A “calculation-heavy” introduction to proof, with support from Lean

Heather Macbeth

Department of Mathematics
Fordham University

Learning Mathematics with Lean 2
19 April 2023

Department syllabus

MATH 2001. Discrete Mathematics. (4 Credits)

This course introduces students to the language and writing of mathematical proofs in the context of discrete structures. Topics include

- elementary logic;
- basic proof techniques such as direct proof, proof by contradiction, contraposition, case division, induction;
- division, the Euclidean algorithm, modular arithmetic;
- set theory, relations and equivalence, functions.

Additional topics may include cardinality of sets, combinatorics, and graphs.

Prerequisites: MATH 1206 or MATH 1207 or MATH 12AB or MATH 12BC.

Audience

Class size of 20 (3 dropped out)

10 first-year/5 sophomore/2 junior/3 senior

Median background: Calculus I and II

Fordham's undergraduate acceptance rate is ≈55%

Goal: **Rigorous, but concrete.**

Lecture notes

Full lecture notes (300 pages) custom-written for the course.

https://hrmacbeth.github.io/math2001

Juxtapose:

- text solution to a problem written in "mature mathematician" style
- Lean solution to the same problem
- informal commentary on both

8.1.7. Example

Problem

Show that the function $x \mapsto x^2$ from $\mathbb{R}$ to $\mathbb{R}$ is not surjective.

Solution

We will show that there exists a real number y, such that for all real numbers x, $x^2 \neq y$.

Indeed, let us show that -1 has this property. Let x be a real number. Then

$$\begin{aligned} -1 &< 0 \\ &\leq x^2 \end{aligned}$$

so $x^2 \neq -1$.

As in Example 8.1.4, the first sentence constitutes a negation-normalization of the definition of "surjective" in this context. Effectively we are stating what would be the goal state in the Lean proof after `push_neg`.

```
⊢ ∃ y, ∀ (x : ℝ), x ^ 2 ≠ y
```

And here is the full Lean proof.

```
example : ¬ Surjective (fun x : ℝ ↦ x ^ 2) := by
  dsimp [Surjective]
  push_neg
  take -1
  intro x
  apply ne_of_gt
  calc -1 < 0 := by numbers
    _ ≤ x ^ 2 := by extra
```

Classes

75-minute classes, twice a week for 13 weeks.

Typical class structure:

- 25 minutes traditional blackboard lecture
- 5 minutes screenshare lecture doing the same problems in Lean
- 20 minutes working in Lean in pairs, lecturer circulating
- 25 minutes traditional blackboard lecture, perhaps on a more theoretical topic which doesn't translate directly into a "machine to generate homework problems"

Assessment

- 25% homework
  - students complete the same problems in Lean and on paper
  - 12.5% Lean homework
  - 12.5% written homework
  - `https://hrmacbeth.github.io/math2001/Homework.html`
- 10% + 10% one-on-one interviews assessing Lean fluency
  - students solve previously-unseen Lean exercises
  - different exercises for each student
- 20% + 35% traditional written exams

1. The niche
2. Bilingual between text and Lean
3. Custom automation
4. The first weeks
5. Logistics
6. Observations

Keeping the Lean and the text-mathematics close to each other is a worthy goal. But different people interpret this in different ways.

1. Keeping the descriptions of the steps close to their descriptions in the text proof ("procedural" vs "declarative" proof style)
2. Writing the Lean proof itself in (controlled) natural language
3. Keeping the sequence of goal states close to the sequence implicit in the text proof

These approaches can coexist.

I have spent my intellectual energy on (3), designing a "custom dialect" of Lean whose tactics match proof steps at the level of the course.

Inductive proof of a congruence

Consider the sequence (y_n) defined recursively by,

$$y_0 = 17$$
$$\text{for } n : \mathbb{N}, \quad y_{n+1} = 4y_n + 1.$$

Problem: Show that for all natural numbers n, y_n is congruent to either 7 or 9 modulo 10.

Solution: We prove this by induction on n. First,

$$\begin{aligned} y_0 &= 17 \\ &= 10 \cdot 1 + 7 \\ &\equiv 7 \mod 10. \end{aligned}$$

Now, let k be a natural number, and suppose that we know that y_k is congruent to either 7 or 9 modulo 10.

Case 1 ($y_k \equiv 7 \mod 10$): Then

$$\begin{aligned} y_{k+1} &= 4y_k + 1 \\ &\equiv 4 \cdot 7 + 1 \mod 10 \\ &= 2 \cdot 10 + 9 \\ &\equiv 9 \mod 10. \end{aligned}$$

Case 2 ($y_k \equiv 9 \mod 10$): Then

$$\begin{aligned} y_{k+1} &= 4y_k + 1 \\ &\equiv 4 \cdot 9 + 1 \mod 10 \\ &= 3 \cdot 10 + 7 \\ &\equiv 7 \mod 10. \end{aligned}$$

```
def y : ℕ → ℤ
  | 0 => 17
  | n + 1 => 4 * y n + 1

example (n : ℕ) :
    y n ≡ 7 [ZMOD 10]
    ∨ y n ≡ 9 [ZMOD 10] := by
  simple_induction n with k IH
  · left
    calc y 0
        = 17 := by rw [y]
      _ = 10 * 1 + 7 := by numbers
      _ ≡ 7 [ZMOD 10] := by extra
  obtain h1 | h2 := IH
  · right
    calc y (k + 1)
        = 4 * y k + 1 := by rw [y]
      _ ≡ 4 * 7 + 1 [ZMOD 10] := by rel [h1]
      _ = 2 * 10 + 9 := by numbers
      _ ≡ 9 [ZMOD 10] := by extra
  · left
    calc y (k + 1)
        = 4 * y k + 1 := by rw [y]
      _ ≡ 4 * 9 + 1 [ZMOD 10] := by rel [h2]
      _ = 3 * 10 + 7 := by numbers
      _ ≡ 7 [ZMOD 10] := by extra
```

Inductive proof of an inequality

Problem: Show that for all sufficiently large natural numbers n, $2^n \geq n^2$.

Solution: We will show this for all natural numbers $n \geq 4$.

We prove this by induction on n, starting at 4. The base case, $2^4 \geq 4^2$, is clear.

Suppose now that for some natural number $k \geq 4$, it is true that $2^k \geq k^2$. Then

$$\begin{aligned} 2^{k+1} &= 2 \cdot 2^k \\ &\geq 2k^2 \\ &= k^2 + k \cdot k \\ &\geq k^2 + 4k \\ &= k^2 + 2k + 2k \\ &\geq k^2 + 2k + 2 \cdot 4 \\ &= (k+1)^2 + 7 \\ &\geq (k+1)^2. \end{aligned}$$

```
example : forall_sufficiently_large n : ℕ,
    2 ^ n ≥ n ^ 2 := by
  dsimp
  take 4
  intro n hn
  induction_from_starting_point n, hn with k hk IH
  · -- base case
    numbers
  · -- inductive step
    calc 2 ^ (k + 1) = 2 * 2 ^ k := by ring
      _ ≥ 2 * k ^ 2 := by rel [IH]
      _ = k ^ 2 + k * k := by ring
      _ ≥ k ^ 2 + 4 * k := by rel [hk]
      _ = k ^ 2 + 2 * k + 2 * k := by ring
      _ ≥ k ^ 2 + 2 * k + 2 * 4 := by rel [hk]
      _ = (k + 1) ^ 2 + 7 := by ring
      _ ≥ (k + 1) ^ 2 := by extra
```

Injectivity of a linear map

Problem: Consider the function $(x, y) \mapsto (x+y, x+2y, x+3y)$ from $\mathbb{R}^2$ to $\mathbb{R}^3$. Show that this function is injective.

Solution: Let (x_1, y_1) and (x_2, y_2) be points in $\mathbb{R}^2$ and suppose that
$(x_1 + y_1, x_1 + 2y_1, x_1 + 3y_1) = (x_2 + y_2, x_2 + 2y_2, x_2 + 3y_2)$.
Then, inspecting co-ordinate by co-ordinate, we have that

$$\begin{aligned} x_1 + y_1 &= x_2 + y_2 \\ x_1 + 2y_1 &= x_2 + 2y_2 \\ x_1 + 3y_1 &= x_2 + 3y_2 \end{aligned}$$

We calculate that

$$\begin{aligned} y_1 &= (x_1 + 2y_1) - (x_1 + y_1) \\ &= (x_2 + 2y_2) - (x_2 + y_2) \\ &= y_2 \end{aligned}$$

Subtracting this equation from the first equation, we also have that $x_1 = x_2$. So $(x_1, y_1) = (x_2, y_2)$.

```
example : Injective (fun ((x, y) : ℝ × ℝ)
    ↦ (x + y, x + 2 * y, x + 3 * y)) := by
  intro (x1, y1) (x2, y2) h
  dsimp at h
  obtain ⟨h, h', h''⟩ := h
  have : y1 = y2
  calc y1 = (x1 + 2 * y1) - (x1 + y1) := by ring
    _ = (x2 + 2 * y2) - (x2 + y2) := by rw [h, h']
    _ = y2 := by ring
  constructor
  · addarith [this, h]
  · apply this
```

Lemmas are a failure of imagination

Often when people who are "deep in the Lean ecosystem" write Lean proofs, these proofs are a thicket of cross-references: experts know exactly the library lemma about monoids they need.

This is not friendly to beginners, especially if the beginners don't know what a monoid is.

And it also doesn't represent how a mathematician would *think* about that kind of argument.

We should aspire to uncover the algorithm represented when an expert Lean user would apply a sequence of lemmas in some particular way, and write a tactic for each such algorithm.

Lean as an enforcer of writing style

Lean has the capability to be a true "proof assistant," helping the user discover proofs by alternating forward/backward reasoning, and applying known transformations and normalizations wherever convenient. My Lean dialect does *not* take full advantage of this.

Instead my dialect enforces a proof style similar to the "final version" of a text write-up.

This is appropriate for students at this level, who don't yet know the conventions for such write-ups ... you need to know the rules before you can break them.

Notes for Lean experts

Some of my custom tactics replace more powerful or flexible tactics from idiomatic Lean.

I only use tactics implementing algorithms which students at this level have intuition for.

- No `linarith` (a souped-up Gaussian elimination)
- No `polyrith` (Grobner bases)

I avoid tactics which enable proof styles which are awkward to describe in text. If no one would ever write a text proof in a certain way, this leads to divergence between text and Lean.

- No nonterminal `rw` (it's painful to write "by substituting the equality ($\star$) it suffices to prove that ...")
- Future work: No `apply`, instead implement and use a user-friendly forward-reasoning tactic

How to make the learning curve flatter at the start

Develop “sandboxes,” topics which contain nontrivial mathematics but can be expressed in a fixed small Lean grammar.

Sandbox 1: calculational proofs

At this stage, all proofs are straight calculations with each step justified by one of

- `ring`: algebra, normalize by expanding out and regrouping
- `numbers`: numeric calculations
- `rel`: substitution
- `extra`: dropping terms to prove $>/\geq/</\leq$
- `addarith`: move terms from left to right of equation or vice versa

Problem: Let x and y be integers, and suppose that $x + 3 \leq 2$ and $y + 2x \geq 3$. Show that $y > 3$.

Solution:

$$\begin{aligned} y &= (y + 2x) - 2x \\ &\geq 3 - 2x \\ &= 9 - 2(x + 3) \\ &\geq 9 - 2 \cdot 2 \\ &> 3. \end{aligned}$$

```
example {x y : ℤ} (hx : x + 3 ≤ 2)
    (hy : y + 2 * x ≥ 3) :
    y > 3 :=
calc
  y = y + 2 * x - 2 * x := by ring
  _ ≥ 3 - 2 * x := by rel [hy]
  _ = 9 - 2 * (x + 3) := by ring
  _ ≥ 9 - 2 * 2 := by rel [hx]
  _ > 3 := by numbers
```

Sandbox 2: elementary number theory

Proofs involving parity, divisibility, modular arithmetic: at this stage we use the techniques for calculational proofs plus

- the fragment of logic comprising only $\vee$, $\wedge$ and $\exists$ (no $\to$, $\leftrightarrow$, $\neg$, or $\forall$)
- intermediate goals (`have`)
- definitions
- the infoview

Problem: Show that if for some integer n we have that 5 divides $3n$, then 5 also divides n.

Solution: Since $5 \mid 3n$, there exists an integer x such that $3n = 5x$. Then

$$\begin{aligned} n &= 2(3n) - 5n \\ &= 2(5x) - 5n \\ &= 5(2x - n), \end{aligned}$$

so $5 \mid n$.

```
example {n : ℤ} (h1 : 5 ∣ 3 * n) : 5 ∣ n := by
  obtain ⟨x, hx⟩ := h1
  take 2 * x - n
  calc
    n = 2 * (3 * n) - 5 * n := by ring
    _ = 2 * (5 * x) - 5 * n := by rw [hx]
    _ = 5 * (2 * x - n) := by ring
```

Support

Copious office hours including a two-hour session the afternoon before homework is due.

Extensive back-and-forth with individual students by email.

Chose not to have a Discord/Zulip/Piazza/online forum for the class, to provide a partial check on homework "spoilers."

To provide this level of support, would be hard to go beyond my ratio of 20 students : 1 Lean-expert staff.

Coding environment

It is very useful to have an online coding environment, to avoid requiring students to install Lean on their own computers.

I use Gitpod, with a setup borrowed from Rob Lewis.

Other options:

- GitHub Codespaces
- CoCalc
- Lean 3 “game” environment (written by Mohammad Pedramfar)
- Lean 4 “game” environment (WIP Patrick Massot, Alex Bentkamp, Jon Eugster)

Autograding

It is nice to have an "autograder," so that students can submit their homework and immediately see that they have been awarded the points they are supposed to earn.

(Of course, writing an error-free proof in Lean ought to guarantee it, but sometimes students don't quite believe it until they see it.)

I use Gradescope, with a system written by Gabriel Ebner, Vanessa Rodrigues and Rob Lewis.

Other options: GitHub Classroom, system written by Matthew Ballard.

1. The niche
2. Bilingual between text and Lean
3. Custom automation
4. The first weeks
5. Logistics
6. Observations

Observations

Grading is a breeze. (Not just of the auto-graded Lean homework, but also of the matching paper write-ups.)

Students (even the weaker ones) incorporate Lean and logical terminology into their informal discourse ("hypothesis," "goal," "witness").

Students are faster to learn topics involving radical changes of goal state, e.g. induction (especially nonstandard induction principles) and disproofs. Some can only solve these problems using Lean as a crutch – ok with me.

Certain "fiddly" skills (e.g. intricate inequality and modular arithmetic calculations) are mastered by most students – this was not the case when I taught without Lean.