Formalization in Lean

Sahan Wijetunga

2025-07-24

Outline

What is Formalization?

"Expressing mathematics (objects, arguments) in a format that a computer can handle and interact with rigorously." - Kevin Buzzard

Benefits of Formalization

- Finding Mistakes
- Allowing large scale collaboration
  - Equational theories project
  - Public formalization projects
  - Base productivity shrunk by a large constant factor
- Pedagogical benefits

Lean works using Type Theory

- Every object has an associated *Type*
  - $\sqrt{2} : \mathbb{R}$
  - $1 : \mathbb{N}$
  - $\mathrm{id} : \mathbb{Q} \to \mathbb{Q}$
  - $\mathbb{N} : \mathrm{Type}$
- Definitions must be created with types specified.

```
def foo (a: ℕ): ℕ := 2 * a

#eval foo 3      -- 6
```

- Types can be omitted with implicit notation.

```
def foo' {α β: Type} (f: α → β) (a: α): β := f a

#eval foo' foo 2 -- 4
```

Type Classes in Lean

- Implicit {} notation allows you to avoid giving types explicitly.
- Type classes allows you to have much more inferred from context

```
import Mathlib

open Module

variable {F V: Type} [Field F] [AddCommGroup V] [Module F V] [FiniteDimensional F V]

noncomputable example (W: Subspace F V): W ≃ₗ[F] Dual F (Dual F W) := evalEquiv F W
```

```
def Module.evalEquiv
      (R : Type u_3) (M : Type u_4) [CommSemiring R] [AddCommMonoid M]
      [Module R M] [IsReflexive R M] :
   M ≃ₗ[R] Dual R (Dual R M)
```
source

The bijection between a reflexive module and its double dual, bundled as a `LinearEquiv`.

▶ Equations

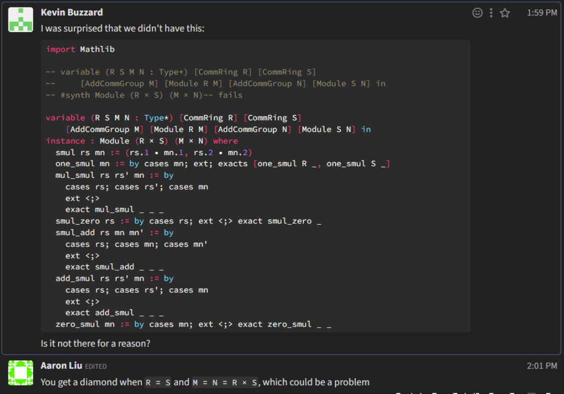
Issues with Type Classes

- Hard to see whats going on
- Slow compile times

```
import Mathlib.Tactic

open Module
open LinearMap
open LinearMap (BilinForm)

variable  {k V :Type} [Field k]
  [AddCommGroup V] [Module k V] [Module.Finite k V]

example : Ring (Module.End k V) := inferInstance        -- baseline; this succeeds

example : Ring (Module.End k (BilinForm k V)) := inferInstance -- fails

example : Algebra k (Module.End k (BilinForm k V)) := inferInstance -- succeeds?!
```

Diamonds

- Suppose A has an instance for B and C, which both have instances for D. Then A has two instances for D. Which to choose?



Kevin Buzzard 1:59 PM

I was surprised that we didn't have this:

```
import Mathlib

-- variable (R S M N : Type*) [CommRing R] [CommRing S]
--     [AddCommGroup M] [Module R M] [AddCommGroup N] [Module S N] in
-- #synth Module (R × S) (M × N)-- fails

variable (R S M N : Type*) [CommRing R] [CommRing S]
    [AddCommGroup M] [Module R M] [AddCommGroup N] [Module S N] in
instance : Module (R × S) (M × N) where
  smul rs mn := (rs.1 • mn.1, rs.2 • mn.2)
  one_smul mn := by cases mn; ext; exacts [one_smul R _, one_smul S _]
  mul_smul rs rs' mn := by
    cases rs; cases rs'; cases mn
    ext <;>
    exact mul_smul _ _ _
  smul_zero rs := by cases rs; ext <;> exact smul_zero _
  smul_add rs mn mn' := by
    cases rs; cases mn; cases mn'
    ext <;>
    exact smul_add _ _ _
  add_smul rs rs' mn := by
    cases rs; cases rs'; cases mn
    ext <;>
    exact add_smul _ _ _
  zero_smul mn := by cases mn; ext <;> exact zero_smul _ _
```

Is it not there for a reason?

Aaron Liu EDITED 2:01 PM

You get a diamond when `R = S` and `M = N = R × S`, which could be a problem

Proofs in Lean are guaranteed to be correct.

But what if your definitions are wrong?

Orthogonal Complement

- We say V is the internal direct sum of W_1 and W_2 with respect to a bilinear form $B : V \times V \to F$ if
  - W_1 and W_2 are *orthogonal*, i.e. that $B(w_1, w_2) = 0$ for all $w_1 \in W_1$ and $w_2 \in W_2$,
  - $W_1 + W_2 = V$,
  - $W_1 \cap W_2 = 0$.
- This doesn't force W_2 orthogonal to W_1! I.e, the form is only to be block upper triangular rather than block diagonal.
- Explicitly checking examples in Lean, and immediately proving compatibility results with similar theorems helps avoid this

Forced Implementation Choice

- Exact definition p of an object not viewed with much importance in math, as we can just prove $p \iff q$ and then use q everywhere
- Lean forces us to pick a privileged definition
- A basis in Lean internally is an F-linear isomorphism $V \simeq F^{\alpha}$
- F-linear isomorphisms are themselves implemented as maps $V \to W$ and $W \to V$ which are inverses.
- In the REU, I formalized Hyperbolic bilinear forms using a basis and a predicate, whereas the Mathlib one uses an isomorphism to $V^* \times V$
- Allowing computations or not

Canonical Isomorphism

- There is an obvious isomorphism $X \times (Y \times Z) \simeq (X \times Y) \times Z$ to where we just write $=$ directly.
- Often in algebra we just write $=$ for canonical isomorphism or even for definitions (via universal properties), like with $A \otimes_R B$.
- Limits practical freedom and forces people to be more explicit in choices

Quadratic Forms Extension by Scalars

- A given quadratic form $\phi : V \to F$ can be extended to one $A \otimes_F V \to A$ for F-algebras A.
- The Lean implementation of this goes through Bilinear forms, forcing $2 \neq 0$ in F.
- Fairly easy to find what is exactly needed/true for definitions from Mathlib
  - Improvement to traditional textbooks

Quadratic Forms Extension by Scalars

```
variable (R A) in
/-- The tensor product of two quadratic maps injects into quadratic maps on tensor products.

Note this is heterobasic; the quadratic map on the left can take values in a module over a larger
ring than the one on the right. -/
def tensorDistrib :
    QuadraticMap A M₁ N₁ ⊗[R] QuadraticMap R M₂ N₂ →ₗ[A] QuadraticMap A (M₁ ⊗[R] M₂) (N₁ ⊗[R] N₂) :=
  letI : Invertible (2 : A) := (Invertible.map (algebraMap R A) 2).copy 2 (map_ofNat _ _).symm
  -- while `letI`s would produce a better term than `let`, they would make this already-slow
  -- definition even slower.
  let toQ := BilinMap.toQuadraticMapLinearMap A A (M₁ ⊗[R] M₂)
  let tmulB := BilinMap.tensorDistrib R A (M₁ := M₁) (M₂ := M₂)
  let toB := AlgebraTensorModule.map
      (QuadraticMap.associated : QuadraticMap A M₁ N₁ →ₗ[A] BilinMap A M₁ N₁)
      (QuadraticMap.associated : QuadraticMap R M₂ N₂ →ₗ[R] BilinMap R M₂ N₂)
  toQ ∘ₗ tmulB ∘ₗ toB
```

Ended up using the external framework of theorems instead

Lean

Current Issues

- Strict Definitions
- *Dependent* Type Theory
- Universe issues (rarely)
- Manually using type classes not automatically instantiated to avoid diamonds
- Mathlib having large gaps

To be Improved On:

- Tooling
  - Better Tactics
  - AI Usage
- The Math

Contributing to Mathlib

- Code to be ported to Mathlib over time, cleaning portions at a time
- In the form of “pull requests”
- Results to be ported over:
  - Bilinear Form Isometries (Isomorphisms)
  - Notion of degree over polynomial modules, and the surrounding theory
  - Quotients of bilinear and quadratic forms
  - Surrounding theory for Hyperbolic Spaces
  - Compatibility of quadratic and bilinear forms and degree notions with extensions of scalars by the polynomial ring
  - Cassels-Pfister Theorem: The values taken by the extension of a quadratic map $\phi : V \to F$ to $V(X) \to F(X)$ that are in $F[X]$ are taken by the extension $V[X] \to F[X]$.

Current State

- The Symmetric algebra of a vector space has been formalized, however the grading on it has not (though being actively discussed).
  - This blocks progress on constructing extension by scalars in characteristic 2
- Fermat's Last Theorem: Current effort to formalize led by Kevin Buzzard
- The polynomial Freiman–Ruzsa conjecture was proved in 2023 by Tim Gowers, Ben Green, Freddie Manners, and Terry Tao, and has been formalized in Lean.

Thank you!

Special thanks to Dr. George McNinch, the VERSEIM REU, and National Science Foundation for their support under REU Site grant DMS-2349058."