

# Proving Gauss's Lemma in Lean

Aditya Agarwal

Australian National University

March 1, 2019

# Outline

Lean

Dependent Type Theory

mathlib

Gauss's Lemma

The Proof in Lean

# What is Lean?

- ▶ Lean is an Automated/Interactive theorem prover.
- ▶ Other examples include HOL and Coq.
- ▶ You encode proofs in a formal language and Lean checks it's correctness
- ▶ Lean can also infer some of the details of the proof for you.

# Proofs in Lean

There are two ways of writing proofs in Lean, Term mode and Tactic mode.

# Term Mode

In term mode you explicitly write down every step of the proof.

Example:

```
theorem zero_id (x : N) : plus z x = x :=
  N.rec_on x
    (show plus z z = z, by refl)
    (assume x,
      assume ih: plus z x = x,
      show plus z (s x) = (s x), from calc
        plus z (s x) = s (plus z x) : by refl
        ... = s x : by rw ih)
```

# Tactics Mode

Tactics mode allows you to use “tactics”, which direct Lean on how it should work towards the next step.

Example:

```
lemma not_const_imp_non_zero : ¬is_const p → p ≠ (0 : polynomial α)
mt begin
| intro hp,
| show is_const p, by {rw [hp, is_const], simp}
end
```

# Dependent Type Theory

Lean is built on Dependent Type Theory, an alternative foundation of mathematics. Specifically, it implements the Calculus of Inductive Constructions.

# Dependent Type Theory

- ▶ Each object has a type.
- ▶ E.g.
  - ▶  $n : \mathbb{N}$
  - ▶  $p \iff p : Prop$
- ▶ This type may depend on a parameter such as
  - ▶  $\text{list } \alpha$
- ▶ The type of this object is fixed



# Mathlib

Mathlib is the mathematical components library for Lean.

- ▶ It contains the very beginnings of Algebra, Analysis, Topology, and Category Theory.
- ▶ On the Algebra side, it contains basic definitions about Groups, Rings, Modules etc.
- ▶ And properties about Integral Domains, Euclidean Domains, Unique Factorisation Domains etc.
- ▶ Galois Theory is missing entirely.

# Mathlib

Generally, mathlib implements theories in the most abstract setting possible.

For example, polynomial is defined as

```
def polynomial (α : Type*) [comm_semiring α] := N →0 α
```

- ▶ This is motivated by the desire for code reuse.
- ▶ E.g. We know that any ordered finite set has a maximum. This property can be used wherever ordered finsets come up. like to argue that there is a coefficient of maximum degree.
- ▶ Alternative implementations like lists of coefficients would require a separate proof about the fact.
- ▶ Similarly, it provides a uniform API between components of the library.
- ▶ However, this does have the disadvantage that definitions can become non-intuitive.

# Gauss's Lemma

## Theorem

*Let  $\alpha$  be a Unique Factorisation Domain.*

*Let  $p$  be a polynomial with coefficients in  $\alpha$ . Then  $p$  factors in  $\alpha$  if and only if it factors in  $\text{Frac}(\alpha)$ .*

# Gauss's Lemma

## Definition

Let  $p$  be a polynomial. We say  $p$  is primitive if the only constants in  $\alpha$  which divide  $p$  are units.

## Lemma (Gauss's Primitive Polynomial Lemma)

*Let  $p, q$  be primitive polynomials. Then  $pq$  is a primitive polynomial.*

# Gauss's Primitive polynomial Lemma

## Proof.

Assume for a contradiction that  $pq$  is not primitive. Then we have some  $c \in \alpha$  such that  $c$  is not a unit and  $c \mid pq$ .

$\alpha$  is a UFD, so  $c$  has some irreducible factor that divides  $pq$ . So WLOG, we may assume  $c$  is irreducible, and hence prime.

$\therefore \alpha/(c)$  is a domain.

$\therefore \alpha/(c)[x]$  is a domain.

$c \mid pq$  so  $pq$  vanishes in  $\alpha/(c)[x]$ .

As  $\alpha/(c)[x]$  Integral Domain,  $p$  vanishes or  $q$  vanishes.

$\therefore c \mid p$  or  $c \mid q$ .

$\therefore$  Either  $p$  or  $q$  is not primitive.

Contradiction. □

# Gauss's Lemma ( $\implies$ )

## Proof.

Let  $p$  be an irreducible in  $\alpha$ . Any irreducible term must be primitive, so we know  $p$  is primitive.

Suppose for a contradiction that  $p$  is not irreducible in  $\text{Frac}(\alpha)$ .

Then  $p = ab$ , for some  $a, b \in \text{Frac}(\alpha)[x]$ .

We may multiply  $a$  and  $b$  by some  $c_1, c_2 \alpha$  so that  $c_1 a, c_2 b$  have  $\alpha$  coefficients.  $c_1 c_2 p = c_1 a c_2 b$ . We can factor  $c_1 a$  to some  $c'_1 a'$  and  $c_2 b$  to  $c'_2 b'$ , so that  $a'$  and  $b'$  are primitive.



Proof.

(contd.)

Hence  $p = \frac{c'_1 c'_2}{c_1 c_2} a' b'$ .

If  $\frac{c'_1 c'_2}{c_1 c_2}$  in  $\alpha$ , then we have produced a factorisation for  $p$ . A contradiction.

If  $\frac{c'_1 c'_2}{c_1 c_2}$  is not a unit in  $\alpha$ , then the denominator in reduced form of  $\frac{c'_1 c'_2}{c_1 c_2}$  must divide each coefficient of  $a' b'$ , because  $p = \frac{c'_1 c'_2}{c_1 c_2} a' b'$  has  $\alpha$  coefficients.

Hence  $a' b'$  is not primitive. Contradiction.



# Proof in Lean

I took two broad approaches towards implementing the proof in Lean.

- ▶ Proof Above
- ▶ A proof using the notion of content.

The content is defined as the ideal generated by all the coefficients of  $p$ .

The content proof is broadly similar to the proof above, but by using properties of Ideals, it avoids the need for Unique Factorisation (Just the presence of a GCD).

However, Lean doesn't have a decent API for dealing with  $R$ -linear combinations, so I abandoned it.



# Defining the Lemma