CUMC 2021 at UWO

Teaching Math to Computers

Apurva Nakade

apurva.nakade@northwestern.edu

Open the following link in a web browser:

https://leanprover-community.github.io/lean-web-editor/

Open 5-6 tabs with this link to avoid delays later.

Plan

1. Overview of Proof Assistants/Theorem Provers
2. Introduction to Lean Theorem Prover
3. Work on Lean worksheets in breakout rooms
    - You'll be learning a new programming language.

Big Questions:
Will computers ever be able to ...

- Verify existing math?
    - eg. Formal proof of Feit–Thompson theorem, Liquid tensor experiment
- Help humans write proofs?
    - Train an AI that can understand existing proofs!
- Understand existing math?
    - Train an AI that can understand existing proofs!
- Create new math?
    - Train an AI that can understand existing proofs!

Progress so far: Lean-gptf (Code, Presentation).

Current goal:
Create a database of existing math

- Computers cannot understand *natural language proofs*.
- Need to translate *natural language proofs* into *programs*.
- Curry–Howard correspondence (mid 1900's)
- Several competing languages for writing these *programs*:
    - Mizar, HOL, Isabelle, Coq, **Lean**, PVS, MetaMath, and more
- Formalizing 100 Theorems (https://www.cs.ru.nl/~freek/100/)

Lean Theorem Prover

Lean Theorem Prover

Pros:

- Open source
- Large math library *actively* maintained by mathematicians
  - https://leanprover-community.github.io/mathlib-overview.html
- Welcoming and friendly community
  - Leanprover Zulipchat
- A LOT of online resources for learning
  - https://leanprover-community.github.io/learn.html
- Somewhat human readable
- *Undergraduate students contribute to the Lean math library.*

Lean Theorem Prover

Cons:

- Kernel developed and maintained by Microsoft Research
- Relatively new (still being developed)
- Based on *type theory* instead of *set theory*

Sample Lean Code

Theorem: There are infinitely many prime numbers.

Theorem: For every natural number n, there exists a prime p which is greater than or equal to n.

1. Let $m = n! + 1$.
2. Let p be the smallest prime factor of m.
3. p is prime, by construction.
4. Remains to show that p is greater than or equal to n.
   1. Proof by contradiction:
   2. Suppose $p < n$.
   3. Then p divides $n!$.
   4. Hence, p divides $m - n!$ which equals 1. Contradiction!

```
theorem infinitude_of_primes (n : ℕ) : ∃ (p : ℕ), prime p ∧ n ≤ p :=

begin
  let m := n.factorial + 1,
  let p := min_fac m,
  use p,

  have p_is_prime : prime p := min_fac_prime (ne_of_gt (succ_lt_succ (factorial_pos n))),
  split,
  begin
    exact p_is_prime,
  end,

  begin
    by_contradiction h, push_neg at h,
    have p_dvd_n : p ∣ n.factorial := dvd_factorial (min_fac_pos _) (le_of_lt h),
    have p_dvd_one : p ∣ 1 := (nat.dvd_add_iff_right p_dvd_n).2 (min_fac_dvd _),
    exact p_is_prime.not_dvd_one p_dvd_one,
  end
end
```

- Download the file shared in the Zoom chat.
- Go to: https://leanprover-community.github.io/lean-web-editor/
- Click on "Choose file" and select the downloaded file (infinitude_of_primes.lean).
- Wait for the bar on the top to change color from orange to green.
- **Place your cursor at the end of any line to see the "goal state" at that point.**
- **Hover your mouse over a command to see more info about it.**

Lean jargon

- Every theorem in Lean has some *hypotheses* (can be empty) and a single *target*.
- In the previous example,
  - "$(n : \mathbb{N})$" is a hypothesis, and
  - "$\exists (p : \mathbb{N})$, prime $p \wedge n \leq p$" is the target.
- The goal is to construct a proof of the target using the hypotheses (and already proven theorems) and logically valid arguments.
- Lean lets you do this dynamically using "tactics".
- After each valid "tactic" command, the "goal state" is updated and is visible in the "goal window".

Goal window

In the previous example, the initial proof state as shown in the goal window is:

```
n : ℕ
⊢ ∃ (p : ℕ), prime p ∧ n ≤ p
```

After the tactics

```
let m := n.factorial + 1,
let p := min_fac m,
use p,
```

the new proof state becomes:

```
n : ℕ,
m : ℕ := n.factorial + 1,
p : ℕ := m.min_fac
⊢ prime p ∧ n ≤ p
```

The content after the symbol "⊢" is the current target and the rest are the current hypotheses.

Propositional logic in Lean

Propositional logic in Lean

We'll learn how to prove theorems involving

- Implication
- Negation
- And
- Or

Proofs in Lean

- For a computer, proofs are "more important" than theorems.
  - This is (arguably) the fundamental difference between the way we do math and the way machines "do math".
- Proof checkers like Lean need to keep track of *proofs*.

We pass proofs as arguments to tactics.

Notation

- P, Q, R will denote propositions.
- $h : P$ stands for "h is a proof of P".

We pass proofs as arguments to tactics.

For today,

```
exact h,
```

is a valid tactic but

```
exact P,
```

is invalid.

Implication

Implication

- "P implies Q" is denoted by $P \to Q$.
- Three basic tactics:
    - exact
    - intro
    - apply

Cheat Sheet

Purpose	Tactic
Closing goal	exact

	Hypothesis	Target
Implication	apply	intro

Negation

Negation

- "Negation of P" is denoted by $\neg P$.
- There is an in-built proposition **false** which is false.
- $\neg P$ is defined as the proposition "P implies **false**".
  - If P is true then "P implies **false**" is false.
  - If P is false then "P implies **false**" is true.
- Implications are all you need for manipulating negations!

Cheat Sheet

Purpose	Tactic
Closing goal	exact

	Hypothesis	Target
Implication	apply	intro
Negation	apply	intro

And, Or

And, Or

- "P and Q" is denoted by $P \wedge Q$ and "P or Q" is denoted by $P \vee Q$.
- Tactics for "And" and "Or" operators:
    - cases
    - split
    - left
    - right

Cheat Sheet

Purpose	Tactic
Closing goal	exact

	Hypothesis	Target
Implication	apply	intro
Negation	apply	intro
And	cases	split
Or	cases	left/right

Law of excluded middle

Law of excluded middle

- So far, we've "constructed proofs" from *hypotheses*.
- Constructive proofs cannot be used to prove "$\neg\neg P$ implies P".
- This requires on an extra axiom called the *Law of excluded middle*.
- The *Law of excluded* middle states that
  - For any proposition P, either P is true or $\neg P$ is true.
- Proof by contradiction relies on the *Law of excluded middle*.

Proof by Contradiction

- Tactics that use the law of excluded middle:
    - push_neg
    - by_cases
    - by_contradiction

Cheat Sheet

Purpose	Tactic
Closing goal	exact
Simplify negations	push_neg
Proof by contradiction	by_contradiction
LEM	by_cases

	Hypothesis	Target
Implication	apply	intro
Negation	apply	intro
And	cases	split
Or	cases	left/right

Learning resources

- https://leanprover-community.github.io/learn.html
- https://leanprover.zulipchat.com/