Algebraic Geometry in Lean's mathematical library mathlib

Christian Merten

Utrecht University

Sep 15, 2025

What is algebraic geometry?

- Classically: Study of zeros of polynomials over a field.

$$X = \{x \in k^n \mid 0 = f_1(x) = \ldots = f_k(x)\}.$$

What is algebraic geometry?

- Classically: Study of zeros of polynomials over a field.

$$X = \{x \in k^n \mid 0 = f_1(x) = \ldots = f_k(x)\}.$$

- Reformulation in the language of schemes by Grothendieck: Instead of studying the set X, study the ring of rational functions on X.

$$A = k[T_1, \ldots, T_n]/(f_1, \ldots, f_k).$$

What is algebraic geometry?

- Classically: Study of zeros of polynomials over a field.

$$X = \{x \in k^n \mid 0 = f_1(x) = \ldots = f_k(x)\}.$$

- Reformulation in the language of schemes by Grothendieck: Instead of studying the set X, study the ring of rational functions on X.

$$A = k[T_1, \ldots, T_n]/(f_1, \ldots, f_k).$$

- General construction: Spec associates an affine scheme to any commutative ring.

What is algebraic geometry?

- Classically: Study of zeros of polynomials over a field.

$$X = \{x \in k^n \mid 0 = f_1(x) = \ldots = f_k(x)\}.$$

- Reformulation in the language of schemes by Grothendieck: Instead of studying the set X, study the ring of rational functions on X.

$$A = k[T_1, \ldots, T_n]/(f_1, \ldots, f_k).$$

- General construction: Spec associates an affine scheme to any commutative ring.
- Affine schemes can be completely understood via the study of commutative rings.

What is algebraic geometry?

- Classically: Study of zeros of polynomials over a field.

$$X = \{x \in k^n \mid 0 = f_1(x) = \ldots = f_k(x)\}.$$

- Reformulation in the language of schemes by Grothendieck: Instead of studying the set X, study the ring of rational functions on X.

$$A = k[T_1, \ldots, T_n]/(f_1, \ldots, f_k).$$

- General construction: Spec associates an affine scheme to any commutative ring.
- Affine schemes can be completely understood via the study of commutative rings.
- A scheme is a geometric object, that locally looks like an affine scheme.

Outline

Short history

2003 Formalization of affine schemes in Rocq (formerly Coq) by Chicli.

Short history

2003	Formalization of affine schemes in Rocq (formerly Coq) by Chicli.
2018	First definition of schemes by Buzzard and two undergraduates Hughes and Lau in Lean 3.

Short history

2003 Formalization of affine schemes in Rocq (formerly Coq) by Chicli.

2018 First definition of schemes by Buzzard and two undergraduates Hughes and Lau in Lean 3.

2020 Definition of schemes enters `mathlib` (Lean 3) with contributions by Livingston, Fernández Mir and Morrison.

Short history

2003	Formalization of affine schemes in Rocq (formerly Coq) by Chicli.
2018	First definition of schemes by Buzzard and two undergraduates Hughes and Lau in Lean 3.
2020	Definition of schemes enters `mathlib` (Lean 3) with contributions by Livingston, Fernández Mir and Morrison.
2021	Elliptic curves as cubic equations by Angdinata and Buzzard.

Short history

2003 Formalization of affine schemes in Rocq (formerly Coq) by Chicli.

2018 First definition of schemes by Buzzard and two undergraduates Hughes and Lau in Lean 3.

2020 Definition of schemes enters `mathlib` (Lean 3) with contributions by Livingston, Fernández Mir and Morrison.

2021 Elliptic curves as cubic equations by Angdinata and Buzzard.

2022 Definition of schemes in Isabelle by Bordg, Paulson, Li.

Short history

2003	Formalization of affine schemes in Rocq (formerly Coq) by Chicli.
2018	First definition of schemes by Buzzard and two undergraduates Hughes and Lau in Lean 3.
2020	Definition of schemes enters `mathlib` (Lean 3) with contributions by Livingston, Fernández Mir and Morrison.
2021	Elliptic curves as cubic equations by Angdinata and Buzzard.
2022	Definition of schemes in Isabelle by Bordg, Paulson, Li.
2022	Construction of fibred products by Yang (Lean 3).

Short history

2003	Formalization of affine schemes in Rocq (formerly Coq) by Chicli.
2018	First definition of schemes by Buzzard and two undergraduates Hughes and Lau in Lean 3.
2020	Definition of schemes enters `mathlib` (Lean 3) with contributions by Livingston, Fernández Mir and Morrison.
2021	Elliptic curves as cubic equations by Angdinata and Buzzard.
2022	Definition of schemes in Isabelle by Bordg, Paulson, Li.
2022	Construction of fibred products by Yang (Lean 3).
2023	Port of definition to `mathlib4` (Lean 4).

Short history

- 2003 Formalization of affine schemes in Rocq (formerly Coq) by Chicli.
- 2018 First definition of schemes by Buzzard and two undergraduates Hughes and Lau in Lean 3.
- 2020 Definition of schemes enters `mathlib` (Lean 3) with contributions by Livingston, Fernández Mir and Morrison.
- 2021 Elliptic curves as cubic equations by Angdinata and Buzzard.
- 2022 Definition of schemes in Isabelle by Bordg, Paulson, Li.
- 2022 Construction of fibred products by Yang (Lean 3).
- 2023 Port of definition to `mathlib4` (Lean 4).
- 2024 Etale site in `mathlib4`.

A word on Lean and mathlib

- Lean is a dependently typed interactive theorem prover, initially developed by Leonardo de Moura at Microsoft Research and since 2023 mainly developed by the Lean FRO.

A word on Lean and mathlib

- Lean is a dependently typed interactive theorem prover, initially developed by Leonardo de Moura at Microsoft Research and since 2023 mainly developed by the Lean FRO.
- Mathlib is a user-maintained mathematical library for Lean, covering a broad range of mathematics.

Attributions

- The algebraic geometry library in mathlib has seen contributions by many people in the past, including Angdinata, Buzzard, Commelin, Morrison, Riou, Xu, Yang, Zhang, M.

Attributions

- The algebraic geometry library in mathlib has seen contributions by many people in the past, including Angdinata, Buzzard, Commelin, Morrison, Riou, Xu, Yang, Zhang, M.
- Angdinata and Xu are the driving forces behind elliptic curves.

Attributions

- The algebraic geometry library in mathlib has seen contributions by many people in the past, including Angdinata, Buzzard, Commelin, Morrison, Riou, Xu, Yang, Zhang, M.
- Angdinata and Xu are the driving forces behind elliptic curves.
- The schemes library has recently been mainly developed by Andrew Yang and C.M.

For the algebraic geometry experts: What do we have?

- Limit properties of the category of schemes: existence of finite limits and coproducts, properties of inverse limits

For the algebraic geometry experts: What do we have?

- Limit properties of the category of schemes: existence of finite limits and coproducts, properties of inverse limits
- Group law on elliptic curves

For the algebraic geometry experts: What do we have?

- Limit properties of the category of schemes: existence of finite limits and coproducts, properties of inverse limits
- Group law on elliptic curves
- Many properties of morphisms, e.g., closed immersion, finite, separated, universally closed, locally of finite type, smooth, unramified, étale, etc.

For the algebraic geometry experts: What do we have?

- Limit properties of the category of schemes: existence of finite limits and coproducts, properties of inverse limits
- Group law on elliptic curves
- Many properties of morphisms, e.g., closed immersion, finite, separated, universally closed, locally of finite type, smooth, unramified, étale, etc.
- Valuative criteria

For the algebraic geometry experts: What do we have?

- Limit properties of the category of schemes: existence of finite limits and coproducts, properties of inverse limits
- Group law on elliptic curves
- Many properties of morphisms, e.g., closed immersion, finite, separated, universally closed, locally of finite type, smooth, unramified, étale, etc.
- Valuative criteria
- Ideal sheafs

For the algebraic geometry experts: What do we have?

- Limit properties of the category of schemes: existence of finite limits and coproducts, properties of inverse limits
- Group law on elliptic curves
- Many properties of morphisms, e.g., closed immersion, finite, separated, universally closed, locally of finite type, smooth, unramified, étale, etc.
- Valuative criteria
- Ideal sheafs
- Big and small Zariski and étale sites

For the algebraic geometry experts: What do we have?

- Limit properties of the category of schemes: existence of finite limits and coproducts, properties of inverse limits
- Group law on elliptic curves
- Many properties of morphisms, e.g., closed immersion, finite, separated, universally closed, locally of finite type, smooth, unramified, étale, etc.
- Valuative criteria
- Ideal sheafs
- Big and small Zariski and étale sites
- Projective space

Definition of Schemes

```
structure Scheme extends (X : LocallyRingedSpace) where
  local_affine :
    ∀ x : X,
      ∃ (U : OpenNhds x) (R : CommRingCat),
        Nonempty
          (X.restrict U ≅ Spec.obj (op R))
```

Definition of Schemes

```
structure Scheme extends (X : LocallyRingedSpace) where
  local_affine :
    ∀ x : X,
      ∃ (U : OpenNhds x) (R : CommRingCat),
        Nonempty
          (X.restrict U ≅ Spec.obj (op R))
```

- To introduce a scheme, we do:

```
variable (X : Scheme)
```

Definition of Schemes

```
structure Scheme extends (X : LocallyRingedSpace) where
  local_affine :
    ∀ x : X,
      ∃ (U : OpenNhds x) (R : CommRingCat),
        Nonempty
          (X.restrict U ≅ Spec.obj (op R))
```

- To introduce a scheme, we do:
  ```
  variable (X : Scheme)
  ```
- Main reason: many arguments in algebraic geometry use category theoretical tools.

(Un)bundled geometric objects

- The differential geometry library follows the unbundled design:

```
variable {𝕜 : Type} [NontriviallyNormedField 𝕜]
  {E : Type} [NormedAddCommGroup E] [NormedSpace 𝕜 E]
  {H : Type} [TopologicalSpace H] {I : ModelWithCorners 𝕜 E H}
  {M : Type} [TopologicalSpace M] [ChartedSpace H M]
  {n : WithTop ℕ∞} [IsManifold I n M]
```

(Un)bundled geometric objects

- The differential geometry library follows the unbundled design:

```
variable {𝕜 : Type} [NontriviallyNormedField 𝕜]
  {E : Type} [NormedAddCommGroup E] [NormedSpace 𝕜 E]
  {H : Type} [TopologicalSpace H] {I : ModelWithCorners 𝕜 E H}
  {M : Type} [TopologicalSpace M] [ChartedSpace H M]
  {n : WithTop ℕ∞} [IsManifold I n M]
```

- Main advantage: A manifold is automatically also a topological space and every result immediately applies.

From commutative algebra to algebraic geometry

- Reminder: affine schemes are of the form $\mathrm{Spec}(R)$ for some ring R and every scheme locally looks like an affine scheme.

From commutative algebra to algebraic geometry

- Reminder: affine schemes are of the form $\mathrm{Spec}(R)$ for some ring R and every scheme locally looks like an affine scheme.
- To reason about schemes, one reduces the problem to affine schemes and then solves the resulting commutative algebra problem.

From commutative algebra to algebraic geometry

Pour démontrer (i), notons que la question est locale sur X ; on peut donc se restreindre au cas où X est affine. Soit U un ouvert affine dans V ; $j(X) \cap U$ est un ouvert

Figure: EGA II

From commutative algebra to algebraic geometry

Pour démontrer (i), notons que la question est locale sur X ; on peut donc se restreindre au cas où X est affine. Soit U un ouvert affine dans V ; $j(X) \cap U$ est un ouvert

Figure: EGA II

$y \in f(X)$. Now we can replace Y by an affine neighborhood of y, and so assume that Y is affine. Then since f is quasi-compact, X will be a finite

Figure: *Hartshorne*, Algebraic Geometry

From commutative algebra to algebraic geometry

Pour démontrer (i), notons que la question est locale sur X ; on peut donc se restreindre au cas où X est affine. Soit U un ouvert affine dans V ; $j(\mathrm{X}) \cap \mathrm{U}$ est un ouvert

Figure: EGA II

$y \in f(X)$. Now we can replace Y by an affine neighborhood of y, and so assume that Y is affine. Then since f is quasi-compact, X will be a finite

Figure: *Hartshorne*, Algebraic Geometry

Proof. Assume X is Nagata. Let $Z \subset X$ be an integral closed subscheme. Let $z \in Z$. Let $\mathrm{Spec}(A) = U \subset X$ be an affine open containing z such that A is Nagata.

Figure: Stacks Project

Open subsets

- A consequence of bundling: An open subset of a scheme is not a scheme.

Open subsets

- A consequence of bundling: An open subset of a scheme is not a scheme.
- In particular:

```
example (X : Scheme) (U : Opens X) (hU : IsAffineOpen U) :
    ∃ (R : CommRingCat), U = Spec R :
  sorry
```

does not typecheck!

Open subsets

- A consequence of bundling: An open subset of a scheme is not a scheme.
- In particular:

```
example (X : Scheme) (U : Opens X) (hU : IsAffineOpen U) :
    ∃ (R : CommRingCat), U = Spec R :
  sorry
```

  does not typecheck!
- Workaround: develop the API in terms of abstract open immersions `f : U → X`.

Local properties

- Define a predicate:

```
def IsLocal (P : Scheme → Prop) : Prop :=
  ∀ (X : Scheme), P X ↔ (∀ (U : X.affineOpens), P U)
```

Local properties

- Define a predicate:

```
def IsLocal (P : Scheme → Prop) : Prop :=
  ∀ (X : Scheme), P X ↔ (∀ (U : X.affineOpens), P U)
```

and an induction principle:

```
lemma of_isLocal (P : Scheme → Prop) (h : IsLocal P) (X : Scheme)
    (hX : ∀ R, P (Spec R)) :
    P X :=
  /- ... -/
```

Local properties

- Define a predicate:

```
def IsLocal (P : Scheme → Prop) : Prop :=
  ∀ (X : Scheme), P X ↔ (∀ (U : X.affineOpens), P U)
```

  and an induction principle:

```
lemma of_isLocal (P : Scheme → Prop) (h : IsLocal P) (X : Scheme)
    (hX : ∀ R, P (Spec R)) :
    P X :=
  /- ... -/
```

- What about properties involving multiple objects and morphisms between them?

Local properties of morphisms

- A property of morphisms can be local at the source and at the target.

Local properties of morphisms

- A property of morphisms can be local at the source and at the target.
- For the target, we have:

```
class IsLocalAtTarget (P : MorphismProperty Scheme) : Prop where
  iff_of_openCover :
    ∀ {X Y : Scheme} (f : X → Y) (𝒰 : Y.OpenCover),
      P f ↔ ∀ i, P (𝒰.pullbackHom f i)
```

Local properties of morphisms

- This yields structured proofs:

```
lemma foo {X Y : Scheme} (f : X → Y) : P f := by
  wlog hY : ∃ R, Y = Spec R
  · rw [LocalAtTarget.iff_of_openCover Y.affineCover]
    /- ... -/
  obtain ⟨R, rfl⟩ := hY
  wlog hX : ∃ S, X = Spec S
  · /- ... -/
  obtain ⟨S, rfl⟩ := hX
  obtain ⟨φ, rfl⟩ := Spec.map_surjective f
  /- ... -/
```

Local properties of morphisms

- This yields structured proofs:

```
lemma foo {X Y : Scheme} (f : X → Y) : P f := by
  wlog hY : ∃ R, Y = Spec R
  · rw [LocalAtTarget.iff_of_openCover Y.affineCover]
    /- ... -/
  obtain ⟨R, rfl⟩ := hY
  wlog hX : ∃ S, X = Spec S
  · /- ... -/
  obtain ⟨S, rfl⟩ := hX
  obtain ⟨φ, rfl⟩ := Spec.map_surjective f
  /- ... -/
```

- Clear separation of concerns.

Local properties of morphisms

- This yields structured proofs:

```
lemma foo {X Y : Scheme} (f : X → Y) : P f := by
  wlog hY : ∃ R, Y = Spec R
  · rw [LocalAtTarget.iff_of_openCover Y.affineCover]
    /- ... -/
  obtain ⟨R, rfl⟩ := hY
  wlog hX : ∃ S, X = Spec S
  · /- ... -/
  obtain ⟨S, rfl⟩ := hX
  obtain ⟨φ, rfl⟩ := Spec.map_surjective f
  /- ... -/
```

- Clear separation of concerns.
- We are relying on the bundled approach here.

Defining new properties

- So far, we only considered properties that happen to satisfy some locality condition.

Defining new properties

- So far, we only considered properties that happen to satisfy some locality condition.
- In practice, many definitions are even *defined* from local conditions.

Defining new properties

- So far, we only considered properties that happen to satisfy some locality condition.
- In practice, many definitions are even *defined* from local conditions.

Definition

A morphism $f \colon X \to Y$ of schemes is *smooth* if for every $x \in X$, there exist affine neighbourhoods $x \in U$ and $f(x) \in V$ such that $f(U) \subseteq V$ and the restriction $f : U \to V$ is a smooth morphism of affine schemes.

Defining new properties

```
/-- A morphism of schemes is smooth if locally it is a smooth morphism
of affine schemes. -/
def Smooth {X Y : Scheme} (f : X → Y) : Prop :=
  ∀ (x : X), ∃ (U : X.affineOpens) (V : Y.affineOpens),
    x ∈ U ∧ f '' U ⊆ V ∧ AffineScheme.Smooth (f.restrict U V)
```

Defining new properties

```
/-- A morphism of schemes is smooth if locally it is a smooth morphism
of affine schemes. -/
def Smooth {X Y : Scheme} (f : X → Y) : Prop :=
  ∀ (x : X), ∃ (U : X.affineOpens) (V : Y.affineOpens),
    x ∈ U ∧ f '' U ⊆ V ∧ AffineScheme.Smooth (f.restrict U V)
lemma Smooth.comp {X Y Z : Scheme} {f : X → Y} {g : Y → Z}
    (hf : Smooth f) (hg : Smooth g) :
    Smooth (f ≫ g) :=
  /- ... -/
```

Defining new properties

```
/-- A morphism of schemes is smooth if locally it is a smooth morphism
of affine schemes. -/
def Smooth {X Y : Scheme} (f : X → Y) : Prop :=
  ∀ (x : X), ∃ (U : X.affineOpens) (V : Y.affineOpens),
    x ∈ U ∧ f '' U ⊆ V ∧ AffineScheme.Smooth (f.restrict U V)
lemma Smooth.comp {X Y Z : Scheme} {f : X → Y} {g : Y → Z}
    (hf : Smooth f) (hg : Smooth g) :
    Smooth (f ≫ g) :=
  /- ... -/
```

- These proofs are tedious and repetitive (we have to do this for locally of finite type, locally of finite presentation, finite, smooth, unramified, étale, flat, etc.)

Defining new properties

```
/-- A morphism of schemes is smooth if locally it is a smooth morphism
of affine schemes. -/
def Smooth {X Y : Scheme} (f : X ⟶ Y) : Prop :=
  ∀ (x : X), ∃ (U : X.affineOpens) (V : Y.affineOpens),
    x ∈ U ∧ f '' U ⊆ V ∧ AffineScheme.Smooth (f.restrict U V)
lemma Smooth.comp {X Y Z : Scheme} {f : X ⟶ Y} {g : Y ⟶ Z}
    (hf : Smooth f) (hg : Smooth g) :
    Smooth (f ≫ g) :=
  /- ... -/
```

- These proofs are tedious and repetitive (we have to do this for locally of finite type, locally of finite presentation, finite, smooth, unramified, étale, flat, etc.)
- The textbook proof of this fact is:

Defining new properties

```
/-- A morphism of schemes is smooth if locally it is a smooth morphism
of affine schemes. -/
def Smooth {X Y : Scheme} (f : X → Y) : Prop :=
  ∀ (x : X), ∃ (U : X.affineOpens) (V : Y.affineOpens),
    x ∈ U ∧ f '' U ⊆ V ∧ AffineScheme.Smooth (f.restrict U V)
lemma Smooth.comp {X Y Z : Scheme} {f : X → Y} {g : Y → Z}
    (hf : Smooth f) (hg : Smooth g) :
    Smooth (f ≫ g) :=
  /- ... -/
```

- These proofs are tedious and repetitive (we have to do this for locally of finite type, locally of finite presentation, finite, smooth, unramified, étale, flat, etc.)
- The textbook proof of this fact is:

Proof.
The assertion is local, so it follows from the fact that the composition of smooth morphisms of affine schemes is smooth. □

Defining new properties

- From a property of morphisms on affine schemes, we obtain a property of morphisms of schemes:

```
def induced (P : MorphismProperty AffineScheme) :
    MorphismProperty Scheme :=
  fun f ↦ ∀ (x : X), ∃ (U : X.affineOpens) (V : Y.affineOpens),
    x ∈ U ∧ f '' U ⊆ V ∧ P (f.restrict U V)
```

Defining new properties

- From a property of morphisms on affine schemes, we obtain a property of morphisms of schemes:

```
def induced (P : MorphismProperty AffineScheme) :
    MorphismProperty Scheme :=
  fun f ↦ ∀ (x : X), ∃ (U : X.affineOpens) (V : Y.affineOpens),
    x ∈ U ∧ f '' U ⊆ V ∧ P (f.restrict U V)
```

- We immediately obtain a definition of smooth:

```
def Smooth {X Y : Scheme} (f : X → Y) : Prop :=
  induced AffineScheme.Smooth f
```

Defining new properties

- We can now define meta properties:

```
def MorphismProperty.StableUnderComposition
    (P : MorphismProperty C) : Prop :=
  ∀ {X Y Z : C} {f : X ⟶ Y} {g : Y ⟶ Z}, P f → P g → P (f ≫ g)
```

Defining new properties

- We can now define meta properties:

```
def MorphismProperty.StableUnderComposition
    (P : MorphismProperty C) : Prop :=
  ∀ {X Y Z : C} {f : X ⟶ Y} {g : Y ⟶ Z}, P f → P g → P (f ≫ g)
```

- And prove meta theorems:

```
lemma stableUnderComposition_induced
    {P : MorphismProperty AffineScheme}
    (h : P.StableUnderComposition) :
    (induced P).StableUnderComposition :=
    /- ... -/
```

Are we done?

No, because:

- Reductions still contain a lot of boilerplate code.

```
lemma isClosedMap_iff_specializingMap (f : X → Y) [QuasiCompact f] :
    IsClosedMap f.base ↔ SpecializingMap f.base := by
  refine ⟨fun h ↦ h.specializingMap, fun H ↦ ?_⟩
  wlog hY : ∃ R, Y = Spec R
  · change topologically @IsClosedMap f
    rw [IsLocalAtTarget.iff_of_openCover Y.affineCover]
    intro i
    refine this (Y.affineCover.pullbackHom f i) ?_ ⟨_, rfl⟩
    exact IsLocalAtTarget.of_isPullback (.of_hasPullback _ _) H
  obtain ⟨S, rfl⟩ := hY
  intro Z hZ
  replace H := hZ.stableUnderSpecialization.image H
  wlog hX : ∃ R, X = Spec R
  · obtain ⟨R, g, hg⟩ := compactSpace_iff_exists.mp (/- ... -/)
    have inst : QuasiCompact (g ≫ f) :=
      HasAffineProperty.iff_of_isAffine.mpr (by infer_instance)
    have := this _ (g ≫ f) (g.base ⁻¹' Z) (hZ.preimage g.continuous)
    /- ... -/
    exact this H ⟨_, rfl⟩
  obtain ⟨R, rfl⟩ := hX
  obtain ⟨φ, rfl⟩ := Spec.homEquiv.symm.surjective f
  exact PrimeSpectrum.isClosed_image_of_stableUnderSpecialization
    φ.hom Z hZ H
```

```
lemma isClosedMap_iff_specializingMap (f : X → Y) [QuasiCompact f] :
    IsClosedMap f.base ↔ SpecializingMap f.base := by
  refine ⟨fun h ↦ h.specializingMap, fun H ↦ ?_⟩
  wlog hY : ∃ R, Y = Spec R
  · change topologically @IsClosedMap f
    rw [IsLocalAtTarget.iff_of_openCover Y.affineCover]
    intro i
    refine this (Y.affineCover.pullbackHom f i) ?_ ⟨_, rfl⟩
    exact IsLocalAtTarget.of_isPullback (.of_hasPullback _ _) H
  obtain ⟨S, rfl⟩ := hY
  intro Z hZ
  replace H := hZ.stableUnderSpecialization.image H
  wlog hX : ∃ R, X = Spec R
  · obtain ⟨R, g, hg⟩ := compactSpace_iff_exists.mp (/- ... -/)
    have inst : QuasiCompact (g ≫ f) :=
      HasAffineProperty.iff_of_isAffine.mpr (by infer_instance)
    have := this _ (g ≫ f) (g.base ⁻¹' Z) (hZ.preimage g.continuous)
    /- ... -/
    exact this H ⟨_, rfl⟩
  obtain ⟨R, rfl⟩ := hX
  obtain ⟨φ, rfl⟩ := Spec.homEquiv.symm.surjective f
  exact PrimeSpectrum.isClosed_image_of_stableUnderSpecialization
    φ.hom Z hZ H
```

Are we done?

No, because:

- Reductions still contain a lot of boilerplate code.

Are we done?

No, because:

- Reductions still contain a lot of boilerplate code.
- For a single morphism, we already have two properties:

```
class IsLocalAtTarget (P : MorphismProperty Scheme) : Prop where
  iff_of_openCover :
    ∀ {X Y : Scheme} (f : X → Y) (𝒰 : Y.OpenCover),
      P f ↔ ∀ i, P (𝒰.pullbackHom f i)

class IsLocalAtSource (P : MorphismProperty Scheme) : Prop where
  iff_of_openCover :
    ∀ {X Y : Scheme} (f : X → Y) (𝒰 : X.OpenCover),
      P f ↔ ∀ i, P (𝒰.map i ≫ f)
```

Are we done?

No, because:

- Reductions still contain a lot of boilerplate code.
- For a single morphism, we already have two properties:

```
class IsLocalAtTarget (P : MorphismProperty Scheme) : Prop where
  iff_of_openCover :
    ∀ {X Y : Scheme} (f : X → Y) (𝒰 : Y.OpenCover),
      P f ↔ ∀ i, P (𝒰.pullbackHom f i)

class IsLocalAtSource (P : MorphismProperty Scheme) : Prop where
  iff_of_openCover :
    ∀ {X Y : Scheme} (f : X → Y) (𝒰 : X.OpenCover),
      P f ↔ ∀ i, P (𝒰.map i ≫ f)
```

- What about diagrams with more schemes?

Are we done?

$$\begin{array}{ccc} & & Y \\ & & \downarrow \\ X & \longrightarrow & S \end{array} \qquad \begin{array}{ccc} & & Y \\ & & \uparrow \\ X & \longleftarrow & S \end{array} \qquad \begin{array}{ccc} T & \longleftarrow & Y \\ \downarrow & & \uparrow \\ X & \longrightarrow & S \end{array}$$

Are we done?

$$
\begin{array}{ccc}
 & & Y \\
 & & \downarrow \\
X & \longrightarrow & S
\end{array}
\qquad
\begin{array}{ccc}
 & & Y \\
 & & \uparrow \\
X & \longleftarrow & S
\end{array}
\qquad
\begin{array}{ccc}
T & \longleftarrow & Y \\
\downarrow & & \uparrow \\
X & \longrightarrow & S
\end{array}
$$

- Locality of properties of diagrams $D\colon \mathcal{J} \to \text{Scheme}$?

Local properties of diagrams

- Encode a diagram of shape $\mathcal{J}$ as a functor $D\colon \mathcal{J} \to \text{Scheme}$.

Local properties of diagrams

- Encode a diagram of shape $\mathcal{J}$ as a functor $D\colon \mathcal{J} \to \text{Scheme}$.
- A *localisation data* of $\mathcal{J}$ at an object $j \in J$ is for every $U \to D(j)$ a localised diagram $D_U\colon \mathcal{J} \to \text{Scheme}$ with $D_U(j) = U$.

Local properties of diagrams

- Encode a diagram of shape $\mathcal{J}$ as a functor $D\colon \mathcal{J} \to \text{Scheme}$.
- A *localisation data* of $\mathcal{J}$ at an object $j \in J$ is for every $U \to D(j)$ a localised diagram $D_U\colon \mathcal{J} \to \text{Scheme}$ with $D_U(j) = U$.
- Given a localisation data of $\mathcal{J}$ at $j \in J$, a property P of diagrams $\mathcal{J} \to \text{Scheme}$ is *local* at j, if for every diagram $\mathcal{J} \to \text{Scheme}$ and open cover $(U_i)_i$ of $D(j)$, P holds for D if and only if it holds for D_{U_i} for all i.

Local properties of diagrams

- Encode a diagram of shape $\mathcal{J}$ as a functor $D\colon \mathcal{J} \to \text{Scheme}$.
- A *localisation data* of $\mathcal{J}$ at an object $j \in J$ is for every $U \to D(j)$ a localised diagram $D_U\colon \mathcal{J} \to \text{Scheme}$ with $D_U(j) = U$.
- Given a localisation data of $\mathcal{J}$ at $j \in J$, a property P of diagrams $\mathcal{J} \to \text{Scheme}$ is *local* at j, if for every diagram $\mathcal{J} \to \text{Scheme}$ and open cover $(U_i)_i$ of $D(j)$, P holds for D if and only if it holds for D_{U_i} for all i.
- A metaprogram can construct the diagram $\mathcal{I} \to \text{Scheme}$ from a given concrete situation and synthesize the localisation data for $\mathcal{I}$.

Ongoing and future work

- Generalise `LocalAtTarget` etc., to other topologies beyond the Zariski topology (ongoing).

Ongoing and future work

- Generalise `LocalAtTarget` etc., to other topologies beyond the Zariski topology (ongoing).
- Develop algebraic cycles and divisors (ongoing).

Ongoing and future work

- Generalise `LocalAtTarget` etc., to other topologies beyond the Zariski topology (ongoing).
- Develop algebraic cycles and divisors (ongoing).
- Čech cohomology (ongoing).

Ongoing and future work

- Generalise `LocalAtTarget` etc., to other topologies beyond the Zariski topology (ongoing).
- Develop algebraic cycles and divisors (ongoing).
- Čech cohomology (ongoing).
- Toric varieties and group schemes (ongoing).

Ongoing and future work

- Generalise `LocalAtTarget` etc., to other topologies beyond the Zariski topology (ongoing).
- Develop algebraic cycles and divisors (ongoing).
- Čech cohomology (ongoing).
- Toric varieties and group schemes (ongoing).
- Quasi-coherent sheafs.

Ongoing and future work

- Generalise `LocalAtTarget` etc., to other topologies beyond the Zariski topology (ongoing).
- Develop algebraic cycles and divisors (ongoing).
- Čech cohomology (ongoing).
- Toric varieties and group schemes (ongoing).
- Quasi-coherent sheafs.
- Connect elliptic curves to schemes.

Ongoing and future work

- Generalise `LocalAtTarget` etc., to other topologies beyond the Zariski topology (ongoing).
- Develop algebraic cycles and divisors (ongoing).
- Čech cohomology (ongoing).
- Toric varieties and group schemes (ongoing).
- Quasi-coherent sheafs.
- Connect elliptic curves to schemes.
- (Elementary version of) Zariski-Main theorem.

Ongoing and future work

- Generalise `LocalAtTarget` etc., to other topologies beyond the Zariski topology (ongoing).
- Develop algebraic cycles and divisors (ongoing).
- Čech cohomology (ongoing).
- Toric varieties and group schemes (ongoing).
- Quasi-coherent sheafs.
- Connect elliptic curves to schemes.
- (Elementary version of) Zariski-Main theorem.
- Cohomology of quasi-coherent sheafs.