Young Researchers in Algebraic Number Theory

Wednesday, 24 August 2022

Formalisation of elliptic curves in Lean

David Kurniadi Angdinata

London School of Geometry and Number Theory

The Lean theorem prover

LEAN

THEOREM PROVER

The Lean theorem prover

A functional programming language...

The Lean theorem prover

A functional programming language...

and an interactive theorem prover!

Programming in Lean

Idea: *set theory* is replaced by `Type Theory`.

$$\textit{element} \in \textit{set} \quad \Longrightarrow \quad \texttt{Term} \; : \; \texttt{Type}$$

Programming in Lean

Idea: *set theory* is replaced by `Type Theory`.

$$\textit{element} \in \textit{set} \quad \Longrightarrow \quad \texttt{Term} \ : \ \texttt{Type}$$

Can define inductive types.

```
inductive Nat
  | zero : Nat
  | succ : Nat → Nat
```

Programming in Lean

Idea: *set theory* is replaced by `Type Theory`.

$$\textit{element} \in \textit{set} \quad \Longrightarrow \quad \texttt{Term} : \texttt{Type}$$

Can define inductive types.

```
inductive Nat
  | zero : Nat
  | succ : Nat → Nat
```

Can define functions recursively.

```
def add : Nat → Nat → Nat
  | n zero := n
  | n (succ m) := succ (add n m)
```

Programming in Lean

How to prove $\forall n \in \mathbb{N},\ 0 + n = n$?

Programming in Lean

How to prove $\forall n \in \mathbb{N},\ 0 + n = n$?

A theorem is a `Type` (of type `Prop`).

```
theorem zero_add : ∀ (n : Nat), add zero n = n :=
```

Programming in Lean

How to prove $\forall n \in \mathbb{N},\ 0 + n = n$?

A theorem is a `Type` (of type `Prop`).

```
theorem zero_add : ∀ (n : Nat), add zero n = n :=
```

A proof of this theorem (if it exists) is the unique `Term` of this type.

Programming in Lean

How to prove $\forall n \in \mathbb{N},\ 0 + n = n$?

A theorem is a `Type` (of type `Prop`).

```
theorem zero_add : ∀ (n : Nat), add zero n = n :=
```

A proof of this theorem (if it exists) is the unique `Term` of this type.

```
begin
  intro n,
  induction n with m hm,
  { refl },
  { rw [add, hm] }
end
```

The keywords `intro`, `induction`, `refl`, and `rw` are **tactics**.

Programming in Lean

How to prove $\forall n \in \mathbb{N},\ 0 + n = n$?

A theorem is a `Type` (of type `Prop`).

```
theorem zero_add : ∀ (n : Nat), add zero n = n :=
```

A proof of this theorem (if it exists) is the unique `Term` of this type.

```
begin
  intro n,
  induction n with m hm,
  { refl },
  { rw [add, hm] }
end
```

The keywords `intro`, `induction`, `refl`, and `rw` are **tactics**.

Play **The Natural Number Game**!

Lean's mathematical library `mathlib`

Community-driven unified library of mathematics formalised in Lean.

Lean's mathematical library `mathlib`

Community-driven unified library of mathematics formalised in Lean.

- algebra
- algebraic_geometry
- algebraic_topology
- analysis
- category_theory
- combinatorics
- computability
- dynamics
- field_theory
- geometry
- group_theory
- information_theory
- linear_algebra
- measure_theory
- model_theory
- number_theory
- order
- probability
- representation_theory
- ring_theory
- set_theory
- topology

Lean's mathematical library `mathlib`

Community-driven unified library of mathematics formalised in Lean.

- algebra
- algebraic_geometry
- algebraic_topology
- analysis
- category_theory
- combinatorics
- computability
- dynamics
- field_theory
- geometry
- group_theory
- information_theory
- linear_algebra
- measure_theory
- model_theory
- number_theory
- order
- probability
- representation_theory
- ring_theory
- set_theory
- topology

3k files, 1m lines, 40k definitions, 100k theorems, 270 contributors.

Lean's mathematical library mathlib

Consider the following theorem in group_theory/quotient_group.

```
variables {G H : Type} [group G] [group H]
variables (φ : G →* H) (ψ : H → G) (hφ : right_inverse ψ φ)

def quotient_ker_equiv_of_right_inverse : G / ker φ ≃* H :=
 { to_fun := ker_lift φ,
   inv_fun := mk ∘ ψ,
   left_inv := ...,
   right_inv := hφ,
   map_mul' := ... }
```

Lean's mathematical library mathlib

Consider the following theorem in group_theory/quotient_group.

```
variables {G H : Type} [group G] [group H]
variables (φ : G →* H) (ψ : H → G) (hφ : right_inverse ψ φ)

def quotient_ker_equiv_of_right_inverse : G / ker φ ≃* H :=
 { to_fun := ker_lift φ,
   inv_fun := mk ∘ ψ,
   left_inv := ...,
   right_inv := hφ,
   map_mul' := ... }
```

Why is this a definition?

Lean's mathematical library mathlib

Consider the following theorem in group_theory/quotient_group.

```
variables {G H : Type} [group G] [group H]
variables (φ : G →* H) (ψ : H → G) (hφ : right_inverse ψ φ)

def quotient_ker_equiv_of_right_inverse : G / ker φ ≃* H :=
 { to_fun := ker_lift φ,
   inv_fun := mk ∘ ψ,
   left_inv := ...,
   right_inv := hφ,
   map_mul' := ... }
```

Why is this a definition?

Consider an immediate corollary.

```
def quotient_bot : G / (⊥ : subgroup G) ≃* G :=
 quotient_ker_equiv_of_right_inverse (monoid_hom.id G) id (λ _, rfl)
```

Lean's mathematical library mathlib

Consider the following theorem in group_theory/quotient_group.

```
variables {G H : Type} [group G] [group H]
variables (φ : G →* H) (ψ : H → G) (hφ : right_inverse ψ φ)

def quotient_ker_equiv_of_right_inverse : G / ker φ ≃* H :=
 { to_fun := ker_lift φ,
   inv_fun := mk ∘ ψ,
   left_inv := ...,
   right_inv := hφ,
   map_mul' := ... }
```

Why is this a definition?

Consider an immediate corollary.

```
def quotient_bot : G / (⊥ : subgroup G) ≃* G :=
 quotient_ker_equiv_of_right_inverse (monoid_hom.id G) id (λ _, rfl)
```

Why is this not trivial?

Lean's mathematical library mathlib

Consider the following theorem in group_theory/quotient_group.

```
variables {G H : Type} [group G] [group H]
variables (φ : G →* H) (ψ : H → G) (hφ : right_inverse ψ φ)

def quotient_ker_equiv_of_right_inverse : G / ker φ ≃* H :=
 { to_fun := ker_lift φ,
   inv_fun := mk ∘ ψ,
   left_inv := ...,
   right_inv := hφ,
   map_mul' := ... }
```

Why is this a definition?

Consider an immediate corollary.

```
def quotient_bot : G / (⊥ : subgroup G) ≃* G :=
 quotient_ker_equiv_of_right_inverse (monoid_hom.id G) id (λ _, rfl)
```

Why is this not trivial?

Canonical isomorphisms are important data!

Elliptic curves in Lean

What generality?

Elliptic curves in Lean

What generality? Ideally, defined abstractly over a scheme or a ring...

Elliptic curves in Lean

What generality? Ideally, defined abstractly over a scheme or a ring...
However, `mathlib`'s algebraic geometry is still quite primitive.

Elliptic curves in Lean

What generality? Ideally, defined abstractly over a scheme or a ring...
However, mathlib's algebraic geometry is still quite primitive.

Here is a working definition in algebraic_geometry/EllipticCurve.

```
def Δ_aux {R : Type} [comm_ring R] (a₁ a₂ a₃ a₄ a₆ : R) : R :=
 let
  b₂ := a₁^2 + 4*a₂,
  b₄ := 2*a₄ + a₁*a₃,
  b₆ := a₃^2 + 4*a₆,
  b₈ := a₁^2*a₆ + 4*a₂*a₆ - a₁*a₃*a₄ + a₂*a₃^2 - a₄^2
 in
  -b₂^2*b₈ - 8*b₄^3 - 27*b₆^2 + 9*b₂*b₄*b₆

structure EllipticCurve (R : Type) [comm_ring R] :=
 (a₁ a₂ a₃ a₄ a₆ : R) (Δ : units R) (Δ_eq : ↑Δ = Δ_aux a₁ a₂ a₃ a₄ a₆)
```

Elliptic curves in Lean

What generality? Ideally, defined abstractly over a scheme or a ring...
However, `mathlib`'s algebraic geometry is still quite primitive.

Here is a working definition in `algebraic_geometry/EllipticCurve`.

```
def Δ_aux {R : Type} [comm_ring R] (a₁ a₂ a₃ a₄ a₆ : R) : R :=
 let
  b₂ := a₁^2 + 4*a₂,
  b₄ := 2*a₄ + a₁*a₃,
  b₆ := a₃^2 + 4*a₆,
  b₈ := a₁^2*a₆ + 4*a₂*a₆ - a₁*a₃*a₄ + a₂*a₃^2 - a₄^2
 in
  -b₂^2*b₈ - 8*b₄^3 - 27*b₆^2 + 9*b₂*b₄*b₆

structure EllipticCurve (R : Type) [comm_ring R] :=
 (a₁ a₂ a₃ a₄ a₆ : R) (Δ : units R) (Δ_eq : ↑Δ = Δ_aux a₁ a₂ a₃ a₄ a₆)
```

Accurate for rings R with $\mathrm{Pic}(R)[12] = 0$, such as PIDs!

Elliptic curves in Lean

What generality? Ideally, defined abstractly over a scheme or a ring...
However, mathlib's algebraic geometry is still quite primitive.

Here is a working definition in algebraic_geometry/EllipticCurve.

```
def Δ_aux {R : Type} [comm_ring R] (a₁ a₂ a₃ a₄ a₆ : R) : R :=
 let
  b₂ := a₁^2 + 4*a₂,
  b₄ := 2*a₄ + a₁*a₃,
  b₆ := a₃^2 + 4*a₆,
  b₈ := a₁^2*a₆ + 4*a₂*a₆ - a₁*a₃*a₄ + a₂*a₃^2 - a₄^2
 in
  -b₂^2*b₈ - 8*b₄^3 - 27*b₆^2 + 9*b₂*b₄*b₆

structure EllipticCurve (R : Type) [comm_ring R] :=
 (a₁ a₂ a₃ a₄ a₆ : R) (Δ : units R) (Δ_eq : ↑Δ = Δ_aux a₁ a₂ a₃ a₄ a₆)
```

Accurate for rings R with $\mathrm{Pic}(R)[12] = 0$, such as PIDs!

Much can be done just with this definition.

Elliptic curves in Lean

Can define K-points.

```
variables {F : Type} [field F] (E : EllipticCurve F) (K : Type) [field K] [algebra F K]

inductive point
  | zero
  | some (x y : K) (w : y^2 + E.a₁*x*y + E.a₃*y = x^3 + E.a₂*x^2 + E.a₄*x + E.a₆)

notation E(K) := point E K
```

Elliptic curves in Lean

Can define K-points.

```
variables {F : Type} [field F] (E : EllipticCurve F) (K : Type) [field K] [algebra F K]

inductive point
  | zero
  | some (x y : K) (w : y^2 + E.a₁*x*y + E.a₃*y = x^3 + E.a₂*x^2 + E.a₄*x + E.a₆)

notation E(K) := point E K
```

Can define zero.

```
instance : has_zero E(K) := ⟨zero⟩
```

Elliptic curves in Lean

Can define K-points.

```
variables {F : Type} [field F] (E : EllipticCurve F) (K : Type) [field K] [algebra F K]

inductive point
  | zero
  | some (x y : K) (w : y^2 + E.a₁*x*y + E.a₃*y = x^3 + E.a₂*x^2 + E.a₄*x + E.a₆)

notation E(K) := point E K
```

Can define zero.

```
instance : has_zero E(K) := ⟨zero⟩
```

Can define negation.

```
def neg : E(K) → E(K)
  | zero := zero
  | (some x y w) := some x (-y - E.a₁*x - E.a₃)
    begin
      rw [← w],
      ring
    end

instance : has_neg E(K) := ⟨neg⟩
```

Elliptic curves in Lean

Can define K-points.

```
variables {F : Type} [field F] (E : EllipticCurve F) (K : Type) [field K] [algebra F K]

inductive point
  | zero
  | some (x y : K) (w : y^2 + E.a₁*x*y + E.a₃*y = x^3 + E.a₂*x^2 + E.a₄*x + E.a₆)

notation E(K) := point E K
```

Can define addition.

```
def add : E(K) → E(K) → E(K)
  | zero P := P
  | P zero := P
  | (some x₁ y₁ w₁) (some x₂ y₂ w₂) :=
    if x_ne : x₁ ≠ x₂ then
     let
      L := (y₁ - y₂) / (x₁ - x₂),
      x₃ := L^2 + E.a₁*L - E.a₂ - x₁ - x₂,
      y₃ := -L*x₃ - E.a₁*x₃ - y₁ + L*x₁ - E.a₃
     in
      some x₃ y₃ ... -- 100 lines
    else ... -- 100 lines

instance : has_add E(K) := ⟨add⟩
```

Elliptic curves in Lean

Can define K-points.

```
variables {F : Type} [field F] (E : EllipticCurve F) (K : Type) [field K] [algebra F K]

inductive point
  | zero
  | some (x y : K) (w : y^2 + E.a₁*x*y + E.a₃*y = x^3 + E.a₂*x^2 + E.a₄*x + E.a₆)

notation E(K) := point E K
```

Can prove group axioms

```
lemma zero_add (P : E(K)) : 0 + P = P := ...

lemma add_zero (P : E(K)) : P + 0 = P := ...

lemma add_left_neg (P : E(K)) : -P + P = 0 := ...

lemma add_comm (P Q : E(K)) : P + Q = Q + P := ... -- 100 lines

lemma add_assoc (P Q R : E(K)) : (P + Q) + R = P + (Q + R) := ... -- ?? lines
```

Elliptic curves in Lean

Can define K-points.

```
variables {F : Type} [field F] (E : EllipticCurve F) (K : Type) [field K] [algebra F K]

inductive point
  | zero
  | some (x y : K) (w : y^2 + E.a₁*x*y + E.a₃*y = x^3 + E.a₂*x^2 + E.a₄*x + E.a₆)

notation E(K) := point E K
```

Can prove group axioms (except associativity, which is left as a sorry).

```
lemma zero_add (P : E(K)) : 0 + P = P := ...

lemma add_zero (P : E(K)) : P + 0 = P := ...

lemma add_left_neg (P : E(K)) : -P + P = 0 := ...

lemma add_comm (P Q : E(K)) : P + Q = Q + P := ... -- 100 lines

lemma add_assoc (P Q R : E(K)) : (P + Q) + R = P + (Q + R) := ... -- ?? lines
```

Elliptic curves in Lean

Can define K-points.

```
variables {F : Type} [field F] (E : EllipticCurve F) (K : Type) [field K] [algebra F K]

inductive point
  | zero
  | some (x y : K) (w : y^2 + E.a₁*x*y + E.a₃*y = x^3 + E.a₂*x^2 + E.a₄*x + E.a₆)

notation E(K) := point E K
```

Can prove group axioms (except associativity, which is left as a `sorry`).

```
lemma zero_add (P : E(K)) : 0 + P = P := ...

lemma add_zero (P : E(K)) : P + 0 = P := ...

lemma add_left_neg (P : E(K)) : -P + P = 0 := ...

lemma add_comm (P Q : E(K)) : P + Q = Q + P := ... -- 100 lines

lemma add_assoc (P Q R : E(K)) : (P + Q) + R = P + (Q + R) := ... -- ?? lines
```

Can also prove Galois-theoretic properties and structure of torsion points.

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ *is finitely generated.*

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite).

- Reduce to $K \supseteq E[2]$, so that $y^2 = (x - e_1)(x - e_2)(x - e_3)$.

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite).

- Reduce to $K \supseteq E[2]$, so that $y^2 = (x - e_1)(x - e_2)(x - e_3)$.
- Define the complete 2-descent homomorphism

$$\begin{array}{rcc} E(K) & \longrightarrow & K^\times/(K^\times)^2 \times K^\times/(K^\times)^2 \\ \mathcal{O} & \longmapsto & (1, 1) \\ (x, y) & \longmapsto & (x - e_1, x - e_2) \end{array} .$$

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite).

- Reduce to $K \supseteq E[2]$, so that $y^2 = (x - e_1)(x - e_2)(x - e_3)$.
- Define the complete 2-descent homomorphism

$$\begin{array}{rcc} E(K) & \longrightarrow & K^\times/(K^\times)^2 \times K^\times/(K^\times)^2 \\ \mathcal{O} & \longmapsto & (1,1) \\ (x,y) & \longmapsto & (x - e_1, x - e_2) \end{array} .$$

- Prove its kernel is $2E(K)$.

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite).

- Reduce to $K \supseteq E[2]$, so that $y^2 = (x - e_1)(x - e_2)(x - e_3)$.
- Define the complete 2-descent homomorphism

$$\begin{array}{rcc} E(K) & \longrightarrow & K^\times/(K^\times)^2 \times K^\times/(K^\times)^2 \\ \mathcal{O} & \longmapsto & (1, 1) \\ (x, y) & \longmapsto & (x - e_1, x - e_2) \end{array} .$$

- Prove its kernel is $2E(K)$.
- Prove its image lies in a Selmer group $K(S, 2)$.

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite).

- Reduce to $K \supseteq E[2]$, so that $y^2 = (x - e_1)(x - e_2)(x - e_3)$.
- Define the complete 2-descent homomorphism

$$\begin{array}{rcc} E(K) & \longrightarrow & K^\times/(K^\times)^2 \times K^\times/(K^\times)^2 \\ \mathcal{O} & \longmapsto & (1, 1) \\ (x, y) & \longmapsto & (x - e_1, x - e_2) \end{array}.$$

- Prove its kernel is $2E(K)$.
- Prove its image lies in a Selmer group $K(S, 2)$.
- Prove $0 \to \mathcal{O}_K^\times/(\mathcal{O}_K^\times)^n \to K(\emptyset, n) \to \mathrm{Cl}_K[n] \to 0$ is exact.

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite).

- Reduce to $K \supseteq E[2]$, so that $y^2 = (x - e_1)(x - e_2)(x - e_3)$.
- Define the complete 2-descent homomorphism

$$\begin{array}{rcc} E(K) & \longrightarrow & K^\times/(K^\times)^2 \times K^\times/(K^\times)^2 \\ \mathcal{O} & \longmapsto & (1, 1) \\ (x, y) & \longmapsto & (x - e_1, x - e_2) \end{array} .$$

- Prove its kernel is $2E(K)$.
- Prove its image lies in a Selmer group $K(S, 2)$.
- Prove $0 \to \mathcal{O}_K^\times/(\mathcal{O}_K^\times)^n \to K(\emptyset, n) \to \mathrm{Cl}_K[n] \to 0$ is exact.
- Prove Cl_K is finite (done) and $\mathcal{O}_K^\times$ is finitely generated (soon). □

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite $\implies$ $E(\mathbb{Q})$ finitely generated).

- Define the naïve height

$$\begin{array}{cccccc} h & : & E(\mathbb{Q}) & \longrightarrow & \mathbb{R} & \\ & & \mathcal{O} & \longmapsto & 0 & . \\ & & (\frac{n}{d}, y) & \longmapsto & \log\max(|n|,|d|) & \end{array}$$

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite $\implies$ $E(\mathbb{Q})$ finitely generated).

- Define the naïve height

$$\begin{array}{cccccc} h & : & E(\mathbb{Q}) & \longrightarrow & \mathbb{R} & \\ & & \mathcal{O} & \longmapsto & 0 & . \\ & & (\frac{n}{d}, y) & \longmapsto & \log\max(|n|, |d|) & \end{array}$$

- Prove $\forall Q \in E(\mathbb{Q}),\ \exists C \in \mathbb{R},\ \forall P \in E(\mathbb{Q}),\ h(P + Q) \leq 2h(P) + C$.

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite $\implies$ $E(\mathbb{Q})$ finitely generated).

- Define the naïve height

$$\begin{array}{rccl} h : & E(\mathbb{Q}) & \longrightarrow & \mathbb{R} \\ & \mathcal{O} & \longmapsto & 0 \\ & (\frac{n}{d}, y) & \longmapsto & \log\max(|n|, |d|) \end{array}.$$

- Prove $\forall Q \in E(\mathbb{Q}),\ \exists C \in \mathbb{R},\ \forall P \in E(\mathbb{Q}),\ h(P+Q) \leq 2h(P) + C$.
- Prove $\exists C \in \mathbb{R},\ \forall P \in E(\mathbb{Q}),\ 4h(P) \leq h(2P) + C$.

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite $\implies$ $E(\mathbb{Q})$ finitely generated).

- Define the naïve height

$$\begin{array}{rcrcl} h & : & E(\mathbb{Q}) & \longrightarrow & \mathbb{R} \\ & & \mathcal{O} & \longmapsto & 0 \\ & & (\frac{n}{d}, y) & \longmapsto & \log\max(|n|, |d|) \end{array} .$$

- Prove $\forall Q \in E(\mathbb{Q}),\ \exists C \in \mathbb{R},\ \forall P \in E(\mathbb{Q}),\ h(P+Q) \leq 2h(P) + C$.
- Prove $\exists C \in \mathbb{R},\ \forall P \in E(\mathbb{Q}),\ 4h(P) \leq h(2P) + C$.
- Prove $\forall C \in \mathbb{R}$, the set $\{P \in E(\mathbb{Q}) \mid h(P) \leq C\}$ is finite.

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite $\implies$ $E(\mathbb{Q})$ finitely generated).

- Define the naïve height

$$\begin{array}{rcrcl} h & : & E(\mathbb{Q}) & \longrightarrow & \mathbb{R} \\ & & \mathcal{O} & \longmapsto & 0 \\ & & (\frac{n}{d}, y) & \longmapsto & \log\max(|n|, |d|) \end{array}.$$

- Prove $\forall Q \in E(\mathbb{Q}),\ \exists C \in \mathbb{R},\ \forall P \in E(\mathbb{Q}),\ h(P+Q) \leq 2h(P) + C$.
- Prove $\exists C \in \mathbb{R},\ \forall P \in E(\mathbb{Q}),\ 4h(P) \leq h(2P) + C$.
- Prove $\forall C \in \mathbb{R}$, the set $\{P \in E(\mathbb{Q}) \mid h(P) \leq C\}$ is finite.
- Prove the descent theorem (done). $\square$

The Mordell-Weil theorem in Lean

Can prove Mordell's theorem by complete 2-descent and naïve heights.

Theorem (Mordell)

$E(\mathbb{Q})$ is finitely generated.

Proof ($E(\mathbb{Q})/2E(\mathbb{Q})$ finite $\implies$ $E(\mathbb{Q})$ finitely generated).

- Define the naïve height

$$\begin{array}{rcrcl} h & : & E(\mathbb{Q}) & \longrightarrow & \mathbb{R} \\ & & \mathcal{O} & \longmapsto & 0 \\ & & (\frac{n}{d}, y) & \longmapsto & \log \max(|n|, |d|) \end{array}.$$

- Prove $\forall Q \in E(\mathbb{Q}),\ \exists C \in \mathbb{R},\ \forall P \in E(\mathbb{Q}),\ h(P+Q) \leq 2h(P) + C$.
- Prove $\exists C \in \mathbb{R},\ \forall P \in E(\mathbb{Q}),\ 4h(P) \leq h(2P) + C$.
- Prove $\forall C \in \mathbb{R}$, the set $\{P \in E(\mathbb{Q}) \mid h(P) \leq C\}$ is finite.
- Prove the descent theorem (done). $\square$

Can finally define the algebraic rank of $E(\mathbb{Q})$.

Algebraic number theory in Lean

Here are some recent developments.

Algebraic number theory in Lean

Here are some recent developments.

Completed:

- Quadratic reciprocity
- Hensel's lemma

Algebraic number theory in Lean

Here are some recent developments.

Completed:

- Quadratic reciprocity
- Hensel's lemma
- UF in Dedekind domains
- $\#\mathrm{Cl}_K < \infty$ for global fields

Algebraic number theory in Lean

Here are some recent developments.

Completed:

- Quadratic reciprocity
- Hensel's lemma
- UF in Dedekind domains
- $\#\mathrm{Cl}_K < \infty$ for global fields
- Adèles and idèles
- Statement of global CFT

Algebraic number theory in Lean

Here are some recent developments.

Completed:

- Quadratic reciprocity
- Hensel's lemma
- UF in Dedekind domains
- $\#\mathrm{Cl}_K < \infty$ for global fields
- Adèles and idèles
- Statement of global CFT
- L-series of arithmetic functions
- Bernoulli polynomials

Algebraic number theory in Lean

Here are some recent developments.

Completed:

- Quadratic reciprocity
- Hensel's lemma
- UF in Dedekind domains
- $\#\mathrm{Cl}_K < \infty$ for global fields
- Adèles and idèles
- Statement of global CFT
- L-series of arithmetic functions
- Bernoulli polynomials
- Perfectoid spaces
- Liquid tensor experiment

Algebraic number theory in Lean

Here are some recent developments.

Completed:

- Quadratic reciprocity
- Hensel's lemma
- UF in Dedekind domains
- $\#\mathrm{Cl}_K < \infty$ for global fields
- Adèles and idèles
- Statement of global CFT
- L-series of arithmetic functions
- Bernoulli polynomials
- Perfectoid spaces
- Liquid tensor experiment

Ongoing:

- S-unit theorem (**HELP**)
- FLT for regular primes
- p-adic L-functions
- B_{dR}, B_{HT}, and $\mathrm{B}_{\mathrm{cris}}$

Algebraic number theory in Lean

Here are some recent developments.

Completed:

- Quadratic reciprocity
- Hensel's lemma
- UF in Dedekind domains
- $\#\mathrm{Cl}_K < \infty$ for global fields
- Adèles and idèles
- Statement of global CFT
- L-series of arithmetic functions
- Bernoulli polynomials
- Perfectoid spaces
- Liquid tensor experiment

Ongoing:

- S-unit theorem (**HELP**)
- FLT for regular primes
- p-adic L-functions
- B_{dR}, B_{HT}, and $\mathrm{B}_{\mathrm{cris}}$
- Modular forms
- Étale cohomology
- Local CFT

Algebraic number theory in Lean

Here are some recent developments.

Completed:

- Quadratic reciprocity
- Hensel's lemma
- UF in Dedekind domains
- $\#\mathrm{Cl}_K < \infty$ for global fields
- Adèles and idèles
- Statement of global CFT
- L-series of arithmetic functions
- Bernoulli polynomials
- Perfectoid spaces
- Liquid tensor experiment

Ongoing:

- S-unit theorem (**HELP**)
- FLT for regular primes
- p-adic L-functions
- B_{dR}, B_{HT}, and $\mathrm{B}_{\mathrm{cris}}$
- Modular forms
- Étale cohomology
- Local CFT
- Statement of BSD
- Statement of GAGA
- Statement of R=T

Thank you!

Check out the leanprover community!