SMS SPRING MEETING: FORMALIZATION AND PROOF ASSISTANTS

Formalizing Finitely Presented Groups

From writing math to writing mathlib:

a beginner story in progress.

Hang Lu Su
Math PhD, ICMAT | MadLean organizer
¬ Lean expert | ¬ ML expert

March 2026

I want to share my beginner story but make it generally relevant

If you care about the math:

implementation details of `IsFinitelyPresented` and the roadmap ahead.

If you're just getting started:

a story of a Zulip lurker becoming a mathlib contributor.

If you're already an expert in Lean or ML:

a story of starting from Dec 2025 to March 2026 with AI tools.

Full disclosure

Beginners are notoriously bad at estimating how long it takes to finish a task…



what I thought my project would look like vs what it actually looks like now

Story outline

- **How LLMs got me formalizing**
  - How ChatGPT-4 re-awakened my formalization dream.
  - From Zulip lurking to project pitching at ItaLean2025.
- **Battle of definitions: iterating over every review**
  - Implementation details of `IsFinitelyPresented`.
  - Writing math vs mathlib: a culture shock.
- **Misadventures in vibe-proving**
  - I couldn't code mathlib by hand so I thought I'd vibe-code.
  - What could go wrong with that?

SECTION

1. The start

My dream theorem and coming out of Zulip lurking.

Why I got interested in formalizing

- **My favorite part of math is the deductive reasoning built on axioms.**
  - Considered going to grad school in formal methods.
  - Had friends formalizing number theory in Rocq
    - Seemed tedious, couldn't scale.
    - Mathematicians don't even finish their proofs in their own papers.
- **ChatGPT-4 changed my mind:** came out when I was thesis-writing.
  - Transformers were designed for machine translation [1]:
    - perfect start to translate informal math to formal math!
  - Formalization means built-in proof checker! What could go wrong?!

I discovered that mathlib4 existed, and my dream was revived!

My first dream formalization theorem

THEOREM: Reidemeister-Schreier theorem

If G is finitely presented and H is a finite-index subgroup of G, then H is finitely presented. See p. 169-175 of my thesis.

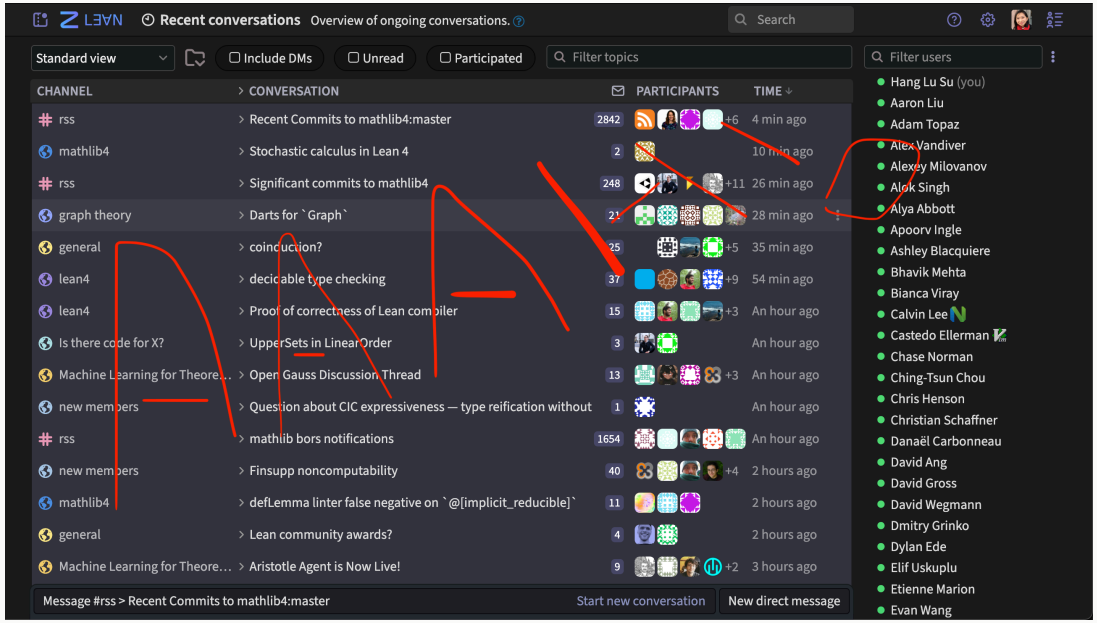
- Spent a week stuck understanding the *meaning* of the algebraic proof.
- The geometric insights helped me prove my favorite doctoral results.

WHAT I LEARNED

Target theorem helps build useful API by working backwards.

My lurker phase: learning by stalking

Initial approach to the problem: lurk on Zulip for months, overwhelmed.



Eventually, I heard about ItaLean2025 and signed up.

From lurking to project pitching

WHAT I LEARNED

It's important to do something.

When ItaLean2025 came around, I pitched a project.

USE OF AI

To avoid potential embarrassment, I used ChatGPT-5.1 as my first sounding board before posting. I had no sense of whether what I was proposing made sense or not, so I asked it to go through the Zulip and see if what I was saying was in line with what had been pitched before.

SECTION

2. Battle of definitions

Math writing vs mathlib: a culture shock.

Formalizing finitely presented groups

DEFINITION: Finitely presented group

A group G is **finitely presented** if there exist a **finite** set of generators S and a **finite** set of relations R such that

$$G \cong \langle S \mid R \rangle := F(S)/\langle\langle R \rangle\rangle$$

where $F(S)$ is the free group on S and $\langle\langle R \rangle\rangle$ is the normal closure of R.

Main issue: there are many valid ways to formalize this!

Going from set theory to type theory wasn't trivial either.

Using a surjection (due to Riccardo Brasca)

DEFINITION: Finitely presented group

$$\exists S, R \text{ finite such that } G \cong F(S)/\langle\langle R \rangle\rangle$$

LEAN

```
class IsFinitelyPresented (G : Type*) [Group G] : Prop where
  out : ∃ (n : ℕ) (f : FreeGroup (Fin n) →* G),
    Function.Surjective f ∧ IsNormalClosureFG f.ker
```

- S is `Fin n`, $\langle\langle R \rangle\rangle$ is `f.ker`
- "`Fin n` *is a natural number i with the constraint that i < n. It is the canonical type with n elements.*" docs: Init/Prelude/Fin

This is the definition we started with at ItaLean.

Using a surjection - Part 2

```
class IsFinitelyPresented (G : Type*) [Group G] : Prop where
  out : ∃ (n : ℕ) (f : FreeGroup (Fin n) →* G),
    Function.Surjective f ∧ IsNormalClosureFG f.ker
```
LEAN

Feedback from project teammates: (also mathematicians and beginners)

- **1**: *"This isn't really what a finitely presented group is."*
- **2** *"Do you even know what a finitely presented group is?!"* (hyperbole)

Our code at the end of the workshop.

- Feels like the start of a nice independent project but not mathlib-ready.
  - There are no proofs, just definitions which is a no-no for new contributors.
- I decide to re-work it over Christmas break with the goal of upstreaming.

Using isomorphisms (mine)

DEFINITION: Finitely presented group

$$\exists S, R \text{ finite such that } G \cong F(S)/\langle\langle R \rangle\rangle$$

LEAN

```
class IsFinitelyPresented (G : Type*) [Group G] : Prop where
  out : ∃ (α : Type) (_: Finite α) (rels : Set (FreeGroup α)) (_ : rels.Finite),
  Nonempty (G ≃* (PresentedGroup rels))
```

- S is `(α : Type) (_: Finite α)`, R is `rels : Set (FreeGroup α)`

Things felt smooth and easy-ish because that's what I was used to.

Using isomorphisms - Part 2

```
class IsFinitelyPresented (G : Type*) [Group G] : Prop where
  out : ∃ (α : Type) (_: Finite α) (rels : Set (FreeGroup α)) (_ : rels.Finite),
  Nonempty (G ≃* (PresentedGroup rels))
```
LEAN

I wrote a lot of code with this!

- Code to go between the different definitions + isomorphism theorem.
- `FP_is_FG` instance, `FreeGroup` instance, trivial instance, mul $\mathbb{Z}$ instance, etc

USE OF AI

I used Aristotle, then GPT-5.2, 5.3 and 5.4 to do this (will come back to this).

Using isomorphisms - Part 3

```
class IsFinitelyPresented (G : Type*) [Group G] : Prop where
  out : ∃ (α : Type) (_: Finite α) (rels : Set (FreeGroup α)) (_ : rels.Finite),
  Nonempty (G ≃* (PresentedGroup rels))
```
LEAN

Review: *"both the quantification over* `Type` *as wells as the* `Nonempty` *strike me as a little strange for the main definition."*- Source.

WHAT I LEARNED

- **Math:** *"a group is FP because ∃ isomorphism to FP group"* is fine.
- **Lean:** have to deal with a non-specific isomorphism and unspecific `Type` (or `Type*`) which can cause universe issues *as a default*.

Using isomorphisms - Part 4: Type digression

Universe hierarchy in CIC to avoid Girard's paradox [2] (the type-theoretic analogue of Russell's paradox: `Type : Type` is inconsistent).

- **Consequence:** everything needs to be grounded at a universe level.

WHAT I LEARNED

- It's enough to say "*for some* `Type`" in math, but not Lean. Demo.

```
    Group.FG G ↔ ∃ (α : Type*) (_ : Fintype α) (φ : FreeGroup α →* G),
Function.Surjective φ := by
```
LEAN

- `←` You're providing a type of arbitrary universe level.
- `→` Lean has to figure out the universe level with nothing to go on!

Using generating set `S : Set G` (due to Thomas Browning)

DEFINITION: Finitely presented group

$$\exists S, R: \quad G \cong F(S)/\langle\langle R \rangle\rangle, \quad |S|, |R| < \infty$$

LEAN

```
class IsFinitelyPresented (G : Type*) [Group G] : Prop where
  out: ∃ S : Set G, S.Finite ∧ Subgroup.closure S = ⊤ ∧
  IsNormalClosureFG (FreeGroup.lift ((↑) : S → G)).ker
```

- S is `S : Set G`, $\langle\langle R \rangle\rangle$ is `f.ker`

Easily extractable data, `S` guaranteed to be in the same universe as `G`.

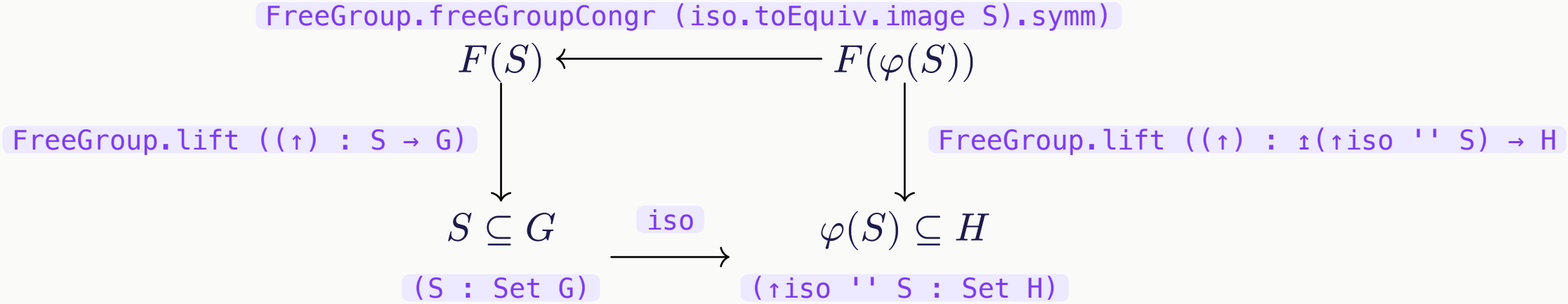
Use gen set `S : Set G` - Part 2

```
class IsFinitelyPresented (G : Type*) [Group G] : Prop where
  out: ∃ S : Set G, S.Finite ∧ Subgroup.closure S = ⊤ ∧
  IsNormalClosureFG (FreeGroup.lift ((↑) : S → G)).ker
```
LEAN

I wrote a minimal API for this. (`d59b5c3`)

- This definition.
- Keeps the `IsNormalClosureFG.map` property.
  - That is, being FG in the normal closure is invariant under surjection.
- Wrote isomorphism invariance theorem and factored things this time!
  - I was so proud of myself!

`S : Set G` Part 3: commutative diagram for the kernel transport




```
lemma FreeGroup.lift_mulEquiv_image (iso : G ≃* H) (S : Set G) :
  FreeGroup.lift ((↑) : ↥(↑iso '' S) → H) =
    (↑iso : G →* H).comp ((FreeGroup.lift ((↑) : S → G)).comp
      (FreeGroup.freeGroupCongr (iso.toEquiv.image S).symm)) := by
ext ⟨_, s, hs, rfl⟩; simp [Equiv.image]
```

LEAN

`S : Set G` Part 4: Feedback

```
lemma FreeGroup.lift_mulEquiv_image (iso : G ≃* H) (S : Set G) :
  FreeGroup.lift ((↑) : ↥(↑iso '' S) → H) =
    (↑iso : G →* H).comp ((FreeGroup.lift ((↑) : S → G)).comp
      (FreeGroup.freeGroupCongr (iso.toEquiv.image S).symm)) := by
ext ⟨_, s, hs, rfl⟩; simp [Equiv.image]
```

LEAN

Review: *"The fact that S is coerced from a term to a type has caused to you have to write the horrible* `FreeGroup.lift_mulEquiv_image` *which does not really look like a library lemma; it is not clean, it looks more like a hack written to dig yourself out of a hole, which should be a red flag."* - Source.

- Gemini 3.1 Pro explains the messy coercions to me.
- Reviewer proposes what we had from the start.

Back at the start, but learned something

```
class IsFinitelyPresented (G : Type*) [Group G] : Prop where
  out : ∃ (n : ℕ) (f : FreeGroup (Fin n) →* G),
    Function.Surjective f ∧ IsNormalClosureFG f.ker
```
LEAN

WHAT I LEARNED

Why was this definition good? Best of both `=*` and `S : Set G` approach.

- No `Nonempty`.
- No `Type` or `Type*` universe shenanigans with generating sets.
- No coercions, which makes API use easier. Potentially avoids diamond.
- Isomorphism-invariance proof still flows. Demo

SECTION

3. More of everything that went wrong

I knew nothing: misadventures in vibe-proving.

My background before start of project

- Finished Natural Number Game and loved it.
- Tried to do exercises like Mathematics in Lean.
  - Decided I wanted a more abstract exposition.
- Started reading Theorem-Proving in Lean.
  - Was still reading it by the time ItaLean2025 started…
  - Was very proud to understand `True.intro : True`.
    - *"True is simply true, and has a canonical proof, True.intro."* [3]

WHAT I LEARNED

If ItaLean didn't happen, I'd still be reading by now. Just start a project.

SUBSECTION

3a. I didn't actually know how to write mathlib code

Seriously, I spent the entire ItaLean2025 workshop watching people work.

My first setup: taught by an expert mathlib maintainer

- VS code
  - Error Lens extension
- A bunch of open tabs for reference:
  - Lean 4 tactic cheatsheet
  - mathlib4 docs
  - Lean search with natural language

USE OF AI

- I ended up preferring frontier models for natural language search.
- Frontier models not terrible at naming things using mathlib conventions.

My second setup: the Aristotle click and watch strategy

Aristotle is an automated theorem-proving API for Lean that's helped a lot.

- Blank page syndrome is real.
- Statements I wanted to prove seemed tautological.

USE OF AI

- Having Aristotle as a first solution taught me the proof-granularity required for mathlib.
- Watching proof states evolved taught me what needed to be done.

Aristotle demo (Jan 2026) vs merged version.

Writing code with Co-pilot and frontier models

Convenient, gradual, and can make custom requests i.e. “copy mathlib-style”.

USE OF AI

My three favorite Co-pilot prompts: I highlight a part of the code and go:

- *“Give me a lemma that says X.”*
- *“Why doesn’t this work? Fix and explain.”*
- *“Reviewer said this. ELI5 then help me fix.”*

- LLMs have infinite patience for me only limited by me paying for tokens.
- I usually have to fix it afterwards by steering or rewriting.
  - Still leagues better than working from nothing.

Writing code with frontier models

- **GPT-5.2** kept giving me stuff that didn't compile.
  - Had to keep copy-pasting the context for it to self-correct.
  - Writes way too much, not good at succinctness or API design.
- **GPT-5.3+** still does that but can figure out that loop by itself.
  - Good at writing targetted code, but the re-factoring is so-so.
    - Global refactor with GPT-5.3: before, after.
    - Variable refactor with GPT-5.4: before, after.
  - Still tends to write too much with no regard to API design.
    - Scaffolding and steering improves performance drastically.
    - Improvement might be due to human user's own eval function tbh.
- **Claude Opus 4.6:** kinda same but slightly better idk why.

SUBSECTION

3c. Writing the code isn't enough, need good API

Finding useful predicates

```
/-- A group is presented if it admits an isomorphism to a presented group. -/
class IsPresented (G : Type*) [Group G] : Prop where
  out: ∃ (α : Type*) (rels : Set (FreeGroup α)), Nonempty (G ≃* PresentedGroup rels)
```
LEAN

Review: *"This class is bad for two reasons. Firstly you're quantifying over universes, so this is really infinitely many definitions […]. Secondly, if you fix this in the natural way [by making both* `G` *and alpha in* `Type u` *]then this class is true all the time, so cannot be carrying any useful information."*

WHAT I LEARNED

- Type inference is costly, must be carried judiciously.

Refactor into API from point 1 to point 2

In math, saying “let x be y”, then you “have z” is standard.

This is meandering by mathlib standards.

Review:

- *“I can see the Aristotle origins […] It’s a process!”* Source.
- *“rewrite it to use less* `have ...; have ...; simpa` *style and more backwards reasoning? Or should some of the haves be intermediate lemmas?”* Source

Here’s what it looks like after.

Two examples of merged dependencies

- #34624: surjection between types induces surjection between free groups
  - My original proof was using induction. (`d165011`)
  - Reviewer helped me break it down into little re-usable chunks. (`225e8a3`)
    - Each chunk is a "bridge" from A to B. **SUPER FUN DEMO OPEN ME.**
- #34580: kernel of a homomorphism composed with an isomorphism
  - PRs can be as straightforward as that.

WHAT I LEARNED

If I had to formalize FP groups again, I'd PR like an onion.

Going back to my original code

Going back to my original code was a bit of a shock to say the least.



SECTION

4. The roadmap ahead

Getting ahead of myself yet again, what could possibly go wrong?

Proof of the Reidemeister-Schreier method has actually an efficient algebraic proof, but I really like the geometric ideas (esp: Cayley graphs).

- Cayley graphs are not formalized in mathlib.
- I got into geometric group theory because I loved the interplay between geometrical instincts and rigorous algebraic computations.
- When I would indulge that geometric instinct in informal math, I'd always get told my proofs were too "handwavy", and I'd have to write them again.

My updated dream: be able to communicate geometric ideas formally 🤩.

- Can we teach AI to do this? (I'm scared of asking bc then I won't have a job.)

Big thanks to everyone who guided me

USE OF HUMAN INTELLIGENCE

- **ItaLean2025 organizers:** for giving me the space to start.
- **Riccardo Brasca:** for getting me started.
- **Kevin Buzzard:** for his continued guidance over Zulip and Github.
- **Nehal Patel:** for investigating beginner struggles with me.
- **Thomas Browning:** for reviewing and rewriting my PRs with me.
- **Bhavik Mehta:** for teaching me type polymorphism.
- **Michael Rothgang:** for tips and reviews.
- **Sidharth Hariharan** & **Auguste Poiroux:** for last minute talk prep help.

Thank you!

- Thank you **David Loeffler** for organizing this conference.
- Thank **you** for listening!

Some links:

- The updated PR.
  - Merged as of this morning!!!
- The Finitely Presented Groups discussion thread on Zulip.
  - A lot of iterations went on, more than I can say in this talk.
- MadLean on Zulip
  - If you're in Madrid, these people kept me sane through the project.

References

[1] A. Vaswani *et al.*, "Attention Is All You Need," in *Advances in Neural Information Processing Systems 30 (NeurIPS)*, 2017.

[2] J.-Y. Girard, "Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur," Doctoral dissertation, 1972.

[3] J. Avigad, L. de Moura, S. Kong, and S. Ullrich, *Theorem Proving in Lean 4*. 2024. [Online]. Available: https://lean-lang.org/theorem_proving_in_lean4/

Thank you!