University of Alberta
Fall 2023, Math 681
Formalization of Mathematics

Adam Topaz

July 4, 2023

What is this course about?

As the title suggests, this course is about the *formalization of mathematics*, but the title *"digitization of mathematics"* would have been just as appropriate. More precisely, the goal of this course is to teach students how to explain mathematics (definitions, theorems, proofs, etc.) to a computer in such a way so that the consistency of definitions and correctness proofs can be *verified* by the machine. There are a number of reasons that would motivate a mathematician to digitize a piece of mathematics, and here are just a few:

- The formal verification of the correctness of a complicated proof.
- It helps us come up with better abstractions and definitions.
- When asynchronously collaborating with others, it helps in ensuring consistency at all times.
- It allows us to build open-source libraries of formally verified mathematics.
- In the (near) future, it will help incorporate tools from machine learning and AI in the mathematician's workflow.

Many of these reasons, among others, are summarized nicely in a note written by P. Massot, titled *Why formalize mathematics?*. This course will act as an introduction to the formalization/digitization of mathematics, using the Lean4 interactive proof assistant. Students will learn how to write mathematical definitions, theorems, and proofs, in a way that can be understood and checked by the computer. At the same time, this course will provide a tour of portions of Lean4's formally verified mathematics library `mathlib4`.

Why now?

Formal verification has existed for a while now, but in the past it mostly focused on the formal verification of critical software and hardware, or the formal verification of simple mathematical statements. Some notable exceptions exist, among them is the *Flyspeck Project*, which formally verified Tom Hales' proof of the Kepler conjecture; see this paper for more details. Another is the formalization of the odd-order theorem from finite group theory.

In recent years, the subject has seen a surge of activity. In my opinion, there are two main reasons for this. First, it has become clear that fashionable research-level mathematics can be formalized in reasonably short amounts of time. Some notable recent examples include:

1. The formalization of *perfectoid spaces*, which are complicated objects appearing in modern arithmetic geometry.
2. The sphere eversion project, which formally verified the existence of a sphere eversion via Gromov's h-principle.
3. The liquid tensor experiment, which formally verified the *main theorem of liquid vector spaces* due to Clausen-Scholze. This project arose in response to a challenge posed by fields medalist P. Scholze in December 2020.

Second, it has become clear to many that *digitization* will play an increasingly important role in the usual practice of mathematics, particularly with recent advances in AI. Just in the last 12 months, there have been three high profile events on the use of computers in mathematics where formalization has played a prominent role:

1. The IPAM workshop on *machine assisted proofs*.
2. The *fields medal symposium* in honour of fields medalist A. Venkatesh.
3. A workshop on interactions between AI and mathematics organized by the national academies of sciences, engineering and mathematics.

What is certainly clear to anyone working in this area, myself included, is that the digitization of mathematics change how mathematicians work in the future. Right now we have an opportunity to help shape what such changes will look like.

More about the course

This Math 681 course is also a PIMS network-wide graduate course, meaning that anyone from the PIMS network can register to receive credit. This is an *online course*, and lectures will take place on Tuesday and Thursdays at 11am to 12:20pm (Edmonton time), using zoom (or something similar). A complete syllabus can be found here.

This course has no strict mathematical prerequisites. While I don't expect students to be absolute experts in the mathematical areas listed on the syllabus, some familiarity is expected. The most important (and only actual) prerequisite for this course is a sufficient amount of mathematical maturity. Students should be able to pick up any unfamiliar concepts as the term progresses.

If you might be interested in taking this course but still have some reservations, please don't hesitate to contact me; my email address is `topaz@ualberta.ca`.

Lean4

As I mentioned above, we will use the Lean4 interactive proof assistant for this course. This is an open-source language, developed primarily by L. de Moura and S. Ullrich, and is backed by a large community and mathematics library.

Below are some code snippets which illustrate what this language looks like for the purposes of formalized mathematics, copied directly from `mathlib4`. I should stress that this code is meant to be read with the help of an *editor*, which activates the *interactive* nature of an interactive proof assistant. By using the editor, the user can see, at *each step of the proof*, the precise mathematical context and goals. Here is a short video about proving the infinitude of primes using Lean3, highlighting the *interactive* nature of the system. So, while the code below may look intimidating at first, the process of *writing* this code is quite similar to the usual practice of mathematics, albeit with a additional help from the computer.

Example. Any normal Sylow p-subgroup is characteristic:

```
theorem characteristic_of_normal {p : ℕ} [Fact p.Prime] [Finite (Sylow p G)]
    (P : Sylow p G) (h : (P : Subgroup G).Normal) :
    (P : Subgroup G).Characteristic := by
  haveI := Sylow.subsingleton_of_normal P h
  rw [characteristic_iff_map_eq]
  intro Φ
  show (Φ • P).toSubgroup = P.toSubgroup
  congr
  simp
```

Example. Any category which has all (small) products and all equalizers has all (small) limits.

```
theorem has_limits_of_hasEqualizers_and_products [HasProducts.{w} C] [HasEqualizers C] :
    HasLimitsOfSize.{w, w} C :=
  { has_limits_of_shape :=
    fun _ _ => { has_limit := fun F => hasLimit_of_equalizer_and_product F } }
```

Example. The law of quadratic reciprocity.

```
theorem quadratic_reciprocity (hp : p ≠ 2) (hq : q ≠ 2) (hpq : p ≠ q) :
    legendreSym q p * legendreSym p q = (-1) ^ (p / 2 * (q / 2)) := by
  have hp₁ := (Prime.eq_two_or_odd <| @Fact.out p.Prime _).resolve_left hp
  have hq₁ := (Prime.eq_two_or_odd <| @Fact.out q.Prime _).resolve_left hq
  have hq₂ : ringChar (ZMod q) ≠ 2 := (ringChar_zmod_n q).substr hq
  have h :=
    quadraticChar_odd_prime ((ringChar_zmod_n p).substr hp) hq ((ringChar_zmod_n
    p).substr hpq)
  rw [card p] at h
  have nc : ∀ n r : ℕ, ((n : ℤ) : ZMod r) = n := fun n r => by norm_cast
  have nc' : (((-1) ^ (p / 2) : ℤ) : ZMod q) = (-1) ^ (p / 2) := by norm_cast
  rw [legendreSym, legendreSym, nc, nc, h, map_mul, mul_rotate', mul_comm (p / 2), ←
    pow_two,
    quadraticChar_sq_one (prime_ne_zero q p hpq.symm), mul_one, pow_mul, χ₄
    _eq_neg_one_pow hp₁, nc',
    map_pow, quadraticChar_neg_one hq₂, card q, χ₄_eq_neg_one_pow hq₁]
```