Formalization project: proof assistants and some algebra

George McNinch

2025-06-04 22:45:34 EDT (george@valhalla)

Outline

Formalization via `Lean`

"Finite Algebra"

Proof assistants

- What is a proof assistant?
  A proof assistant is a piece of software that provides a language for defining objects, specifying properties of these objects, and proving that these specifications hold. The system checks that these proofs are correct down to their logical foundation.
- goal: produce verified proofs
  This is in contrast to automated theorem proving, where the focus is on the computer *constructing* the argument(s)
- What are some examples of proof assistants?
  - Lean, Agda, Coq, Mizar, HOL Light, HOL4, ...
  - differences? choice of logical foundations

why choose Lean?

- the foundations of Lean involve dependent type theory
  More precisely, Lean uses:
  a version of dependent type theory that is powerful enough to prove almost any conventional mathematical theorem, and expressive enough to do it in a natural way. More specifically, Lean is based on a version of a system known as the Calculus of Constructions with inductive types.
- main reason for the choice of Lean
  there has been a lot of "pure-math" activity in mathlib, a community maintained library (github repository) of pure mathematics results.

Topics of study in this area

- there is interest in studying type theoretic foundations, for example Homotopy Type Theory
- but the goal of this project is aligned with the philosophy of the mathlib community, namely to use Lean to formalize "ordinary" mathematics familiar to the community of pure mathematicians.

What does Lean look like?

- example: defn of convergence of a sequence

```
-- definition of "u tends to ℓ"
def seq_limit (u : ℕ → ℝ) (ℓ : ℝ) :=
  ∀ ε > 0, ∃ N, ∀ n ≥ N, |u n - ℓ| ≤ ε
```

- statement of the “squeeze theorem”

```
-- squeeze theorem
theorem (hu : seq_limit u ℓ)
        (hw : seq_limit w ℓ)
        (h : ∀ n, u n ≤ v n)
        (h' : ∀ n, v n ≤ w n) :
        seq_limit v ℓ := by
  sorry
```

more examples of Lean

- example: the first isomorphism theorem for rings

```
variable {R} [CommRing R]
variable {S} [CommRing S]

def firstIsomorphismTheorem (f : R →+* S)
    (hf : Function.Surjective f) :
    R / ker f ≃+* S := by
    sorry
```

(caveat: for TeXnical reasons, I've written the wrong unicode symbol '/' for the quotient operation)

Proving statements about constructions

- Lean is really just a programming language
- we can define the type constructor `List`

```
inductive List (α:Type) where
| nil : List α       -- empty list
| cons (x:α) (xs:List α) : List α
```

  in fact, under the hood Lean defines notation `[]` for `nil` and `x :: xs` for `cons x xs`.
- and e.g.

```
1 :: 2 :: 3 :: []
```

  can be represented by the notation `[1,2,3]`

appending lists

- Here is some `Lean` code that *appends two lists*.

```
def append {α:Type} (xs ys : List α)
  : List a :=
  match xs with
  | [] => ys
  | z :: zs => z :: append zs ys
```

- e.g.

```
append ["a", "b", "c"] ["d", "e"]
```

evaluates to

```
["a", "b", "c", "d", "e"] : List String
```

the proof

- Now, we can use Lean to prove a property of this function: namely, the length of the result is the sum of the lengths.
- here is the proof in `Lean`

```
theorem append_length {α:Type}
      (xs ys : List α)
      : (append xs ys).length =
        xs.length + ys.length := by
  induction xs with
  | nil => simp [append]
  | cons z zs ih =>
      simp [append, ih]
      linarith
```

- you can view this theorem `append_length` as a function of `xs` and `ys`, whose return value is the indicate *Proposition* confirming equality.

Finite vector spaces

- For the project, I have in mind producing formal proofs of statements about "finite algebraic objects". I'm going to formulate here an example of such a statement.
- Let k be a finite field.
  Recall (or accept my assertion for now!) that $|k|$ is a power p^n of a prime number p for some $n : \mathbf{N}$, and that up to isomorphism there is exactly one field of order p^n.
- thus $k \simeq \mathbf{F}_q$ where $q = p^n$, and $\mathbf{F}_p \simeq \mathbf{Z}/p\mathbf{Z}$.
- e.g. if $p \equiv 3 \pmod 4$ then $\mathbf{F}_{p^2} \simeq \mathbf{F}_p(i)$ where $i^2 = -1$. ("like forming $\mathbf{C}$ from $\mathbf{R}$")
- now let V be a finite dimensional vector space over k. If $\dim_k V = m$ then $|V| = q^m$.

Forms on finite vector spaces

- Let $\beta : V \times V \to k$ be a bilinear form
- and suppose that β is nondegenerate and symmetric
- (also suppose for convenience that $p > 2$)

Examples of forms

- when $\dim V = 2m$ is even, we can choose a basis $e_1, \cdots, e_m, f_1, \cdots, f_m$ of V.
  and we can define a form β_h by the rules

  $$\beta_h(e_i, f_j) = \delta_{i,j}, \quad \beta_h(e_i, e_j) = \beta_h(f_i, f_j) = 0$$

  - note e.g. that $\beta_h(e_i, e_i) = 0$ for any i.
- view $V = \mathbf{F}_{q^2}$ as a two-dimensional $\mathbf{F}_q$-vector space and consider the form

  $$\beta_a : V \times V \to \mathbf{F}_q$$

  given by $\beta(x, y) = \dfrac{(x+y)^{q+1} - x^{q+1} - y^{q+1}}{2}$
  note that $\beta_a(x, x) = x^{q+1} = 0 \implies x = 0$.

Classification of forms

- suppose $\dim V = d = 2m$ is even
- up to isomorphism, there are only two possibilities for β
  - either β is the hyperbolic form for some choice of basis $\{e_i, f_i\}$
  - or β is the orthogonal sum of a hyperbolic form of dimension $d-2$ and the two dimensional form β_a.
- this is analogous to Sylvester's theorem which describes the isomorphism classes of non-degenerate symmetric bilinear forms on a finite dimensional real vector space

What will be involved in formalizing this statement?

- must describe "orthogonal sums" of bilinear forms
- must describe what it means for two bilinear forms on V to be *isomorphic*
- probably need some way of describing β via a matrix, in order to keep track of what happens under *change-of-basis*
- must have a good understanding of the pen-and-paper proof!!