Schemes in Mathlib4
A Companion to `Scheme.lean`

1 Introduction

This document provides a natural language companion to the `Scheme.lean` file in Mathlib4. The file defines the fundamental concept of schemes in algebraic geometry, establishing schemes as locally ringed spaces that are locally isomorphic to affine schemes (spectra of commutative rings).

A scheme is the central object of study in modern algebraic geometry, generalizing both algebraic varieties and the spectrum of a ring. The definition captures the idea that a scheme should "look like" the spectrum of a commutative ring when examined locally.

2 Main Definitions

2.1 The Structure of a Scheme

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

Natural Language: A scheme is a locally ringed space with the additional property that every point has an open neighborhood that, when restricted to that neighborhood, is isomorphic (as a locally ringed space) to the spectrum of some commutative ring. This captures the fundamental idea that schemes are "locally affine."

2.2 Morphisms of Schemes

```
structure Hom (X Y : Scheme)
  extends toLRSHom' : X.toLocallyRingedSpace.Hom Y.toLocallyRingedSpace where
```

Natural Language: A morphism between schemes is simply a morphism between their underlying locally ringed spaces. This makes the category of schemes a full subcategory of locally ringed spaces.

2.3 The Category Structure

```
instance : Category Scheme where
  Hom := Hom
  id X := Hom.mk (𝟙 X.toLocallyRingedSpace)
  comp f g := Hom.mk (f.toLRSHom >> g.toLRSHom)
```

Natural Language: Schemes form a category where morphisms are as defined above, identity morphisms are induced from the identity morphisms of locally ringed spaces, and composition is induced from composition of locally ringed space morphisms.

3 Notation and Conventions

3.1 Preimage Notation

```
scoped[AlgebraicGeometry] notation3:90 f:91 " ⁻¹U " U:90 =>
  @Functor.obj (Scheme.Opens _) _ (Scheme.Opens _) _
    (Opens.map (f : Scheme.Hom _ _).base) U
```

Natural Language: The notation $f^{-1}U$ denotes the preimage of an open set U under a morphism f of schemes. This is the pullback of the open set along the underlying continuous map.

3.2 Global Sections Notation

```
scoped[AlgebraicGeometry] notation3 "Γ(" X ", " U ")" =>
  (PresheafedSpace.presheaf (SheafedSpace.toPresheafedSpace
    (LocallyRingedSpace.toSheafedSpace (Scheme.toLocallyRingedSpace X)))).obj
    (op (α := Scheme.Opens _) U)
```

Natural Language: The notation $\Gamma(X, U)$ denotes the sections of the structure sheaf of scheme X over the open set U. This gives us the "functions" (more precisely, ring elements) defined on the open set U.

4 Basic Properties

4.1 Continuity of Morphisms

```
lemma Hom.continuous {X Y : Scheme} (f : X.Hom Y) : Continuous f.base
```

Natural Language: The underlying map of any morphism of schemes is continuous. This is inherited from the fact that morphisms of locally ringed spaces have continuous underlying maps.

4.2 Specialization Preorder

```
instance {X : Scheme.{u}} : Preorder X := specializationPreorder X

lemma le_iff_specializes {X : Scheme.{u}} {a b : X} : a ≤ b ↔ b ⤳ a
```

Natural Language: Every scheme carries a natural preorder structure given by specialization: $a \leq b$ if and only if b specializes to a. This captures the geometric intuition that generic points are "larger" than their specializations.

5 Morphism Operations

5.1 Induced Maps on Sections

```
abbrev app (U : Y.Opens) : Γ(Y, U) → Γ(X, f ⁻¹U U) := f.c.app (op U)

abbrev appTop : Γ(Y, ⊤) → Γ(X, ⊤) := f.app ⊤
```

Natural Language: A morphism $f : X \to Y$ induces a map on sections: for any open set $U \subseteq Y$, we get a ring homomorphism $\Gamma(Y, U) \to \Gamma(X, f^{-1}U)$. The special case of global sections gives us $\Gamma(Y, \top) \to \Gamma(X, \top)$.

5.2 Naturality

```
lemma naturality (i : op U' → op U) :
    Y.presheaf.map i >> f.app U = f.app U' >> X.presheaf.map ((Opens.map f.base).
        map i.unop).op
```

Natural Language: The induced maps on sections are natural with respect to restriction maps. This means that restricting sections and then applying the induced map is the same as applying the induced map and then restricting.

5.3 Stalk Maps

```
def stalkMap (x : X) : Y.presheaf.stalk (f.base x) → X.presheaf.stalk x :=
  f.toLRSHom.stalkMap x
```

Natural Language: A morphism $f : X \to Y$ induces local ring homomorphisms between stalks: for each point $x \in X$, we get a map from the stalk at $f(x)$ in Y to the stalk at x in X.

6 The Spec Construction

6.1 Spec as a Scheme

```
def Spec (R : CommRingCat) : Scheme where
  toLocallyRingedSpace := Spec.toLocallyRingedSpace.obj (op R)
  local_affine x := ⟨⟨Set.univ, isOpen_univ⟩, x.2⟩, R, ⟨(Spec.
      toLocallyRingedSpace.obj (op R)).restrictTopIso⟩⟩
```

Natural Language: For any commutative ring R, we can construct a scheme $\text{Spec}(R)$ whose underlying locally ringed space is the prime spectrum of R with the structure sheaf. This scheme is "globally affine" in the sense that the entire space is isomorphic to a spectrum.

6.2 Functoriality

```
def Spec.map {R S : CommRingCat} (f : R → S) : Spec S → Spec R :=
  (Spec.toLocallyRingedSpace.map f.op).hom

theorem Spec.map_comp {R S T : CommRingCat} (f : R → S) (g : S → T) :
  Spec.map (f >> g) = Spec.map g >> Spec.map f
```

Natural Language: The Spec construction is functorial: a ring homomorphism $f : R \to S$ induces a scheme morphism $\text{Spec}(f) : \text{Spec}(S) \to \text{Spec}(R)$ (note the direction reversal). This makes Spec a contravariant functor from commutative rings to schemes.

7 Basic Opens

7.1 Definition

```
def basicOpen : X.Opens :=
  { carrier := { x | IsUnit (X.presheaf.germ U x hx f) }
    is_open' := ... }
```

Natural Language: For a section $f \in \Gamma(X, U)$, the basic open $D(f)$ consists of all points $x \in U$ where f becomes a unit in the stalk at x. This generalizes the classical notion of basic opens in affine schemes.

7.2 Properties

```
theorem basicOpen_mul : X.basicOpen (f * g) = X.basicOpen f      X.basicOpen g

theorem basicOpen_zero (U : X.Opens) : X.basicOpen (0 : Γ(X, U)) =

theorem basicOpen_one : X.basicOpen (1 : Γ(X, U)) = U
```

Natural Language: Basic opens behave well with respect to ring operations: the basic open of a product is the intersection of the basic opens, the basic open of zero is empty, and the basic open of the unit element is the entire open set.

8 Zero Loci

8.1 Definition and Properties

```
def zeroLocus {U : X.Opens} (s : Set Γ(X, U)) : Set X := X.toRingedSpace.
    zeroLocus s

lemma zeroLocus_isClosed {U : X.Opens} (s : Set Γ(X, U)) :
  IsClosed (X.zeroLocus s)
```

Natural Language: The zero locus $V(s)$ of a set of sections $s \subseteq \Gamma(X, U)$ is the set of points where all sections in s vanish. Zero loci are always closed subsets.

8.2 Relationship to Basic Opens

```
lemma codisjoint_zeroLocus {U : X.Opens}
  {f : Γ(X, U)} : Codisjoint (X.zeroLocus {f} : Set X) (X.basicOpen f : Set X)
```

Natural Language: The zero locus of a section and its basic open are codisjoint (their union covers the underlying open set). This expresses the fundamental duality between "where a function vanishes" and "where a function is invertible."

9 Global Sections Functor

9.1 Definition

```
def Γ : Scheme^op ⥤ CommRingCat :=
  Scheme.forgetToLocallyRingedSpace.op ⋙ LocallyRingedSpace.Γ
```

Natural Language: The global sections functor Γ assigns to each scheme its ring of global sections and to each morphism the induced ring homomorphism. This is a contravariant functor from schemes to commutative rings.

9.2 Spec- Identity

```
def SpecΓIdentity : Scheme.Spec.rightOp ⋙ Scheme.Γ ≅       _ :=

def ΓSpecIso : Γ(Spec R, ⊤) ≅ R := SpecΓIdentity.app R
```

Natural Language: There is a natural isomorphism between the global sections of Spec(R) and R itself. This is a fundamental relationship that will be part of the adjunction between Γ and Spec.

10 Forgetful Functors

10.1 To Locally Ringed Spaces

```
def forgetToLocallyRingedSpace : Scheme ⥤ LocallyRingedSpace where
  obj X := X.toLocallyRingedSpace
  map f := f.toLRSHom

instance : forgetToLocallyRingedSpace.Full
instance : forgetToLocallyRingedSpace.Faithful
```

Natural Language: There is a fully faithful forgetful functor from schemes to locally ringed spaces. This makes the category of schemes a full subcategory of locally ringed spaces.

10.2 To Topological Spaces

```
def forgetToTop : Scheme ⥤ TopCat :=
  forgetToLocallyRingedSpace ⋙ LocallyRingedSpace.forgetToTop
```

Natural Language: By composing forgetful functors, we can also view schemes as topological spaces, forgetting all the algebraic structure.

11 Special Cases and Examples

11.1 Empty Scheme

```
def empty : Scheme where
  toLocallyRingedSpace := LocallyRingedSpace.empty
  local_affine := False.elim

instance : EmptyCollection Scheme
instance : Inhabited Scheme
```

Natural Language: There is an empty scheme, which serves as the initial object in certain contexts. The category of schemes is also inhabited (has a canonical element).

11.2 Field Spectra

```
instance {K} [Field K] : Unique Spec(K) :=

lemma default_asIdeal {K} [Field K] : (default : Spec(K)).asIdeal =
```

Natural Language: The spectrum of a field is a single point, corresponding to the zero ideal (which is the unique maximal ideal in a field).