From Mathematical Components to Mathlib

Jeremy Avigad

Department of Philosophy
Department of Mathematical Sciences

Hoskinson Center for Formal Mathematics

Carnegie Mellon University

December 7, 2022

Outline

- Some history
- Lean and mathlib today
- An overview of the library
- Comparisons between mathlib and Mathematical Components
- Mathematicians' point of view

A personal history

1989	BA in Mathematics, Harvard
1995	PhD in Mathematics, UC Berkeley
1990's	I discovered Isabelle and Coq
2002	Quadratic reciprocity in Isabelle
2004	The prime number theorem in Isabelle
2009–2010	Sabbatical at INRIA/Saclay, Mathematical Components
2013	Starting working with Lean
2015	Homotopy limits in Coq
2017	The central limit theorem in Isabelle

A history of Lean

2013 Lean 0.1 is released

2014 Lean 2 is released

2016 Lean 3 is released

2017 The Lean community is born.

- Big Proof meeting at the Isaac Newton Institute.
- Mario Carneiro splits off mathlib.
- Kevin Buzzard, Patrick Massot, Sébastien Gouëzel, and others get involved.

2022 Lean 4 is released

A personal history

Some Carnegie Mellon connections to Lean:

Jason Rute (PhD student), Cody Roux (Postdoc), Floris van Doorn (PhD student), Robert Lewis (PhD student), Mario Carneiro (PhD student, now Hoskinson Center postdoc), Jakob von Raumer (visiting MS student), Johannes Hölzl (postdoc), Andrew Zipperer (MS student), Sebastian Ullrich (visiting MS student), Minchao Wu (MS student), Gabriel Ebner (visiting PhD student, first Hoskinson Center postdoc), Paula Neeley (PhD student), Simon Hudon (postdoc), Bruno Bentzen (postdoc), Seul Baek (PhD student), Edward Ayers (Hoskinson Center postdoc), Alexander Bentkamp (visited as a PhD student and postdoc), Ramon Fernández Mir (visiting PhD student), Bartosz Piotrowski (visiting PhD student), Zhangir Azerbayev (visiting undergraduate), Wojciech Naworcki (PhD student), Joshua Clune (PhD student), Cayden Codel (PhD student)

Also Tom Hales, Reid Barton, Jesse Han, and Kody Vajjha at the University of Pittsburgh.

Design goals

Initially, Leo's goal was to bring automation and interaction closer together.

The design of Lean was heavily influenced by Isabelle, PVS, ACL2, Agda, HOL Light, Coq, Coq/SSReflect, ...

Insights from Mathematical Components: dependent type theory, encoding an algebraic hierarchy, the rewrite tactic, parts of the library.

Initially, the goal was to be as constructive as possible in Lean, but also support classical reasoning.

Lean 2 even had a Homotopy Type Theory mode.

Leo later distinguished constructivity from *computability*.

Design goals

Lean 3 solved some performance problems, and had a faster, more deterministic elaborator.

Most notably, it introduced a metaprogramming language (i.e. the object language, with hooks to internals).

Lean 4 is a performant functional programming language, with clever handling of monads, memory management, imperative features.

It is written in Lean 4 itself (so it's easy to write performant tactics and automation).

It has flexible syntax and a hygienic macro system, and also impressive editor interfaces and user interactions.

The port of mathlib is underway.

Lean and mathlib today

The Lean community experienced explosive growth through the pandemic. (We'll see statistics later.)

Lean has been getting good press:

- *Quanta:* "Building the mathematical library of the future"
- *Quanta:* "At the Math Olympiad, computers prepare to go for the gold"
- *Nature:* "Mathematicians welcome computer-assisted proof in 'grand unification' theory"
- *Quanta:* "Proof Assistant Makes Jump to Big-League Math"

Kevin Buzzard recently gave a talk, "The Rise of Formalism in Mathematics," at the 2022 International Congress of Mathematicians.

Lean and mathlib today

There are have been lots of Lean-related meetings:
https://leanprover-community.github.io/events.html

Some achievements:

- the formalization of the independence of the continuum hypothesis (Han and van Doorn)
- the formalization of perfectoid spaces (Buzzard, Commelin, and Massot)
- the liquid tensor experiment (Commelin, Topaz, and many others)
- the formalization of Bloom's theorem on unit fractions (Bloom, Mehta)
- the formalization of the sphere eversion theorem (Massot, Nash, van Doorn)

Lean and mathlib today

On December 5, 2020, Peter Scholze challenged anyone to formally verify some of his recent work with Dustin Clausen.

Johan Commelin led the response from the Lean community. On June 5, 2021, Scholze acknowledged the achievement.

"Exactly half a year ago I wrote the Liquid Tensor Experiment blog post, challenging the formalization of a difficult foundational theorem from my Analytic Geometry lecture notes on joint work with Dustin Clausen. While this challenge has not been completed yet, I am excited to announce that the Experiment has verified the entire part of the argument that I was unsure about. I find it absolutely insane that interactive proof assistants are now at the level that within a very reasonable time span they can formally verify difficult original research."

Lean and mathlib today

Scholze went on:

"When I wrote the blog post half a year ago, I did not understand why the argument worked. . . .

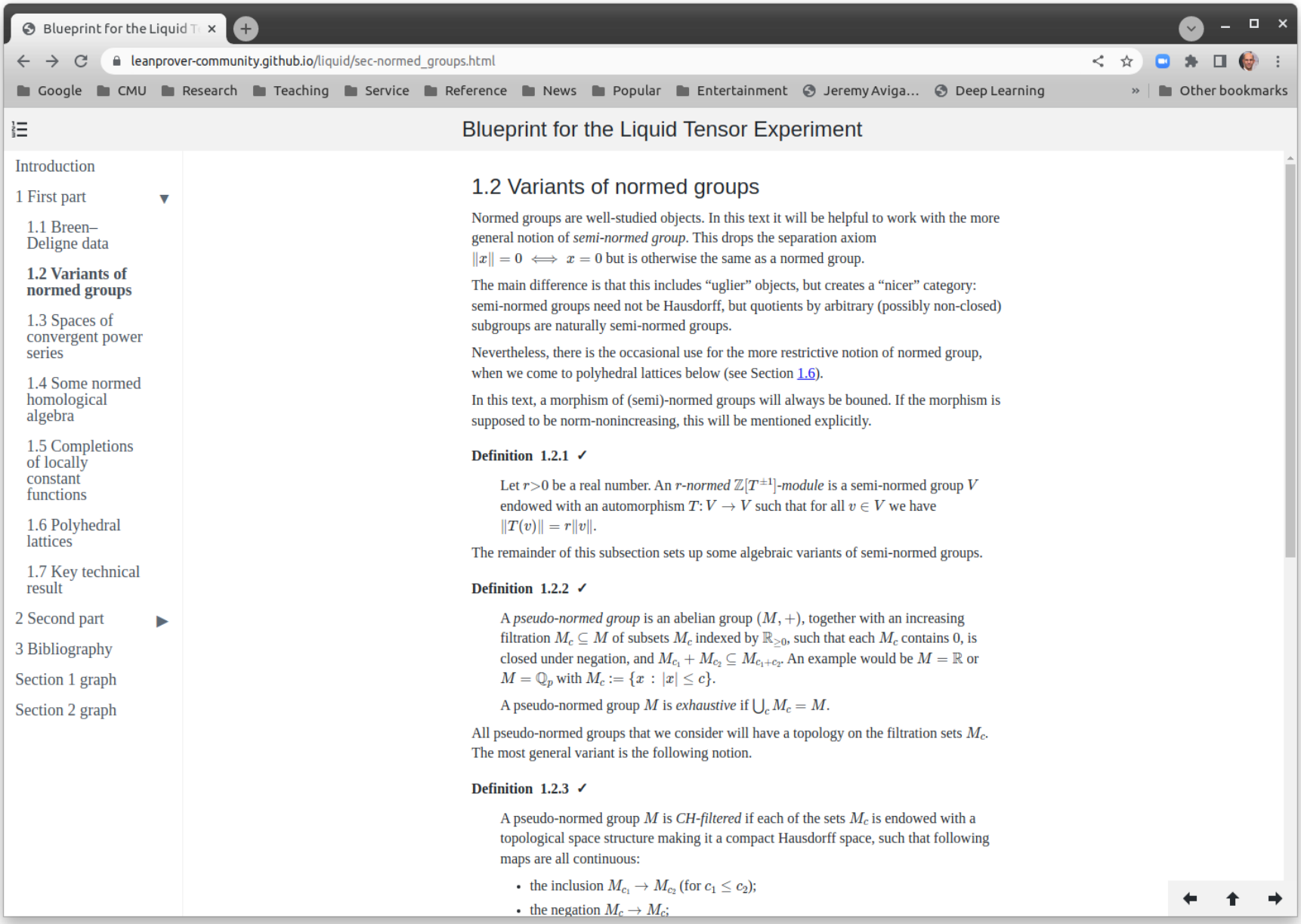
But during the formalization, a significant amount of convex geometry had to be formalized . . . and this made me realize that . . . the key thing happening is a reduction from a non-convex problem over the reals to a convex problem over the integers."

Lean and mathlib today

The liquid tensor experiment is also a model for digital collaboration.

- The formalization was in kept in a shared online repository.
- Participants followed an informal blueprint with links to the repository.
- Participants were in constant contact on Zulip.
- Lean made sure the pieces fit together.
- The recent formalization of sphere eversion uses the same framework.

Lean and mathlib today



Lean and mathlib today

In early 2022, Thomas Bloom solved a problem posed by Paul Erdős and Ronald Graham.

The headline in Quanta read "Math's 'Oldest Problem Ever' Gets a New Answer."

Within in a few months, Bloom and Bhavik Mehta verified the correctness of the proof in Lean.

Lean and mathlib today

Timothy Gowers
@wtgowers · Jun 13

Very excited that Thomas Bloom and Bhavik Mehta have done this. I think it's the first time that a serious contemporary result in "mainstream" mathematics doesn't have to be checked by a referee, because it has been checked formally. Maybe the sign of things to come ... 1/

> **Kevin Buzzard** @XenaProject · Jun 12
>
> Happy to report that Bloom went on to learn Lean this year and, together with Bhavik Mehta, has now formalised his proof in Lean b-mehta.github.io/unit-fractions/ (including formalising the Hardy-Littlewood circle method), finishing before he got a referee's report for the paper ;-)
>
> Show this thread

2 26 141

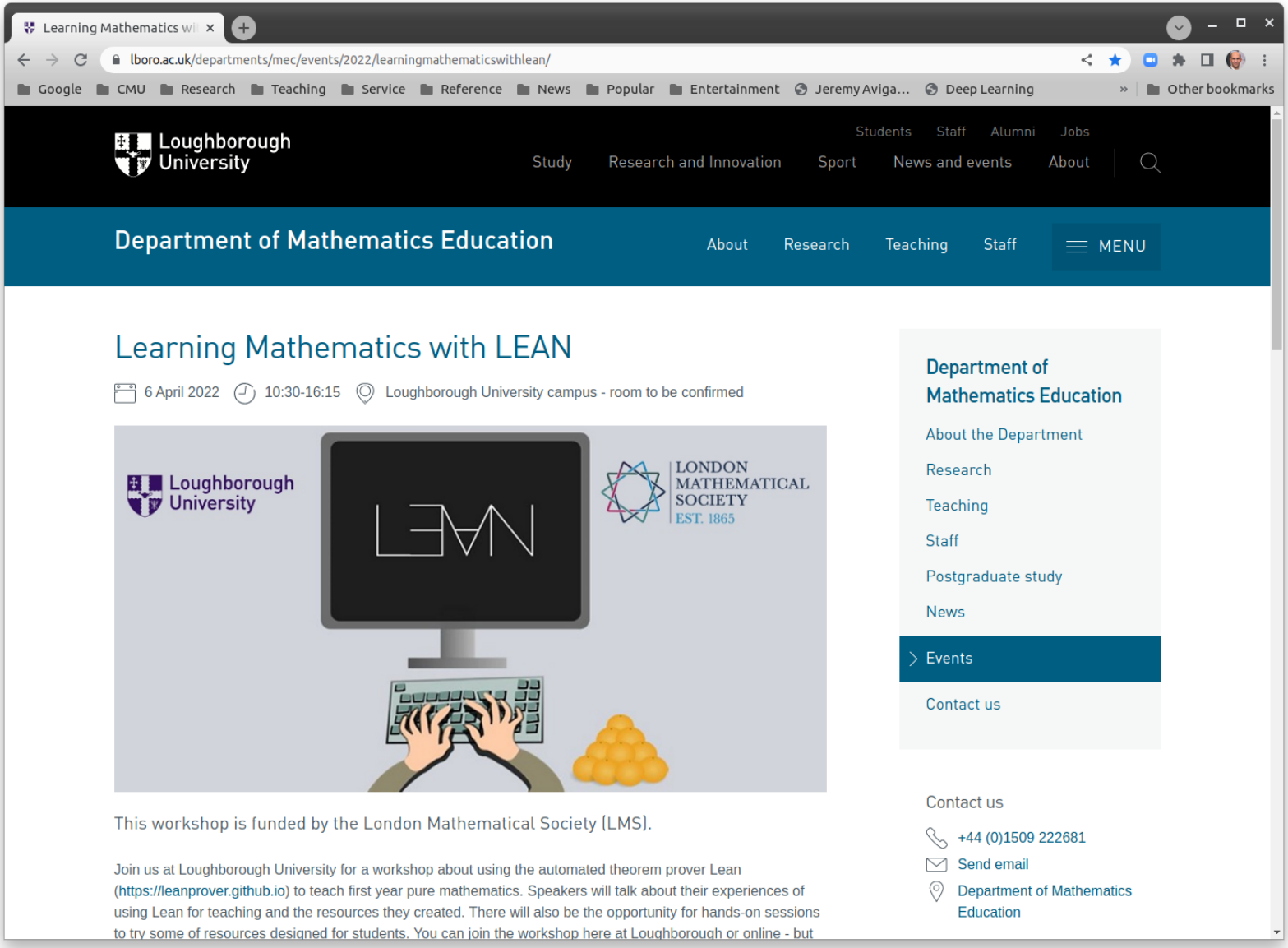
Lean and mathlib today

A number of people are teaching courses with Lean.

More importantly: a number of people (Patrick Massot, Rob Lewis, Heather Macbeth, Gihan Marasingha, Sina Hazratpour, Dan Velleman, . . .) are exploring ways to use Lean to teach *mathematics*.

There have been workshops and conference sessions dedicated to learning how to use the technology effectively.

Lean and mathlib today



Lean and mathlib today

Since Daniel Selsam's *Grand IMO Challenge*, Lean has been a target platform for machine learning for mathematics.

- Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, Stanislas Polu, Proof Artifact Co-training for Theorem Proving with Language Models
- Kunhao Zheng, Jesse Michael Han, Stanislas Polu, MiniF2F: a cross-system benchmark for formal Olympiad-level mathematics
- Ayush Agrawal, Siddhartha Gadgil, Navin Goyal, Ashvni Narayanan, Anand Tadipatri, Towards a Mathematics Formalisation Assistant using Large Language Models
- Zhanghir Azerbayev, Bartosz Piotrowski, and Jeremy Avigad, ProofNet: A Benchmark for Autoformalizing and Formally Proving Undergraduate-Level Mathematics Problems

Lean and mathlib today

A philosophy:

- If you want to do something mathematical, do it in a system with a precise mathematical semantics.
- Complete verification isn't always the most important thing.
- Having a mathematical specification language with a precise semantics is really useful.

Alexander Bentkamp, Ramon Ferández Mir, and I are working on a system for doing convex optimization in Lean 4.

Tomáš Skřivan is working on using Lean 4 as a basis for scientific computation.

Cayden Codel, Marijn Heule, and I are working on a library of verified SAT encodings in Lean.

Why Lean?

- Lean is a well designed system, with nice syntax, a nice user interface, etc.
- Modern mathematics is very algebraic. Simple type theory won't do.
- Lean was designed with classical mathematics in mind (as well as software verification).
- The documentation was written with classical mathematicians in mind.
- Mario Carneiro has boundless energy and was a constant presence on Zulip early on (and still is).
- It attracted a vibrant, enthusiastic community that welcomes newcomers and supports one another.
- The community focused on results of contemporary interest to the broader mathematical community.

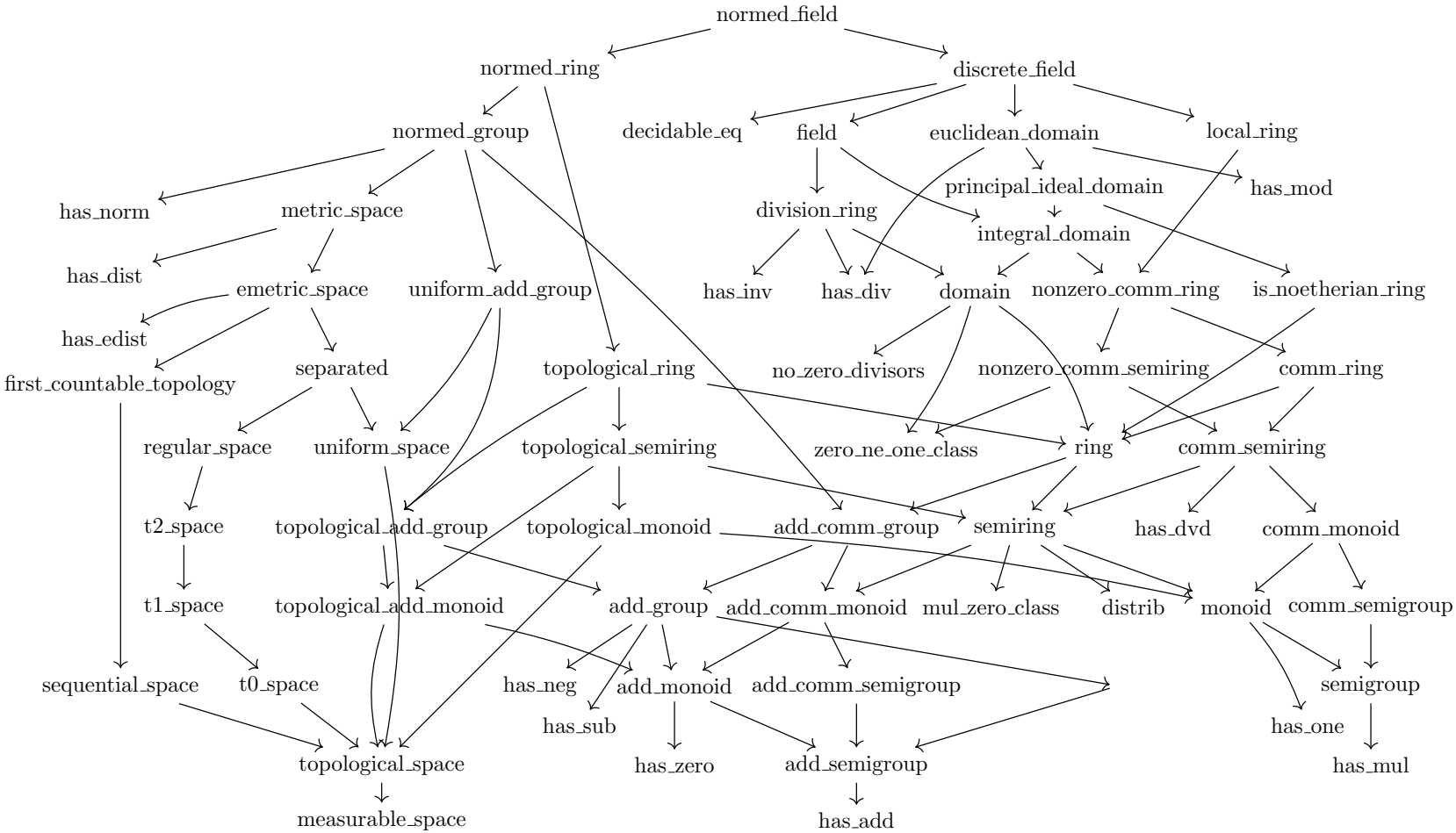
Overview of the library

On the Leanprover community pages:

- statistics
- overview of contents
- tactics
- Zulip
- mathlib port progress

The algebraic hierarchy

In Lean's library *mathlib*, the algebraic hierarchy has hundreds of classes and thousands of instances.



The algebraic hierarchy

Type classes are used for notation, bookkeeping (decidable types, inhabited types, coercions), order structures, linear algebra, topological spaces, category theory, function spaces (inner product spaces, normed spaces), measure theory, manifolds, computability, and more.

There are tons of dependencies between them.

The real numbers are simultaneously an instance of a field, an ordered field, a normed field, a metric space, a topological space, a uniform space, a vector space (over the reals), a manifold, a measure space, ...

The algebraic hierarchy

Think about what is needed to make sense of this:

```
variables (f g : ℝ × ℝ → ℝ)

#check f + g
#check 3 · f
#check continuous f
#check measurable f
```

The algebraic hierarchy

```
@has_add.add.{0} (prod.{0 0} real real → real)
  (@pi.has_add.{0 0} (prod.{0 0} real real) (λ (α :
    prod.{0 0} real real), real)
      (λ (i : prod.{0 0} real real), real.has_add))
  f
  g :
  prod.{0 0} real real → real

@has_smul.smul.{0 0} nat (prod.{0 0} real real → real)
  (@function.has_smul.{0 0 0} (prod.{0 0} real real) nat
    real (@add_monoid.has_smul_nat.{0} real
    real.add_monoid))
  (@bit1.{0} nat nat.has_one nat.has_add
    (@has_one.one.{0} nat nat.has_one))
  f :
  prod.{0 0} real real → real
```

The algebraic hierarchy

```
@continuous.{0 0} (prod.{0 0} real real) real
  (@prod.topological_space.{0 0} real real
     (@uniform_space.to_topological_space.{0} real
        (@pseudo_metric_space.to_uniform_space.{0} real
    real.pseudo_metric_space))
     (@uniform_space.to_topological_space.{0} real
        (@pseudo_metric_space.to_uniform_space.{0} real
    real.pseudo_metric_space)))
  (@uniform_space.to_topological_space.{0} real
     (@pseudo_metric_space.to_uniform_space.{0} real
    real.pseudo_metric_space))
  f

@measurable.{0 0} (prod.{0 0} real real) real
  (@prod.measurable_space.{0 0} real real
    real.measurable_space real.measurable_space)
  real.measurable_space
  f
```

The algebraic hierarchy

How about these?

```
variables (f g : metric.sphere (0 : ℝ × ℝ) 3 → ℝ)

#check f + g
#check 3 · f
#check continuous f
#check measurable f
```

The algebraic hierarchy

```
@has_add.add.{0}
(@coe_sort.{1 2} (set.{0} (prod.{0 0} real real)) Type (@set.has_coe_to_sort.{0} (prod.{0 0}
    real real))
   (@metric.sphere.{0} (prod.{0 0} real real)
      (@prod.pseudo_metric_space_max.{0 0} real real real.pseudo_metric_space
    real.pseudo_metric_space)
      (@has_zero.zero.{0} (prod.{0 0} real real) (@prod.has_zero.{0 0} real real
    real.has_zero real.has_zero))
      (@bit1.{0} real real.has_one real.has_add (@has_one.one.{0} real real.has_one))) →
 real)
(@pi.has_add.{0 0}
   (@coe_sort.{1 2} (set.{0} (prod.{0 0} real real)) Type (@set.has_coe_to_sort.{0} (prod.{0
    0} real real))
      (@metric.sphere.{0} (prod.{0 0} real real)
         (@prod.pseudo_metric_space_max.{0 0} real real real.pseudo_metric_space
    real.pseudo_metric_space)
         (@has_zero.zero.{0} (prod.{0 0} real real) (@prod.has_zero.{0 0} real real
    real.has_zero real.has_zero))
         (@bit1.{0} real real.has_one real.has_add (@has_one.one.{0} real real.has_one))))
   (λ
    (α :
      @coe_sort.{1 2} (set.{0} (prod.{0 0} real real)) Type (@set.has_coe_to_sort.{0}
    (prod.{0 0} real real))
        (@metric.sphere.{0} (prod.{0 0} real real)
           (@prod.pseudo_metric_space_max.{0 0} real real real.pseudo_metric_space
    real.pseudo_metric_space)
           (@has_zero.zero.{0} (prod.{0 0} real real) (@prod.has_zero.{0 0} real real
    real.has_zero real.has_zero))
           (@bit1.{0} real real.has_one real.has_add (@has_one.one.{0} real real.has_one)))),
    real)
   (λ
    (i :
      @coe_sort.{1 2} (set.{0} (prod.{0 0} real real)) Type (@set.has_coe_to_sort.{0}
    (prod.{0 0} real real))
```

Outline

Recap:

- Some history
- Lean and mathlib today
- An overview of the library
- Comparisons between mathlib and Mathematical Components
- Mathematicians' point of view

Let's consider some comparisons between mathlib and Mathematical Components.

Comparisons: the algebraic hierarchy

Lean uses type class inference. Lean 4 introduces a new memoizing search, similar to those used for prolog.

Mathematical Components uses canonical structures.

I don't consider these to be fundamental differences. The two encode essentially the same information.

Getting them to work is a matter of implementation and optimization.

Comparisons: constructivity and computability

Lean distinguishes constructivity from computability.

The library contains:

- Things that are meant to be run efficiently, i.e. are part of the programming language (e.g. for metaprogramming).
- Things that are resolutely classical. (Like division on the real numbers.)
- Things in between: things that are constructive in principle, but not meant to be efficient.

There is little effort to be constructive just for the sake of being constructive.

On the programming side, monads are used to great effect.

Comparisons: automation

In mathlib, definitional reduction is essential for algebraic reasoning (projections of structures), but otherwise it is used very sparingly.

The library is refactored constantly, and there many layers of abstraction.

SSReflect makes more aggressive use of definitional reduction.

In mathlib, we have no concerns about rewriting elements of `Prop`.

Common tactics: `simp` (and `squeeze_simp`), `linarith`, `ring`, `norm_num`, and lots of small-scale custom automation.

Lean 4 promises to be very amenable to automation.

Comparisons: the community

Mathlib is:

- very large.
- developed by a large community.

Isabelle's library and Mathematical Components are also very large, but more tightly curated.

Pull requests are carefully reviewed by reviewers and maintainers.

The community has developed an organizational structure and is trying to scale.

Working on mathlib is like raising a barn.

Mathematicians' point of view

There has been some bad blood between the Lean and Coq communities.

There have been failures of diplomacy.

Mathematicians are deeply appreciative of the foundational research that has given rise to dependent type theory and Lean, as well as all the research that has gone into formalization.

They are, at the same time, very protective of their mathematics.

For example, they bristle at intimations that their mathematics would be better if it were done more constructively.

Mathematicians' point of view

A common complaint is that reviewers at ITP and CPP give them a hard time for not focusing on specifics of the encodings, etc.

To them, these are beside the point: what they value more are the *mathematical* ideas and insights.

Examples:

- Finding the mathematical definitions that make a formalization go smoothly.
- Finding the mathematical *perspectives* that make a formalization go smoothly.
- Figuring out exactly where hypotheses are, and are not, needed.
- Finding mathematical generalizations that unify results and reduce effort.

Mathematicians' point of view

Some insights on how to organize mathematics to make it formalizable:

- Filters are the right way to deal with topological limits. (Hölzl, Immler, and Huffman; Beeson and Wiedijk; Bourbaki)
- The Bochner integral and the Frechet derivative are the right level of generality (following Isabelle).
- Uniform spaces are useful (generalizing both metric spaces and topological groups).
- Embeddings and homomorphic images often work better than subsets and substructures.
- One has to be careful formalizing sheaves and schemes (Buzzard et. al).

Mathematicians' point of view

- If K is a field extension of F, it is often better to treat K as an F-algebra, especially for towers of extensions (Lau, Baanen, Dahmen, Narayanen, Nuccio, Browning, Lutz).
- It's possible to unify real and complex inner product spaces with the right type class (also following Isabelle).
- Semilinear maps are a useful generalization of linear maps and conjugate linear maps (Kudryashov, Macbeth, Lewis, Dupuis).
- Manifolds with corners are better than manifolds with boundary (Gouëzel).
- Isolating schematic principles (i.e. quantifying over predicates) is often useful.

Mathematicians' point of view

- You can get nice formulas for the differentiating a convolution of multivariate functions with a suitable generalization of convolution (van Doorn).
- When dealing with complexes of objects in an abelian category, it's convenient to include maps between $A(i)$ and $A(j)$ for every i and j (LTE).
- To treat the sphere as a manifold, it is convenient to put a chart at every point (Macbeth, Massot).

Let a hundred flowers bloom

Sometimes it feels that interactive theorem provers are in competition.

There are lots of programming languages. That doesn't cause problems: people can make choices based on

- the particular strengths of the language
- libraries and reusable code
- personal preference.

Programming languages come and go. (I grew up with Pascal, Fortran, and Cobol.)

What is important is what we do with them, and what we learn from them.

Let a hundred flowers bloom

Formal methods have a lot to offer mathematics.

Our goal should be to explore them, and to get them in the hands of as many mathematicians as possible.