Formalizing Mathematical Structures

Jeremy Avigad

April 23, 2025

Department of Philosophy
Department of Mathematical Sciences
Director, Hoskinson Center for Formal Mathematics
Carnegie Mellon University

Formalization of mathematics

In the early 20th century, logicians developed formal axiomatic systems for mathematics.

It soon became clear that these systems were expressive enough to formalize most mathematics, in principle.

In the early 1970s, the first proof assistants made it possible to formalize and verify proofs in practice.

Today, the practice is known as *interactive theorem proving*.

Working with a proof assistant, users construct formal definitions and proofs.

In many systems, the proofs can be extracted and verified independently.

Formalization of mathematics

Some systems with substantial mathematical libraries:

- Mizar (set theory)
- HOL (simple type theory)
- Isabelle (simple type theory)
- HOL Light (simple type theory)
- Coq / Rocq (constructive dependent type theory)
- ACL2 (primitive recursive arithmetic)
- PVS (classical dependent type theory)
- Agda (constructive dependent type theory)
- Metamath (set theory)
- Lean (dependent type theory)

An important landmark

In 1998, Thomas Hales announced a proof of the Kepler conjecture.

The result relied on extensive computation.

He found the review process at the *Annals* unsatisfying:

- It was four years before they began their work.
- They cautioned that they could not check the code.

In response, he launched the *Flyspeck* project to verify the results formally.

An important landmark

He announced the completion of the verification in August 2014.

- Most of the proof was verified in HOL Light.
- The classification of tame graphs was verified in Isabelle.
- Verifying several hundred nonlinear inequalities required roughly 5000 processor hours on the Microsoft Azure cloud.

This was the first time formal verification served to certify a major result in mathematics.

The Lean proof assistant

The Lean project was launched by Leonardo de Moura, then at Microsoft Research, in 2013.

His original goal was to combine the best of interactive and automated theorem proving.

I had the good fortune to be there at the beginning.

My students and I worked on the first libraries and documentation.

Formalization of mathematics

In 2017, there were very few mathematicians using proof assistants.

At a *Big Proof* workshop at the Isaac Newton Institute in Cambridge, Tom gave a talk about the verification of the Kepler Conjecture, and he mentioned Lean at the end.

Since then, many mathematicians have embraced Lean.

Mathematicians and formalization

Why have so many mathematicians taken to Lean?

- It's a well-designed system.
- It was written with both mathematicians and computer scientists in mind.
- Documentation was written with both mathematicians and computer scientists in mind.
- Some enthusiastic mathematicians drew attention early on.
- There's an energetic online community.
- Mario Carneiro was there early on.

Mathematicians and formalization

More reasons mathematicians are comfortable with Lean:

- It has good support for classical mathematical reasoning.
- It has good support for reasoning about structures.

Systems like Isabelle and HOL Light meet the first requirement.

Systems like Coq and Agda meet the second requirement.

Both are essential.

Classical reasoning

```
noncomputable section
open Classical

def fact : ℕ → ℕ
  | 0 => 1
  | (n + 1) => (n + 1) * fact n

#eval fact 1000

def f (x : ℝ) : ℝ := if x ≤ 0 then 0 else 1

def g (x : ℝ) : ℝ := if Irrational x then 0 else 1

example : ∀ x, f x ≤ 1 := by
  intro x; simp [f]; split <;> linarith
```

Structural reasoning

Since the early twentieth century, axiomatically characterized structures have been central to mathematics.

Mathematicians now

- take products of structures,
- take powers of structures,
- take limits of structures,
- build quotients of structures,
- and lots more.

In short, they calculate with structures as easily as they calculate with numbers.

Structures have to be first-class objects in a proof assistant.

Structural reasoning

From Sébastian Gouëzel's web page (c. 2018?):

"Out of curiosity, I have given a try to several proof assistants, i.e., computer programs on which one can formalize and check mathematical proofs, from the most basic statements (definition of real numbers, say) to the most advanced ones (hopefully including current research in a near or distant future). The first one I have managed to use efficiently is Isabelle/HOL. In addition to several facts that have been added to the main library (for instance conditional expectations), I have developed the following theories..."

Structural reasoning

"However, I have been stuck somewhat by the limitations of the underlying logic in Isabelle (lack of dependent types, making it hard for instance to define the p-adic numbers as this should be a type depending on an integer parameter p, and essentially impossible to define the Gromov-Hausdorff distance between compact metric spaces without redefining everything on metric spaces from scratch, and avoiding typeclasses). These limitations are also what makes Isabelle/HOL simple enough to provide much better automation than in any other proof assistant, but still I decided to turn to a more recent system, Lean, which is less mature, has less libraries, and less automation, but where the underlying logic (essentially the same as in Coq) is stronger (and, as far as I can see, strong enough to speak in a comfortable way about all mathematical objects I am interested in)."

Structural reasoning

In his 2017 talk at Big Proof, Tom enumerated topics need to handle results in his original research field.

> Formalizing statements in Automorphic Representation Theory (a branch of number theory).
> This will require a great deal of work just to state the theorems that are proved: algebraic geometry (schemes, motives, stacks, moduli spaces, and sheaves), measure theory and functional analysis, algebra (rings, modules, Galois theory, homological algebra, derived categories), category theory, complex analysis (L-functions and modular forms), class field theory (local and global), Lie theory and linear algebraic groups (Cartan classification and structure theory), representation theory (infinite dimensional, spectral theory), Shimura varieties, locally symmetric spaces, Hecke operators, cohomology (singular, deRham, intersection homology, l-adic), rigid geometry, perfectoid spaces,...

Structural reasoning

Kevin Buzzard watched the talk remotely from London. He now likes to talk about "the slide that changed my life."

He, Kenny Lao, and some other students formalized schemes. Nobody was all that impressed.

Soon after he, Johan Commelin, and Patrick Massot decided to formalize the definition of a perfectoid space. They succeeded in 2019.

For many in the mathematics community, this was a sign that Lean was ready for "real mathematics."

Overview

I will talk about Lean's handling of structures.

- *The good:* Lean and Mathlib support an extensive network of structures.
- *The bad:* The complexity poses challenges for library development and maintenance.
- *The ugly:* It also poses challenges for automation.

Structural reasoning is one of the sources of the impressive power of modern mathematical reasoning.

With great power there must also come great responsibility.

Structures

Quiz:

- Who first gave an axiomatic characterization of a group?
- Who first defined a quotient group?
- Who first defined the notion of an ideal in a ring (and proved unique factorization of ideals)?
- Who first gave the modern definition of a Riemann surface?
- Who first gave an axiomatic characterization of a topological space?
- Who first gave an axiomatic characterization of a Hilbert space?
- Who first gave the modern definition on a measure on a space as a σ-additive function on a σ-algebra?
- Who defined the p-adic integers?

Structural language

```
def quadraticChar (α : Type) [MonoidWithZero α] (a : α) : ℤ :=
if a = 0 then 0 else if IsSquare a then 1 else -1

def legendreSym (p : ℕ) (a : ℤ) : ℤ := quadraticChar (ZMod p) a

variable {p q : ℕ} [Fact p.Prime] [Fact q.Prime]

theorem quadratic_reciprocity (hp : p ≠ 2) (hq : q ≠ 2) (hpq : p ≠ q) :
  legendreSym q p * legendreSym p q = (-1) ^ (p / 2 * (q / 2))
```

Structural language

```
def Padic (p : ℕ) [Fact p.Prime] :=
  CauSeq.Completion.Cauchy (padicNorm p)

def PadicInt (p : ℕ) [Fact p.Prime] :=
  {x : ℚ_[p] // ‖x‖ ≤ 1}

variable {p : ℕ} [Fact p.Prime] {F : Polynomial ℤ_[p]} {a : ℤ_[p]}

theorem hensels_lemma :
    (hnorm : ‖Polynomial.eval a F‖ <
      ‖Polynomial.eval a (Polynomial.derivative F)‖ ^ 2)
  ∃ z : ℤ_[p],
    F.eval z = 0 ∧
    ‖z - a‖ < ‖F.derivative.eval a‖ ∧
    ‖F.derivative.eval z‖ = ‖F.derivative.eval a‖ ∧
    ∀ z', F.eval z' = 0 → ‖z' - a‖ < ‖F.derivative.eval a‖ → z' = z
```

Structural language

```
def FreeAbelianGroup : Type :=
  Additive <| Abelianization <| FreeGroup α

def IsPGroup (p : ℕ) (G : Type) [Group G] : Prop :=
  ∀ g : G, ∃ k : ℕ, g ^ p ^ k = 1

theorem IsPGroup.exists_le_sylow {P : Subgroup G} (hP : IsPGroup p P) :
  ∃ Q : Sylow p G, P ≤ Q
```

Structural language

```
variable {R S : Type} (K L : Type) [EuclideanDomain R]
variable [CommRing S] [IsDomain S]
variable [Field K] [Field L]
variable [Algebra R K] [IsFractionRing R K]
variable [Algebra K L] [FiniteDimensional K L] [IsSeparable K L]
variable [algRL : Algebra R L] [IsScalarTower R K L]
variable [Algebra R S] [Algebra S L]
variable [ist : IsScalarTower R S L]
variable [iic : IsIntegralClosure S R L]
variable (abv : AbsoluteValue R ℤ)

/-- The main theorem: the class group of an integral closure `S` of `R`
in a finite extension `L` of `K = Frac(R)` is finite if there is an
admissible absolute value. -/
noncomputable def fintypeOfAdmissibleOfFinite : Fintype (ClassGroup S) :=
  ...
```

Structural language

```
variable {α β ι : Type} {m : MeasurableSpace α}
variable [MetricSpace β] {μ : Measure α}
variable [SemilatticeSup ι] [Nonempty ι] [Countable ι]
variable  {f : ι → α → β} {g : α → β} {s : Set α}

/-- Egorov's theorem: A sequence of almost everywhere convergent functions
    converges uniformly except on an arbitrarily small set. -/
theorem tendstoUniformlyOn_of_ae_tendsto
    (hf : ∀ n, StronglyMeasurable (f n))
    (hg : StronglyMeasurable g)
    (hsm : MeasurableSet s) (hs : μ s ≠ ∞)
    (hfg : ∀ᵐ x ∂μ, x ∈ s → Tendsto (fun n => f n x) atTop (𝓝 (g x)))
    {ε : ℝ} (hε : 0 < ε) :
  ∃ t ⊆ s, MeasurableSet t ∧ μ t ≤ ENNReal.ofReal ε ∧
    TendstoUniformlyOn f g atTop (s \ t) :=
  ...
```

Structures in dependent type theory

```
structure Point where
  x : ℝ
  y : ℝ
  z : ℝ

def myPoint : Point where
  x := 2
  y := -1
  z := 4

def add (a b : Point) : Point where
  x := a.x + b.x
  y := a.y + b.y
  z := a.z + b.z
```

Structures in dependent type theory

```
structure StandardTwoSimplex where
  x : ℝ
  y : ℝ
  z : ℝ
  x_nonneg : 0 ≤ x
  y_nonneg : 0 ≤ y
  z_nonneg : 0 ≤ z
  sum_eq : x + y + z = 1

def midpoint (a b : StandardTwoSimplex) : StandardTwoSimplex where
  x := (a.x + b.x) / 2
  y := (a.y + b.y) / 2
  z := (a.z + b.z) / 2
  x_nonneg := ...
  y_nonneg := ...
  z_nonneg := ...
  sum_eq := ...
```

Structures in dependent type theory

```
structure Group where
  carrier : Type
  mul : carrier → carrier → carrier
  one : carrier
  inv : carrier → carrier
  mul_assoc : ∀ x y z : carrier, mul (mul x y) z = mul x (mul y z)
  mul_one : ∀ x : carrier, mul x one = x
  one_mul : ∀ x : carrier, mul one x = x
  mul_left_inv : ∀ x : carrier, mul (inv x) x = one

variable (G : Group) (g₁ g₂ : G.carrier)
```

Structures in dependent type theory

```
structure Group (α : Type) where
  mul : α → α → α
  one : α
  inv : α → α
  mul_assoc : ∀ x y z : α, mul (mul x y) z = mul x (mul y z)
  mul_one : ∀ x : α, mul x one = x
  one_mul : ∀ x : α, mul one x = x
  mul_left_inv : ∀ x : α, mul (inv x) x = one

variable {G : Type} [Group G] (g₁ g₂ : G)
```

Design specifications

Doing mathematics requires:

- defining algebraic structures and reasoning about them (groups, rings, fields, ...)
- defining instances of structures and recognizing them as such ($\mathbb{R}$ is an ordered field, a metric space, ...)
- overloading notation ($x + y$, "f is continuous")
- inheriting structure: every normed additive group is a metric space, which is a topological space.
- defining functions and operations on structures: we can take products, powers, limits, quotients, and so on.

Design specifications

Structure is inherited in various ways:

- Some structures extend others by adding more axioms (a commutative ring is a ring, a Hausdorff space is a topological space).
- Some structures extend others by adding more data (a module is an abelian group with a scalar multiplication, a normed field is a field with a norm).
- Some structures are defined in terms of others (every metric space is a topological space, there are various topologies on function spaces).

Defining structures and instances

We have seen how to define the group structure `Group` α on a type α.

We can define instances of `Group` α the same way we define instances of `Point` and `StandardTwoSimplex`.

```
def permGroup {α : Type} : Group (Perm α) where
  mul f g := ...
  one := ...
  inv := ...
  mul_assoc f g h := ...
  one_mul := ...
  mul_one := ...
  mul_left_inv := ...
```

Defining structures and instances

We are not there yet. We need:

- *Notation:* given `g1 g2 : Perm α`, we want to write `g1 * g2` and g_1^{-1} for the multiplication and inverse.
- *Definitions:* we want to use defined notions like `g1^n` and `conj g1 g2`.
- *Theorems:* we want to apply theorems about arbitrary groups to the permutation group.

Defining structures and instances

The magic depends on three things:

1. *Logic.* A definition that makes sense in any group takes the type of the group and the group structure as arguments.

   A theorem about the elements of an arbitrary group quantifies over the type of the group and the group structure.

2. *Implicit arguments.* The arguments for the type and the structure are generally left implicit.
3. *Type class inference.*
   - Instance relations are registered with the system.
   - The system uses this information to resolve implicit arguments.

Notation

We overload notation by associating it to trivial structures.

```
class Add (α : Type u) where
  add : α → α → α

#check @Add.add
-- @Add.add : {α : Type u_1} → [self : Add α] → α → α → α

infixl:65 " + "   => Add.add

instance : Add Point where
  add := Point.add
```

Notation

```
variable (p q : Point)

#check p + q
-- p + q : Point

set_option pp.notation false
#check p + q
-- Add.add p q

set_option pp.explicit true
#check p + q
-- @Add.add Point instPointAdd p q
```

Classes and instances

The `class` command is a variant of the structure command that makes the structure a target for *type class inference*.

The `instance` command registers particular instances for type class inference.

We can register concrete instances ($\mathbb{R}$ is a field, the permuations of α form a group), as well as generic instances (every field is a ring, every metric space is a topological space, every normed abelian group is a metric space.)

Defining structures and instances

```
class Group (α : Type) :=
  ...

instance {α : Type} : Group (Perm α) :=
  ...

instance : Ring ℝ :=
  ...

instance {M : Type} [MetricSpace M] : TopologicalSpace M :=
  ...

-- Again, this is a simplification.
```

Defining structures and instances

```
#check @Add.add
-- @Add.add : {α : Type u_1} → [self : Add α] → α → α → α

#check @add_comm
-- @add_comm : ∀ {G : Type u_1} [inst : AddCommSemigroup G]
--   (a b : G), a + b = b + a

#check @abs_add
-- @abs_add : ∀ {α : Type u_1}
--    [inst : LinearOrderedAddCommGroup α] (a b : α),
--      |a + b| ≤ |a| + |b|

#check @Continuous
-- @Continuous : {α : Type u_2} → {β : Type u_1} →
--    [inst : TopologicalSpace α] →
--    [inst : TopologicalSpace β] →
--    (α → β) → Prop
```

Defining structures and instances

```
variable (f g : ℝ × ℝ → ℝ)

#check f + g
-- f + g : ℝ × ℝ → ℝ

example : f + g = g + f := by rw [add_comm]

#check Continuous f
-- Continuous f : Prop
```

Defining structures and instances

```
set_option pp.explicit true
#check Continuous f
/-
@Continuous (ℝ × ℝ) ℝ
  (@instTopologicalSpaceProd ℝ ℝ
    (@UniformSpace.toTopologicalSpace ℝ (@PseudoMetricSpace.toUniformSpace ℝ
    Real.pseudoMetricSpace))
    (@UniformSpace.toTopologicalSpace ℝ (@PseudoMetricSpace.toUniformSpace ℝ
    Real.pseudoMetricSpace)))
  (@UniformSpace.toTopologicalSpace ℝ (@PseudoMetricSpace.toUniformSpace ℝ
    Real.pseudoMetricSpace)) f : Prop
-/
```

Defining a hierarchy of structures

Last I checked, Mathlib had:

- 1,781 classes
- 31,165 instances.

I will pause here to show you:

- some graphs
- how to look up the (direct) instances of a class in the Mathlib documentation
- how to look up the classes that an object is an instance of.

Defining a hierarchy of structures

What are all these classes?

- Notation (`Add`, `Mul`, `Inv`, `Norm`, ...)
- Algebraic structures (`Group`, `OrderedRing`, `Lattice`, `Module`, ...)
- Computation and bookkeeping: `Inhabited`, `Decidable`
- Mixins and add-ons: `LeftDistribClass`, `Nontrivial`
- Unexpected generalizations: `GroupWithZero`, `DivInvMonoid`

Defining a hierarchy of structures

```
class DivisionSemiring (α : Type*) extends Semiring α, GroupWithZero α

class DivisionRing (K : Type u) extends Ring K, DivInvMonoid K, Nontrivial K,
    RatCast K

class Semifield (α : Type*) extends CommSemiring α, DivisionSemiring α,
    CommGroupWithZero α

class Field (K : Type u) extends CommRing K, DivisionRing K
```

Automation

Current automation for Lean:

- lots of small-scale tactics for doing useful things (casing on data, doing calculations, logical manipulations)
- domain specific automation, `linarith`, `ring`, `omega`, `monotonicity`, `FunProp`
- `simp`, equational reasoning and conditional simplification
- `Aesop`, a tableaux reasoner like Isabelle's `auto`.

Isabelle's sledgehammer:

- uses premise selection to choose a manageable set of relevant facts from the library
- exports problems from Isabelle to external ATPs and SMT solvers
- uses the results of the external tools to construct formally verified proofs.

Automation

We currently have a prototype sledgehammer:

- Premise selection: Thomas Zhu, Joshua Clune, A, Albert Jiang, Sean Welleck
- Translation (*Lean-auto*): Yicheng Qian, Joshua Clune, Clark Barrett, A
- Reconstruction (*Duper*): Joshua Clune, Yicheng Qian, Alexander Bentkamp, A

Lean-auto does a lot of work, instantiating generic theorems and finding the type class instances.

Conclusions

- Structural reasoning is one of the most salient and powerful features of modern mathematics.
- Any viable proof assistant for mathematics has to support it.
- It's not easy.
- Lean and Mathlib do pretty well.
- We are not out of the woods.
  - The complexity poses challenges for library development and maintenance.
  - It also poses challenges for automation.
- Automated reasoning is getting better.
- Machine learning will help.
- The next few years will be exciting.

Conclusions

Automated and interactive theorem proving are only of mathematical interest insofar as they useful for mathematics.

If we care about making these technologies useful for mathematics, the mathematics has to come first.

We owe a lot to Tom for keeping us mindful of that fact, and showing us what is possible.