Day - 4 : Using Existing Lemmas

Touseef Haider

May 15, 2025

Study Mathematics In Lean(MIL) Section 2.2

4.1. Lean's Vast Library of Lemmas

Lean, especially when paired with its standard mathematical library **Mathlib**, has a *huge* collection of pre-proven theorems, called **lemmas**. Mathlib contains tens of thousands of lemmas about numbers theory, algebra, topology, analysis, and much more.

Properties like:

- `a + b = b + a` (commutativity of addition)
- `(a + b) + c = a + (b + c)` (associativity of addition)
- `a * (b + c) = a * b + a * c` (distributivity)

...and countless others are already proven and waiting for you to use.

The skill, then, becomes:

1. Recognizing what mathematical property your goal represents.
2. Knowing how to *find* the name of the lemma in Lean that corresponds to that property.

How to Find Lemmas :

This is a bit of an art and a science, and it gets easier with experience. Here are common ways:

1. **Guessing Standard Names**: For very common properties, the names are often predictable:
   - `add_assoc` for associativity of addition.
   - `add_comm` for commutativity of addition.
   - `mul_assoc` for associativity of multiplication.
   - `zero_add`, `add_zero`, `one_mul`, `mul_one`, etc.
   - The type usually comes first, e.g., `Nat.add_assoc`, `Int.mul_comm`, `Real.sqrt_le_sqrt_iff_le`.
2. **Mathlib Documentation**: If you're using Mathlib, the developers maintain extensive online documentation. You can search this documentation for keywords.
   - For example, searching the Mathlib4 docs for "associativity of addition" would quickly lead you to `Nat.add_assoc` or a more general `add_assoc`.
3. **The exact? Tactic (or similar)**: This is a very powerful tool!
   - In Lean (often via Mathlib), there's a tactic usually called `exact?` and `apply?`
   - If you have a goal like ⊢ `(x + y) + z = x + (y + z)`, and you're stuck, you can try:

```
theorem add_assoc_practice (x y z : Nat) : (x + y) + z = x + (y + z) := by
  exact? -- Or try `apply?`
```

- Lean will then search its library for a known lemma to prove the goal. If it finds one (like `Nat.add_assoc`), it might suggest `exact Nat.add_assoc` or `rw [Nat.add_assoc]`.

4. **VS Code Infoview / Autocompletion**: As you type, especially within `rw [...]`, Lean's extension may offer name suggestions like `Nat.add_...`, which help narrow the options.

5. **Looking at Similar Proofs**: When learning, reviewing how other theorems are proven in Mathlib can show you which lemmas are commonly used.

Specific Case:

- The goal `(x + y) + z = x + (y + z)` as "associativity of addition."
- A good guess for the lemma name would involve `add` and `assoc`.
- You'd then try `Nat.add_assoc` (since `x,y,z` are `Nat`).
- If unknown, you could use `exact?`, or `apply?`.

4.2. Finding and Stating an Existing Lemma (e.g., `add_comm`)

Suppose you need to prove something that involves swapping the order of addition, like `a + b = b + a`. This is the commutativity of addition.

1. **Guessing the Name**: You'd probably guess `add_comm` or `Nat.add_comm`.

2. **Verifying and Checking the Lemma's Statement**: Before you use a lemma, it's good practice to check its exact statement. You can use the `#check` command for this. If you have the necessary imports (like `import Mathlib.Data.Nat.Basic` or a more general algebra import), you can type this in your Lean file:

```
#check Nat.add_comm
```

   Or, for a more general version if it exists (many algebraic lemmas are generalized beyond `Nat`):

```
-- Import a relevant algebra module if not already imported
-- #check add_comm -- for the general version
```

   If `Nat.add_comm` is the one relevant to natural numbers, the Infoview will display its type, which *is* the statement of the theorem. It would look something like:

```
Nat.add_comm (n m : Nat) : n + m = m + n
```

   This confirms that `Nat.add_comm` is indeed the lemma `∀ (n m : Nat), n + m = m + n`. It takes two natural numbers, `n` and `m`, as arguments and proves that `n + m = m + n`.

3. **Using `exact?` or `apply?`**:

   If you had the goal `⊢ a + b = b + a` and didn't know the lemma name, you could write:

```
example (a b : Nat) : a + b = b + a := by
  exact? or apply?
```

   And Lean would likely suggest `Nat.add_comm`.

   So, the typical workflow is:

- Encounter a goal.
- Recognize the mathematical property.

- Try to find the lemma (guess name, search docs, use `library_search`).
- `#check` the lemma to be sure it's what you expect.
- Use it in your proof, typically with `rw` for equalities.

Consider the following theorem. We want to prove that if you add a and b first, and then add c, it's the same as adding b and a first, and then adding c. Your Goal: Prove the theorem ab_plus_c_eq_ba_plus_c.

```
import Mathlib.Data.Nat.Basic --

theorem ab_plus_c_eq_ba_plus_c (a b c : Nat) : (a + b) + c = (b + a) + c := by
  -- Your proof tactics go here!
```

https://leansearch.net/
https://loogle.lean-lang.org/
https://www.moogle.ai/
https://www.leanexplore.com/