

Machine learning, formal methods, and mathematics

Jeremy Avigad

Department of Philosophy
Department of Mathematical Sciences
Hoskinson Center for Formal Mathematics

Carnegie Mellon University

AI to Assist Mathematical Reasoning Workshop

June 13, 2023

Table of contents

- symbolic methods and neural methods in AI
- formal methods in mathematics
- formal methods and AI
- mathematics and computer science
- cooperation and collaboration in the new digital environments
- institutional challenges

Symbolic methods and neural methods

Symbolic methods (good old fashioned AI):

- logic-based representations
- precise, exact
- explicit rules of inference

Neural methods (machine learning)

- distributed representations
- probabilistic, approximate
- based on lots of data

Mathematics needs both:

- neural methods can discover patterns and connections
- symbolic methods can help us get the answers right

Symbolic methods and neural methods

Mathematical knowledge:

- The sum of the positive integers up to 100 is 5,050.
- For $n > 2$, there are no integer solutions to $x^n + y^n = z^n$ with all of x , y , and z nonzero.

Empirical knowledge:

- It is likely to rain tomorrow.
- Raising interest rates is likely to lead to a recession.
- Jones is not likely to default a loan.

These are all things we might want to know.

Symbolic methods and neural methods

Mathematics often gives us ways of being precise about imprecise knowledge.

For example, we may extract a model from empirical data and reason about the model.

- The model may be only probably approximately correct.
- But we can reason precisely about the evidence for it and the implications.

We need this type of reasoning to think through the consequences of our actions, deliberate, and plan.

Formal methods in mathematics

Formal methods are a body of logic-based methods used in computer science to

- write specifications for hardware, software, protocols, and so on, and
- verify that artifacts meet their specifications.

The same technology is useful for mathematics.

I use “formal methods in mathematics” and “symbolic AI for mathematics” roughly interchangeably.

Formal methods in mathematics

Since the early twentieth century, we have known that mathematics can be represented in formal axiomatic systems.

Computational “proof assistants” allow us to write mathematical definitions, theorems, and proofs in such a way that they can be

- processed,
- verified,
- shared, and
- searched

by mechanical means.

Formal methods in mathematics

```
lake-packages > mathlib > Mathlib > Analysis > NormedSpace > RieszLemma

39  /-- Riesz's lemma, which usually states that it is
40     possible to find a
41     vector with norm 1 whose distance to a closed
42     proper subspace is
43     arbitrarily close to 1. The statement here is in
44     terms of multiples of
45     norms, since in general the existence of an element
46     of norm exactly 1
47     is not guaranteed. For a variant giving an element
48     with norm in `[1, R]`, see
49     `riesz_lemma_of_norm_lt`. -/
50     theorem riesz_lemma {F : Subspace ℓk E} (hF :
51     IsClosed (F : Set E)) (hF : ∃ x : E, x ∉ F) {r : ℝ}
52     (hr : r < 1) : ∃ x₀ : E, x₀ ∉ F ∧ ∀ y ∈ F, r *
53     ‖x₀‖ ≤ ‖x₀ - y‖ := by
54     classical
55     obtain (x, hx) : ∃ x : E, x ∉ F := hF
56     let d := Metric.infDist x F
57     have hFn : (F : Set E).Nonempty := (⟦_, F.
58     zero_mem)
59     have hdp : 0 < d :=
60     lt_of_le_of_ne Metric.infDist_nonneg fun heq
61     =>
62     | hx ((hF.mem_iff_infDist_zero hFn).2 heq.
63     symm)
64     let r' := max r 2⁻¹
65     have hr' : r' < 1 := by
```

```
Lean Infoview ×
▼ RieszLemma.lean:53:54
▼ Tactic state
1 goal
▼ case intro
k : Type u_1
inst^4 : NormedField ℓk
E : Type u_2
inst^3 : NormedAddCommGroup E
inst^2 : NormedSpace ℓk E
F : Type ?u.309
inst^1 : SeminormedAddCommGroup F
inst : NormedSpace ℝ F
F : Subspace ℓk E
hF : IsClosed ↑F
r : ℝ
hx : ¬x ∈ F
d : ℝ := infDist x ↑F
hFn : Set.Nonempty ↑F
hdhp : 0 < d
├ ∃ x₀, ¬x₀ ∈ F ∧ ∀ (y : E), y
∈ F → r * ‖x₀‖ ≤ ‖x₀ - y‖
► Expected type
► All Messages (0)
```

Ln 53, Col 55 Spaces: 2 UTF-8 LF lean4 Spell

Formal methods in mathematics

Some talks (with links):

- Thomas Hales, [Big Conjectures](#)
- Sébastien Gouëzel, [On a Mathematician's Attempts to Formalize his Own Research in Proof Assistants](#)
- Patrick Massot, [Why Explain Mathematics to Computers?](#)
- Kevin Buzzard, [The Rise of Formalism in Mathematics](#)
- Johan Commelin, [Abstract Formalities](#)
- Adam Topaz, [The Liquid Tensor Experiment](#)
- Heather Macbeth, [Algorithm and Abstraction in Formal Mathematics](#)

Formal methods in mathematics

MATHEMATICS AND THE FORMAL TURN

JEREMY AVIGAD

ABSTRACT. Since the early twentieth century, it has been understood that mathematical definitions and proofs can be represented in formal systems systems with precise grammars and rules of use. Building on such foundations, computational proof assistants now make it possible to encode mathematical knowledge in digital form. This article enumerates some of the ways that these and related technologies can help us do mathematics.

INTRODUCTION

One of the most striking contributions of modern logic is its demonstration that mathematical definitions and proofs can be represented in formal axiomatic systems. Among the earliest were Zermelo's axiomatization of set theory, which was introduced in 1908, and the system of ramified type theory, which was presented by Russell and Whitehead in the first volume of *Principia Mathematica* in 1911. These were so successful that Kurt Gödel began his famous 1931 paper on the incompleteness theorems with the observation that “in them all methods of proof used today in mathematics are formalized, that is, reduced to a few axioms and rules of inference.” Cast in this light, Gödel's results are unnerving: no matter what mathematical methods we subscribe to now or at any point in the future, there will always be mathematical questions, even ones about the integers, that cannot be settled on that basis—unless the methods are in fact inconsistent. But the positive

Formal methods in mathematics

Executive summary: formal methods can be useful for

- verifying theorems
- correcting mistakes
- gaining insight
- building libraries
- searching for definitions and theorems
- refactoring proofs
- refactoring libraries
- engineering concepts
- communicating
- collaborating
- managing complexity
- managing the literature
- teaching
- improving access
- using mathematical computation
- using automated reasoning
- using AI

The technology holds a lot of promise.

Formal methods and AI

Applications of machine learning to mathematics are a new frontier.

There have been important machine-learning projects using Mizar, HOL Light, Metamath, Isabelle, Coq, Lean, and others.

“Draft, sketch, and prove” combines neural and symbolic methods:

- First, a large language model drafts an informal proof.
- Then it sketches a formal proof.
- Automated reasoners fill in the details and verify that they are correct.

Searching for formally checkable content provides a clear signal.

Formal methods and AI

The image shows a browser window displaying an arXiv paper page. The browser's address bar shows the URL `arxiv.org/abs/2210.12283`. The page header includes the Cornell University logo and a message of gratitude from the Simons Foundation. The arXiv logo is prominently displayed in red, with the breadcrumb `cs > arXiv:2210.12283`. A search bar is located in the top right of the header area. The main content area is titled **Computer Science > Artificial Intelligence**. Below this, the submission information is provided: *[Submitted on 21 Oct 2022 (v1), last revised 20 Feb 2023 (this version, v3)]*. The paper title is **Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs**. The authors listed are Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. The abstract begins with the sentence: "The formalization of existing mathematical proofs is a notoriously difficult process. Despite decades of research on automation and proof assistants, writing formal proofs remains arduous and only accessible to a few experts. While previous studies to automate formalization focused on powerful search algorithms, no attempts were made to take advantage of available informal proofs. In this work, we introduce Draft, Sketch, and Prove (DSP), a method that maps informal proofs to formal proof sketches, and uses the sketches to guide an automated prover by directing its search to easier sub-problems. We investigate two relevant setups where informal proofs are either written by humans or generated by a language model. Our experiments and ablation studies show that large language models are able to produce well-structured formal sketches that follow the informal proofs. Guiding an automated prover with these sketches". On the right side of the page, there is a **Download:** section with links for PDF and Other formats, a Creative Commons BY license icon, and a 'Current browse context:' section showing 'cs.AI' and navigation options like '< prev', 'next >', 'new', 'recent', and '2210'. Below that is a 'Change to browse by:' section with 'cs' and 'cs.LG'. Further down is a 'References & Citations' section with links to NASA ADS, Google Scholar, and Semantic Scholar, followed by an 'Export BibTeX Citation' link. At the bottom right, there is a 'Bookmark' section with icons for various services.

Cornell University

We gratefully acknowledge support from the Simons Foundation and member institutions.

arXiv > cs > arXiv:2210.12283

Search... All fields Search

Help | Advanced Search

Computer Science > Artificial Intelligence

[Submitted on 21 Oct 2022 (v1), last revised 20 Feb 2023 (this version, v3)]

Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, Guillaume Lample

The formalization of existing mathematical proofs is a notoriously difficult process. Despite decades of research on automation and proof assistants, writing formal proofs remains arduous and only accessible to a few experts. While previous studies to automate formalization focused on powerful search algorithms, no attempts were made to take advantage of available informal proofs. In this work, we introduce Draft, Sketch, and Prove (DSP), a method that maps informal proofs to formal proof sketches, and uses the sketches to guide an automated prover by directing its search to easier sub-problems. We investigate two relevant setups where informal proofs are either written by humans or generated by a language model. Our experiments and ablation studies show that large language models are able to produce well-structured formal sketches that follow the informal proofs. Guiding an automated prover with these sketches

Download:

- PDF
- Other formats

BY

Current browse context:
cs.AI
< prev | next >
new | recent | 2210

Change to browse by:
cs
cs.LG

References & Citations

- NASA ADS
- Google Scholar
- Semantic Scholar

Export BibTeX Citation

Bookmark

`https://info.arxiv.org/help/index.html`

Mathematics and computer science

Mathematicians and computer scientists have different skill sets and outlooks.

Mathematicians enjoy:

- solving hard problems
- finding patterns
- finding deep connections
- developing powerful abstractions.

Computer scientists enjoy:

- implementing complex systems
- finding clever optimizations
- making systems more reliable and robust.

Mathematics and computer science need each other.

Cooperation and collaboration

Digital technology provides new platforms for cooperation and collaboration.

Communities of practitioners use social media to:

- ask questions and get help
- pose challenges to one another
- make plans and coordinate efforts.

The Liquid Tensor Experiment is a good model:

- The formalization was kept in a shared online repository.
- Participants followed an informal blueprint with links to the repository.
- Participants were in constant contact on Zulip.
- A proof assistant made sure the pieces fit together.

Cooperation and collaboration

Blueprint for the Liquid Tensor Experiment

Introduction

1 First part

1.1 Breen–Deligne data

1.2 Variants of normed groups

1.3 Spaces of convergent power series

1.4 Some normed homological algebra

1.5 Completions of locally constant functions

1.6 Polyhedral lattices

1.7 Key technical result

2 Second part

3 Bibliography

Section 1 graph

Section 2 graph

1.2 Variants of normed groups

Normed groups are well-studied objects. In this text it will be helpful to work with the more general notion of *semi-normed group*. This drops the separation axiom $\|x\| = 0 \iff x = 0$ but is otherwise the same as a normed group.

The main difference is that this includes “uglier” objects, but creates a “nicer” category: semi-normed groups need not be Hausdorff, but quotients by arbitrary (possibly non-closed) subgroups are naturally semi-normed groups.

Nevertheless, there is the occasional use for the more restrictive notion of normed group, when we come to polyhedral lattices below (see Section 1.6).

In this text, a morphism of (semi)-normed groups will always be bounded. If the morphism is supposed to be norm-nonincreasing, this will be mentioned explicitly.

Definition 1.2.1 ✓

Let $r > 0$ be a real number. An r -normed $\mathbb{Z}[T^{\pm 1}]$ -module is a semi-normed group V endowed with an automorphism $T: V \rightarrow V$ such that for all $v \in V$ we have $\|T(v)\| = r\|v\|$.

The remainder of this subsection sets up some algebraic variants of semi-normed groups.

Definition 1.2.2 ✓

A *pseudo-normed group* is an abelian group $(M, +)$, together with an increasing filtration $M_c \subseteq M$ of subsets M_c indexed by $\mathbb{R}_{\geq 0}$, such that each M_c contains 0, is closed under negation, and $M_{c_1} + M_{c_2} \subseteq M_{c_1+c_2}$. An example would be $M = \mathbb{R}$ or $M = \mathbb{Q}_p$ with $M_c := \{x : |x| \leq c\}$.

A pseudo-normed group M is *exhaustive* if $\bigcup_c M_c = M$.

All pseudo-normed groups that we consider will have a topology on the filtration sets M_c . The most general variant is the following notion.

Definition 1.2.3 ✓

A pseudo-normed group M is *CH-filtered* if each of the sets M_c is endowed with a topological space structure making it a compact Hausdorff space, such that following maps are all continuous:

- the inclusion $M_{c_1} \rightarrow M_{c_2}$ (for $c_1 \leq c_2$);
- the negation $M_c \rightarrow M_c$.

Cooperation and collaboration

The port of Mathlib, a large formal mathematical library, is another example.

Since 2016, the community has been using Lean 3. We are now just beginning to use Lean 4, which is not backward compatible.

The Lean 3 library has over a million lines of formal proof.

Cooperation and collaboration

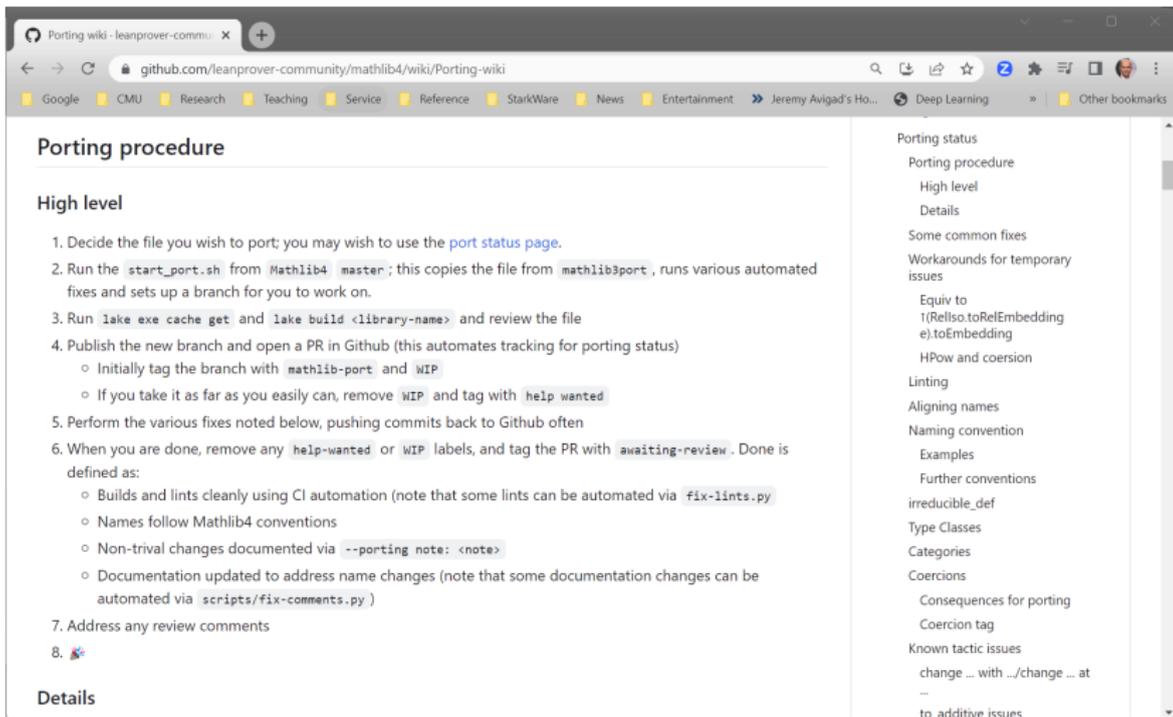
How do you port a million lines of formal proof?

- Mario Carneiro wrote an automatic translator that comes close, but needs user intervention.
- About 40,000 lines of tactics — small scale automation — had to be rewritten entirely.
- Carneiro and Scott Morrison are leading a team of volunteers.
- There were months of planning and discussion.
- The effort requires repairing translations manually and adapting to changes in automation.
- Contributions to the old library have continued.
- There has been an endless stream of problems to address.

Cooperation and collaboration



Cooperation and collaboration



The image shows a web browser window with the address bar displaying `github.com/leanprover-community/mathlib4/wiki/Porting-wiki`. The page title is "Porting procedure". The main content is under the heading "High level" and contains an 8-step list of instructions for porting a library. The steps include deciding the file to port, running `start_port.sh`, building the library, publishing a PR, tagging the branch, performing fixes, removing labels, and addressing review comments. A sidebar on the right lists various topics related to porting, such as "Porting status", "Workarounds for temporary issues", "Naming convention", and "Coercions".

Porting procedure

High level

1. Decide the file you wish to port; you may wish to use the [port status page](#).
2. Run the `start_port.sh` from `Mathlib4 master`; this copies the file from `mathlib3port`, runs various automated fixes and sets up a branch for you to work on.
3. Run `lake exe cache get` and `lake build <library-name>` and review the file
4. Publish the new branch and open a PR in Github (this automates tracking for porting status)
 - Initially tag the branch with `mathlib-port` and `WIP`
 - If you take it as far as you easily can, remove `WIP` and tag with `help wanted`
5. Perform the various fixes noted below, pushing commits back to Github often
6. When you are done, remove any `help-wanted` or `WIP` labels, and tag the PR with `awaiting-review`. Done is defined as:
 - Builds and lints cleanly using CI automation (note that some lints can be automated via `fix-lints.py`)
 - Names follow Mathlib4 conventions
 - Non-trivial changes documented via `--porting note: <note>`
 - Documentation updated to address name changes (note that some documentation changes can be automated via `scripts/fix-comments.py`)
7. Address any review comments
8. 🚀

Details

- Porting status
- Porting procedure
 - High level
 - Details
- Some common fixes
- Workarounds for temporary issues
 - Equiv to `!(RelIso.toRelEmbedding e).toEmbedding`
 - HPow and coercion
- Linting
- Aligning names
- Naming convention
 - Examples
 - Further conventions
- irreducible_def
- Type Classes
- Categories
- Coercions
 - Consequences for porting
 - Coercion tag
- Known tactic issues
 - change ... with .../change ... at ...
 - to additive issues

Cooperation and collaboration

Mathlib porting status [alternate view](#) [out of sync](#)

2453 files ported (79.4%)
836231 lines ported (80.3%)

In progress files

See also [the open mathlib-port PRs](#) on GitHub.

Show entries

File	PR	comments	labels	dependents	lines
measure_theory.integral.bochner	#4590		help-wanted mathlib-port	134	1836
field_theory.splitting_field	#4339		help-wanted mathlib-port	92	620
analysis.special_functions.log_deriv	#4669		help-wanted mathlib-port	63	316
category_theory.bicategory.coherence_tactic	#4610		help-wanted mathlib-port meta	43	244
measure_theory.measure.lebesgue.eq_haar	#4666		awaiting-author mathlib-port	36	867
analysis.box_integral.partition.measure	#4611		help-wanted mathlib-port	30	127

Cooperation and collaboration

The screenshot shows a Zulip chat window titled "All messages - Lean - Zulip". The browser address bar is "leanprover.zulipchat.com/#all_messages". The chat interface has a sidebar on the left with navigation options like "All messages", "Recent conversations", "Mentions", "Starred messages", "Drafts", "DIRECT MESSAGES", and "STREAMS". The "STREAMS" section lists various channels including "lean4", "Machine Learning for Theorem ...", "mathlib maintainers", "mathlib reviewers", "mathlib4", "maths", "metaprogramming / tactics", "MSRI23 preparation", "new members", "PR reviews", "Program verification", "SLMath23", and "Type theory".

The main chat area shows a message from the "mathlib4 > port progress" stream, dated "TODAY". The message contains two tables showing porting progress for "probability.kernel.conexp" and "all".

mathlib port progress	probability.kernel.conexp	
Unported files:	16/1044	(98.5% of total)
Unported lines:	9270/473316	(98.0% of total)
Longest unported chain:	5/108	(95.4% progress)

mathlib port progress	all	
Unported files:	366/3025	(87.9% of total)
Unported lines:	132899/1038756	(87.2% of total)
Longest unported chain:	14/117	(88.0% progress)

676/3026 files **completely ported** (22.33%), in the sense that also all downstream dependencies are ported. 1:34 PM

Below this is another message from the "mathlib4 > group cohomology mathlib3 PR" stream, dated "TODAY". It is from Kevin Buzzard and says: "Group cohomology is three files away from being ported, but after some discussion with Amelia on Thurs she convinced herself that there should be a mathlib3 refactor and more".

At the bottom of the chat area, there are three buttons: "Message #mathlib4 > group cohomology mathlib3 PR", "New topic", and "New direct message".

Institutional challenges

Main challenges:

- Industrial research has to answer to corporate interests.
- Academic environments encourage specialization.

Neither is aligned with developing technology for mathematical research.

Academia is governed by traditional means of assessment:

- Mathematicians are evaluated by the judgments of experts and publication in top journals.
- Computer scientists are judged by citation counts, which are a proxy for impact.

Institutional challenges

There's a chicken and egg problem:

Computer scientists *can* get credit for developing useful technology for mathematics by showing that mathematicians are using it.

Mathematician's can't get credit for using technology unless they use it to make progress on traditional problems.

Both communities need to make substantial investments of time and energy before either will have anything to show for it.

Institutional challenges

Some of the contributors to the mathlib port:

Yury Kudryashov

Gabriel Ebner

Jason Yuen

Jeremy Tan Jie Rui

Moritz Doll

David Renshaw

Johan Commelin

Jon Eugster

Riccardo Brasca

Adam Topaz

Yakov Pechersky

Jakob von Raumer

Henrik Böving

Maxwell Thum

Richard Osborn

Scott Morrison

Mario Carneiro

Joël Riou

Eric Wieser

Jireh Loreaux

Arien Malec

Lukas Miaskiowski

Kevin Buzzard

Kyle Miller

Jujian Zhang

Alex Best

Reid Barton

Siddhartha Gadgil

Zachary Battleman

Ruben Van de Velde

Chris Hughes

Moritz Firsching

Matthew Ballard

Floris van Doorn

Yaël Dillies

Xavier Roblot

Heather Macbeth

Arthur Paulino

Frédéric Dupuis

Violeta Hernández

Anatole Dedecker

Winston Yin

Eric Rodriguez

Institutional challenges

We need to understand how to incentivize and reward:

- ongoing system development and maintenance
- ongoing library development and maintenance
- development and maintenance of web pages and collaboration platforms
- answering questions and training new users.

Table of contents

- symbolic methods and neural methods in AI
- formal methods in mathematics
- formal methods and AI
- mathematics and computer science
- cooperation and collaboration in the new digital environments
- institutional challenges

Summary

Progress on AI for mathematics requires input from three distinct communities:

- computer scientists working in formal methods (proof assistants, automated reasoning)
- computer scientists working in machine learning (large language models, reinforcement learning, and so on)
- mathematicians figuring out how to use the technology to do mathematics

These communities are for the most part disjoint and have disjoint expertise.

Summary

Progress in AI-assisted mathematics is going to require working together:

- Symbolic methods are good at computation, verification, and search, but struggles with combinatorial explosion and heterogeneous reasoning.
- Neural methods can gather, process, and synthesize huge amounts of data, but struggle to get the details right.
- Mathematicians understand the mathematics, and computer scientists won't get anywhere without that.

Summary

- Progress in AI for mathematics needs a combination of neural and symbolic methods.
- It also requires mathematicians and computer scientists working together.
- Advances in technology for mathematics require new forms of collaboration and interaction, and, at the same time, provide new means and platforms for that.
- We need better institutional support for collaborative, cross-disciplinary work and we need ways of assessing new kinds of mathematical contributions.