Lean Seminar Series

Getting Started: Proving with the Lean Interactive Theorem Prover

Session 2 UTSC

December 1, 2021

Welcome to the LEAN Seminar Series, Session 2!

While you are getting settled, please enjoy

Symphony in F Major
Op. 33 No. 3 II. Allegretto
By Paul Wranitzky

Our Research Team : Theorem Proving for Math Education

Gila Hanna
Mathematics Education
Professor
OISE/UT

Kitty Yan
Mathematics Education
Postdoc Fellow
OISE/UT

Japleen Kaur Anand
Mathematics Education
Master's Student
OISE/UT

Logan Murphy
Computer Science
Master's Student
CS/UT

Overview

- Revisit the "Malice and Alice" puzzle
- Review tactics and theorems
- A Lean proof using the four tactics
- The Natural Number Game - Addition and Multiplication Worlds

Revisit "Malice and Alice"

1. A man and a woman were together in a bar at the time of the murder.
2. The victim and the killer were together on a beach at the time of the murder.
3. One of Alice's two children was alone at the time of the murder.
4. Alice and her husband were not together at the time of the murder.
5. The victim's twin was not the killer.
6. The killer was younger than the victim.

❑ Man and woman in the bar

❑ Killer and victim on the beach

❑ Child alone

Revisit "Malice and Alice"

- ❑ Man and woman in the bar
- ❑ Reasoning by cases (systematic search)

(A-H) V (A-B) V (A-S) V (D-B) V (D-H) V (D-S)

Revisit "Malice and Alice"

- Man and woman in the bar
- Reasoning by cases (systematic search)

~~(A-H)~~ ∨ (A-B) ∨ (A-S) ∨ (D-B) ∨ (D-H) ∨ (~~D-S~~)

Contradiction by (4) — Contradiction by (3)

→ (A-B) ∨ (A-S) ∨ (D-B) ∨ (D-H)

cases

1. (A-B) → (H-D) ∨ (H-S)
   - 1.1) (H-D)
   - 1.2) (H-S)
2. (A-S)
3. (D-B)
4. (D-H)

Revisit "Malice and Alice"

- Man and woman in the bar
- Reasoning by cases (systematic search)

~~(A-H)~~ ∨ (A-B) ∨ (A-S) ∨ (D-B) ∨ (D-H) ∨ (~~D-S)~~

Contradiction by (4) Contradiction by (3)

⟶ (A-B) ∨ (A-S) ∨ (D-B) ∨ (D-H)

cases

1. (A-B) ⟶ (H-D) ∨ (H-S) ~~1.1) (H-D)~~ (6)&(5)
   ~~1.2) (H-S)~~
2. (A-S)
3. (D-B)
4. (D-H)

Propositional Logic

- Implication: P implies (if P then Q): $P \to Q$
- If and only if: $P \leftrightarrow Q$
- Conjunction (and): $P \wedge Q$
- Disjunction (or): $P \vee Q$

- "If I have two heads, then circles are squares."
- "If I had two heads, then circles would be squares."

Review: The five Peano Axioms of Number Theory

1. Zero is a natural number.
2. Every natural number has a successor in the natural numbers.
3. Zero is not the successor of any natural number.
4. The successors of two natural numbers are same iff the two original numbers are the same.*
5. If a set contains zero and the successor of every number is in the set, then the set contains the natural numbers.**

Peano’s Axioms in LEAN THEOREM PROVER

- import mynat.definition
  - imports Peano's definition of the natural numbers {0,1,2,3,4,...}
- It gives us:
  - a term 0 : mynat, interpreted as the number zero.
  - a function succ : mynat → mynat, with succ n interpreted as "the number after n”.
  - The principle of mathematical induction.

Review Tactic : Reflexivity

Abbreviation: `refl`

Used to close a goal of the form "P = Q",
where P and Q can be "reduced" to the same value

```
theorem add_three_ones : 1 + 1 + 1 = 3 :=
begin
  refl,
end
```

Tactic : Rewrite

Abbreviation: `rw`

Given a hypothesis of the form "A = B", replaces occurrences of A with B, or vice versa.

```
theorem my_nat_theorem
(a b c d : ℕ)
(h₁ : a = b)
(h₂ : c = d) : a + b + c = b + c + d :=
begin
-- ⊢ a + b + c = b + c + d
    rw h₁,
-- ⊢ b + b + c = b + c + d
    rw ← h₂,
-- ⊢ b + b + c = b + c + c
end
```

Tactic: Induction

If n : mynat is in our assumptions,

then `induction n with d hd` attempts to prove the goal by induction on n, with the inductive assumption in the succ case being hd.

Proofs of Theorems: Addition

- ❑ add_zero (a : mynat) : a + 0 = a

  Use with `rw add_zero`.

  It simplifies a + 0 to a.

- ❑ zero_add (a : mynat) : 0 + a = a

  Use with `rw zero_add`.

  It simplifies 0 + a to a.

Proofs of Theorems: Addition

- add_succ (a b : mynat) : a + succ (b) = succ (a + b)

  Use with `rw add_succ`.

- succ_add (a b : mynat) : succ (a) + b = succ (a + b)

  Use with `rw succ_add`.

Proof of the Theorem: Addition is Commutative

Addition of natural numbers is commutative.

- add_comm (a b : mynat) : a + b = b + a

  `rw add_comm`, will just find the first ? + ? it sees and swap it around. Target more specific additions like this: `rw add_comm a` will swap around additions of the form a + ?, and `rw add_comm a b`, will only swap additions of the form a + b.

- add_right_comm (a b c : mynat) : a + b + c = a + c + b

Proofs of the Theorem: Addition is Associative

- Addition of natural numbers is associative.
  add_assoc (a b c : mynat) : (a + b) + c = a + (b + c)
  `rw add_assoc` will change (a + b) + c to a + (b + c), but to change it back you will need `rw ← add_assoc`.
- a + b + c = (a + b) + c
- Note: Get the left arrow by typing \l then the space bar in Lean.

Proofs of Theorems: Multiplication

- import mynat.mul

  imports the definition of multiplication on mynat

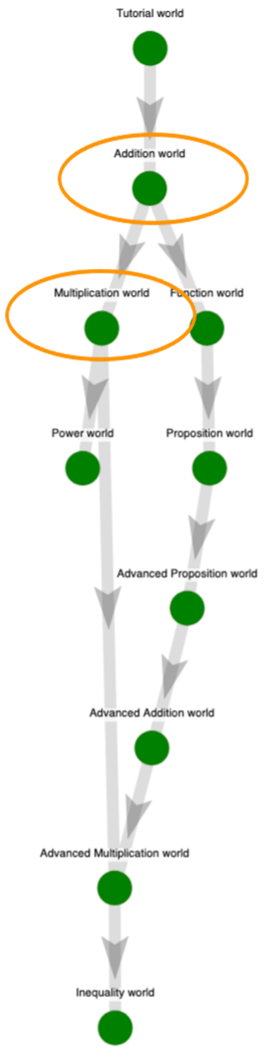
  - mul_zero (a : mynat) : a * 0 = 0
  - zero_mul (m : mynat) : 0 * m = 0
  - mul_succ (a b : mynat) : a * succ (b) = a * b + a
  - mul_one, one_mul, succ_mul?
  - Addition is distributive over multiplication.

    add_mul (a b t : mynat) : (a + b) * t = a * t + b * t

The Natural Number Game

- Addition World
- Multiplication World

```
• refl
• rw
• induction n with d hd
• simp
```



Session 3 will be in January, 2022.

See you all then!