

---

# **Topology Filters - Notes**

***Release 0.1***

**Carlos Caralps**

**Jul 16, 2021**



## CONTENTS

|          |                               |          |
|----------|-------------------------------|----------|
| <b>1</b> | <b>Introduction</b>           | <b>1</b> |
| <b>2</b> | <b>Pretty Symbols in Lean</b> | <b>3</b> |
| <b>3</b> | <b>Glossary of tactics</b>    | <b>5</b> |



## INTRODUCTION

### 1.1 What is Lean?

Lean is an open source proof-checker and a proof-assistant. One can *explain* mathematical proofs to it and it can check their correctness. It also simplifies the proof writing process by providing *goals* and *tactics*.

Lean is built on top of a formal system called type theory. In type theory, the basic notions are “terms” and “types” — compare to “elements” and “sets” in set theory. Every term has a type, and types are just a special kind of term. Terms can be interpreted as mathematical objects, functions, propositions, or proofs. The only two things Lean can do is *create* terms and *check* their types. By iterating these two operations, we can teach Lean to verify complex mathematical proofs.

```
def x := 2 + 2 -- a natural number
def f (x : ℕ) := x + 3 -- a function
def easy_theorem_statement := 2 + 2 = 4 -- a proposition
def fermats_last_theorem_statement -- another proposition
  :=
  ∀ n : ℕ,
  n > 2
  →
  ¬ (∃ x y z : ℕ, (x^n + y^n = z^n) ∧ (x ≠ 0) ∧ (y ≠ 0) ∧ (z ≠ 0))

theorem
easy_proof : easy_theorem_statement -- proof of easy_theorem
:=
begin
  exact rfl,
end

theorem
my_hard_proof : fermats_last_theorem_statement -- cheating!
:=
begin
  sorry,
end

#check x
#check f
#check easy_theorem_statement
#check fermats_last_theorem_statement
#check easy_proof
#check my_hard_proof
```

## 1.2 How to use these notes

Every once in a while, you will see a code snippet like this:

```
#eval "Hello, World!"
```

Clicking on the `try it!` button in the upper right corner will open a copy in a window so that you can edit it, and Lean provides feedback in the `Lean Infoview` window. We use this feature to provide exercises inline in the notes. We recommend attempting each exercise as you go along.

These notes are based a 5-day Lean crash course at Mathcamp 2020. We have adapted them to BIYSC 2021.

These notes provide a sneak-peek into the world of theorem proving in Lean and are by no means comprehensive. It is recommended that you simultaneously attempt the [Natural Number Game](#). It is a fun (and highly addictive!) game that proves some basic properties of natural numbers in Lean.

## 1.3 Acknowledgments.

These notes are based on work of [Apurva Nakade](#) and [Jalex Stark](#). Large chunks of these notes are taken directly from [https://apurvanakade.github.io/courses/lean\\_at\\_MC2020/](https://apurvanakade.github.io/courses/lean_at_MC2020/) and [\\_\\_\\_](#).

## 1.4 Useful Links.

1. [Formalizing 100 theorems](#)
2. [Formalizing 100 theorems in Lean](#)
3. **Articles, videos, blog posts, etc.**
  1. [The Xena Project](#)
  2. [The Mechanization of Mathematics](#)
  3. [The Future of Mathematics](#)
4. [Lean Zulip chat group](#)

## PRETTY SYMBOLS IN LEAN

To produce a pretty symbol in Lean, type the *editor shortcut* followed by space or tab.

| Unicode           | Editor Shortcut        | Definition                         |
|-------------------|------------------------|------------------------------------|
| $\rightarrow$     | <code>\to</code>       | function or implies                |
| $\leftrightarrow$ | <code>\iff</code>      | if and only if                     |
| $\leftarrow$      | <code>\l</code>        | used by the <code>rw</code> tactic |
| $\neg$            | <code>\not</code>      | negation operator                  |
| $\wedge$          | <code>\and</code>      | and operator                       |
| $\vee$            | <code>\or</code>       | or operator                        |
| $\exists$         | <code>\exists</code>   | there exists quantifier            |
| $\forall$         | <code>\forall</code>   | for all quantifier                 |
| $\mathbb{N}$      | <code>\nat</code>      | type of natural numbers            |
| $\mathbb{Z}$      | <code>\int</code>      | type of integers                   |
| $\circ$           | <code>\circ</code>     | composition of functions           |
| $\neq$            | <code>\ne</code>       | not equal to                       |
| $\in$             | <code>\in</code>       | belongs to                         |
| $\notin$          | <code>\notin</code>    | does not belong to                 |
| $\angle$          | <code>\angle</code>    | angle                              |
| $\triangle$       | <code>\triangle</code> | triangle                           |
| $\cong$           | <code>\cong</code>     | congruence of segments             |
| $\simeq$          | <code>\simeq</code>    | congruence of angles               |





## GLOSSARY OF TACTICS

### 3.1 Implications in Lean

|                    |   |
|--------------------|---|
| <code>exact</code> | <p>If <math>P</math> is the target of the current goal and <math>hp</math> is a term of type <math>P</math>, then <code>exact hp</code>, will close the goal.</p> <p>Mathematically, this saying “this is <i>exactly</i> what we were required to prove”.</p>   |
| <code>intro</code> | <p>If the target of the current goal is a function <math>P \rightarrow Q</math>, then <code>intro hp</code>, will produce a hypothesis <math>hp : P</math> and change the target to <math>Q</math>.</p> <p>Mathematically, this is saying that in order to define a function from <math>P</math> to <math>Q</math>, we first need to choose an arbitrary element of <math>P</math>.</p> |

|                    |  |
|--------------------|--|
| <code>have</code>  | <p><code>have</code> is used to create intermediate variables.</p> <p>If <math>f</math> is a term of type <math>P \rightarrow Q</math> and <math>hp</math> is a term of type <math>P</math>, then <code>have hq := f hp</code>, creates the hypothesis <math>hq : Q</math>.</p>  |
| <code>apply</code> | <p><code>apply</code> is used for backward reasoning.</p> <p>If the target of the current goal is <math>Q</math> and <math>f</math> is a term of type <math>P \rightarrow Q</math>, then <code>apply f</code>, changes target to <math>P</math>.</p> <p>Mathematically, this is equivalent to saying “because <math>P</math> implies <math>Q</math>, to prove <math>Q</math> it suffices to prove <math>P</math>”.</p> |

## 3.2 Proof by contradiction

|                  |   |
|------------------|---|
| ex-falso         | Changes the target of the current goal to false.<br>The name derives from “ <i>ex falso, quodlibet</i> ” which translates to “from contradiction, anything”. You should use this tactic when there are contradictory hypotheses present.  |
| by_cases         | If $P : \text{Prop}$ , then <code>by_cases P</code> , creates two goals, the first with a hypothesis $hp : P$ and second with a hypothesis $hp : \neg P$ .<br>Mathematically, this is saying either $P$ is true or $P$ is false. <code>by_cases</code> is the most direct application of the law of excluded middle.  |
| by_contradiction | If the target of the current goal is $Q$ , then <code>by_contradiction</code> , changes the target to false and adds $hnq : \neg Q$ as a hypothesis.<br>Mathematically, this is proof by contradiction.   |
| push_neg         | <code>push_neg</code> , simplifies negations in the target.<br>For example, if the target of the current goal is $\neg \neg P$ , then <code>push_neg</code> , simplifies it to $P$ .<br>You can also push negations across a hypothesis $hp : P$ using <code>push_neg at hp</code> .  |
| contrapose!      | If the target of the current goal is $P \rightarrow Q$ , then <code>contrapose!</code> , changes the target to $\neg Q \rightarrow \neg P$ .<br>If the target of the current goal is $Q$ and one of the hypotheses is $hp : P$ , then <code>contrapose! hp</code> , changes the target to $\neg P$ and changes the hypothesis to $hp : \neg Q$ .<br>Mathematically, this is replacing the target by its contrapositive. |

## 3.3 And / Or

|       |   |
|-------|---|
| cases | <code>cases</code> is a general tactic that breaks a complicated term into simpler ones.<br>If $hpq$ is a term of type $P \wedge Q$ , then <code>cases hpq with hp hq</code> , breaks it into $hp : P$ and $hq : Q$ .<br>If $hpq$ is a term of type $P \times Q$ , then <code>cases hpq with hp hq</code> , breaks it into $hp : P$ and $hq : Q$ .<br>If $fg$ is a term of type $P \leftrightarrow Q$ , then <code>cases fg with f g</code> , breaks it into $f : P \rightarrow Q$ and $g : Q \rightarrow P$ .<br>If $hpq$ is a term of type $P \vee Q$ , then <code>cases hpq with hp hq</code> , creates two goals and adds the hypotheses $hp : P$ and $hq : Q$ to one each. |
| split | <code>split</code> is a general tactic that breaks a complicated goal into simpler ones.<br>If the target of the current goal is $P \wedge Q$ , then <code>split</code> , breaks up the goal into two goals with targets $P$ and $Q$ .<br>If the target of the current goal is $P \times Q$ , then <code>split</code> , breaks up the goal into two goals with targets $P$ and $Q$ .<br>If the target of the current goal is $P \leftrightarrow Q$ , then <code>split</code> , breaks up the goal into two goals with targets $P \rightarrow Q$ and $Q \rightarrow P$ .   |
| left  | If the target of the current goal is $P \vee Q$ , then <code>left</code> , changes the target to $P$ .  |
| right | If the target of the current goal is $P \vee Q$ , then <code>right</code> , changes the target to $Q$ .   |

## 3.4 Quantifiers

|       |  |
|-------|--|
| have  | If $hp$ is a term of type $\forall x : X, P\ x$ and $y$ is a term of type $Y$ then <code>have hpy := hp (y)</code> creates a hypothesis $hpy : P\ y$ .   |
| intro | If the target of the current goal is $\forall x : X, P\ x$ , then <code>intro x</code> , creates a hypothesis $x : X$ and changes the target to $P\ x$ . |

|       |  |
|-------|--|
| cases | If $hp$ is a term of type $\exists x : X, P\ x$ , then <code>cases hp with x key</code> , breaks it into $x : X$ and $key : P\ x$ .  |
| use   | If the target of the current goal is $\exists x : X, P\ x$ and $y$ is a term of type $X$ , then <code>use y</code> , changes the target to $P\ y$ and tries to close the goal. |

## 3.5 Proving “trivial” statements

|          |   |
|----------|---|
| norm_num | <code>norm_num</code> is Lean’s calculator. If the target has a proof that involves <i>only</i> numbers and arithmetic operations, then <code>norm_num</code> will close this goal.<br>If $hp : P$ is an assumption then <code>norm_num at hp</code> , tries to use <code>simplify hp</code> using basic arithmetic operations. |
| ring     | <code>ring</code> , is Lean’s symbolic manipulator. If the target has a proof that involves <i>only</i> algebraic operations, then <code>ring</code> , will close the goal.<br>If $hp : P$ is an assumption then <code>ring at hp</code> , tries to use <code>simplify hp</code> using basic algebraic operations.              |
| linarith | <code>linarith</code> , is Lean’s inequality solver.  |
| simp     | <code>simp</code> , is a very complex tactic that tries to use theorems from the <code>mathlib</code> library to close the goal. You should only ever use <code>simp</code> , to <i>close a goal</i> because its behavior changes as more theorems get added to the library.  |

## 3.6 Equality

|      |  |
|------|--|
| refl | If the current goal is of the form $X = X$ or $P \leftrightarrow P$ , then <code>refl</code> will finish the proof. As long as both sides are defined to be equal, this will work. For example, it will work with the goal $3 = 2 + 1$ because <i>by definition</i> the number 3 is defined to be 2 plus one.<br>Mathematically, this says “check that both sides are equal <i>by definition</i> ”.  |
| rw   | If $f$ is a term of type $P = Q$ (or $P \leftrightarrow Q$ ), then<br><code>rw f</code> , searches for $P$ in the target and replaces it with $Q$ .<br><code>rw &lt;f</code> , searches for $Q$ in the target and replaces it with $P$ .<br>If additionally, $hr : R$ is a hypothesis, then<br><code>rw f at hr</code> , searches for $P$ in the expression $R$ and replaces it with $Q$ .<br><code>rw &lt;f at hr</code> , searches for $Q$ in the expression $R$ and replaces it with $P$ .<br>Mathematically, this is saying because $P = Q$ , we can replace $P$ with $Q$ (or the other way around). |