it is not part of the logic of Lean, and not all definitions that can be evaluated will reduce, and vice versa. In particular, Lean includes –and Mathlib makes consistent use of– the axiom of choice, resulting in *noncomputable* definitions that cannot be evaluated. The user can additionally override with arbitrary code the computational meaning of a definition, using the @[implemented_by] attribute.

Evaluation is the notion of computation powering the tactic framework. Broadly speaking, tactics are evaluated Lean code that operate directly on the term level. Tactics can also

[1] https://github.com/alainchmt/RingOfIntegersProject

be said to perform computation, by taking in a term t and producing a new term t′ alongside a proof term of type t = t′. The norm_num tactic for computing with numerical expressions is an example of this design. Often, a tactic works by *reflecting* a given term into a data structure internal to the tactic, using computation on internal data to construct a proof term. Since tactics can choose the internal representation of the data structures, they reimbue noncomputable terms with computational meaning. An example is the ring tactic of Mathlib which computes on polynomial expressions although the ring structure of the type Polynomial itself is marked noncomputable.

If a tactic returns rfl as the proof term for the equality t = t′, verifying the definitional equality of the terms t and t′ can invoke reduction: this principle of *computational reflection* [9] saves the need to go through the computation twice, once on the tactic-internal data structure and once on the constructed proof term. The tactic decide is an example of a reflective tactic in Lean, proving a given goal P by finding a corresponding decision procedure d such that d = true if and only if P, and then verifying that d = true holds. This tactic has two variants: decide first computes whether d = true in the elaborator, before submitting the same computation to the kernel, while decide! directly invokes kernel reduction and thus can unfold every definition.

Efficiency is an important consideration in the use of computation in Lean. In the highest generality, evaluation is faster than reduction since evaluation does not rely on the abstraction of terms that must be constructed and matched for each reduction step. The higher efficiency of evaluation comes at the cost of being separated from the logic: proof checking can only invoke reduction. Lean provides an escape hatch in the form of the native_decide tactic: unlike decide, this allows a proof whenever a decision procedure *evaluates* to true. Use of native_decide not only means that the entire evaluation mechanism has to be added to the trusted codebase, in addition the correctness of all @[implemented_by] attributes must be trusted. Use of native_decide is therefore a tradeoff between efficiency and certainty. See Section 7 for some performance measurements in our project.

4 Irreducibility

For a primitive polynomial f in $\mathbf{Z}[X]$ – a polynomial whose coefficients have greatest common divisor equal to 1– proving that $\mathbf{Q}[X]/\langle f\rangle$ is a number field is equivalent to proving that f is irreducible over $\mathbf{Z}[X]$. Thus, generating an irreducibility proof for polynomials over the integers in Lean is crucial for our purposes.

4.1 Polynomials as Lists

In Mathlib, the underlying representation of polynomials is implemented in such a way that polynomial arithmetic is noncomputable. Therefore, we work with an alternative representation of polynomials as lists.

For a semiring R with decidable equality, we define a map Polynomial.ofList : List R → R[X] that converts a list of coefficients $[a_0, \ldots, a_n]$ into the polynomial $a_0 + a_1X + \ldots + a_nX^n$. We then define computable operations on List R that correspond to operations on polynomials, such as addition and multiplication. This translation of polynomial arithmetic into list arithmetic will be our general strategy for dealing with polynomial computations. For example, to prove that ofList l_1 * ofList l_2 = ofList l_3, we show:

$(l_2 * l_2)$.dropTrailingZeros = l_3.dropTrailingZeros

This equality can be proven using the rfl and decide tactics. Here, dropTrailingZeros removes the trailing zeros of a list.

We note that this approach to polynomial computation relies on the input polynomials being provided in the list-based format, as the standard way to define polynomials in Mathlib is noncomputable. Alternatively, one could consider using the Mathlib representation of polynomials along with tactics such as ring [4] (which is partly based on the Coq tactic [26]) and norm_num extensions like reduce_mod_char to prove equalities involving additions and multiplications. However, this method proved too slow for our purposes, which led us to adopt the list-based approach instead.

In addition to Polynomial.ofList and the arithmetic operations on lists, we defined ComputablePolynomial R as the subtype of lists l over R such that l = l.dropTrailingZeros, and proved that it is isomorphic to Mathlib polynomials. The algebraic structure of ComputablePolynomial R, namely, that it is a ring when R is a ring, is pulled from the corresponding instance in Mathlib polynomials. This representation is similar to the one in Coq's Mathematical Components library [36], which uses lists with a non-zero last entry to define polynomials, though our approach does not assume that R is nontrivial.

Having multiple encodings for the same mathematical object, one better suited for proving theoretical properties and another for computation, has been explored in previous formalization efforts of algebraic algorithms [20]. While we do not employ a proof transfer framework such as Trocq [16], this could potentially be explored to automate translations between equivalent data structures.

4.2 Over Finite Fields

In this section, we describe a way to prove irreducibility of polynomials over a finite field, which will aid us in proving irreducibility in $\mathbf{Z}[X]$. We then present a certificate for this irreducibility and its implementation in Lean.

A finite field is a field with finite cardinality. This cardinality is necessarily a prime power and two finite fields of equal cardinality are always isomorphic. Thus, we may speak of *the* finite field of cardinality q, which we denote by $\mathbf{F}_q$. Remarkably, the irreducible factors of $X^{q^n} - X$ in $\mathbf{F}_q[X]$ are precisely the irreducible polynomials in $\mathbf{F}_q[X]$ that have degree dividing n. This gives rise to an irreducibility test due to

Rabin [39], which forms the basis to our formal irreducibility proofs.

Theorem 4.1 (Rabin's test). *Let f be a polynomial over $\mathbf{F}_q$ of degree $n > 0$. Then f is irreducible if and only if:*

1. *f divides $X^{q^n} - X$*
2. *$\gcd(f, X^{q^{n/t}} - X) = 1$ for every prime t dividing n.*

In Lean, to talk about a finite field, we follow the practice in Mathlib and endow (F : Type*) with instances [Field F] [Fintype F]. The instance [Fintype F] carries data: a finite set containing all elements of F. We use it rather than [Finite F], which merely asserts that F is in bijection with Fin n, the canonical type with n elements, for some natural number n. The previous statement –which we formalized– reads:

```
theorem irreducible_iff_dvd_X_pow_sub_X' (f : F[X])
  (hd : 0 < f.natDegree) : Irreducible f ↔
f ∣ (X ^ ((Fintype.card F) ^ f.natDegree) - X) ∧
∀ (p : ℕ), Nat.Prime p → p ∣ f.natDegree →
IsCoprime f (X ^ (Fintype.card F ^ (f.natDegree / p)) - X)
```

The term Fintype.card F is the cardinality of the finite set of all elements of F coming from the [Fintype F] instance.

Currently, automatic computations in finite fields can only be carried out in Lean for fields of prime order (our work on irreducible polynomials can contribute to extending this to higher degree extensions), therefore we focus on the case $q = p$, with p a prime number. In Lean, the integers modulo p are represented as ZMod p, which has decidable equality, allowing us to solve identity goals simply with the decide tactic. Furthermore, if we have the instance Fact (Nat.Prime p) available, then Lean infers that ZMod p is a field.

Rabin's test starts by proving $f \mid X^{p^n} - X$, where n is the degree of the polynomial f in $\mathbf{F}_p[X]$. The obvious way to do this is to provide the factor s such that $X^{p^n} - X = f * s$ and verify the equality. There is, however, a problem. In general, s will not be sparse and it will be of huge degree equal to $p^n - n$. This makes storing it and verifying the corresponding identity infeasible. The standard approach taken in computer algebra [44] is instead to show that $X^{p^n} \equiv X \pmod f$ using a *square-and-multiply* algorithm, reducing modulo f at each step by performing polynomial division with remainder. We currently do not have an effective implementation of polynomial division with remainder in Lean. However, with our implementation of polynomials as lists, we can quickly verify multiplications. This suggest a certificate for $f \mid X^{p^n} - X$ based on the *square-and-multiply* algorithm.

This well-known algorithm performs fast exponentiation on any monoid. When applied to the ring $\mathbf{F}_p[X]/\langle f \rangle$, the underlying proposition looks like this:

Proposition 4.2. *Let f and g in $\mathbf{F}_p[X]$ and let m be a non-negative integer. Let $(\beta_i)_{i=0}^s$ be the binary digits of m so that $m = \sum_{i=0}^s \beta_i 2^i$. Set $y_s \equiv g^{\beta_s} \pmod f$ and $y_i \equiv y_{i+1}^2 * g^{\beta_i} \pmod f$ for $i = s-1, \ldots, 0$. Then $y_0 \equiv g^m \pmod f$.*

Computing $g^m \pmod f$ this way requires significantly fewer operations than repeated multiplication. The polynomials y_i can be chosen with degrees less than n. Note that, when a and b do not have very large degree, we *can* certify $a \equiv b \pmod f$ in the obvious way.

The second step in Rabin's test asks us to prove $\gcd(f, X^{p^m} - X) = 1$ for divisors m of n. This also involves polynomials of large degree. However, by certifying $X^{p^m} \equiv h_m \pmod f$ for $1 \le m \le n$, with h_m of degree less than n, we can instead prove $\gcd(f, h_m - X) = 1$. This can be done by providing polynomials a_m and b_m –computed using the extended Euclidean algorithm– such that $a_m * f + b_m * (h_m - X) = 1$.

Putting it all together, we get a certificate which can be verified using only polynomial addition and multiplication.

Certificate. Let f in $\mathbf{F}_p[X]$ of degree $n > 0$, let $(\beta_i)_{i=0}^s$ be the binary digits of p, so $p = \sum_{i=0}^s \beta_i 2^i$. A certificate for the irreducibility of f consists of the following data:

- An $(n+1)$-tuple $(h_0, \ldots, h_n)$ over $\mathbf{F}_p[X]$.
- An $n \times s$ matrix (g_{ij}) over $\mathbf{F}_p[X]$.
- An $n \times (s+1)$ matrix (h'_{ij}) over $\mathbf{F}_p[X]$.
- n-tuples $(a_0, \ldots, a_{n-1})$ and $(b_0, \ldots, b_{n-1})$ over $\mathbf{F}_p[X]$.

The verification proceeds by checking the following statements:

(i) For all $0 \le i < n$, $h'_{is} = h_i^{\beta_s}$ and $h'_{i0} = h_{i+1}$.
(ii) For all $0 \le i < n$ and $0 \le j < s$,
$$f * g_{ij} = {h'_{i(j+1)}}^2 * h_i^{\beta_j} - h'_{ij}.$$
(iii) $h_0 = X$ and $h_n = X$.
(iv) $a_{n/t} * f + b_{n/t} * (h_{n/t} - X) = 1$ for every prime $t \mid n$.

The first two statements in the verification imply that $h_i^p \equiv h_{i+1} \pmod f$ for all $0 \le i < n$. Together with $h_0 = X$, we get that $X^{p^i} \equiv h_i \pmod f$ for every $0 \le i \le n$. Since $h_n = X$, the first part of Rabin's test is shown. The fourth statement proves the second part of the test. Note that in this last statement, only some entries a_i and b_i, with i dividing n, are used. Consequently, the rest of the entries in these n-tuples, which are included solely to simplify the indexing, can be set to zero.

For our formalization, we use the following strategy repeatedly: given a certification scheme, we define a structure in Lean where the fields include both the certifying data and the proofs of the verification statements. Often, we include more fields than what the informal description may suggest since some implicit assumptions require formal proof.

For the above irreducibility certificate, we use our list-based approach to handle polynomial arithmetic. We bundle the certifying data and proofs of the verification statements into the structure:

```
structure CertificateIrreducibleZModOfList' (p n t s : ℕ)
    [Fact (Nat.Prime p)] [NeZero n] (L : List (ZMod p))
```

It is parametrized over p, n and s as before. Here, L is the list of coefficients of the polynomial f. The parameter

t allows flexibility in choosing a base for the *square-and-multiply*-type approach. We have described the method using the typical $t = 2$. However, choosing $t = p$ might be advantageous in some cases because computing p-powers in $\mathbf{F}_p[X]$ is easy, although the g_{ij} will have larger degree.

As part of the fields in this structure, we include the factorization data for the degree n, which is used in (iv):

```
m : ℕ
P : Fin m → ℕ
exp : Fin m → ℕ
hneq : ∏ i : Fin m, (P i) ^ (exp i) = n
hP : ∀ i, Nat.Prime (P i)
```

If the prime factors of n are small (one or two digits), then a proof hP can be obtained automatically using the decide tactic (we say more about primality proofs in Section 4.4). The tuples $(a_i)_{i=0}^{n-1}$, $(b_i)_{i=0}^{n-1}$ are also fields in the structure:

```
a : Fin n → List (ZMod p)
b : Fin n → List (ZMod p)
```

As well as a proof of the fourth verification statement, written in terms of list arithmetic:

```
hgcd : ∀ (i : Fin m),
(a ↑(n / P i) * L + b ↑(n / P i) * (h ↑(n / P i) - [0, 1]))
  .dropTrailingZeros = 1
```

The rest of the certifying data together with proofs of the verification statements is included in the structure in this way. Once a term of this type has been constructed, we get a proof that the polynomial `ofList L` is irreducible.

For an irreducible polynomial over $\mathbf{F}_p$, a certificate of this form always exists. The function `CertificateIrrModpFFP`, implemented in SageMath, computes the certifying data – including the decomposition of the degree into prime factors– and writes the corresponding term in Lean. For small degrees (usually $n \leq 12$) all of the proofs included as fields in this structure can be solved automatically either by using the rfl or the decide tactic. For higher degrees, the computation may exceed Lean's preset maximum recursion depth or heartbeats limits. A variety of approaches could be used to deal with this timeout issue: it might be possible to develop better decision procedures for use with decide, or we might use (a custom extension to) the norm_num tactic that performs this computation.

4.3 Degree Analysis

Here, we describe a certificate for the irreducibility of polynomials in $\mathbf{Z}[X]$, constructed by combining information from the factorization of the polynomial modulo distinct primes.

Let f in $\mathbf{Z}[X]$ of degree n and p a prime number not dividing $\mathrm{lc}(f)$, the leading coefficient of f. Let $\bar{f} = f \pmod p$ be the polynomial over $\mathbf{F}_p$ obtained by reducing the coefficients of f modulo p. Write $\bar{f} = \prod_{i=1}^{m} g_i$, with g_i irreducible in $\mathbf{F}_p[X]$. Consider the tuple $D_p = (\deg(g_i))_i$ and let $S(D_p)$ be the set of all possible subset sums of D_p. It is not hard to see that if a divides f, then $\deg a \in S(D_p)$. Computing $S(D_p)$ for different primes p will thus give us refined information on the degree of the factors of f. In particular:

Proposition 4.3. *Let f in $\mathbf{Z}[X]$ be primitive and $p_1, \ldots, p_m$ a collection of distinct primes not dividing $\mathrm{lc}(f)$. Let*

$$d = \min\left(\bigcap_{i=1}^{m} S(D_{p_i}) \setminus \{0\}\right).$$

If $a \in \mathbf{Z}[X]$ divides f and is neither 1 or -1, then $d \leq \deg a$.

It follows that if $d = \deg f$, then f is irreducible. This gives an irreducibility certification scheme, which we formalized as the structure IrreducibleCertificateIntPolynomial. As parts of its fields, it includes data on the irreducible factors of $f \pmod p$ for different primes p. The proofs of irreducibility are handled using the certifying structure from Section 4.2.

A certificate based on degree analysis, when it exists, is not unique. It is advisable to prioritize smaller primes that produce factors of small degrees for faster verification. Most polynomials (in an asymptotic sense) will have a certificate of this form, depending on their Galois group [40]. However, there are cases where such a certificate does not exist.

4.4 LPFW Certificate

This section describes an alternative certificate for irreducibility of polynomials in $\mathbf{Z}[X]$ which is more widely applicable. We discuss its formalization in Lean, and a script that can automatically generate a Lean proof of irreducibility based on these certificates.

There is a way of reducing irreducibility testing in $\mathbf{Z}[X]$ to primality testing in $\mathbf{Z}$: if a 'large enough' integer m is found such that $|f(m)|$ is prime, then f can be shown to be irreducible. This test was proposed by Brillhart in [10]. Abbott [1], using a refined calculation that incorporates a known lower bound on the degrees of f's factors, introduced the *large prime factor witness* (LPFW) certificate, which is similar to Brillhart's, but works for a larger number of cases and often produces smaller prime witnesses. It relies on the following proposition, which we have formalized.

Proposition 4.4. *Let f be a non-constant polynomial in $\mathbf{Z}[X]$. Suppose that $d \in \mathbf{Z}_{\geq 1}$ is a known lower bound for the degree of the (non-unit) factors of f. Let ρ in $\mathbb{R}$ be such that $|\alpha| \leq \rho$ for every root $\alpha \in \mathbf{C}$ of f. If there exists $m \in \mathbb{Z}$ such that $|m| \geq \rho + 1$ and $|f(m)| = sP$ with an integer $s < (|m| - \rho)^d$ and $P \in \mathbf{Z}$ prime, then f is irreducible.*

If a conjecture stated by Bouniakovsky [8] is true, then there will always exist an LPFW certificate for any irreducible polynomial f in $\mathbf{Z}[X]$. Ideally, the prime witness P should be as small as possible so its primality can be quickly verified. The higher the lower bound d for the degree of the factors of f, the more flexibility we have in finding such P. Note that d can be obtained using Proposition 4.3. This leads us to combine the degree analysis techniques from the previous section with the aforementioned Proposition.

To use Proposition 4.4, we must find a root bound ρ. Classical bounds are given by Lagrange [29] and Cauchy [14]. For dense polynomials with integer coefficients, Cauchy's bound is typically sharper (unless $|a_n|$ is very large). By applying it to $f(rX)$, with $r > 0$ a real number, one gets:

Proposition 4.5. *Let $f = \sum_{i=0}^{n} a_i X^i$ in $\mathbf{C}[X]$ of degree $n > 0$. Let $r > 0$ be a real number. If $\alpha \in \mathbf{C}$ is a root of f, then*

$$|\alpha| \leq r\left(1 + \frac{1}{|a_n|} \max_{0 \leq i \leq n-1} \frac{|a_i|}{r^{n-i}}\right)$$

By setting $r = 1$, we recover the original Cauchy bound. However, choosing the right r can yield a sharper bound. We formalized this (scaled) Cauchy bound and proved:

```
lemma polynomial_roots_le_cauchy_bound_scale
  (f : Polynomial ℂ) (z : ℂ) (hd : f.natDegree ≠ 0)
  (hr : z ∈ f.roots) (r : ℝ) (hs : 0 < r) :
  Complex.abs z ≤ cauchyBoundScaled f r := ...
```

The term cauchyBoundScaled takes as parameters a complex polynomial f and real number r and outputs a real number. However, since Mathlib polynomials are noncomputable and both complex and real numbers lack decidable equality, this definition is not optimal for computation. Thus, we defined a computable version, cauchyBoundScaledOfList, for polynomials over $\mathbf{Z}$ (given as lists) using a rational scaling factor. This allows us to prove its value by computation:

```
example : cauchyBoundScaledOfList [3,14,15,92,65] (1/2) =
  249/130 := by decide!
```

Recall from Section 3 that decide! performs kernel reduction to prove decidable propositions.

We formalized the LPFW certification scheme as:

```
structure CertificateIrreducibleIntOfPrimeDegreeAnalysis
  (f : Polynomial ℤ) (L : List ℤ)
```

It takes as parameters a polynomial f over the integers and its coefficient list L. Similar to the certificate in the previous section, it includes fields that certify the factorization of f modulo a given list of primes in order to obtain a lower bound d on the degree of the factors of f, as in Proposition 4.3. The structure also includes fields such as the Cauchy bound ρ –obtained using a scaling factor r–, the large prime witness P, the factor s, the evaluation argument m, as well as the proofs of the corresponding verification statements.

One field in the structure is a proof of Nat.Prime P. This is a decidable statement, and Mathlib provides the tactic norm_num that can prove a number is prime if it is not too large (< 7 digits). However, the prime witness P is typically big, so an alternative primality test might be necessary. Markus Himmel formalized Pratt certificates and a corresponding tactic [27], which can be used to prove such goals.

As before, irreducibility proofs for the factors of f modulo primes can be handled as in Section 4.2. All remaining proofs included as fields in the above structure can be solved automatically using decide, decide!, and/or norm_num. We also formalized a variant with the trivial degree lower bound $d = 1$, resulting in a shorter certificate.

We wrote a SageMath script that outputs a Lean proof of the irreducibility of f by selecting the appropriate certificate from Section 4.3 or Section 4.4. As noted in Section 4.2, large polynomial degrees may cause issues with recursion depth or excesive heartbeats when checking the proofs in Lean. This approach of generating Lean code from SageMath worked well for our applications. However, exploring a tighter integration, such as calling SageMath directly from Lean, could be an interesting direction for future work.

In the search for a smaller prime witness P, it can be helpful to transform f. Möbius transformations [1], and more generally, Tschirnhaus transformation [17, Algorithm 6.3.4], can be used. Incorporating this into our proof generator is something we also intend to pursue in the future.

5 Verifying Rings of Integers

As stated in the introduction, our main goal is to formally verify that a ring $\mathcal{O}$, with an explicitly given basis and contained inside a number field K, equals –in the set-theoretic sense– the ring of integers of K. Computation in $\mathcal{O}$ is key for both our proof and future applications. This raises two initial questions: how should we define $\mathcal{O}$ in Lean, and how can we compute in it?

5.1 Construction of the Subalgebra

The goal of this section is to describe the construction of a ring $\mathcal{O} \subseteq K$ in Lean, given by an explicit basis.

Consider a more general setting where R and K are commutative rings and K is actually an R-algebra. We recall that this means that there is a ring homomorphism $\phi : R \to K$ and as such K carries the natural structure of an R-module by defining the scalar multiplication $r \cdot x$ (where $r \in R$ and $x \in K$) by $\phi(r)x$. An R-subalgebra $\mathcal{O}$ of K can be formalized in multiple ways. For example, we could consider a type ($\mathcal{O}$: Type*) with instances [CommRing $\mathcal{O}$] [Algebra R $\mathcal{O}$] and equip it with an injective R-algebra homomorphism f : $\mathcal{O}$ $\to_a$[R] K. However, carrying along this map through constructions and proofs can be rather cumbersome. A better approach is to use the subalgebra structure in Mathlib, which bundles together a subset of K, the carrier set, and the proofs that it is an R-algebra. In fact, the integral closure of R in K is defined in Mathlib with type Subalgebra R K, allowing us to state our goal as $\vdash \mathcal{O}$ = integralClosure R K.

Defining $\mathcal{O}$ simply as the R-span of a given set gives a term of type Submodule R K. To make this into a subalgebra we also need proofs that it is closed under multiplication and contains 1. In order to automate this construction in Lean starting with an R-basis, we created the `SubalgebraBuilder`.

To describe how it works, let us further specialize our setup and fix some notation. Suppose that R is a domain, Q its fraction field, and K the extension of Q obtained by

adjoining a root θ of a monic polynomial T in $R[X]$ of degree n (e.g. $R = \mathbf{Z}$, $Q = \mathbf{Q}$ and K a number field). Suppose that $\{w_1, \ldots, w_n\}$ is an R-basis for a subalgebra $\mathcal{O}$ of K. Since $K \cong Q[X]/\langle T \rangle$, we can find a common denominator $d \in R \setminus \{0\}$ and polynomials $b_i \in R[X]$ of degree less than n such that $w_i = \frac{1}{d} b_i(\theta)$.

As a module, $\mathcal{O}$ is the R-span of $\{w_1, \ldots, w_n\}$. However, as previously remarked, in order to give it a type Subalgebra R K instead of just Submodule R K, we need to convince Lean that $\mathcal{O}$ is closed under multiplication and contains 1. To prove closure under multiplication it suffices to show that $w_i * w_j$ is in $\mathcal{O}$ for all i and j. A way to certify this is to give the coefficients $a_{ijk} \in R$ such that $w_i * w_j = \sum_k a_{ijk} \cdot w_k$. Verifying this equality in K is equivalent to proving that for all i and j there exists a polynomial $s_{ij} \in R[X]$ such that

$$b_i * b_j = d \sum_k a_{ijk} \cdot b_k - T * s_{ij}. \tag{1}$$

Thus, the elements a_{ijk} and polynomials s_{ij} act as a certificate for the closure of $\mathcal{O}$ under multiplication. In fact, they provide a certificate for the precise representation of $w_i * w_j$ with respect to the basis $\{w_1, \ldots, w_n\}$. Since multiplication is commutative, verifying this certificate requires checking $(n^2 + n)/2$ identities in $R[X]$. Proving that 1 is in $\mathcal{O}$ reduces to a single polynomial identity check.

Besides constructing $\mathcal{O}$ as a subalgebra in Lean, we also want a basis for it, which would be a term of type Basis ι R $\mathcal{O}$ (with ι some indexing type). To prove that $\{w_1, \ldots, w_n\}$ is indeed a basis, we must show that it is linearly independent. Write $b_j = \sum_{i=0}^{n-1} c_{ij} X^i$ with $c_{ij} \in R$, then

$$w_j = \tfrac{1}{d} \textstyle\sum_{i=0}^{n-1} c_{ij} \theta^i.$$

The set $\{w_1, \ldots, w_n\}$ is linearly independent if and only if the determinant of the matrix $B = (c_{ij})$ is non-zero. To simplify this verification, we can *choose* a basis where B is upper triangular. In that case, it suffices to check that the diagonal entries are non-zero.

Since any matrix over a Bézout domain has an echelon form [28], if R is Bézout (e.g. $R = \mathbf{Z}$ or $R = \mathbf{F}_p[X]$) there is a basis for $\mathcal{O}$ with B upper triangular. For $R = \mathbf{Z}$, a basis in Hermite normal form [17, Theorem 4.7.3] may be used.

We incorporate all this data and proofs into the structure:

```
structure SubalgebraBuilderLists
(n : ℕ) (R Q K : Type*) [CommRing R] [IsDomain R]
[DecidableEq R] [Field Q] [CommRing K] [Algebra Q K]
[Algebra R Q] [Algebra R K] [IsScalarTower R Q K]
[IsFractionRing R Q] (T : R[X]) (L : List R)
```

It takes as explicit parameters R, Q, K, the degree n, the monic polynomial T, and the list L of coefficients of T. We follow the practice in Mathlib to encode the structure of inclusion of R into its fraction field Q, and Q into the ring K using the Algebra and IsFractionRing typeclasses. The transitive inclusion of R into K is witnessed by [Algebra R K] [IsScalarTower R Q K] instances; the latter asserts the triangle of inclusions $R \to Q \to K$ commutes. Browning and Lutz [11] explain Mathlib's approach to extensions (of fields) in depth, and Wieser [45] covers scalar actions in Mathlib more generally.

The fields of the structure include the matrix B^T –with B upper triangular containing the basis coefficients c_{ij} as described above–, the common denominator d, the coefficients a_{ijk}, the certifying polynomials s_{ij} in list form, as well as the proofs that B is upper triangular with non-zero elements in the diagonal, that the identities in (1) hold (using list-based arithmetic), and that K is obtained by adjoining a root of T. All of these proofs, except the latter, can be solved automatically using the decide, decide!, and/or norm_num tactics. The latter statement is conveyed with the predicate IsAdjoinRoot K (map (algebraMap R Q) T). Its proof will depend on the specific nature of K. If K is *defined* as AdjoinRoot (map (algebraMap R Q) T), then this proof is obtained for free.

With A of this type, the term

```
subalgebraOfBuilderLists T L A : Subalgebra R K
```

is the corresponding R-subalgebra $\mathcal{O}$ of K which, as a set, is the R-span of $\{w_1, \ldots, w_n\}$, specified by B. We obtain a basis $(w_i)_{i=1}^n$ for this subalgebra –indexed with Fin n– in basisOfBuilderLists T L A, with $w_i = \frac{1}{d} b_i(\theta)$.

An element in $\mathcal{O}$ can be represented by a vector Fin n → R using its coordinates with respect to this basis. As previously noted, the polynomial identities in (1) prove that the coordinates of $w_i * w_j$ are given by $(a_{ijk})_{k=1}^n$. This information can be collected in a *times table*: an $n \times n$ matrix where the ij-entry is the vector $(a_{ijk})_{k=1}^n \in R^n$. This notion is more generally formalized in Lean as a structure introduced in [5] called TimesTable, which bundles the data of a basis and the associated times table.

```
structure TimesTable ( ι R S : Type*) [Semiring R]
  [AddCommMonoid S] [Mul S] [Module R S] : Type*
 where
basis : Basis ι R S
table : ι → ι → ι → R
basis_mul_basis : ∀ i j k, basis.repr (basis i * basis j) k =
     table i j k
```

Here, basis is an R-basis for S, and basis.repr sends an element in S to its vector representation. The last field states that the product of the i-th and j-th basis elements is represented by the ij-th entry of table. The `SubalgebraBuilder` contains the information to construct the times table for $\mathcal{O}$. Indeed, timesTableOfSubalgebraBuilderLists T L A has type TimesTable (Fin n) R $\mathcal{O}$.

5.2 Arithmetic in $\mathcal{O}$

We discuss how we can use a times table to compute in $\mathcal{O}$.

More generally, let S be an R-algebra which is finite and free as an R-module and has basis $\mathcal{B} = \{w_1, \ldots, w_n\}$. An element x in S can be uniquely written as $x = \sum_i c_i w_i$, and thus represented as a vector $(c_1, \ldots, c_n) \in R^n$. We have that $S \cong R^n$

as R-modules. Furthermore, we can *define* a multiplication on R^n that corresponds to the multiplication in S:

Let $x = \sum_i c_i w_i$ and $y = \sum_i d_i w_i$, with $c_i, d_i \in R$, be elements in S. We have $xy = \sum_i \sum_j (c_i d_j) \cdot (w_i * w_j)$. Let T be the times table for S with respect to $\mathcal{B}$, so that $T_{ij} = (a_{ijk})_k \in R^n$, with $w_i * w_j = \sum_k a_{ijk} w_k$. By defining

$$(c_i)_i * (d_i)_i := \sum_i \sum_j (c_i d_j) \cdot T_{ij} \tag{2}$$

in R^n, we can translate arithmetic in S to arithmetic in R^n and vice versa. If we have a times table for S, this approach allows us to compute in S with Lean using lists as follows:

Consider a $n \times n$ matrix T with entries in R^n. In Lean, this is a term `T : Fin n → Fin n → Fin n → R`. Using our previously defined list arithmetic, we formalize (2) as:

```
def table_mul_list' (c d : List R) : List R :=
  List.sum (List.ofFn (fun i => List.sum (List.ofFn
  (fun j => List.mulPointwise ((List.getD c i 0) * (List.getD
      d j 0))  (List.ofFn (T i j))))))
```

This function is meant to be used with c and d of length n. However, we give them type `List R` instead of `Fin n → R` because lists offer faster computation. The term `List.getD c i 0` retrieves the i-th entry of c, returning 0 if i is out of bounds. A variant where T has type `Fin n → Fin n → List R` is also defined for faster computation.

With this approach, we also recursively defined efficient exponentiation using the square-and-multiply method. Note that addition in S corresponds to pointwise addition on lists.

We formally proved that, when T is a times table for S, these functions on lists correspond to the arithmetic operations in S. Thus, identities in S –involving additions and multiplications– can be proven by first translating them into their equivalent operations on lists. This method is used to prove identities in the subalgebra $\mathcal{O}$ of a number field for which we have a times table `TimesTable (Fin n) ℤ 𝒪`. This verification on lists of integers can be done automatically by a tactic such as decide!. A similar strategy is used for the $\mathbb{F}_p$-algebra $\mathcal{O}/p\mathcal{O}$ from Section 5.6.

5.3 Local Maximality

Here, we present a local notion of maximality which serves as a tool to prove an equality between subalgebras.

Let K be a number field of degree n and $\mathcal{O}$ a subalgebra which is free and finite as a $\mathbf{Z}$-module. Then $\mathcal{O}$ is contained in $\mathcal{O}_K$, and we can compare them by considering the index $[\mathcal{O}_K : \mathcal{O}]$, typically defined as the cardinality of the quotient $\mathcal{O}_K/\mathcal{O}$. When $\mathcal{O}$ has rank n, then $[\mathcal{O}_K : \mathcal{O}]$ is a finite integer.

If p is a prime that does not divide $[\mathcal{O}_K : \mathcal{O}]$, we say that $\mathcal{O}$ is *p-maximal*. Thus, if one can prove that $\mathcal{O}$ is p-maximal for every prime p, it will follow that $\mathcal{O} = \mathcal{O}_K$.

This idea, which will guide our approach for constructing a formal proof of $\mathcal{O} = \mathcal{O}_K$, is also central to the currently used algorithms for computing rings of integers. However, the approach of showing a *global* equality by verifying a *local* property is not limited to subrings in number fields.

We formalized a local notion of maximality that applies to free and finite modules over Principal Ideal Domains (abbreviated as *PID*s, which means that every ideal can be generated by one element) using a generalized notion of index, similar to [12]. This allows us to state and prove several useful results in wider generality. For M a free module over a PID of finite rank, the index of a submodule N, which we also denote by $[M : N]$, is an element in R. If $[M : N]$ is a unit, then $M = N$. When $R = \mathbf{Z}$ and M and N have equal rank, this index coincides (up to sign) with the cardinality of M/N.

In Lean, to say that R is a PID and M is an R-module we use the hypothesis `[CommRing R] [IsPrincipalIdealRing R] [IsDomain R] [AddCommGroup M] [Module R M]`. We formalized the above notion of index as:

```
def Submodule.indexPID (N : Submodule R M)
      [Module.Free R M] [Module.Finite R M] : R
```

where we include `[Module.Free R M] [Module.Finite R M]` as instance parameters, indicating that M is free and finite. The definition is tagged `noncomputable` as it depends on choice of bases, which are noncomputable.

For an element π in R, we say that N is π-maximal if π does not divide $[M : N]$. In formal terms, this reads

```
def piMaximal [Module.Free R M] [Module.Finite R M]
  ( π : R) (N : Submodule R M) : Prop :=
  ¬ ( π ∣ Submodule.indexPID N)
```

From the properties of the index, it follows that if N is π-maximal for every prime π, then $M = N$; where we recall that π is prime means that it is a nonzero non-unit element such that $\pi | ab$ implies $\pi | a$ or $\pi | b$ for any $a, b \in R$.

As outlined at the start of this section, we will use this definition when K is an R-algebra, $\mathcal{O}$ and $\mathcal{O}'$ have type `Subalgebra R K`, and we have the hypothesis `(hm : 𝒪 ≤ 𝒪')`. To state that $\mathcal{O}$ is π-maximal (with respect to $\mathcal{O}'$), we need to view $\mathcal{O}$ as a submodule of $\mathcal{O}'$. This is done via the inclusion map `↥𝒪 →ₐ[R] ↥𝒪'`. The term

```
Subalgebra.toSubmodule (Subalgebra.inclusion hm).range
```

is the range of this map, which we abbreviate as `𝒪*`, and has type `Submodule R ↥𝒪'`, where `↥𝒪'` is the coercion of $\mathcal{O}'$ into a subtype of K.

5.4 The Pohst–Zassenhaus Theorem

In this section, we introduce a crucial theorem that can be used to establish local maximality and forms the basis for the certificate discussed in Section 5.6.

Let R be a PID, Q its fraction field, and K an R-algebra. Consider two R-subalgebras of K, denoted by $\mathcal{O}$ and $\mathcal{O}'$, which are free and finite as R-modules, have the same rank, and satisfy $\mathcal{O} \subseteq \mathcal{O}'$. The primary case of interest to keep in mind is: $R = \mathbf{Z}$, K is a number field, and $\mathcal{O}' = \mathcal{O}_K$.

To construct a proof of $\mathcal{O} = \mathcal{O}'$ we follow the strategy described in the previous section by establishing π-maximality for every prime π of R. The main result we use is based on

a theorem due to Pohst and Zassenhaus [38, Lemma 5.53], which underpins the widely used *Round-2 algorithm* for computing rings of integers [17, Algorithm 6.1.8].

For I and ideal of $\mathcal{O}$, we define the *multiplier ring* of I by $r(I) = \{x \in K \mid xI \subseteq I\}$. It is an R-subalgebra of K which contains $\mathcal{O}$. We formalized this definition as

```
def multiplierRing (I : Ideal 𝒪) : Subalgebra R K where
  carrier := {(x : K) | ∀ (i : 𝒪), i ∈ I →
                ( ∃ (j : 𝒪), j ∈ I ∧ i * x = j )} ...
```

The radical of I is the ideal $\{x \in \mathcal{O} \mid \exists m \geq 1, x^m \in I\}$. For $r \in R$, we denote by I_r the radical of the principal ideal $r\mathcal{O}$. The main theorem we use to prove π-maximality is:

Theorem 5.1. *Let π in R be a prime and suppose that $\mathcal{O}$ and $\mathcal{O}'$ have equal rank. If $\mathcal{O} = r(I_\pi)$, then $\mathcal{O}$ is π-maximal.*

This is a version of the Pohst–Zassenhaus theorem which is sufficient for our purposes. However, it is more commonly presented with K being a finite dimensional and separable Q-algebra, and $\mathcal{O}'$ the integral closure of R in K. In that case, it is an if-and-only-if statement [17], [38].

We formalized Theorem 5.1 as follows:

```
variable (hm : 𝒪 ≤ 𝒪') {π : R} (hp : Prime π )
local notation "𝒪*" => Subalgebra.toSubmodule
    (Subalgebra.inclusion hm).range
theorem order_piMaximal_of_order_eq_multiplierRing
     [Module.Free R 𝒪'] [Module.Finite R 𝒪']
    (heqr : Module.rank R 𝒪 = Module.rank R 𝒪')
    (heq : 𝒪 =
multiplierRing (Ideal.span {algebraMap R 𝒪 π}).radical) :
  piMaximal π 𝒪*:=
```

As before, $\mathcal{O}*$ represents $\mathcal{O}$ as a submodule of $\mathcal{O}'$. The term Ideal.span {algebraMap R O π} is the ideal $\pi\mathcal{O}$, with algebraMap R $\mathcal{O}$ π used to regard π as an element of $\mathcal{O}$.

The proof strategy for this theorem, which we adapted from [17], consists of defining an intermediate subalgebra $\mathcal{O}_\pi = \{x \in \mathcal{O}' \mid \exists j \geq 0$ such that $\pi^j x \in \mathcal{O}\}$, which sits between $\mathcal{O}$ and $\mathcal{O}'$ and is π-maximal by construction, and showing that $\mathcal{O} = \mathcal{O}_\pi$ whenever $\mathcal{O} = r(I_\pi)$.

Using Theorem 5.1, the proof of π-maximality of $\mathcal{O}$ reduces to the verification of the equality $\mathcal{O} = r(I_\pi)$. While an explicit description of I_π may not be simple, in cases where $\mathcal{O} \cong R[X]/\langle T\rangle$ –with T a monic polynomial– there is an explicit description of I_π that leads to a simple criterion for π-maximality, called the Dedekind criterion. The case where $\mathcal{O}$ is not necessarily of this form will be addressed in Section 5.6 for $R = \mathbf{Z}$.

5.5 Dedekind Criterion

We present a practical criterion involving only polynomial arithmetic that provides a simple way of establishing π-maximality in certain cases.

Let R be a PID, Q its fraction field, and $\mathcal{O} = R[X]/\langle T\rangle = R[\theta]$, with θ a root of T, a monic polynomial. For a prime $\pi \in R$, the quotient $R/\pi R$ is a field, and $(R/\pi R)[X]$ a PID. To describe the ideal I_π of $\mathcal{O}$, we consider the reduction of T modulo π. Denote by $T \pmod\pi$, or $\bar{T}$, the polynomial in $(R/\pi R)[X]$ obtained by mapping the coefficients of T through the reduction map $R \to R/\pi R$. If $g \in R[X]$ is a lift of the radical of $\bar{T}$ –meaning that $\bar{g}$ is the product of the distinct irreducible factors of $\bar{T}$ – then I_π can be expressed as $I_\pi = \pi\mathcal{O} + g(\theta)\mathcal{O}$. This leads to a criterion that guarantees $\mathcal{O} = r(I_\pi)$.

Definition 5.2. Let g in $R[X]$ be a lift of the radical of $\bar{T}$. Set h to be a lift of $\bar{T}/\bar{g}$ and define $f = (gh - T)/\pi$. We say that T *satisfies the Dedekind criterion* at π if $\gcd(\bar{f}, \bar{g}, \bar{h}) = 1$.

If K is a torsion-free R-algebra containing $\mathcal{O}$, then T satisfying the Dedekind criterion at π implies $\mathcal{O} = r(I_\pi)$. Combined with Theorem 5.1, this leads to the proposition:

Proposition 5.3. *Let K be torsion-free R-algebra containing $\mathcal{O} \cong R[X]/\langle T\rangle$. Let $\mathcal{O}'$ be an R-subalgebra of K, containing $\mathcal{O}$ and with the same rank. If T satisfies the Dedekind criterion at π, then $\mathcal{O}$ is π-maximal (i.e. π does not divide $[\mathcal{O}' : \mathcal{O}]$).*

To formalize the reduction modulo π of polynomials as in Definition 5.2, perhaps the obvious choice would be to use the quotient map from R to R / (Ideal.span {π}). However, for verifying the Dedekind criterion we need to compute in $(R/\pi R)[X]$, and thus in $R/\pi R$. Hence, flexibility in the representation of $R/\pi R$ is important. For instance, we prefer to use polynomials over ZMod p rather than $\mathbb{Z}$ / (Ideal.span {p}) as we have decidable equality and efficient modular arithmetic in the former type but not in the latter.

For this reason, we model the quotient $R/\pi R$ as some type (F : Type*) with [Field F] instance, along with a ring homomorphism q : R →+* F which is surjective and its kernel equals Ideal.span {π}. This ensures that $F \cong R/\pi R$. This approach of capturing the essential relationship between structures without fixing a specific construction is common in Mathlib as it allows for greater flexibility. In this context, a proof transfer framework such as [16] could also be useful for automating the translation of results involving different representations of $R/\pi R$, though we did not explore this further.

We formalized the Dedekind criterion as follows:

```
def satisfiesDedekindCriterion [Field F] (q : R →+* F)
    (π : R) (T : Polynomial R) : Prop :=
  ∃ (f g h : Polynomial R) (a b c : Polynomial F),
  IsRadicalPart (g mod π ) (T mod π)
  ∧ f * (C π) = g * h – T
  ∧ (a * (f mod π) + b * (g mod π) + c * (h mod π) = 1)
```

The notation f mod π stands for Polynomial.map q f, which applies q to the coefficients of f. This formulation of Definition 5.2 lets us avoid polynomial divisions and is more convenient, after providing the witness polynomials, for a proof by computation involving only simple operations. The proposition IsRadicalPart (g mod π) (T mod π) states that

$\bar{g}$ and $\bar{T}$ have the same prime divisors and $\bar{g}$ is squarefree. We can prove it by verifying that $\bar{g} \mid \bar{T}$, that $\bar{T} \mid \bar{g}^n$ for some n, and that $\bar{g}$ is squarefree. The latter can be done by checking that $\gcd(\bar{g}, \bar{g}') = 1$, with $\bar{g}'$ the formal derivative of $\bar{g}$.

Observe that the discussed assumptions on q : R →+* F are not needed to write down the previous definition. However, they do appear in the following theorem, which is the formal version of Proposition 5.3, adapted from [17]:

```
variable (q : R →+* F) (hqsurj : Function.Surjective q)
(hqker : RingHom.ker q = Ideal.span {π}) (hmc : 𝒪 ≤ 𝒪')
local notation "𝒪*" => Subalgebra.toSubmodule
      (Subalgebra.inclusion hmc).range
theorem piMaximal_of_satisfiesDedekindCriteria
    [Module.Free R 𝒪'] [Module.Finite R 𝒪']
    (j: IsAdjoinRoot 𝒪 T) (hp : Prime π ) (hm: T.Monic)
    (heqr : Module.rank R 𝒪= Module.rank R 𝒪')
    (h : satisfiesDedekindCriterion q π T) :
  piMaximal π 𝒪* := ...
```

Note that no assumption of irreducibility of T is required.

We also defined a specialized version of the Dedekind criterion for the case $R = \mathbf{Z}$ and F = ZMod p. We created a structure, called CertificateDedekindCriterionLists, with parameters (T : ℤ[X]) (p : ℕ). Its fields include data certifying that T satisfies the Dedekind criterion at p, and proofs of the associated identities –involving additions and multiplications and using our list-based approach– which can be solved automatically by rfl or decide.

The Dedekind criterion can still be useful even if $\mathcal{O}$ is not given by adjoining a root of a monic polynomial. If $\alpha \in \mathcal{O}$ has minimal polynomial $T \in R[X]$, then the R-subalgebra $R[\alpha] \subseteq \mathcal{O}$ is isomorphic to $R[X]/\langle T \rangle$. If T satisfies the Dedekind criterion at π and $\mathcal{O}'$ has rank equal to $\deg(T)$, then $R[\alpha]$, and hence $\mathcal{O}$, is π-maximal. Yet, for a number field K, the hope of using the Dedekind criterion to prove p-maximality of its ring of integers at every prime p soon fades. While this criterion always applies if $\mathcal{O}_K$ is monogenic, in some number fields the index $[\mathcal{O}_K : \mathbf{Z}[\alpha]]$ is divisible by a fixed prime for all $\alpha \in \mathcal{O}_K$. Dedekind gave an example of this [19].

5.6 The Non-monogenic Case

In this section, we introduce a certificate for local maximality which can be used even if the Dedekind criterion does not apply. We specialize to $R = \mathbf{Z}$, our main interest. Our formalization efforts were primarily focused on this case, with $\mathbf{F}_p$ modeled as ZMod p. However, we formalized many theorems about semilinear maps (introduced in Lean in [22]) and their kernels –which we use extensively– in a more general setting, laying the groundwork for future extensions.

The strategy used in the *Round-2 algorithm* is to reduce the computation of I_p and $r(I_p)$ to a computation in linear algebra over the field $\mathbf{F}_p$. We will use an analogous strategy to certify p-maximality by verifying that the kernel of certain linear map is trivial. Let $\mathcal{O}$ be a commutative ring which is free as a $\mathbf{Z}$-module and of finite rank r, and let p be a prime number. The quotient ring $\mathcal{O}/p\mathcal{O}$ is a finite dimensional $\mathbf{F}_p$-algebra. Similarly, the quotient I_p/pI_p is a finite dimensional $\mathbf{F}_p$-module (albeit not an $\mathbf{F}_p$- algebra). Consider the map from $\mathcal{O}/p\mathcal{O}$ to the $\mathbf{F}_p$-linear endomorphisms of I_p/pI_p.

$$\begin{aligned} \varphi : \mathcal{O}/p\mathcal{O} &\to \mathrm{End}(I_p/pI_p) \\ \bar{\alpha} &\mapsto (\bar{\beta} \to \overline{\alpha\beta}) \end{aligned} \tag{3}$$

This is the map in [17, Lemma 6.1.7], it is well defined and $\mathbf{F}_p$-linear. The following proposition follows directly from the definitions of the multiplier ring and the radical I_p and will be the basis for our certificate of p-maximality.

Proposition 5.4. *Let K be $\mathbf{Z}$-torsion-free ring containing $\mathcal{O}$. If the kernel of the map φ in (3) is trivial, then $r(I_p) = \mathcal{O}$.*

When K is a $\mathbf{Q}$-algebra, this is an if-and-only-if statement. If $\mathcal{O}'$ is a subring of K containing $\mathcal{O}$ and of equal rank, then the previous proposition together with Theorem (5.1) ensures that $\mathcal{O}$ is p-maximal if the kernel of φ is trivial. Therefore, finding a way to certify that φ has trivial kernel becomes our main goal.

To define (3) in Lean, we need to endow both I_p/pI_p and $\mathcal{O}/p\mathcal{O}$ with an $\mathbf{F}_p$-module structure. However, these quotient constructions are not exactly the same. If we want Lean to automatically infer that $\mathcal{O}/p\mathcal{O}$ has a CommRing instance, the quotient $\mathcal{O}/p\mathcal{O}$ should be constructed with $p\mathcal{O}$ of type Ideal $\mathcal{O}$ (which is definitionally equal to Submodule $\mathcal{O}$ $\mathcal{O}$). However, since I_p is not a ring (it lacks a one), pI_p cannot be treated as an ideal of I_p.

To unify these two constructions, we model M/nM for any abelian additive group M and a non-zero natural n, by using a ring R as a parameter (such that M is an R-module) and considering the quotient as a quotient of R-modules. We represent it as M / (n : R) · (⊤ : Submodule R M), where ⊤ is M viewed as a submodule of itself. We then define:

```
instance module_modp_is_zmodp_module
  (R M : Type*) (n : ℕ)[Module R M] [NeZero n] :
  Module (ZMod n) (M / (n : R) · ⊤) :=
```

Thus, I_p/pI_p (taking $M = I_p$ and $R = \mathbf{Z}$) and O/pO (taking $M = \mathcal{O}$ and $R = \mathcal{O}$) are both special cases of this construction, allowing Lean to infer an $\mathbf{F}_p$-module instance for each. Furthermore, Lean will automatically synthesize a CommRing instance for $\mathcal{O}/p\mathcal{O}$, which we then use to give it a $\mathbf{F}_p$-algebra structure. Another advantage of this unified approach is that we can define associated objects –such as a $\mathbf{Z}/n\mathbf{Z}$-basis for M/nM given a $\mathbf{Z}$-basis for M– and apply it to both cases. We also provide DistribMulAction and SMulCommClass instances that act as a shortcut for typeclass inference, and thereby prevent a timeout in downstream code.

To define map (3) in Lean, we used multiple steps. First, for a parameter (α : $\mathcal{O}$), we defined the map $I_p \to I_p/pI_p$ sending $\beta \mapsto \overline{\alpha\beta}$. Using Quotient.lift we lifted this to a map $I_p/pI_p \to I_p/pI_p$. This was bundled with its proofs of linearity

into a linear function, giving an element in $\mathrm{End}(I_p/pI_p)$. After defining the corresponding map $\mathcal{O} \to \mathrm{End}(I_p/pI_p)$, lifting again, and proving linearity, we obtain:

```
def map_to_end_lin (𝒪 : Type*) [CommRing 𝒪]
(p : ℕ) [Fact (Nat.Prime p)] :
𝒪 ⧸ ↑p • ⊤ →ₗ[ZMod p] ↥(Ideal.radical (↑p • ⊤)) ⧸ ↑p • ⊤
→ₗ[ZMod p] ↥(Ideal.radical (↑p • ⊤)) ⧸ ↑p • ⊤ := ...
```

The arrows have the following meanings. The expression ↑p appearing in 𝒪 ⧸ ↑p • ⊤ and Ideal.radical (↑p • ⊤) denotes the coercion of p : ℕ into 𝒪. The arrow ↥ coerces Ideal.radical (↑p • ⊤) into a type. Lastly, the rightmost appearance of ↑p in (Ideal.radical (↑p • ⊤)) ⧸ ↑p • ⊤ is the coercion of p : ℕ into ℤ.

Formally, Proposition 5.4 reads:

```
theorem
    mult_ring_eq_ring_of_trivial_ker_map_to_end_lin
  {K : Type*} [CommRing K] [NoZeroSMulDivisors ℤ K]
  (𝒪 : Subalgebra ℤ K) (p : ℕ) [hpI : Fact (Nat.Prime p)]
  (hk : LinearMap.ker (map_to_end_lin ↥𝒪 p) = ⊥) :
𝒪 = multiplierRing (Ideal.span {↑p}).radical :=
```

Let us return to the matter of certifying that the kernel of φ is trivial. Given a $\mathbf{Z}$-basis for an abelian additive group M, we obtain a $\mathbf{Z}/n\mathbf{Z}$-basis for M/nM by mapping it through the σ-semilinear map $M \to M/nM$, with $\sigma : \mathbf{Z} \to \mathbf{Z}/n\mathbf{Z}$. With these bases, the image of an element in M represented by $(x_1, \ldots, x_r) \in \mathbf{Z}^r$ will have coordinates $(\bar{x}_1, \ldots, \bar{x}_r) \in (\mathbf{Z}/n\mathbf{Z})^r$ in M/nM. In this way, we obtain an $\mathbf{F}_p$-basis for $\mathcal{O}/p\mathcal{O}$ from a $\mathbf{Z}$-basis for $\mathcal{O}$. Finding an $\mathbf{F}_p$-basis for I_p/pI_p is more complex. We will make use of the Frobenius endomorphism Frob_p, which, in a ring of characteristic p, is the $\mathbf{F}_p$-linear map $x \mapsto x^p$. For $t \in \mathbf{N}$, the *iterated* Frobenius endomorphism Frob_p^t sends $x \mapsto x^{p^t}$. The image of I_p under the quotient map $\mathcal{O} \to \mathcal{O}/p\mathcal{O}$ is $I_p/p\mathcal{O}$, an $\mathbf{F}_p$-subspace of $\mathcal{O}/p\mathcal{O}$. The following proposition lets us realize $I_p/p\mathcal{O}$ as the kernel of a linear map:

Proposition 5.5. *Let t be an integer such that $r \leq p^t$. The kernel of $\mathrm{Frob}_p^t : \mathcal{O}/p\mathcal{O} \to \mathcal{O}/p\mathcal{O}$ is equal to $I_p/p\mathcal{O}$.*

While we could theoretically find a matrix representation for this map, an algorithm for computing the kernel of a matrix has not yet, as far as we know, been implemented in Lean. However, we can *certify* that a given set of elements in $\mathcal{O}/p\mathcal{O}$ is a basis for the kernel of Frob_p^t as follows:

Denote by $\bar{x}$ the image in $\mathcal{O}/p\mathcal{O}$ of x in $\mathcal{O}$. Provide multisets $\mathcal{V} = \{v_1, \ldots, v_m\}$ and $\mathcal{W} = \{w_1, \ldots, w_n\}$ with elements in $\mathcal{O}$ –where $r = m + n$– such that $\overline{\mathcal{V}} = \{\bar{v}_1, \ldots, \bar{v}_m\}$ and $\mathcal{U} = \{\mathrm{Frob}_p^t(\bar{w}_1), \ldots, \mathrm{Frob}_p^t(\bar{w}_n)\}$ are $\mathbf{F}_p$-linearly independent and $\mathrm{Frob}_p^t(\bar{v}_i) = 0$ for all i. By a dimension argument, this guarantees that $\{\bar{v}_1, \ldots, \bar{v}_m\}$ is an $\mathbf{F}_p$-basis for the kernel of Frob_p^t – and thus for $I_p/p\mathcal{O}$.

We can represent $\overline{\mathcal{V}}$ as an $m \times r$ matrix over $\mathbf{F}_p$, the rows being the coordinates of the elements in $\overline{\mathcal{V}}$ with respect to the basis for $\mathcal{O}/p\mathcal{O}$. Since we are working over a field, we can *choose* the elements in $\mathcal{V}$ so that the corresponding matrix is in reduced row echelon form. The linear independence of $\overline{\mathcal{V}}$ can then be read off directly, without any determinant calculation (which is rather slow using the current Lean implementation). Similarly, the right choice of $\mathcal{W}$ will make the linear independence of $\mathcal{U}$ immediate to determine.

With the v_i and w_i as above, it can be shown that

$$\{\bar{v}_1, \ldots, \bar{v}_m, \overline{pw}_1, \ldots, \overline{pw}_n\} \subseteq I_p/pI_p \tag{4}$$

is an $\mathbf{F}_p$-basis for I_p/pI_p, with $\bar{y}$ the image of $y \in I_p$ in I_p/pI_p.

Many different quotients and reductions come into play in this last statement, this made our initial formalization attempt quite challenging. To address this, we proved it in a more general setting as basisSubmoduleModOfBasisMod. Instead of using the quotient construction, we worked with general types endowed with the appropiate instances, incorporating surjective maps with conditions on their kernels. This approach also allowed the construction to type-check faster. The result is a term of type Basis (Fin m ⊕ Fin n) S J (in this case, J models I_p/pI_p and S corresponds to $\mathbf{F}_p$).

Using Fin m ⊕ Fin n as the index type, which is the disjoint union of Fin m and Fin n, mimics our informal description and makes it more convenient to work with.

With a basis for I_p/pI_p, we can now represent an element in this space as an r-tuple, and an element in $\mathrm{End}(I_p/p_p)$ as an $r \times r$ matrix over $\mathbf{F}_p$. To *certify* that the kernel of the map φ in (3) is trivial, we can provide a multiset $\mathcal{X} = \{x_1, \ldots, x_r\}$ of elements in $\mathcal{O}$ such that $\varphi(\overline{\mathcal{X}}) = \{\varphi(\bar{x}_1), \ldots, \varphi(\bar{x}_r)\}$ is linearly independent. This last verification can be simplified by observing that, generically, we expect to find an element $\beta_W \in I_p/pI_p$ such that $\{\varphi(\bar{x}_1)(\beta_W), \ldots, \varphi(\bar{x}_r)(\beta_W)\}$ is linearly independent. This β_W acts as a witness of the linear independence of $\varphi(\overline{\mathcal{X}})$. Furthermore, we can *choose* $\mathcal{X}$ in a way that makes the linear independence of the multiset easy to verify. This leads us to the following certificate for p-maximality.

Certificate. Consider $\mathcal{O}$ and $\mathcal{O}'$ subalgebras of a $\mathbf{Z}$-torsion-free commutative ring such that $\mathcal{O} \subseteq \mathcal{O}'$, and both of rank r. Let $\mathcal{B} = \{b_1, \ldots, b_r\}$ be a $\mathbf{Z}$-basis for $\mathcal{O}$. Let m, n and t be non-negative integers with $r = m + n$ and $r \leq p^t$. A certificate for p-maximality consists of the following data:

- $m \times r$ matrix $\mathcal{V}$ over $\mathbf{Z}$
- $n \times r$ matrix $\mathcal{W}$ over $\mathbf{Z}$
- $n \times r$ matrix $\mathcal{U}$ over $\mathbf{F}_p$
- m-tuple ν over $\mathbf{Z}_{>0}$
- n-tuple ω over $\mathbf{Z}_{>0}$
- $r \times r$ matrix $\mathcal{X}$ over $\mathbf{Z}$
- m-tuple β over $\mathbf{Z}$
- n-tuple γ over $\mathbf{Z}$
- $r \times m$ matrix $\mathbf{a}$ over $\mathbf{Z}$
- $r \times n$ matrix $\mathbf{c}$ over $\mathbf{Z}$
- r-tuple η over the disjoint union $A \sqcup B$ with $A = \{1, \ldots, m\}$ and $B = \{1, \ldots, n\}$.

For an integer s, let $\bar{s}$ be its reduction modulo p in $\mathbf{F}_p$. For b in $\mathcal{O}$, let $\bar{b}$ be its reduction in $\mathcal{O}/p\mathcal{O}$. For verification, check that:

(i) For all i, $\overline{\mathcal{V}}_{i\nu_i} \neq 0$ and $\overline{\mathcal{V}}_{j\nu_i} = 0$ for all $j \neq i$.

(ii) For all i, $\mathcal{U}_{i\omega_i} \neq 0$ and $\mathcal{U}_{j\omega_i} = 0$ for all $j \neq i$.
(iii) For all i,
- if η_i is in A: $\overline{a_{i\eta_i}} \neq 0$ and $\forall\, j \neq i$, $\overline{a_{j\eta_i}} = 0$;
- if η_i is in B: $\overline{c_{i\eta_i}} \neq 0$ and $\forall\, j \neq i$, $\overline{c_{j\eta_i}} = 0$.

(iv) For all i, $(\sum_j \overline{\mathcal{V}_{ij}} \cdot \overline{b_j})^{p^t} = 0$.
(v) For all i, $(\sum_j \overline{\mathcal{W}_{ij}} \cdot \overline{b_j})^{p^t} = \sum_j \mathcal{U}_{ij} \cdot \overline{b_j}$.
(vi) For all i :

$$(\textstyle\sum_j \mathcal{X}_{ij} \cdot b_j)(\sum_k \left(\beta_k \cdot \sum_j \mathcal{V}_{kj} \cdot b_j\right) + \sum_k \left(p\gamma_k \cdot \sum_j \mathcal{W}_{kj} \cdot b_j\right)) = \sum_k \left(a_{ik} \cdot \sum_j \mathcal{V}_{kj} \cdot b_j\right) + \sum_k \left(pc_{ik} \cdot \sum_j \mathcal{W}_{kj} \cdot b_j\right).$$

The rows of $\mathcal{X}$, $\mathcal{V}$ and $\mathcal{W}$ represent elements in $\mathcal{O}$ as remarked before. The matrix $\mathcal{U}$ represents elements in $\mathcal{O}/p\mathcal{O}$. Statement (i) and (ii) prove that $\overline{\mathcal{V}}$ and $\mathcal{U}$ are linearly independent. This, together with (iv) and (v) guarantee that the collection as in (4) is a basis for I_p/pI_p. The tuples γ and β give the coordinates of a witness element β_W in I_p/pI_p with respect to this basis. Statement (vi) together with (iii) imply that $\{\varphi(\bar{x}_1)(\beta_W), \ldots, \varphi(\bar{x}_r)(\beta_W)\}$ is linearly independent, establishing the p-maximality of $\mathcal{O}$.

We formalized this certification scheme in Lean as:

```
structure MaximalOrderCertificateWLists
  {K : Type*} [CommRing K] [NoZeroSMulDivisors ℤ K]
  (p : ℕ) [hpI : Fact $ Nat.Prime p] (𝒪 : Subalgebra ℤ K)
  (𝒪' : Subalgebra ℤ K) (hm : 𝒪 ≤ 𝒪') where ...
```

It includes as part of its fields a term of type TimesTable (Fin (m + n)) ℤ 𝒪, a term of type Basis (Fin (m + n)) ℤ 𝒪′ (which we do not know explicitly, of course, but serves as a proof that, as a $\mathbf{Z}$-module, $\mathcal{O}'$ is free, finite and has rank r), and the certifying data described earlier, together with proofs of the verification statements.

The statements (iv), (v), and (vi), which involve r identities in $\mathcal{O}/p\mathcal{O}$ and r in $\mathcal{O}$, are stated in terms of lists over $\mathbf{F}_p$ and $\mathbf{Z}$, respectively, as described in Section 5.2. The proofs can then be solved using decide and/or decide!. From a term of this type, we get a proof of piMaximal ↑p 𝒪*.

In case a witness β_W does not exist or cannot be found, we formalized a longer certification scheme for which a certificate always exists when K is a number field, $\mathcal{O}' = \mathcal{O}_K$, and $\mathcal{O}$ is p-maximal. This requires verifying r^2 identities in $\mathcal{O}$. For details, we refer to Appendix A. Additionally, in cases where $m = 0$ (corresponding to p being *unramified* in K) a simplified certificate is available.

We wrote `CertificatePMaximalityF` in SageMath, to compute these certificates and write the corresponding Lean term.

5.7 Global Maximality

Here, we describe how we can automatically generate a Lean proof of $\mathcal{O} = \mathcal{O}_K$ using the certificates previously introduced.

Let K be the number field obtained by adjoining a root θ of the monic irreducible polynomial T in $\mathbf{Z}[X]$. We aim to verify that a subalgebra $\mathcal{O}$, with an explicitly given basis, is equal to $\mathcal{O}_K$. Since T is separable, we can find $a, b \in \mathbf{Z}[X]$ such that $aT + bT' = n$ for some nonzero integer n. If p does not divide n, then $T \pmod p$ is squarefree and it easily follows that T satisfies the Dedekind criterion at p. Given $\theta \in \mathcal{O}$, this gives a quick way to prove p-maximality of $\mathcal{O}$ for *all* but finitely many primes p (those dividing n). In fact, T may still satisfy the criterion for some of these primes. For the rest, the certificates in Section 5.6 can be used, leading to a proof of $\mathcal{O} = \mathcal{O}_K$.

We wrote a SageMath function, called `LeanProof`, that takes as input an irreducible and monic polynomial $T \in \mathbf{Z}[X]$, a $\mathbf{Z}$-basis $\{w_i\}_i$ for the ring of integers of $K = \mathbf{Q}[X]/\langle T \rangle$, and outputs a Lean proof of :

```
𝒪 = NumberField.RingOfIntegers K
```

where K is AdjoinRoot (map (algebraMap ℤ ℚ) T) , and $\mathcal{O}$ is the subalgebra of K with integral basis $\{w_i\}_i$, constructed as in Section 5.1. The proof of irreducibility of T is generated as in Section 4. Lean can then infer a NumberField K instance. The term NumberField.RingOfIntegers K is simply the coercion of IntegralClosure ℤ K into a type. As mentioned, while generating Lean code from SageMath was sufficient for our purposes, tighter integration between the CAS and Lean remains an interesting direction for future work.

The previous approach avoids any computation of the discriminant (defined in the next section) in Lean. However, a proof of the discriminant of $\mathcal{O}$, and consequently of K, can help narrow down the primes one needs to check. These concept, and its computation, are the subject of the next section.

6 Resultants and Discriminants

The goal of this section is to provide a method for computing the discriminant of an integral basis of $\mathcal{O}$. Consider an R-algebra A with a finite R-basis. Given an element $x \in A$, the map $y \mapsto xy$ is an R-linear map, and thus can be represented as a matrix with entries in R. The trace of this matrix, which is independent of the choice of basis, is known as the *trace* of x and denoted by $\mathrm{Tr}_{A/R}(x)$. For a collection of elements $a_1, \ldots, a_n$ of A, its *discriminant* is defined by $\mathrm{disc}(a_1, \ldots, a_n) = \det(\mathrm{Tr}_{A/R}(a_i a_j))_{ij}$. When we started our project, Mathlib already contained this definition.

If $K = \mathbf{Q}(\alpha)$ is a number field of degree n and $\mathcal{O}$ is a subring which is free of rank n as a $\mathbf{Z}$-module, we define $\mathrm{disc}(\mathcal{O}) \in \mathbf{Z}$ as the discriminant of a $\mathbf{Z}$-basis for $\mathcal{O}$. In particular, $\mathrm{disc}(\mathcal{O}_K)$ is known as the *discriminant* of the number field K and it is an important arithmetic invariant encoding information about K. If $\alpha \in \mathcal{O}$, then we have

$$\mathrm{disc}(1, \alpha, \ldots, \alpha^{n-1}) = [\mathcal{O} : \mathbf{Z}[\alpha]]^2 \cdot \mathrm{disc}(\mathcal{O}). \tag{5}$$

The index $[\mathcal{O} : \mathbf{Z}[\alpha]]$ can be determined from an explicit basis for $\mathcal{O}$ in terms of α. Knowing the discriminant of the power basis $1, \alpha, \ldots, \alpha^{n-1}$ then allows us to compute $\mathrm{disc}(\mathcal{O})$.

More generally, let $K = F(\alpha)$ be a finite separable field extension of F (e.g. K a number field and $F = \mathbb{Q}$). Letting the conjugate roots of α in the algebraic closure be numbered $\alpha_1, \ldots, \alpha_n$, then the discriminant of $1, \alpha, \ldots, \alpha^{n-1}$ is equal to $\prod_i \prod_{j>i} (\alpha_j - \alpha_i)^2$. This result was already available in Mathlib. However, the value of the discriminant is not directly computable. Thus, we define a computable notion of discriminant for an explicitly given polynomial.

In the remainder of this section, let R be an integral domain, f, g be polynomials of degree n and m respectively with coefficients given by $f(X) = a_n X^n + \cdots + a_1 X + a_0 \in R[X]$ and $g(X) = b_m X^m + \cdots + b_1 X + b_0 \in R[X]$. Our path towards formalized computation of the discriminant starts with the *resultant* $\mathrm{Res}(f, g)$ defined as the determinant of the *Sylvester matrix* of dimension $(m + n) \times (m + n)$, determined by the coefficients of f and g (see [30, IV, §8] for a transposed version with f and g interchanged and coefficients reversed); we work out an example in Appendix B. Let f be a polynomial with formal derivative f', then $\mathrm{Res}(f, f')$ is divisible by the leading coefficient a_n of f; one sees this by inspection of the last row of the Sylvester matrix, which contains two nonzero entries: a_n and na_n. We define the discriminant of a polynomial f with leading coefficient a_n such that $a_n \mathrm{Disc}(f) = (-1)^{n(n-1)/2} \mathrm{Res}(f, f')$, by modifying the Sylvester matrix to divide the last row by a_n, and multiplying the determinant of this modified matrix by the desired sign.

Our definition of the resultant allows for direct evaluation, but we need the relation to the discriminant of a power basis. The main theorem we proved expresses the resultant of two polynomials in terms of their roots:

Theorem 6.1. *Let $f, g \in R[X]$ be polynomials and assume they split completely: $f(X) = a_n \prod_i (X - \alpha_i)$ and $g(X) = b_m \prod_i (X - \beta_i)$, then:*

$$\mathrm{Res}(f, g) = a_n^m b_m^n \prod_i \prod_j (\alpha_i - \beta_j).$$

Note that the assumption of Theorem 6.1 always applies after mapping f and g to the algebraic closure of the fraction field of R, and this map will preserve the resultant, so we will assume that f and g split completely. It then follows from the product rule and elementary algebraic manipulations that, for f monic with roots α_i, $\prod_i \prod_{j>i} (\alpha_j - \alpha_i)^2 = \mathrm{Disc}(f)$. Since the former term is shown in Mathlib to equal the discriminant of the power basis $1, \ldots, \alpha^{n-1}$, and the latter term is computable, this gives us a way to compute, given a basis for $\mathcal{O}_K$ and using (5), the discriminant of a number field.

It remains to prove Theorem 6.1. We first showed that for two homogeneous polynomials $p, q \in R[x, y, \ldots, z]$ of the same degree: if p divides q, then they are equal up to a multiplicative constant: $q = cp$ for $c \in R$. This required formalizing a substantial amount of preliminaries in the theory of (homogeneous) multivariate polynomials.

Next, we may take f and g monic and view $\mathrm{Res}(f, g)$ and $\prod_i \prod_j (\alpha_i - \beta_j)$ as polynomials in $R[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m]$. We showed that these are indeed homogeneous polynomials, of degree mn, where an essential ingredient was characterizing the homogeneous polynomials $p \in R[x, y, \ldots, z]$ as those polynomials where $p(cx, cy, \ldots, cz) = c^k p(x, y, \ldots, z)$, assuming R is infinite. The coefficient of $(\alpha_1 \alpha_2 \cdots \alpha_n)^m$ in $\mathrm{Res}(f, g)$ and in $\prod_i \prod_j (\alpha_i - \beta_j)$ equals 1, so if the two expressions are equal up to multiplication by c, we must have $c = 1$ and the expressions are actually equal.

Finally, our approach to showing $\prod_i \prod_j (\alpha_i - \beta_j)$ divides $\mathrm{Res}(f, g)$ was by showing $\alpha_i - \beta_j$ divides $\mathrm{Res}(f, g)$ for each i, j. We proved that the latter statement is equivalent, under the condition that R is infinite, to the statement that $\mathrm{Res}(f, g) = 0$ if f and g share a root. To show the latter statement, we considered the R-linear map $(p, q) \mapsto (pf + qg)$ on the subspace $M = \{(p, q) \in R[X] \times R[X] \mid \deg p < m \text{ and } \deg q < n\}$, and computed that the determinant of this map is exactly equal to the resultant. If f and g have a common root, then $(p, q) \mapsto (pf + qg)$ has a nontrivial kernel, so the determinant is zero.

Although this approach does not differ significantly from one found in textbooks such as Lang's Algebra [30], a particular source of difficulty we had to deal with was the textbooks' implicitly shifting point of view between different polynomial rings. For example, Lang argues that $\mathrm{Res}(f, g)$ is homogeneous of degree m as a polynomial in $R[a_0, \ldots, a_n]$ and of degree n as a polynomial in $R[b_0, \ldots, b_m]$, but that $\alpha_i - \beta_j$ divides $\mathrm{Res}(f, g)$ in $R[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m]$. Rather than having to repeat the boilerplate involved with a change of rings repeatedly throughout our proofs, we chose to adapt the proofs from the literature to remain within one ring. Our formalization works exclusively in the polynomials $R[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m]$. For example, polynomials that are expressed in terms of $a_0, \ldots, a_n, b_0, \ldots, b_m$ can be translated to $R[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m]$ using the monadic bind operator [18], since each coefficient a_i can be given as a polynomial in the roots α_j.

To evaluate discriminants, we use the Polynomial.ofList definition of Section 4.1. Expressions such as resultant (ofList [5, 4, 3, 2, 1]) (ofList [4, 6, 6, 4]) are computable, although we run into a subsequent error: the implementation of determinants in Mathlib is not optimized for computation and therefore causes a stack overflow. However, we were still able to compute the discriminant of degree 3 polynomials such as $f = X^3 - 30X - 80$ and, after formally verifying an integral basis for the ring of integers of $K = \mathbb{Q}[X]/\langle f \rangle$, proved

```
theorem K_discr : NumberField.discr K = -16200 := ...
```

7 Formally Verifiying LMFDB Entries

A popular publicly available online database containing number theoretic data is the *L-functions and modular forms database* (LMFDB) [34]. It gathers data from various sources, relying on several programming languages and CASes. Much of the data is currently beyond the reach of formalization, but

our work makes it possible to formally verify some nontrivial entries for *number fields* of arbitrary degree. In particular, we focus on checking the *integral basis* for a choice of number fields up to degree 8. To a lesser extent, we also compute the corresponding *discriminant*, but only up to degree 3, as our computational methods are currently still under development. At this point the aim is not to formally verify integral bases and discriminants for a large fraction of the LMFDB, but to show the feasibility, and current computational limits, of the approach.

Concretely, there are 142 number fields in the LMFDB which are of degree 5, unramified outside $2, 3, 5$, and have (to focus on complications) non-monogenic ring of integers. (This list, up to isomorphism, is complete, but this is not our concern here.) For all these 142 number fields we successfully used our tools to formally verify the given *integral basis*. The most time-consuming step when checking these proofs appears to be the exponentiation step in the certificates of Section 5.6. We recall that this is done with repeated squaring and multiplication. Using a virtual machine running an AMD EPYC 7B13 processor (2.45 GHz) and 16GB of RAM, each ring of integers took, on average, around 33 seconds to check. Approximately 14% of that time is spent on the irreducibility proof for the defining polynomial. Replacing decide and decide! with native_decide whenever possible reduces the overall checking time by about 20%. During development, we encountered a couple of examples that required more than the default number of maximum heartbeats (200,000) for type-checking, and the proofs had to be manually adjusted. This was resolved after a Mathlib update.

As our discriminant computations cannot yet handle large degrees, we also tested degree 3 cases. There are 7 number fields in the LMFDB which are of degree 3, unramified outside $2, 3, 5$, and have non-monogenic ring of integers. For all these number fields we successfully formally verify the given *integral basis* as well as the *discriminant*. We note that in all of these cases, the longer form of the certificate for p-maximality (given in detail in Appendix A) had to be used. In contrast, for all of degree 5 examples except one, the shorter certificate with a witness was sufficient. In this case, each ring of integers took, on average, around 28 seconds to check. Using native_decide decreases this time by around 10%.

We also successfully verified a degree 6 example and a simpler degree 8 example. Both took around 48 seconds to check. For these, native_decide reduces the checking time by 61% and 32%, respectively.

8 Discussion

8.1 Future Work

The discriminant computations should be improved. At this point, these are directly computed from the corresponding resultant. For repeated use of discriminant evaluation of a fixed degree, general formulae for the discriminant of a degree n polynomial should be formalized, up to some 'reasonable' n. The computations for the ring of integers are in a more advanced stage, though further optimization is needed to deal with higher degrees in order to be able to cover all number fields from the LMFDB. We note that much of the mathematical foundation is formalized in enough generality to allow for possible extension to more general contexts. Although not essential for database verification – since all necessary Lean files can be generated in advance– integrating SageMath and Lean, for instance by calling our SageMath scripts directly from Lean, could be convenient for on-demand computations. Finally, certifying other fundamental invariants, such as class groups and unit groups, would be very interesting. While these would constitute completely new projects, they could build on the formalized rings of integers.

8.2 Related Work

As mentioned above, some rings of integers for quadratic number fields have been formalized before [5]; this concerns both the determination as well as concrete computation. Specific instances, such as the Gaussian integers $\mathbf{Z}[i]$, have been considered in various systems; see e.g. [23, 25]. Mathematical certificates outside of algebraic number theory have received much attention. Some examples relevant for (general) number theory and computer algebra include [35, 43]. Previous work bridging Lean with Computer Algebra Systems includes the tactics polyrith, written by Dhruv Bhatia, and sageify [5], both of which integrate with SageMath. Additionally, [32, 33] develop a connection between Lean and Mathematica. The use of a proof assistant as a CAS itself has also been explored in prior work. In [20], efficient algorithms in linear algebra and polynomial arithmetic are implemented and verified in Coq. Similarly, [2, 3] implements various linear algebra algorithms in Isabelle/HOL.

8.3 Conclusions

We have successfully built tools to formally verify rings of integers in number fields of, in principle, arbitrary degree, as well as set up the basis for computing discriminants. This represents a step towards bridging the gap between traditional Computer Algebra Systems and formal verification. Developing some of the actual certification schemes, especially for the ring of integers, required several new ideas. While the formalization of the mathematical theory proved challenging at times, a major part of the effort involved translating the abstract mathematical concepts and results into a representation suitable for computation.

In total, for our developments, we wrote about 13000 lines of Lean code. Our SageMath scripts automatically generated about 32000 lines of Lean code (excluding the alternative code containing native_decide) to perform the certifications we considered.

With these kind of projects, it is very hard to reliably measure the De Bruijn factor.

A Certificate for p-maximality

The following certificate is guaranteed to always exist if K is a number field, $\mathcal{O}' = \mathcal{O}_K$, and $\mathcal{O}$ is p-maximal.

Certificate

Consider $\mathcal{O}$ and $\mathcal{O}'$ subalgebras of a $\mathbf{Z}$-torsion-free commutative ring such that $\mathcal{O} \subseteq \mathcal{O}'$, and both of rank r. Let $\mathcal{B} = \{b_1, \ldots, b_r\}$ be a $\mathbf{Z}$-basis for $\mathcal{O}$. Let m, n and t be non-negative integers with $r = m + n$ and $r \leq p^t$. A certificate for p-maximality consists of the following data:

- $m \times r$ matrix $\mathcal{V}$ over $\mathbf{Z}$
- $n \times r$ matrix $\mathcal{W}$ over $\mathbf{Z}$
- $n \times r$ matrix $\mathcal{U}$ over $\mathbf{F}_p$
- m-tuple v over $\mathbf{Z}_{>0}$
- n-tuple ω over $\mathbf{Z}_{>0}$
- $r \times r$ matrix $\mathcal{X}$ over $\mathbf{Z}$
- $r \times m \times m$ array $\mathbf{a}$ over $\mathbf{Z}$.
- $r \times m \times n$ array $\mathbf{c}$ over $\mathbf{Z}$.
- $r \times n \times m$ array $\mathbf{d}$ over $\mathbf{Z}$.
- $r \times n \times n$ array $\mathbf{e}$ over $\mathbf{Z}$.
- r pairs (η_i, η_i') with η_i' and η_i in the disjoint union $A \sqcup B$ with $A = \{1, \ldots, m\}$ and $B = \{1, \ldots, n\}$

For an integer s, let $\bar{s}$ be its reduction modulo p in $\mathbf{F}_p$. For b in $\mathcal{O}$, let $\bar{b}$ be its reduction in $\mathcal{O}/p\mathcal{O}$. For verification, check that:

(i) For all i, $\overline{\mathcal{V}}_{iv_i} \neq 0$ and $\overline{\mathcal{V}}_{jv_i} = 0$ for all $j \neq i$
(ii) For all i, $\mathcal{U}_{i\omega_i} \neq 0$ and $\mathcal{U}_{j\omega_i} = 0$ for all $j \neq i$
(iii) For all i,
- If η_i and η_i' in A, $\overline{a_{i\eta_i\eta_i'}} \neq 0$ and $\forall\, j \neq i$, $\overline{a_{j\eta_i\eta_i'}} = 0$.
- If η_i in A and η_i' in B, $\overline{c_{i\eta_i\eta_i'}} \neq 0$ and $\forall\, j \neq i$, $\overline{c_{j\eta_i\eta_i'}} = 0$.
- If η_i in B and η_i' in A, $\overline{d_{i\eta_i\eta_i'}} \neq 0$ and $\forall\, j \neq i$, $\overline{d_{j\eta_i\eta_i'}} = 0$.
- If η_i and η_i' in B, $\overline{e_{i\eta_i\eta_i'}} \neq 0$ and $\forall\, j \neq i$, $\overline{e_{j\eta_i\eta_i'}} = 0$.

(iv) For all i, $(\sum_j \overline{\mathcal{V}_{ij}} \cdot \overline{b_j})^{p^t} = 0$
(v) For all i, $(\sum_j \overline{\mathcal{W}_{ij}} \cdot \overline{b_j})^{p^t} = \sum_j \mathcal{U}_{ij} \cdot \overline{b_j}$.
(vi) For all i and j :

$$(\textstyle\sum_l \mathcal{X}_{il} \cdot b_l)(\sum_l \mathcal{V}_{jl} \cdot b_l) = \sum_k \left(a_{ijk} \cdot \sum_l \mathcal{V}_{kl} \cdot b_l\right) + \sum_k \left(pc_{ijk} \cdot \sum_l \mathcal{W}_{kl} \cdot b_l\right).$$

(vii) For all i and j:

$$(\textstyle\sum_l \mathcal{X}_{il} \cdot b_l)(\sum_l p\mathcal{W}_{jl} \cdot b_l) = \sum_k \left(d_{ijk} \cdot \sum_l \mathcal{V}_{kl} \cdot b_l\right) + \sum_k \left(pe_{ijk} \cdot \sum_l \mathcal{W}_{kl} \cdot b_l\right).$$

The rows of $\mathcal{X}$, $\mathcal{V}$ and $\mathcal{W}$ represent elements in $\mathcal{O}$. The matrix $\mathcal{U}$ represents elements in $\mathcal{O}/p\mathcal{O}$. Statement (i) and (ii) prove that $\overline{\mathcal{V}}$ and $\mathcal{U}$ are linearly independent. This, together with (iv) and (v) guarantee that the collection as in (4) is a basis for I_p/pI_p. Statements (vi) and (vii) imply that the matrix representing the endomorphism $\varphi(\bar{x}_i)$ with respect to this basis is given by:

$$\begin{pmatrix} \overline{a_{i11}} & \cdots & \overline{a_{im1}} & \overline{d_{i11}} & \cdots & \overline{d_{in1}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \overline{a_{i1m}} & \cdots & \overline{a_{imm}} & \overline{d_{i1m}} & \cdots & \overline{d_{inm}} \\ \overline{c_{i11}} & \cdots & \overline{c_{im1}} & \overline{e_{i11}} & \cdots & \overline{e_{in1}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \overline{c_{i1n}} & \cdots & \overline{c_{imn}} & \overline{e_{i1n}} & \cdots & \overline{e_{inn}}. \end{pmatrix} \tag{6}$$

From (iii) the linear independence of $\{\varphi(\bar{x}_1), \ldots, \varphi(\bar{x}_r))\}$ follows, establishing the p-maximality of $\mathcal{O}$.

We formalized this certificate in Lean as the structure

```
structure MaximalOrderCertificateLists
{K : Type*} [CommRing K] [NoZeroSMulDivisors ℤ K]
(p : ℕ) [hpI : Fact $ Nat.Prime p] (𝒪 : Subalgebra ℤ K)
  (𝒪' : Subalgebra ℤ K) (hm : 𝒪 ≤ 𝒪') where ...
```

B Sylvester matrix computation examples

Consider the polynomial $f = 2X^3 - X^2 - 2X + 1 \in \mathbf{Z}[X]$, which has derivative $f' = 6X^2 - 2X - 2$. The Sylvester matrix for f and f' is given by:

$$A = \begin{pmatrix} -2 & 0 & 0 & 1 & 0 \\ -2 & -2 & 0 & -2 & 1 \\ 6 & -2 & -2 & -1 & -2 \\ 0 & 6 & -2 & 2 & -1 \\ 0 & 0 & 6 & 0 & 2 \end{pmatrix}.$$

The resultant is defined as the determinant of the Sylvester matrix, giving us $\mathrm{Res}(f, f') = |A| = -72$.

To compute the discriminant of f we modify the Sylvester matrix by dividing out the leading coefficient of f, 2, in the last row:

$$A = \begin{pmatrix} -2 & 0 & 0 & 1 & 0 \\ -2 & -2 & 0 & -2 & 1 \\ 6 & -2 & -2 & -1 & -2 \\ 0 & 6 & -2 & 2 & -1 \\ 0 & 0 & 3 & 0 & 1 \end{pmatrix}.$$

We obtain $\mathrm{Disc}(f) = (-1)^{\deg f(\deg f - 1)/2}|A'| = (-1)^3 \cdot -36 = 36$.

Passing to the field of fractions, we can divide f by its leading coefficient to get the monic polynomial $g = X^3 - \frac{1}{2}X^2 - X + \frac{1}{2} \in \mathbf{Q}[X]$, with discriminant computed as before $\mathrm{Disc}(g) = \frac{9}{4} = \left(\frac{1}{2}\right)^{2(\deg f - 1)} \mathrm{Disc}(f)$. The roots of f and g in $\mathbf{Q}$ are given by $X \in \{-1, \frac{1}{2}, 1\}$, and we find as expected that the product of root differences equals the discriminant of g:

$$\left(\frac{1}{2} - -1\right)^2 (1 - -1)^2 \left(1 - \frac{1}{2}\right)^2 = \frac{9}{4}.$$

Data-Availability Statement

Full source code of our formalization and SageMath scripts are available online at https://github.com/alainchmt/RingOfIntegersProject and are archived as [15].

Acknowledgments

Alain Chavarri Villarello was funded by NWO Vidi grant No. 613.009.143, Formalizing Diophantine algorithms. We thank Assia Mahboubi, Filippo A. E. Nuccio and the anonymous referees for their helpful comments.

References

[1] John Abbott. [2020] ©2020. Certifying irreducibility in $\mathbb{Z}[x]$. In *Mathematical software—ICMS 2020.* Lecture Notes in Comput. Sci.,

Vol. 12097. Springer, Cham, 462–472. https://doi.org/10.1007/978-3-030-52200-1_46

[2] Jesús Aransay and Jose Divasón. 2016. Formalisation of the computation of the echelon form of a matrix in Isabelle/HOL. *Form. Asp. Comput.* 28, 6 (2016), 1005–1026. https://doi.org/10.1007/s00165-016-0383-1

[3] Jesús Aransay and Jose Divasón. 2017. A formalisation in HOL of the fundamental theorem of linear algebra and its application to the solution of the least squares problem. *J. Automat. Reason.* 58, 4 (2017), 509–535. https://doi.org/10.1007/s10817-016-9379-z

[4] Anne Baanen. [2020] ©2020. A lean tactic for normalising ring expressions with exponents (short paper). In *Automated reasoning. Part II*. Lecture Notes in Comput. Sci., Vol. 12167. Springer, Cham, 21–27. https://doi.org/10.1007/978-3-030-51054-1_2

[5] Anne Baanen, Alex J. Best, Nirvana Coppola, and Sander R. Dahmen. 2023. Formalized Class Group Computations and Integral Points on Mordell Elliptic Curves. In *CPP 2023*, Robbert Krebbers, Dmitriy Traytel, Brigitte Pientka, and Steve Zdancewic (Eds.). ACM, 47–62. https://doi.org/10.1145/3573105.3575682

[6] Anne Baanen, Sander R. Dahmen, Ashvni Narayanan, and Filippo A. E. Nuccio Mortarino Majno di Capriglio. 2022. A Formalization of Dedekind Domains and Class Groups of Global Fields. *J. Autom. Reason.* 66, 4 (2022), 611–637. https://doi.org/10.1007/s10817-022-09644-0

[7] Wieb Bosma, John Cannon, and Catherine Playoust. 1997. The Magma algebra system. I. The user language. *J. Symbolic Comput.* 24, 3-4 (1997), 235–265. https://doi.org/10.1006/jsco.1996.0125 Computational algebra and number theory (London, 1993).

[8] Viktor Bouniakowsky. 1854. *Sur les diviseurs numériques invariables des fonctions rationnelles entieres.* De l'Imprimerie de l'Académie impériale des sciences.

[9] Samuel Boutin. 1997. Using reflection to build efficient and certified decision procedures. In *TACS 1997 (LNCS, Vol. 1281)*, Martín Abadi and Takayasu Ito (Eds.). Springer, Berlin, Heidelberg, 515–529. https://doi.org/10.1007/11541868_7

[10] John Brillhart. 1980. Note on irreducibility testing. *Math. Comp.* 35, 152 (1980), 1379–1381. https://doi.org/10.2307/2006403

[11] Thomas Browning and Patrick Lutz. 2022. Formalizing Galois Theory. *Exp. Math.* 31, 2 (2022), 413–424. https://doi.org/10.1080/10586458.2021.1986176

[12] J. A. Buchmann and H. W. Lenstra, Jr. 1994. Approximating rings of integers in number fields. *J. Théor. Nombres Bordeaux* 6, 2 (1994), 221–260. http://jtnb.cedram.org/item?id=JTNB_1994__6_2_221_0

[13] Mario Carneiro. 2019. The Type Theory of Lean. https://github.com/digama0/lean-type-theory/releases MSc thesis.

[14] Augustin Louis Baron Cauchy. 1829. *Exercices de mathématiques.* Œuvres 2, Vol. 9. 122 pages.

[15] Alain Chavarri Villarello, Sander Dahmen, and Anne Baanen. 2024. *Certifying rings of integers in number fields.* https://doi.org/10.5281/zenodo.14283856

[16] Cyril Cohen, Enzo Crance, and Assia Mahboubi. 2024. Trocq: proof transfer for free, with or without univalence. In *European Symposium on Programming*. Springer, 239–268.

[17] Henri Cohen. 1993. *A course in computational algebraic number theory.* Graduate Texts in Mathematics, Vol. 138. Springer. xii+534 pages. https://doi.org/10.1007/978-3-662-02945-9

[18] Johan Commelin and Robert Y. Lewis. 2021. Formalizing the ring of Witt vectors. In *CPP '21 (Certified Programs and Proofs)*, Catalin Hriţcu and Andrei Popescu (Eds.). ACM, 264–277. https://doi.org/10.1145/3437992.3439919

[19] R. Dedekind. 1878. Ueber den Zusammenhang zwischen der Theorie der ideale und der Theorie der höheren Congruenzen. *Abhandlungen der Königlichen Gesellschaft der Wissenschaften in Göttingen* 23 (1878), 3–38. http://eudml.org/doc/135827

[20] Maxime Dénès, Anders Mörtberg, and Vincent Siles. 2012. A refinement-based approach to computational algebra in Coq. In *Interactive theorem proving*. Lecture Notes in Comput. Sci., Vol. 7406. Springer, Heidelberg, 83–98. https://doi.org/10.1007/978-3-642-32347-8_7

[21] David S. Dummit and Richard M. Foote. 2004. *Abstract algebra* (third edition ed.). John Wiley & Sons, Inc., Hoboken, NJ, USA. xii+932 pages.

[22] Frédéric Dupuis, Robert Y. Lewis, and Heather Macbeth. 2022. Formalized functional analysis with semilinear maps. In *13th International Conference on Interactive Theorem Proving*. LIPIcs. Leibniz Int. Proc. Inform., Vol. 237. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, Art. No. 10, 19. https://doi.org/10.4230/lipics.itp.2022.10

[23] Manuel Eberl. 2020. Gaussian Integers. *Archive of Formal Proofs* (April 2020). https://isa-afp.org/entries/Gaussian_Integers.html, Formal proof development.

[24] David James Ford. 1978. *ON THE COMPUTATION OF THE MAXIMAL ORDER IN A DEDEKIND DOMAIN.* ProQuest LLC, Ann Arbor, MI. 145 pages. http://gateway.proquest.com/openurl?url_ver=Z39.88-2004&rft_val_fmt=info:ofi/fmt:kev:mtx:dissertation&res_dat=xri:pqdiss&rft_dat=xri:pqdiss:7902127 Thesis (Ph.D.)–The Ohio State University.

[25] Yuichi Futa, Daichi Mizushima, and Hiroyuki Okazaki. 2012. Formalization of Gaussian integers, Gaussian rational numbers, and their algebraic structures with Mizar. In *2012 International Symposium on Information Theory and its Applications*. 591–595.

[26] Benjamin Grégoire and Assia Mahboubi. 2005. Proving equalities in a commutative ring done right in Coq. In *Theorem proving in higher order logics*. Lecture Notes in Comput. Sci., Vol. 3603. Springer, Berlin, 98–113. https://doi.org/10.1007/11541868_7

[27] Markus Himmel. 2024. Formalization of Pratt Certificates. https://github.com/leanprover-community/mathlib4/blob/6439ce3f194a2acd309af6831d753e560c46bcf6/Mathlib/NumberTheory/LucasPrimality.lean. Accessed: 2024-09-06.

[28] Leslie Hogben (Ed.). 2014. *Handbook of linear algebra* (second ed.). CRC Press, Boca Raton, FL. xxx+1874 pages.

[29] Joseph-Louis Lagrange. 1798. Traité de la résolution des équations numériques, Paris (1798). *Œuvres de Lagrange.* 8 (1798), 637–649.

[30] Serge Lang. 2002. *Algebra* (third edition ed.). Graduate Texts in Mathematics, Vol. 211. Springer. xvi+914 pages.

[31] Serge Lang. 2005. *Undergraduate Algebra.* Springer.

[32] Robert Y. Lewis. 2017. An Extensible Ad Hoc Interface between Lean and Mathematica. *Electronic Proceedings in Theoretical Computer Science* 262 (Dec. 2017), 23–37. https://doi.org/10.4204/eptcs.262.4

[33] Robert Y. Lewis and Minchao Wu. 2022. A bi-directional extensible interface between Lean and Mathematica. *J. Automat. Reason.* 66, 2 (2022), 215–238. https://doi.org/10.1007/s10817-021-09611-1

[34] The LMFDB Collaboration. 2024. The L-functions and modular forms database. https://www.lmfdb.org. [Online; accessed 17 September 2024].

[35] Assia Mahboubi and Thomas Sibut-Pinote. 2021. A Formal Proof of the Irrationality of $\zeta(3)$. *Logical Methods in Computer Science* 17, 1 (2021). DOI: 10.23638/LMCS-17(1:16)2021.

[36] Assia Mahboubi and Enrico Tassi. 2020. *Mathematical Components.* Zenodo. https://doi.org/10.5281/zenodo.4282710

[37] J. Neukirch. 1999. *Algebraic number theory.* Fundamental Principles of Mathematical Sciences, Vol. 322. Springer. xviii+571 pages. https://doi.org/10.1007/978-3-662-03983-0 Translated from the 1992 German original and with a note by Norbert Schappacher, With a foreword by G. Harder.

[38] M. Pohst and H. Zassenhaus. 1997. *Algorithmic algebraic number theory.* Encyclopedia of Mathematics and its Applications, Vol. 30. Cambridge University Press, Cambridge. xiv+499 pages. Revised reprint of the 1989 original.

[39] Michael O. Rabin. 1980. Probabilistic algorithms in finite fields. *SIAM J. Comput.* 9, 2 (1980), 273–280. https://doi.org/10.1137/0209024

[40] P. Stevenhagen and H. W. Lenstra, Jr. 1996. Chebotarëv and his density theorem. *Math. Intelligencer* 18, 2 (1996), 26–37. https://doi.org/10.1007/BF03027290

[41] The PARI Group 2023. *PARI/GP version* 2.15.4. The PARI Group, Univ. Bordeaux. available from http://pari.math.u-bordeaux.fr/.

[42] The Sage Developers. 2024. *SageMath, the Sage Mathematics Software System (Version 10.3)*. https://www.sagemath.org. DOI: 10.5281/zenodo.10841614.

[43] René Thiemann, Ralph Bottesch, Jose Divasón, Max W. Haslbeck, Sebastiaan J. C. Joosten, and Akihisa Yamada. 2020. Formalizing the LLL basis reduction algorithm and the LLL factorization algorithm in Isabelle/HOL. *J. Autom. Reasoning* 64, 5 (2020), 827–856. https://doi.org/10.1007/s10817-020-09552-1

[44] Joachim von zur Gathen and Jürgen Gerhard. 2013. *Modern computer algebra* (third ed.). Cambridge University Press, Cambridge. xiv+795 pages. https://doi.org/10.1017/CBO9781139856065

[45] Eric Wieser. 2021. Scalar actions in Lean's mathlib. *CoRR* abs/2108.10700 (2021). arXiv:2108.10700