Formalising Hilbert Polynomials in Lean

Fangming Li[1][0009−0006−0967−6746]⋆ and Jujian Zhang[2][0000−0001−7340−2703]

[1] Imperial College London, London SW7 2AZ, UK
fangming.li17@imperial.ac.uk
[2] Axiom Math, San Francisco, CA, USA
jujian@axiommath.ai

Abstract. In commutative algebra and algebraic geometry, Hilbert polynomials are an important tool used in the proof of Bézout's Theorem, the construction of Hilbert schemes, and so on. We have used the proof assistant Lean to formalise the Hilbert polynomial of a graded module. Several times, we discovered issues which were not present in the informal proofs but which mattered in the formalisation. As the notions of Hilbert polynomials associated with other algebraic structures rely on the version we have formalised, our project provides a foundation for the future formalisation of them.

Keywords: Lean · Formalisation · Commutative algebra · Algebraic geometry · Hilbert polynomial.

1 Introduction

1.1 Sequences that are Eventually Polynomials

The fundamental idea behind Hilbert polynomials is that, from some point onward, an infinite sequence may coincide with the values of a polynomial on natural numbers. For example, the power series expansion of $1/(1-x)$ is $1+x+x^2+\cdots$. The sequence of coefficients in this expansion is $1, 1, 1, \ldots$, which is given by the constant polynomial 1. The expansion of $1/(1-x)^2$ is $1+2x+3x^2+\cdots$, where the coefficient of x^n is $n+1$, a polynomial in n.

We may consider a slightly more complicated example: what is the power series expansion of $(1+x^3)/(1-x)^2$? By an elementary combinatorial argument, one finds that this expansion is $1+2x+3x^2+5x^3+7x^4+9x^5+\cdots$. Thus, for any natural number $n > 1$, the coefficient of x^n is $2n-1$, again a polynomial in n. In fact, for any polynomial $p(x)$ and any natural number d, there exists a

⋆ Corresponding author

This version of the contribution has been accepted for publication, after peer review (when applicable) but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/10.1007/978-3-032-38214-6_13. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms.

(unique) polynomial $h(x)$ such that the power series expansion of $p(x)/(1-x)^d$ agrees with $h(0)+h(1)x+h(2)x^2+h(3)x^3+\cdots$ for all terms whose degrees are larger than $deg(p)$.

Another source of examples comes from counting monomials. For instance, one may ask: for each natural number n, how many monic monomials in the variables x and y have degree n? When $n=0$, the only monic monomial is 1, so the answer is 1. When $n=1$, the answer is 2, since the only monic monomials of degree one are x and y. When $n=2$, we have x^2, y^2, and xy, so the answer is 3. A combinatorial argument shows that for any n, the number of monic monomials of degree n in the variables x and y is $n+1$, which is a polynomial in n. In fact, for any positive integer s, the number of monic monomials of degree n in the variables $x_1,\ldots,x_s$ is given by a polynomial in n.

We may slightly alter the question and ask: for each natural number n, how many monic monomials of degree n in the variables x, y, and z are *not* divisible by x^2y? When $n=0$, the answer is 1. When $n=1$, the answer is 3. When $n=2$, the answer is 6. However, using a combinatorial argument we have that, for every natural number $n\geq 3$, the answer is $3n$, again a polynomial in n.

In fact, all polynomials arising from the monomial counting problems described above are instances of Hilbert polynomials. Moreover, the existence of Hilbert polynomials is closely related to the power series expansions of expressions of the form $p(x)/(1-x)^d$.

The next subsection provides a brief introduction to Hilbert polynomials, including a motivating example and an overview of their importance in mathematics. Our aim there is only to give the reader an intuitive understanding of Hilbert polynomials. For this reason, the notion of an additive function will be mentioned without a detailed definition, and the full statement of the Hilbert–Serre Theorem will not be introduced at this stage; these will be presented in Section 2.

In 1.3, we briefly outline the main challenges encountered in our formalisation. 1.4 discusses related works. Section 2 presents the mathematical content formalised in our development, including the definitions of the main concepts and the statements of the main results. Section 3 describes our formalisation and provides a detailed discussion of the challenges and design choices in our development. Finally, Section 4 concludes the paper and introduces directions for future work.

1.2 About Hilbert Polynomials

The existence of Hilbert polynomials relies on the Hilbert–Serre Theorem. Roughly speaking, this theorem states that for a graded module $\oplus_{n=0}^{\infty}M_n$ and a suitable additive function μ that takes the graded pieces M_n as inputs, the power series $\Sigma_{n=0}^{\infty}\mu(M_n)x^n$ is equal to a rational expression of the form $p(x)/\Pi_{i=1}^{s}(1-x^{d_i})$. Consequently, if all the d_i are equal to 1, then the values of $\mu(M_n)$ are eventually given by a polynomial. This polynomial is unique and is called the Hilbert polynomial of $\oplus_{n=0}^{\infty}M_n$ with respect to μ [2, pp.116–118]. For the definition of

graded modules, we refer the reader to Atiyah and Macdonald's *Introduction to Commutative Algebra* [2].

To gain some intuition for the Hilbert–Serre Theorem, we revisit an example discussed earlier. The set of all polynomials in the variables x and y with real coefficients forms a ring, denoted by $\mathbb{R}[x, y]$. This ring can be written as a direct sum $\oplus_{n=0}^{\infty} R_n$, where each R_n consists of all polynomials p in $\mathbb{R}[x, y]$ such that every term of p has degree n. For instance, R_2 contains polynomials such as $x^2 - 2y^2$ and $3x^2 + 4xy + 100y^2$, while R_3 contains polynomials such as x^3, $x^2y + y^3$ and $4x^2y + 5xy^2 + y^3$. The ring $\mathbb{R}[x, y]$ is a graded module that satisfies all assumptions of the Hilbert–Serre Theorem. Moreover, there exists a suitable additive function μ such that $\mu(R_n)$ equals the number of monic monomials in x and y of degree n for each natural number n. As noted earlier, this number is always $n + 1$. Hence, the polynomial $h(x) := x + 1$ is the Hilbert polynomial of $\mathbb{R}[x, y]$ with respect to μ.

Hilbert polynomials are an important tool in commutative algebra and algebraic geometry. They help us define the degrees of projective algebraic varieties [6, pp.52], they provide a way to express the Krull dimensions of modules [10, pp.98], they allow us to construct Hilbert schemes [1, pp.6–7], and so on.

The concept of Hilbert polynomials is associated with quite a few different algebraic structures. In literature, people talk about Hilbert polynomials of projective algebraic varieties, Hilbert polynomials of coherent sheaves on projective schemes [6, pp.230], etc. However, all these notions rely on the version of Hilbert polynomials arising from the Hilbert–Serre Theorem. For example, given any projective variety X, its homogeneous coordinate ring $S(X)$ is a graded module satisfying the conditions in the Hilbert–Serre Theorem, and the Hilbert polynomial of X is defined as that of $S(X)$ [6, pp.52].

This paper presents a formalisation in the proof assistant Lean of Hilbert polynomials arising from the Hilbert–Serre Theorem. The content of our formalisation is available in an online repository [7].

1.3 Main Challenges in Our Formalisation

Lean [4] is a proof assistant based on dependent type theory. People in the Lean community sometimes say, "the Achilles heel of dependent type theory is its dependent types". In many situations, mathematical objects that are taken to be equal in informal mathematics are not automatically equal in Lean, and their equality must be established explicitly; in some cases, equalities can only be expressed via isomorphisms. This has made the manipulation of graded structures in our formalisation tricky.

Another source of difficulty in our formalisation is the distinction between *types* and *terms*. In Lean, types are organised into a hierarchy of universes. A universe level is introduced by a declaration like `universe u`, and the corresponding universe is written as `Type u`. Here, `u` denotes an abstract universe level used to control the stratification of types. When the specific universe level is not essential for a given construction, we may simply write `Type*`.

Types in Lean reside in universes. For example, to declare a group, one may write `G : Type*` and then provide an instance[3] `Group G`. In this case, `G` is a type. Following the existing infrastructure in Mathlib4 [9] (the online mathematics library of Lean), the expression `Subgroup G` denotes the type containing all subgroups of `G`. Many other mathematical concepts are represented as types in Mathlib4, such as polynomial rings. Terms inhabit types. For instance, a declaration `g : G` introduces an element of `G`; here `g` is a term. Likewise, a declaration `S : Subgroup G` introduces a term `S`, corresponding to a subgroup of `G`. Due to the distinction between terms and types, some argument that appears straightforward on paper has become significantly more subtle in our formalisation. We will discuss this issue in detail in this paper.

Furthermore, beyond achieving correctness, attention must also be paid to the conciseness and efficiency of the formalisation. A successful formalisation is not only expected to be correct, but also to be concise and computationally efficient. These considerations play an important role in our development.

1.4 Related Works

Despite the importance of Hilbert polynomials in mathematics, their formalisation in interactive theorem provers remains relatively limited. To the best of our knowledge, there is currently no published formalisation of Hilbert polynomials of algebraic-geometric objects such as projective varieties or coherent sheaves. Existing developments primarily address more basic algebraic settings.

Alexander Maletzky [8], in his Isabelle development, presents a formalisation of a particular form of Hilbert polynomials (see [8, pp.11]). However, in his work, Hilbert polynomials appear primarily as a technical tool for formalising Dubé's degree bounds, rather than as a general framework applicable to Hilbert polynomials of structures in algebraic geometry. More precisely, Maletzky formalises the Hilbert polynomial of a *homogeneous subset* I of a multivariate polynomial ring $F[x_1, \ldots, x_d]$, under the additional assumption that I admits an *exact k-standard homogeneous cone decomposition* (see [8, pp.11–12]). We do not reproduce the definitions of these notions here, as they are not directly related to the main content of our paper; interested readers may consult Maletzky's original work for details.

By contrast, in algebraic geometry, one usually considers a homogeneous ideal I of a polynomial ring $F[x_1, \ldots, x_d]$ and studies the Hilbert polynomial of the quotient $F[x_1, \ldots, x_d]/I$. Such Hilbert polynomials arise from the fact that $F[x_1, \ldots, x_d]/I$ is a graded module over the graded ring $F[x_1, \ldots, x_d]$. However, this graded-module perspective is not covered by Maletzky's formalisation. Our development formalises Hilbert polynomials of graded modules, providing foundational components towards future formalisations of Hilbert polynomials of structures in algebraic geometry.

[3] In Lean, instances declare that some given types carry structures specified by certain typeclasses (for example, groups and rings). Once declared, instances are automatically registered in Lean's typeclass inference mechanism, enabling implicit use of the associated structural data in later proofs.

Another related formalisation is due to Eberl [5], who studied rational generating functions and linear recurrences in Isabelle. Although the primary goal of that work is not the formalisation of Hilbert polynomials, it contains results related to one of the technical ingredients used in our development. In particular, in Sections 2.5 and 2.6, Eberl studies rational expressions of the form $p(x)/((1-c_1x)^{d_1}\dots(1-c_nx)^{d_n})$, whose power-series coefficients are eventually represented by explicit expressions. This framework includes the special case $p(x)/(1-x)^d$, which arises in our formalisation of Hilbert polynomials.

However, Eberl's work and ours adopt different formalisation strategies. In the special case of expressions of the form $p(x)/(1-x)^d$, which is covered by both developments, the eventual behaviour of the power-series coefficients is derived in different ways. Eberl's approach decomposes $p(x)/(1-x)^d$ as a sum of rational expressions with constant numerators, whereas our development expands $p(x)$ directly as $\sum_{i=0}^{N} r_i x^i$ and studies the shifted contributions of the terms $r_i x^i/(1-x)^d$. Both approaches provide reasonable formalisation strategies and illustrate different ways of handling generating-function arguments in proof assistants.

In our development, we rely heavily on the existing infrastructure in Mathlib4, so that it is unnecessary to build everything from first principles. A good example is our treatment of graded structures. Eric Wieser and Jujian Zhang were the first to develop infrastructure for expressing graded structures in Lean; their work has been merged into Mathlib4, and its original form is described in their paper [11].

In that paper, they first introduced a way to represent graded structures as *externally graded* structures. For instance, an externally graded ring is expressed as an indexed family of different types, with each graded piece being an additive commutative monoid (see [11, pp.4–10]). As noted in their paper, it was suggested that infrastructure should also be developed for *internally graded* structures, although such constructions are generally more complicated than the external approach. For example, an internally graded ring is expressed as an indexed family of additive submonoids of a given ring. In this setting, each graded piece appears as a term rather than a type (see [11, pp.10–15]). Our development is an example of why internally graded structures are needed, and we will explain this in more detail.

2 The Mathematical Content

This section briefly presents the mathematical content that we have formalised. The definitions and results introduced here are taken from Atiyah and Macdonald's *Introduction to Commutative Algebra* [2]. The statement of the Hilbert–Serre Theorem relies on these assumptions: $A = \oplus_{n=0}^{\infty} A_n$ is a commutative Noetherian graded ring, $a_1, \dots, a_s$ are nonzero homogeneous elements of A of degrees $d_1, \dots, d_s > 0$ such that $A = A_0[a_1, \dots, a_s]$, and $M = \oplus_{n=0}^{\infty} M_n$ is a finitely generated graded module over A.

Lemma 1. *A_0 is a Noetherian ring.*

Proof. See [2, pp.106].

Lemma 2. *For any $n \in \{0, 1, \ldots\}$, M_n is a finitely generated module over A_0.*

Proof. See [2, pp.116].

Definition 1. *Let R be a ring, C be a class of modules over R and $(G, +)$ be an abelian group. An **additive function** on C with values in G is a map $\mu : C \to G$ such that whenever there is a short exact sequence $0 \to M_1 \to M_2 \to M_3 \to 0$ with $M_1, M_2, M_3 \in C$, we have that $\mu(M_1) + \mu(M_3) = \mu(M_2)$.*

Definition 2. *Let C be the class of all finitely generated modules over A_0 and suppose μ is an additive function on C with values in $\mathbb{Z}$. Then the **Poincaré series** of M in terms of μ is the formal power series*

$$P(M, \mu, x) = \sum_{n=0}^{\infty} \mu(M_n) x^n. \tag{1}$$

Theorem 1 (Hilbert–Serre). *Let μ be an additive function on the class of all finitely generated A_0-modules with values in $\mathbb{Z}$. Then $P(M, \mu, x)$ can be written as*

$$\frac{p(x)}{\Pi_{i=1}^{s}(1 - x^{d_i})} \tag{2}$$

for some polynomial p with integer coefficients [2, pp.117].

We will not present the whole proof for the Hilbert–Serre Theorem in this paper, as all its content can be found in Atiyah and Macdonald's *Introduction to Commutative Algebra* (see [2, pp.117]). However, it is worth mentioning that the proof is by induction on s. In the inductive step, we assume that Theorem 1 holds for s and try to prove that it also holds for $s+1$. Note that for any $n \in \mathbb{N}$, the map g_n defined by $m \mapsto a_{s+1}m$ is an A_0-module homomorphism from M_n to $M_{n+d_{s+1}}$, where d_{s+1} is the degree of a_{s+1}. We let K_n be $\ker(g_n)$ and then construct the graded modules $K := \oplus_{n=0}^{\infty} K_n$ and $L := M/a_{s+1}M$. These parts of the proof required particular care in the formalisation.

Corollary 1. *Assume that $d_1 = \cdots = d_s = 1$. Then there exists a unique polynomial $h \in \mathbb{Q}[x]$ such that for any $n \in \mathbb{N}$ that is sufficiently large, $\mu(M_n) = h(n)$. This h is called the **Hilbert polynomial** of M with respect to μ. Moreover, if $h \neq 0$, then the degree of h is $s - m - 1$, where m is the multiplicity of 1 in the polynomial $p(x)$ mentioned in Theorem 1 [2, pp.117–118].*

We provide a sketch proof of Corollary 1 here. For any polynomial $u(x) = \Sigma_{i=0}^{N} r_i x^i \in \mathbb{Q}[x]$ and positive integer d, we define

$$h_{u,d}(x) := \sum_{i=0}^{N} r_i \frac{(x - i + 1)(x - i + 2) \cdots (x - i + d - 1)}{(d - 1)!}. \tag{3}$$

Then by considering the power series expansion of each fraction $r_i x^i/(1-x)^d$, we see that for any $n > N$, $h_{u,d}(n)$ equals the coefficient of x^n in the power series expansion of $u(x)/(1-x)^d$. Moreover, we can check that for any $u \in \mathbb{Q}[x]$ and $d \in \mathbb{N}_{>0}$, if $u(x)$ is not a multiple of $1-x$, then $h_{u,d}$ has degree $d-1$. Now since we have assumed that $d_1 = \cdots = d_s = 1$, the Poincaré series of M with respect to μ equals $p(x)/(1-x)^s$ by the Hilbert-Serre Theorem. If $(1-x)^s$ divides $p(x)$, then $P(M, \mu, x)$ is a polynomial and so $\mu(M_n)$ is eventually 0. If $(1-x)^s$ does not divide $p(x)$, then $p(x) = (1-x)^m q(x)$ for some polynomial $q(x)$ that is not a multiple of $1-x$; since $p(x)/(1-x)^s = q(x)/(1-x)^{s-m}$, we may take the desired polynomial h to be $h_{q,s-m}$, which has degree $s-m-1$ [2, pp.117].

3 The Formalisation

Now we will present our formalisation of Hilbert polynomials. From this point on, when we say that something happens *in Lean*, we mean that it holds in the Lean language and core system, independently of whether it is present in Mathlib4. In particular, this may also refer to code developed in our own repository that may have not yet been merged into Mathlib4. By contrast, when we say that something happens *in Mathlib4*, we mean that the corresponding content is already available in Mathlib4. So far, approximately 700 lines originating from our development have been merged into Mathlib4, while roughly 4400 lines of our repository remain unmerged.

3.1 Expressing (Internally) Graded Structures

According to the statement and proof of the Hilbert–Serre Theorem, representing graded structures using internal gradings is more suitable for our formalisation than using external gradings. In particular, in the statement of Theorem 1 we assume the existence of an additive function μ on the class of all finitely generated modules over A_0. In Lean, elements of this class are naturally represented as terms. According to the definition of Poincaré series, we consider the values of μ at the graded pieces of the given graded module. Since the inputs of μ are terms rather than types, graded modules need to be expressed using internal gradings.

Moreover, in the inductive step of the proof of the Hilbert–Serre Theorem, we need to show that $A_0[a_1, \ldots, a_s]$ is also a graded ring. This ring is a subring of $A_0[a_1, \ldots, a_{s+1}]$. Mathematically, for each natural number n, the n-th graded piece of $A_0[a_1, \ldots, a_s]$ consists of those elements of $A_0[a_1, \ldots, a_s]$ that also lie in the n-th graded piece of $A_0[a_1, \ldots, a_{s+1}]$. Representing graded pieces as terms therefore simplifies the formalisation. For this reason, we also adopt internal gradings when expressing graded rings in our development.

According to the existing framework in Mathlib4 (originally developed by Eric Wieser and Jujian Zhang as described in their paper [11]), a graded structure need not be graded by natural numbers. The index type for the graded pieces can be any additive monoid. The graded pieces of an internally graded ring or module are given by additive subgroups of the structure.

For a ring or module `M`, we may declare an additive subgroup `S` of `M` either by directly writing `S : AddSubgroup M`, or by introducing a type σ `: Type*` together with the instances `SetLike` σ `M` and `AddSubgroupClass` σ `M`, and then writing `S` : σ. `SetLike` σ `M` expresses that objects of the type σ can be coerced to subsets of `M`, while `AddSubgroupClass` σ `M` ensures that each such object, viewed as a subset of `M`, carries the structure of an additive subgroup. Both `AddSubgroup` and `AddSubgroupClass` will be used in later parts of the paper. We do not discuss here the distinction between using `AddSubgroup` and `AddSubgroupClass`, nor the respective advantages of these approaches, as these are not essential for the main development of this paper. For further details, we refer the reader to Chapter 6 of Anne Baanen's paper [3].

Here we briefly explain how (internally) graded structures are represented in Mathlib4. For example, suppose that we want to declare a graded ring `A`. We first declare that `A` is a ring, and then introduce a function $\mathcal{A}$ from the index type to the type of additive subgroups of `A`. For any element `i` in the index type, $\mathcal{A}$ `i` denotes the `i`-th graded piece. We then declare that `A` is isomorphic to the direct sum of these graded pieces. Similar ideas are used for declaring other graded structures, such as graded modules. Following the conventions in Mathlib4, we normally use `A`, `M`, and so on to denote the graded objects themselves, while $\mathcal{A}$, $\mathcal{M}$, and so on denote the functions specifying their graded pieces.

3.2 Proving Lemma 1

We have already merged a generalised version of Lemma 1 (which is called `GradedRing.GradeZero.isNoetherianRing`) into Mathlib4: for any Noetherian ring `A` (not necessarily commutative) that is graded by a canonically ordered[4] additive commutative monoid, the `0`-th graded piece $\mathcal{A}$ `0` is a Noetherian ring. The proof seems quite easy to formalise. For any ring homomorphism `f : R →+* S`[5], if we have that `f` is surjective and `R` is Noetherian, then we may use the theorem `isNoetherianRing_of_surjective` (which already exists in Mathlib4) to conclude that `S` is also Noetherian. At the time of our formalisation, Mathlib4 contained an instance `DirectSum.GradeZero.ring`, which stated that for any ring graded by an additive monoid, its `0`-th graded piece is a ring. Moreover, the canonical projection function from `A` to $\mathcal{A}$ `0` is a surjective ring homomorphism.

When it came to the actual formalisation, things were not so easy. Lean could not infer the instance that $\mathcal{A}$ `0` is a ring when we carried out the proof. The root of the problem is the difference between externally and internally graded rings. The full declaration of `DirectSum.GradeZero.ring` was as follows:

```
instance DirectSum.GradeZero.ring {ι : Type*} [DecidableEq ι]
    (𝒜 : ι → Type*) [(i : ι) → AddCommGroup (𝒜 i)] [AddMonoid ι]
    [GRing 𝒜] : Ring (𝒜 0)
```

[4] In Mathlib4, "canonically ordered" means that for any two elements `i` and `j`, `i ≤ j` if and only if there is some `k` such that `j` equals the sum of `i` and `k`.

[5] In Mathlib4, given two rings `R` and `S`, the notation `R →+* S` denotes the type of ring homomorphisms from `R` to `S`.

Note that if we use an abstract index set ι, we must explicitly assume `DecidableEq` ι, namely that ι has decidable equality. When the index set is a concrete type such as $\mathbb{N}$, no additional declaration is needed, since Mathlib4 already has an instance of `DecidableEq` $\mathbb{N}$. Note also that for any `i : ` ι, $\mathcal{A}$ `i` is a type. The expression `GRing` $\mathcal{A}$ means that $\oplus_{i\in\iota}$`(`$\mathcal{A}$ `i)` is a graded ring. Hence, we see that `DirectSum.GradeZero.ring` was designed for externally graded rings.

However, when we stated the assumptions in Lemma 1, `A` was declared as an internally graded ring. Specifically, we declared a type σ such that `SetLike` σ `A` and `AddSubgroupClass` σ `A`, and defined $\mathcal{A}$ as a function from ι to σ, so $\mathcal{A}$ `i` is a term for any `i : ` ι.

Consider the following goal:

```
example {A ι σ : Type*} [Ring A] [SetLike σ A] [AddSubgroupClass σ A]
    [DecidableEq ι] [AddMonoid ι] (𝒜 : ι → σ)
    [DirectSum.GRing (fun i => 𝒜 i)] : Ring (𝒜 0) := ...
```

The predicate `Ring` takes a type as its input, but $\mathcal{A}$ `0` is a term for an internally graded ring `A` $\simeq \oplus_{i\in\iota}$`(`$\mathcal{A}$ `i)`; Lean recognises the goal as `Ring` $\uparrow$`(`$\mathcal{A}$ `0)`, where $\uparrow$ `(`$\mathcal{A}$ `0)` means "the term $\mathcal{A}$ `0` coerced into a type". At the time, if we manually typed in the instance `DirectSum.GradeZero.ring (fun i => `$\mathcal{A}$ `i)`, then the goal could be completed. By writing `fun i => `$\mathcal{A}$ `i`, we have turned $\mathcal{A}$ into a function belonging to the type $\iota \to$ `Type*`, because Lean can automatically recognise `fun i => `$\mathcal{A}$ `i` as `fun i =>` $\uparrow$`(`$\mathcal{A}$ `i)`. However, typing in `inferInstance` did not allow Lean to find `DirectSum.GradeZero.ring`; that is, Lean was unable to infer the instance automatically.

The reason was as follows. `DirectSum.GradeZero.ring` stated that "given some function $\mathcal{A}$, $\mathcal{A}$ `0` is a ring". However, if we want to prove that $\uparrow$`(`$\mathcal{A}$ `0)` is a ring for some internal grading, then our goal is essentially to show that an expression of the form `C (`$\mathcal{A}$ `0)` is a ring, where `C` is a function. Hence, Lean could not infer `DirectSum.GradeZero.ring`. Note that although the distinction between terms and types matters a lot in Lean, we hardly ever care about it in proofs written on paper. In the end, we constructed a new instance `SetLike.GradeZero.instRing` (which is already in Mathlib4), which says that if `A` is an internally graded ring, then $\uparrow$`(`$\mathcal{A}$ `0)` is a ring. By `SetLike.GradeZero.instRing`, we managed to formalise Lemma 1.

In fact, instances like `DirectSum.GradeZero.ring` have caused difficulties not only in our project but also elsewhere. In Mathlib4, instances are assigned different priorities; Lean's typeclass search first attempts to use higher-priority instances when trying to solve a goal. In some situations, an instance with higher priority may appear applicable but is not the intended one, leading to inefficient search. This can significantly increase compilation time and even make the code unable to compile.

For example, consider the following goal, where `Mul` is a typeclass in Mathlib4 saying that a type has a multiplication operation, and $\bot$ means the zero ideal of `R[X]`:

```
example {R : Type*} [CommRing R] : Mul (R[X] ⧸ (⊥ : Ideal R[X])) := ...
```

The goal seems straightforward to complete, since a quotient ring inherits a ring structure and so a multiplication. However, a discussion on the Lean Zulip reported a case where this goal could not be solved by typeclass inference.[6] The reason was as follows: although there is an instance `Ideal.Quotient.instRingQuotient` in Mathlib4 saying that quotients of a commutative ring are rings, it had lower priority than instances like `DirectSum.GradeZero.ring`. Lean treated these as potentially applicable instances, since the type `Ideal R[X]` carries an additive monoid structure.

In particular, for `I J : Ideal R[X]`, their sum `I + J` is defined as the smallest ideal containing both `I` and `J`; the zero element of `Ideal R[X]` is just the zero ideal. Recall that the index set for the graded pieces of a graded ring can be any additive monoid. Hence, Lean regarded `R[X] ⧸ (⊥ : Ideal R[X])` as the `0`-th graded piece of $\oplus_{\mathtt{I}\in\mathtt{Ideal\,R[X]}}$`(R[X] ⧸ I)` and tried to use instances like `DirectSum.GradeZero.ring`, but Lean could not give a ring structure on $\oplus_{\mathtt{I}\in\mathtt{Ideal\,R[X]}}$`(R[X] ⧸ I)`. At the time, this issue was addressed by adjusting instance priorities in Mathlib4.

3.3 Stating the Hilbert–Serre Theorem

It is well-known that for any commutative Noetherian ring `R`, the class of finitely generated `R`-modules forms an abelian category. Hence, we decided to formalise the definition of additive functions in this way: given an abelian category $\mathcal{C}$ and an additive commutative group `G`, a function μ from the class of all objects of $\mathcal{C}$ to `G` is **additive** if for any short exact sequence `0 ⟶ A ⟶ B ⟶ C ⟶ 0` with elements in $\mathcal{C}$ (that is, a short complex in $\mathcal{C}$), we have μ`(A) + `μ`(C) = `μ`(B)`. Also, we denote the type of additive functions from $\mathcal{C}$ to `G` as $\mathcal{C}$ `⟹+ G` (see [7, `HilbertPolynomial/HilbertSerre/AdditiveFunction.lean`]):

```
variable (𝒞 : Type u) [Category.{v} 𝒞] [Abelian 𝒞]
variable (G : Type w) [AddCommGroup G]

structure AdditiveFunction :=
  toFun : 𝒞 → G
  additive (s : ShortComplex 𝒞) (hs : s.ShortExact) :
    toFun s.X₁ + toFun s.X₃ = toFun s.X₂

notation 𝒞 "⟹+" G => AdditiveFunction 𝒞 G
```

In Lean, a `structure` defines a record type consisting of data and associated fields, often used to represent mathematical objects such as groups and rings.

The statement of the Hilbert–Serre Theorem is based on the following variables:

```
universe u
variable {A M : Type u}
```

[6] Details of this issue can be found in `https://leanprover.zulipchat.com/#narrow/channel/287929-mathlib4/topic/Typeclass.20Synthesis.20Regression/near/564417997`.

```
variable [CommRing A] [IsNoetherianRing A]
variable [AddCommGroup M] [Module A M] [Module.Finite A M]
variable (𝒜 : ℕ → AddSubgroup A) (ℳ : ℕ → AddSubgroup M)
variable [GradedRing 𝒜] [DirectSum.Decomposition ℳ]
variable [SetLike.GradedSMul 𝒜 ℳ]
variable (μ : (FGModuleCat (𝒜 0)) ⟹+ ℤ)
```

`DirectSum.Decomposition` $\mathcal{M}$ tells Lean that the canonical function $(\mathtt{m}_n)_{n\in\mathbb{N}} \mapsto \Sigma_{n=0}^{\infty}\mathtt{m}_n$ from $\oplus_{n=0}^{\infty}(\mathcal{M}\ \mathtt{n})$ to `M` is bijective. The expression `GradedRing` $\mathcal{A}$ means that `DirectSum.Decomposition` $\mathcal{A}$ is true, the degree of `1` is zero, and the product of two elements of degrees `i` and `j` has degree `i + j`. `SetLike.GradedSMul` $\mathcal{A}\ \mathcal{M}$ tells Lean that the scalar multiplication `A` × `M` → `M` satisfies the axioms of an action of a graded ring on a graded module.

One assumption in the Hilbert–Serre Theorem is that the graded ring has a finite collection of nonzero positive-degree homogeneous elements that generates the ring as an algebra over the `0`-th graded piece. In our project, we call such a collection as a "generating set over the base ring" (the code can be found in our repository [7, `HilbertPolynomial/HilbertSerre/Theorem.lean`]):

```
structure generatingSetOverBaseRing where
  toFinset : Finset A
  deg : ∀ {a : A}, a ∈ toFinset → ℕ
  mem_deg : ∀ {a : A} (h : a ∈ toFinset), a ∈ 𝒜 (deg h)
  deg_pos : ∀ {a : A} (h : a ∈ toFinset), 0 < deg h
  ne_zero' : ∀ {a : A}, a ∈ toFinset → a ≠ 0
  span_eq : Algebra.adjoin (𝒜 0) toFinset = (⊤ : Subalgebra (𝒜 0) A)
```

For any `S : generatingSetOverBaseRing` $\mathcal{A}$, we define `S.poles` as the polynomial $\Pi_{i\in S}(\mathtt{1 - X}^{d_i})$, where for each element `i` in `S`, $\mathtt{d}_i$ means the degree of `i`. Note that if we write the Poincaré series as a rational expression, then this polynomial is just the denominator. We have formalised this as the following (see [7, `HilbertPolynomial/HilbertSerre/Theorem.lean`]):

```
variable (S : generatingSetOverBaseRing 𝒜)

noncomputable def generatingSetOverBaseRing.poles : ℤ⟦X⟧ˣ where
  val := Π i ∈ S.toFinset.attach, (1 - PowerSeries.X ^ S.deg i.2)
  inv := PowerSeries.invOfUnit
    (Π i ∈ S.toFinset.attach, (1 - PowerSeries.X ^ S.deg i.2)) 1
  val_inv := ...
  inv_val := ...
```

In Lean, a definition is declared `noncomputable` when it may rely on classical axioms and hence lacks guaranteed computational content.

The Poincaré series is defined as the power series in $\mathbb{Z}[\![\mathtt{X}]\!]$ such that for any natural number `n`, the coefficient of $\mathtt{X}^n$ is $\mu(\mathcal{M}\ \mathtt{n})$, as introduced before:

```
def AdditiveFunction.poincareSeries : ℤ⟦X⟧ :=
  PowerSeries.mk fun n ↦ μ (FGModuleCat.of (𝒜 0) (ℳ n))
```

Note that although $\mathcal{M}$ `n` is a finitely generated module over $\mathcal{A}$ `0`, it belongs to `AddSubgroup M` rather than `FGModuleCat (`$\mathcal{A}$ `0)`. The expression `FGModuleCat.of (`$\mathcal{A}$ `0) (`$\mathcal{M}$ `n)` denotes the element in `FGModuleCat (`$\mathcal{A}$ `0)` that corresponds to $\mathcal{M}$ `n`. We formally state the Hilbert–Serre Theorem as follows:

```
theorem hilbert_serre : ∃ (p : Polynomial ℤ),
    μ.poincareSeries 𝒜 ℳ = p · S.poles⁻¹
```

Both the declarations `AdditiveFunction.poincareSeries` and `hilbert_serre` are in the file `HilbertPolynomial/HilbertSerre/Theorem.lean` of our repository. We now describe the technical aspects of our formalisation of the proof of the Hilbert–Serre Theorem.

3.4 Subgrading and Quotient Grading

The Lean code for the constructions discussed in this subsection can be found in the file `HilbertPolynomial/Module/Graded/Grading.lean` in our repository. In the Hilbert–Serre Theorem's proof given by Atiyah and Macdonald [2, pp.117], both the module K and the subring $A_0[a_1, \ldots, a_s] \subseteq A_0[a_1, \ldots, a_{s+1}]$ mentioned in the induction step are examples of a more general construction — if `A'` is a homogeneous subobject[7] of a graded structure `A`, then `A'` normally inherits a grading from `A`. In this project, we have formalised graded subobjects. We now describe the main idea of our formalisation.

Assume that `A` is a graded ring that is isomorphic to the direct sum $\oplus_{i\in\iota}$`(`$\mathcal{A}$ `i)`, where ι is an arbitrary additive monoid. Let `A'` be a homogeneous subring of `A`. In order to convince Lean that `A'` is also graded by ι, we need to come up with a family of additive subgroups of `A'` indexed by ι; these will be the graded pieces of `A'`. Mathematically, for any `i : `ι, the `i`-th graded piece of `A'` should be `(`$\mathcal{A}$ `i) ∩ A'`. In practice, we have represented `(`$\mathcal{A}$ `i) ∩ A'` as the preimage of $\mathcal{A}$ `i` under the canonical inclusion ring homomorphism from `A'` to `A`. This formulation is preferable in practice, as Lean automatically endows such preimages with additive subgroup structures. We have constructed the graded pieces of graded submodules in a similar way.

Then we formalise the concept of graded quotient modules, which is necessary for the L mentioned in the Hilbert–Serre Theorem's proof. Let `M ≃ `$\oplus_{i\in\iota}$`(`$\mathcal{M}$ `i)` be a graded module. Suppose `p` is a homogeneous submodule of `M`. Then mathematically `M/p` is a graded module such that for any `i ∈ `ι`M`, its `i`-th graded piece is the quotient `(`$\mathcal{M}$ `i)/((`$\mathcal{M}$ `i) ∩ p)`. In practice, we have defined the `i`-th graded piece as the image of the canonical map from `(`$\mathcal{M}$ `i)/((`$\mathcal{M}$ `i) ∩ p)` to `M/p`, so that Lean can automatically recognise it as an additive subgroup of `M/p`.

[7] A "subobject" of a mathematical object is an object of the same category that is embedded into the original object. For example, a subobject of a group is a subgroup, a subobject of a module is a submodule, etc. By "homogeneous", we mean that for any `a ∈ A`, all the homogeneous components of `a` also belong to `A`.

3.5 The Inductive Argument

Let `s` be the cardinality of the collection `S` of homogeneous elements that generates `A` as an `(𝒜 0)`-algebra. The base case `s = 0` is easy to formalise. For the inductive step, we declared that `x` is an arbitrary element of `A` that belongs to the generating set `S` over the base ring, and `S'` is a finite subset of `A` such that `insert x S' = S.toFinset`, i.e. `S' ∪ {x} = S`. Then we defined a ring `A'` that is equal to the subring of `A` generated by `(𝒜 0) ∪ S'`. Using the subgrading that we have constructed before, we obtained a function $\mathcal{A}'$ such that `A' ≃` $\oplus_{n=0}^{\infty}$`(𝒜' n)`. By the induction hypothesis, `A'` satisfies the statement of the Hilbert–Serre Theorem, which can help complete the proof.

By our construction, we noticed something tricky when trying to complete the inductive step: the function $\mathcal{A}'$ is different from $\mathcal{A}$, so Lean does not regard $\mathcal{A}'$ `0` and $\mathcal{A}$ `0` as the same thing. The variable μ is an additive function on the class of all finitely generated modules over $\mathcal{A}$ `0`, rather than $\mathcal{A}'$ `0`. We solved this problem by proving that if $\mathcal{C}$ and $\mathcal{D}$ are equivalent abelian categories and μ is an additive function on $\mathcal{C}$, then we have an obvious additive function on $\mathcal{D}$ induced by μ (see [7, `HilbertPolynomial/HilbertSerre/AdditiveFunction.lean`]). $\mathcal{A}$ `0` and $\mathcal{A}'$ `0` are different in Lean because they are not definitionally equal. However, as they are mathematically equal, we can easily prove that they are isomorphic. As a result, the category of all finitely generated modules over $\mathcal{A}$ `0` is equivalent with that of $\mathcal{A}'$ `0`, so that we could carry on with the induction step. The complete formalisation of the Hilbert–Serre Theorem can be found in the file `HilbertPolynomial/HilbertSerre/Theorem.lean` of our repository.

3.6 Coefficient-Representing Polynomials and Hilbert Polynomials

The existence of Hilbert polynomials is based on the fact that, for any polynomial $p \in \mathbb{Q}[x]$ and natural number $d \in \mathbb{N}$, there exists a polynomial $h \in \mathbb{Q}[x]$ whose values eventually coincide with the coefficients of the power series expansion of $p(x)/(1-x)^d$. In the formalisation of this fact, we encountered a design-choice issue. Recall that in Section 2 we introduced, for every polynomial $p \in \mathbb{Q}[x]$ and positive integer d, a polynomial $h_{p,d}$, which has degree $d-1$ whenever p is not divisible by $1-x$. In the formal development, if $p(x)$ is divisible by $(1-x)^d$, one may simply define the coefficient-representing polynomial to be zero. Otherwise, there are two natural approaches. Suppose that m is the multiplicity of 1 in $p(x)$, and $p(x) = (1-x)^m q(x)$. One may define the coefficient-representing polynomial as $h_{q,d-m}$ or simply as $h_{p,d}$.

In our initial formalisation of coefficient-representing polynomials, we chose $h_{q,d-m}$ rather than $h_{p,d}$ for the case when $p(x)$ is not divisible by $(1-x)^d$. The motivation came from the proof of Corollary 1, as sketched in Section 2. Since $q(x)$ is not divisible by $1-x$, the degree of the coefficient-representing polynomial can be established directly from the properties of $h_{q,d-m}$. By contrast, using $h_{p,d}$ appears to require additional arguments to establish the degree result, making the formalisation less straightforward.

However, using $h_{q,d-m}$ has a drawback. Compared with using $h_{p,d}$, introducing the auxiliary polynomial q leads to a more complicated definition. As a result, formalising properties other than the degree theorem becomes less convenient. For example, additional work is required to prove that the values of the coefficient-representing polynomial eventually coincide with the coefficients of the power series expansion.

Later, we found a way to work directly with $h_{p,d}$ while keeping the formalisation of the degree result straightforward. We have also generalised the construction: by the same proof as before, we can show that for any field F, polynomial $p \in F[x]$ and natural number d, there exists a polynomial $h \in F[x]$ such that as long as F has characteristic zero and $n \in \mathbb{N}$ is sufficiently large, $h(n)$ equals the coefficient of x^n in the power series expansion of $p(x)/(1-x)^d$. This h is defined as follows: if $d = 0$, then $h := 0$; otherwise, $h := h_{p,d}$. Note that the formula defining $h_{p,d}$ only involves addition, subtraction, multiplication, and division by $(d-1)!$. Moreover, in Mathlib4, division is defined for any pair of elements of a field: division by zero is defined to be zero rather than left undefined. Consequently, the formula for $h_{p,d}$ introduced in Section 2 can be expressed over an arbitrary field in Lean and gives rise to a well-defined polynomial.

In Mathlib4, the variable of a univariate polynomial is denoted by `X`. In our development, we have created a declaration `Polynomial.hilbertPoly` such that for any field `F`, polynomial `p : F[X]`, and natural number `d`, `Polynomial.hilbertPoly p d` is the coefficient-representing polynomial for `p/(1 - X)^d`. The full API for `Polynomial.hilbertPoly` has been merged into Mathlib4.

Since our new definition does not involve an auxiliary polynomial `q` that is not divisible by `1 - X`, it is much simpler to prove the key property of `hilbertPoly p d`, that is, if `F` has characteristic zero and `n : ℕ` is sufficiently large, then the coefficient of `X^n` in the power series of `p/(1 - X)^d` equals the value of `hilbertPoly p d` at `n`. Note that the assumption that `F` has characteristic `0` is needed for the coefficient-representing property. If `(d - 1) !` is divisible by the characteristic of `F`, then `(d - 1) !` is zero in `F`. As a result, `hilbertPoly p d` will always be `0`, so it may not satisfy the coefficient-representing property. The uniqueness of `hilbertPoly p d` is easy to formalise, because any polynomial satisfying the key property of `hilbertPoly p d` agrees with `hilbertPoly p d` on an infinite subset of $\mathbb{N}$.

However, with the new definition of the coefficient-representing polynomial, we needed to rethink how to compute its degree. We did it as follows. We first proved that for any `p1 : F[X]` and `d : ℕ`, `hilbertPoly (p1 * (1 - X)) (d + 1) = hilbertPoly p1 d`, which is easy because `p1 * (1 - X)/(1 - X)^(d+1) = p1/(1 - X)^d` and so `hilbertPoly (p1 * (1 - X)) (d + 1)` satisfies the key property of `hilbertPoly p1 d`; hence we may use the uniqueness of `hilbertPoly p1 d` to complete the argument. Then by induction, we showed that for any `p1 : F[X]` and `d e : ℕ`, `hilbertPoly (p1 * (1 - X) ^ e) (d + e) = hilbertPoly p d`. Now given any `p : F[X]` and `d : ℕ` with `m < d`, where `m` is the multiplicity of `1` in `p`, we let `q : F[X]` be the factor of `p` such that `p = q * (1 - X)^m`. Then by the result we have

already shown, `hilbertPoly p d` can be changed into `hilbertPoly q (d - m)`, so the degree of `hilbertPoly p d` has become easier to formalise.

Now we return to the settings in the Hilbert–Serre Theorem. In our formalisation, we let `numeratorPolynomial` $\mathcal{M}$ μ `S` be the numerator when the Poincaré series is written as a rational expression, and then define the Hilbert polynomial of `M` (in terms of μ) with the help of the declaration `hilbertPoly` (see [7, `HilbertPolynomial/HilbertPolynomial.lean`]):

```
namespace HilbertSerre

noncomputable def numeratorPolynomial : Polynomial ℤ :=
  (hilbert_serre 𝒜 ℳ μ S).choose

noncomputable def hilbertPolynomial : Polynomial ℚ :=
  (Polynomial.map (Int.castRingHom ℚ)
    (numeratorPolynomial ℳ μ S)).hilbertPoly
  S.toFinset.card
```

With this definition, we have formalised the key property of Hilbert polynomials, that is, if all elements in `S` have degree `1`, then for any natural number `n` that is sufficiently large, μ`(`$\mathcal{M}$ `n)` equals the value of the Hilbert polynomial at `n`. We have also formalised the uniqueness and the degree of the Hilbert polynomial (see [7, `HilbertPolynomial/HilbertPolynomial.lean`]).

4 Conclusion and Future Work

Using the Hilbert–Serre Theorem, we have formalised the Hilbert polynomial of a finitely generated graded module $M = \oplus_{n=0}^{\infty} M_n$ over a commutative Noetherian graded ring $A = \oplus_{n=0}^{\infty} A_n$, where $A = A_0[a_1, \ldots, a_s]$ for some homogeneous elements $a_1, \ldots, a_s \in A$ of degree 1.

Due to the formulation and proof of the Hilbert–Serre Theorem, we found it necessary to use internal gradings rather than external gradings to represent graded structures. Consequently, the distinction between terms and types made some preliminary content tricky to formalise, although the proof looks quite simple on paper. We also took extra effort to complete the inductive step of the Hilbert–Serre Theorem's proof, since it frequently happens that objects that are regarded as equal in informal mathematics can only be identified up to isomorphism in Lean. Our experience emphasises the difference between proofs written on paper and proofs formalised in Lean.

Moreover, the old and new methods to represent coefficient-representing polynomials and prove relevant results have demonstrated that formalisation in Lean is not only something mathematical, but also something artistic — the successful formalisation of some content is not the end; there may always exist ways to make definitions more beautiful and proofs more efficient.

As we have formalised the basic form of Hilbert polynomials, our project has provided a foundation for the future formalisation of Hilbert polynomials of other algebraic structures, such as projective varieties [6, pp.52]. Consequently,

it has become more hopeful that further concepts and results can be formalised using Lean, such as Hilbert schemes [1, pp.6–7] and Bézout's Theorem [6, pp.54].

Acknowledgments. We thank our supervisor, Kevin Buzzard, for his guidance and constructive feedback. His review of the Lean formalisation and suggestions on language and presentation have significantly improved this work. We also thank Heather Macbeth for helpful comments and suggestions on earlier versions of this work. The second author was supported by the Schrödinger Scholarship Scheme at Imperial College London during the preparation of this work.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Arbarello, E., Cornalba, M., Griffiths, P.A.: The Hilbert scheme. In: Geometry of Algebraic Curves, Grundlehren der mathematischen Wissenschaften, vol. 268. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-540-69392-5_1
2. Atiyah, M.F., Macdonald, I.G.: Introduction to commutative algebra. Addison-Wesley Publishing Company, Reading, MA (1969).
3. Baanen, A.: Use and abuse of instance parameters in the Lean mathematical library. In: Andronick, J., de Moura, L. (eds.) Interactive Theorem Proving (ITP 2022), LIPIcs, vol. 237, pp. 4:1–4:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.ITP.2022.4
4. de Moura, L., Ullrich, S.: The Lean 4 theorem prover and programming language. In: Platzer, A., Sutcliffe, G. (eds.) Automated Deduction – CADE 28, LNCS, vol. 12699, pp. 625–635. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79876-5_37
5. Eberl, M.: Verified solving and asymptotics of linear recurrences. In: Mahboubi, A., Myreen, M.O. (eds.) Certified Programs and Proofs (CPP 2019), pp. 27–37. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3293880.3294090
6. Hartshorne, R.: Algebraic geometry. Graduate Texts in Mathematics, vol. 52. Springer, New York (1977). https://doi.org/10.1007/978-1-4757-3849-0
7. Li, F., Zhang, J.: HilbertPolynomial v1. Zenodo (2026). https://doi.org/10.5281/zenodo.20752436
8. Maletzky, A.: Formalization of Dubé's degree bounds for Gröbner bases in Isabelle/HOL. In: Kaliszyk, C., Brady, E., Kohlhase, A., Sacerdoti Coen, C. (eds.) Intelligent Computer Mathematics (CICM 2019), LNAI, vol. 11617, pp. 155–170. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23250-4_11
9. Mathlib Contributors: Welcome to the documentation page. https://leanprover-community.github.io/mathlib4_docs/index.html, last accessed 2026/02/05
10. Matsumura, H.: Commutative ring theory. Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge (1987). https://doi.org/10.1017/CBO9781139171762

11. Wieser, E., Zhang, J.: Graded rings in Lean's dependent type theory. In: Buzzard, K., Kutsia, T. (eds.) Intelligent Computer Mathematics (CICM 2022), LNAI, vol. 13467, pp. 122–137. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-16681-5_8