Formalising modular forms, Eisenstein series and the statement of the modularity conjecture

Christopher Birkbeck
University of East Anglia, Norwich, UK

Abstract

Modular forms are special analytic functions of number theoretic interest due to their close links with elliptic curves, Galois representations and L-functions. These functions play a key role in number theory, being the subject of the modularity conjecture and key in the proof of Fermat's Last Theorem. We describe the formalisation of modular forms and Eisenstein series. Furthermore, we formalise the proof that Eisenstein series are indeed modular forms, describing the main results that are required. Finally we give a formalised statement of the modularity conjecture, linking elliptic curves and modular forms.

2012 ACM Subject Classification Mathematics of computing → Mathematical software; Security and privacy → Logic and verification

Keywords and phrases Modular forms, Eisenstein series, modularity, number theory, formal maths, Lean, mathlib

Digital Object Identifier 10.4230/LIPIcs...

Supplementary Material The code described is available in a separate reposition to mathlib
Software: `https://github.com/CBirkbeck/ModularForms`

Acknowledgements This wouldn't have been possible without lots of help from people on Lean's Zulip chat, such as Alex Best, Ricardo Brasca, Kevin Buzzard, Yury Kudryashov, Alex Kontorovich, Heather Macbeth, Marc Masdeu and many others.

1 Introduction

Historically, one of the main objects of study for mathematicians have been Diophantine equations. Such problems, while easy to state, can require a great deal of mathematical machinery to solve. Perhaps the most famous example of this is that of Fermat's Last Theorem, which says that for $n \in \mathbb{Z}_{>2}$, the equation $x^n + y^n = z^n$ has no integer solutions with $xyz \neq 0$. This simple to state problem took mathematicians over 300 years to solve, and its proof was completed by Andrew Wiles. The methods developed in the proof are still widely used and studied in current research in number theory and geometry.

A crucial step in Wiles' proof is that of creating a link between algebra, geometry and analysis. Specifically, one needs to use a special case of the modularity conjecture (also known as the Shimura–Taniyama–Weil conjecture)[1]. This theorem creates a bridge between special kinds of analytic functions, known as modular forms, and geometric objects, known as elliptic curves. On the geometric side, elliptic curves (over $\mathbb{Q}$), are described by equations of the form $y^2 = x^3 + ax + b$ (for a, b rational numbers). On the analytic side, modular forms are complex holomorphic functions transforming nicely under symmetries of the complex upper half plane. These functions have Fourier expansions whose coefficients encode interesting arithmetic information.

[1] Due to work of Breuil, Conrad, Diamond, Taylor, Wiles and many others, this conjectures is now a theorem, see [4, 2].

The modularity conjecture links, in a precise way, the number of solutions an elliptic curve has over a finite field to Fourier coefficients of certain modular forms, and it is this link which is the key step in the proof of Fermat's Last Theorem. We do will not attempt to summarise the proof here but more details can be found in many places, such as [5].

Modular forms (and their generalisations) also have many other interesting connections, such as computing Ramanujan's tau function, Jacobi's four-square theorem, and links with the Leech lattice. Our goal here is to formalise basic aspects of the theory of modular forms, Eisenstein series and formalise the statement of the modularity conjecture. We do this in the Lean theorem prover [6]. This is a dependently typed proof assistant based on the calculus of inductive constructions. We build upon the Lean 3 mathematics library, known as mathlib [10]. This library of formalised definitions/theorems contains many of the basic ingredients required to formalise this conjecture.

One of the goals of mathlib is to create a unified library of computer verified mathematics, with the aim of changing how mathematics is communicated, taught and researched. Recently there have been many examples of high level mathematics being successfully formalised in Lean, for example work by Buzzard–Commelin–Massot formalising the definition of perfectoid spaces [3] or more recently the "Liquid Tensor Experiment" [1] that has formalised recent work of Clausen–Scholze.

In order to begin formalising current number theory research, one needs to first formalise the key definitions and basic properties. Modular forms are interesting definitions to formalise for several reasons:

1. Fermat's Last Theorem is one of the biggest results in number theory and mathematics in general and therefore an obvious target for formalisation. The complete proof will require the theory of modular forms and a proof of (a special case of) the modularity conjecture. The work here therefore is a small stepping stone this direction and opens the possibility for more of the theory to be formalised.
2. These definitions have algebraic and analytic components and therefore serve as a good test of how different parts of mathlib interact and highlights what areas need to be developed. For example, in the process of formalising our constructions, we also had to formalise GL_n, results about differentiable functions between manifolds (such as their sum and product are again differentiable), proving uniform limits of differentiable functions are again differentiable, etc. These definitions/results were first developed here and most of them are in the process of being added the mathlib.
3. In order to develop a good API for modular forms several formalisation challenges need to be overcome. For example, developing a theory that we can efficiently build upon the future by using classes such as `fun_like` and `modular_form_class` which allows for better integration with the existing library and as well allowing future definitions (such as newforms) to automatically inherit previous results. Similarly there the Type-theoretic issues that arise when defining the graded commutative ring of modular forms that needed to be addressed.

We should also note that some basic definitions of modular forms had already been formalised in Lean, here: `https://github.com/semorrison/kbb`. This project by Reid Barton, Johan Commelin, Mario Carneiro, Johannes Hölzl, Kenny Lau, Sean Leather, Patrick Massot and Scott Morrison contained (amongst other mathematical definitions) a basic definition of modular forms, but no examples and they did not state the modularity conjecture. In particular, our work in defining modular forms for general congruence subgroups, Eisenstein series and proving that they are modular forms is new as well as our statement of the modularity conjecture. Nonetheless, while our definitions are mainly independent of the ones found here, this work was extremely useful in guiding us.

Lastly, there is forthcoming work of Manuel Eberl, Larry Paulson and Anthony Bordg in Isabelle/HOL, containing the definition of modular forms, Eisenstein series and more, although at the time of writing the code is not yet public. But as far as we know, definitions for general congruence subgroups and the statement of the modularity conjecture have yet to be formalised in other proof assistants such as Isabelle/HOL, Coq, etc.

Some of the resulting code from this project has already been integrated into mathlib (such as the definitions of modular form and cusp forms) and the rest is currently in the process of being added. We also have a repository containing the source code (that is not yet in mathlib) for this project here : `https://github.com/CBirkbeck/ModularForms`. In what follows we will omit the code which formalises the required proofs, but all details can be found in mathlib or the repository above. Other than the modularity conjecture, the rest of the results discussed here have complete "sorry-free" proofs.

2 Modular forms

We begin by introducing the main mathematical definitions we will be formalising together with some of the key results needed to prove some of their basic properties. We assume some familiarity with basic matrix theory, ring theory and complex analysis. The definitions and theorems below are all standard and can be found in many sources, such as [7, 9].

Let $\mathbb{H}$ denote the complex upper half plane, defined as $\{z \in \mathbb{C} \mid 0 < \mathrm{Im}(z)\}$. On this set we can define an action by the group of 2×2 matrices with real entries and positive determinant, denoted $\mathrm{GL}_2(\mathbb{R})^+$. Specifically, given $\gamma = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ in $\mathrm{GL}_2(\mathbb{R})^+$ and $z \in \mathbb{H}$ we define $\gamma \cdot z = \left(\frac{az+b}{cz+d}\right)$. One easily checks that this is indeed a group action. Specifically, if $\gamma_1, \gamma_2 \in \mathrm{GL}_2(\mathbb{R})^+$ then we have $(\gamma_1\gamma_2) \cdot z = \gamma_1 \cdot (\gamma_2 \cdot z)$ and the identity element of $\mathrm{GL}_2(\mathbb{R})^+$ acts trivially. Using this one can define an action on functions $\mathbb{H} \to \mathbb{C}$ as follows: let $k \in \mathbb{Z}$ be an integer and $\gamma = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in \mathrm{GL}_2(\mathbb{R})^+$. Then the weight k action of γ on f is given by

$$(f \mid_k \gamma)(z) := \det(\gamma)^{k-1}(cz+d)^{-k} f(\gamma \cdot z).$$

Again one easily checks that this defines an action on this space of functions, known as the weight k *slash action.*

Modular forms are holomorphic functions $\mathbb{H} \to \mathbb{C}$ that are invariant under this $|_k \gamma$ action, for γ in certain subgroups of $\mathrm{GL}_2(\mathbb{R})^+$. These are subgroups of $\mathrm{SL}_2(\mathbb{Z})$ (the 2×2 matrices with integer entries and determinant 1) called the level of the modular form. The most common of these are known as congruence subgroups and some basic examples are given by taking $N \in \mathbb{Z}_{>0}$ and defining:

$$\Gamma_0(N) := \{\gamma \in \mathrm{SL}_2(\mathbb{Z}) \mid \gamma \equiv \left(\begin{smallmatrix} * & * \\ 0 & * \end{smallmatrix}\right) \pmod N\}$$
$$\Gamma_1(N) := \{\gamma \in \mathrm{SL}_2(\mathbb{Z}) \mid \gamma \equiv \left(\begin{smallmatrix} * & * \\ 0 & 1 \end{smallmatrix}\right) \pmod N\}$$
$$\Gamma(N) := \{\gamma \in \mathrm{SL}_2(\mathbb{Z}) \mid \gamma \equiv \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) \pmod N\}.$$

▶ **Definition 1.** *Let Γ denote a subgroup of $\mathrm{SL}_2(\mathbb{Z})$, then a modular form of level Γ and weight $k \in \mathbb{Z}$ is a function $f : \mathbb{H} \to \mathbb{C}$ such that:*

1. *For all $\gamma \in \Gamma$ we have $f \mid_k \gamma = f$ (w call such functions slash invariant).*

2. *f is holomorphic on $\mathbb{H}$.*

3. *For all $\gamma \in \mathrm{SL}_2(\mathbb{Z})$, there exist $A, B \in \mathbb{R}$ such that for all $z \in \mathbb{H}$, with $A \leq \mathrm{Im}(z)$, we have $|(f |_k \gamma)(z)| \leq B$. Here $| - |$ denotes the standard complex absolute value.*

This defines a complex vector space which we denote by $M_k(\Gamma)$. By replacing condition (3) in Definition 1 with (4) below defines the subspace of cusp forms, which we denote by $S_k(\Gamma)$.

4. For all $\gamma \in \mathrm{SL}_2(\mathbb{Z})$, and all $0 < \epsilon$, there exists $A \in \mathbb{R}$ such that for all $z \in \mathbb{H}$, with $A \leq \mathrm{Im}(z)$, we have $|(f |_k \gamma)(z)| \leq \epsilon$.

Note that $\Gamma_0(N)$ and $\Gamma_1(N)$ both contain the matrix $\gamma = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$. Therefore, if f is modular form of level $\Gamma_0(N)$ or $\Gamma_1(N)$ then $f(z+1) = (f |_k \gamma)(z) = f(z)$. It follows that f is $\mathbb{Z}$-periodic. Now a basic result in complex analysis tells us that any $\mathbb{Z}$-periodic holomorphic function has a Fourier expansion of the form $f(z) = \sum_{n=0}^{\infty} a_n(f) q^n$ where $q = e^{2\pi i z}$. Moreover, the Fourier coefficients of f can be expressed as (see, for example [7, Exercise 5.11.1]):

$$a_n(f) = \int_0^1 f(x+iy)e^{-2\pi i n(x+iy)}dx \tag{1}$$

These will be key later in stating our version of the modularity conjecture.

Spaces of modular forms come with operators which act on them, induced by the action of $\mathrm{GL}_2(\mathbb{Q})^+$, known as Hecke operators. These are closely linked to Fourier coefficients of modular forms and give us insights in to the structure of spaces of modular forms. We hope to use the results here to later begin formalising these theories.

We note that the definitions given are not the most general definitions possible. One can define modular forms for more general connected reductive groups (for example), but this would require a great deal more work and would delay the development of what are arguably the most common examples of modular forms. So this is an example where working in the highest level of generality is not feasible, which is in contrast to many of the other parts of mathlib.

2.1 Modular forms in Lean

We chose to work with Lean due to its extensive library (mathlib [10]) of formalised mathematical results. This library contains many of the basic notions required to formalise the definition of modular forms. For example, work of Alex Kontorovich, Heather Macbeth and Marc Masdeu formalised the definitions of the upper half plane and defined the above action for the group $\mathrm{SL}_2(\mathbb{R})$.

Since we would like to eventually incorporate Hecke operators, we first need to extend this action on the upper half plane to a larger group. We begin by defining the group $\mathrm{GL}_n(R)^+$ (of invertible matrices with positive determinant) for any commutative ring R equipped with a linear ordering.

```
def GL_pos {n : Type*} {R : Type*} [decidable_eq n] [fintype n]
[linear_ordered_comm_ring R ] :
subgroup (GL n R) :=(units.pos_subgroup R).comap general_linear_group.det
```

Explicitly, this says that given a finite type `n` and `R` a linear ordered commutative ring, then we define a subgroup of $\mathrm{GL}_n(R)$. Now, `comap` is such that given a group homomorphism $G_1 \to G_2$, and a subgroup H_2 of G_2 we can construct a subgroup of G_1 by taking its pre-image. So in order to construct a subgroup of $\mathrm{GL}_n(R)$ we need a group homomorphism $\mathrm{GL}_n(R) \to G_2$ and a subgroup of G_2. The group homomorphism is given by `general_linear_group.det` which denotes the map $\mathrm{GL}_n(R) \to R^\times$ given by the determinant map, and `units.pos_subgroup R` is the subgroup of $R^\times$ of positive elements. So, in words, we take the pre-image of the subgroup of positive elements in R under the determinant map.

Next we define the slash action by creating a new class which we call `slash_action`, bundling the main properties of this action.

```
class slash_action (β G α γ : Type*) [group G] [ring α] [has_smul γ α] :=
(map : β → G → α → α)
(mul_zero : ∀ (k : β) (g : G), map k g 0 = 0)
(one_mul : ∀ (k : β) (a : α), map k 1 a = a)
(right_action : ∀ (k : β) (g h : G) (a : α),
   map k h (map k g a) = map k (g * h) a )
(smul_action : ∀(k : β) (g : G) (a : α)(z : γ),
   map k g (z · a) = z · (map k g a))
(add_action : ∀ (k : β) (g : G) (a b : α),
   map k g (a + b) = map k g a + map k g b)
```

Here `mul_zero` and `one_mul` encode the fact that we want 0 to trivially and 1 to act as the identity. Similarly, `right_action`, `smul_action` and `add_action` encode (respectively) that we want this to be right action, we want the action to be equivariant under scaling[2] and that the action is additive. The advantage of defining this as a new class is that each time we prove an instance of a slash action, we can easily call these basic properties.

We also define the action induced by a monoid homomorphism from H to G. Specifically, we define:

```
def monoid_hom_slash_action {β G H α γ : Type*} [group G] [ring α]
[has_smul γ α] [group H] [slash_action β G α γ] (h : H →* G) :
   slash_action β H α γ :=
-- H →* G denotes a monoid homomorphism from H to G.
-- Arguments in curly brackets are implicit, meaning that, in practice,
    Lean will be able to infer them given the other arguments.
```

Using this, the slash action given by $\mathrm{GL}_2(\mathbb{R})^+$, induces actions by $\mathrm{SL}_2(\mathbb{Z})$ and subgroups Γ of $\mathrm{SL}_2(\mathbb{Z})$ by simply constructing the relevant monoid homomorphisms. It therefore suffices to define the $|_k\, \gamma$ action (for $\gamma \in \mathrm{GL}_2(\mathbb{R})^+$) on functions $\mathbb{H} \to \mathbb{C}$. We do this by first defining the map and then checking it satisfies the listed properties.

```
def slash (k : ℤ) (γ : GL(2, ℝ)⁺) (f : ℍ → ℂ) (x : ℍ) : ℂ :=
f (γ · x) * (((↑ₘ γ).det) : ℝ)^(k-1) * (upper_half_plane.denom γ x)^(-k)
```

This definition takes as input and integer k, a matrix A in $\mathrm{GL}_2(\mathbb{R})^+$ and a function $f : \mathbb{H} \to \mathbb{C}$ and returning a function $\mathbb{H} \to \mathbb{C}$. The definition of the map exploits many of existing matrix functionality in mathlib. Specifically, here $\uparrow_m$ is the coercion $\mathrm{GL}_2(\mathbb{R})^+ \to \mathrm{Mat}_{2,2}(\mathbb{R})$ which has a pre-existing definition of determinant, so $(\uparrow_m \gamma)$.det is the determinant of γ. Moreover `upper_half_plane.denom` is the function which given $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ and $z \in \mathbb{H}$ returns $cz + d$. This

[2] Here we have incorporated a scalar action (`has_smul`) of γ on α, which in practice will come from the $\mathbb{C}$-vector space structure on the space of functions $\mathbb{H} \to \mathbb{C}$.

function was already present in mathlib as it is used in constructing the fundamental domain for the action of $\mathrm{SL}_2(\mathbb{Z})$ on $\mathbb{H}$.

We subsequently formalise the proof that this does indeed define a slash action on the space of functions $\mathbb{H} \to \mathbb{C}$ resulting in an instance of the `slash_action` class defined above:

```
instance : slash_action ℤ GL(2, ℝ)⁺ (ℍ → ℂ) ℂ :=
```

We denote this slash action by $f \mid [k, \gamma]$. Similarly, by constructing the monoid homomorphism $\Gamma \to \mathrm{GL}_2(\mathbb{R})^+$ we get an induced action by subgroups:

```
instance subgroup_action (Γ : subgroup SL(2,ℤ)) :
   slash_action ℤ Γ (ℍ → ℂ) ℂ :=
```

The reason for having several instances of this action is that for the definitions of the spaces of modular forms we only want functions which are invariant under the slash action by a subgroup of $\mathrm{SL}_2(\mathbb{Z})$. By having separate actions we can then avoid having to explicitly coerce our elements into $\mathrm{GL}_2(\mathbb{R})^+$.

2.2 Slash invariant forms

Before defining modular forms and cusp forms, we first define functions that are invariant under the slash action, which we call *slash invariant forms*. For this, probably the simplest would be to define them as the subspace of functions from $\mathbb{H} \to \mathbb{C}$ such that for all $\gamma \in \Gamma$ we have $f \mid_k \gamma = f$. If we then imposed that they are holomorphic and bounded/zero at infinity we could easily define spaces of modular forms/cusp forms. This would make modular forms terms of some type and moreover any results for that hold for slash invariant forms would then have to be manually proved again for modular/cusp forms. For these reasons we instead extend the `fun_like` class to make new classes. This will not only make definitions of modular/cusp forms their own type, but will also allow for lemmas about slash invariant forms automatically extend to modular/cusp forms. Our first definition is therefore:

```
structure slash_invariant_form :=
(to_fun : ℍ → ℂ)
(slash_action_eq' : ∀ γ : Γ, to_fun ∣[k, γ] = to_fun)

class slash_invariant_form_class extends fun_like F ℍ (λ _, ℂ) :=
(slash_action_eq : ∀ (f : F) (γ : Γ), (f : ℍ → ℂ) ∣[k, γ] = f)
--Here we have (F : Type*) and Γ a subgroup of SL(2, ℤ) and k an integer.

instance slash_invariant_form_class.slash_invariant_form :
   slash_invariant_form_class (slash_invariant_form Γ k) Γ k :=
```

With this definition "slash invariant form Γ k" will be the type consisting of functions $\mathbb{H} \to \mathbb{C}$ together with the property that they are invariant under the slash action.[3] We furthermore define a class `slash_invariant_form_class` which extends the `fun_like` class by adding the condition that functions be slash invariant. Lastly we prove that `slash_invariant_form` is an instance of this class. By doing this, any result proven for `slash_invariant_form_class` will automatically hold for `slash_invariant_form` and, as we will see later, also for modular/cusp forms.

[3] If furthermore one requires them to be meromorphic, then such functions are called weakly modular.

We then give instances on `slash_invariant_form` which define addition, subtraction, the zero element, etc, finalising in the construction of the complex vector space of such functions:

```
instance : add_comm_group (slash_invariant_form Γ k) :=
fun_like.coe_injective.add_comm_group _ rfl coe_add coe_neg coe_sub
   coe_smul coe_smul

instance : module ℂ (slash_invariant_form Γ k) :=
coe_hom_injective.module ℂ coe_hom (λ _ _, rfl)
--coe_hom denotes the  hom (slash_invariant_form Γ k) →+ (ℍ → ℂ)
```

We note that in the first instance we are making use of the `fun_like` instance to give the additive commutative group structure[4], which is then needed for the second instance.

2.3 Modular forms and cusp forms

Moving forward to the definition of modular/cusp forms, the next hurdle is to formalise the notion of holomorphic function. Fortunately for us, there are several different options available to us in mathlib. For functions between manifolds, mathlib contains the definition of `mdifferentiable`, which describes when a map $f : M \to M'$ between manifolds M, M' is differentiable. Alternatively, mathlib has `differentiable_on` which describes differentiability of a functions on open subsets of normed vector spaces. Note that in our case, since $\mathbb{H}$ is not a normed vector space, we would need to first extend our functions (non-canonically) from $\mathbb{H}$ to, say, $\mathbb{C}$. For this reason we use `mdifferentiable` instead, which only requires us to turn $\mathbb{H}$ into a (complex) manifold. We do this by using the fact that we already know that the coercion $\mathbb{H} \to \mathbb{C}$ is an open embedding and that this will in turn induce the complex manifold structure on $\mathbb{H}$. For brevity we will skip the details here.

Lastly, we need to formalise Definition 1 (3) and (4). We begin by defining the notion of a function being bounded and zero on a filter.[5]

```
def zero_at_filter [has_zero β] [topological_space β]  (l : filter α)
(f : α → β) : Prop := filter.tendsto f l (𝓝 0)

def bounded_at_filter [has_norm β] [has_one (α → β)] (l : filter α)
(f : α → β) : Prop := asymptotics.is_O l f (1 : α → β)
```

Here the first definition says that the pre-image of f along the filter l approaches 0 (using the filter of neighbourhoods of 0 denoted $\mathcal{N}(0)$). Similarly, the second definition says that a function is bounded on a filter if it is eventually dominated by the constant function (denoted $1 : \alpha \to \beta$).

In order to use this for our definition we then need to define a filter representing the limit of a function on the upper half plane tending to infinity in the imaginary direction.

```
def at_im_infty := filter.at_top.comap upper_half_plane.im
```

[4] Here `fun_like.coe_injective.add_comm_group` says we can define a additive commutative group structure on a type (with $0, +$) if it admits an injection into a `add_comm_group` preserving $0, +$.

[5] We are grateful to David Loeffler for suggesting this filter approach to these definitions.

Here `filter.at_top` is the filter representing the limit to "infinity" on an ordered set. Therefore, `filter.at_top.comap upper_half_plane.im` is the filter representing the limit at infinity along imaginary direction in ℍ. Using this we define:

```
def is_zero_at_im_infty {α : Type*} [has_zero α] [topological_space α]
(f : ℍ → α) : Prop := zero_at_filter at_im_infty f

def is_bounded_at_im_infty {α : Type*} [has_norm α] [has_one (ℍ → α)]
(f : ℍ → α) : Prop := bounded_at_filter at_im_infty f
```

Lastly, we check that these definitions agree with the more classical formulation as in Definition 1 (3) and (4) :

```
lemma zero_at_im_infty (f : ℍ → ℂ) : is_zero_at_im_infty f ↔
∀ ε : ℝ, 0 < ε → ∃ A : ℝ, ∀ z : ℍ, A ≤ im z → abs (f z) ≤ ε :=

lemma bounded_mem (f : ℍ → ℂ) : is_bounded_at_im_infty f ↔
∃ (M A : ℝ), ∀ z : ℍ, A ≤ im z → abs (f z) ≤ M :=
```

We are now in ready to define modular forms and cusp forms as structures extending `slash_invariant_form`

```
structure modular_form extends slash_invariant_form Γ k :=
(holo' : mdifferentiable 𝓘(ℂ) 𝓘(ℂ) (to_fun : ℍ → ℂ))
(bdd_at_infty' : ∀ (A : SL(2, ℤ)), is_bounded_at_im_infty (to_fun ∣[k, A]))

class modular_form_class extends slash_invariant_form_class F Γ k :=
(holo: ∀ f : F, mdifferentiable 𝓘(ℂ) 𝓘(ℂ) (f : ℍ → ℂ))
(bdd_at_infty : ∀ (f : F) (A : SL(2, ℤ)),
   is_bounded_at_im_infty (f ∣[k, A]))

structure cusp_form extends slash_invariant_form Γ k :=
(holo' : mdifferentiable 𝓘(ℂ) 𝓘(ℂ) (to_fun : ℍ → ℂ))
(zero_at_infty' : ∀ (A : SL(2, ℤ)), is_zero_at_im_infty (to_fun ∣[k, A]))

class cusp_form_class extends slash_invariant_form_class F Γ k :=
(holo: ∀ f : F, mdifferentiable 𝓘(ℂ) 𝓘(ℂ) (f : ℍ → ℂ))
(zero_at_infty : ∀ (f : F) (A : SL(2, ℤ)), is_zero_at_im_infty (f ∣[k, A]))
    --Note that throughout 𝓘(ℂ) denotes model_with_corners ℂ ℂ,
   -- (F : Type*) and Γ k are the level and weight respectively.
```

As above we then give a list of instances on these spaces finalising in:

```
instance : module ℂ (modular_form Γ k) :=
instance : module ℂ (cusp_form Γ k) :=
instance [cusp_form_class F Γ k] : modular_form_class F Γ k :=
```

Here the last instance says that a `cusp_form_class` is also a `modular_form_class` (which is just a restatement of the fact that cusp forms are modular forms). As an example of how working with these classes can be beneficial, consider the following lemma:

```
lemma slash_action_eqn' (k : ℤ) (Γ : subgroup SL(2, ℤ))
```

```
[slash_invariant_form_class F Γ k] (f : F)(γ : Γ) (z : ℍ) :
f (γ • z) = ((↑ₘγ 1 0 : ℂ) * z +(↑ₘγ 1 1 : ℂ))^k * f z :=
```

this lemma[6] is stated for `slash_invariant_forms` it will automatically hold for any instance of a `modular_form_class` or `cusp_form_class`.

Lastly we prove a graded commutative ring instance on the space of modular forms of level Γ and any weight. For this we first define the product of two modular forms

```
def mul {k_1 k_2 : ℤ} {Γ : subgroup SL(2, ℤ)} (f : (modular_form Γ k_1))
(g : (modular_form Γ k_2)) : (modular_form Γ (k_1 + k_2)) :=
```

Now, proving the graded commutative ring instance is delicate due to definitional equality issues. For example, we need to prove that for $f \in M_k(\Gamma)$ and $1 \in M_0(\Gamma)$ (the modular form of level Γ and weight 0) we have $1 \cdot f = f$. But with our definitions, we have $1 \cdot f$ is an element of $M_{0+k}(\Gamma)$ and this is not definitionally equal to $M_k(\Gamma)$. To get around these issues we define the map $M_a(\Gamma) \to M_b\Gamma$ under the hypothesis that $a = b$

```
def mcast {a b : ℤ} {Γ : subgroup SL(2, ℤ)} (h : a = b)
(f : modular_form Γ a) : (modular_form Γ b) :=
```

This will then give us a map $M_{0+k}(\Gamma) \to M_k(\Gamma)$ or from $M_{a+b}(\Gamma) \to M_{b+a}(\Gamma)$, etc. Using this with heterogeneous equalities "==" (which allow for writing equalities between terms of different types) we can prove, for example,

```
lemma heq_one_mul (k : ℤ) {Γ : subgroup SL(2, ℤ)} (f : modular_form Γ k):
(1 : modular_form Γ 0).mul f == f := --This says 1 * f = f
```

From which one can then show that, in the graded ring of modular forms, $1 \cdot f = f$, since this requires one to fist prove that the elements of both sides first have the same weight and then that $1 \cdot f == f$. Combining this with similar lemmas we obtain:

```
instance graded_mod_ring (Γ : subgroup SL(2, ℤ)) :
   direct_sum.gcomm_ring (λ k, modular_form Γ k) :=
{ mul := λ k_1, λ k_2, λ f g, f.mul g, ...}
```

2.4 Congruence subgroups

We briefly summarise our formalisation of the standard levels of modular forms. As shown above these are $\Gamma_0(N), \Gamma_1(N)$ and $\Gamma(N)$ for $N \in \mathbb{Z}_{>0}$. In general, a subgroup Γ of $\mathrm{SL}_2(\mathbb{Z})$ is known as a congruence subgroup if it contains some $\Gamma(N)$ (with $N > 0$).

Our strategy is to first formalise $\Gamma(N)$ and using this we then define congruence subgroups. Since mathlib contains the result that the kernel of a group homomorphism is a subgroup, we define $\Gamma(N)$ as the kernel of the map $\mathrm{SL}_2(\mathbb{Z}) \to \mathrm{SL}_2(\mathbb{Z}/N\mathbb{Z})$ given by reducing the entries modulo N.

```
def Gamma (N : ℕ) : subgroup SL(2, ℤ) := (SLMOD(N)).ker
```

which is saying we define $\Gamma(N)$ as the kernel of the group homomorphism `(SLMOD(N))`, which is the entry-wise reduction modulo N map. From this we define congruence subgroups as:

[6] Which simply states the slash invariant forms (for a subgroup of SL_2) f satisfy the usual expression $f\left(\frac{az+b}{cz+d}\right) = (cz+d)^k f(z)$.

```
def is_congruence_subgroup (Γ : subgroup SL(2, ℤ)) : Prop :=
∃ (N : ℕ+), Gamma_N N ≤ Γ -- ℕ+ denotes the positive integers
```

We also prove some basic properties of congruence subgroups, such as a subgroup containing a congruence subgroup is itself a congruence subgroup and a conjugate of a congruence subgroup is again a congruence subgroup. For brevity we omit these details.

We define $\Gamma_0(N)$ as the subgroup of $\mathrm{SL}_2(\mathbb{Z})$ consisting of elements whose lower left-hand entry is zero modulo N.

```
def Gamma0_N (N : ℕ) : subgroup SL(2, ℤ) :={
carrier := { g : SL(2, ℤ) | (g 1 0 : zmod N) = 0}, ...}
-- g 1 0 denotes the lower left-hand entry of g.
```

Next we define the group homomorphism from $\Gamma_0(N) \to \mathbb{Z}/N\mathbb{Z}$ given by $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \to d \pmod N$. Using this we define $\Gamma_1(N)'$ as the kernel of this group homomorphism and then $\Gamma_1(N)$ is defined as the image of this subgroup under the composition of $\Gamma_0(N) \to \mathrm{SL}_2(\mathbb{Z})$ and $\Gamma_1(N)' \to \Gamma_0(N)$.

```
def Gamma_0_map (N : ℕ): (Gamma0_N N) →* (zmod N) :=
{ to_fun :=  λ g, (g 1 1 : zmod N), ...}

def Gamma1_N' (N : ℕ) :subgroup (Gamma0_N N) := (Gamma_0_map N).ker

def Gamma1_map (N : ℕ) : (Gamma1_N' N) →* SL(2, ℤ) :=
((Gamma0_N N).subtype).comp (Gamma1_N' N).subtype

def Gamma1_N (N : ℕ) : subgroup SL(2, ℤ) :=
subgroup.map (Gamma1_map N) ⊤
--Here ⊤ denotes SL(2, ℤ) considered as a subgroup of itself.
```

3 Eisenstein series

Perhaps the most basic examples of non-trivial modular forms (for weights $k > 2$ and even) are Eisenstein series. At their most basic, these are function $\mathbb{H} \to \mathbb{C}$ defined by

$$G_k(z) = \sum_{(c,d)\neq(0,0)} \frac{1}{(cz+d)^k}, \qquad \text{for } c, d \in \mathbb{Z}.$$

The definitions and results described below are all standard and can be found in many sources, see for example [7]. We formalise this is two steps. We first define:

```
def Eise (k: ℤ) (z : ℍ) : ℤ × ℤ → ℂ :=
λ x, 1/(x.1 * z + x.2)^k
```

where for $x \in \mathbb{Z} \times \mathbb{Z}$, `x.i` denotes the i-th component of the element. Note that we have not restricted to $(c, d) \neq 0$ since, by convention, in mathlib we have $0^{-1} = 0$ (with appropriate modifications elsewhere to ensure this does not lead to contradictions). Using this we set:

```
def Eisenstein_series_of_weight_ (k: ℤ) : ℍ → ℂ :=
λ z, Σ' (x : ℤ × ℤ), (Eise k z x)
```

Here $\sum'$ denotes `tsum` which defines infinite sums in topological monoids in mathlib. This is defined such that if the sum converges absolutely then it produces the correct value, otherwise it returns zero.

Note that we cannot use this definition to define $G_2(z)$, which, whilst not a modular form, is still a function of number theoretic interest. One of the reasons this function is not a modular form is that it is not slash invariant, which is due to it only being conditionally convergent. Therefore, setting $k = 2$ in our definition gives the zero function as `tsum` returns zero in this case. On the other hand, with this definition, we can prove that our functions are slash invariant without any conditions on the weight.

In order to verify that they are slash invariant, we begin by defining an equivalence $\mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \times \mathbb{Z}$ induced by the action of $\mathrm{SL}_2(\mathbb{Z})$, where $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ sends (x, y) to $(xa + yc, xb + yd)$. We call this `Ind_equiv A`. Using this we have:

```
lemma Eise_moeb (k : ℤ) (z : ℍ) (A : SL(2,ℤ)) (i : ℤ × ℤ ) :
Eise k ( (A : GL(2, ℝ)⁺)) · z) i =
   ((A.1 1 0 * z + A.1 1 1)^k) * (Eise k z (Ind_equiv A i ) ) :=
```

which describes how `Eise` transforms under Moebius transformations. We can then show that Eisenstein series define a `slash_invariant_form`

```
def Eisenstein_is_slash_inv (Γ : subgroup SL2Z) (k: ℤ) :
(slash_invariant_form Γ k) :=
```

The main lemmas on which we rely are `equiv.tsum_eq` which says that the value of `tsum` is unchanged after permuting the index set by an equivalence (in this case `Ind_equiv`) and `tsum_mul_left` which say that for a fixed a, multiplication by a on the left, commutes with `tsum`. These two results together with `Eise_moeb` then completes the proof.

We next verify that for $k \geq 3$, G_k is uniformly and absolutely convergent. This result will be required to check G_k is holomorphic. We begin by summarising the strategy of proof. For this it is convenient to rewrite G_k as

$$G_k(z) = \sum_{n=0}^{\infty} G_{k,n}(z), \qquad G_{k,n}(z) := \sum_{(c,d)\in S(n)} \frac{1}{(cz+d)^k}$$

where $S(n) = \{(c, d) \in \mathbb{Z} \times \mathbb{Z} \mid \max(|c|, |d|) = n\}$. Noting that for $n \geq 1$, $S(n)$ has $8n$ elements, one has that each $G_{k,n}$ is a finite sum, each of which can be bounded to give a bound on G_k in terms of the Riemann zeta function $\zeta(k) = \sum_{n=1}^{\infty} n^{-k}$. Specifically, we have $G_{k,n,\mathrm{abs}}(z) \leq 8n^{1-k} r(z)^{-k}$, where

$$G_{k,n,\mathrm{abs}}(z) := \sum_{(c,d)\in S(n)} \frac{1}{|(cz+d)^k|} \qquad \text{and} \qquad r(x+iy) := \min\left(y, \left(\frac{y^4 + (xy)^2}{(x^2+y^2)^2}\right)^{1/2}\right).$$

Using the triangle inequality it follows that $|G_k(z)| \leq \sum_n G_{k,n,\mathrm{abs}}(z) \leq 8\zeta(k-1) r(z)^{-k}$. Moreover, if we let $\mathbb{H}_{a,b} := \{z = x + iy \in \mathbb{H} \mid |x| \leq a, |y| \geq b\}$ for $a, b \in \mathbb{R}$, with $0 < b$, then for any $z \in \mathbb{H}_{a,b}$ we have $r(z)^{-k} \leq r(a + ib)^{-k}$. We call $\mathbb{H}_{a,b}$ an *upper half space slice*. Combining this with the bound above, we see that G_k is uniformly bounded on $\mathbb{H}_{a,b}$. This means we can use the Weierstrass M-test[7] to show that on each $\mathbb{H}_{a,b}$, the functions

[7] Originally we had also formalised the proof of the M-test, but a more general version has been added to mathlib by Heather Macbeth.

$\sum_{n=0}^{N} G_{k,n}$ converge absolutely and uniformly to G_k. From this it will follow that G_k is holomorphic on each $\mathbb{H}_{a,b}$ and therefore holomorphic on $\mathbb{H}$.

We have formalised all of the above steps. For brevity we will not show each required lemma, but only the final statements whose proofs have been formalised.

First we note that in mathlib the property of a `tsum` being absolutely convergent is called `summable`. Similarly, `tendsto_uniformly` formalises the notion of a sequence of functions f_n converging uniformly to a limit function f.

```
lemma Eisenstein_series_is_summable (k : ℤ)
(z : ℍ) (h : 3 ≤ k) : summable (Eise k z) :=

lemma Eisen_partial_tendsto_uniformly (k : ℤ) (h : 3 ≤ k)
(A B : ℝ) (ha : 0 ≤ A) (hb : 0 < B) :
tendsto_uniformly (Eisen_par_sum_slice k A B)
(Eisenstein_series_restrict k A B hb) filter.at_top :=
```

Here `Eisen_par_sum_slice` denotes the function sending $N \in \mathbb{N}$ to the function given by mapping $z \in \mathbb{H}_{a,b}$ to $\sum_{n=0}^{N} G_{k,n}(z)$ and `Eisenstein_series_restrict` is the restriction of G_k to $\mathbb{H}_{a,b}$.

To define `Eisen_par_sum_slice`, we first define $G_{k,m}$:

```
def eisen_square (k : ℤ) (n : ℕ) : ℍ → ℂ :=
λ z, Σ x in Square n, Eise k z x --Square n denotes the S(n) from above
```

We can relate this to our definition of Eisenstein series with the following lemma which says that the sum over all n of $G_{k,n}$ equals G_k (for $k \geq 3$).

```
lemma Eisenstein_series_is_sum_eisen_squares (k: ℕ) (z: ℍ) (h : 3 ≤ k) :
(Eisenstein_series_of_weight_ k z) = Σ' (n : ℕ), eisen_square k n z :=
```

The proof of uniform convergence then proceeds by restricting each side of this equality to $\mathbb{H}_{a,b}$ and proving uniform convergence there. The restriction of the right hand side to $\mathbb{H}_{a,b}$ is `Eisen_par_sum_slice`.

As mentioned above, main obstruction to checking G_k is a modular form in Lean is the holomorphicity condition. Whilst it is simple to check that for fixed $(c, d) \neq (0, 0)$ the function $\frac{1}{(cz+d)^k}$ is holomorphic on $\mathbb{H}$, checking that the infinite sum defines a holomorphic function requires a non-trivial complex analysis theorem ([8, Theorem 1.1]):

▶ **Theorem 2.** *Let $\{f_n\}$ be a sequence of holomorphic functions on an open subset S. If for each compact subset C of S, the sequence converges uniformly to a limit function f, then f is holomorphic.*

One way of proving this is to use Cauchy's integral formula, which tells us that, since each f_n is holomorphic, for each $x \in S$, we can write

$$f_n(w) = \int_{C_r(x)} \frac{f_n(\zeta)}{\zeta - w} d\zeta = \frac{1}{2\pi i} \int_0^{2\pi} \frac{re^{i\theta} i f_n(x + re^{i\theta})}{(x + re^{i\theta} - w)} d\theta$$

for $C_r(x)$ a sufficiently small disk around x (in S), of radius r containing w. Since f_n converges uniformly to f, it follows that $f(x) = \int_{C_r(x)} \frac{f(\zeta)}{\zeta - x} d\zeta$ and therefore f is also differentiable (by differentiating under the integral).

Note is that since holomorphicity is a local property, it suffices to restrict to S being an appropriately chosen disk. We start by showing we can interchange the limit with the integral.

For this we define `circle_transform`, which denotes the function $\theta \mapsto \frac{1}{2\pi i}\frac{re^{i\theta}if(x+re^{i\theta})}{(x+re^{i\theta}-w)}$. This is formalised as:

```
variables {E : Type} [normed_add_comm_group E] [normed_space ℂ E] (R : ℝ)
(z w : ℂ)

def circle_transform (f : ℂ → E) (θ : ℝ) : E :=
(2 * ↑π * I)⁻¹ • deriv (circle_map z R) θ • ((circle_map z R θ) - w)⁻¹ •
   f(circle_map z R θ) -- I denotes √−1.
```

Here `circle_map` is the function defining the circle around c and radius R, i.e, $c + Re^{i\theta}$. Using this we have:

```
lemma circle_int_uniform_lim_eq_lim_of_int
{R : ℝ} {F : ℕ → ℂ → ℂ} (hR : 0 < R)
(f : ℂ → ℂ) (z : ℂ) (w : ball z R)
(F_cts : ∀ n, continuous_on (F n) (sphere z R))
(hlim : tendsto_uniformly_on F f filter.at_top (sphere z R)) :
tendsto (λ n, ∫ (θ : ℝ) in 0..2 * π, (circle_transform R z w (F n)) θ)
at_top (𝓝 (∫ (θ : ℝ) in 0..2 * π, (circle_transform R z w f) θ)) :=
```

In words this says that if F_n is a sequence of continuous functions converging uniformly on a sphere to a function f, then the sequence integrals of the functions `circle_transform F_n`, converge to the integral of the function `circle_transform f`. Note that if each F_n is in fact holomorphic, then by Cauchy's integral formula we have F_n is the same as the integral of `circle_transform F_n` (over a suitably chosen disk).

The proof of this result relies on a formalised version of Lebesgue dominated convergence theorem, known in mathlib as `tendsto_integral_of_dominated_convergence`.

Next we want to show that the function $\mathbb{C} \to \mathbb{C}$ given by $w \mapsto \int_{C_R(x)} \frac{f(\zeta)}{\zeta - w} d\zeta$ is differentiable. First define

```
def circle_integral_form [complete_space E]
(R : ℝ) (z : ℂ) (f : ℂ → E) : (ℂ → E) :=
λ w, (2 * π * I : ℂ)⁻¹ • (∮ z in C(z, R), (z - w)⁻¹ • f z)
```

which we check agrees with the integral of the `circle_transform` function:

```
lemma circle_intgral_form_eq_int [complete_space E] (R : ℝ) (z : ℂ)
(f : ℂ → E) : circle_integral_form R z f =
λ w, ∫ (θ : ℝ) in 0..2 * π, (circle_transform R z w f) θ :=
```

We then have:

```
lemma circle_integral_form_differentiable_on
{R : ℝ} {f : ℂ → ℂ} (hR : 0 < R) (z : ℂ)
(hf : continuous_on f (sphere z R)) :
differentiable_on ℂ (circle_integral_form R z f) (ball z R) :=
```

This says that if we take a function f which is continuous on the sphere around a point z of radius R, then function `circle_integral_form f` is differentiable on the open disk around z of radius R. We note that the proof builds upon existing mathlib results, such as `has_deriv_at_integral_of_dominated_loc_of_deriv_le` which gives (in greater generality) the derivative under an integral of such functions. These results then combine to give the desired result:

```
lemma uniform_of_diff_circle_int_is_diff {R : ℝ} (F : ℕ → ℂ → ℂ)
(f : ℂ → ℂ) (z : ℂ) (hR : 0 < R)
(hdiff : ∀ (n : ℕ), differentiable_on ℂ (F n) (closed_ball z R))
(hlim : tendsto_uniformly_on F f filter.at_top (closed_ball z R)) :
differentiable_on ℂ f (ball z R) :=
```

This result is enough to prove that a uniform limit of `mdifferentiable` functions $f_n : \mathbb{H} \to \mathbb{C}$ is again `mdifferentiable`.[8]

Combining the above results and using Cauchy's Integral Formula, we get:

```
lemma Eisenstein_series_is_mdiff (k : ℤ) (hk : 3 ≤ k) :
mdifferentiable 𝓘(ℂ, ℂ) 𝓘(ℂ, ℂ) ↑ₕ(Eisenstein_is_slash_inv ⊤ ↑k) :=
--Here ↑ₕ is the coercion to a function between complex manifolds.
```

The final step is to check that Eisenstein series are bounded at infinity. This actually requires us to use the fact that they are slash invariant. Specifically, using the fact that for any slash invariant function f of level $\mathrm{SL}_2(\mathbb{Z})$ is periodic with period 1, i.e., $f(z+1) = f(z)$. It follows that for any $z = x + iy \in \mathbb{H}$ we can find $z' \in \mathbb{H}_{2,y}$ such that $G_k(z) = G_k(z')$. So it is enough to check its restriction to a chosen `upper_half_space_slice` is bounded at infinity. By using the bounds from the proof uniform convergence, we get:

```
lemma Eisenstein_series_is_bounded (k : ℤ) (hk : 3 ≤ k) (A: SL(2, ℤ)) :
is_bounded_at_im_infty ((↑ₕ(Eisenstein_is_slash_inv ⊤ k))∣[k ,A])
```

Combining all of these results we finally a sorry-free proof that Eisenstein series are modular forms.

```
def Eisenstein_series_is_modular_form (k : ℤ) (hk : 3 ≤ k) :
   modular_form  ⊤ k :=
{ to_fun := ↑ₕ(Eisenstein_is_slash_inv ⊤ k),
  slash_action_eq' := by {convert (Eisenstein_is_slash_inv ⊤ k).2},
  holo' := Eisenstein_series_is_mdiff k hk,
  bdd_at_infty' := λ A, Eisenstein_series_is_bounded k hk A}
```

4 The Modularity Conjecture

Finally lets turn to the statement of the modularity conjecture. We begin by recalling the definition of an elliptic curve, which is a pair $(E, \mathcal{O})$ consisting of a smooth projective curve E of genus one and $\mathcal{O}$ a point on E. Now, every elliptic curve can be embedded as a smooth cubic curve in $\mathbb{P}^2$ given by an equation of the form $E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ and this is the basis for the current definition of an elliptic curve in mathlib, where roughly it is described by the above equation, with the extra condition that the discriminant of this cubic is invertible over the base ring.

▶ **Definition 3.** *Let $E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ be the Weierstrass equation of an elliptic curve and for p a prime number, let $n_p(E)$ denote the number of solutions to $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ in $\mathbb{F}_p$. Then we define $a_p(E) := p - n_p(E)$.*

[8] Recently, a more general version of this result has been added to mathlib by Vincent Beffarra which does not use the circle integral machinery. But this machinery is still being added to mathlib as it is convenient for Cauchy's formulas for higher derivatives.

```
def elliptic_curve.ap (E : elliptic_curve ℚ) (p : ℕ) : ℕ :=
p-(cardinal.mk (set_of_points_mod_n E p)).to_nat
```

Here `cardinal.mk` takes a set and returns its cardinality and `.to_nat` turns this into a natural number (which is non-zero if the set is non-empty and finite). We also have:

```
def rat_red (q : ℚ) ( p : ℕ) : (zmod p) :=
   (q.num : zmod p) * (q.denom : zmod p)⁻¹
```

This function allows us to reduce the rational coefficients of our elliptic curve modulo p. Noting that if any denominator is divisible by p, this will return zero. Lastly we have:

```
def set_of_points_mod_n (E : elliptic_curve ℚ) (n : ℕ) :=
{P : (zmod n) × (zmod n) | let ⟨x, y⟩ := P in  y^2 +
   (rat_red E.a₁ n)* x * y+ (rat_red E.a₃ n) * y   =
   x^3 +(rat_red E.a₂ n)* x^2 + (rat_red E.a₄ n) * x + (rat_red E.a₆ n)}
```

We note that the definition of `elliptic_curve.ap` makes use of Lean's `namespace` functionality. By naming it as `elliptic_curve.ap`, we can then ask for this value, for any elliptic curve E by simply writing `E.ap` and providing the p we desire. Note that `E.ap i` is different from E.a_i.

Turning now to modular forms, note that while we have not defined q-expansions of modular forms in Lean, we can still define the Fourier coefficients $a_n(f)$ of a modular form f, by using the expression given by equation (1) (and setting $y = 1$) above.

```
def modular_form_an (n : ℕ) {N : ℕ} {k : ℤ} (f : cusp_form (Gamma0 N) k)
: ℂ :=  ∫ (x : ℝ) in 0..1,
   ( exp (-2 * π * I * n *(x + I))) * f.1 (map_to_upper x)
--Here map_to_upper send x to x + I in ℍ
```

Next we formalise the definition of an eigenform. Typically eigenforms are defined as functions that are eigenvectors for all Hecke operators, but since we have not yet defined Hecke operators, we give an alternative equivalent definition (see [7, Proposition 5.8.5]) together with the formalised version :

▶ **Definition 4.** *Let $f \in M_k(\Gamma_0(N))$. Then f is a normalised eigenform if:*
1. $a_1(f) = 1$.
2. *For p a prime and $r \geq 2$, $a_{p^r}(f) = a_p(f)a_{p^{r-1}}(f) - p^{k-1}a_{p^{r-2}}(f)$.*
3. *For n, m coprime, $a_{mn}(f) = a_m(f)a_n(f)$.*

```
def is_normalised_eigenform {N : ℕ} {k : ℤ}
(f : cusp_form (Gamma0 N) k) : Prop :=
(a_[1] f) = 1 ∧
∀ (m n : ℕ) (hmn : m.coprime n), ((a_[n * m] f) = (a_[n] f) * (a_[m] f)) ∧
∀ (p r : ℕ) (hp : p.prime ) (hr : 2 ≤ r),
(a_[p^r] f) = (a_[p] f) * (a_[p^(r-1)] f) - (p^(k-1)) * (a_[p^(r-2)] f)
```

Here `cusp_form (Gamma0 N) k` denotes $S_k(\Gamma_0)$ and `a_[n] f` denotes $a_n(f)$. Note, for simplicity, only define the notion for cusp forms since it is all that we will require for the modularity conjecture. Finally, lets look at one equivalent statement of the modularity conjecture (see [7, Theorem 8.8.1]):

▶ **Theorem 5** (Shimura–Taniyama–Weil conjecture: a_p version). *Let E be an elliptic curve over $\mathbb{Q}$. Then there exists $N \in \mathbb{N}$ and a normalised cuspidal eigenform $f \in S_2(\Gamma_0(N))$ such that for all primes p with $p \nmid N$, we have $a_p(E) = a_p(f)$ where $f = \sum_n a_n(f)q^n$ is the q-expansion of f.*

```
theorem modularity_conjecture (E : elliptic_curve ℚ) : ∃ (N : ℕ)
(f : cusp_form (Gamma0 N) 2)
(hf : is_normalised_eigenform f),
∀ (p : ℕ) (hp : p.prime ) (hN : (N : zmod p ) ≠ 0 ), a_[p] f = E.ap p :=
```

▶ Remark 6. We note that since we have not defined the conductor of an elliptic curve, our notion of $a_p(E)$ differs from that in [7, Theorem 8.8.1] for primes p of bad reduction. For this reason we state the modularity conjecture with the assumption that $p \nmid N$, in order to give an equivalent statement.

5 Future work and conclusion

We have shown how one can formalise the classical definitions of modular forms and Eisenstein series, as well as showing the main challenges to proving basic results about these objects, such as the holomorphicity of Eisenstein series. All of the results described above have complete, "sorry-free" proofs (other than the modularity conjecture). The code shown here is currently in the process of being added to mathlib a process which will no doubt improve the code and ensure its future utility.

Formalising the proof of the modularity conjecture and Fermat's Last Theorem will require a great deal more work, but the results here are a small step in this direction. The formalisation of these concepts has opened the door to formalising many other results in this area, which we now describe:

- **Eisenstein series**: The reader will notice that in the above we have not yet formalised a complete proof that Eisenstein series are non-trivial. At the time of writing we do have q-expansions of Eisenstein series and a proof that they are non-trivial, but they currently depend some basic analytic identities, specifically the Mittag–Leffler expansion for the cotangent function, which we are currently formalising.
- **Modular forms** : We would like to in the future give examples of cusp forms in Lean. This will also require understanding the q-expansions of modular forms more generally. Many other basic properties still remain to be formalised, such as the proof that the spaces of modular forms are finite dimensional. While mathematically some of the proofs are not that advanced, such as the proof for level $\mathrm{SL}_2(\mathbb{Z})$, the general case relies on complex analytic methods, such as the valence formula.
- **Hecke operators** : Spaces of modular forms are acted upon by linear transformations, known as Hecke operators. We have formalised the definitions of modular forms with Hecke operators in mind. In the future we hope to define Hecke operator and formalise results relating to their action on spaces of modular forms. The aim being to begin formalising what is known as Atkin–Lehner theory, which allows a much deeper study of modular forms. Another result we hope to formalise in the future is the multiplicity one theorem for modular forms, which would be an excellent test case for the formalised theory.

References

1 R. Brasca, K. Buzzard, J. Commelin, H. Macbeth, P. Massot, B. Mehta, S. Morrison, F. Nuccio, P. Scholze, D. Testa, A. Topaz, et al. Liquid tensor experiment. `https://github.com/leanprover-community/lean-liquid`.

2 C. Breuil, B. Conrad, and R. Diamond, F. andx Taylor. On the modularity of elliptic curves over $\mathbf{Q}$: wild 3-adic exercises. *J. Amer. Math. Soc.*, 14(4):843–939, 2001. `doi:10.1090/S0894-0347-01-00370-8`.

3 K. Buzzard, J. Commelin, and P. Massot. Formalising perfectoid spaces. *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, Jan 2020. URL: `http://dx.doi.org/10.1145/3372885.3373830`, `doi:10.1145/3372885.3373830`.

4 B. Conrad, F. Diamond, and R. Taylor. Modularity of certain potentially Barsotti-Tate Galois representations. *J. Amer. Math. Soc.*, 12(2):521–567, 1999. `doi:10.1090/S0894-0347-99-00287-8`.

5 G. Cornell, J.H. Silverman, and G. Stevens. *Modular Forms and Fermat's Last Theorem*. Springer New York, 2013.

6 L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The Lean theorem prover (system description). In A. P. Felty and A. Middeldorp, editors, *Automated Deduction - CADE-25*, volume 9195 of *LNCS*, pages 378–388. Springer, 2015. `doi:10.1007/978-3-319-21401-6_26`.

7 F. Diamond and J. Shurman. *A first course in modular forms*, volume 228 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2005.

8 S. Lang. *Complex analysis*, volume 103 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, fourth edition, 1999. `doi:10.1007/978-1-4757-3083-8`.

9 T. Miyake. *Modular forms*. Springer Monographs in Mathematics. Springer-Verlag, Berlin, english edition, 2006. Translated from the 1976 Japanese original by Yoshitaka Maeda.

10 The mathlib Community. The Lean mathematical library. In J. Blanchette and C. Hrițcu, editors, *CPP 2020*, page 367–381. ACM, 2020. `doi:10.1145/3372885.3373824`.