A formalization of Deligne's theorem

Presented at HIM Trimester Program:
"Prospects of Formal Mathematics"

Michail Karatarakis

Radboud University Nijmegen

23 May 2024

An overview of the talk

- What is Fermat's Last Theorem (FLT)?
- Why do we want to formalize the proof of FLT?
- Are there people working on the problem as posed?
- The first tentative steps towards such a formalization
- Levels of partial formalization
- What does the proof of FLT look like?
- Deligne's theorem

Fermat's Last Theorem (FLT) [1]

Fermat's Last Theorem is a mathematical theorem that states that there are no three positive integers a, b, and c that satisfy the equation $a^n + b^n = c^n$ for any integer value of n greater than 2.

Fermat's Last Theorem (FLT) [1]

Fermat's Last Theorem is a mathematical theorem that states that there are no three positive integers a, b, and c that satisfy the equation $a^n + b^n = c^n$ for any integer value of n greater than 2.

The theorem was first proposed by Pierre de Fermat around 1637 in the margin of a copy of Diophantus's Arithmetica. Fermat added that he had a proof that was too large to fit in the margin.

Fermat's Last Theorem (FLT) [1]

Fermat's Last Theorem is a mathematical theorem that states that there are no three positive integers a, b, and c that satisfy the equation $a^n + b^n = c^n$ for any integer value of n greater than 2.

The theorem was first proposed by Pierre de Fermat around 1637 in the margin of a copy of Diophantus's Arithmetica. Fermat added that he had a proof that was too large to fit in the margin.

His claim was only discovered after his death, by his son.

Fermat's Last Theorem (FLT) [1]

Fermat's Last Theorem is a mathematical theorem that states that there are no three positive integers a, b, and c that satisfy the equation $a^n + b^n = c^n$ for any integer value of n greater than 2.

The theorem was first proposed by Pierre de Fermat around 1637 in the margin of a copy of Diophantus's Arithmetica. Fermat added that he had a proof that was too large to fit in the margin.

His claim was only discovered after his death, by his son.

It is widely believed that Fermat did not have a correct proof of the result at the time.

Fermat's Last Theorem (FLT) [1]

Fermat's Last Theorem is a mathematical theorem that states that there are no three positive integers a, b, and c that satisfy the equation $a^n + b^n = c^n$ for any integer value of n greater than 2.

The theorem was first proposed by Pierre de Fermat around 1637 in the margin of a copy of Diophantus's Arithmetica. Fermat added that he had a proof that was too large to fit in the margin.

His claim was only discovered after his death, by his son.

It is widely believed that Fermat did not have a correct proof of the result at the time.

In 1994, around 350 years after Fermat's claim, Andrew Wiles announced a proof. However, it had a gap which he fixed a year later through joint work with Taylor.

Fermat's Last Theorem (FLT) [1]

Fermat's Last Theorem is a mathematical theorem that states that there are no three positive integers a, b, and c that satisfy the equation $a^n + b^n = c^n$ for any integer value of n greater than 2.

The theorem was first proposed by Pierre de Fermat around 1637 in the margin of a copy of Diophantus's Arithmetica. Fermat added that he had a proof that was too large to fit in the margin.

His claim was only discovered after his death, by his son.

It is widely believed that Fermat did not have a correct proof of the result at the time.

In 1994, around 350 years after Fermat's claim, Andrew Wiles announced a proof. However, it had a gap which he fixed a year later through joint work with Taylor.

The quest to prove Fermat's Last Theorem spurred the development of new areas in number theory and algebraic geometry.

Motivation

- Freek maintains a list (presumably compiled in the mid-1990s) of 100 theorems for formalizers. Out of these, 99 have been formalized and the last theorem on the list is Fermat's Last Theorem.

Motivation

- Freek maintains a list (presumably compiled in the mid-1990s) of 100 theorems for formalizers. Out of these, 99 have been formalized and the last theorem on the list is Fermat's Last Theorem.
- The technical achievement behind the formalization of the proof of Fermat's Last Theorem will not be just the verification of its validity.

Motivation

- Freek maintains a list (presumably compiled in the mid-1990s) of 100 theorems for formalizers. Out of these, 99 have been formalized and the last theorem on the list is Fermat's Last Theorem.

- The technical achievement behind the formalization of the proof of Fermat's Last Theorem will not be just the verification of its validity.

  It will drive the development of a library (`mathlib`) that can be used for modern research where areas of mathematics work concurrently.

Motivation

- Freek maintains a list (presumably compiled in the mid-1990s) of 100 theorems for formalizers. Out of these, 99 have been formalized and the last theorem on the list is Fermat's Last Theorem.
- The technical achievement behind the formalization of the proof of Fermat's Last Theorem will not be just the verification of its validity.

  It will drive the development of a library (`mathlib`) that can be used for modern research where areas of mathematics work concurrently.
- Formalizing the proof will identify any weaknesses in current proving technology.

Motivation

- Freek maintains a list (presumably compiled in the mid-1990s) of 100 theorems for formalizers. Out of these, 99 have been formalized and the last theorem on the list is Fermat's Last Theorem.

- The technical achievement behind the formalization of the proof of Fermat's Last Theorem will not be just the verification of its validity.

  It will drive the development of a library (`mathlib`) that can be used for modern research where areas of mathematics work concurrently.

- Formalizing the proof will identify any weaknesses in current proving technology.

- The definitions formalized in current and future work, will lay the groundwork for the formalization of results from state-of-the-art number theory such as the Langlands program.

Motivation

- Freek maintains a list (presumably compiled in the mid-1990s) of 100 theorems for formalizers. Out of these, 99 have been formalized and the last theorem on the list is Fermat's Last Theorem.

- The technical achievement behind the formalization of the proof of Fermat's Last Theorem will not be just the verification of its validity.

  It will drive the development of a library (`mathlib`) that can be used for modern research where areas of mathematics work concurrently.

- Formalizing the proof will identify any weaknesses in current proving technology.

- The definitions formalized in current and future work, will lay the groundwork for the formalization of results from state-of-the-art number theory such as the Langlands program.

  The deep theorems by Wiles and others can be summarised as special cases of the "2-dimensional case of the Langlands Philosophy".

Recent-ish News

In October 2023, Kevin Buzzard received an EPSRC research grant to begin working towards a formal proof of Fermat's Last Theorem in Lean.

Recent-ish News

In October 2023, Kevin Buzzard received an EPSRC research grant to begin working towards a formal proof of Fermat's Last Theorem in Lean.

The grant buys out his teaching and administration for 5 years starting October 2024.

Recent-ish News

In October 2023, Kevin Buzzard received an EPSRC research grant to begin working towards a formal proof of Fermat's Last Theorem in Lean.

The grant buys out his teaching and administration for 5 years starting October 2024.

In the announcement, he mentions that he will be working on a more modern version of the proof, which was created during discussions with Taylor and involves less analysis than the original Taylor-Wiles proof.

A non-exhaustive list of ingredients that need to be formalized

All known proofs involve a vast amount of technical machinery (358 years of mathematical knowledge).

A non-exhaustive list of ingredients that need to be formalized

All known proofs involve a vast amount of technical machinery (358 years of mathematical knowledge).

Note also that all known proofs of FLT use classical mathematics (axiom of choice and the law of excluded middle).

A non-exhaustive list of ingredients that need to be formalized

All known proofs involve a vast amount of technical machinery (358 years of mathematical knowledge).

Note also that all known proofs of FLT use classical mathematics (axiom of choice and the law of excluded middle).

Elliptic curves (Angdinata, Xu, . . .), modular forms (Birkbeck, . . .), finite flat group schemes, automorphic representations, p-adic Galois representations (Monnet, . . .), Hecke algebras, universal deformation rings, Galois cohomology (Livingston, . . .), local and global class field theory (De Frutos–Fernández, . . .), harmonic analysis, algebraic geometry, arithmetic geometry, nonabelian Fourier theory.

A non-exhaustive list of ingredients that need to be formalized

All known proofs involve a vast amount of technical machinery (358 years of mathematical knowledge).

Note also that all known proofs of FLT use classical mathematics (axiom of choice and the law of excluded middle).

Elliptic curves (Angdinata, Xu, ...), modular forms (Birkbeck, ...), finite flat group schemes, automorphic representations, p-adic Galois representations (Monnet, ...), Hecke algebras, universal deformation rings, Galois cohomology (Livingston, ...), local and global class field theory (De Frutos–Fernández, ...), harmonic analysis, algebraic geometry, arithmetic geometry, nonabelian Fourier theory.

Then we prove some very deep theorems about some of these objects, using the rest of these objects.

A non-exhaustive list of ingredients that need to be formalized

All known proofs involve a vast amount of technical machinery (358 years of mathematical knowledge).

Note also that all known proofs of FLT use classical mathematics (axiom of choice and the law of excluded middle).

Elliptic curves (Angdinata, Xu, . . .), modular forms (Birkbeck, . . .), finite flat group schemes, automorphic representations, p-adic Galois representations (Monnet, . . .), Hecke algebras, universal deformation rings, Galois cohomology (Livingston, . . .), local and global class field theory (De Frutos–Fernández, . . .), harmonic analysis, algebraic geometry, arithmetic geometry, nonabelian Fourier theory.

Then we prove some very deep theorems about some of these objects, using the rest of these objects.

And then we get a complete formalization of Fermat's Last Theorem.

The proof of FLT

Theorem (Fermat's Last Theorem, [5])

Let $n > 2$. If positive integers a, b, and c satisfy the equation

$$a^n + b^n = c^n \tag{1}$$

then at least one of a, b, and c must be 0.

A flow diagram ([5], p. 1) of the proof can be drawn as follows:

The proof of FLT

Theorem (Fermat's Last Theorem, [5])

Let $n > 2$. If positive integers a, b, and c satisfy the equation

$$a^n + b^n = c^n \tag{1}$$

then at least one of a, b, and c must be 0.

A flow diagram ([5], p. 1) of the proof can be drawn as follows:

$$\begin{aligned} \text{(a solution of (1))} &\implies \\ \text{(an elliptic curve)} &\implies \\ \text{(a Galois representation)} &\implies \\ \text{(a modular form)} &\implies \\ \text{(contradiction)} & \end{aligned} \tag{2}$$

The meaning of diagram (2) goes as follows. We assume there exists a nontrivial solution to the equation (1) and define an elliptic curve using such a solution. We then show that such an elliptic curve is associated to a modular form with certain properties. Finally, we derive a contradiction by showing that such a modular form could not exist.

Overview of the proof of FLT [1]

For an integer $n > 2$, assume the equation $a^n + b^n = c^n$ admits a nontrivial integer solution.

Overview of the proof of FLT [1]

For an integer $n > 2$, assume the equation $a^n + b^n = c^n$ admits a nontrivial integer solution.

Then we obtain an elliptic curve using such a solution using the work of Hellegouarch (from the late 1960s) and others.

Overview of the proof of FLT [1]

For an integer $n > 2$, assume the equation $a^n + b^n = c^n$ admits a nontrivial integer solution.

Then we obtain an elliptic curve using such a solution using the work of Hellegouarch (from the late 1960s) and others.

The curve consists of all points in the plane whose coordinates (x, y) satisfy the relation

$$y^2 = x(x - a^n)(x + b^n). \tag{3}$$

The next step is to construct an irreducible Galois representation associated to this elliptic curve.

Overview of the proof of FLT [1]

For an integer $n > 2$, assume the equation $a^n + b^n = c^n$ admits a nontrivial integer solution.

Then we obtain an elliptic curve using such a solution using the work of Hellegouarch (from the late 1960s) and others.

The curve consists of all points in the plane whose coordinates (x, y) satisfy the relation

$$y^2 = x(x - a^n)(x + b^n). \tag{3}$$

The next step is to construct an irreducible Galois representation associated to this elliptic curve.

Galois representations can simplify the study of elliptic curves by transforming the field of study into a simpler one, such as linear algebra.

Overview of the proof of FLT [1]

For an integer $n > 2$, assume the equation $a^n + b^n = c^n$ admits a nontrivial integer solution.

Then we obtain an elliptic curve using such a solution using the work of Hellegouarch (from the late 1960s) and others.

The curve consists of all points in the plane whose coordinates (x, y) satisfy the relation

$$y^2 = x(x - a^n)(x + b^n). \tag{3}$$

The next step is to construct an irreducible Galois representation associated to this elliptic curve.

Galois representations can simplify the study of elliptic curves by transforming the field of study into a simpler one, such as linear algebra.

For this, we rely on a deep theorem by Mazur from the 1970s that involves advanced algebraic geometry.

Overview of the proof of FLT [1]

For an integer $n > 2$, assume the equation $a^n + b^n = c^n$ admits a nontrivial integer solution.

Then we obtain an elliptic curve using such a solution using the work of Hellegouarch (from the late 1960s) and others.

The curve consists of all points in the plane whose coordinates (x, y) satisfy the relation

$$y^2 = x(x - a^n)(x + b^n). \tag{3}$$

The next step is to construct an irreducible Galois representation associated to this elliptic curve.

Galois representations can simplify the study of elliptic curves by transforming the field of study into a simpler one, such as linear algebra.

For this, we rely on a deep theorem by Mazur from the 1970s that involves advanced algebraic geometry.

Overview of the proof of FLT (continued)

Next step : from the Galois representation, we associate a "modular form" to it using some very profound theorems of Langlands and others on analysis.

Overview of the proof of FLT (continued)

Next step : from the Galois representation, we associate a "modular form" to it using some very profound theorems of Langlands and others on analysis.

Modular forms can be viewed as functions from the complex upper half plane $\mathbb{H} = \{z \in \mathbb{C} \mid \mathrm{Im}\,(z) > 0\}$ to $\mathbb{C}$. A definition of such a function is the following:

Definition (Modular form)

A modular form of level N is a formal power series of the form

$$f = \sum_{m=1}^{\infty} a_m x^m \in \mathbb{C}[\![x]\!] \tag{4}$$

satisfying certain properties.

Given the associated modular form, we now apply deep results of Wiles and others and get a contradiction.

Xena project [2], February 9, 2020

The formalization of the *statement* of Deligne's theorem is the 6th of the ten challenges proposed by Buzzard for testing the limits of the current proof assistants - Lean, Coq, Isabelle/HOL, Mizar and all the others - by identifying which prover can do all ten.

Xena project [2], February 9, 2020

The formalization of the *statement* of Deligne's theorem is the 6th of the ten challenges proposed by Buzzard for testing the limits of the current proof assistants - Lean, Coq, Isabelle/HOL, Mizar and all the others - by identifying which prover can do all ten.

Freek Wiedijk keeps track of 100 theorems to be formalised — but 95 of them are done now and FLT is just silly. We need new challenges. Here are ten off the top of my head:

1. *Formalise the statement of the Riemann Hypothesis.*
2. *Formalise the statement of the Poincare conjecture.*
3. *Formalise the definition of an algebraic stack.*
4. *Formalise the definition of a reductive algebraic group.*
5. *Formalise the definition of an adic space.*
6. *State Deligne's theorem attaching a Galois representation to a weight k eigenform.*
7. *Do the sheaf-gluing exercise in Hartshorne (chapter 2, exercise 1.22).*
8. *Prove sphere eversion.*
9. *Do exercise 1.1.i in Elements of infinity-category theory by Riehl and Verity (note that infinity categories are used in section 5 of Scholze's new paper with Cesnavicius so they're probably here to stay).*
10. *Define singular cohomology of a topological space.*

The challenge is accompanied with the comment "6 is I think a million miles away from anything in any theorem prover".

Deligne's theorem

The formalization of Deligne's theorem requires the combination of results from several mathematical areas, including analysis, algebra, number theory, topology and representation theory, and fits in within the long-term goal of formalizing a proof of Fermat's Last Theorem.

Deligne's theorem

The formalization of Deligne's theorem requires the combination of results from several mathematical areas, including analysis, algebra, number theory, topology and representation theory, and fits in within the long-term goal of formalizing a proof of Fermat's Last Theorem.

Theorem (Deligne [3])

If f is a weight k modular eigenform of level N and character χ, and if p is a prime then there is an associated 2-dimensional Galois representation ρ_f unramified outside Np such that if q not dividing Np is a prime, then the characteristic polynomial of $\rho_f(\mathrm{Frob})$ is $X^2 - a_q X + q^{k-1}\chi(q)$.

The case $k = 2$, historically proven earlier by Eichler and Shimura, constitutes a necessary component in the Wiles/Taylor–Wiles proof.

Deligne's theorem

The formalization of Deligne's theorem requires the combination of results from several mathematical areas, including analysis, algebra, number theory, topology and representation theory, and fits in within the long-term goal of formalizing a proof of Fermat's Last Theorem.

Theorem (Deligne [3])

If f is a weight k modular eigenform of level N and character χ, and if p is a prime then there is an associated 2-dimensional Galois representation ρ_f unramified outside Np such that if q not dividing Np is a prime, then the characteristic polynomial of $\rho_f(\mathrm{Frob})$ is $X^2 - a_q X + q^{k-1}\chi(q)$.

The case $k = 2$, historically proven earlier by Eichler and Shimura, constitutes a necessary component in the Wiles/Taylor–Wiles proof.

Formalizing the statement of Deligne's theorem involves the formalization of the definitions that occur in this statement and play a central role in all the proofs of Fermat's Last theorem.

Formal proof sketch

Compared to the aforementioned work, our approach is top-down; The idea is to create a roadmap for a full formalization.

Formal proof sketch

Compared to the aforementioned work, our approach is top-down; The idea is to create a roadmap for a full formalization.

A Formal Proof Sketch is a formalization in which details of proofs have been omitted, but still has a notion of correctness.

Formal proof sketch

Compared to the aforementioned work, our approach is top-down; The idea is to create a roadmap for a full formalization.

A Formal Proof Sketch is a formalization in which details of proofs have been omitted, but still has a notion of correctness.

We will introduce the higher-level objects used, and we will state, but not prove, the theorems of Wiles and Taylor/Wiles and others about these objects.

Formal proof sketch

Compared to the aforementioned work, our approach is top-down; The idea is to create a roadmap for a full formalization.

A Formal Proof Sketch is a formalization in which details of proofs have been omitted, but still has a notion of correctness.

We will introduce the higher-level objects used, and we will state, but not prove, the theorems of Wiles and Taylor/Wiles and others about these objects.

We want to reduce the proof to several highly nontrivial statements about these 20th century mathematical objects.

The Lean theorem prover [4]

Lean 4 is a programming language together with a proof assistant.

The Lean theorem prover [4]

Lean 4 is a programming language together with a proof assistant.

Lean's formal system is a dependent type theory based on the calculus of inductive constructions.

The Lean theorem prover [4]

Lean 4 is a programming language together with a proof assistant.

Lean's formal system is a dependent type theory based on the calculus of inductive constructions.

Rocq and Lean are based on essentially the same type-theoretic foundations.

The Lean theorem prover [4]

Lean 4 is a programming language together with a proof assistant.

Lean's formal system is a dependent type theory based on the calculus of inductive constructions.

Rocq and Lean are based on essentially the same type-theoretic foundations.

Each element has a unique type, which is represented as `e : t`. For example, the natural number 0 has type $\mathbb{N}$, and the rational 0 has type $\mathbb{Q}$. Types also have types, such as $\mathbb{N}$ `: Type`.

The Lean theorem prover [4]

Lean 4 is a programming language together with a proof assistant.

Lean's formal system is a dependent type theory based on the calculus of inductive constructions.

Rocq and Lean are based on essentially the same type-theoretic foundations.

Each element has a unique type, which is represented as `e : t`. For example, the natural number 0 has type $\mathbb{N}$, and the rational 0 has type $\mathbb{Q}$. Types also have types, such as $\mathbb{N}$ `: Type`.

The hierarchy consists of an impredicative universe `Prop` at the bottom of a noncumulative chain `Prop : Type : Type 1 : Type 2 : ...;`.

The Lean theorem prover [4]

Lean 4 is a programming language together with a proof assistant.

Lean's formal system is a dependent type theory based on the calculus of inductive constructions.

Rocq and Lean are based on essentially the same type-theoretic foundations.

Each element has a unique type, which is represented as `e : t`. For example, the natural number `0` has type $\mathbb{N}$, and the rational `0` has type $\mathbb{Q}$. Types also have types, such as $\mathbb{N}$ `: Type`.

The hierarchy consists of an impredicative universe `Prop` at the bottom of a noncumulative chain `Prop : Type : Type 1 : Type 2 : ...;`.

Propositions correspond to elements of `Prop`, while a (verified) proof of the proposition `P : Prop` corresponds to an element `p : P`.

The Lean theorem prover [4]

Lean 4 is a programming language together with a proof assistant.

Lean's formal system is a dependent type theory based on the calculus of inductive constructions.

Rocq and Lean are based on essentially the same type-theoretic foundations.

Each element has a unique type, which is represented as `e : t`. For example, the natural number `0` has type $\mathbb{N}$, and the rational `0` has type $\mathbb{Q}$. Types also have types, such as $\mathbb{N}$ `: Type`.

The hierarchy consists of an impredicative universe `Prop` at the bottom of a noncumulative chain `Prop : Type : Type 1 : Type 2 : ...;`.

Propositions correspond to elements of `Prop`, while a (verified) proof of the proposition `P : Prop` corresponds to an element `p : P`.

Lean commands I

The `def` and its variant `abbrev` commands are used to define new objects.

Lean commands I

The `def` and its variant `abbrev` commands are used to define new objects.

The general form of a definition is `def foo :` α `:= bar`, where α is the type of the object being defined and `bar` is the term that `foo` should denote.

Lean commands I

The `def` and its variant `abbrev` commands are used to define new objects.

The general form of a definition is `def foo : α := bar`, where α is the type of the object being defined and `bar` is the term that `foo` should denote.

It is often a good idea to write the type explicitly to clarify the intention and avoid errors. For example,

```
def Double (x : ℕ) : ℕ := x + x
```

The name `Double` is defined as a function that takes an input parameter `x` of type $\mathbb{N}$, where the output is `x + x` of type $\mathbb{N}$.

Lean commands II

Similarly, the `theorem` command introduces a new theorem:
`theorem t : A := ...` with the only difference being that `A` has type `Prop` i.e. `A` is a proposition.

Lean commands II

Similarly, the `theorem` command introduces a new theorem:
`theorem t : A := ...` with the only difference being that `A` has type `Prop` i.e. `A` is a proposition.

For example,

```
theorem t (P Q : Prop) : P → Q → P :=
  fun hp : P => fun hq : Q => hp
```

Thus, proving the theorem `P → Q → P` is the same as defining an element of the associated type.
Lastly, the `sorry` identifier is a command that produces a proof of anything or provides an object of any data type.

Mathlib

Lean has a comprehensive mathematics library called `mathlib` that covers the foundations of analysis, geometry, algebra, topology, and number theory.

Mathlib

Lean has a comprehensive mathematics library called `mathlib` that covers the foundations of analysis, geometry, algebra, topology, and number theory.

The library is a community-driven effort and is actively maintained by many research mathematicians who are familiar with the material.

Mathlib

Lean has a comprehensive mathematics library called `mathlib` that covers the foundations of analysis, geometry, algebra, topology, and number theory.

The library is a community-driven effort and is actively maintained by many research mathematicians who are familiar with the material.

According to the `mathlib` philosophy, the formalized work has to be written in the maximal generality and not in an ad hoc way, to ensure that it can be used in as many different applications as possible.

Large-scale formalizations

Large scale formalizations are difficult to achieve.

Large-scale formalizations

Large scale formalizations are difficult to achieve.

The complete formalization of FLT will take a very long time and it will require the sustained efforts of a large number of specialists in many diverse fields.

Large-scale formalizations

Large scale formalizations are difficult to achieve.

The complete formalization of FLT will take a very long time and it will require the sustained efforts of a large number of specialists in many diverse fields.

Furthermore, building upon a large library is the only way to ensure continuous maintenance of the code.

Large-scale formalizations

Large scale formalizations are difficult to achieve.

The complete formalization of FLT will take a very long time and it will require the sustained efforts of a large number of specialists in many diverse fields.

Furthermore, building upon a large library is the only way to ensure continuous maintenance of the code.

New material is initially not ready for use outside of the project as it may undergo significant modifications to suit the needs of the project.

Large-scale formalizations

Large scale formalizations are difficult to achieve.

The complete formalization of FLT will take a very long time and it will require the sustained efforts of a large number of specialists in many diverse fields.

Furthermore, building upon a large library is the only way to ensure continuous maintenance of the code.

New material is initially not ready for use outside of the project as it may undergo significant modifications to suit the needs of the project.

Building on top of a library requires developing it in parallel, and, this is only possible when the code is sufficiently mature and written in the maximal generality.

More on FPS

The first step towards this is to write the proof from the top down, using `sorry` to fill in sub-proofs.

More on FPS

The first step towards this is to write the proof from the top down, using `sorry` to fill in sub-proofs.

Once a proof sketch has been obtained, the proof will be built incrementally by filling in the sub-proofs.

More on FPS

The first step towards this is to write the proof from the top down, using `sorry` to fill in sub-proofs.

Once a proof sketch has been obtained, the proof will be built incrementally by filling in the sub-proofs.

The end product will reveal all the required material and deficiencies of the library that need to be formalized. This will also keep the maintenance of the code to a bare minimum.

More on FPS

The first step towards this is to write the proof from the top down, using `sorry` to fill in sub-proofs.

Once a proof sketch has been obtained, the proof will be built incrementally by filling in the sub-proofs.

The end product will reveal all the required material and deficiencies of the library that need to be formalized. This will also keep the maintenance of the code to a bare minimum.

It will drive the development of `mathlib` by contributing pieces of work as they are finished; continuous maintenance, extensibility and usability of the code.

More on FPS

The first step towards this is to write the proof from the top down, using `sorry` to fill in sub-proofs.

Once a proof sketch has been obtained, the proof will be built incrementally by filling in the sub-proofs.

The end product will reveal all the required material and deficiencies of the library that need to be formalized. This will also keep the maintenance of the code to a bare minimum.

It will drive the development of `mathlib` by contributing pieces of work as they are finished; continuous maintenance, extensibility and usability of the code.

Last but not least, having a “blueprint” of the proof will enable the division and distribution of the work.

More on FPS

The first step towards this is to write the proof from the top down, using `sorry` to fill in sub-proofs.

Once a proof sketch has been obtained, the proof will be built incrementally by filling in the sub-proofs.

The end product will reveal all the required material and deficiencies of the library that need to be formalized. This will also keep the maintenance of the code to a bare minimum.

It will drive the development of `mathlib` by contributing pieces of work as they are finished; continuous maintenance, extensibility and usability of the code.

Last but not least, having a "blueprint" of the proof will enable the division and distribution of the work.

This approach will also help identify any potential issues or challenges that may arise.

Levels of formalization

There is a lattice of seven possibilities of which the first two are full formalizations, and the latter five are only partial approximations.

1. We prove the theorem as well as "`mathlib` versions".

Levels of formalization

There is a lattice of seven possibilities of which the first two are full formalizations, and the latter five are only partial approximations.

1. We prove the theorem as well as "`mathlib` versions".

2. The theorem is fully formalized with all definitions involved and including all proofs.

Levels of formalization

There is a lattice of seven possibilities of which the first two are full formalizations, and the latter five are only partial approximations.

1. We prove the theorem as well as "`mathlib` versions".

2. The theorem is fully formalized with all definitions involved and including all proofs.

3. Only the *statement* is fully formalized.

Levels of formalization

There is a lattice of seven possibilities of which the first two are full formalizations, and the latter five are only partial approximations.

1. We prove the theorem as well as "`mathlib` versions".
2. The theorem is fully formalized with all definitions involved and including all proofs.
3. Only the *statement* is fully formalized.
4. a The statement is fully formalized with all definitions involved but `Props` may be omitted everywhere.

Levels of formalization

There is a lattice of seven possibilities of which the first two are full formalizations, and the latter five are only partial approximations.

1. We prove the theorem as well as "`mathlib` versions".
2. The theorem is fully formalized with all definitions involved and including all proofs.
3. Only the *statement* is fully formalized.
4.
   a The statement is fully formalized with all definitions involved but `Props` may be omitted everywhere.

   b The statement is fully formalized, but with definitions that are "`sorry`-free", but are sometimes unusual, "hacky", to work around difficult proof obligations.

Levels of formalization

There is a lattice of seven possibilities of which the first two are full formalizations, and the latter five are only partial approximations.

1. We prove the theorem as well as "`mathlib` versions".
2. The theorem is fully formalized with all definitions involved and including all proofs.
3. Only the *statement* is fully formalized.
4.
   a The statement is fully formalized with all definitions involved but `Props` may be omitted everywhere.

   b The statement is fully formalized, but with definitions that are "`sorry`-free", but are sometimes unusual, "hacky", to work around difficult proof obligations.

   c A combination of 4a and 4b.

Levels of formalization

There is a lattice of seven possibilities of which the first two are full formalizations, and the latter five are only partial approximations.

1. We prove the theorem as well as "`mathlib` versions".

2. The theorem is fully formalized with all definitions involved and including all proofs.

3. Only the *statement* is fully formalized.

4.
   a The statement is fully formalized with all definitions involved but `Props` may be omitted everywhere.

   b The statement is fully formalized, but with definitions that are "`sorry`-free", but are sometimes unusual, "hacky", to work around difficult proof obligations.

   c A combination of 4a and 4b.

5. The statement is fully formalized and both definitions and Props may be "sorried".

Lattice of partial formalization

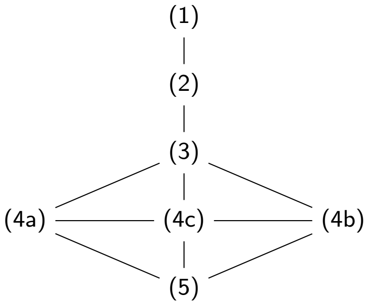

Figure: The lattice of notions of partial formalization.

The interpretation of (4a), (4b) and (4c) is that we are not allowed to say `def foo : a := sorry` if `a : Type` but we are allowed to say `theorem t : P := sorry` if `P : Prop`.

Formalizing FLT

Currently, while it is very challenging to achieve a complete formalization of many of the high-powered theorems used in the proof of Fermat's Last Theorem, even formalizing just the statements of these theorems without omitting any proofs may require substantial effort and time, potentially spanning decades.

Formalizing FLT

Currently, while it is very challenging to achieve a complete formalization of many of the high-powered theorems used in the proof of Fermat's Last Theorem, even formalizing just the statements of these theorems without omitting any proofs may require substantial effort and time, potentially spanning decades.

Thus, we are going to be in situations where (1), (2) and (3) are not feasible, leaving us with the variants (4a), (4b), and (4c) as our only options.

Formalizing FLT

Currently, while it is very challenging to achieve a complete formalization of many of the high-powered theorems used in the proof of Fermat's Last Theorem, even formalizing just the statements of these theorems without omitting any proofs may require substantial effort and time, potentially spanning decades.

Thus, we are going to be in situations where (1), (2) and (3) are not feasible, leaving us with the variants (4a), (4b), and (4c) as our only options.

Formal sketch of Deligne's theorem

Thus, the situation is similar. The mathematics involved in the theorem of Deligne will take a long time to fully formalize.

Formal sketch of Deligne's theorem

Thus, the situation is similar. The mathematics involved in the theorem of Deligne will take a long time to fully formalize.

Our approach will be to work on level (4c) of partial formalization. This means that we are allowed to `sorry` `Props` but not definitions - the definitions can be stated in a general, ad hoc, and/or unorthodox way.

Formal sketch of Deligne's theorem

Thus, the situation is similar. The mathematics involved in the theorem of Deligne will take a long time to fully formalize.

Our approach will be to work on level (4c) of partial formalization. This means that we are allowed to `sorry` `Props` but not definitions - the definitions can be stated in a general, ad hoc, and/or unorthodox way.

As a result, we have defined all mathematical objects in the statement of Deligne's theorem.

Eigenforms for Deligne's theorem

The classical theory of modular forms has been integrated in `mathlib`.

Eigenforms for Deligne's theorem

The classical theory of modular forms has been integrated in `mathlib`.

Among the most important modular forms are what we call eigenforms.

Eigenforms for Deligne's theorem

The classical theory of modular forms has been integrated in `mathlib`.

Among the most important modular forms are what we call eigenforms.

The following code snippet shows how this definition looks in Lean:

```
def IsWeakEigenform {N : ℕ} {k : ℕ}
  (f : ModularForm (Gamma1 N) k)
  (χ : DirChar ℂ N) : Prop :=
f ≠ 0 ∧ ∀ {q : ℕ} (h : Nat.Prime q) (hqN : ¬q ∣ N), ∃ a : ℂ,
∀ z : UpperHalfPlane, heckeOperator f h χ hqN z = a * f z
```

Galois representations for Deligne's theorem

A representation makes an abstract algebraic object more concrete by describing its elements by matrices and their algebraic operations (for example, matrix addition, matrix multiplication).

Galois representations for Deligne's theorem

A representation makes an abstract algebraic object more concrete by describing its elements by matrices and their algebraic operations (for example, matrix addition, matrix multiplication).

For Deligne's theorem we need a two-dimensional Galois representation over a topological ring k which is a continuous group homomorphism

$$\rho : G_{\mathbb{Q}} \to \mathrm{GL}_2(k).$$

```
def GaloisRep := ContinuousMonoidHom
(AlgebraicClosure ℚ ≃ₐ[ℚ] AlgebraicClosure ℚ) (GL (Fin 2) k)
```

The Frobenius element in Deligne's theorem

The Frobenius element Frob is a key element of the Galois group $G_{\mathbb{Q}}$ with some special properties.

The Frobenius element in Deligne's theorem

The Frobenius element Frob is a key element of the Galois group $G_{\mathbb{Q}}$ with some special properties.

The formalization of this special element in Lean looks like this:

```
noncomputable def Frob (K L : Type _) [Field K] [Field L]
  [NumberField K] [Algebra K L] [IsGalois K L]
  (v : ValuationSubring L) (hv : v ≠ ⊤) :
decompositionSubgroup K v := by
   letI := fintypeOfNeBot K (v.comap (algebraMap K L)) (
     ComapNeTopOfAlgebraic K v hv Normal.isAlgebraic')
  have := decompositionSubgroup.surjective K v
  let f : LocalRing.ResidueField v ≃ₐ[LocalRing.ResidueField (v.comap (
     algebraMap K L))]
      LocalRing.ResidueField v :=
      frobenius.equiv
      (algebraComap_algebraic K v)
  specialize this f
  exact this.choose
```

Formalizing Deligne's theorem

The following Lean code states Deligne's theorem with comments.

```
-- If f is a weight k modular eigenform of level N and character χ, and if p is a
     prime
theorem Deligne {N : ℕ} {k : ℕ}
  {f : ModularForm (Gamma1 N) k}
  (χ : DirChar (AlgebraicClosure ℚ) N)
  (hf : IsWeakEigenform f (DirCharComplex χ)) :
-- then there is an associated 2-dimensional Galois representation ρ
∃ (ρ : GaloisRep (AlgebraicClosure ℚ_[p])),
-- unramified outside Np such that if q not dividing Np, q is a prime
∀ (v : ValuationSubring (AlgebraicClosure ℚ))
  (hv : v ≠ ⊤)
  (hqpN : ¬ q v hv ∣ p * N),
  (IsUnramified ρ v) ∧
let a :=
  C ((algClosRatToPAdic p)
    (AlgEigenvalue (q.isPrime hv) hf
    (div N v hv hqpN)))
let χq  :=
 (Units.coeHom (AlgebraicClosure ℚ_[p])).comp
 (DirCharAlgClosRat χ)
 (ZMod.Unit (q.isPrime hv) hqpN)
-- Then the characteristic polynomial of ρ(Frob)
Matrix.charpoly (Matrix.of
(ρ.toMonoidHom (Frob ℚ (AlgebraicClosure ℚ) v hv))) =
-- is X^2 - a_qX + q^{k-1} χ(q).
X ^ 2 - (a * X) + ((q v hv) ^ (k - 1) :
(AlgebraicClosure ℚ_[p])[X]) * (C χq) := sorry
```

Conclusion and future work

We stated Deligne's theorem attaching a Galois representation to a weight k eigenform by implementing the (4c) level of formalization.

Conclusion and future work

We stated Deligne's theorem attaching a Galois representation to a weight k eigenform by implementing the (4c) level of formalization.

We demonstrated a procedure of constructing a blueprint and achieving a partial formalization by stating (but not proving) Deligne's theorem.

Conclusion and future work

We stated Deligne's theorem attaching a Galois representation to a weight k eigenform by implementing the (4c) level of formalization.

We demonstrated a procedure of constructing a blueprint and achieving a partial formalization by stating (but not proving) Deligne's theorem.

One can now build upon this work to achieve a complete formalization of Deligne's theorem.

Conclusion and future work

We stated Deligne's theorem attaching a Galois representation to a weight k eigenform by implementing the (4c) level of formalization.

We demonstrated a procedure of constructing a blueprint and achieving a partial formalization by stating (but not proving) Deligne's theorem.

One can now build upon this work to achieve a complete formalization of Deligne's theorem.

Our future work involves applying the same process to obtain partial formalizations of the main theorems in the Wiles and Taylor–Wiles proof of Fermat's Last Theorem.

References I

[1] Kevin Buzzard. *Formalizing Fermat*. Presented at AITP, Aussois, 7 September 2022. PowerPoint slides. 2022.

[2] Kevin Buzzard. *Lean is better for proper maths than all the other theorem provers*. https://xenaproject.wordpress.com/2020/02/09/lean-is-better-for-proper-maths-than-all-the-other-theorem-provers/. 2020. (Visited on 02/05/2023).

[3] Pierre Deligne and Jean-Pierre Serre. "Formes modulaires de poids 1". In: *Annales scientifiques de l'École Normale Supérieure*. Vol. 7. 1974, pp. 507–530.

[4] Leonardo de Moura et al. "The Lean Theorem Prover (System Description)". In: *Automated Deduction - CADE-25*. Ed. by Amy P. Felty and Aart Middeldorp. Cham: Springer International Publishing, 2015, pp. 378–388. ISBN: 978-3-319-21401-6.

[5] Takeshi Saito. *Fermat's Last Theorem: Basic Tools*. American Mathematical Society Providence, 2013.

The End