S4A6 — Advanced Topics in Formalised Mathematics

Seminar: summer semester 2026: Tuesday 14 (c.t.)-16, room N.008.
Goals of the course

- Understand Lean's more subtle aspects better
- Appreciate the considerations of formalising mathematics at scale
- Be prepared to contribute to mathlib yourself
- Learn about good design of a formalisation library
- Explore formalisation in your favourite area of mathlib/your favourite tactic

No prerequisites beyond e.g. last semester's course "Topics in Computer-assisted Mathematics" are assumed

Organizational remarks

- You are welcome to attend the seminar without giving a talk.
- To get credit for this course, you need to give a seminar talk on a topic (and submit a short write-up containing what you said). Talks typically take 90 minutes (including time for questions). If you cover everything in e.g. 70 minutes, that is also fine.
- There is some flexibility in the schedule: you can also give two shorter talks instead of one larger talks; or split two topics in groups of two.
- There will be no seminar on the following dates: May 12, May 26 (holiday), June 23, June 30.

Presentation topics

Theoretical topics: for each of these topics, I have suggested an outline and suitable references below. Topics are ordered by date.

- [**Fridolin**] equality, transparency and all that
- [**Max**] unification
- [**Jasper/Alex**] the typeclass system (two talks)
- [**Vivek**] simp, simp sets and simp normal forms

Practical topics This list of topics is more flexible.

- **[Ruth]** the mathlib contribution process (see below for an outline)
- **[Arend Mellendijk]** qq, metaprogramming and ring
- "my favourite tactic": choose an interesting mathlib tactic and describe how it works, how to use it, its options and variants and (depending on time) how it is implemented. There can be several talks, depending on interest: suitable tactics could be `gcongr`, `order`, `compute_asymptotic`, `to_additive`, `field_simp`, `aesop` [LF23] and probably more. This list is intentionally open-ended. So far, we have:
  Pascal speaking about `gcongr`, **Hannah** speaking about `grind` and **Yannik** speaking about `aesop`.

Formalisation topics Explain how a particular piece of mathematics is formalised in mathlib. Usually, these are documented in a paper describing the formalisation. Present the paper; pay special attention to any formalisation decisions, trade-offs and lessons learned.

- **[Felix]** *iterated integrals and the Sobolev inequality paper* [DM24]
- *A Formalization of the Change of Variables Formula for Integrals in mathlib* [Gou22]
- *Formalizing the Divergence Theorem and the Cauchy Integral Formula in Lean* [Kud22]
- *Formalizing the Gromov-Hausdorff space* [Gou21]
- *Higher order differential calculus in mathlib* [Gou25]
- differential geometry[1]
- *Measure theory and the Haar measure* [van21].[2]
- **[Shuhan]** *A Formalization of Doob's Martingale Convergence Theorems in mathlib* [YD23].
  Don't assume your audience known what a martingale is, so please briefly review them. The above comments about measure theory also apply.
- **[Abel]** *Formalization of derived categories in Lean/mathlib* [Rio25]
- **[Pablo]** *A formal proof of the independence of the continuum hypothesis* [Hv20]
- You are welcome to suggest your favourite mathematics formalisation paper. If so, write me an email or talk to me.

Details about theoretical topics

Here are some details and suggested outlines for the theoretical topics above.

[1] There is no published reference yet; I can provide one closer to the talk. There is an old article using Lean 3 [BC21]; there are also past and recent talks.

[2] The formalisation of measure theory is discussed here and in [YD23]. Both articles point to an old version of mathlib using Lean 3; in your talk, make sure to point to the current mathlib equivalent. Also note that the definition of integral may have been generalised in the mean-time. Reading the module doc-strings of the relevant mathlib files is highly recommended.

equality, transparency and all that

background: three kinds of equality (This blog post provides an information overview, the Lean language references makes definitional equality more precise)

- syntactic equality (up to notation and alpha-equivalence = renaming bound variables)
- definitional equality (review class/language reference)
- propositional equality (mathematically equal)
- special case: dependent types; syntactically equal arguments yield defeq types (merely defeq ones do not!). give an example
- important fun fact: while definitional equality is reflexive and symmetric, it is not transitive

which tactics, procedures need which equality? give some common examples!

- for example, `rw` requires syntactic inequality
- describe at least `apply`, `exact`, `simp` (and perhaps others)
- typeclass inferences and unification use definitional equality (at possibly different transparency)
- transparency levels: how many definitions are unfolded (Language reference)
- `irreducible_def` in mathlib
- optional: emulating irreducibility in the new module system
- (optional) current example: marking definitions irreducible can help performance
  why does this help? where would we unfold it, what would be unfolded

Unification

Recall: when elaborating a definition or applying a lemma, Lean fills in any arguments in square brackets `[]` by typeclass interence, and arguments in { curly brackets } by unification.

- theoretical background: what is unification, how to do it and its properties (a reference will be provided before the initial meeting)
  - first-order unification; has algorithms
  - higher-order unification is undecidable in general;
    have to use heuristics (possibly involving transparency)
  - proving undecidability of higher-order unification is probably not feasible,
    presenting the main idea of proof might be
- practical side; unification in Lean
  - Lean 2 used higher-order unification; [de +15] describes the system at that point
  - Lean 3 and 4 only use first-order unification (with additional complexity of unfolding definitions); quite complex algorithm; various heuristics on which definitions to unfold (influences complexity a lot)

  - Lean 3 (and perhaps Lean 4) had issues with heuristics being non-ideal; led to performance problems
  - the `induction` tactic and the `elab_as_elim` attribute perform *higher-order pattern matching*: another unification strategy, simpler than higher-order unification and decidable

- rule of thumb: use the typeclass system when there is a unique choice

Typeclass inference

Depending on interest, this topic can fill one or two talks. Let me describe both talks separately; if there is only one talk, feel free to select form there to form one reasonable talk.

First talk: "what is typeclass inference and how to perform it"

- intuition: database of function properties
- works up to defeq, but at a different transparency level
- what is forgetful inheritance? (mathlib library note; uses topological and metric spaces are example)
- what is priority; how do we use it? see library note

The Lean language reference contains a good summary; the *Tabled Typeclass Resolution* paper [SUd20] describes this in more detail.

Second talk: "how to use typeclass inference in mathlib" Useful references are *Use and Abuse of Instance Parameters in the Lean Mathematical Library* [Baa22] and *Multiple-Inheritance Hazards in Dependently-Typed Algebraic Hierarchies* [Wie23]. Both papers are written about Lean 3, but explicitly discuss changes with Lean 4. You can focus only on the Lean 4 situation.
Some additional items you could mention (see [Baa+26, §4]):

- the `FunLike` hierarchy refactor
- the performance/readability trade-off is using bundled or unbundled typeclasses
- optional: currently, unbundling typeclasses also comes with a readability and ergonomics loss. Mention how a new implementation of `class abbrev` could resolve this.

simp, simp sets and simp normal forms

Are there useful abstract properties the set of `simp` lemmas could have? Let's review some of the theory. The notes of last year's *Logic of Proof assistants* course can be a useful reference.

- review: rewriting relations; weakly normalising, strongly normalising, confluence
- implications between those (§3.1 in the notes)
- on untyped lambda-calculus, one-step beta reduction is confluent (Church-Rosser theorem); beta-reduction is neither weakly nor strongly normalising (because of not well-typed terms)

- for well-typed terms, one-step beta reduction is strongly normalising (§3.2)
- in a pure type system, beta-reduction is strongly normalising and confluent
- Rocq has a pure type system; Lean does not! Definitional equality is not transitive (so confluence fails); strong normalisation doesn't hold either.
- while everything can fail in contrived examples, it works well in practice
- Mario Carneiro's master's thesis [Car19] has examples for most of these. (It is written for Lean 3, but is largely still up to date.) There is a zulip thread for the failure of normalisation (this relies proof-irrelevant impredicative prop; if you want to present this, please explain impredicativity).

`dsimp` lemmas

- the `dsimp` is the set of all `simp` lemmas that are proven by `rfl`
- in particular, `dsimp` could have worse properties than definitional equality, if you create bad simp lemmas!
- in practice, a non-terminal `dsimp` is mostly fine, while a non-terminal `simp` is not

basics of how simp works: see the Lean language reference

- traverses the entire expression down to all leaves; traverses bottom-up from all leaves for most simp lemmas
- in a simp lemma, the left hand side must be in simp normal form — including all subexpressions, because these are simplified first
- simp lemmas can have priorities, but subexpressions are still simplified first (even by lower-priority lemmas)
- apply iff lemmas from left to right
- optional: describe how a lemma matches (discrimination trees)

considerations for designing a simp set

- simp lemmas should not loop
  also bad: `1 = 1 + 0` will not repeat the expression, but never terminate
- confluence is a nice goal, exists tooling to check — but too much to hope for in general
- avoid redundant simp lemmas (i.e. provable by simp; would never fire; wasted work)
- "is it normalising?" left hand side should be in simp normal form
  find the normal form for an expression `e` using `#simp e`
  (with variants such as `#simp only [-Nat.factorial] Nat.factorial 5`)
- mathlib: simpNF linter checks for these
- priorities for simp lemmas and why; mathlib library note (optional)
- practical example (zulip discussion): conflicting normal forms (0 vs `bot`) for ideals (optional)

The mathlib contribution process An outline for this talk will be added soon.

References

[Baa+26] Anne Baanen et al. "Growing Mathlib: maintenance of a large scale mathematical library". In: *Intelligent Computer Mathematics*. Ed. by Valeria de Paiva and Peter Koepke. Cham: Springer Nature Switzerland, 2026, pp. 51–70.

[Baa22] Anne Baanen. "Use and Abuse of Instance Parameters in the Lean Mathematical Library". In: *13th International Conference on Interactive Theorem Proving (ITP 2022)*. Ed. by June Andronick and Leonardo de Moura. Vol. 237. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 4:1–4:20. URL: https://drops.dagstuhl.de/opus/volltexte/2022/16713.

[BC21] Anthony Bordg and Nicolò Cavalleri. "Elements of Differential Geometry in Lean: A Report for Mathematicians". In: *CoRR* abs/2108.00484 (2021). arXiv: 2108.00484. URL: https://arxiv.org/abs/2108.00484.

[Car19] Mario Carneiro. *The Type Theory of Lean*. Master thesis. 2019. eprint: https://github.com/digama0/lean-type-theory/releases.

[de +15] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. "The Lean Theorem Prover (System Description)". In: *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*. 2015, pp. 378–388. URL: %5Curl%7Bhttps://doi.org/10.1007/978-3-319-21401-6_26%7D.

[DM24] Floris van Doorn and Heather Macbeth. "Integrals Within Integrals: A Formalization of the Gagliardo-Nirenberg-Sobolev Inequality". In: *15th International Conference on Interactive Theorem Proving (ITP 2024)*. Ed. by Yves Bertot, Temur Kutsia, and Michael Norrish. Vol. 309. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 37:1–37:18. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2024.37.

[Gou21] Sébastien Gouëzel. "Formalizing the Gromov-Hausdorff space". In: *CoRR* abs/2108.13660 (2021). arXiv: 2108.13660. URL: https://arxiv.org/abs/2108.13660.

[Gou22] Sébastien Gouëzel. "A Formalization of the Change of Variables Formula for Integrals in mathlib". In: *Intelligent Computer Mathematics - 15th International Conference, CICM 2022, Tbilisi, Georgia, September 19-23, 2022, Proceedings*. Ed. by Kevin Buzzard and Temur Kutsia. Vol. 13467. Lecture Notes in Computer Science. Springer, 2022, pp. 3–18. URL: https://doi.org/10.1007/978-3-031-16681-5%5C_1.

[Gou25] Sébastien Gouëzel. "Higher order differential calculus in mathlib". arXiv:2509.04922 [cs.LO]. Sept. 25, 2025.

[Hv20] Jesse Michael Han and Floris van Doorn. "A formal proof of the independence of the continuum hypothesis". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*. 2020, pp. 353–366. URL: https://doi.org/10.1145/3372885.3373826.

[Kud22] Yury Kudryashov. "Formalizing the Divergence Theorem and the Cauchy Integral Formula in Lean". In: *13th International Conference on Interactive Theorem Proving (ITP 2022)*. Ed. by June Andronick and Leonardo de Moura. Vol. 237. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 23:1–23:19. URL: https://drops.dagstuhl.de/opus/volltexte/2022/16732.

[LF23] Jannis Limperg and Asta Halkjær From. "Aesop: White-Box Best-First Proof Search for Lean". In: *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*. Ed. by Robbert Krebbers, Dmitriy Traytel, Brigitte Pientka, and Steve Zdancewic. ACM, 2023, pp. 253–266. URL: https://doi.org/10.1145/3573105.3575671.

[Rio25] Joël Riou. "Formalization of derived categories in Lean/mathlib". English. In: *Annals of Formalized Mathematics* Volume 1, 1 (July 2025). URL: https://afm.episciences.org/13609.

[SUd20] Daniel Selsam, Sebastian Ullrich, and Leonardo de Moura. *Tabled Typeclass Resolution*. 2020. arXiv: 2001.04301. URL: https://arxiv.org/abs/2001.04301.

[van21] Floris van Doorn. "Formalized Haar Measure". In: *12th International Conference on Interactive Theorem Proving (ITP 2021)*. Ed. by Liron Cohen and Cezary Kaliszyk. Vol. 193. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 18:1–18:17. URL: https://drops.dagstuhl.de/opus/volltexte/2021/13913.

[Wie23] Eric Wieser. "Multiple-Inheritance Hazards in Dependently-Typed Algebraic Hierarchies". In: *Intelligent Computer Mathematics*. Ed. by Catherine Dubois and Manfred Kerber. Cham: Springer Nature Switzerland, Aug. 2023, pp. 222–236. URL: https://doi.org/10.1007/978-3-031-42753-4_15.

[YD23] Kexing Ying and Rémy Degenne. "A Formalization of Doob's Martingale Convergence Theorems in mathlib". In: *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*. Ed. by Robbert Krebbers, Dmitriy Traytel, Brigitte Pientka, and Steve Zdancewic. ACM, 2023, pp. 334–347. URL: https://doi.org/10.1145/3573105.3575675.