A formalization of Dedekind domains and class groups of global fields

Anne Baanen[1*], Sander R. Dahmen[2], Ashvni Narayanan[3] and Filippo A. E. Nuccio Mortarino Majno di Capriglio[4]

[1]Department of Computer Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands.
[2]Department of Mathematics, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands.
[3]London School of Geometry and Number Theory, London, United Kingdom.
[4]Univ Lyon, Université Jean Monnet Saint-Étienne, CNRS UMR 5208, Institut Camille Jordan, F-42023 Saint-Étienne, France.

*Corresponding author(s). E-mail(s): t.baanen@vu.nl;
Contributing authors: s.r.dahmen@vu.nl;
a.narayanan20@imperial.ac.uk; filippo.nuccio@univ-st-etienne.fr;

Abstract

Dedekind domains and their class groups are notions in commutative algebra that are essential in algebraic number theory. We formalized these structures and several fundamental properties, including number theoretic finiteness results for class groups, in the Lean prover as part of the mathlib mathematical library. This paper describes the formalization process, noting the idioms we found useful in our development and mathlib's decentralized collaboration processes involved in this project.

1 Introduction

In its basic form, number theory studies properties of the integers $\mathbb{Z}$ and its fraction field, the rational numbers $\mathbb{Q}$. For the sake of generalization, as well as for providing powerful techniques to answer questions about the original

objects $\mathbb{Z}$ and $\mathbb{Q}$, it is worthwhile to study finite extensions of $\mathbb{Q}$, called *number fields*, as well as their *rings of integers* (Section 2), whose relations mirror the way $\mathbb{Q}$ contains $\mathbb{Z}$ as a subring. In this paper, we describe our project aiming at formalizing these notions and some of their important properties. Our goal is not to get to the definitions and properties as quickly as possible; rather, we lay the foundations for future work, as part of a natural and more general theory.

In particular, our project resulted in formalized definitions and elementary properties of number fields and their rings of integers (Section 3.4), Dedekind domains (Section 4), and the ideal class group and class number (Section 7). Apart form the very basics concerning number fields, these concepts were not formalized before as far as we know. We note that our formal definition of the class number is an essential requirement for the use of theorem provers in modern number theory research. The main proofs that we formalized show that two definitions of Dedekind domains are equivalent (Section 4.3), that the ring of integers of a number field is a Dedekind domain (Section 6) and that the class group of the ring of integers of a number field is finite (Section 7). In fact, most of our results for number fields are also obtained in the more general setting of *global fields*.

Our work is developed as part of the mathematical library mathlib [1] for the Lean 3 theorem prover [2]. The formal system of Lean is a dependent type theory based on the calculus of inductive constructions, with a proof-irrelevant impredicative universe `Prop` at the bottom of a noncumulative hierarchy of universes `Prop : Type : Type 1 : Type 2 : ...` ; "an arbitrary `Type u`" is abbreviated as `Type*`. In other words, each element has a unique type. The natural number 0 has type $\mathbb{N}$, and the rational 0 has type $\mathbb{Q}$. One can then identify `0 : ℕ` with `0 : ℚ` using a map `ℕ → ℚ` called a coercion (generally denoted by an arrow); that is, `(0 : ℚ) = ↑(0 : ℕ)`. Other important characteristics of Lean as used in mathlib are the use of quotient types, ubiquitous classical reasoning and the use of typeclasses to define the hierarchy of algebraic structures.

Lean uses typeclass inference to "remember" and automatically assign properties to certain objects. If we define a structure with the predicate `class`, then one can prove special lemmas tagged with `instance` that will automatically be inferred by Lean. As an example, consider a ring R with a subring s. The instance `subring.to_ring` says that s is also a ring. Consequently, one can now use lemmas about rings for s without having to invoke `subring.to_ring`. We put the implicit arguments to be remembered by the typeclass system in square brackets. Other implicit arguments remain in curly brackets, while explicit arguments go in round brackets.

Other useful features of Lean include a variety of tactics such as `simp` (simplifies the main goal target using lemmas tagged with the attribute `[simp]`), `library_search` (tries to close the current goal by applying a lemma from the library), `ring` (tactic for solving equations in the language of commutative (semi)rings) etc. Lean uses these to simplify the statement or to close the goal.

These are very efficient when working with proofs that are calculation heavy, or that follow from a small number of easy (or mathematically trivial) steps.

Organizationally, mathlib is characterized by a distributed and decentralized community of contributors, a willingness to refactor its basic definitions, and a preference for small yet complete contributions over larger projects added all at once. In this project, as part of the development of mathlib, we followed this philosophy by contributing pieces of our work as they were finished. We, in turn, used results contributed by others after the start of the project. At several points, we had just merged a formalization into mathlib that another contributor needed, immediately before they contributed a result that we needed. Due to the decentralized organization and fluid nature of contributions to mathlib, its contents are built up of many different contributions from over 100 different authors. Attributing each formalization to a single set of main authors would not do justice to all others whose additions and tweaks are essential to its current use. Therefore, we will make clear whether a contribution is part of our project or not, but we will not stress whom we consider to be the main authors.

The source files of the formalization have been contributed to mathlib. We have preserved the development branch that this paper is based on[1]. We also maintain a repository[2] containing the source code referred to in this paper.

This paper is an extended version of a paper published in the ITP 2021 conference proceedings [3]. The additions to this paper, apart from several small local changes throughout the text, mainly concern the following.

- Code samples throughout have been updated to reflect parts of our formalization contributed to mathlib after the previous publication and to incorporate changes in mathlib after contribution.
- Instead of only considering class groups of Dedekind domains, we briefly describe class groups for general domains; see the end of Section 2 and Section 7.1.
- The new Section 3.1 gives a more detailed explanation of the use of typeclasses in mathlib.
- The further evolution of fraction rings in mathlib is discussed at the end of Section 3.6.
- We elaborate on unique factorization of ideals in Dedekind domains in Section 4.4.
- We give more details on the finiteness of the class number in Section 7.2.
- We elaborate on future directions in Section 8.2.

2 Mathematical background

Let us now introduce some of the main objects we study, described informally. We assume some familiarity with basic ring and field theory.

[1] https://github.com/leanprover-community/mathlib/tree/dedekind-domain-dev
[2] https://github.com/lean-forward/class-number-journal

A *number field* K is a finite extension of the field $\mathbb{Q}$, and as such has the structure of a finite dimensional vector space over $\mathbb{Q}$; its dimension is called the *degree* of K. The easiest example is $\mathbb{Q}$ itself, and the two-dimensional cases are given by the quadratic number fields $\mathbb{Q}(\sqrt{d}) = \{a + b\sqrt{d} : a, b \in \mathbb{Q}\}$ where $d \in \mathbb{Z}$ is not a square. Similarly, adding a cubic root $\sqrt[3]{d}$ of some $d \in \mathbb{Z}$ which is not a cube leads to the number field $\mathbb{Q}(\sqrt[3]{d})$: it has degree 3 but not all cubic number fields arise in this way. An example of a cubic number field that is not of this form, and that will occupy us later for other interesting features, is $\mathbb{Q}(\alpha_0) = \{a + b\alpha_0 + c\alpha_0^2 : a, b, c \in \mathbb{Q}\}$, where α_0 is the unique real number satisfying $\alpha_0^3 + \alpha_0^2 - 2\alpha_0 + 8 = 0$. In general, taking any root α of an irreducible polynomial of degree n over $\mathbb{Q}$ yields a number field of degree n, namely $\mathbb{Q}(\alpha) = \{c_0 + c_1\alpha + \ldots + c_{n-1}\alpha^{n-1} : c_0, c_1, \ldots, c_{n-1} \in \mathbb{Q}\}$, and, up to isomorphism, all number fields of degree n arise in this way.

The *ring of integers* $\mathcal{O}_K$ of a number field K is defined as the integral closure of $\mathbb{Z}$ in K, namely

$$\mathcal{O}_K := \{x \in K : f(x) = 0 \text{ where } f \in \mathbb{Z}[x] \text{ is a monic polynomial}\},$$

where we recall that a polynomial is called *monic* if its leading coefficient equals 1. While it might not be immediately obvious that $\mathcal{O}_K$ is a ring, this follows from general algebraic properties of integral closures. Some examples of rings of integers are the following. Taking $K = \mathbb{Q}$, we get $\mathcal{O}_K = \mathbb{Z}$ back. For $K = \mathbb{Q}(i) = \mathbb{Q}(\sqrt{-1})$ we get that $\mathcal{O}_K$ is the ring of Gaussian integers $\mathbb{Z}[i] = \{a + bi : a, b \in \mathbb{Z}\}$. But for $K = \mathbb{Q}(\sqrt{5})$ we do *not* simply get $\mathbb{Z}[\sqrt{5}] = \{a + b\sqrt{5} : a, b \in \mathbb{Z}\}$ as $\mathcal{O}_K$, since the golden ratio $\varphi := (1 + \sqrt{5})/2 \notin \mathbb{Z}[\sqrt{5}]$ satisfies the monic polynomial equation $\varphi^2 - \varphi - 1 = 0$; hence by definition, $\varphi \in \mathcal{O}_K$. It turns out that $\mathcal{O}_K = \mathbb{Z}[\varphi] = \{a + b\varphi : a, b \in \mathbb{Z}\}$. Finally, if $K = \mathbb{Q}(\alpha_0)$ with α_0 as before, then $\mathcal{O}_K = \{a + b\alpha_0 + c(\alpha_0 + \alpha_0^2)/2 : a, b, c \in \mathbb{Z}\}$, illustrating that explicitly writing down $\mathcal{O}_K$ can quickly become complicated. Further well-known rings of integers are the Eisenstein integers $\mathbb{Z}[(1+\sqrt{-3})/2]$ and the ring $\mathbb{Z}[\sqrt{2}]$.

Thinking of $\mathcal{O}_K$ as a generalization of $\mathbb{Z}$, it is natural to ask which of its properties still hold in $\mathcal{O}_K$ and, when this fails, if a reasonable weakening does.

An important property of $\mathbb{Z}$ is that it is a principal ideal domain (PID), meaning that every ideal is generated by one element. This implies that every nonzero nonunit element can be written as a finite product of prime elements, which is unique up to reordering and multiplying by ± 1: a ring where this holds is called a unique factorization domain, or UFD. For example, 6 can be factored in primes in 4 equivalent ways, namely $6 = 2 \cdot 3 = 3 \cdot 2 = (-2) \cdot (-3) = (-3) \cdot (-2)$. In fact, the previously mentioned examples of rings of integers are UFDs, but this is certainly not true for all rings of integers. For example, unique factorization *does not* hold in $\mathcal{O}_{\mathbb{Q}(\sqrt{-5})} = \mathbb{Z}[\sqrt{-5}]$: it is easy to prove

that $6 = 2 \cdot 3$ and $6 = (1 + \sqrt{-5})(1 - \sqrt{-5})$ provide two essentially different[3] ways to factor 6 into prime elements of $\mathbb{Z}[\sqrt{-5}]$.

As it turns out, there is a way to weaken this notion of unique factorization in a meaningful way. Namely, by considering factorization of *ideals* instead of elements: given a number field K, with ring of integers $\mathcal{O}_K$, a beautiful and classical result by Dedekind shows that every nonzero ideal of $\mathcal{O}_K$ can be factored as a product of prime ideals in a unique way, up to reordering.

Although unique factorization in terms of ideals is of great importance, it is still interesting, and sometimes necessary, to also consider factorization properties in terms of elements. We have already mentioned that unique factorization in $\mathbb{Z}$ follows from the fact that every ideal is generated by a single element. Now, it is convenient to extend the notion of ideals of $\mathbb{Z}$ to that of *fractional ideals*. These are additive subgroups of $\mathbb{Q}$ of the form $\frac{1}{d}I$ with I an ideal of $\mathbb{Z}$ and d a nonzero integer. When the distinction is important, we refer to an ideal $I \subseteq \mathbb{Z}$ as an *integral ideal*. The nonzero fractional ideals of $\mathbb{Z}$ naturally form a multiplicative group (whereas, for instance, there is no integral ideal $I \subseteq \mathbb{Z}$ such that $I * (2\mathbb{Z}) = (1)$). The statement that every ideal is generated by a single element translates to the fact that the quotient group of nonzero fractional ideals modulo $\mathbb{Q}^\times$ is trivial (where $\frac{a}{b} \in \mathbb{Q}^\times$ corresponds to $\frac{1}{b}a\mathbb{Z}$, and the multiplicative group of invertible elements of a ring R is denoted by $R^\times$).

It turns out that this quotient group can be defined for every ring of integers $\mathcal{O}_K$. The fundamental theoretical notion beneath this construction is that of Dedekind domain: these are integral domains D which are Noetherian (every ideal of D is finitely generated), integrally closed (if an element x in the fraction field $\operatorname{Frac} D$ of D is a root of a monic polynomial with coefficients in D, then actually $x \in D$), and of Krull dimension at most 1 (every nonzero prime ideal of D is maximal). It can be proved that the nonzero fractional ideals of a Dedekind domain D form a group under multiplication, and that the quotient of this group by the image of the natural embedding of $(\operatorname{Frac} D)^\times$ is called the (*ideal*) *class group* $\mathcal{C}l_D$. For later reference, fractional ideals generated by one element of $\operatorname{Frac} D$ are called principal fractional ideals, so the image of the natural embedding of $(\operatorname{Frac} D)^\times$ consists exactly of the nonzero principal fractional ideals.

What is arithmetically crucial is the theorem ensuring that the ring of integers $\mathcal{O}_K$ of every number field K is a Dedekind domain, and that in this case the class group $\mathcal{C}l_{\mathcal{O}_K}$ is actually *finite*. In particular, $\mathcal{C}l_{\mathcal{O}_K}$ can be seen as "measuring" how far ideals of $\mathcal{O}_K$ are from being generated by a single element and, consequently, as a measure of the failure of unique factorization. The order of $\mathcal{C}l_{\mathcal{O}_K}$ is called the *class number* of K. Intuitively, then, the smaller the class number, the fewer factorizations are possible. In particular, the class number of K is equal to 1 if and only if $\mathcal{O}_K$ is a UFD.

[3] By "essentially different" we mean that one factorization cannot be obtained from the other via multiplication by units.

The statements in the previous paragraph also hold for *function fields*, namely fields which are finite extensions of $\mathbb{F}_q(t) \simeq \operatorname{Frac} \mathbb{F}_q[t]$, where $\mathbb{F}_q[t]$ stands for the ring of univariate polynomials (in a free variable t) with coefficients in a finite field $\mathbb{F}_q$ with q elements. Recall that when q is a prime number, $\mathbb{F}_q$ is simply the field $\mathbb{Z}/q\mathbb{Z}$. A field which is either a number field or a function field is called a *global field*.

The concept of class group actually not only makes sense for Dedekind domains but more generally for (at least) any integral domain R as follows. While the nonzero fractional ideals of R in general need not be a group, they do form a commutative monoid. Hence, the invertible fractional ideals of R form a group, and the class group of R (denoted $\mathcal{C}l_R$) is now defined as the quotient of this group by the image of the natural embedding of $(\operatorname{Frac} R)^\times$.

In the next sections we will describe the formalization of the above concepts.

3 Number fields, global fields and rings of integers

We refer the reader to Section 2 for the mathematical background needed in this section.

We formalized number fields as the following typeclass:

```
class is_number_field (K : Type*) [field K] : Prop :=
[to_char_zero : char_zero K]
[to_finite_dimensional : finite_dimensional ℚ K]
```

The *class* keyword declares a structure type (in other words, a type of records) and enables typeclass inference for terms of this type; we describe the use of typeclasses in mathlib in Section 3.1. Round brackets mark parameters that must explicitly be supplied by the user, such as `(K : Type*)`, while square brackets mark instance parameters inferred by the typeclass system, such as `[field K]`. The condition `[to_char_zero : char_zero K]` states that K has characteristic zero, so the unique ring homomorphism $\mathbb{Z} \to K$ is an embedding. This implies that there is a $\mathbb{Q}$-algebra structure on K (found by typeclass instance synthesis), endowing K with the $\mathbb{Q}$-vector space structure used in the hypothesis `[to_finite_dimensional : finite_dimensional ℚ K]`.

Similarly, we defined the class of function fields over a finite field $\mathbb{F}_q$ as:

```
class function_field (Fq F : Type*) [field Fq] [field F] :
  Prop :=
[to_algebra : algebra (ratfunc Fq) F]
[to_finite_dimensional : finite_dimensional (ratfunc Fq) F]
```

The hypothesis `[to_algebra : algebra (ratfunc` $\mathbb{F}_q$`) F]` witnesses that F is a field extension of the field $\mathbb{F}_q(t)$ of rational functions over $\mathbb{F}_q$, where $\mathbb{F}_q$ is a finite field. Again, the condition that this extension is finite is written using the `finite_dimensional` typeclass. We present a more detailed analysis of

`algebra` in Section 3.2 and of fraction fields including `ratfunc` in Section 3.6. For now, we point out that there are many fields K that are isomorphic to the field of rational functions $\mathbb{F}_q(t)$; we provided a theorem `function_field_iff` that shows that the choice of K does not matter. Note that there is no requirement that the field `Fq` is finite, since this is not needed to state the conditions on `F`. We instead add a `[fintype Fq]` hypothesis only to those results that require finiteness.

3.1 Use of typeclasses

Typeclasses were originally introduced in Haskell as a mechanism for operator overloading [4], and are used throughout Lean's core library and mathlib to endow types with mathematical structures consisting of both operators and their properties [1]. When the elaborator sees a function with an instance parameter being applied, such as the `[field K]` parameter of `is_number_field K`, a Prolog-like search is started to automatically synthesize a suitable value for this parameter. Each of the local parameters and the declarations marked as `instance` is tested in turn to see if their type matches the expected type of the instance synthesis. All instance parameters of candidate instances are themselves recursively inferred, until either a suitable term is constructed or no more candidates remain; an error is raised in the latter case [5, Section 10]. Compared to Haskell's, Lean's typeclasses have few structural restrictions: notably, classes and instances can depend on any term, instances may overlap, classes can apply to multiple types and can have functional dependencies.

In our development, we followed the common practice in mathlib of providing structure on a type, whenever such a structure exists canonically, through typeclasses. The informal notion of providing a certain mathematical structure on a type should not be confused with the `structure` keyword formally declaring a *structure type* whose elements are tuples. To confuse matters further, Lean implements typeclasses as structure types, where the instances are tuples of the typeclass's fields. Typeclasses provided us a way to treat uniformly situations that are informally considered the same, as we discuss in Sections 3.2 and 3.3. Our reliance on typeclasses did not cause any noticeable slowness in proof checking: there was no instance that should be found but could not due to timeouts.

A central consideration in formalizing definitions for mathlib is choosing the appropriate amount of *bundling*: determining whether information about an object should be carried by the object itself (bundled), or passed as a separate value (unbundled) [6]. For example, the `is_number_field` typeclass is considered to have unbundled inheritance from the `field` class because instances of these classes are passed in separate parameters, while it has bundled inheritance from `char_zero` and `finite_dimensional` since both are included as fields of the structure. Similarly, the formalization of admissible absolute values discussed in Section 7.2 features a bundled structure `absolute_value` which includes a map along with proofs stating that this map is an absolute value,

and an unbundled structure `is_admissible` which takes the absolute value map as a separate parameter.

Unbundling has an advantage in expressivity: because each property of an object is passed in a separate parameter, modifying one hypothesis requires modifying one parameter. In contrast, bundling hypotheses means that each subset of hypotheses requires its own structure declaration; any results proved for a given structure have to be made available for other structures manually or through automation such as typeclass inference. On the other hand, bundled structures result in simpler parameter lists, since fully unbundling the `field` class would result in each of its 38 structure fields becoming a separate parameter. Technical considerations play another important role in choosing the level of bundling: bundled properties are easily found by automation compared to unbundled properties which require a search of the local context, bundled inheritance between classes can only be applied when the two classes have the same type parameters, while long unbundled inheritance chains cause exponentially large terms, resulting in slowdowns and high memory consumption. Although there is no general rule governing bundling, in general mathlib prefers to bundle if possible, unbundling only when the additional properties are all `Prop`-valued and are not involved in long inheritance chains.

3.2 Field extensions

The definition of `is_number_field` illustrates our treatment of field extensions. A field L containing a subfield K is said to be a field extension L/K. Often we encounter towers of field extensions: we might have that $\mathbb{Q}$ is contained in K, K is contained in L, L is contained in an algebraic closure $\overline{K}$ of K, and $\overline{K}$ is contained in $\mathbb{C}$. We might formalize this situation by viewing $\mathbb{Q}$, K, L and $\overline{K}$ as sets of complex numbers $\mathbb{C}$ and defining field extensions as subset relations between these subfields. This way, no coercions need to be inserted in order to map elements of one field into a larger field. Unfortunately, we can only avoid coercions as far as we are able to stay within one largest field. For example, the definition of complex numbers depends on many results for rational numbers, which would need to be proved again, or transported, for the subfield of $\mathbb{C}$ isomorphic to $\mathbb{Q}$.

Instead, we formalized results about field extensions through parametrization. The fields K and L can be arbitrary types and the hypothesis "L is a field extension of K" is represented by an instance parameter `[algebra K L]` denoting a K-algebra structure on L. The `algebra` structure provides us with a ring homomorphism `algebra_map K L` : $K \to L$; this map is injective because K and L are fields. In other words, field extensions are given by their embeddings.

There are multiple possible K-algebra structures for a field L and Lean does not enforce uniqueness of typeclass instances, but the mathlib maintainers try to ensure all instances that can be inferred are *definitionally equal*. Definitional equality is a syntactical notion of equality found in dependent type theories that reflects the possibility of computation: for example, the term `2 + 2 : ℕ`

is definitionally equal to `4`. Whenever Lean can infer the definitional equality of two terms (the terms are said to *unify*), one can be substituted for the other. Thus, ensuring definitional equality for instances means that overlapping instances will not lead to conflicts when one instance is expected and another is found.

3.3 Scalar towers

The main drawback of using arbitrary embeddings to represent field extensions is that we need to prove that these maps commute. For example, we might start with a field extension $L/\mathbb{Q}$, then define a subfield K of L, resulting in a tower of extensions $L/K/\mathbb{Q}$. In such a tower, the map $\mathbb{Q} \to L$ should be equal to the composition $\mathbb{Q} \to K$ followed by $K \to L$. Such an equality cannot always be achieved by defining the map $\mathbb{Q} \to L$ to be this composition: in the example, the map $\mathbb{Q} \to K$ depends on the map $\mathbb{Q} \to L$.

The solution in mathlib is to parametrize over all three maps, as long as there is also a proof of coherence: a hypothesis of the form "$L/K/F$ is a tower of field extensions" is translated into three instance parameters `[algebra F K]`, `[algebra K L]` and `[algebra F L]`, along with a parameter `[is_scalar_tower F K L]` expressing that the maps commute.

The `is_scalar_tower` typeclass derives its name from its applicability to any three types among which scalar multiplication operations exist:

```
class is_scalar_tower (M N α : Type*)
  [has_scalar M N] [has_scalar N α] [has_scalar M α] : Prop :=
(smul_assoc : ∀(x : M) (y : N) (z : α), (x • y) • z = x • (y • z))
```

For example, if R is a ring, A is an R-algebra and M an A-module, we can state that M is also an R-module by adding a `[is_scalar_tower R A M]` parameter. Since `x • y` for an R-algebra A is defined as `algebra_map R A x * y`, applying `smul_assoc` for each $x : K$ with $y = (1 : L)$ and $z = (1 : F)$ shows that the `algebra_map`s indeed commute in a tower of field extensions $L/K/F$.

Common `is_scalar_tower` instances are declared in mathlib, such as for the maps $R \to S \to B$ when S is a R-subalgebra of A and B is an A-algebra such that `is_scalar_tower R A B`; this also implies that the maps $R \to S \to A$ form a tower. The effect is that almost all coherence proof obligations are automated through typeclass instance synthesis. Only when defining a new algebra structure were we required to supply the `is_scalar_tower` instances ourselves.

3.4 Rings of integers

When K is a number field (defined as a field satifying `is_number_field`), the ring $\mathcal{O}_K$ of integers in K is defined as the integral closure of $\mathbb{Z}$ in K. This is the subring containing those $x : K$ that are a root of a monic polynomial with coefficients in $\mathbb{Z}$:

```
def number_field.ring_of_integers (K : Type*) [field K]
```

```
  [is_number_field K] : subalgebra ℤ K :=
integral_closure ℤ K
```

where `integral_closure` was already defined in mathlib. When K is a function field over the finite field $\mathbb{F}_q$, we defined $\mathcal{O}_K$ analogously as `integral_closure (`$\mathbb{F}_q$`[X]) K`.

Since the integers $\mathbb{Z}$ are integrally closed in $\mathbb{Q}$, this construction of the ring of integers of the number field $\mathbb{Q}$ is isomorphic, but not definitionally equal, to $\mathbb{Z}$. To avoid dealing with these isomorphisms, and also to treat the two definitions of rings of integers on an equal footing, we introduced a typeclass `is_integral_closure A R B` stating that A is the integral closure of R in B, and worked with a generic `is_integral_closure` instance instead of the specific `ring_of_integers` construction when possible.

3.5 Subobjects

The ring of integers is one example of a subobject, such as a subfield, subring or subalgebra, defined through a characteristic predicate. In mathlib, subobjects are "bundled", in the form of a `structure` comprising the carrier set and proofs showing the carrier set is closed under the relevant operations. Bundled subobjects provide similar benefits as bundled morphisms; the choice for the latter is explained in the mathlib overview paper [1]. Where the `algebra` and `is_scalar_tower` typeclasses provide an interface generalizing over multiple equivalent definitions, subobjects provide a specific implementation of that interface in the form of a subtype.

Two new subobjects that we defined in our development were `subfield` as well as `intermediate_field`. We defined a subfield of a field K as a subset of K that contains 0 and 1 and is closed under addition, negation, multiplication and taking inverses. If L is a field extension of K, we defined an intermediate field as a subfield of L that is also a K-subalgebra: in other words, a subfield that contains the image of `algebra_map K L`. Other examples of subobjects available in mathlib are submonoids, subgroups and submodules (with ideals as a special case of submodules); all of these are provided with an instance of the `set_like` typeclass that supplies notation such as a membership relation "$x \in S$".

The new definitions found immediate use: soon after we contributed our definition of `intermediate_field` to mathlib, the Berkeley Galois theory group used it in a formalization of the primitive element theorem. Soon after the primitive element theorem was merged into mathlib, we used it in our development of the trace form. This anecdote illustrates the decentralized development style of mathlib, with different groups and people building on each other's results in a collaborative process.

Through the `set_like` typeclass, subobjects can be coerced to types, by sending a subobject S to the subtype of all elements of S. By putting typeclass instances on this subtype, we could reason about inductively defined rings such as $\mathbb{Z}$ and subrings such as `integral_closure ℤ K` uniformly. If S : `subfield` K, there is a ring embedding, the map that sends $x : S$ to K

by "forgetting" that $x \in S$, and we registered this map as an `algebra S K` instance, also allowing us to treat field extensions of the form $\mathbb{Q} \to \mathbb{C}$ and subfields uniformly. Similarly, for F : `intermediate_field K L`, we defined the corresponding `algebra K F`, `algebra F L` and `is_scalar_tower K F L` instances.

3.6 Fields of fractions

The fraction field $\operatorname{Frac} R$ of an integral domain R can be defined explicitly as a quotient type as follows: starting from the set of pairs (a, b) with $a, b \in R$ such that $b \neq 0$, one quotients by the equivalence relation generated by $(a, b) \sim (a\alpha, b\alpha)$ for all $\alpha \neq 0 : R$, writing the equivalence class of (a, b) as $\frac{a}{b}$. It can easily be proved that the ring structure on R extends uniquely to a field structure on $\operatorname{Frac} R$; in mathlib this construction is called `fraction_ring R`, and is used to define the field of rational functions $K(X) =$ `ratfunc K`. When $R = \mathbb{Z}$, this yields the traditional description of $\mathbb{Q}$ as the set of equivalence classes of fractions, where $\frac{2}{3} = \frac{-4}{-6}$, etc.

The drawback of this construction is that there are many other fields that can serve as the field of fractions for the same ring. Consider the field $\{z \in \mathbb{C} : \Re z \in \mathbb{Q}, \Im z \in \mathbb{Q}\}$, which is isomorphic to $\operatorname{Frac}(\mathbb{Z}[i])$ but not definitionally equal to it. Indeed, the mathlib definition of the rational numbers $\mathbb{Q}$ is a product type, not a quotient type, so we would not be able to treat $\mathbb{Q}$ as the field of fractions of $\mathbb{Z}$ in this setup. Any properties proven for $\mathbb{Q}$ would have to be repeated for $\operatorname{Frac}(\mathbb{Z})$, using transfer lemmas stating these properties are preserved by the isomorphism between $\mathbb{Q}$ and $\operatorname{Frac}(\mathbb{Z})$.

The strategy used in mathlib is to rather allow for many different *fraction fields* of our given integral domain R — as fields K with a suitable `[algebra R K]` instance, where the map `algebra_map R K` witnesses that all elements K are "fractions" of elements of R — and to parametrize every result over the choice of K. The conditions on the R-algebra structure on K are encoded as a typeclass `is_fraction_ring R K`. In the definition used by mathlib, a fraction ring is a special case of a *ring localization*, which is defined for any commutative ring R. Different localizations restrict the denominators to different multiplicative submonoids of $R \setminus \{0\}$.

The conditions on `algebra_map R K` imply that K is the smallest field, up to isomorphism, containing R, expressed by the following unique mapping property. If $g \colon R \to A$ is an injective map to a ring A such that $g(x)$ has a multiplicative inverse for all $x \neq 0 : R$, then it can be extended uniquely to a map $K \to A$ compatible with `algebra_map R K` and g. In particular, given `is_fraction_ring R` K_1 and `is_fraction_ring R` K_2, we can derive an isomorphism $K_1 \simeq K_2$. The construction of $\operatorname{Frac} R$ then results in *a* field of fractions (with an instance `is_fraction_ring R (fraction_ring R)`) rather than *the* field of fractions.

The above description of fraction fields is the third such formalization in mathlib. The first version consisted of a quotient type `quotient_ring R`, constructed similarly to the current definition of `fraction_ring R`. Due to the

aforementioned drawback — namely, that this provided no easy way to view $\mathbb{Q}$ as the field of fractions of $\mathbb{Z}$, for instance — this was refactored to use a characteristic predicate instead.

The second version defined K to be the field of fractions of R if there existed an injective *fraction map* $f : R \to K$, which is a ring homomorphism witnessing that all elements of K are "fractions" of elements of R; the map and its properties were bundled as a type `fraction_map R K`. Results on fraction fields were parametrized over a choice of fraction map f. This made it possible to view $\mathbb{Q}$ as the fraction field of $\mathbb{Z}$, by providing a suitable map called `int.fraction_map : fraction_map ℤ ℚ`. This came at a price: informally, at any given stage of one's reasoning, the field K is fixed and the map $f\colon R \to K$ is applied implicitly, just viewing every $x : R$ as $x : K$. It is now impossible to view $R \leq K$ as an inclusion of R-subalgebras, because the map f is needed explicitly to give the R-algebra structure on K. As a workaround, mathlib used a type synonym `codomain f := K` and instantiated the R-algebra structure given by f on this synonym. Again we encountered a distinction between $\mathbb{Q}$ "itself" and Frac($\mathbb{Z}$) = `codomain int.fraction_map`, still requiring the transfer of results such as typeclass instances.

The most recent version is the one described above. Inspired by our success in using the `algebra` typeclass to denote inclusions of rings, we unbundled the explicit `(f : fraction_map R K)` parameters into an instance parameter `[algebra R K]` that specifies the map, and an instance parameter `[is_fraction_ring R K]` that specifies the conditions satisfied by the map. Separating out these parameters finally allowed us to painlessly view $\mathbb{Q}$ as the fraction ring of $\mathbb{Z}$ while preserving the original $\mathbb{Z}$-algebra structure on $\mathbb{Q}$.

3.7 Representing monogenic field extensions

In Section 2 we have informally said that every number field K can be written as $K = \mathbb{Q}(\alpha)$ for a root α of an irreducible polynomial $P \in \mathbb{Q}[X]$. This can be made precise in several ways. For instance, one can consider a large field L (of characteristic 0) where P splits completely, then choose a root $\alpha \in L$ and let $K = \mathbb{Q}(\alpha)$ be the smallest subfield of L containing α. Or, one can consider the quotient ring $\mathbb{Q}[X]/P$ and observe that this is a field where the class $X \pmod P$ is a root of P. The assignment $\alpha \mapsto X \pmod P$ yields an isomorphism of the two fields, but any other choice of a root $\alpha' \in L$ leads to another isomorphism $\mathbb{Q}(\alpha') \cong \mathbb{Q}[X]/P$. Although mathematically we often tacitly identify these constructions, there is no canonical representation of the *monogenic* extensions of $\mathbb{Q}$, those which can be obtained by adjoining a single root of one polynomial.

The same continues to hold if we replace the base field $\mathbb{Q}$ with another field F, thus considering extensions of the form $F(\alpha)$, now requiring that α be a root of some $P \in F[X]$. Various constructions of $F(\alpha)$ have already been formalized in mathlib. The ability to switch between these representations is important: sometimes K and F are fixed and we want an arbitrary α; sometimes α is fixed and we want an arbitrary type representing $F(\alpha)$.

To find a uniform way to reason about all these definitions, we chose to formalize the notion of *power basis* to represent monogenic field extensions: this is a basis of the form $1, \alpha, \alpha^2, \ldots, \alpha^{n-1} : K$ (viewing K as a F-vector space). We defined a structure type bundling the information of a power basis. Omitting some generalizations not needed in this paper, the definition reads:

```
structure power_basis (F K : Type*) [field F] [field K]
  [algebra F K] :=
(gen : K) (dim : ℕ) (basis : basis (fin dim) F K)
(basis_eq_pow : ∀ i, basis i = gen ^ (i : ℕ))
```

We formalized that the previously defined notions of monogenic field extensions are equivalent to the existence of a power basis.

With the `power_basis` structure, we gained the ability to parametrize our results, being able to choose the F and K in a monogenic field extension K/F, or being able to choose the α generating $F(\alpha)$ (by setting the `gen` field to α). To specialize a result from an arbitrary K with a power basis over F to a specific construction of $K = F(\alpha)$, one can apply the result to the power basis `pb` generated by α and rewrite `power_basis.gen pb` $= \alpha$.

4 Dedekind domains

The right setting to study algebraic properties of number fields are *Dedekind domains*. We formalized fundamental results on Dedekind domains, including the equivalence of two definitions of Dedekind domains.

4.1 Definitions

There are various equivalent conditions, used at various times, for an integral domain D to be a Dedekind domain. The following three have been formalized in mathlib:

- `is_dedekind_domain D`: D is a Noetherian integral domain, integrally closed in its fraction field and has Krull dimension at most 1;
- `is_dedekind_domain_inv D`: D is an integral domain and nonzero fractional ideals of D have a multiplicative inverse (we discuss the notion and formalization of fractional ideals in Section 4.2);
- `is_dedekind_domain_dvr D`: D is a Noetherian integral domain and the localization of D at each nonzero prime ideal is a discrete valuation ring.

Note that fields are Dedekind domains according to these conventions.

The mathlib community chose `is_dedekind_domain` as the main definition, since this condition is usually the one checked in practice [7]. The other two equivalent definitions were added to mathlib, but before formalizing the proof that they are indeed equivalent. Having multiple definitions allowed us to do our work in parallel without depending on unformalized results. For example, the proof of unique ideal factorization in a Dedekind domain initially assumed `is_dedekind_domain_inv D`, and the proof that the

ring of integers $\mathcal{O}_K$ is a Dedekind domain concluded `is_dedekind_domain (ring_of_integers K)`. After the equivalence between `is_dedekind_domain D` and `is_dedekind_domain_inv D` was formalized, we could easily replace usages of `is_dedekind_domain_inv` with `is_dedekind_domain`.

The conditions `is_dedekind_domain` and `is_dedekind_domain_inv` require a fraction field K, although the truth value of the predicates does not depend on the choice of K. For ease of use, we let the type of `is_dedekind_domain` depend only on the domain D by instantiating K in the definition as `fraction_ring D`. From now on, we fix a fraction field K of D.

```
class is_dedekind_domain (D : Type*)
  [comm_ring D] [is_domain D] :=
(is_noetherian_ring : is_noetherian_ring D)
(dimension_le_one : dimension_le_one D)
(is_integrally_closed : is_integrally_closed D)
```

Applications of `is_dedekind_domain` can choose a specific fraction field through the following lemma exposing the alternate definition:

```
lemma is_dedekind_domain_iff [is_fraction_ring D K] :
  is_dedekind_domain D ↔
    is_noetherian_ring D ∧ dimension_le_one D ∧
    ∀ {x : K}, is_integral D x →
      ∃ (y : D), algebra_map D K y = x
```

We marked `is_dedekind_domain` as a typeclass by using the keyword `class` rather than `structure`, allowing the typeclass system to automatically infer the Dedekind domain structure when an appropriate instance is declared, such as for PIDs or for rings of integers.

4.2 Fractional ideals

The notion which is pivotal to the definition of the ideal class group of a Dedekind domain is that of *fractional ideals*: given any integral domain R with a field of fractions F, we define `is_fractional` as a predicate on R-submodules J of F, informally as "there is an $x : R$ with $xJ \subseteq R$". For a Dedekind domain, nonzero fractional ideals form a group under multiplication. As seen in Section 3.6, this notion depends on the field F as well as on the embedding f `:= algebra_map R F`. A more precise way of stating the above condition is then $f(x)J \subseteq f(R)$. We formalized the definition of fractional ideals of R contained in F as a type `fractional_ideal R F`, whose elements consist of an R-submodule of F along with a proof of `is_fractional`. The structure of fractional ideals does not depend on the choice of a fraction field, which we formalized as an isomorphism `fractional_ideal.canonical_equiv` between two types of fractional ideals on R, corresponding to different fields of fractions.

We defined the addition, multiplication and intersection operations on fractional ideals, by showing that the corresponding operations on submodules

map fractional ideals to fractional ideals. We also formalized that these operations give a commutative semiring structure on the type of fractional ideals. For example, multiplication of fractional ideals is defined as

```
lemma is_fractional.mul (I J : submodule R F)
  is_fractional R I → is_fractional R J →
  is_fractional R (I * J) := _ -- proof omitted

instance : has_mul (fractional_ideal R F) :=
⟨λ I J, ⟨I * J : submodule R F,
 is_fractional.mul I.is_fractional J.is_fractional⟩⟩
```

Defining the quotient of two fractional ideals requires slightly more work. Consider any R-algebra A and an injection $R \hookrightarrow A$. Given ideals $I, J \leq R$, the submodule $I/J \leq A$ is defined by the property

```
lemma submodule.mem_div_iff_forall_mul_mem {x : A}
  {I J : submodule R A} :
  x ∈ I / J ↔ ∀ y ∈ J, x * y ∈ I
```

Beware that the notation $1/I$ might be misleading here: indeed, for general integral domains, the equality $I * 1/I = 1$ might not hold. As an example, one can consider the ideal (X, Y) in $\mathbb{C}[X, Y]$. On the other hand, we formalized that this equality holds for Dedekind domains (Section 4.3) as the following lemma:

```
theorem fractional_ideal.mul_inv_cancel [is_dedekind_domain D]
  {I : fractional_ideal D F} (hne : I ≠ 0) : I * (1 / I) = 1
```

This justifies the notation $I^{-1} = 1/I$. In fact, we define this notation even for the ideal 0, by declaring that $0^{-1} = 0$. This reflects the existence of the typeclass `group_with_zero` in `mathlib`, consisting of groups endowed with an extra element `0` whose inverse is again `0`.

Moreover, `mathlib` used to define $a/b := a * b^{-1}$, but our definition of $I^{-1} = 1/I$ would cause circularity. This led us to a major refactor of this core definition. In particular, we had to weaken the definitional equality to a proposition; this involved many small changes throughout `mathlib`[4].

4.3 Equivalence of the definitions

We now describe how we proved and formalized that the two definitions `is_dedekind_domain` and `is_dedekind_domain_inv` of being a Dedekind domain are equivalent. Let D be a Dedekind domain, and let $f\colon D \to K$ a fraction map to a field of fractions K of D.

To show that `is_dedekind_domain_inv` implies `is_dedekind_domain`, we follow the proof given by Fröhlich in [8, Chapter 1, Section 2, Proposition 1.2.1]. A constant challenge that was faced while coding this proof was already mentioned in Section 3.6, namely the fact that elements of the domain must be

[4]The pull requests are available as https://github.com/leanprover-community/mathlib/pull/5302 and https://github.com/leanprover-community/mathlib/pull/5303.

traced along the inclusion into the chosen field of fractions. The proofs for being integrally closed and of dimension being less than or equal to 1 are fairly straightforward.

Formalizing the Noetherian condition was the most challenging. Fröhlich considers elements $a_1, \ldots, a_n \in I$ and $b_1, \ldots, b_n \in I^{-1}$ for any nonempty fractional ideal I, satisfying $\sum_i a_i b_i = 1$. Observe now that, in mathlib, the definition of the product $A * B$ of two fractional ideals A, B is a special case of the product of two submodules, and therefore it is defined as

```
submodule.has_mul = {mul := λ (A B : submodule D K),
 ⨆ (a : A), submodule.map ((algebra.lmul D K) a.val) B}
```

Unravelling this definition, we see that it defines $A*B$ as the smallest (*i. e.*, the infimum with respect to set-theoretic inclusion as order relation) submodule containing all submodules $a \cdot B$ for $a \in A$, where $a \cdot B$ is the range of the function `λ b : B, a*b`. However, it is quite challenging to formalize that an element of $A * B$ must be a *finite* sum $\sum_i a_i * b_i$, for $a_i \in A$ and $b_i \in B$. Instead, we show that, for every element $x \in A * B$, there are finite sets $T \subseteq A$, $T' \subseteq B$ such that `x ∈ span (T * T')`, formalized as `submodule.mem_span_mul_finite_of_mem_mul`. Now considering a nonzero integral ideal I of the ring D, by definition of invertibility we can write `1 ∈ (1 : fractional_ideal D K) = I * I⁻¹`. Hence, we obtain finite sets $T \subset I$ and $T' \subset I^{-1}$ such that 1 is contained in the D-span of $T * T'$. We used the `norm_cast` tactic [9] to resolve most coercions, however, this tactic did not solve coercions coming from the inclusion `algebra_map D K`. With coercions, the actual statement of the latter expression in Lean is `↑T' ⊆ ↑↑(↑I)⁻¹`, which reads

```
(T' : set K) ⊆ (((I : fractional_ideal D K)⁻¹ : submodule D K)
    : set K)
```

From the existence of T and T' we concluded that I is indeed finitely generated, thus finishing the proof.

The theorem `fractional_ideal.mul_inv_cancel` proves the converse, namely that `is_dedekind_domain` implies `is_dedekind_domain_inv`. The classical proof consists of three steps: first, every maximal ideal $M \subseteq D$, seen as a fractional ideal, is invertible; secondly, every nonzero ideal is invertible, using that it is contained in a maximal ideal; thirdly, the fact that every fractional ideal J satisfies $xJ \leq I$ for a suitable element $x \in D$ and a suitable ideal $I \subseteq D$ implies that every fractional ideal is invertible, concluding the proof that nonzero fractional ideals form a group. The third step was easy, building upon the material developed for the general theory of `fractional_ideal`. Concerning the first two, we found that passing from the case where M is maximal to the general case required more code than directly showing invertibility of arbitrary nonzero ideals. The formal statement reads

```
lemma coe_ideal_mul_inv [is_dedekind_domain D]
  (I : ideal D) (hI0 : I ≠ 0) :
  (↑I * (↑I)⁻¹ : fractional_ideal D K) = 1
```

from where it becomes apparent that we had to repeatedly distinguish between `I : ideal D`, and its coercion `↑I : fractional_ideal D K` although these objects, from a mathematical point of view, are identical.

The formal proof of this result relies on the lemma `exists_not_mem_one_of_ne_bot`, which says that for every non-trivial ideal $0 \subsetneq I \subsetneq D$, there exists an element in the field K which is not integral (so, not in `1 : fractional_ideal D K`) but lies in I^{-1}. The proof begins by invoking that every nonzero ideal in the Noetherian ring D contains a product of nonzero prime ideals. This result was not previously available in mathlib. The dimension condition shows its full force when applying this lemma: each prime ideal in the product $I * I^{-1}$, being nonzero, will be maximal because the Krull dimension of D is at most 1; from this, `exists_not_mem_one_of_ne_bot` follows easily. Having the above lemma at our disposal, we were able to prove that every ideal $I \neq 0$ is invertible by arguing by contradiction: if $I * I^{-1} \neq D$, we can find an element $x \in K \setminus D$ which is in $(I * I^{-1})^{-1}$ thanks to `exists_not_mem_one_of_ne_bot`; some easy algebraic manipulation then implies that x is actually integral over D. Since D is integrally closed, $x \in D$, contradicting the construction of x. Combining these results gives the equivalence between the two conditions for being a Dedekind domain.

4.4 Unique ideal factorization

As briefly indicated before, we also formalized a proof that in a Dedekind domain every nonzero ideal can be expressed as a product of prime ideals in a unique way up to the order of the factors. In fact, for an integral domain, every nonzero ideal is a product of prime ideals if and only if all nonzero fractional ideals are invertible; the uniqueness follows separately. We have formalized one direction of this equivalence, a proof of the converse can be found in [10, Chapter 5, §6, Theorem 10].

We formalized the unique ideal factorization property of a Dedekind domain D by instantiating a unique factorization monoid structure on its ideals.

```
instance ideal.unique_factorization_monoid :
  unique_factorization_monoid (ideal D)
```

In mathlib, unique factorization domains are actually a special case of unique factorization monoids (UFMs). A commutative monoid R with an absorbing element 0 and injective multiplication is defined to be a UFM, if the relation "x properly divides y" is well-founded (implying that every element can be factored as a product of irreducibles) and an element of R is prime if and only if it is irreducible (implying uniqueness of the factorization). Examples in mathlib of UFMs are the unique factorization domains $\mathbb{N}$ and $\mathbb{Z}$ as well as, for any UFM α, the quotient of α by the subgroup of invertible elements `associates` α. With much of the necessary definitions and properties already formalized, the formalization of this unique factorization result has been done

in well under 100 lines of Lean code. One of the main mathematical ingredients (interesting in its own right) is that for ideals in a Dedekind domain, to divide is to contain:

```
lemma ideal.dvd_iff_le {I J : ideal D} : (I ∣ J) ↔ J ≤ I
```

Similarly, to strictly contain is to properly divide, so the well-foundedness condition of UFMs is exactly the property that a Dedekind domain is Noetherian. In order to show that all irreducible elements of the monoid of nonzero prime ideals in D are prime elements, we formalized that irreducible ideals in a Dedekind domain are maximal and therefore prime (note that prime ideals of a Dedekind domain D coincide with prime elements of the monoid of its nonzero ideals); the converse holds in every monoid.

We note that the unique factorization result, or actually an easy corollary thereof, is an important ingredient in our finiteness proof for the class group of rings of integers, as we will elaborate on in Section 7.2.

5 Principal ideal domains are Dedekind

As an example of our definitions, we discuss in some detail our formalization of the fact that a principal ideal domain is a Dedekind domain. In the same way that unique factorization domains are generalized in mathlib to unique factorization monoids, there is no explicit definition of PIDs in mathlib, rather it is split up into multiple hypotheses. One uses `[comm_ring R] [is_domain R] [is_principal_ideal_ring R]` to denote a PID R, where `is_domain` is a typeclass asserting that the ring is nontrivial and there are no zero divisors, where `is_principal_ideal_ring` is a typeclass defined for all commutative rings:

```
class is_principal_ideal_ring (R : Type*) [comm_ring R] :=
(principal : ∀ (I : ideal R), is_principal I)
```

Our proof that the hypotheses `[comm_ring R] [is_domain R] [is_principal_ideal_ring R]` imply `is_dedekind_domain R` was relatively short:

```
instance principal_ideal_ring.to_dedekind_domain (R : Type*)
  [comm_ring R] [is_domain R] [is_principal_ideal_ring R] :
  is_dedekind_domain R :=
⟨principal_ideal_ring.is_noetherian_ring,
 dimension_le_one.principal_ideal_ring R,
 unique_factorization_monoid.is_integrally_closed⟩
```

The Noetherian property of a Dedekind domain followed easily by the previously defined lemma `principal_ideal_ring.is_noetherian_ring`, since, by definition, each ideal in a principal ideal ring is finitely generated (by a single element).

We proved the lemma `dimension_le_one.principal_ideal_ring`, which is an instantiation of the existing result `is_prime.to_maximal_ideal`, showing

that a nonzero prime ideal in a PID is maximal. The latter lemma uses the characterization that I is a maximal ideal if and only if any strictly larger ideal $J \supsetneq I$ is the full ring R. If I is a nonzero prime ideal and $J \supsetneq I$ in the PID R, we see that a generator j of J is a divisor of any generator i of I. Since I is prime, this implies that either $j \in I$, contradicting the assumption that $J \supsetneq I$, or $i = 0$, contradicting that I is nonzero, or finally that j is a unit, implying $J = R$ as desired.

The final condition of a PID being integrally closed was the most challenging. We used the previously defined instance `principal_ideal_ring.to_unique_factorization_monoid` to deduce that a PID is a unique factorization monoid (UFM), to instantiate our proof that every UFM is integrally closed. In a PID, the Noetherian property implies that the division relation is well-founded, and `principal_ideal_ring.irreducible_iff_prime` shows that irreducible elements and prime elements coincide. To prove that an irreducible element p is prime, the proof uses that prime elements generate prime ideals and irreducible elements of a PID generate maximal ideals. Since all maximal ideals are prime ideals, the ideal generated by p is maximal, hence prime, thus p is prime. We proved the lemma `irreducible_of_prime`, which shows the converse holds in any commutative monoid with zero.

To show that a UFM is integrally closed, we first formalized the Rational Root Theorem, named `denom_dvd_of_is_root`, which states that for a polynomial $p : R[X]$ and an element of the fraction field $x : \operatorname{Frac} R$ such that $p(x) = 0$, the denominator of x divides the leading coefficient of p. If x is integral with minimal polynomial p, the leading coefficient is 1, therefore the denominator is a unit and x is an element of R. This gave us the required lemma `unique_factorization_monoid.integrally_closed`, which states that the integral closure of R in its fraction field is R itself.

6 Rings of integers are Dedekind domains

An important classical result in algebraic number theory is that the ring of integers of a number field K, defined as the integral closure of $\mathbb{Z}$ in K, is a Dedekind domain. We formalized a stronger result: given a Dedekind domain D and a field of fractions F, if K is a finite separable extension of F, then the integral closure of D in K is a Dedekind domain with fraction field K. Our approach was adapted from Neukirch [7, Theorem 3.1]. Throughout this section, let D be a Dedekind domain with a field of fractions F, K a finite, separable field extension of F and let S denote the integral closure of D in K.

The first step was to show that K is a field of fractions for the integral closure, namely, that there is an instance `is_fraction_ring_of_finite_extension D F K : is_fraction_ring S K`. The main content of `is_fraction_ring_of_finite_extension` consisted of showing that all elements $x : K$ can be written as y/z for elements $y \in S$, $z \in D \subseteq S$; the standard proof of this fact (see [11, Theorem 15.29]) formalized readily.

We could then show that the integral closure of D in K is a Dedekind domain, by proving it is integrally closed in K, has Krull dimension at most 1 and is Noetherian. The fact that the integral closure is integrally closed was immediate.

To show the Krull dimension is at most 1, we needed to develop basic going-up theory for ideals. In particular, we showed that an ideal I in an integral extension is maximal if it lies over a maximal ideal, and used a result already available in mathlib that a prime ideal I in a ring extension lies over a prime ideal.

```
lemma is_maximal_of_is_integral_of_is_maximal_comap
  [algebra R S] (hRS : algebra.is_integral R S)
  (I : ideal S) [is_prime I]
  (hI : is_maximal (comap (algebra_map R S) I)) : is_maximal I
theorem is_prime.comap (f : R →+* S) (I : ideal S)
  [hI : is_prime I] : is_prime (comap f I)
```

The final condition, that the integral closure S of D in L is a Noetherian ring, required the most work. We started by following the first half of Dummit and Foote [11, Theorem 15.29], so that it sufficed to find a non-degenerate bilinear form B such that all integral $x, y : K$ satisfy $B(x, y) \in$ `integral_closure` D K. We then formalized the results in Neukirch [7, Sections 2.5–2.8] to show that the *trace form* is a bilinear form satisfying these requirements.

6.1 The trace form

In the notation from the previous section, consider the bilinear map `lmul := λ x y : K, x * y`. The trace of the linear map `lmul x` is called the *algebra trace* $\mathrm{Tr}_{K/F}(x)$ of x. We defined the algebra trace as a linear map, in this case from K to F:

```
noncomputable def trace : K →ₗ[F] F :=
linear_map.comp (linear_map.trace F K)
  (to_linear_map (lmul F K))
```

This definition was marked noncomputable since `linear_map.trace` makes a case distinction on the existence of a finite basis, choosing an arbitrary finite basis if one exists (since the value of `linear_map.trace` does not depend on this choice) and returning 0 otherwise. This latter case did not occur in our development.

We defined the *trace form* to be an F-bilinear form on K, mapping $x, y : K$ to $\mathrm{Tr}_{K/F}(xy)$.

```
noncomputable def trace_form : bilin_form F K :=
{ bilin := λ x y, trace F K (x * y), .. /- proofs omitted -/ }
```

In the following, let $L/K/F$ be a tower of finite extensions of fields, namely we assume `[algebra F K] [algebra K L] [algebra F L] [is_scalar_tower F K L]`, as described in Section 3.3.

The value of the trace depends on the choice of F and K; we formalized this as lemmas `trace_algebra_map x : trace F K (algebra_map F K x) = finrank F K * x` as well as `trace_trace x : trace F K (trace (K L x)) = trace F L x`. These results followed by direct computation.

To compute $\mathrm{Tr}_{K/F}(x)$, it therefore suffices to consider the trace of x in the smallest field containing x and F, which is the monogenic extension $F(x)$ discussed in Section 3.7. There is a nice formula for the trace in $F(x)$, although the terms in this formula are elements in a larger field L (such as the *splitting field* of `minpoly F x`, the minimal polynomial of x over F). In formalizing this formula, we first mapped the trace to L using the embedding `algebra_map F L`, which gave the following statement:

```
lemma power_basis.trace_gen_eq_sum_roots (pb : power_basis F K)
  (h : polynomial.splits (algebra_map F L) (minpoly F pb.gen)) :
  algebra_map F L (trace F K pb.gen) =
    sum (roots (map (algebra_map F L) (minpoly F pb.gen)))
```

We formulated the lemma in terms of the power basis, since we needed to use it for $F(x)$ here and for an arbitrary finite separable extension L/K later in the proof.

The elements of `roots (map (algebra_map F L) (minpoly F pb.gen))` are called *conjugates* of x in L. Each conjugate of x is integral since it is a root of the same monic polynomial, and integer multiples and sums of integral elements are integral. Combining `trace_gen_eq_sum_roots` and `trace_algebra_map` showed that the trace of x is an integer multiple (namely `finrank F(x) L`) of a sum of conjugate roots, hence we concluded that the trace (and trace form) of an integral element is also integral.

Finally, we showed that the trace form is nondegenerate, following Neukirch [7, Proposition 2.8]. Since K/F is a finite, separable field extension, it has a power basis `pb` generated by an element $x : K$. Letting x_k denote the k-th conjugate of x in an algebraically closed field $L/K/F$, the main difficulty was in checking the equality $\sum_k x_k^{i+j} = \mathrm{Tr}_{K/F}(x^{i+j})$. Directly applying `trace_gen_eq_sum_roots` was tempting, since we had a sum over conjugates of powers on both sides. However, the two expressions did not precisely match: the left hand side is a sum of conjugates of x, where each conjugate is raised to the power $i + j$, while the conclusion of `trace_gen_eq_sum_roots` resulted in a sum over conjugates of x^{i+j}.

Instead, the paper proof switched here to an equivalent definition of conjugate: the conjugates of x in L are the images (counted with multiplicity) of x under each embedding $\sigma\colon F(x) \to L$ that fixes F. This equivalence between the two notions of conjugate was contributed to mathlib by the Berkeley group in the week before we realized we needed it. Mapping `trace_gen_eq_sum_roots` through the equivalence gave $\mathrm{Tr}_{K/F}(x) = \sum_\sigma \sigma x$. Since each σ is a ring homomorphism, $\sigma\ x^{i+j} = (\sigma\ x)^{i+j}$, so the conjugates of x^{i+j} are the $(i + j)$-th powers of conjugates of x, which concluded the proof.

7 Class group and class number

7.1 The class group

Recall from Section 2 that the ideal class group $\mathcal{C}l_D$ of a Dedekind domain D is the quotient of the group of nonzero fractional ideals of D by the nonzero principal fractional ideals. More generally, given an integral domain R with fraction field K, we can define the class group $\mathcal{C}l_R$ as the quotient of the invertible fractional ideals by the nonzero principal fractional ideals. We formalized this in Lean by first defining a map `to_principal_ideal R K : K`$^\times$ `→ (fractional_ideal R K)`$^\times$, and defined the class group as

```
def class_group (R K : Type*) [comm_ring R] [is_domain R]
  [field K] [algebra R K] [is_fraction_ring R K] :=
(fractional_ideal R K)^× / (range (to_principal_ideal R K))
```

Here, `R`$^\times$ for a semiring R denotes the multiplicative group of its invertible elements. Recall from Section 4.2 that in the general case of an integral domain R the type of fractional ideals of R is endowed with the structure of a commutative semiring. Therefore, the quotient of the abelian group `(fractional_ideal R K)`$^\times$ by the subgroup of nonzero principal fractional ideals is well-defined. In the case where R is a Dedekind domain, we provided a map `class_group.mk0` sending nonzero integral ideals of R to the corresponding class in the class group.

7.2 Finiteness results

In general, Dedekind domains can have infinite class groups: in fact, a celebrated result by Claborn shows that for every abelian group G, there exists a Dedekind domain D with $\mathcal{C}l_D \cong G$ (see [12, Theorem 7]). For an extreme — but somewhat classical — example, the Dedekind domain

$$D = \mathbb{C}[X, Y]/(Y^2 - X^3 - X - 1)$$

has class group isomorphic to $\mathbb{C}/\mathbb{Z}^2$. However, as discussed in Section 2, the rings of integers of global fields have finite class groups.

We let K be a number field and let K' be a function field, with ring of integers $\mathcal{O}_K$ and $\mathcal{O}_{K'}$ (we fix a choice of a model $\mathbb{F}_q[t]$), respectively. Most proofs of the finiteness of $\mathcal{C}l_{\mathcal{O}_K}$ available in a modern textbook (see [7, Theorems 4.4, 5.3, 6.3]) depend on Minkowski's lattice point theorem, a result from the geometry of numbers (which has been formalized in Isabelle/HOL [13]). Extending this proof to show the finiteness of $\mathcal{C}l_{\mathcal{O}_{K'}}$ is quite involved and does not result in a uniform proof for $\mathcal{C}l_{\mathcal{O}_K}$ and $\mathcal{C}l_{\mathcal{O}_{K'}}$. Our formalization instead adapted and generalized a classical approach to the finiteness of $\mathcal{C}l_{\mathcal{O}_K}$, where the use of Minkowski's theorem is replaced by the pigeonhole principle. For an informal writeup of the proof, used in the formalization efforts, see https://github.com/lean-forward/class-number-journal/blob/jar-reviews/FiniteClassGroup.pdf. The classical approach seems to go back to Kronecker

and can be found, for instance, in [14]. We note that some other "uniform" approaches can be found in [15] and [16].

Let D be an Euclidean domain: in particular, it will be a PID and hence a Dedekind domain. Given a fraction field F of D, let K be a finite separable field extension of F. We formalized, in the theorem `class_group.fintype_of_admissible_of_finite`, that the integral closure S of D in K has a finite class group whenever D has an "admissible" absolute value `abs`. This notion originated in our project from the adaptation and generalization of the classical finiteness proof in interaction with the formalization efforts. Very informally, the admissibility conditions require that the remainder operator `%` produces values that are not too far apart. More precisely, and in more "ordinary" mathematical notation, writing for instance mod instead of % and $x \mapsto |x|$ for the absolute value function $D \to \mathbb{Z}$, the latter is called admissible if both:

- we have a function card : $\mathbb{R}_{>0} \to \mathbb{N}$;
- for all $n \in \mathbb{N}$, $\epsilon \in \mathbb{R}_{>0}$, $b \in D - \{0\}$, and $A \in D^n$, there exists a function $t : \mathbb{N}_{<n} \to \mathbb{N}_{<card(\epsilon)}$, such that for all indices $i_0, i_1 \in \mathbb{N}_{<n}$ with $t(i_0) = t(i_1)$ we have that
$$|A_{i_1} \bmod b - A_{i_0} \bmod b| < \epsilon |b|.$$

Formally, we translated this, making minor modifications like turning card into a total function on $\mathbb{R}$, to defining the predicate classifying admissible absolute values `abv` on D as follows:

```
structure is_admissible (abv : absolute_value D ℤ) extends
    is_euclidean abv :=
(card : ℝ → ℕ) (exists_partition' :
  ∀ (n : ℕ) {ε : ℝ} (hε : 0 < ε) {b : D} (hb : b ≠ 0)
  (A : fin n → D), ∃ (t : fin n → fin (card ε)),
  ∀ i₀ i₁, t i₀ = t i₁ →
  (abv (A i₁ % b - A i₀ % b) : ℝ) < abv b · ε)
```

The `is_euclidean abv` predicate asserts that the absolute value `abv : D → ℤ` respects the remainder operator of the Euclidean domain D, in particular `abv (a % b) < abv b`.

The above condition formalizes and generalizes an intermediate result in paper proofs of the finiteness of the class group; the different proofs for number fields and function fields (still assuming K/F separable) become the same after this point. We used division with remainder to replace the *fractional part* operator on F in the classical proof, which was essential to incorporate function fields, and at the same time allowed our proof to stay entirely within D to avoid coercions.

In a similar way to the algebra trace of Section 6.1, we defined the norm of an element `x : S` as the determinant of the linear map `lmul x`. We used the admissibility of `abs` to find a finite set `finset_approx` of elements of D, such that the following generalization of [14, Theorem 12.2.1] holds.

```
theorem exists_mem_finset_approx' (a b : S) (hb : b ≠ 0) :
```

```
∃ (q : S) (r ∈ finset_approx),
abv (algebra.norm D (r · a - q * b)) < abv (algebra.norm D b)
```

Translated back into more "ordinary" mathematical notation, this theorem tells us that, for all $a, b \in S$ with b nonzero, there exist $q \in S$ and $r \in$ finset_approx, such that

$$|\mathrm{Norm}_{S/D}(ra - qb)| < |\mathrm{Norm}_{S/D}(b)|.$$

After this, the classical approach mentioned above formalized smoothly: we show that each class in $\mathcal{C}l_K$ contains an ideal J with $M \in J$, where M is the product of all elements of `finset_approx`, hence M is nonzero. Since the ideals of the Dedekind domain S have unique factorization, the nonzero ideal $\langle M \rangle$ spanned by M has only finitely many divisors. To contain is to divide in Dedekind domains, so there are only finitely many ideals J with $M \in J$. Thus, we concluded that $\mathcal{C}l_K$ is finite under the condition of the existence of an admissible absolute value.

It remained to define an admissible absolute value for $\mathbb{Z}$ and $\mathbb{F}_q[t]$. On $\mathbb{Z}$, it was straightforward to formalize that the usual Archimedean absolute value fulfills the requirements. For $\mathbb{F}_q[t]$, we showed that $|f|_{\deg} := q^{\deg f}$ for $f \in \mathbb{F}_q[t]$ is an admissible absolute value; we note that this was somewhat more involved to formalize. We concluded that when K is a global field, restricting to *separable* extensions of $\mathbb{F}_q(t)$ in the function field case (but see the remark below), the class group is finite:

```
noncomputable instance : fintype
  (class_group (number_field.ring_of_integers K) K) :=
class_group.fintype_of_admissible_of_finite ℚ K
  absolute_value.abs_is_admissible

noncomputable instance : fintype
  (class_group (function_field.ring_of_integers Fq F) F) :=
class_group.fintype_of_admissible_of_finite (ratfunc Fq) F
  polynomial.card_pow_degree_is_admissible
```

Finally, we defined `number_field.class_number` and `function_field.class_number` as the cardinality of the respective class groups.

We remark that it is possible to get rid of the `[is_separable F K]` assumption above. For instance, using that any function field K, given as finite extension of $\mathbb{F}_q(t)$, contains an $s \in K$ such that $K/\mathbb{F}_q(s)$ is a finite *and separable* extension; see for example [17, Corollary 4.4 in Chapter VIII] (noting that $\mathbb{F}_q$ is perfect and K has transcendence degree 1 over $\mathbb{F}_q$). One then also needs to show that finiteness of the class group of the integral closure of $\mathbb{F}_q[s]$ in K is preserved upon replacing $\mathbb{F}_q[s]$ by $\mathbb{F}_q[t]$. A trivial way to get rid of the assumption in the statement above is to simply move it to our definition of function field. While this would be mathematically consistent by the result just cited, we did not opt to do this (for instance showing a finite extension of a function

field is a function field would become nontrivial). Alternatively, one could aim at dropping the separability condition in the formalized result mentioned in the first paragraph of Section 6. Having a formalization of this generalization would be interesting in its own right. This approach would also still need the adaptation of some of the details in the final steps for the finiteness of the class group in the admissible case.

We rounded off our development by determining the class number in the simplest possible case: the rational numbers $\mathbb{Q}$. First, we formalized the theorem `class_number_eq_one_iff`, stating that the class number of K is 1 if and only if $\mathcal{O}_K$ is a principal ideal domain. After defining the isomorphism `rat.ring_of_integers_equiv` showing $\mathcal{O}_{\mathbb{Q}}$ is $\mathbb{Z}$, we could use the fact that $\mathbb{Z}$ is a PID to conclude that the class number of $\mathbb{Q}$ is equal to 1:

```
theorem rat.class_number : number_field.class_number ℚ = 1 :=
class_number_eq_one_iff.mpr
  (is_principal_ideal_ring.of_surjective _
    rat.ring_of_integers_equiv.symm.surjective)
```

8 Discussion

8.1 Related work

Broadly speaking, one could see our formalization work as part of number theory. There are several formalization results in this direction. Most notably, Eberl formalized a substantial part of analytic number theory in Isabelle/HOL [18]. Narrowing somewhat to a more algebraic setting, Cano, Cohen, Dénès, Mörtberg and Siles formalized constructive definitions in ring theory, most notable for our discussion being the Krull dimension [19]. We are not aware of any other formal developments of fractional ideals, Dedekind domains or class groups of rings of integers.

There are many libraries formalizing basic notions of commutative algebra such as field extensions and ideals, including the Mathematical Components library in Coq [20], the algebraic library for Isabelle/HOL [21], the `set.mm` database for MetaMath [22] and the Mizar Mathematical Library [23]. The field of algebraic numbers, or more generally algebraic closures of arbitrary fields, are also available in many provers. For example, Blot [24] formalized algebraic numbers in Coq, Cohen [25] constructed the subfield of real algebraic numbers in Coq, Thiemann, Yamada and Joosten [26] formalized algebraic numbers in Isabelle/HOL, Carneiro [27] in MetaMath, and Watase [28] in Mizar. To our knowledge, the Coq Mathematical Components library is the only formal development beside ours specifically dealing with number fields [20, `field/algnum.v`].

Apart from the general theory of algebraic numbers, there are formalizations of specific rings of integers. For instance, the Gaussian integers $\mathbb{Z}[i]$ have been formalized in Isabelle/HOL by Eberl [29], in MetaMath by Carneiro [30] and in Mizar by Futa, Mizushima, and Okazaki [31]. Eberl's Isabelle/HOL

formalization deserves special mention in this context since it introduces techniques from algebraic number theory, defining the integer-valued norm on $\mathbb{Z}[i]$ and classifying the prime elements of $\mathbb{Z}[i]$.

Finally, since our project became available in the mathlib library, a team led by Brasca has begun formalizing Fermat's Last Theorem for regular primes. Fermat's Last Theorem is the assertion that, for all integers $n \geq 3$,

$$\forall x, y, z \in \mathbb{Z}, \quad x^n + y^n = z^n \Longrightarrow x \cdot y \cdot z = 0. \tag{FLT$_n$}$$

It is immediate to see that, for positive integers n and m, if n divides m, then the validity of (FLT$_n$) implies that of (FLT$_m$). Therefore, also taking into account that (FLT$_4$) was already dealt with by Fermat himself, it suffices to only consider exponents that are odd prime numbers.

Now, an odd prime number p is said to be *regular* if it does not divide the order of the class group $\mathcal{C}l_{\mathbb{Q}(\zeta_p)}$ of the number field obtained by adjoining to $\mathbb{Q}$ a *primitive pth root of unity* ζ_p; the latter means that $\zeta_p^p = 1$ and $\zeta_p \neq 1$ or, equivalently, that ζ_p is a root of the irreducible polynomial

$$\frac{X^p - 1}{X - 1} = X^{p-1} + X^{p-2} + \cdots + X + 1.$$

A classical result, due to Kummer's work in 1847, is that (FLT$_p$) is true for every *regular* prime number p. This is the result Brasca and his team have begun formalizing in Lean 3 in the on-going work [32], and it evidently requires the finiteness of the class group in order to define the notion of a *regular prime* as above[5]. Moreover, most arguments occurring in Kummer's proof pertain to the structure of the ring of integers $\mathbb{Z}[\zeta_p] = \mathcal{O}_{\mathbb{Q}(\zeta_p)}$ as a Dedekind domain, and our work lies at the core of the formalization of these structures.

8.2 Future directions

Having formalized various basic results of algebraic number theory, there are several natural directions for future work, including formalizing some of the following results.

- The group of units of the ring of integers $\mathcal{O}_K^\times$ in a number field K is finitely generated, or even Dirichlet's unit theorem [7, Theorem 7.4], stating that $\mathcal{O}_K^\times$ has rank $r + s - 1$ and that its torsion subgroup is the cyclic group of roots of unity in K. Here r denotes the number of real embeddings of K and s the number of conjugate pairs of complex non-real embeddings of K. The finite generation result also holds in function fields, again with a precise description of the rank and of the torsion.

[5] It is actually possible to simply *define* a regular prime only in terms of divisibility of some Bernoulli numbers, instead of mentioning class groups. But this definition would at any rate need to be translated in terms of class numbers in order to implement Kummer's proof.

- Other finiteness results in algebraic number theory, most notably Hermite's theorem about the existence of finitely many number fields, up to isomorphism, with bounded discriminant [7, Theorem 2.16]. While this could be done without interpreting the primes dividing the discriminant as the primes that *ramify* in the number field, it would certainly be interesting to set up some basic ramification theory: on the one hand, this would also prove essential for many other developments and, on the other, it would allow to prove a version of Hermite's theorem stating that, up to isomorphism, there are only finitely many number fields with bounded degree and restricted ramification. As usual, there are analogous results in the function field setting, though they are less straightforward. One reason for this is that the nondegeneracy of the trace form from Section 6.1 does not hold any more when the separability condition is dropped.
- Class number computations, starting with, say, quadratic number fields. This could be a step towards the verification of correctness of number-theoretic software, such as KASH/KANT [33] and PARI/GP [34]. Along the same lines, unit group computations would also be of much interest, most notably the explicit computation of $r + s - 1$ generators for the free part of $\mathcal{O}_K^\times$. Restricting to quadratic fields, we see that the rank is positive (and equal to 1) if and only if the field is of the shape $\mathbb{Q}(\sqrt{d})$ for some positive integer d that is not a square. Finding a generator can be done by using continued fractions, of which the basics are already implemented in Lean by Kevin Kappelmann, though certifying that a given (perhaps externally computed) element is indeed a generator could also be done without continued fractions.
- Applications of algebraic number theory to solving Diophantine equations, such as determining all pairs of integers (x, y) such that $y^2 = x^3 + D$ for some nonzero $D \in \mathbb{Z}$. It would be interesting to deal with some values of D where no elementary techniques are available and where factorization in the ring of integers of $\mathbb{Q}(\sqrt{D})$, along with information about the class number, could solve the equation.

8.3 Conclusion

In this project, we confirmed the rule that the hardest part of formalization is to get the definitions right. Once this is accomplished, the paper proof (sometimes first adapted with formalization in mind) almost always translates into a formal proof without too much effort. In particular, we regularly had to invent abstractions to treat instances of the "same" situation uniformly. Instead of fixing a canonical representation, be it $F \subseteq K \subseteq L$ as subfields or the field of fractions $\operatorname{Frac} R$, or the monogenic $K(\alpha)$, we found that making the essence of the situation an explicit parameter, as in `is_scalar_tower`, `is_fraction_ring` or `power_basis`, allows to treat equivalent viewpoints uniformly without the need for transferring results.

The formalization efforts described in this paper cannot be cleanly separated from the development of mathlib as a whole. The decentralized

organization and highly integrated design of mathlib meant that we could contribute our formalizations as we completed them, resulting in a quick integration into the rest of the library. Other contributors building on these results often extended them to meet our requirements, before we could identify that we needed them, as the anecdote in Section 3.5 illustrates. In other words, the low barriers for contributions ensured mutually beneficial collaboration.

Quantifying the ratio between the length of our formal proofs and their paper counterparts in an accurate and meaningful way will be very difficult as background assumptions and levels of detail varied significantly. We actually did not always literally follow some written text, but deviated from the paper mathematics (often discussed orally, on blackboards, through Zulip, etc.) on many occasions. An important aspect we had to take into account was to consistently combine different descriptions of mathematical objects from different sources. The formalization project described in this paper resulted in the contribution of thousands of lines of Lean code involving hundreds of declarations. A rough estimate concerning the former would be that about five thousand lines of project specific code were added, and about half of that number of lines of more generic background code. We validated existing design choices used in mathlib, refactored those that did not scale well and contributed our own set of designs. The real achievement was not to complete each proof, but to build a better foundation for formal mathematics.

Acknowledgements. We would like to thank Jasmin Blanchette and the anonymous reviewers for useful comments on previous versions of the manuscript, which found their way into this paper.
A. N. would like to thank Prof. Kevin Buzzard for his constant support and encouragement, and for introducing her to the other co-authors.
A. N. and F. N. wish to express their deepest gratitude to Anne Baanen for the generosity shown along all stages of the project. Without Anne's never-ending patience, it would have been impossible for them to contribute to this project, and to overcome several difficulties.
Finally, we would like to thank the whole mathlib community for invaluable advice all along the project.

Declarations.

Funding

Anne Baanen was funded by NWO Vidi grant No. 016.Vidi.189.037, Lean Forward.
Sander R. Dahmen was funded by NWO Vidi grant No. 639.032.613, New Diophantine Directions.
Ashvni Narayanan was funded by EPSRC Grant EP/S021590/1 (UK).

Conflicts of interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Availability of data and material

See *Code availability.*

Code availability

Full source code of the formalization is maintained as part of mathlib, https://github.com/leanprover-community/mathlib. Copies of the source files relevant to this paper are available in a separate repository at https://github.com/lean-forward/class-number-journal.

Authors' contributions

All authors contributed to the formalization project and to the initial draft of the manuscript. Anne Baanen and Sander R. Dahmen wrote the extra material for the extended version. All authors commented on previous versions of the manuscript.

References

[1] The mathlib Community: The Lean mathematical library. In: Blanchette, J., Hriţcu, C. (eds.) CPP 2020, pp. 367–381. ACM, New York, USA (2020). https://doi.org/10.1145/3372885.3373824

[2] de Moura, L., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The Lean theorem prover (system description). In: Felty, A.P., Middeldorp, A. (eds.) Automated Deduction - CADE-25. LNCS, vol. 9195, pp. 378–388. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_26

[3] Baanen, T., Dahmen, S.R., Ashvni N., Nuccio Mortarino Majno di Capriglio, F.A.E.: A Formalization of Dedekind Domains and Class Groups of Global Fields. In: Cohen, L., Kaliszyk, C. (eds.) ITP 2021. LIPIcs, vol. 193, pp. 5–1519. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.ITP.2021.5. https://drops.dagstuhl.de/opus/volltexte/2021/13900

[4] Wadler, P., Blott, S.: How to make ad-hoc polymorphism less ad hoc. In: Principles of Programming Languages. POPL '89, pp. 60–76. ACM, Austin, TX, USA (1989). https://doi.org/10.1145/75277.75283

[5] Avigad, J., de Moura, L., Kong, S.: Theorem Proving in Lean. Carnegie Mellon University, Pittsburgh, PA, USA (2021). Release 3.23.0, https://leanprover.github.io/theorem_proving_in_lean/

[6] Baanen, T.: Use and abuse of instance parameters in the Lean mathematical library. CoRR **abs/2202.01629** (2022) https://arxiv.org/abs/2202.01629. Accepted for publication at ITP 2022, Haifa, Israel.

[7] Neukirch, J.: Algebraic Number Theory. Fundamental Principles of Mathematical Sciences, vol. 322, p. 571. Springer, Cham (1999). 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

[8] Fröhlich, A.: Local fields. In: Algebraic Number Theory (Proc. Instructional Conf., Brighton, 1965), pp. 1–41. Thompson, Washington, D.C. (1967)

[9] Lewis, R.Y., Madelaine, P.: Simplifying casts and coercions (extended abstract). In: Fontaine, P., Korovin, K., Kotsireas, I.S., Rümmer, P., Tourret, S. (eds.) Practical Aspects of Automated Reasoning. CEUR Workshop Proceedings, vol. 2752, pp. 53–62. CEUR-WS.org, Aachen, Germany (2020). http://ceur-ws.org/Vol-2752/paper4.pdf

[10] Zariski, O., Samuel, P.: Commutative Algebra, Volume I. The University Series in Higher Mathematics, p. 329. D. Van Nostrand Company, Inc., Princeton, NJ, USA (1958)

[11] Dummit, D.S., Foote, R.M.: Abstract Algebra, 3rd edn., p. 932. John Wiley & Sons, Inc., Hoboken, NJ, USA (2004)

[12] Claborn, L.: Every abelian group is a class group. Pacific Journal of Mathematics **18**(2), 219–222 (1966). https://doi.org/pjm/1102994263

[13] Eberl, M.: Minkowski's theorem. Archive of Formal Proofs (2017). https://isa-afp.org/entries/Minkowskis_Theorem.html, Formal proof development

[14] Ireland, K., Roosen, M.: A Classical Introduction to Modern Number Theory, 2nd edn. Springer, Cham (1990)

[15] Artin, E., Whaples, G.: Axiomatic characterization of fields by the product formula for valuations. Bull. Amer. Math. Soc. **51**(7), 469–492 (1945)

[16] Stasinski, A.: A uniform proof of the finiteness of the class group of a global field. Amer. Math. Monthly **128**(3), 239–249 (2021). https://doi.org/10.1080/00029890.2021.1855036

[17] Lang, S.: Algebra, 3rd edn. Graduate Texts in Mathematics, vol. 211, p. 914. Springer, Cham (2002). https://doi.org/10.1007/978-1-4613-0041-0. https://doi.org/10.1007/978-1-4613-0041-0

[18] Eberl, M.: Nine chapters of analytic number theory in Isabelle/HOL. In: Harrison, J., O'Leary, J., Tolmach, A. (eds.) ITP 2019. LIPIcs, vol.

141, pp. 16–11619. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2019). https://doi.org/10.4230/LIPIcs.ITP.2019.16

[19] Cano, G., Cohen, C., Dénès, M., Mörtberg, A., Siles, V.: Formalized linear algebra over elementary divisor rings in Coq. Logical Methods in Computer Science **12**(2) (2016). https://doi.org/10.2168/LMCS-12(2:7)2016

[20] Mahboubi, A., Tassi, E.: The Mathematical Components Libraries. Zenodo, Genève, Switzerland (2017). https://doi.org/10.5281/zenodo.4457887

[21] Ballarin (editor), C., Aransay, J., Baillon, M., de Vilhena, P.E., Hohe, S., Kammüller, F., Paulson, L.C.: The Isabelle/HOL Algebra Library. http://isabelle.in.tum.de/dist/library/HOL/HOL-Algebra/index.html

[22] Megill, N.D., Wheeler, D.A.: Metamath: A Computer Language for Mathematical Proofs. Lulu Press, Morrisville, NC, USA (2019). http://us.metamath.org/downloads/metamath.pdf

[23] Grabowski, A., Kornilowicz, A., Schwarzweller, C.: On algebraic hierarchies in mathematical repository of Mizar. In: Ganzha, M., Maciaszek, L., Paprzycki, M. (eds.) Proceedings of the 2016 Federated Conference on Computer Science and Information Systems. ACSIS, vol. 8, pp. 363–371 (2016)

[24] Blot, V.: Basics for algebraic numbers and a proof of Liouville's theorem in C-CoRN. MSc internship report (2009)

[25] Cohen, C.: Construction of real algebraic numbers in Coq. In: Beringer, L., Felty, A.P. (eds.) ITP 2012. Lecture Notes in Computer Science, vol. 7406, pp. 67–82. Springer, Cham (2012). https://doi.org/10.1007/978-3-642-32347-8_6

[26] Thiemann, R., Yamada, A., Joosten, S.: Algebraic numbers in Isabelle/HOL. Archive of Formal Proofs (2015). https://isa-afp.org/entries/Algebraic_Numbers.html, Formal proof development

[27] Carneiro, M.: Definition `df-aa`. http://us.metamath.org/mpeuni/df-aa.html

[28] Watase, Y.: Algebraic numbers. Formalized Mathematics **24**(**4**), 291–299 (2016). https://doi.org/10.1515/forma-2016-0025

[29] Eberl, M.: Gaussian integers. Archive of Formal Proofs (2020). https://isa-afp.org/entries/Gaussian_Integers.html, Formal proof development

[30] Carneiro, M.: Definition `df-gz`. http://us.metamath.org/mpeuni/df-gz.html

[31] Futa, Y., Mizushima, D., Okazaki, H.: Formalization of Gaussian integers, Gaussian rational numbers, and their algebraic structures with Mizar. In: 2012 International Symposium on Information Theory and Its Applications, pp. 591–595 (2012)

[32] Brasca, R.a.o.: Fermat's Last Theorem for regular primes. https://github.com/leanprover-community/flt-regular/

[33] Pohst, M.E., et al.: The Computer Algebra System KASH/KANT. http://www.math.tu-berlin.de/~kant

[34] The PARI Group: PARI/GP Version `2.11.2`. Univ. Bordeaux (2019). The PARI Group. http://pari.math.u-bordeaux.fr/