Introduction to Mathlib

Michael B. Rothgang (he/him)

Formalised mathematics group, Universität Bonn
Mathlib Initiative

Polyhedra in Lean workshop
FU Berlin, August 2026

Outline of today's talk

1. Lean and mathematical formalization
2. Mathlib and the formalization ecosystem
3. How mathlib operates
4. How to contribute to mathlib
5. Navigating mathlib

Section 1

Lean and mathematical formalization

What is Lean?

Lean is

- a functional programming language
- a theorem prover

What is Lean?

Lean is

- a functional programming language
- a theorem prover

In Lean you can

- write programs/definitions
- state theorems
- write proofs
- write programs that write proofs $\rightarrow$ tactics

Program verification, formal mathematics

What is Lean?

Is it LaTeX?

- No: LaTeX typesets mathematics, but does not check correctness

Is it like Sage or Mathematica?

- No: those programs perform computations, you don't build proofs in them

What is Lean?

Is it LaTeX?

- No: LaTeX typesets mathematics, but does not check correctness

Is it like Sage or Mathematica?

- No: those programs perform computations, you don't build proofs in them

Is it AI that proves theorems automatically?

- No, but such AI can be built on top of proof assistants

Is it a program that automatically proves theorems in first order logic?

- No, but it can use such programs in tactics

Lean is not AI

You can use AI to write Lean code.

Using AI is neither sufficient nor necessary to write Lean.

"Vibe-writing" code without understanding is not recommended.

How does it work?

You want to prove a theorem in Lean.

- You import other definitions and theorems you need from a library
- You write a statement in the Lean language
- You write a proof, using tactics to help you
- The Lean engine checks that your proof is correct

Now your theorem is part of the library, and can be used in other proofs.

How does it work?

You want to prove a theorem in Lean.

- You import other definitions and theorems you need from a library
- You write a statement in the Lean language
- You write a proof, using tactics to help you
- The Lean engine checks that your proof is correct

Now your theorem is part of the library, and can be used in other proofs.

Some other interactive theorem provers:
Rocq, Isabelle/HOL, Agda, Metamath, Mizar...

Lean and Mathlib

Lean is an interactive theorem prover.

Mathlib is its mathematical library

- 285 000 theorems about 135 000 definitions (many automatically generated)
- 770 contributors
- 60 reviewers, among which 28 maintainers who accept contributions
- All kinds of math: a monolithic library, because any part of math could be useful in combination with any other, and different parts of the library should be compatible.

What has been formalised already: let's guess

- Banach–Schauder open mapping theorem
- Birkhoff Ergodic Theorem
- Mandelbrot set is connected
- Cauchy-Kovalevskaya Theorem on existence of an analytical solution of an analytical PDE
- Denjoy's theorem: a C^2 orientation-preserving diffeomorphism of the circle with an irrational rotation number is conjugate to a rotation
- Sphere eversion
- Existence of Haar measure
- Existence of a smooth partition of unity
- Feit–Thompson theorem/odd order theorem
- Fermat's Last Theorem
- Four colour theorem
- Freyd–Mitchell embedding theorem
- Galois correspondence
- Herman-Yoccoz theorem on linearization of a circle diffeomorphism
- Jordan curve theorem
- Liouville theorem: an entire holomorphic function is a constant
- Hilbert's Nullstellensatz
- Picard-Lindelöf theorem (existence and uniqueness of solutions of ODEs)
- Poincaré-Bendixson Theorem
- Poincaré recurrence theorem
- Prime number theorem
- Sard's Theorem
- The continuum hypothesis is independent of ZFC.

Let's guess: the answer

Only three are not formalised yet (AFAIK)

- Denjoy's theorem on rotation number
- Herman–Yoccoz theorem on linearization of a circle diffeomorphism
- Fermat's Last Theorem

Let's guess: the answer

Only three are not formalised yet (AFAIK)

- Denjoy's theorem on rotation number
- Herman–Yoccoz theorem on linearization of a circle diffeomorphism
- Fermat's Last Theorem (in progress)

Notable formalisation projects

2005 Four colour theorem

2012 Odd Order Theorem

Notable formalisation projects

2005 Four colour theorem

2012 Odd Order Theorem

2014 Kepler's conjecture (Hales et al)

2019 Ellenberg-Gijswijt's result on the cap set conjecture

2022 Liquid Tensor Experiment (Commelin et al): fundamental lemma about condensed mathematics

Notable formalisation projects

2005 Four colour theorem

2012 Odd Order Theorem

2014 Kepler's conjecture (Hales et al)

2019 Ellenberg-Gijswijt's result on the cap set conjecture

2022 Liquid Tensor Experiment (Commelin et al): fundamental lemma about condensed mathematics

2022 Unit fractions project

2023 Upper bound on diagonal Ramsey numbers

Notable formalisation projects

2005 Four colour theorem

2012 Odd Order Theorem

2014 Kepler's conjecture (Hales et al)

2019 Ellenberg-Gijswijt's result on the cap set conjecture

2022 Liquid Tensor Experiment (Commelin et al): fundamental lemma about condensed mathematics

2022 Unit fractions project before referee report

2023 Upper bound on diagonal Ramsey numbers before referee report

Notable formalisation projects

2005 Four colour theorem

2012 Odd Order Theorem

2014 Kepler's conjecture (Hales et al)

2019 Ellenberg-Gijswijt's result on the cap set conjecture

2022 Liquid Tensor Experiment (Commelin et al): fundamental lemma about condensed mathematics

2022 Unit fractions project before referee report

2023 Upper bound on diagonal Ramsey numbers before referee report

2023 Marton's polynomial Freiman-Rusza conjecture (Tao et al)

Notable formalisation projects

2005 Four colour theorem

2012 Odd Order Theorem

2014 Kepler's conjecture (Hales et al)

2019 Ellenberg-Gijswijt's result on the cap set conjecture

2022 Liquid Tensor Experiment (Commelin et al):
fundamental lemma about condensed mathematics

2022 Unit fractions project before referee report

2023 Upper bound on diagonal Ramsey numbers before referee report

2023 Marton's polynomial Freiman-Rusza conjecture (Tao et al)
took 3 weeks; complete before paper submitted

Notable formalisation projects

2005 Four colour theorem

2012 Odd Order Theorem

2014 Kepler's conjecture (Hales et al)

2019 Ellenberg-Gijswijt's result on the cap set conjecture

2022 Liquid Tensor Experiment (Commelin et al): fundamental lemma about condensed mathematics

2022 Unit fractions project before referee report

2023 Upper bound on diagonal Ramsey numbers before referee report

2023 Marton's polynomial Freiman-Rusza conjecture (Tao et al) took 3 weeks; complete before paper submitted

2025 Disproof of the Aharoni–Korman conjecture (Mehta)

2025 Carleson's theorem

Notable formalisation projects

2005 Four colour theorem

2012 Odd Order Theorem

2014 Kepler's conjecture (Hales et al)

2019 Ellenberg-Gijswijt's result on the cap set conjecture

2022 Liquid Tensor Experiment (Commelin et al): fundamental lemma about condensed mathematics

2022 Unit fractions project before referee report

2023 Upper bound on diagonal Ramsey numbers before referee report

2023 Marton's polynomial Freiman-Rusza conjecture (Tao et al) took 3 weeks; complete before paper submitted

2025 Disproof of the Aharoni–Korman conjecture (Mehta)

2025 Carleson's theorem

2026 Viazovska's 8d sphere packing (partially AI)

2026 Prime Number theorem (classical and modern error bounds) (partially AI)

Notable AI formalization projects

- Jordan–Schönflies theorem
- Bieberbach conjecture (de Branges' theorem)
- Conway–Schneeberger 15 theorem
- Odd Order theorem
- 17 wallpaper groups
- Green–Tao theorem (primes in arithmetic progressions)
- Counterexample to the Erdos unit distance conjecture

Notable AI formalization projects

- Jordan–Schönflies theorem
- Bieberbach conjecture (de Branges' theorem)
- Conway–Schneeberger 15 theorem
- Odd Order theorem
- 17 wallpaper groups
- Green–Tao theorem (primes in arithmetic progressions)
- Counterexample to the Erdos unit distance conjecture
- current projects include the Stacks project, Perelman's theorem, the classification of finite simple groups

Section 2

Mathlib and the formalization ecosystem

Lean and Mathlib, and other projects

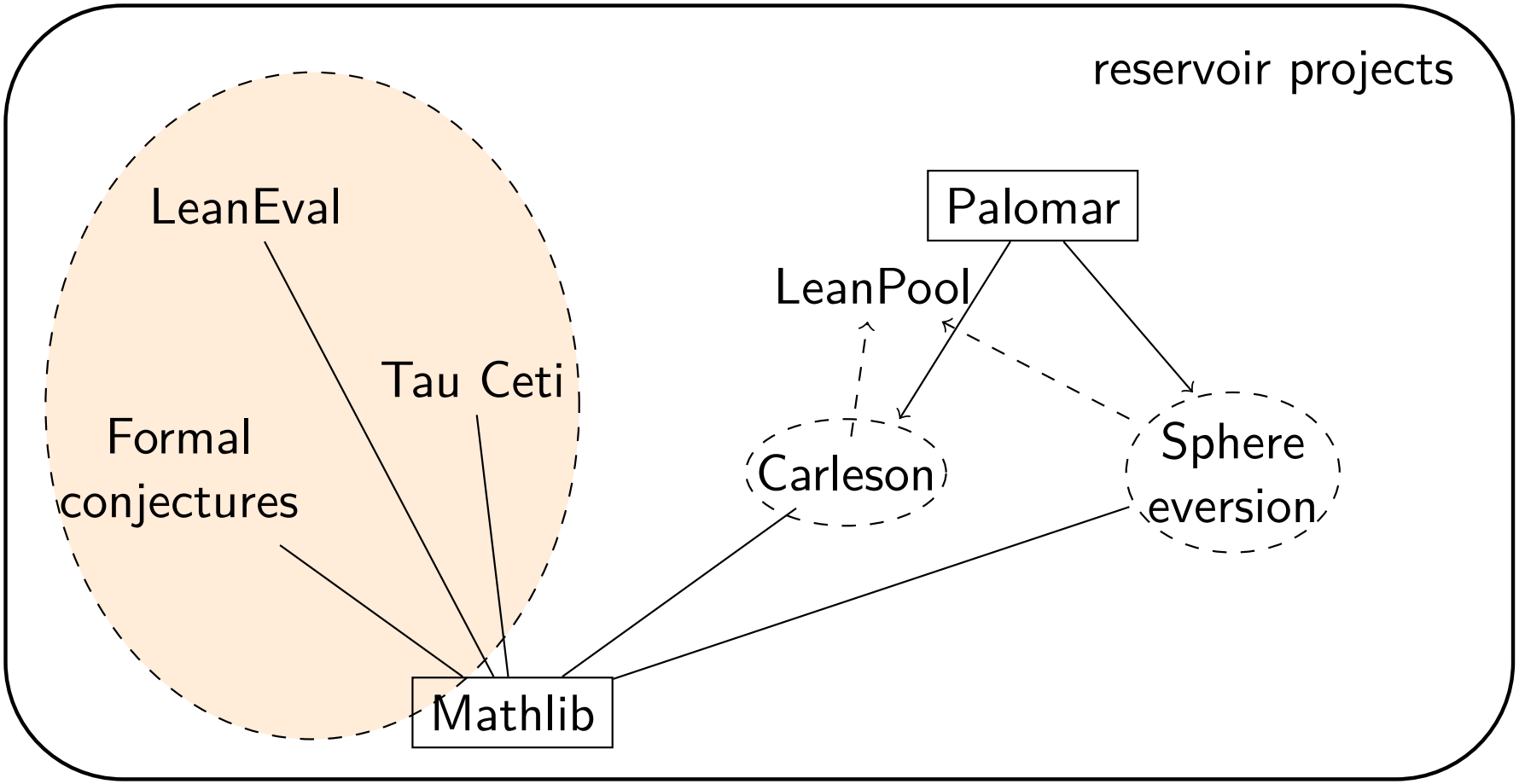
Lean is an interactive theorem prover.

Mathlib is its integrated, coherent mathematical library

Many ongoing projects use Lean and Mathlib:

- Fermat's last theorem
- Brownian motion and stochastic integrals
- Toric varieties
- Equational theories of magmas
- Formal conjectures
- ...

Lean's formal math ecosystem



Lean's formal math ecosystem

- **Reservoir**: registry for "all" Lean packages
- **LeanEval**: (autoformalization system) benchmark of hard formalization problems. Solved problems with short statements; submitted solutions stay secret.
- **Formal conjectures**: repository of formalized mathematical conjectures
- **Tau Ceti**: AI-generated library of formal mathematics (building on and following mathlib)

Lean's formal math ecosystem

- **Reservoir**: registry for "all" Lean packages
- **LeanEval**: (autoformalization system) benchmark of hard formalization problems. Solved problems with short statements; submitted solutions stay secret.
- **Formal conjectures**: repository of formalized mathematical conjectures
- **Tau Ceti**: AI-generated library of formal mathematics (building on and following mathlib)
- **LeanPool**: monorepository collecting "significant" projects outside of mathlib
  "AFP for Lean" — duplicate or conflicting definitions posible
- **Palomar** indexes significant mathematical formalizations: clear main statements, robustly audited
  Could become the arXiv for mathematical formalizations.

Lean's formal math ecosystem

- **Reservoir**: registry for "all" Lean packages
- **LeanEval**: (autoformalization system) benchmark of hard formalization problems. Solved problems with short statements; submitted solutions stay secret.
- **Formal conjectures**: repository of formalized mathematical conjectures
- **Tau Ceti**: AI-generated library of formal mathematics (building on and following mathlib)
- **LeanPool**: monorepository collecting "significant" projects outside of mathlib
  "AFP for Lean" — duplicate or conflicting definitions posible
- **Palomar** indexes significant mathematical formalizations: clear main statements, robustly audited
  Could become the arXiv for mathematical formalizations.

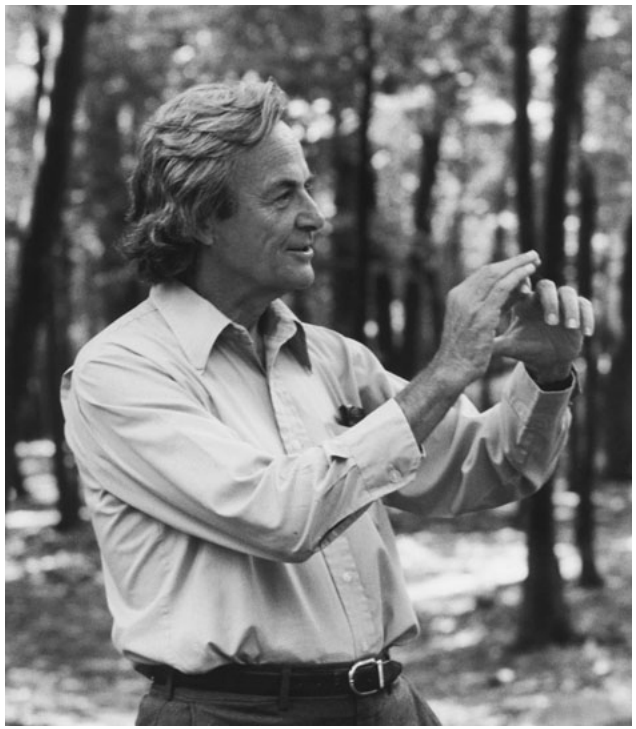
Mathlib is the foundation for all of these.

Why formalize? "Must I?"

- mathematical formalization has the potential to revolutionize mathematics
- the future of AI for mathematics lies in formalization

Why formalize? (cont.)

~~Physics~~ Formalization is like sex:
sure, there are practical benefits, but that's not why we do it.



Why formalize? "It's fun!"

Formalization is fun — like a video game or programming

Correctness/Verification

Lean checks the proof down to its axioms → certainty of correctness.

There are always folklore results, or literature gaps known only to experts.
Yet, the foundations of our various domains are solid. That's not where the biggest gain is.

Correctness/Verification

Lean checks the proof down to its axioms → certainty of correctness.

There are always folklore results, or literature gaps known only to experts.
Yet, the foundations of our various domains are solid. That's not where the biggest gain is.

New papers may contain mistakes,

- but we detect them when we try to reuse the results,
- requiring formal proofs could put a big burden on the authors
- but still, wouldn't it be great to review a paper with certified proofs?
  - You would still need to review the definitions!

Formalization can verify AI-generated mathematics

Formalization can change mathematics

verified proofs, down to the axioms

reviewing: focus on importance, techniques and presentation

enable massive collaboration

understanding: expand any detail

creation: faster modification and generalisation

AI-generated mathematics is only trustworthy when formalised.

Understanding and creation

- understanding: reader chooses amount of detail

  Demo by Patrick Massot and Kyle Miller:
  https://kmill.github.io/informalization/ContinuousFrom.html

Understanding and creation

- understanding: reader chooses amount of detail

  Demo by Patrick Massot and Kyle Miller:
  https://kmill.github.io/informalization/ContinuousFrom.html

- creation: can this lemma be generalised? unused assumptions?
- database of theorems: searching known and related results only requires statements of main results

Community and collaboration

Lean checks the proof correctness → enables large scale collaboration.

Example: Carleson project

- Carleson: Fourier series of $f \in L^2$ converges pointwise a.e. to f
- subtle statement; all known proofs are hard and technical
- formalize modern generalization by Thiele et al to doubling metric measure spaces
- 28 contributors, many undergraduates or not analysts
- a website, a github repository, a blueprint, a dependency graph, a Zulip channel
- a detailed LaTeX proof ("blueprint")
- 35–40k lines of new Lean code
- many basics in harmonic analysis, suitable for Mathlib

To summarize: why mathlib? A human-curated community project

- Build trust in mathematical literature: audited definitions and statements with formal proof
- Just a large open library of formal mathematics: use Tau Ceti
- Modern Bourbaki: digest and curate mathematics into general form
- Discover formalization best practices for each area
- Community: create a common good; collaborate across mathematical disciplines

Mathlib's values

- Trustworthy: expert-audited definitions and statements
- Open source
- maintainable

Mathlib's values

- Trustworthy: expert-audited definitions and statements
- Open source
- maintainable
- General definitions
- Weak hypotheses, strong conclusions
- classical, not constructive
- accessibility for non-experts

See https://leanprover-community.github.io/contribute/values.html for details.

Examples of mathlib's values

- no separate definition of vector space:
  "V is a K-vector space" is written as `Module K V`
  submodules instead of subspaces
- No definition of "algebraic variety":
  say "an XYZ scheme" instead
- Work with linear maps more than matrices
- Trivial submodule of V is called "bot": is the
  bottom element of the lattice formed by submodules of V
- Definition of manifolds is more general than any textbook

Example of mathlib's values: weaker hypothesis

```
example {s : Set ℝ} {f f' : ℝ → ℝ}
    (hs : MeasurableSet s)
    (hf' : ∀ x ∈ s, HasDerivWithinAt f (f' x) s x)
    (hf : InjOn f s) (g : ℝ → E) :
    ∫ x in f '' s, g x = ∫ y in s, |f' y| • g (f y) :=
  integral_image_eq_integral_abs_deriv_smul hs hf' hf g
```

Has weaker assumptions than usual:

- s is not required to be open;
- f is not required to be C^n
  (integral of non-integrable functions has junk value 0),
- g is not required to be integrable.

How does mathlib operate?

- open source; "standard github project"
- Continuous integration; not rocket science rule (Graydon Hoare)
- Mandatory code review before merge

Why do we need code review?

- catch bad actors and prevent malicious code
  (no problem for very small groups)
- quality control: want to build a library that scales

Why do we need code review?

- catch bad actors and prevent malicious code
  (no problem for very small groups)
- quality control: want to build a library that scales
- teaching opportunity for new(er) contributors
- build shared understanding $\rightarrow$ resilience

Organising mathlib reviewing

- everybody can review: look at review guidelines
- 28 maintainers, and 32 additional reviewers (experienced community members with a history of helpful reviews)
- three-stage model: anybody reviews → "approve" on github; mathlib reviewers → "maintainer merge" maintainer → final merge
- statistics: about 1/3 merged PRs was maintainer merge'd first

You can help!

Ask not what mathlib can do for you —
ask what you can do for mathlib!

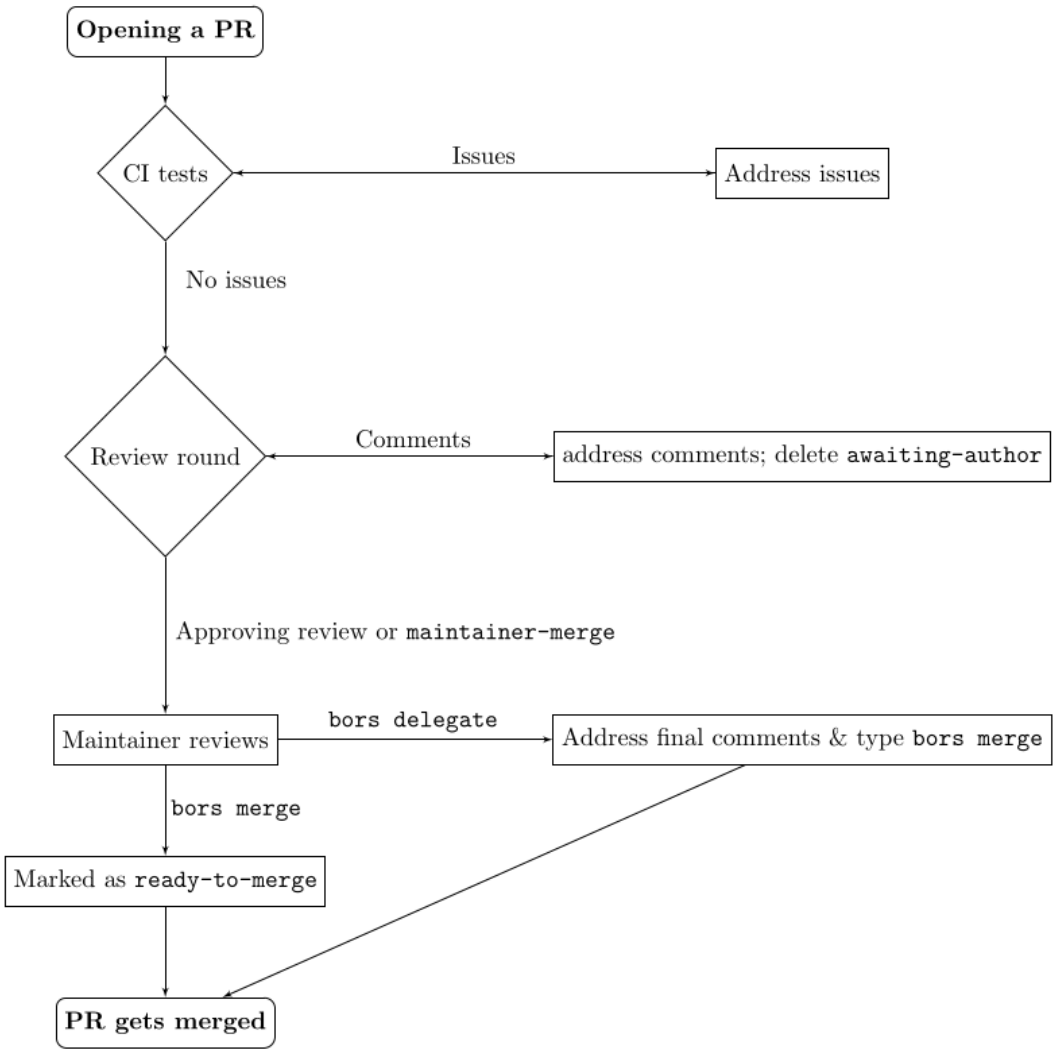
- Mathlib can only grow if its reviewer base grows
- Anybody can review!
- Save the date: reviewer bootcamp, mid-January 2027, near Bonn

Community organisation

- Github: code changes, discussing specific pull requests
- Zulip: discussion broader points,
  connect as community. Job postings, event announcements, questions about Lean
- Growing number of live events: `https://leanprover-community.github.io/events.html`

How to contribute to mathlib

Fork the repository and make a pull request.



Courtesy of Ruth Plümer.

How to contribute: tips and tricks

- **No AI-written comments**; disclose AI usage
- start small. medium-term, aim for 100–200 lines of code
- queueboard shows if your PR is ready for review: https://leanprover-community.github.io/queueboard/on_the_queue.html
- follow mathlib's style guide
- mathlib has strict naming convention
- ask in # PR reviews on Zulip (or ask your local expert to review)

How to navigate mathlib: finding something

- What's the right statement? Perhaps generalise!
  e.g. metric vs topological spaces, submodules vs order-theoretic lattice, vector spaces versus modules, etc.
- Know the Lean "spelling": Loogle is your friend

Loogle demos

"The composition of continuous functions is continuous."

"A linear isomorphism is surjective."

How to navigate mathlib: finding something

- Have you checked the mathlib phrasebook already?
- What's the right statement? Perhaps generalise!
  e.g. metric vs topological spaces, submodules vs order-theoretic lattice, vector spaces versus modules, etc.
- Know the Lean "spelling": Loogle is your friend
- Low-tech: search source code; scroll through file for related content; mathlib documentation
- Leansearch allows searching in natural language: search for "convex cone" and get the right answer
- search tactics: `exact?`, `apply?`
  modern versions: `rw??` or `#click_suggestions`
- Short demo!

Summary

1. Formalising is fun, and has the potential to transform mathematics.
2. There is a vibrant ecosystem of formalised maths in Lean, based on mathlib.
3. Mathlib welcomes your contributions, and it needs your help!

Summary

1. Formalising is fun, and has the potential to transform mathematics.
2. There is a vibrant ecosystem of formalised maths in Lean, based on mathlib.
3. Mathlib welcomes your contributions, and it needs your help!

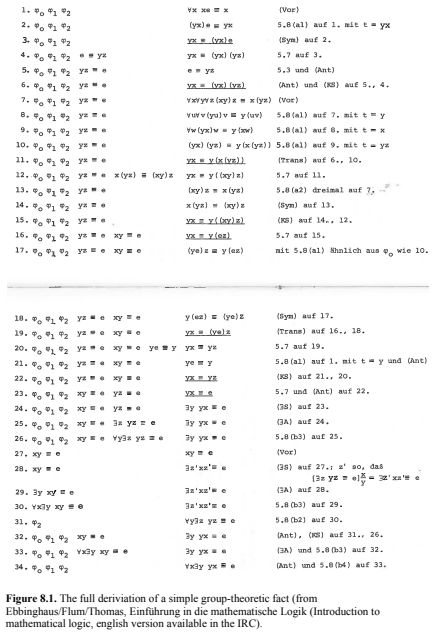
Thanks for listening! Any questions?

Bonus slides

- ITPs vs ATPs vs more rigorous proofs
- How review somebody else's formalization
- Scope of mathlib
- Mathlib governance/how are decisions made

What does formalisation mean?

answer 1: humans write more detailed proofs



1. $\varphi_0\,\varphi_1\,\varphi_2$	$\forall x\; xe = x$	(Vor)
2. $\varphi_0\,\varphi_1\,\varphi_2$	$(yx)e = yx$	5.8(a1) auf 1. mit $t = yx$
3. $\varphi_0\,\varphi_1\,\varphi_2$	$yx = (yx)e$	(Sym) auf 2.
4. $\varphi_0\,\varphi_1\,\varphi_2$ $e = yz$	$yx = (yx)(yz)$	5.7 auf 3.
5. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$	$e = yz$	5.3 und (Ant)
6. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$	$yx = (yx)(yz)$	(Ant) und (KS) auf 5., 4.
7. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$	$\forall xy\forall z(xy)z = x(yz)$	(Vor)
8. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$	$\forall u\forall v(yu)v = y(uv)$	5.8(a1) auf 7. mit $t = y$
9. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$	$\forall w(yx)w = y(xw)$	5.8(a1) auf 8. mit $t = x$
10. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$	$(yx)(yz) = y(x(yz))$	5.8(a1) auf 9. mit $t = yz$
11. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$	$yx = y(x(yz))$	(Trans) auf 6., 10.
12. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$ $x(yz) = (xy)z$	$yx = y((xy)z)$	5.7 auf 11.
13. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$	$(xy)z = x(yz)$	5.8(a2) dreimal auf 7.,
14. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$	$x(yz) = (xy)z$	(Sym) auf 13.
15. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$	$yx = y((xy)z)$	(KS) auf 14., 12.
16. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$ $xy = e$	$yx = y(ez)$	5.7 auf 15.
17. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$ $xy = e$	$(ye)z = y(ez)$	mit 5.8(a1) Ähnlich aus φ_0 wie 10.
18. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$ $xy = e$	$y(ez) = (ye)z$	(Sym) auf 17.
19. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$ $xy = e$	$yx = (ye)z$	(Trans) auf 16., 18.
20. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$ $xy = e$ $ye = y$	$yx = yz$	5.7 auf 19.
21. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$ $xy = e$	$ye = y$	5.8(a1) auf 1. mit $t = y$ und (Ant)
22. $\varphi_0\,\varphi_1\,\varphi_2$ $yz = e$ $xy = e$	$yx = yz$	(KS) auf 21., 20.
23. $\varphi_0\,\varphi_1\,\varphi_2$ $xy = e$ $yz = e$	$yx = e$	5.7 und (Ant) auf 22.
24. $\varphi_0\,\varphi_1\,\varphi_2$ $xy = e$ $yz = e$	$\exists y\; yx = e$	(3S) auf 23.
25. $\varphi_0\,\varphi_1\,\varphi_2$ $xy = e$ $\exists z\; yz = e$	$\exists y\; yx = e$	(3A) auf 24.
26. $\varphi_0\,\varphi_1\,\varphi_2$ $xy = e$ $\forall y\exists z\; yz = e$	$\exists y\; yx = e$	5.8(b3) auf 25.
27. $xy = e$	$xy = e$	(Vor)
28. $xy = e$	$\exists z'xz' = e$	(3S) auf 27., z' so, daß $(\exists z\; yz = e)^z_y = \exists z'xz' = e$
29. $\exists y\; xy = e$	$\exists z'xz' = e$	(3A) auf 28.
30. $\forall x\exists y\; xy = e$	$\exists z'xz' = e$	5.8(b3) auf 29.
31. φ_2	$\forall y\exists z\; yz = e$	5.8(b2) auf 30.
32. $\varphi_0\,\varphi_1\,\varphi_2$ $xy = e$	$\exists y\; yx = e$	(Ant), (KS) auf 31., 26.
33. $\varphi_0\,\varphi_1\,\varphi_2$ $\forall x\exists y\; xy = e$	$\exists y\; yx = e$	(3A) und 5.8(b3) auf 32.
34. $\varphi_0\,\varphi_1\,\varphi_2$	$\forall x\exists y\; yx = e$	(Ant) und 5.8(b4) auf 33.

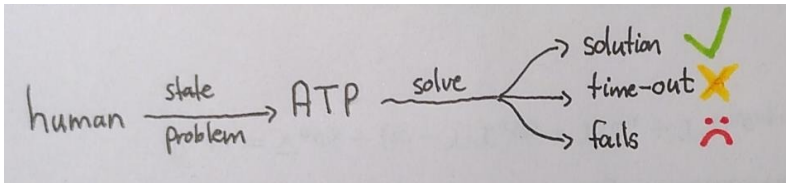
Figure 8.1. The full deriviation of a simple group-theoretic fact (from Ebbinghaus/Flum/Thomas, Einführung in die mathematische Logik (Introduction to mathematical logic, english version available in the IRC).

87

problem: impractical in the large
how to formalise “draw a picture”?

What does formalisation mean? (cont.)

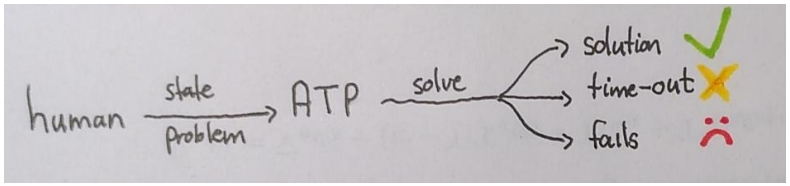
answer 2: automated theorem proving



problems: hit or miss; opaque

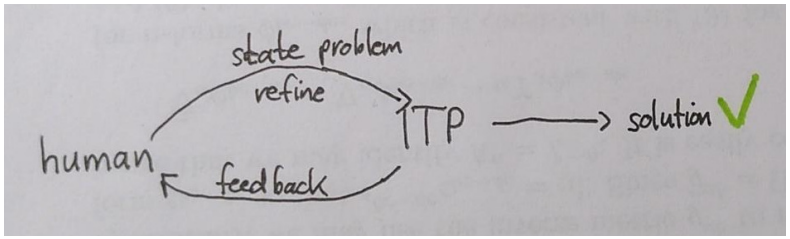
What does formalisation mean? (cont.)

answer 2: automated theorem proving



problems: hit or miss; opaque

answer 3: interactive theorem proving



Reviewing somebody else's formalization

- Be cautious about building it locally
- Double-check the main statement
- Check definitions particularly careful; use mathlib ones if you can.
- Checking good-faith proofs: `#print axioms foo` should say (a subset of) `propext, Classical.choice, Quot.sound`
- Fully robust checking: use comparator
  Create a Challenge file with your main statement; your proof is verified against that.

"Have you submitted to Palomar yet?"

What do we review? The scope of mathlib

- "all areas of mathematics", of course!

What do we review? The scope of mathlib

- "all areas of mathematics", of course! Right?
- really everything? just research? just current research?
  controversial topic

What do we review? The scope of mathlib

- "all areas of mathematics", of course! Right?
- really everything? just research? just current research? controversial topic
- practical considerations: maintainer expertise? review capacity?
- case studies: combinatorial game theory, PhysLean, CSLib, `Mathlib/Computability`

Governance around mathlib: who makes the decisions?

- large distributed project:
  much communication is asynchronous, via text
- different community teams: admin, webpage, CI admins, code of conduct, mathlib maintainers (and reviewers)
  Details at
  https://leanprover-community.github.io/teams.html
- separate entities: Lean FRO; mathlib initiative

Governance around mathlib: who makes the decisions?

- large distributed project:
  much communication is asynchronous, via text
- different community teams: admin, webpage, CI admins, code of conduct, mathlib maintainers (and reviewers)
  Details at
  https://leanprover-community.github.io/teams.html
- separate entities: Lean FRO; mathlib initiative
- technical leadership: maintainers team
- MI community coordinator to help coordinate this

Decision-making in mathlib

- try to avoid majority votes,
  consensus-based decision making (if possible)
- involve broader community when possible — almost all technical discussions happen in the open!
- "serving leadership" — lots of listening, empathy, looking for positive-sum outcomes

Challenge: dealing with AI slop

AI slop projects posted on zulip

LLM-written Zulip posts (forbidden!)

LLM-generated PRs (without good understanding)

using LLM to respond to review comments

Problems of AI slop

AI slop projects posted on zulip; LLM-written comments

LLM-generated PRs (without good understanding)
using LLM to respond to review comments

Can be well-meant, but risks burning out reviewers and maintainers

Quality bar: AI-generated code is not good enough

signal to noise ratio on zulip

often invisible as moderated!

Current solutions

- moderation tooling on zulip
- AI policy: **must not** use AI to write comments on Zulip or github
- AI policy: AI use must be disclosed on the PR
- label `LLM-generated` to highlight LLM-generated PRs
- further policy changes under discussion