Rings, categories and schemes in Coq (II)

formalizing the functor of points in Coq/SSReflect/MathComp

Xuanrui Qi [1]

[1]Graduate School of Mathematics, Nagoya University

November 21, 2021
Theorem Proving and Provers 2021
Kitami, Japan

Starting from last time

The right way to work with rings

Previous goal Define a scheme using Coq/SSReflect, as a usability benchmark for SSReflect and MathComp libraries

Our conclusion... It seems that Coq is not very good for this kind of formalization work

Scheme So how do we define a scheme?

This talk is mainly about planned work and the author's position. The technical content outlined here might change.

What's the problem with Coq?

Quotient hell

2 ways to define quotients in Coq:

1. as setoids, or types with equivalence relations: "setoid hell" problem
2. as canonical projection maps, the SSReflect/MathComp approach

So why not `generic_quotient`?

- Just like setoid hell, `generic_quotient` has its own hell
- Quotient types $A/\sim$ defined as a type B with left invertible canonical projection $\pi : A \to B$
- Proving theorems rely on "lifting" lemmas, soon become tedious
- Gets even worse than setoid hell!

Alternatives to quotient hell

- add quotients to the language (Lean). Problem: breaks normalization, etc.
- use homotopy type theory (HoTT) with HITs, get quotients for free, and transport along equivalences
  - pros: is the natural way to do things, fits intuition about quotient types
  - cons: incompatible with choice & other non-constructive reasoning principles
  - cons (Coq only): HIT implementation in Coq HoTT lib is very dirty
  - cons (MathComp): MathComp has not been ported to HoTT yet

Alternatives to quotient hell

- add quotients to the language (Lean). Problem: breaks normalization, etc.
- use homotopy type theory (HoTT) with HITs, get quotients for free, and transport along equivalences
  - pros: is the natural way to do things, fits intuition about quotient types
  - cons: incompatible with choice & other non-constructive reasoning principles
  - cons (Coq only): HIT implementation in Coq HoTT lib is very dirty
  - cons (MathComp): MathComp has not been ported to HoTT yet

Alternatives to quotient hell

- add quotients to the language (Lean). Problem: breaks normalization, etc.
- use homotopy type theory (HoTT) with HITs, get quotients for free, and transport along equivalences
  - pros: is the natural way to do things, fits intuition about quotient types
  - cons: incompatible with choice & other non-constructive reasoning principles
  - cons (Coq only): HIT implementation in Coq HoTT lib is very dirty
  - cons (MathComp): MathComp has not been ported to HoTT yet

Alternatives to quotient hell

- add quotients to the language (Lean). Problem: breaks normalization, etc.
- use homotopy type theory (HoTT) with HITs, get quotients for free, and transport along equivalences
  - pros: is the natural way to do things, fits intuition about quotient types
  - cons: incompatible with choice & other non-constructive reasoning principles
  - cons (Coq only): HIT implementation in Coq HoTT lib is very dirty
  - cons (MathComp): MathComp has not been ported to HoTT yet

The dilemma

Coq/SSReflect/MathComp

- no support for quotient types, so either setoid hell or lifting hell
- no support for subset types, requires simple but tedious lifting
- HoTT fixes some problems, but incompatible with classical reasoning (& no MathComp)

The dilemma

Algebraic geometry

- lots of ideals and therefore quotients (e.g., in localization of rings)
- lots of reasoning about inclusions (e.g., maximal ideals, chains of ideals)
- generally requires classical reasoning principles
  - every ring has a maximal ideal
  - the ascending chain condition on ideals iff every ideal is finitely generated
  - Hilbert's basis theorem ($R[X]$ is Noetherian iff R Noetherian)
- easy quotient/subset reasoning ($\approx$ HoTT) + classical reasoning principles = inconsistency!

Algebraic geometry without (so many) ideals

Schemes can be defined without ideals!

Keyword: **functor of points**
Idea due to Grothendieck, references: Grothendieck's EGA I, Demazure & Gabriel (1970), Mumford & Oda (2015)

The Yoneda Lemma

Lemma: for any $X : \mathcal{C}$ and functor $F : \mathcal{C}^{\mathrm{op}} \to$ **Set**, there is a canonical isomorphism $\mathrm{Hom}(h_X, F) \cong F(X)$.

Basic result in category theory, not hard to prove (WIP in categories, along with theory of representable functors).

Idea: represent spaces/geometric objects as functors!

Affine schemes using functor of points

Definition: an affine scheme is just a representable functor from **CRing** to some suitable category **Set** of sets.
So simple! No prime ideals at all!

Replacing **Set** with **Type** should just work. More general schemes can be defined with the help of Grothendieck topologies.

Challenge: define a function to recover the underlying topological space. What if we replace spaces with locales? (both mathematical and technical)

Affine schemes using functor of points

Definition: an affine scheme is just a representable functor from **CRing** to some suitable category **Set** of sets.
So simple! No prime ideals at all!

Replacing **Set** with **Type** should just work. More general schemes can be defined with the help of Grothendieck topologies.

Challenge: define a function to recover the underlying topological space. What if we replace spaces with locales? (both mathematical and technical)

Why "functorial mathematics" in formalization

- type theory is structural, not material, so it is only natural to describe things structurally
- we generally care not about what something "is", but about how it interacts with other things (cf. Grothendieck's relative PoV)
- "material" mathematics is harder to make constructive
- looking ahead towards HoTT

See our code!

https://github.com/xuanruiqi/commalg
https://github.com/xuanruiqi/categories