The Mordell-Weil theorem in Lean

David Kurniadi Angdinata

Spring 2022

Abstract

In 1922, Louis Mordell proved that the group of rational points of an elliptic curve is finitely generated. Subsequently, in 1928, André Weil generalised this for abelian varieties over number fields, and their collective result has since been known as the Mordell-Weil theorem. This report documents an ongoing attempt to formalise relevant definitions and a proof of the Mordell-Weil theorem for an elliptic curve over a number field in the Lean theorem prover, along with any design decisions made in the process.

Contents

1 Background **2**
1.1 Elliptic curves . . . 2
1.2 The Mordell-Weil theorem . . . 3
1.3 The Lean theorem prover . . . 4

2 Implementation **5**
2.1 Weierstrass equations . . . 5
2.1.1 The `EllipticCurve` structure . . . 5
2.1.2 Changes of variables . . . 5
2.2 Group law . . . 6
2.2.1 The `point` inductive . . . 6
2.2.2 Group operations . . . 6
2.2.3 Group axioms . . . 7
2.2.4 Functoriality and Galois actions . . . 8
2.2.5 Changes of variables . . . 9
2.3 Torsion points . . . 10
2.3.1 Multiplication by n maps . . . 10
2.3.2 Functoriality and Galois actions . . . 10
2.3.3 Cubic discriminants . . . 11
2.3.4 2-division polynomials . . . 11
2.3.5 2-torsion subgroups . . . 12
2.4 Weak Mordell-Weil theorem . . . 13
2.4.1 Reduction lemma . . . 13
2.4.2 Complete 2-descent . . . 14
2.4.3 The Selmer group . . . 15
2.5 Height functions . . . 18
2.5.1 Naive heights . . . 18
2.5.2 Properties of heights . . . 18
2.5.3 Descent theorem . . . 19

3 Future **20**
3.1 Current progress . . . 20
3.2 Potential projects . . . 20
3.3 Concluding retrospect . . . 20

1 Background

This section provides the relevant definitions in the arithmetic of elliptic curves, outlines a proof of the Mordell-Weil theorem for an elliptic curve over a number field, and briefly introduces the Lean theorem prover and its mathematical library. For more details on the constructions, consult Silverman's book *The Arithmetic of Elliptic Curves* [Sil09] and Katz-Mazur's book *Arithmetic Moduli of Elliptic Curves* [KM85].

1.1 Elliptic curves

Depending on the ambient category the reader wishes to consider, there are many seemingly different definitions of an elliptic curve with varying degrees of abstraction, which are non-trivially equivalent in many cases. For the sake of a short exposition, this will begin at the generality of schemes.

An **elliptic curve** over a scheme S is an S-scheme E, equipped with a **zero section** $S \to E$ of the natural morphism $f : E \to S$, such that f is smooth, proper, and all its geometric fibres are integral curves of genus one. For an S-scheme T, the **set of T-points** $E(T)$ of E is the set of S-scheme morphisms $T \to E$, which furnishes a contravariant functor $\mathbf{Sch}_S \to \mathbf{Set}$ given by $T \mapsto E(T)$. On the other hand, E represents the relative Picard functor $\mathbf{Sch}_S \to \mathbf{Ab}$ given by $T \mapsto \operatorname{Pic}^0_{E/S}(T)$, and there is a natural identification $E(T) \xrightarrow{\sim} \operatorname{Pic}^0_{E/S}(T)$, giving E the structure of an abelian scheme. In some sense this is a global definition, great for the purposes of proving general results in algebraic geometry, but for the purposes of the Mordell-Weil theorem it suffices to consider the local definition over a field.

When $S = \operatorname{Spec} F$ for a field F, the above definition translates to a single smooth, proper, geometrically integral curve of genus one E, equipped with a choice of a point $\mathcal{O} \in E$ given by the zero section that will play the role of an identity element later. For the S-scheme $T = \operatorname{Spec} K$ given by a field extension K of F, the set of T-points, now denoted by the **set of K-points** $E(K)$, can be naturally identified with the relative Picard group, now just the usual **Picard group** $\operatorname{Pic}^0_{E/F}(K)$ of degree zero divisors modulo principal divisors. More concretely, this identification sends a K-point $P \in E(K)$ to the class of the degree zero divisor $[P] - [\mathcal{O}]$, which can be shown to be bijective due to the Riemann-Roch theorem. Considering only the field extensions of F then furnishes a covariant functor from category of fields containing F to $\mathbf{Ab}$ given by $K \mapsto E(K)$, which can be enriched to a functor $\mathbf{Alg}_F \to \mathbf{Ab}$ with a bit of work.

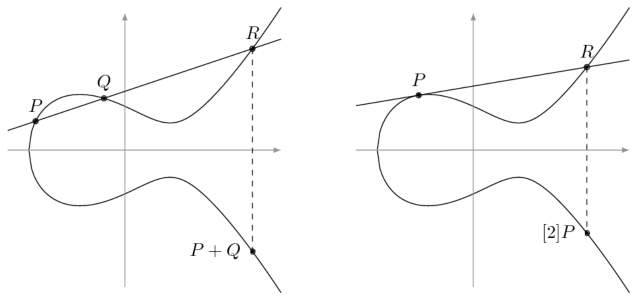
As a further consequence of the Riemann-Roch theorem, elliptic curves over a field F can also be given a **Weierstrass model**, namely the projective plane curve given by the equation

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3, \qquad a_i \in F.$$

The distinguished point $\mathcal{O}$ is often chosen to be the unique point at infinity $[X : Y : Z] = [0 : 1 : 0]$, and the remaining technical conditions are reduced to the non-vanishing of a quantity called the **discriminant**

$$\begin{aligned}\Delta \coloneqq &-\left(a_1^2 + 4a_2\right)^2 \left(a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2\right) - 8\left(2a_4 + a_1a_3\right)^3 \\ &- 27\left(a_3^2 + 4a_6\right)^2 + 9\left(a_1^2 + 4a_2\right)\left(2a_4 + a_1a_3\right)\left(a_3^2 + 4a_6\right).\end{aligned}$$

Under the identification $E(K) \xrightarrow{\sim} \operatorname{Pic}^0_{E/F}(K)$, the **group law** is simply a chord-and-tangent process on the K-points of E, governed by the property that three K-points sum to zero precisely if they are collinear, as illustrated below. This is well-defined, as a projective line necessarily intersect E at exactly three K-points, counted with multiplicity and possibly at $\mathcal{O}$, by Bézout's theorem. In particular, the group law can be given by completely explicit equations, albeit having to consider several cases at all times.



Morphisms of elliptic curves over a fixed base scheme is complicated to define, but given the Weierstrass model over a field, isomorphisms preserving $\mathcal{O}$ can be given explicitly by a linear change of variables

$$[X:Y:Z] \mapsto \left[u^2X + rZ : u^3Y + u^2sX + tZ : Z\right], \qquad u \in F^\times, \qquad r,s,t \in F,$$

which extend to an isomorphism of their corresponding group of K-points by functoriality. For instance, completing the square and completing the cube are both admissible changes of variables provided $\operatorname{ch} F \neq 2,3$, so that the Weierstrass model can be reduced to a **short Weierstrass model**

$$Y^2Z = X^3 + AXZ^3 + BZ^3, \qquad A,B \in F,$$

with the same unique point at infinity, and whose affine K-points are symmetric about the $x := X/Z$-axis. Furthermore, any such change of variables preserves a quantity called the **j-invariant**

$$\mathrm{j} := \frac{\left(-48a_4 - 24a_1a_3 + 16a_2^2 + 8a_1^2a_2 + a_1^4\right)^3}{\Delta},$$

which also classifies elliptic curves up to isomorphism over all finite extensions of F simultaneously.

More general morphisms preserving $\mathcal{O}$ are called **isogenies**, with a crucial example being the multiplication by $n \in \mathbb{N}$ map $[n] : E \to E$, whose kernel is the **n-torsion subgroup** scheme $E[n]$. The x-coordinates of the K-points in $E(K)[n]$ can then be given by the K-roots of a certain **n-division polynomial** $\psi_n(x) \in F[X]$, which are defined inductively. For instance, under the symmetric short Weierstrass model, an affine K-point $(x,y) \in E(K)$ lies in $E(K)[2]$ precisely if $y = 0$, by definition of the group law, so that the affine K-points in $E(K)[2]$ are exactly the K-roots of $\psi_2(x) := x^3 + Ax + B$. In particular, $E(K)[2]$ has at most four K-points including $\mathcal{O}$, and has exactly four K-points when $\psi_2(x)$ splits in K.

Finally, given extensions $L/K/F$, the group of F-automorphisms $\operatorname{Aut}(L/K)$ acts on the $E(L)$ and $E(L)[n]$ naturally, which, when they are Galois, have invariants $E(K)$ and $E(K)[n]$ respectively. This will be crucial in the explicit Galois cohomology of elliptic curves to be seen later.

1.2 The Mordell-Weil theorem

The following is perhaps the most fundamental result in the arithmetic of elliptic curves.

Theorem 1.1 (Mordell-Weil). *Let F be a number field. Then $E(F)$ is finitely generated.*

In particular, the structure theorem for finitely generated abelian groups says that $E(F)$ is isomorphic to the direct product of a finite group T and a finite number $r \in \mathbb{N}$ of copies of $\mathbb{Z}$. Here, T is the **torsion subgroup** of E, which for $F = \mathbb{Q}$ has been completely classified by Mazur, with substantial partial results for other number fields. On the other hand, the **rank** $r \in \mathbb{N}$ of E is subtly more mysterious, lending itself to a multitude of important problems in number theory, most notably the Birch and Swinnerton-Dyer conjecture that equates it with the order of vanishing of the Hasse-Weil L-function of E.

Weil's original proof of Theorem 1.1, as well as its subsequent interpretations, can generally be divided into two large chunks, the first of which is often referred to as the **weak Mordell-Weil theorem**, which involves proving that $E(F)/2E(F)$ is finite. With modern technical advances, this is done by implicitly embedding $E(F)/2E(F)$ in a computably finite group $\operatorname{Sel}(F, E[n])$ obtained via the machinery of Galois cohomology. This embedding can be given by an explicit **complete 2-descent** homomorphism

$$\delta : E(F) \to F^\times/\left(F^\times\right)^2 \times F^\times/\left(F^\times\right)^2,$$

whose kernel is exactly $2E(F)$ and whose image is contained in two copies of a certain provably finite subgroup $F(S,2)$ called a **Selmer group**. It is worth noting that there are other descent methods, such as by a cyclic 2-isogeny, and that there are other less direct paths to the same conclusion, such as via reducing the problem to the finiteness of a certain composite extension analogous to Kummer theory, but complete 2-descent remains to be the fastest and least technical proof of the weak Mordell-Weil theorem.

The weak Mordell-Weil theorem is certainly necessary for the full Mordell-Weil theorem, and its sufficiency is a consequence of the existence of an explicit **naive height** on $E(F)$ measuring the complexity of F-points on $E(F)$. In particular, it is a function $\mathrm{h} : E(F) \to \mathbb{R}_{\geq 0}$ satisfying the following three properties.

- For all $Q \in E(F)$, there exists $C_1 \in \mathbb{R}$ such that for all $P \in E(F)$, we have $\mathrm{h}(P+Q) \leq 2\mathrm{h}(P) + C_1$.
- There exists $C_2 \in \mathbb{R}$ such that for all $P \in E(F)$, we have $4\mathrm{h}(P) \leq \mathrm{h}(2P) + C_2$.
- For all $C_3 \in \mathbb{R}$, we have $\{P \in E(F) \mid \mathrm{h}(P) \leq C_3\}$ is finite.

This forms the latter half of the proof, and completes the proof of Theorem 1.1 by the **descent theorem**, which says that any abelian group A endowed with a height function $A \to \mathbb{R}_{\geq 0}$ such that $A/2A$ is finite is necessarily finitely generated. It is also worth noting that there is a more canonical height function studied by Néron and Tate that happens to be a quadratic form, which can also be used to deduce the same result, but for the purposes of the Mordell-Weil theorem it suffices to consider the aforementioned naive height.

Detailed steps of both halves, especially the definition of the complete 2-descent homomorphism and the naive height function, will be elucidated during a discussion of their implementation in the following section.

1.3 The Lean theorem prover

The formalisation of the Mordell-Weil theorem will be done in the **Lean theorem prover** [Mic], a functional programming language and an interactive theorem prover based on the calculus of constructions with inductive types, developed principally by Leonardo de Moura at Microsoft Research in 2013. Essentially, dependent type theory in Lean plays the role of set theory in mathematics via a version of the Curry-Howard correspondence, where the statement of a theorem is formalised as a type in Lean, and a proof of said theorem, if it exists, will be the unique term of that type. This interpretation may present unexpected challenges to a working mathematician, such as the subtle notion of equality in Lean between two sets that are obviously equal in conventional mathematics, but are far from being the same type in Lean.

While a proof of a theorem in Lean is merely a term, clever monadic metaprogramming has enabled the feature known as **tactics**, programmes that act on a proof state rather than specifying the exact term necessary to inhabit a type. By hiding the various bookkeeping complications, such as the composition of many terms within a single proof, specifying tactics to fill a proof in tactic mode allows the working mathematician to focus on the proof argument more naturally than in the usual term mode. For instance, the type `p → p`, inhabited by the term `λ p, p` in term mode, is more naturally expressed in tactic mode as `begin intro p, exact p end`, via the aptly-named tactics `intro` and `exact` and delimited by the keywords `begin` and `end`. There is a long list of other more sophisticated tactics available in the community website, such as `simp`, which simplifies the proof state with existing lemmas, and `ring`, which normalises expressions in commutative ring theory. Finally, note that the keyword `sorry` inhabits all types, and acts as a placeholder in both term and tactic modes to the proof of a theorem whenever deemed necessary.

The formal statement and eventual proof of the Mordell-Weil theorem is only made remotely possible by the existing Lean mathematical library **mathlib** [Lea], a community-driven effort to build a unified library of formalised mathematics, including large chunks of undergraduate-level algebra, analysis, topology, geometry, and other areas. In particular, the ongoing proof of the Mordell-Weil theorem crucially uses many basic results from the `group_theory`, `ring_theory`, `field_theory`, and `number_theory` libraries, while awaiting basic results of local fields and Galois cohomology to be formalised in the near future. The current definition of elliptic curves is located in the `algebraic_geometry` library despite not using anything in the library itself, but should eventually use the newly-developed Proj construction of schemes.

For the remainder of the report, code snippets of theorem statements will be presented whenever informative, but are often pseudocode to simplify cumbersome notation or highlight certain parts, while encouraging the reader to explore the actual repository. Most of the code is found in the `mordell_weil` branch of mathlib under the `algebraic_geometry/EllipticCurve` directory, but specific files will be listed in each section.

As a final remark, it is worth noting that the Weierstrass model of an elliptic curve over a field F with $\mathrm{ch}\, F \neq 2, 3$ has been formalised in other theorem provers, namely in CoCoA [Fri98] and in Coq [HT07] [BS14], but all of them have stopped at the proof of the group law. The weak Birch and Swinnerton-Dyer conjecture has also been stated in Lean [Bel21], leaving a `sorry` for the proof of the Mordell-Weil theorem.

2 Implementation

This section describes the ongoing formalisation of the Mordell-Weil theorem mostly in the `mordell_weil` branch of mathlib, along with any difficulties encountered and design decisions made in the process.

2.1 Weierstrass equations

The code in this section can be found in `algebraic_geometry/EllipticCurve.lean`.

2.1.1 The `EllipticCurve` structure

The first order was to select an appropriate mathematically-accurate definition of elliptic curves that is sufficiently general and suitable for computational purposes, yet sufficiently realistic under the current state of mathlib. While schemes do exist in mathlib, only affine schemes, and very recently projective schemes, have been properly constructed as examples. There are few isolated definitions and results on schemes, insufficient to define the genus of a curve or prove the Riemann-Roch theorem, which excludes the possibility of defining elliptic curves over a scheme while still allowing for locally-defined Weierstrass models. The choice was eventually made by Kevin Buzzard to define elliptic curves directly by the coefficients of a Weierstrass model over a commutative ring, along with important quantities such as the discriminant and the j-invariant.

```
def disc_aux {R : Type} [comm_ring R] (a₁ a₂ a₃ a₄ a₆ : R) : R :=
-(a₁^2 + 4*a₂)^2*(a₁^2*a₆ + 4*a₂*a₆ - a₁*a₃*a₄ + a₂*a₃^2 - a₄^2)
  - 8*(2*a₄ + a₁*a₃)^3 - 27*(a₃^2 + 4*a₆)^2 + 9*(a₁^2 + 4*a₂)*(2*a₄ + a₁*a₃)*(a₃^2 + 4*a₆)

structure EllipticCurve (R : Type) [comm_ring R] :=
(a₁ a₂ a₃ a₄ a₆ : R) (disc_unit : units R) (disc_unit_eq : disc_unit.val = disc_aux a₁ a₂ a₃ a₄ a₆)

variables {R : Type} [comm_ring R] (E : EllipticCurve R)

def disc : R := disc_aux E.a₁ E.a₂ E.a₃ E.a₄ E.a₆

def j : R := E.disc_unit.inv*(-48*E.a₄ - 24*E.a₁*E.a₃ + 16*E.a₂^2 + 8*E.a₁^2*E.a₂ + E.a₁^4)^3
```

Note that while this is the current mathlib definition, it is only mathematically-accurate for commutative rings whose class group has trivial 12-torsion, which includes fields and principal ideal domains.

2.1.2 Changes of variables

Under the Weierstrass model, isomorphisms are then given as simple linear changes of variables, with completing the square `cov2` and completing the cube `cov3` as special cases. The fact that the j-invariant is invariant under these changes of variables is a simple application of `ring1`.

```
def cov (u : units R) (r s t : R) : EllipticCurve R :=
{ a₁           := u.inv*(E.a₁ + 2*s),
  a₂           := u.inv^2*(E.a₂ - s*E.a₁ + 3*r - s^2),
  a₃           := u.inv^3*(E.a₃ + r*E.a₁ + 2*t),
  a₄           := u.inv^4*(E.a₄ - s*E.a₃ + 2*r*E.a₂ - (t + r*s)*E.a₁ + 3*r^2 - 2*s*t),
  a₆           := u.inv^6*(E.a₆ + r*E.a₄ + r^2*E.a₂ + r^3 - t*E.a₃ - t^2 - r*t*E.a₁),
  disc_unit    := ⟨u.inv^12*E.disc_unit.val, u.val^12*E.disc_unit.inv, by ..., by ...⟩, -- ring1
  disc_unit_eq := by { simp only [disc_unit_eq, disc_aux], ring1 } }

lemma cov.j_eq (u : units R) (r s t : R) : (E.cov u r s t).j = E.j := by ... -- ring1

def cov2 : EllipticCurve R := E.cov 1 0 (-E.a₁/2) (-E.a₃/2)

def cov3 : EllipticCurve R := E.cov 1 (-(a₁^2 + 4*a₂)/12) 0 0
```

Again, this is only mathematically-accurate for rings whose class group has trivial 12-torsion. Throughout, whenever a proof is not informative or does not fit in a line, it will be shortened to `by ...` for brevity.

2.2 Group law

The code in this section can be found in `algebraic_geometry/EllipticCurve/point.lean`.

2.2.1 The `point` inductive

Due to the choice of defining elliptic curves by coefficients of the Weierstrass equation, the group structure on $E(K)$ does not come for free via the identification with $\mathrm{Pic}^0_{E/F}(K)$, and the type of K-points has to be defined manually as projective coordinates satisfying the Weierstrass equation in $\mathbb{P}^2$. However, using projective coordinates is clearly an overkill, as the point at infinity is unique, so instead the K-points are defined as an `inductive` between $\mathcal{O}$ and the remaining affine K-points satisfying a Weierstrass equation.

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

local notation F↑K := algebra_map F K

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

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

Here, a clean notation is introduced for $E(K)$. Note that the coefficients of the Weierstrass equation lie in the base field F, and has to be coerced via `algebra_map` to the coordinate field K to be type-correct.

2.2.2 Group operations

The group operation on $E(K)$ is then defined via the aforementioned chord-and-tangent process, with $\mathcal{O}$ acting as the identity, which is achieved by writing down an instance of the typeclass `has_zero`.

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

Recall that negation of a K-point is a reflection $(x, y) \mapsto (x, -y)$ about the x-axis in the short Weierstrass model, but in general there are additional contributions from a_1 and a_3. It is then necessary, by definition of the `inductive`, to supply a proof that the negated K-point still lies on E, given a proof that the original K-point lies on E. This is a simple matter of rewriting the unaltered right hand side of the Weierstrass equation `w` with the left hand side via `rw`, then running the fast commutative ring theory normaliser `ring1`.

```
def neg : E(K) → E(K)
| zero         := zero
| (some x y w) := some x (-y - (F↑K)E.a₁*x - (F↑K)E.a₃)
begin
  rw [← w],
  ring1
end

instance : has_neg E(K) := ⟨neg⟩
```

Addition, however, is quite a bit more involved. After splitting the trivial cases involving $\mathcal{O}$, the primary computation is that of the line `L` joining two affine K-points, which further splits into cases depending on whether they are distinct or symmetric about the x-axis. The third intersection is obtained by inspecting the coefficients of a cubic equation and then negated, which is essentially the same code in both cases.

On the other hand, the proof that the resulting K-point still lies on E is a matter of checking the equality of two huge rational functions, which in theory is doable by the field theory simplifier `field_simp`. Yet, the tactic seems to be relatively primitive, in the sense that it cannot simplify increasing denominator exponents, resulting in expressions that are far too large to work with, and causing frustrating deterministic timeouts. Instead, the final decision was to manually break the expressions down into several sub-expressions, automating steps with the fast `ring1` whenever division is avoidable, resulting in about 200 lines of hideous polynomial equalities but guaranteeing that the definitions will compile within reasonable time.

```
def add : E(K) → E(K) → E(K)
| zero            P                := P
| P               zero             := P
| (some x₁ y₁ w₁) (some x₂ y₂ w₂) :=

-- add distinct points
if x_ne : x₁ ≠ x₂ then
  let L  := (y₁ - y₂)/(x₁ - x₂),
      x₃ := L^2 + (F↑K)E.a₁*L - (F↑K)E.a₂ - x₁ - x₂,
      y₃ := -L*x₃ - (F↑K)E.a₁*x₃ - y₁ + L*x₁ - (F↑K)E.a₃
  in some x₃ y₃ $ by ... -- field_simp but explicit

-- double a point
else if y_ne : y₁ + y₂ + (F↑K)E.a₁*x₂ + (F↑K)E.a₃ ≠ 0 then
  let L  := (3*x₁^2 + 2*(F↑K)E.a₂*x₁ + (F↑K)E.a₄ - (F↑K)E.a₁*y₁)/(2*y₁ + (F↑K)E.a₁*x₁ + (F↑K)E.a₃),
      x₃ := L^2 + (F↑K)E.a₁*L - (F↑K)E.a₂ - 2*x₁,
      y₃ := -L*x₃ - (F↑K)E.a₁*x₃ - y₁ + L*x₁ - (F↑K)E.a₃
  in some x₃ y₃ $ by ... -- field_simp but explicit

-- draw vertical line
else
  zero

instance : has_add E(K) := ⟨add⟩
```

Note that the actual code extracts the case of point doubling into a separate function, and separates the actual definitions from the proofs that the resulting K-point lies in E.

2.2.3 Group axioms

Most of the five remaining axioms of an add_comm_group are relatively easy to prove, with commutativity being split into six cases that are relatively straightforward to check just by unfolding definitions.

```
lemma zero_add (P : E(K)) : 0 + P = P := by cases P; refl

lemma add_zero (P : E(K)) : P + 0 = P := by cases P; refl

lemma add_left_neg (P : E(K)) : -P + P = 0 :=
begin
  cases P,
  { refl },
  { ... } -- ring1
end

lemma add_comm (P Q : E(K)) : P + Q = Q + P :=
begin
  rcases ⟨P, Q⟩ with ⟨_ | _, _ | _⟩,
  ... -- six cases
end
```

On the other hand, associativity seems to be an entirely different beast.

```
lemma add_assoc (P Q R : E(K)) : (P + Q) + R = P + (Q + R) :=
begin
  rcases ⟨P, Q, R⟩ with ⟨_ | _, _ | _, _ | _⟩,
  ... -- ??? cases
end
```

Preliminary attempts to replicate the methods for checking commutativity have proved to be futile due to the sheer number of cases to consider, and some estimates put the number of coefficients of the relevant polynomial identities to be checked to be well over a hundred thousand. This is a problem known to be difficult even in conventional mathematics, with several indirect proofs to avoid bashing out the algebra.

- One could use the Cayley-Bacharach theorem in classical incidence geometry [Cas91, Lemma 7.1] to directly equate two affine K-points `(P + Q) + R` and `P + (Q + R)` defined differently by the chord-and-tangent process. This requires Bézout's theorem, which in turn requires the notion of intersection multiplicities, which need significant work to define given the current state of mathlib.

- One could also turn to the theory of elliptic functions in complex analysis, and prove a version of the uniformisation theorem [Sil09, Corollary VI.5.1.1], giving a lattice $\Lambda \subset \mathbb{C}$ and an explicit isomorphism of groups $\mathbb{C}/\Lambda \xrightarrow{\sim} E(\mathbb{C})$. This is again far from being formalised in mathlib, and only works well when $\operatorname{ch} F = 0$ by the Lefschetz principle, yet delving into $\operatorname{ch} F \neq 0$ will be inevitable.

- The proper way to obtain the group structure in $E(K)$ is via the identification with $\operatorname{Pic}^0_{E/F}(K)$ [Sil09, Proposition III.3.4], which boils down to divisors, differentials, and the Riemann-Roch theorem. This was achieved in Coq albeit only for the short Weierstrass model, and is likely the way forward.

Despite the tedium, there is currently an ongoing attempt to bash out the algebra in the `ell_add_assoc` branch, but the ultimate decision was to leave associativity as a `sorry` and move on, as all the methods listed here will lead the project too far afield from the original aim of proving the Mordell-Weil theorem.

In retrospect, the primary benefit of using affine coordinates is the uniqueness of each affine K-point (x, y) given coordinates `x` and `y` and a proof `w` that $(x, y) \in E(K)$, whereas using projective coordinates will introduce an additional layer of quotients and a third variable that is often redundant. However, using affine coordinates also introduces an additional layer of cases to check at every stage and having to resort to the awkward `field_simp`, whereas no division is necessary in projective coordinates, allowing `ring` to be used a lot more generously. Given that projectivisation is only recently developed in mathlib, it is perhaps worth venturing into projective coordinates eventually to automate the computations with `ring`.

2.2.4 Functoriality and Galois actions

Now the functoriality of $K \mapsto E(K)$ has to be proven manually, but this is a simple matter of unfolding the definitions and pushing an F-algebra homomorphism $\phi : K \to L$ of fields through several cases.

```
variables {L M : Type} [field L] [field M] [algebra F L] [algebra F M] [algebra K L] [algebra K M]
          [algebra L M] [is_scalar_tower F K L] [is_scalar_tower F K M] [is_scalar_tower F L M]

def point_hom.to_fun (φ : K →ₐ[F] L) : E(K) → E(L)
| zero         := zero
| (some x y w) := some (φ x) (φ y) $ by ... -- push it inwards

def point_hom (φ : K →ₐ[F] L) : E(K) →+ E(L) :=
{ to_fun    := point_hom.to_fun φ,
  map_zero' := rfl,
  map_add'  := by ... } -- push it inwards

local notation K→[F]L := (algebra.of_id K L).restrict_scalars F

lemma point_hom.id (P : E(K)) : point_hom (K→[F]K) P = P := by cases P; refl

lemma point_hom.comp (P : E(K)) :
  point_hom (L→[F]M) (point_hom (K→[F]L) P) = point_hom ((L→[F]M).comp (K→[F]L)) P :=
by cases P; refl
```

While the category of fields extending a base field F does not exist in mathlib, `point_hom.id` and `point_hom.comp` are precisely the axioms for a covariant functor from this category to **Ab**. More work needs to be done to turn this into a functor $\mathbf{Alg}_F \to \mathbf{Ab}$, since this requires defining the group law over an arbitrary F-algebra, but just having these lemmas suffices for most purposes.

Similarly, defining the distributive multiplicative action $\mathrm{Aut}(L/K) \curvearrowright E(L)$ is again a matter of unfolding the definitions and pushing a K-automorphism $\sigma \in \mathrm{Aut}(L/K)$ through several cases.

```
def point_gal (σ : L ≃ₐ[K] L) : E(L) → E(L)
| zero         := zero
| (some x y w) := some (σ • x) (σ • y) $ by ... -- push it inwards

instance : distrib_mul_action (L ≃ₐ[K] L) E(L) :=
{ smul      := point_gal,
  one_smul  := λ P, by cases P; refl,
  mul_smul  := λ _ _ P, by cases P; refl,
  smul_add  := by ..., -- push it inwards
  smul_zero := λ _, rfl }
```

When L/K is Galois, Galois theory shows that the invariant subgroup of $\mathrm{Gal}(L/K) \curvearrowright E(L)$ is precisely $E(K)$, but this is only proven for finite L/K as only finite Galois theory exists in mathlib. It is worth noting that, to ensure type-correctness for the subgroups of $E(L)$, this invariant subgroup is really the injective image of $E(K)$ in $E(L)$, which requires a separate proof that `point_hom` is injective.

```
lemma point_hom.injective (φ : K →ₐ[F] L) : function.injective $ point_hom φ := by ...

def point_gal.fixed : add_subgroup E(L) :=
{ carrier   := mul_action.fixed_points (L ≃ₐ[K] L) E(L),
  zero_mem' := λ _, rfl,
  add_mem'  := λ _ _ hP hQ _, by rw [smul_add, hP, hQ],
  neg_mem'  := λ _ hP σ, by simpa only [smul_neg, neg_inj] using hP σ }

lemma point_gal.fixed.eq [finite_dimensional K L] [is_galois K L] :
  point_gal.fixed E K L = (point_hom $ K→[F]L).range :=
by ... -- finite Galois theory
```

2.2.5 Changes of variables

Finally, recall that an admissible change of variables $E \xrightarrow{\sim} E'$ extends to a group isomorphism $E(K) \xrightarrow{\sim} E(K')$, the proof of which is a simple manipulation of coefficients with `ring1`.

```
variables (u : units F) (r s t : F)

def cov.to_fun : (E.cov u r s t)(K) → E(K)
| zero         := zero
| (some x y w) := some (u.val^2*x + r) (u.val^3*y + u.val^2*s*x + t) $ by ... -- ring1

def cov.inv_fun : E(K) → (E.cov u r s t)(K)
| zero         := zero
| (some x y w) := some (u.inv^2*(x - r)) (u.inv^3*(y - s*x + r*s - t)) $ by ... -- ring1

def cov.equiv_add : (E.cov u r s t)(K) ≃+ E(K) :=
{ to_fun    := cov.to_fun u r s t,
  inv_fun   := cov.inv_fun u r s t,
  left_inv  := by ..., -- ring1
  right_inv := by ..., -- ring1
  map_add'  := by ... } -- ring1

def cov2.equiv_add : E.cov2(K) ≃+ E(K) := cov.equiv_add 1 0 (-E.a₁/2) (-E.a₃/2)

def cov3.equiv_add : E.cov3(K) ≃+ E(K) := cov.equiv_add 1 (-(a₁^2 + 4*a₂)/12) 0 0
```

2.3 Torsion points

The code in this section can be found in `algebraic_geometry/EllipticCurve/torsion.lean`.

2.3.1 Multiplication by n maps

With the more restricted definition of an elliptic curve, defining general morphisms or isogenies $E \to E'$, even when $E = E'$, would be rather tricky. Fortunately, the maps $[n] : E \to E$ are often the whole story at least in the case $\operatorname{ch} F = 0$, and they are trivial to define on $E(L)$ via existing `has_scalar` instances on abelian groups, taking into account the equivariant action $\operatorname{Aut}(L/K) \curvearrowright E(L)[n]$ that will be needed later.

```
variables (n : ℕ) (σ : L ≃ₐ[K] L) (P : E(L))

def mul_by' : E(L) →+[L ≃ₐ[K] L] E(L) :=
{ to_fun    := (·) n,
  map_smul' := by ..., -- simp
  map_zero' := smul_zero n,
  map_add'  := smul_add n }

lemma mul_by.map_smul : n · σ · P = σ · n · P := (mul_by' n).map_smul' σ P

def mul_by : E(K) →+ E(K) := mul_by' n

notation E(K)[n] := (mul_by n : E(K) →+ E(K)).ker
notation E(K)·n := (mul_by n : E(K) →+ E(K)).range
notation E(K)/n := E(K) / E(K)·n
```

Here, three suggestive notations are introduced for ease of reading the remainder of this report, but they are likely to be removed in the actual code given an ongoing discussion on notation for pointwise operations.

2.3.2 Functoriality and Galois actions

While functoriality and properties of the Galois action are just as easily proven for the image $nE(K)$ and the cokernel $E(K)/n$, for the purposes of complete 2-descent it suffices to prove them for the kernel $E(K)[n]$, all of which follow almost immediately from their respective statements for $E(K)$.

```
def ker_hom (φ : K →ₐ[F] L) : E(K)[n] →+ E(L)[n] :=
⟨λ ⟨P, hP⟩, ⟨point_hom φ P, by ...⟩, rfl, by ...⟩

lemma ker_hom.id (P : E(K)[n]) : ker_hom n (K→[F]K) P = P := by rcases P with _ | _; refl

lemma ker_hom.comp (P : E(K)[n]) :
  ker_hom n (L→[F]M) (ker_hom n (K→[F]L) P) = ker_hom n ((L→[F]M).comp (K→[F]L)) P :=
by rcases P with _ | _; refl

lemma ker_hom.injective (φ : K →ₐ[F] L) : function.injective $ ker_hom n φ := by ...

def ker_gal (σ : L ≃ₐ[K] L) : E(L)[n] → E(L)[n] := λ ⟨P, hP⟩, ⟨σ · P, by ...⟩

instance : has_scalar (L ≃ₐ[K] L) E(L)[n] := ⟨ker_gal n⟩

instance : mul_action (L ≃ₐ[K] L) E(L)[n] := ⟨ker_gal n, by ..., by ...⟩

def ker_gal.fixed : add_subgroup E(L)[n] :=
⟨mul_action.fixed_points (L ≃ₐ[K] L) E(L)[n], λ _, rfl, by ..., by ...⟩

lemma ker_gal.fixed.eq [finite_dimensional K L] [is_galois K L] :
  ker_gal.fixed n E K L = (ker_hom n $ K→[F]L).range :=
by ...
```

2.3.3 Cubic discriminants

In fact, for the purposes of complete 2-descent it suffices to analyse $E[2]$, whose affine K-points have x-coordinates that are exactly the K-roots of the cubic polynomial $\psi_2(x)$. The discriminant of $\psi_2(x)$ happens to be a non-zero multiple of the discriminant Δ of E, so its roots are distinct in a splitting field precisely because $\Delta \neq 0$. Unfortunately, there was no API to work with discriminants of cubic polynomials, particularly to extract the distinct roots from a cubic polynomial with non-vanishing discriminant, so this was developed independently and is now available in mathlib at `algebra/cubic_discriminant.lean`.

```
structure cubic (R : Type) := (a b c d : R)

variables {R S : Type} [comm_ring R] [is_domain R] [semiring S] (φ : R →+* S) (P : cubic R)

def to_poly : R[X] := C P.a*X^3 + C P.b*X^2 + C P.c*X + C P.d

def equiv : cubic R ≃ {p : R[X] // p.degree ≤ 3} := by ... -- to_poly

def map : cubic S := ⟨φ P.a, φ P.b, φ P.c, φ P.d⟩

def roots : multiset R := P.to_poly.roots

def disc : R := P.b^2*P.c^2 - 4*P.a*P.c^3 - 4*P.b^3*P.d - 27*P.a^2*P.d^2 + 18*P.a*P.b*P.c*P.d
```

The actual file itself contains many trivial lemmas about coefficients and degrees, primarily to incorporate some potentially useful API for degenerate cubic polynomials whose leading coefficients may be zero, so to prove results about a non-degenerate cubic polynomial requires a proof that its leading coefficient is non-zero. For instance, fixing a ring homomorphism to a ring S where a cubic polynomial `P` has three roots `x`, `y`, and `z`, a relevant result is that its discriminant is non-zero precisely if these roots are distinct, in which case `P` can be written as a product of factors corresponding to these roots when viewed as a polynomial over S.

```
theorem eq_prod_three_roots (ha : P.a ≠ 0) (h3 : (map φ P).roots = {x, y, z}) :
  (map φ P).to_poly = C (φ P.a) * (X - C x) * (X - C y) * (X - C z) :=
by ... -- ring1

theorem disc_ne_zero_iff_roots_ne (ha : P.a ≠ 0) (h3 : (map φ P).roots = {x, y, z}) :
  P.disc ≠ 0 ↔ x ≠ y ∧ x ≠ z ∧ y ≠ z :=
by ... -- eq_prod_three_roots
```

2.3.4 2-division polynomials

Under this API, the fact that the discriminant of $\psi_2(x)$ is 16Δ is then an application of `ring1` and unfolding the definitions, which in the relevant case of $\operatorname{ch} F \neq 2$ implies that the roots of $\psi_2(x)$ are distinct.

```
def ψ₂_x : cubic K :=
⟨4, (F↑K)E.a₁^2 + 4*(F↑K)E.a₂, 4*(F↑K)E.a₄ + 2*(F↑K)E.a₁*(F↑K)E.a₃, (F↑K)E.a₃^2 + 4*(F↑K)E.a₆⟩

lemma ψ₂_x.disc_eq_disc : (ψ₂_x E K).disc = 16*E.disc := by ... -- ring1

variables [invertible (2 : F)]

lemma ψ₂_x.roots_ne {a b c : K} (h3 : (ψ₂_x E K).roots = {a, b, c}) : a ≠ b ∧ a ≠ c ∧ b ≠ c :=
by ... -- disc_ne_zero_iff_roots_ne
```

It is perhaps worth noting that the condition `invertible 2` is imposed on F rather than K, since the former implies the latter, but due to the instance resolution system it is not possible to automatically derive an instance of `invertible 2` on K without running into cycles. Several instances had to be rewritten manually as definitions, and the entire problem took quite a while to be satisfactorily resolved.

General n-division polynomials can probably be defined pretty easily by induction, but generalisations of the above results will require a good API for discriminants of general polynomials, which has yet to exist.

2.3.5 2-torsion subgroups

An important ingredient for the weak Mordell-Weil theorem will be the finiteness of $E(K)[n]$, which for complete 2-descent translates to the finiteness of $E(K)[2]$. Since the affine K-points in $E(K)[2]$ can now be characterised by polynomial identities on their coordinates, finiteness is achieved by projecting an affine K-point to the x-coordinate, then utilising the existing `fintype` instance on roots of polynomials.

```
lemma E₂_y {x y w} : some x y w ∈ E(K)[2] ↔ y = -((F↑K)E.a₁*x + (F↑K)E.a₃)/2 := by ... -- ring1

lemma E₂_x {x y w} : some x y w ∈ E(K)[2] ↔ x ∈ (ψ₂_x E K).roots := by ... -- ring1

def E₂_to_ψ₂ : E(K)[2] → option {x // x ∈ (ψ₂_x E K).roots}
| ⟨zero       , _⟩ := none
| ⟨some x y w, h⟩ := some ⟨x, E₂_x.mp h⟩

lemma E₂_to_ψ₂.injective : function.injective E₂_to_ψ₂ := by ... -- injection from definition

instance : fintype E(K)[2] := fintype.of_injective E₂_to_ψ₂ E₂_to_ψ₂.injective
```

While an injection suffices for the finiteness of $E(K)[2]$, it is easy to prove an upper bound on its cardinality and that the projection map is in fact a surjection, and hence an equivalence.

```
theorem E₂.card_le_four : fintype.card E(K)[2] ≤ 4 := by ... -- properties of cubics

lemma E₂_to_ψ₂.surjective : function.surjective E₂_to_ψ₂ := by ... -- construct explicit lift

def E₂_to_ψ₂.equiv : E(K)[2] ≃ option {x // x ∈ (ψ₂_x E K).roots} :=
equiv.of_bijective E₂_to_ψ₂ ⟨E₂_to_ψ₂.injective, E₂_to_ψ₂.surjective⟩
```

Now the finiteness of $E(K)[2]$ will suffice for the first reduction lemma in complete 2-descent [Sil09, Lemma VIII.1.2], but a preliminary attempt to reproduce the proof of the second reduction step in the literature [Sil09, Proposition VIII.1.2] did not lead very far. This involved reducing the finiteness of $E(F)/2E(F)$ to the finiteness of the splitting field of $\psi_2(x)$ over F via a Galois cohomological Kummer pairing, yet working with finiteness of degrees of extensions was far too infuriating, and the idea was ultimately scrapped in favour of explicitly writing down the complete 2-descent homomorphism. In retrospect, it is perhaps worth returning to this more conceptual proof in the future, especially once Galois cohomology and the notion of ramification are defined in mathlib. Nevertheless, some basic lemmas proven in said attempt may prove to be useful, such as the exact cardinality of $E(K)[2]$ and the trivial Galois action over the splitting field.

```
variables [algebra (ψ₂_x E F).splitting_field K] [algebra (ψ₂_x E F).splitting_field L]

theorem E₂.card_eq_four : fintype.card E(K)[2] = 4 := by ... -- properties of cubics

lemma E₂.hom.bijective : function.bijective $ ker_hom 2 $ K→[F]L := by ... -- ker_hom.injective

def E₂.hom.equiv : E(K)[2] ≃ E(L)[2] := equiv.of_bijective (ker_hom 2 $ K→[F]L) E₂.hom.bijective

lemma E₂.gal.fixed (σ : L ≃ₐ[K] L) (P : E(L)[2]) : σ • P = P := by ... -- ker_gal.fixed.eq
```

Given the above results, it is probably not too difficult to construct an injective group homomorphism $E(K)[2] \hookrightarrow (\mathbb{Z}/2\mathbb{Z})^2$ that is surjective over the splitting field. Again, with a good definition of n-division polynomials, it should not be too difficult to define $E(K)[n] \hookrightarrow (\mathbb{Z}/n\mathbb{Z})^2$ as well.

2.4 Weak Mordell-Weil theorem

The code in this section can be found in `algebraic_geometry/EllipticCurve/mordell_weil.lean` and in `algebraic_geometry/EllipticCurve/selmer.lean`, the latter of which is likely to be moved in the future.

2.4.1 Reduction lemma

With a working explicit definition of $E(K)[n]$, much of the weak Mordell-Weil theorem can be proven via complete 2-descent, but to define the homomorphism itself requires an assumption that K contains the splitting field of $\psi_2(x)$ over F, so that $E(K)[2]$ has exactly three distinct roots. The first reduction step would be to prove that this assumption is mild, in the sense that the finiteness of $E(K)/2E(K)$ is sufficient for the finiteness of $E(F)/2E(F)$. This holds for an arbitrary finite Galois extension L/K, with no difficulty introduced by replacing 2 with a general $n \in \mathbb{N}$, and is proven by considering the exact diagram

$$\begin{tikzcd}
0 \arrow[r] & \Phi \arrow[r] \arrow[d, hook, "\kappa"] & E(K)/nE(K) \arrow[r, "\iota"] \arrow[d, hook, "\kappa_K"] & E(L)/nE(L) \arrow[d, hook, "\kappa_L"] \\
0 \arrow[r] & \mathrm{H}^1(\mathrm{Gal}(L/K), E(L)[n]) \arrow[r, "\mathrm{inf}"'] & \mathrm{H}^1(K, E[n]) \arrow[r, "\mathrm{res}"'] & \mathrm{H}^1(L, E[n])
\end{tikzcd},$$

where the bottom row is the inflation-restriction exact sequence, and the vertical injections are truncated connecting homomorphisms of the long exact sequence in Galois cohomology. The required result translates to the finiteness of the kernel Φ, which follows, via the injection κ, from the finiteness of L/K by assumption and of $E(L)[n]$ by the previous section. Now Φ is easy to define, knowing that $nE(K) \leq nE(L)$.

```
variables [finite_dimensional K L] [is_galois K L] (n : ℕ)

lemma range_le_comap_range : E(K)·n ≤ add_subgroup.comap (point_hom $ K→[F]L) E(L)·n := by ...

def ι : E(K)/n →+ E(L)/n := quotient_add_group.map _ _ _ $ range_le_comap_range n

def Φ : add_subgroup E(K)/n := (ι n).ker
```

As Galois cohomology has yet to exist in mathlib, κ has to be defined explicitly by tracing the definition of the connecting homomorphisms. It turns out that $\kappa([P]) = (\sigma \mapsto \sigma \cdot Q - Q)$, where $Q \in E(L)[n]$ is a choice of a lift such that $nQ = P$ obtained by `classical.some`, but making it type-correct and proving injectivity turned out to be slightly tricky due to layers of subtypes, quotients, and existentials.

```
lemma Φ_mem_range (P : Φ n) : point_hom (K→[F]L) P.val.out' ∈ E(L)·n := by ...

def κ : Φ n → L ≃ₐ[K] L → E(L)[n] :=
λ P σ, ⟨σ · (Φ_mem_range n P).some - (Φ_mem_range n P).some, by ...⟩

lemma κ.injective : function.injective $ κ n := by ...
```

Note that this merely provides an injection from Φ into the finite set of functions $\mathrm{Gal}(L/K) \to E(L)[n]$, which is sufficient, rather than into the actual cohomology group $\mathrm{H}^1(\mathrm{Gal}(L/K), E(L)[n])$.

The reduction lemma is then immediate by the newly-written `fintype_of_ker_of_codom` that expresses finiteness over left exact sequences. This does assume that $\mathrm{ch}\, F \neq 2$ to detect the `fintype` instance of $E(L)[2]$ automatically, and is written as a definition rather than an instance to avoid cycles.

```
variables [invertible (2 : F)]

instance : fintype (Φ 2 E K) := fintype.of_injective (κ 2) $ κ.injective 2

def coker_2_of_fg_extension.fintype [fintype E(L)/2] : fintype E(K)/2 :=
add_group.fintype_of_ker_of_codom $ ι 2
```

2.4.2 Complete 2-descent

Now let F be a number field. Assuming $E[2] \subseteq E(K)$, it follows immediately by `disc_ne_zero_iff_roots_ne` that $\psi_2(x)$ has exactly three distinct K-roots $e_1, e_2, e_3 \in K$. Since $\operatorname{ch} F \neq 2, 3$, a change of variables reduces E to a short Weierstrass equation, which by `eq_prod_three_roots` can be written as

$$y^2 = (x - e_1)(x - e_2)(x - e_3).$$

The complete 2-descent homomorphism δ is then defined by generically sending

$$\begin{array}{ccc} E(K) & \longrightarrow & K^\times / \left(K^\times\right)^2 \times K^\times / \left(K^\times\right)^2 \\ (x, y) & \longmapsto & ([x - e_1], [x - e_2]) \end{array},$$

which is well-defined on affine K-points except when $x = e_1$ or $x = e_2$, in which case the undefined component is modified slightly. Note that while the short Weierstrass equation hypotheses are unused in the definition of δ, they are crucial in proving that it respects addition and in computing the kernel later.

```
variables (ha₁ : E.a₁ = 0) (ha₃ : E.a₃ = 0) (h3 : (ψ₂_x E K).roots = {e₁, e₂, e₃})

def δ.to_fun : E(K) → (units K) / (units K)^2 × (units K) / (units K)^2
| zero         := 1
| (some x y w) :=
if he₁ : x = e₁ then
  (units.mk0 ((e₁ - e₃)*(e₁ - e₂)) $ mul_ne_zero (sub_ne_zero.mpr (ψ₂_x.roots_ne h3).2.1)
                                             (sub_ne_zero.mpr (ψ₂_x.roots_ne h3).1),
  units.mk0 (e₁ - e₂) $ sub_ne_zero.mpr (ψ₂_x.roots_ne h3).1)
else if he₂ : x = e₂ then
  (units.mk0 (e₂ - e₁) $ sub_ne_zero.mpr (ψ₂_x.roots_ne h3).1.symm,
  units.mk0 ((e₂ - e₃)*(e₂ - e₁)) $ mul_ne_zero (sub_ne_zero.mpr (ψ₂_x.roots_ne h3).2.2)
                                            (sub_ne_zero.mpr (ψ₂_x.roots_ne h3).1.symm))
else
  (units.mk0 (x - e₁) $ sub_ne_zero.mpr he₁, units.mk0 (x - e₂) $ by sub_ne_zero.mpr he₂)

def δ : E(K) →+ additive ((units K) / (units K)^2 × (units K) / (units K)^2) :=
{ to_fun    := δ.to_fun ha₁ ha₃ h3,
  map_zero' := rfl,
  map_add'  := by ... }
```

An important remark here is the distinction between the additive group $E(K)$ and the multiplicative group $K^\times$, so for a homomorphism between them to be constructed, one of $E(K)$ or $K^\times$ has to be tagged as the other type. This is a known design issue in mathlib that surfaces often, say in valuation theory and in Kummer theory, with one solution being an overhaul of `monoid_hom` structures to include a binary operation.

With this definition of δ, proving that its kernel is $2E(K)$ is completely constructive, with one inclusion being obvious by `map_nsmul`. The other inclusion follows from the fact that if $P = (x, y) \in E(K)$ is such that $x - e_i = t_i^2$ and $y = t_1 t_2 t_3$ for some $t_i \in K^\times$, then by unfolding point doubling,

$$2 \cdot (x + t_1 t_2 + t_1 t_3 + t_2 t_3, (t_1 + t_2)(t_1 + t_3)(t_2 + t_3)) = P.$$

```
lemma δ.ker : (δ ha₁ ha₃ h3).ker = E(K)·2 := by ...
```

On the other hand, proving that its image is finite seems to be a lot more work. This image lies in two copies of $K(B, 2)$, which is defined in the next section and depends on a finite set of bad primes

$$B = \left\{\mathfrak{p} \in \mathrm{M}_K^0 \;\middle|\; E \text{ has bad reduction at } \mathfrak{p}\right\} \cup \left\{\mathfrak{p} \in \mathrm{M}_K^0 \;\middle|\; \mathfrak{p} \text{ divides } 2\right\}.$$

Now the notion of having bad reduction requires defining minimal Weierstrass models, but B can simply be enlarged to contain slightly more primes dividing Δ for a non-minimal Weierstrass model, so the finiteness of B is doable. However, to show that the image of δ lies in this group, the standard conceptual argument in the literature [Sil09, Proposition X.1.1] seemingly boils this down to the fact that the splitting field of $\psi_2(x)$ is unramified over F outside B, then applying Kummer theory. This would entail formalising many definitions and basic results of local fields and the action of inertia, which are far from being settled in mathlib, and likely needing a few more facts about elliptic curves over local fields.

However, an argument by Cassels over $\mathbb{Q}$ [Cas91, Theorem 15.1] leads to the following generalisation over number fields that circumvents the local theory entirely. By a change of variables, assume without loss of generality that $A, B \in \mathcal{O}_K$, so that $e_i \in \mathcal{O}_K$. Taking valuations over the short Weierstrass equation yields

$$\mathrm{ord}_{\mathfrak{p}}(x - e_1) + \mathrm{ord}_{\mathfrak{p}}(x - e_2) + \mathrm{ord}_{\mathfrak{p}}(x - e_3) \equiv 0 \mod 2, \qquad \mathfrak{p} \in \mathrm{M}_K^0,$$

so there are either zero or two of $\{e_1, e_2, e_3\}$ such that $\mathrm{ord}_{\mathfrak{p}}(x - e_i) \equiv 1 \mod 2$ by parity. An easy valuation argument shows that $\mathrm{ord}_{\mathfrak{p}}(x - e_i) \geq 0$, and in the latter case of two roots that $\mathrm{ord}_{\mathfrak{p}}(e_i - e_j) > 0$, or in other words that $\mathfrak{p} \in B$, which turns out to be the precise condition for $\mathrm{im}\,\delta \subseteq K(B, 2)^2$ as seen in the next section. Due to time constraints however, this was left as a `sorry`, but this will be the next step to complete.

```
lemma bad_primes : finset $ height_one_spectrum $ O K :=
@set.to_finset _
  {p : height_one_spectrum $ O K | p.valuation (F↑K)E.disc_unit ≠ 1 ∨ p.valuation (ℤ↑K)n < 1}
  sorry

lemma δ.range_le : (δ ha₁ ha₃ h3).range ≤ K(bad_primes E 2, 2) × K(bad_primes E 2, 2) := sorry
```

Note again that valuations of `height_one_spectrum` are multiplicative, so that `p.valuation x < 1` means $|x|_{\mathfrak{p}} < 1$ or $\mathrm{ord}_{\mathfrak{p}}\, x > 0$. In any case, combining the reduction lemma and the injection induced by the complete 2-descent homomorphism, assuming that $K(B, 2)$ is finite, proves the weak Mordell-Weil theorem.

```
def δ.lift : E(K)/2 →+ K(bad_primes E 2, 2) × K(bad_primes E 2, 2) :=
(add_subgroup.inclusion $ δ.range_le ha₁ ha₃ h3).comp $
  (quotient_add_group.range_ker_lift $ δ ha₁ ha₃ h3).comp $
  (quotient_add_group.equiv_quotient_of_eq $ δ.ker ha₁ ha₃ h3).symm.to_add_monoid_hom

lemma δ.lift.injective : function.injective $ δ.lift ha₁ ha₃ h3 := by ...

instance : fintype E(F)/2 := by ...
```

2.4.3 The Selmer group

It remains to define $K(B, 2)$ and prove that it is finite, which holds true for an arbitrary finite set of primes $S \subseteq \mathrm{M}_K^0$, again with no difficulty introduced by replacing 2 with a general $n \in \mathbb{N}$. This is the Selmer group

$$K(S, n) := \left\{[b] \in K^\times / \left(K^\times\right)^n \;\middle|\; \forall \mathfrak{p} \in S,\ \mathrm{ord}_{\mathfrak{p}}\, b \equiv 0 \mod n\right\},$$

which is a well-defined closed condition on $K^\times / (K^\times)^n$. However, precisely formalising the condition turned out to be quite tricky due to layers of the `multiplicative` tag, quotients, and units, which further required an equivalence between a group G and $(G \cup \{0\})^\times$ that is part of an equivalence of categories.

```
def group.with_zero_units {G : Type} [group G] : units (with_zero G) ≃* G :=
{ to_fun    := λ x, (with_zero.ne_zero_iff_exists.mp x.ne_zero).some,
  inv_fun   := λ x, ⟨x, x⁻¹, by ..., by ...⟩,
  left_inv  := by ...,
  right_inv := by ...,
  map_mul'  := by ... }

variables (p : height_one_spectrum $ O K)

def val_of_ne_zero : units K →* multiplicative ℤ :=
group.with_zero_units.to_monoid_hom.comp $ units.map p.valuation

def val_of_ne_zero_mod : (units K) / (units K)^n →* multiplicative (zmod n) :=
(int.quotient_zmultiples_nat_equiv_zmod n).to_multiplicative.to_monoid_hom.comp $
  quotient_group.map (units K)^n (add_subgroup.zmultiples n).to_subgroup (val_of_ne_zero p) $ by ...
```

The Selmer group can then be defined using the homomorphism `val_of_ne_zero_mod`, with a suggestive notation, and proving that it is a subgroup follows immediately from properties of kernels of homomorphisms.

```
variables (S : finset $ height_one_spectrum $ O K) (n : ℕ)

def selmer : subgroup $ (units K) / (units K)^n :=
{ carrier  := {b | ∀ p ∉ S, val_of_ne_zero_mod p b = 1},
  one_mem' := λ _ _, by rw [map_one],
  mul_mem' := λ _ _ hx hy p hp, by rw [map_mul, hx p hp, hy p hp, one_mul],
  inv_mem' := λ _ hx p hp, by rw [map_inv, hx p hp, inv_one] }

notation K(S, n) := selmer K S n
```

Now to prove that the Selmer group is finite, it suffices to prove that $K(\emptyset, n)$ is finite, as it is by definition the kernel of the natural homomorphism, denoted by `selmer.val` in code, given by

$$\begin{array}{cccc} f & : & K(S,n) & \longrightarrow & (\mathbb{Z}/n\mathbb{Z})^{\#S} \\ & & [b] & \longmapsto & (\operatorname{ord}_{\mathfrak{p}} b)_{\mathfrak{p}\in S} \end{array}.$$

```
def selmer.val : K(S, n) →* S → multiplicative (zmod n) :=
{ to_fun   := λ b p, val_of_ne_zero_mod p b,
  map_one' := funext $ λ p, map_one $ val_of_ne_zero_mod p,
  map_mul' := λ x y, funext $ λ p, map_mul (val_of_ne_zero_mod p) x y }

lemma selmer.val_ker : selmer.val.ker = K(∅, n).subgroup_of K(S, n) := by ...
```

The finiteness of $K(\emptyset, n)$ in turn relies on two fundamental theorems in algebraic number theory, namely the finiteness of the class group Cl_K and the finite generation of the unit group $\mathcal{O}_K^\times$. The former was formalised in mathlib only rather recently [BDNN21], while the latter is part of Dirichlet's unit theorem, which has yet to be formalised but should be doable with the currently available machinery. Assuming this `sorry`, a trivial consequence of the structure theorem for finitely generated abelian groups, which was merged into mathlib literally a few days ago, yields the finiteness of $\mathcal{O}_K^\times / \left(\mathcal{O}_K^\times\right)^n$ for $n > 0$.

```
instance : group.fg (units $ O K) := sorry

instance [fact $ 0 < n] : fintype $ (units $ O K) / (units $ O K)^n := by ...
```

Along with Cl_K, they nest $K(\emptyset, n)$ in a left exact sequence

$$0 \to \mathcal{O}_K^\times / \left(\mathcal{O}_K^\times\right)^n \xrightarrow{g} K(\emptyset, n) \xrightarrow{h} \mathrm{Cl}_K .$$

Now g is simply the quotient map of the canonical inclusion $\mathcal{O}_K^\times \hookrightarrow K^\times$, but to map into the subtype $K(\emptyset, n)$ rather than all of $K^\times / (K^\times)^n$, this was broken into a `selmer.from_unit` map $\mathcal{O}_K^\times \hookrightarrow K(\emptyset, n)$ and a kernel computation. The latter is merely a double-sided inclusion, one being trivial and the other required the assumption $n > 0$ to prove that $x^n \in \mathcal{O}_K$ implies that $x \in \mathcal{O}_K$ via standard facts on integral closure.

```
def ne_zero_of_unit : units (O K) →* units K :=
{ to_fun   := λ ⟨⟨v, _⟩, ⟨i, _⟩, vi, iv⟩, ⟨v, i, by injection vi, by injection iv⟩,
  map_one' := rfl,
  map_mul' := λ ⟨⟨_, _⟩, ⟨_, _⟩, _, _⟩ ⟨⟨_, _⟩, ⟨_, _⟩, _, _⟩, rfl }

def selmer.from_unit : units (O K) →* K(∅, n) :=
{ to_fun   := λ x, ⟨quotient_group.mk $ ne_zero_of_unit x, by ...⟩,
  map_one' := rfl,
  map_mul' := λ ⟨⟨_, _⟩, ⟨_, _⟩, _, _⟩ ⟨⟨_, _⟩, ⟨_, _⟩, _, _⟩, rfl }

lemma selmer.from_unit_ker [fact $ 0 < n] : selmer.from_unit.ker = (units (O K))^n := by ...
```

On the other hand, h, or rather `selmer.to_class` in code, is slightly more complicated to define even mathematically. Given a representative x of a class in $K(\emptyset, n)$, its principal fractional ideal $\langle x \rangle$ necessarily factorises into the n-th power of a fractional ideal $\mathfrak{x}$ by unique factorisation into prime ideals, whose class lands in Cl_K. Then h is defined to map the class $[x] \in K(\emptyset, n)$ to the class $[\mathfrak{x}] \in \mathrm{Cl}_K$. The two layers of quotients coupled with a currently weak API for unique factorisation in Dedekind domains made h quite tedious to formalise, let alone exactness at $K(\emptyset, n)$. For instance, a preliminary attempt on a direct proof that $\langle x \rangle$ is an n-th power resulted in a mathematically-accurate set-theoretic function h, but proving that it is a homomorphism was incredibly tricky due to having different lifts for $[x]$.

```
def fractional_ideal.units_of_factorization (f : height_one_spectrum (O K) → ℤ) :
  (units $ fractional_ideal (non_zero_divisors $ O K) K) :=
units.mk0 (finprod (λ p : height_one_spectrum $ O K, p.as_ideal ^ f p)) $ by ...

lemma selmer.val_exists (x : K(∅, n)) : ∃ z : ℤ, z * n = -(val_of_ne_zero p x.val.out').to_add :=
by ...

def selmer.to_class.to_fun (x : K(∅, n)) : class_group (O K) K :=
quotient_group.mk' (to_principal_ideal (O K) K).range $
  fractional_ideal.units_of_factorization $ λ p, (selmer.val_exists p x).some
```

A more elegant solution was to slightly modularise the definition of h, by first proving a similar lemma `selmer.val_exists_of_mk` for an element $x \in K^\times$, then the lemma `selmer.to_class_of_mk` that it is well-defined, in the sense that taking its class and choosing a different lift computes the same expected output, which made the actual code somewhat longer but more comprehensible.

```
lemma selmer.val_exists_of_mk {x : units K} (hx : quotient_group.mk x ∈ K(∅, n)) :
  ∃ z : ℤ, z * n = -(val_of_ne_zero p x).to_add :=
by ...

lemma selmer.to_class_of_mk [fact $ 0 < n] {x : units K} (hx : quotient_group.mk x ∈ K(∅, n)) :
  selmer.to_class.to_fun ⟨quotient_group.mk x, hx⟩
    = quotient_group.mk' (to_principal_ideal (O K) K).range
      (fractional_ideal.units_of_factorization $ λ p, (selmer.val_exists_of_mk p hx).some) :=
by ...

def selmer.to_class [fact $ 0 < n] : K(∅, n) →* class_group (O K) K :=
{ to_fun   := selmer.to_class.to_fun,
  map_one' := by ...,
  map_mul' := by ... }

lemma selmer.to_class_ker [fact $ 0 < n] : selmer.to_class.ker = selmer.from_unit.range := by ...
```

It is worth noting that the proofs of `selmer.to_class_of_mk` and `selmer.to_class_of_ker` crucially use the `fractional_ideal.lean` file in Maria Inés de Frutos-Fernandez's formalisation of adèles and idèles of global fields [Fru22], primarily the `fractional_ideal.factorization_principal` lemma that expresses a principal ideal as a `finprod` over all primes, which is in the process of being incorporated into mathlib.

Once g and h are defined and the relevant sequence is shown to be left exact, finiteness of $K(\emptyset, n)$, and hence of all Selmer groups, is a matter of applying `fintype_of_ker_of_codom` to f and h.

```
def selmer.fintype [fact $ 0 < n] : fintype K(∅, n) :=
group.fintype_of_ker_of_codom $ selmer.to_class n

instance [fact $ 0 < n] : fintype K(S, n) := group.fintype_of_ker_of_codom $ selmer.val S n
```

Note that the actual code supplies the `fintype` proofs in hidden arguments that cannot be inferred. It is worth emphasising that the finiteness of Selmer groups is a result in pure algebraic number theory, and can be incorporated into mathlib a lot sooner than the arithmetic of elliptic curves once Dirichlet's unit theorem is formalised. Defining Selmer groups independently also has the added benefit of reusability, even if complete 2-descent were to be scrapped in the future for a more conceptual Galois cohomological proof.

2.5 Height functions

The code in this section can be found in `algebraic_geometry/EllipticCurve/heights.lean` in the main branch, and in `algebraic_geometry/height_function.lean` in the `descent_theorem` branch.

2.5.1 Naive heights

The remaining ingredient to the Mordell-Weil theorem is a coherent definition of heights on elliptic curves. The standard theory in the literature [Sil09, Section VIII.6] involves first defining a relative height function on $\mathbb{P}^1_K$ that restricts to E via a surjective morphism $f : E \to \mathbb{P}^1_K$, then considering its absolute version on $\mathbb{P}^1_{\overline{\mathbb{Q}}}$ and taking logarithms. By choosing $f = x$, this results in a function defined on affine K-points by

$$\begin{array}{rcll} h & : & E(K) \setminus \{\mathcal{O}\} & \longrightarrow & \mathbb{R}_{>0} \\ & & (x,y) & \longmapsto & \dfrac{1}{[K:\mathbb{Q}]} \displaystyle\sum_{v \in \mathrm{M}_K} [K_v : \mathbb{Q}_v] \log \max\left(|x|_v, |1|_v\right) \end{array},$$

requiring that the places $v \in \mathrm{M}_K$ have normalised absolute values, and setting $h(\mathcal{O}) := 0$. Notice that h is normalised with local and global degrees to satisfy a certain extension formula, which requires a robust theory of local field extensions to prove, but the three properties of heights remain true without these scaling factors for any particular K, so they can be safely omitted. In contrast, to show that h is well-defined irrespective of homogeneous coordinates requires the product formula, which should be relatively doable. Due to time constraints however, only the special case of $K = \mathbb{Q}$ is formalised, since the three properties can be proven with explicit inequalities that are not too cumbersome, and h becomes the naive height

$$\begin{array}{rcll} \mathrm{h} & : & E(\mathbb{Q}) \setminus \{\mathcal{O}\} & \longrightarrow & \mathbb{R}_{>0} \\ & & (x,y) & \longmapsto & \log \max\left(|p|_\infty, |q|_\infty\right) \end{array},$$

where $x = p/q$ is written in lowest terms. Note that this possible since $\mathbb{Z}$ is a GCD domain, while an arbitrary ring of integers $\mathcal{O}_K$ may not be, which justifies the complication in the definition of h.

```
def height : E(ℚ) → ℝ
| zero          := 0
| (some x _ _) := (max (|x.num|) (|x.denom|) : ℝ).log
```

2.5.2 Properties of heights

In what follows, as done previously, assume a short Weierstrass model with $A, B \in \mathbb{Z}$.

The finiteness of $\{P \in E(\mathbb{Q}) \mid \mathrm{h}(P) \le C\}$, sometimes called the Northcott property, is proven by observing that there are finitely many choices for the numerator and the denominator of the x-coordinate of a $\mathbb{Q}$-point, which boils down to a very explicit embedding into a product of types known to be finite.

```
def height_le_constant.fun {C : ℝ} :
  {P : E(ℚ) // height P ≤ C} → option (fin (2*C.exp.floor + 1) × fin (C.exp.floor + 1) × fin 2)
| ⟨zero       , _⟩ := none
| ⟨some x y w, h⟩ := some ⟨(x.num + C.exp.floor).to_nat, x.denom, if y ≤ 0 then 0 else 1⟩

variables (ha₁ : E.a₁ = 0) (ha₃ : E.a₃ = 0)

lemma height_le_constant.injective {C : ℝ} :
  function.injective $ @height_le_constant.function E C :=
by ...

def height_le_constant.fintype (C : ℝ) : fintype {P : E(ℚ) // height P ≤ C} :=
fintype.of_injective height_le_constant.fun $ height_le_constant.injective ha₁ ha₃
```

Here, the embedding maps the numerator into integers between $-\lfloor e^C \rfloor$ and $\lfloor e^C \rfloor$, and as such has be translated into non-negative integers to reuse the `fintype` instance of the `fin` type. Note that its definition does not require the short Weierstrass equation hypotheses, but proving injectivity for the y-coordinate does.

The remaining two properties are essentially consequences of a certain parallelogram law on the canonical height, expressing $\mathrm{h}(P+Q)$ in terms of $\mathrm{h}(P)$ up to some controllable constant. Their proofs are essentially huge case bashes, each specifying the constant explicitly and involves lengthy manipulations of inequalities, which ended up being about 600 lines in total, so they shall not be described in detail here.

The only worthwhile remark is in the proof of the first property $\mathrm{h}(P+Q) \le 2\mathrm{h}(P) + C_1$, which crucially uses properties of `padic_val_rat` to express an affine $\mathbb{Q}$-point (x, y) as $\left(a/d^2, b/d^3\right)$ for some $a, b, d \in \mathbb{Z}$ such that $(a, d) = (b, d) = 1$. This turned out to be quite tedious, as the API for `padic_val_rat` was missing a lot of boilerplate, but more annoyingly because its definition requires the rational number in question to be non-zero for it to be mathematically-accurate, so cases had to be opened at every step, in stark contrast to `height_one_spectrum.valuation` with codomain `with_zero (multiplicative Z)` to avoid the extra layer of cases. Perhaps the current disconnect between the older `padics` library and the newly-developed `adic_valuation` library should be resolved by redefining the former as a special case of the latter.

```
lemma padic_val_point.denom {A B : ℤ} {x y : ℚ} (w : y ^ 2 = x ^ 3 + A * x + B) :
  ∃ n : ℕ, x.denom = n ^ 2 ∧ y.denom = n ^ 3 :=
by ...
```

Assuming this, the constant for an affine $\mathbb{Q}$-point $Q = \left(a/d^2, b/d^3\right)$ can then be given explicitly by

$$\begin{aligned} C_1 := \max\{&2\mathrm{h}(Q), \log\left(d^4 + 2|a|\,d^2 + a^2\right), \\ &\log\left(|a|\,d^2 + \left|a^2 + Ad^4\right| + \left|Aad^2 + 2Bd^4\right| + 2|bd|\sqrt{1 + |A| + |B|}\right)\}. \end{aligned}$$

```
lemma exists_constant_height_add_le (Q : E(ℚ)) :
  ∃ C : ℝ, ∀ P : E(ℚ), height (P + Q) ≤ 2 * height P + C :=
by ...
```

The second property $4\mathrm{h}(P) \le \mathrm{h}(2P) + C_2$ does not require p-adic analysis, but requires bounding the cancellation between the numerator and the denominator of the x-coordinate of an affine K-point, which boils down to checking a few polynomial identities. The constant can then be given explicitly by

$$\begin{aligned} C_2 := \max\{&4\max\{\mathrm{h}(Q) \mid Q \in E(\mathbb{Q})[2]\}, \log 2\,(12 + 16|A|), \log 2\,(3 + 5|A| + 27|B|), \\ &\log 8\left(\left|4A^3 + 27B^2\right| + A^2|B| + |A|\left|3A^3 + 22B^2\right| + 3|B|\left|A^3 + 8B^2\right|\right), \\ &\log 2\left(A^2|B| + |A|\left|5A^3 + 32B^2\right| + 2|B|\left|13A^3 + 96B^2\right| + 3A^2\left|A^3 + 8B^2\right|\right)\}. \end{aligned}$$

```
lemma exists_constant_le_height_dbl : ∃ C : ℝ, ∀ P : E(ℚ), 4 * height P ≤ height (2 · P) + C :=
by ...
```

2.5.3 Descent theorem

The final ingredient to the Mordell-Weil theorem is the descent theorem [Sil09, Theorem VIII.3.1], which was formalised entirely by Jujian Zhang, independently and prior to this project, in the `descent_theorem` branch of mathlib. Zhang assumes the existence of a structure `height_function` whose fields are precisely the three properties of heights, and subsequently derives the proof of the theorem `descent`.

```
variables (A : Type) [add_comm_group A]

structure height_function :=
(to_fun : A → ℝ) (nonneg : ∀ P : A, 0 ≤ to_fun P) (C₁ : A → ℝ) (C₁_pos: ∀ a : A, 0 < C₁ a)
(height_add_le : ∀ P Q : A, to_fun (P + Q) ≤ 2 * to_fun P + C₁ Q) (m : ℕ) (hm : 2 ≤ m)
(C₂ : ℝ) (C₂_pos : 0 < C₂) (height_nsmul_ge : ∀ P : A, m^2 * to_fun P ≤ to_fun (m · P) + C₂)
(finite : ∀ C₃ : ℝ, {P : A | to_fun P < C₃}.finite)

theorem descent : A.fg := by ...
```

Zhang's actual definition of `height_function` has a few other differences from the naive height function defined in the previous section, but they are all minor and are essentially equivalent.

3 Future

This section concludes the report with a summary on the progress so far, a list of suggestions for future potential projects related to elliptic curves and the Mordell-Weil theorem, and a final personal remark.

3.1 Current progress

The overall strategy of the proof of the Mordell-Weil theorem is mostly settled, but there is currently a disconnect between the three parts that were mostly written independently of each other. Bridging these gaps should not be too difficult, but the current repository also contains a few `sorry` lemmas.

- In `point.lean`, the only block is a proof of the associativity of the group law. This will likely either be a brute-force approach given enough time, or an identification with $\mathrm{Pic}^0_{E/F}(K)$, probably under a restricted definition that works only for Weierstrass models of elliptic curves.
- In `mordell_weil.lean`, the proof that $\operatorname{im}\delta \subseteq K(B,2)^2$ and the accompanying definition of bad primes are currently being worked on. The other missing component is the bridge between weak and full Mordell-Weil via heights, which will require a well-developed theory over K rather than just over $\mathbb{Q}$.
- In `selmer.lean`, the only missing result is the finite generation part of Dirichlet unit's theorem, which can be delegated as its own project. Over time, there may be algebraic lemmas with a temporary `sorry` sprinkled throughout the file as a result of actively merging other parts of the file into mathlib.

All other relevant files, including `heights.lean` for naive heights over $\mathbb{Q}$, are `sorry`-free.

3.2 Potential projects

Formalising the basics of elliptic curves and a primitive proof of the Mordell-Weil theorem opens up several other potential formalisation work. The following are some ideas in roughly increasing accessibility.

- Generalise the definition of $\psi_2(x)$ to $\psi_n(x)$ by induction, construct an injective group homomorphism $E(K)[n] \hookrightarrow (\mathbb{Z}/n\mathbb{Z})^2$ surjective over the splitting field, and define the Tate module.
- Explore the theory over finite fields, such as proving the Hasse-Weil bound, defining the Hasse-Weil zeta-function, and perhaps verifying the Weil conjectures for elliptic curves.
- Verify the correctness of algorithms, such as Schoof or Lenstra, and of some cryptographic protocols.
- Explore the theory over local fields, such as developing formal groups abstractly, defining reduction types, proving that $E(K)[n] \hookrightarrow \widetilde{E}_{\mathfrak{p}}(\kappa_{\mathfrak{p}})$ for good primes $\mathfrak{p} \in \mathrm{M}^0_K$, and perhaps proving the Néron-Ogg-Shafarevich criterion. This will require the ramification theory of local fields.
- Define $\mathrm{Sel}(K, E[n])$ and the Tate-Shafarevich group, and reprove the weak Mordell-Weil theorem more conceptually by embedding $E(K)/2E(K) \hookrightarrow \mathrm{Sel}(K, E[n])$. This will require Galois cohomology.
- Redefine elliptic curves using the Proj construction, and reprove all results using this definition. This will require much algebraic geometry, particularly the Riemann-Roch theorem.
- Explore the theory over global function fields, such as reproving the Mordell-Weil theorem, defining elliptic surfaces and the Néron-Severi group, and perhaps stating the Birch and Swinnerton-Dyer conjecture over function fields. This will again require much algebraic geometry.
- Explore the complex theory, such as defining elliptic functions and integrals, proving the uniformisation theorem, defining modular curves, and perhaps stating some version of the modularity theorem and some form of modular parameterisation. This will require a very robust theory of modular forms.

3.3 Concluding retrospect

On hindsight, given the severe lack of algebraic geometry in mathlib and the timeline for the project, one cannot say that elliptic curves have been truly formalised, but trying to prove the Mordell-Weil theorem remains a good stress test for the usability and compatibility of various mathlib libraries.

References

[BDNN21] A. Baanen, S. Dahmen, A. Narayanan and F. Nuccio. *A formalization of Dedekind domains and class groups of global fields*. 2021.

[Bel21] J. Bell. *Formalising the BSD conjecture*. 2021.

[BS14] E.-I. Bartzia and P.-Y. Strub. *A formal library for elliptic curves in the Coq proof assistant*. 2014.

[Cas91] J. W. S. Cassels. *Lectures on elliptic curves*. 1991.

[Fri98] S. Friedl. *An elementary proof of the group law for elliptic curves*. 1998.

[Fru22] M. I. de Frutos-Fernandez. *Adèles and idèles of global fields*. 2022.

[HT07] G. Hanrot and L. Théry. *Primality proving with elliptic curves*. 2007.

[KM85] N. Katz and B. Mazur. *Arithmetic moduli of elliptic curves*. 1985.

[Lea] Lean Community. *mathlib*. URL: `https://leanprover-community.github.io/`.

[Mic] Microsoft Research. *Lean theorem prover*. URL: `https://leanprover.github.io/`.

[Sil09] J. Silverman. *The arithmetic of elliptic curves*. 2009.