# Algebraic Geometry in `Mathlib`

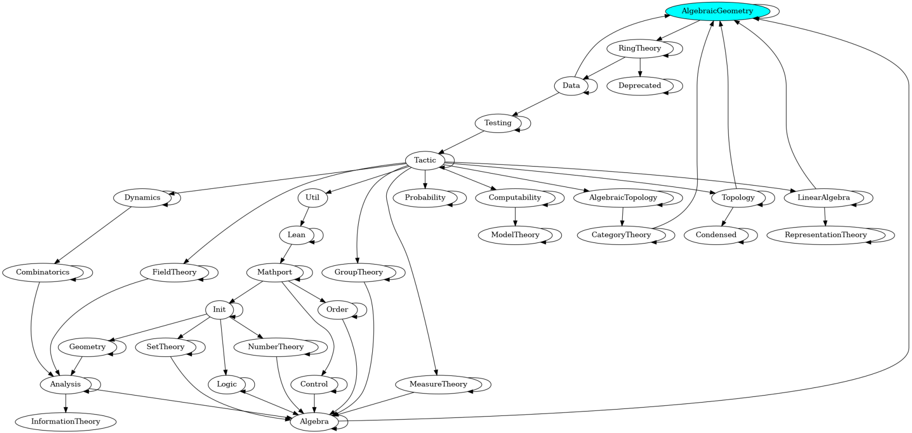
Damiano Testa

University of Warwick

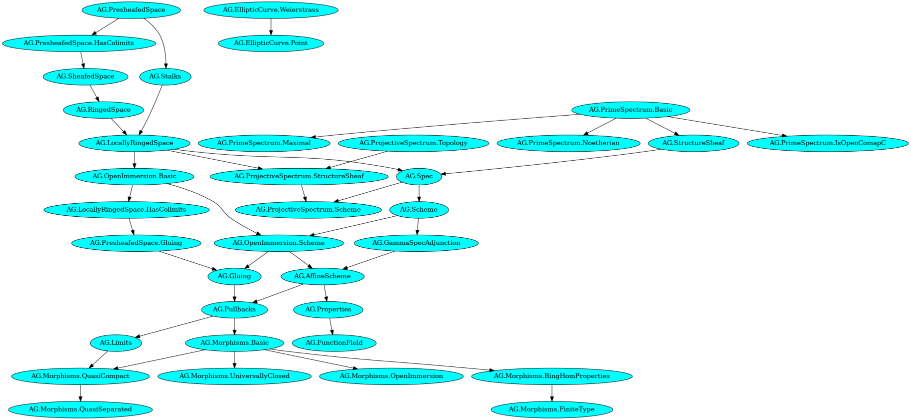
Lean for the Curious Mathematician 2023
Heinrich Heine University, Düsseldorf

September 7th, 2023

# An overview of Mathlib

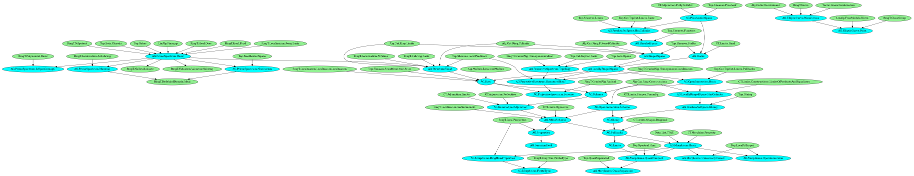


# AlgebraicGeometry in Mathlib

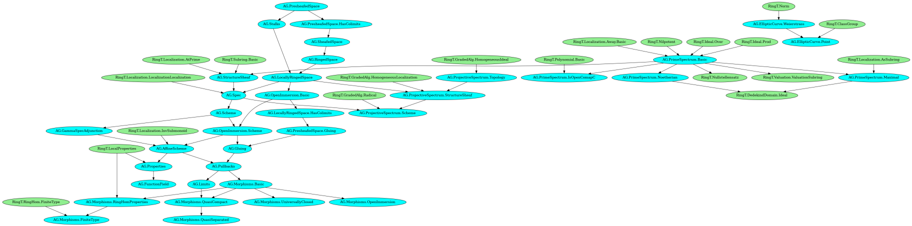



| | AG | Mathlib |
|---|---|---|
| Files | 36 | 3.7k |
| Lines of code | 18.5k | 1.3M |

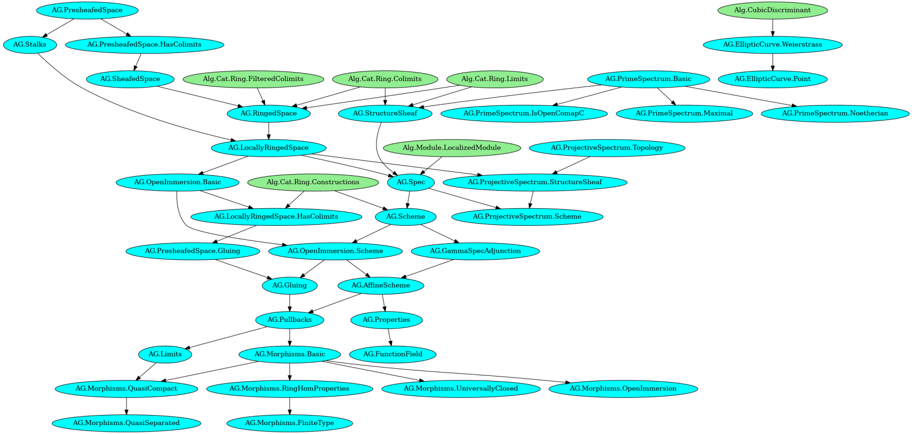
# AlgebraicGeometry and anything else



# AlgebraicGeometry and RingTheory



# AlgebraicGeometry and Algebra



If you were quick, you will have seen that `AlgebraicGeometry` has direct imports from

- `RingTheory`, importing directly 20 files;
- `Topology`, importing directly 17 files;
- `CategoryTheory`, importing directly 9 files;
- `Algebra`, importing directly 6 files;
- `LinearAlgebra`, importing directly 2 files;
- `Tactic`, importing directly 1 file;
- `Data`, importing directly 1 file.

Most of the files in `AlgebraicGeometry` make substantial use of the `CategoryTheory` library of `Mathlib`.

However, the `CategoryTheory` library deals explicitly with `TypeClass` inference and `Universe Levels`.

Neither of these topics is particularly beginner-friendly.

This presentation and the exercises rely almost entirely on the dependencies outside of `CategoryTheory`.

# What is Algebraic Geometry?

Broadly, the study of vanishing sets of polynomials.

Polynomials provide the `algebra`, vanishing sets convert them to `geometry`.

Mixing the two concepts involves working with (commutative, unital semi-)rings.

For this presentation, we focus on rings.

In `Mathlib`, this is the typeclass `CommRing`: commutative, associative, unital `Semiring`s with additive opposites.

In `Mathlib` you should not give *any* of these assumptions for granted!

As we said, we focus on `CommRing`s.

Let `R` be a ring.

Algebraic Geometry extracts mainly two layers of information from `R`:

- a topological space $\operatorname{Spec} R$ – the *spectrum* of `R`;
- a sheaf of rings on $\operatorname{Spec} R$ – the *structure sheaf* $\mathscr{O}_{\operatorname{Spec} R}$ on $\operatorname{Spec} R$.

The pair $(\operatorname{Spec} R, \mathscr{O}_{\operatorname{Spec} R})$ is an *affine scheme*.

The construction mentioned above is such that the resulting affine scheme is a *locally ringed space*.

A *scheme* is a locally ringed space that admits a cover by open subsets each of which is isomorphic to an affine scheme.

None of the above will matter for solving the exercises.

Review. A ring `R` gives us an affine scheme: the locally ringed space $\operatorname{Spec} R$.

The ring `R` is uniquely determined by $\operatorname{Spec} R$.

A *scheme* is any (locally ringed) space that is obtained by gluing together affine schemes.

Here is the notion of affine space over a field embedded into `Lean`, using `Mathlib`.

```
import Mathlib.AlgebraicGeometry.AffineScheme
import Mathlib.AlgebraicGeometry.ProjectiveSpectrum.Scheme

noncomputable section Spec_and_Proj

open  AlgebraicGeometry Scheme -- self-explanatory?
      CommRingCat  -- the Category of Commutative Rings
      Opposite     --  Opposite categories

abbrev Spec (R) [CommRing R] := Scheme.Spec.obj (op (of R))

-- The n-dimensional affine space over the field k.
def 𝔸 (k) [Field k] (n : ℕ) : Scheme :=
Spec (MvPolynomial (Fin n) k)

#check 𝔸 ℚ 4   -- 𝔸 ℚ 4 : Scheme
```

```
example (k) [Field k] (n : ℕ) : IsAffine (𝔸 k n) := by
  sorry
  --exact?     -- fails, since it does not see through '𝔸'
  --unfold 𝔸    -- now 'exact?' works
  --exact SpecIsAffine (op (of (MvPolynomial (Fin n) k)))

--  A quick definition of k-valued points
def k_valued_points (k) [Field k] (X : Scheme) :=
Spec k ⟶ X  --  ← is an arrow in the Scheme category,
            --  not a 'usual' function!

variable {k R} [Field k] [CommRing R]

example (f : R →+* k) : k_valued_points k (Spec R) := by
  change of R ⟶ of k at f
  exact (specMap f)
```

# Projective schemes (as locally ringed spaces)

```
variable {A : Type*} [CommRing A] [Algebra R A]

variable (𝒜 : ℕ → Submodule R A) [GradedAlgebra 𝒜]

def P : LocallyRingedSpace := Proj.toLocallyRingedSpace 𝒜

#check P 𝒜    -- P 𝒜 : LocallyRingedSpace

end Spec_and_Proj
```