Lean Theorem Prover: The Lean, Mean, Math-Proving Machine

Sarah Herr, Joseph Kopp, Kailey Peppard, Ethan Shade

MTH 342, Dr. Jun Li

University of Dayton

Scan here to learn more and view sources

What is Lean?

Lean was created by Microsoft in 2013 as a functional programming language to write correct and maintainable code. Lean was not initially designed to check mathematical proofs, but mathematicians found this feature useful and expanded upon it. Lean's design makes it easy to write proofs by having many functions and definitions already written into its programming. The software doesn't say how to write proofs, but it can perform some of the simpler sub-steps of the proof and check your logic as you proceed.

Lean pulls from MathLib; a library of definitions, lemmas, and theorems that are formalized in code. As this library grows, more theorems will be formalized in Lean.

In 1999, mathematicians Paul and Jack Abad wrote a list "The Hundred Greatest Theorems". The Lean community has been trying to formalize these 100 proofs and has succeeded with 76 of them.

Successfully formalized theorems include:

- The square root of 2 is irrational
- The denumerability of rational numbers

Theorems yet to be formalized include:

- Godel's Incompleteness Theorem
- Pi is transcendental

Future of Lean

As more theorems are added into MathLib, Lean will have capabilities to help prove more theorems in math. With everything digitized in Lean, the proofs become available to a wider audience. Between reaching a wider audience and the way Lean automatically checks for correctness, it will become easier to make advancements in math.

Syntax

Lean uses type theory as a foundation instead of set theory. In essence, type theory is simply another way to talk about a "collection of things". But there is one key difference: rather than having both axioms and rules of inference, a type theory only has rules. This results in proofs being treated as mathematical objects, which has important impacts on logic and notation.

Take ⟨⟨a, b⟩, h1, h2⟩, for example, where a, b are integers and h1 is a proof b is positive and h2 is a proof that a and b are coprime. This is seen in the example proof of the denumerability of the rationals.

Lean uses all the typical Boolean logic operators, the universal quantifiers of "for all" and "there exists", and lambda notation for functions.

Unicode	Ascii	Lean input
∧	/\	\and
∨	\/	\or
→	->	\to, \r, \imp
↔	<->	\iff, \lr
∀	forall	\all
∃	exists	\ex
λ	fun	\lam, \fun
≠	~=	\ne

One thing to note however is that because using type theory necessitates using "intuitionistic" logic, "not P" is instead thought of as "P implies absurdity".

As you use "tactics" in Lean, the software will continually update what your "goal" is; that is, what you need to prove to prove the original statement. The tactics you use must connect to this goal. Some tactics include:

- The "intro" or "intros" tactic introduces a new hypothesis, which is what the "h" in front of a proposition stands for.
- The "apply" tactic applies a premise to reach your goal.
- The "exact" tactic is used when your goal is exactly one of your premises.

8:8: goal

```
P Q : Prop,
hP : P,
hPQ : P → Q
⊢ Q
```

```
4  variables (P Q R : Prop)
5  example : P → (P → Q) → Q :=
6  begin
7    intros hP hPQ,
8    apply hPQ,
9    exact hP,
10 end
11
```

Denumerability of Rationals

The proof sets up a bijection from $\mathbb{Q} \cong \mathbb{Z} \times \mathbb{N}$, where $x.1 \in \mathbb{Z}, x.2 \in \mathbb{N}_1$ and $\gcd(x.1, x.2) = 1$.

Thus, there are two goals:

22:17: goal

```
x : ℚ
⊢ 0 < (x.num, x.denom).snd ∧ (x.num,
x.denom).fst.nat_abs.coprime (x.num, x.denom).snd
```

Or show that given an element in $\mathbb{Q}$, the denominator of that element is positive, and the numerator and denominator are coprime. And:

23:18: goal

```
x : {x // 0 < x.snd ∧ x.fst.nat_abs.coprime x.snd}
⊢ ℚ
```

Or show that given a pair where the second element is positive and the elements are coprime, that pair is the same as a fraction in $\mathbb{Q}$.

Then it remains to show that $\mathbb{Z} \times \mathbb{N}$ (with restrictions) is countably infinite. Again, there are two goals:

31:27: goal

```
T : Type := {x // 0 < x.snd ∧ x.fst.nat_abs.coprime
x.snd}
⊢ infinite T
```

Or show that $\mathbb{Z} \times \mathbb{N}$ is infinite. And:

32:32: goal

```
T : Type := {x // 0 < x.snd ∧ x.fst.nat_abs.coprime
x.snd},
_inst : infinite T := infinite.of_injective
↑denumerable_aux denumerable_aux.injective
⊢ encodable T
```

Or show that $\mathbb{Z} \times \mathbb{N}$ is countable (called encodable by Lean). Then denumerable is defined as infinite and encodable so $\mathbb{Z} \times \mathbb{N}$ is denumerable. And since $\mathbb{Q} \cong \mathbb{Z} \times \mathbb{N}$ it must be that $\mathbb{Q}$ is also denumerable.