

Formalization in Lean

Clea Bergsman, Katherine Buesing, Sahan Wijetunga, Mentor:
George McNinch

VERSEIM REU

06/12/2025

Lean Basics

- 1 An **expression** in lean is essentially a mathematical expression converted into computer code, ex: numbers, functions, sets
- 2 Every expression in lean has a **type**. Examples of types:
 - natural numbers \mathbb{N}
 - functions $\mathbb{R} \rightarrow \mathbb{R}$
 - sets with elements in \mathbb{R}

```
3  #check N
4  #check ℝ → ℝ
5  #check Set ℝ
```

Lean Syntax

```
7  #check (2: N)
8  #check (fun (x: R) => x^2)
9  #check {x: R | x ≥ 0}
10
11
12
```

► Expected type

▼ Messages (1)

▼ presentation.lean:7:0

2 : N

Lean Syntax

```
7  #check (2: N)
8  #check (fun (x: R) => x^2)
9  #check {x: R | x ≥ 0}
10
11
12
```

► Expected type

▼ Messages (1)

▼ presentation.lean:8:0

fun x ↦ x ^ 2 : R → R



Lean Syntax

```
7  #check (2: N)
8  #check (fun (x: R) => x^2)
9  #check {x: R | x ≥ 0}
10
```

▼ Messages (1)

▼ presentation.lean:9:0

$\{x \mid x \geq 0\} : \text{Set } \mathbb{R}$

Propositions

- ① A **proposition** is a mathematical statement converted to computer code
- ② **Prop** is a type
- ③ Each proposition is an expression of type **Prop**
- ④ Examples of propositions:
 - $2 + 2 = 4$
 - $a < b$
 - $\forall x, \exists y$ such that $2x = y$

Proofs

Expressions of type P where P is of type *Prop* are proofs of P .

- 1 Expressions of type $1 < 2$ prove that $1 < 2$.
- 2 Expressions of type $x + y = y + x$ prove that $x + y = y + x$.

```
✓ 23   example: x+y=y+x :=  
    24   | add_comm x y  
    25
```


Lean proof example: $x * x = x^2$

```
example (x : ℤ) : x*x = x^2 := by
|   rw [ pow_two ]
```

1 goal

```
x : ℤ
⊢ x * x = x ^ 2
```

pow_two definition: $\vdash \forall \{M : \text{Type } u_2\} [\text{inst} : \text{Monoid } M] (a : M), a ^ 2 = a * a$

Lean proof example: $z * x * y * w = z * y * x * w$

```
example (x y z w : ℕ) : z * x * y * w = z * y * x * w := by
```

```
  rw [mul_assoc z x y] Step 1
```

```
  rw [mul_assoc z y x] Step 2
```

```
  rw [mul_comm x y] Step 3
```

Step 1

```
x y z w : ℕ
```

```
⊢ z * (x * y) * w = z * y * x * w
```

Step 2

```
x y z w : ℕ
```

```
⊢ z * (x * y) * w = z * (y * x) * w
```

Step 3

```
x y z w : ℕ
```

```
⊢ z * (y * x) * w = z * (y * x) * w
```

```
example (x y z w : ℕ) : z * x * y * w = z * y * x * w := by
```

```
  ring
```

"Human" Proof of odd function times an odd function equals an even function

- 1 Let $f(x)$ and $g(x)$ be odd functions, so that $f(x) = -f(-x)$ and $g(x) = -g(-x)$. Let $h(x) = f(x) * g(x)$.
- 2 We want to show that $h(x)$ is even, or in other words, $h(x) = h(-x)$.
- 3 We have $h(x) = f(x) * g(x)$.
- 4 Using the definition of an odd function,
 $h(x) = -f(-x) * -g(-x) = f(-x) * g(-x)$.
- 5 We know $h(-x) = f(-x) * g(-x)$, so $h(x) = h(-x)$.
- 6 Thus, $h(x)$ is an even function.

Our definition of even and odd functions

```
def FnEven (f : ℝ → ℝ) : Prop :=  
  ∀ x, f x = f (-x)  
  
def FnOdd (f : ℝ → ℝ) : Prop :=  
  ∀ x, f x = -f (-x)
```

Odd/Even Functions Proof

Lean proof of odd function times an odd function equals an even function

```
example (of : FnOdd f) (og : FnOdd g) : FnEven fun x ↦ f x * g x := by
  intro x
  calc
    (fun x ↦ f x * g x) x = f x * g x := rfl
    _ = -f (-x) * -g (-x) := by rw [of, og]
    _ = f (-x) * g (-x) := by ring
```

Original Goal

```
f g : ℝ → ℝ
of : FnOdd f
og : FnOdd g
⊢ FnEven fun x ↦ f x * g x
```

After of

```
f g : ℝ → ℝ
of : FnOdd f
og : FnOdd g
x : ℝ
⊢ -f (-x) * g x = -f (-x) * -g (-x)
```

After intro x

```
f g : ℝ → ℝ
of : FnOdd f
og : FnOdd g
x : ℝ
⊢ ((fun x ↦ f x * g x) x = (fun x ↦ f x * g x) (-x))
```

After og

```
f g : ℝ → ℝ
of : FnOdd f
og : FnOdd g
x : ℝ
⊢ -f (-x) * -g (-x) = -f (-x) * -g (-x)
```

Alternative lean proof of odd function times an odd function equals an even function

```
0 example {f g} (of : FnOdd f) (og : FnOdd g) : FnEven fun x ↦ f x * g x := by
1   intro x₀
2   dsimp
3   rw[of, og]
4   exact neg_mul_neg (f (-x₀)) (g (-x₀))
```

1 goal

```
0 f g : ℝ → ℝ
  of : FnOdd f
  og : FnOdd g
  ⊢ FnEven fun x ↦ f x * g x
```

1 goal

```
1 f g : ℝ → ℝ
  of : FnOdd f
  og : FnOdd g
  x₀ : ℝ
  ⊢ (fun x ↦ f x * g x) x₀ =
    (fun x ↦ f x * g x) (-x₀)
```

1 goal

```
2 f g : ℝ → ℝ
  of : FnOdd f
  og : FnOdd g
  x₀ : ℝ
  ⊢ f x₀ * g x₀ = f (-x₀) * g (-x₀)
```

1 goal

```
3 f g : ℝ → ℝ
  of : FnOdd f
  og : FnOdd g
  x₀ : ℝ
  ⊢ -f (-x₀) * -g (-x₀) = f (-x₀) * g (-x₀)
```

Potential applications

- ① Forms over finite fields
- ② Quaternion algebra
- ③ Graph theory
- ④ The Grassmannian

What's the point?

- 1 The transformation from "human mathematics" to "formalized mathematics" removes inferences from proofs, continuously checking that proofs are correct
- 2 **mathlib** facilitates collaboration between mathematicians and allows proofs to be viewed by hundreds of scholars and students
- 3 Proofs can be continuously modified and maintained by other mathematicians rather than becoming incompatible with other developments in Lean
- 4 New methods to formalize a proof may have important pedagogical benefits

References

- ① Browning, T., & Lutz, P. (2022). Formalizing Galois Theory. *Experimental Mathematics*, 31(2), 413-424.
- ② Avigad, J. Buzzard, K. Lewis R. Y. Massot, P. (2020). *Mathematics in Lean*.

Thank you!

Special thanks to Dr. George McNinch and the REU