1. Introduction

This document provides a natural language companion to the `Limits.lean` file in Mathlib4. The file constructs various limits and colimits in the category of schemes, establishing fundamental categorical properties that are essential for the theory of schemes. These constructions generalize classical geometric operations like taking products, fiber products, and disjoint unions to the scheme-theoretic setting.

The existence of limits and colimits in the category of schemes is crucial for many geometric constructions and provides the categorical foundation for advanced topics in algebraic geometry. The constructions are built on the fundamental relationship between schemes and commutative rings through the Spec functor, which reverses the direction of morphisms and transforms colimits in rings to limits in schemes.

2. Terminal Objects

2.1. Spec $\mathbb{Z}$ as Terminal Object

```
noncomputable def specZIsTerminal : IsTerminal Spec(ℤ) :=
  @IsTerminal.isTerminalObj _ _ _ _ _ Scheme.Spec _ inferInstance
    (terminalOpOfInitial CommRingCat.zIsInitial)
```

Mathematical Significance: The scheme $\mathrm{Spec}(\mathbb{Z})$ is the terminal object in the category of schemes. This means that for every scheme X, there exists a unique morphism $X \to \mathrm{Spec}(\mathbb{Z})$.

Geometrically, this reflects the fundamental role of $\mathbb{Z}$ as the "initial ring" - every ring has a unique ring homomorphism from $\mathbb{Z}$ (determined by where 1 maps), and by the contravariant nature of the Spec functor, this becomes a morphism into every scheme from $\mathrm{Spec}(\mathbb{Z})$.

The terminal property captures the idea that $\mathrm{Spec}(\mathbb{Z})$ is the "universal base scheme" over which all schemes can be considered.

2.2. Finite Limits

```
instance : HasFiniteLimits Scheme :=
  hasFiniteLimits_of_hasTerminal_and_pullbacks
```

Mathematical Significance: The category of schemes has all finite limits. This follows from the general categorical theorem that a category with terminal objects and fiber products (pullbacks) automatically has all finite limits.

This result is fundamental because it ensures that many classical geometric constructions have scheme-theoretic generalizations:
- Products of schemes exist
- Fiber products (the scheme-theoretic intersection) exist
- Equalizers and other finite limit constructions are available

3. Initial Objects and Empty Schemes

3.1. The Empty Scheme

```
@[simps]
def Scheme.emptyTo (X : Scheme.{u}) : ∅ ⟶ X :=
  ⟨{  base := TopCat.ofHom ⟨fun x => PEmpty.elim x, by fun_prop⟩
      c := { app := fun _ => CommRingCat.punitIsTerminal.from _ } }, fun x =>
PEmpty.elim x⟩
```

Mathematical Significance: The empty scheme $\emptyset$ has a unique morphism to every scheme X. This is the scheme-theoretic version of the empty set, and its morphisms are defined by the fact that there are no points to map, making the morphism trivially well-defined.

3.2. Initial Property

```
def emptyIsInitial : IsInitial (∅ : Scheme.{u}) :=
  IsInitial.ofUnique _
```

Mathematical Significance: The empty scheme is the initial object in the category of schemes. This means that for every scheme X, there exists a unique morphism $\emptyset \to X$.

The initial property reflects the fact that the empty scheme has no geometric content and can be “included” into any other scheme in a canonical way. This is the scheme-theoretic generalization of the fact that the empty set can be included into any set.

3.3. Characterization of Empty Schemes

```
noncomputable def isInitialOfIsEmpty {X : Scheme} [IsEmpty X] : IsInitial X :=
  emptyIsInitial.ofIso (asIso <| emptyIsInitial.to _)
```

Mathematical Significance: Any scheme with empty underlying topological space is initial. This shows that the notion of “emptiness” in scheme theory is captured exactly by being an initial object.

This characterization is important because it connects the topological notion of being empty with the categorical notion of being initial.

4. Coproducts and Disjoint Unions

4.1. Sigma Construction

```
noncomputable
def sigmaMk : (Σ i, f i) ≃ₜ (∐ f :) :=
  TopCat.homeoOfIso ((colimit.isoColimitCocone ⟨_, TopCat.sigmaCofanIsColimit
_⟩).symm ≪≫
    (PreservesCoproduct.iso Scheme.forgetToTop f).symm)
```

Mathematical Significance: The underlying topological space of the coproduct of schemes $\sqcup_i X_i$ is homeomorphic to the disjoint union $\bigsqcup_i X_i$ as topological spaces. This establishes that the scheme-theoretic coproduct has the expected topological behavior.

4.2. Disjoint Images

```
lemma disjoint_opensRange_sigmaι (i j : ι) (h : i ≠ j) :
    Disjoint (Sigma.ι f i).opensRange (Sigma.ι f j).opensRange := by
  intro U hU hU' x hx
  obtain ⟨x, rfl⟩ := hU hx
  obtain ⟨y, hy⟩ := hU' hx
  obtain ⟨rfl⟩ := (sigmaι_eq_iff _ _ _ _ _).mp hy
  cases h rfl
```

Mathematical Significance: The images of different components in the coproduct are disjoint as open subsets. This formalizes the intuitive idea that the coproduct is indeed a “disjoint union” - the different components don’t overlap.

This disjointness property is crucial for many applications and ensures that the coproduct behaves like a genuine disjoint union of geometric objects.

4.3. Open Immersions in Coproducts

```
open scoped Function in
lemma isOpenImmersion_sigmaDesc [Small.{u} σ]
    {X : Scheme.{u}} (α : ∀ i, g i ⟶ X) [∀ i, IsOpenImmersion (α i)]
    (hα : Pairwise (Disjoint on (Set.range <| α · |>.base))) :
    IsOpenImmersion (Sigma.desc α) := by
```

Mathematical Significance: If we have morphisms from components of a coproduct to a scheme X that are open immersions with pairwise disjoint images, then the induced morphism from the coproduct to X is also an open immersion.

This result is fundamental for understanding when a scheme can be decomposed as a disjoint union of open subschemes.

5. Binary Coproducts

5.1. Coproduct Structure

```
noncomputable
def coprodIsoSigma : X ⨿ Y ≅ ∐ fun i : ULift.{u} WalkingPair ↦ i.1.casesOn X Y :=
  Sigma.whiskerEquiv Equiv.ulift.symm (fun _ ↦ by exact Iso.refl _)
```

Mathematical Significance: The binary coproduct $X \sqcup Y$ is isomorphic to the general coproduct indexed by a two-element set. This shows that binary and indexed coproducts are essentially the same construction.

5.2. Open Immersion Property

```
instance : IsOpenImmersion (coprod.inl : X ⟶ X ⨿ Y) := by
  rw [← ι_left_coprodIsoSigma_inv]; infer_instance

instance : IsOpenImmersion (coprod.inr : Y ⟶ X ⨿ Y) := by
  rw [← ι_right_coprodIsoSigma_inv]; infer_instance
```

Mathematical Significance: The canonical inclusions of X and Y into their coproduct $X \sqcup Y$ are open immersions. This means that both X and Y appear as open subschemes of their coproduct, which is the expected geometric behavior.

5.3. Complementary Structure

```
lemma isCompl_range_inl_inr :
    IsCompl (Set.range (coprod.inl : X ⟶ X ⨿ Y).base)
      (Set.range (coprod.inr : Y ⟶ X ⨿ Y).base) :=
  ((TopCat.binaryCofan_isColimit_iff _).mp
    ⟨mapIsColimitOfPreservesOfIsColimit Scheme.forgetToTop.{u} _ _ (coprodIsCoprod X
Y)⟩).2.2
```

Mathematical Significance: The images of X and Y in their coproduct $X \sqcup Y$ form complementary subsets - they are disjoint and their union is the entire space. This confirms that the coproduct is indeed a disjoint union where every point comes from exactly one of the two components.

6. Affine Coproducts

6.1. Coproduct of Spectra

```
noncomputable
def coprodSpec : Spec(R) ⨿ Spec(S) ⟶ Spec(R × S) :=
  coprod.desc (Spec.map (CommRingCat.ofHom <| RingHom.fst _ _))
    (Spec.map (CommRingCat.ofHom <| RingHom.snd _ _))
```

Mathematical Significance: The coproduct of two affine schemes $\mathrm{Spec}(R) \sqcup \mathrm{Spec}(S)$ is canonically isomorphic to $\mathrm{Spec}(R \times S)$. This establishes the fundamental relationship between coproducts in schemes and products in rings.

The construction uses the fact that the Spec functor reverses the direction of morphisms, so the product ring $R \times S$ with its projection maps corresponds to the coproduct of the corresponding spectra.

6.2. Isomorphism Property

```
instance : IsIso (coprodSpec R S) := by
  rw [isIso_iff_stalk_iso]
  refine ⟨?_, isIso_stalkMap_coprodSpec R S⟩
```

Mathematical Significance: The canonical morphism $\mathrm{Spec}(R) \sqcup \mathrm{Spec}(S) \to \mathrm{Spec}(R \times S)$ is an isomorphism. This can be verified by checking that it induces isomorphisms on all stalks and is a homeomorphism on the underlying topological spaces.

This result establishes the fundamental correspondence between coproducts of affine schemes and products of rings, which is one of the key examples of how the Spec functor transforms colimits in rings into limits in schemes.

7. Preservation Properties

7.1. Spec Preserves Finite Coproducts

```
instance : PreservesColimitsOfShape (Discrete WalkingPair) Scheme.Spec.{u} :=
  ⟨fun {_} ↦
    have (X Y : CommRingCat.{u}ᵒᵖ) := PreservesColimitPair.of_iso_coprod_comparison
Scheme.Spec X Y
    preservesColimit_of_iso_diagram _ (diagramIsoPair _).symm⟩
```

Mathematical Significance: The Spec functor preserves binary coproducts. This means that $\mathrm{Spec}(R \times S) \simeq \mathrm{Spec}(R) \sqcup \mathrm{Spec}(S)$, establishing the fundamental duality between products in rings and coproducts in schemes.

7.2. Finite Coproduct Preservation

```
instance {J : Type*} [Finite J] : PreservesColimitsOfShape (Discrete J) Scheme.Spec.
{u} :=
  preservesFiniteCoproductsOfPreservesBinaryAndInitial _ _
```

Mathematical Significance: The Spec functor preserves all finite coproducts. This extends the binary case to arbitrary finite collections, showing that $\mathrm{Spec}\left(\prod_i R_i\right) \simeq \bigsqcup_i \mathrm{Spec}(R_i)$ for finite index sets.

This preservation property is fundamental for understanding how algebraic constructions (products of rings) translate to geometric constructions (coproducts of schemes).

8. Universal Properties and Characterizations

8.1. Universal Property of Coproducts

```
lemma nonempty_isColimit_cofanMk_of [Small.{u} σ]
    {X : σ → Scheme.{u}} {S : Scheme.{u}} (f : ∀ i, X i ⟶ S) [∀ i, IsOpenImmersion (f
i)]
    (hcov : ⨆ i, (f i).opensRange = ⊤) (hdisj : Pairwise (Disjoint on (f · |
>.opensRange))) :
    Nonempty (IsColimit <| Cofan.mk S f) := by
```

Mathematical Significance: If a scheme S can be written as a disjoint union of open subschemes that are isomorphic to given schemes X_i, then S is the coproduct of the X_i. This provides a practical criterion for recognizing when a scheme is a coproduct.

This characterization is essential for applications because it allows us to identify coproducts through geometric properties (disjoint open covers) rather than through abstract categorical constructions.

8.2. Binary Case

```
lemma nonempty_isColimit_binaryCofanMk_of_isCompl {X Y S : Scheme.{u}}
    (f : X ⟶ S) (g : Y ⟶ S) [IsOpenImmersion f] [IsOpenImmersion g]
    (hf : IsCompl f.opensRange g.opensRange) :
    Nonempty (IsColimit <| BinaryCofan.mk f g) := by
```

Mathematical Significance: If a scheme S is the disjoint union of two open subschemes isomorphic to X and Y respectively (i.e., their images are complementary), then S is the coproduct $X \sqcup Y$.

This provides the binary version of the general characterization and is particularly useful for understanding when schemes decompose as binary coproducts.

9. Affine Coproduct Examples

9.1. Infinite Coproducts

```
noncomputable
def sigmaSpec (R : ι → CommRingCat) : (∐ fun i ↦ Spec (R i)) ⟶ Spec(Π i, R i) :=
  Sigma.desc (fun i ↦ Spec.map (CommRingCat.ofHom (Pi.evalRingHom _ i)))
```

Mathematical Significance: For an arbitrary family of rings $(R_i)_{i \in I}$, there is a canonical morphism from the coproduct of their spectra to the spectrum of their product. When the index set is finite, this morphism is an isomorphism.

9.2. Open Immersion Structure

```
instance (R : ι → CommRingCat.{u}) : IsOpenImmersion (sigmaSpec R) := by
  classical
  apply isOpenImmersion_sigmaDesc
  intro ix iy h
  refine Set.disjoint_iff_forall_ne.mpr ?_
```

Mathematical Significance: The canonical morphism from the coproduct of spectra to the spectrum of the product is always an open immersion, even when the index set is infinite. This means that $\coprod_i \operatorname{Spec}(R_i)$ always embeds as an open subscheme of $\operatorname{Spec}\left(\prod_i R_i\right)$.

When the index set is finite, this open immersion is actually an isomorphism, but for infinite index sets, the morphism is typically not surjective.

10. Geometric Significance

The limit and colimit constructions developed in this file provide the categorical foundation for many geometric operations:

10.1. Classical Interpretations

- **Terminal Object**: Spec($\mathbb{Z}$) serves as the universal base scheme, generalizing the notion of the “ground field” in classical algebraic geometry.
- **Empty Scheme**: The initial object provides the scheme-theoretic version of the empty set, essential for describing “nowhere defined” constructions.
- **Disjoint Unions**: Coproducts generalize the classical notion of taking disjoint unions of varieties, providing a systematic way to combine geometric objects.
- **Products vs. Coproducts**: The reversal of products and coproducts by the Spec functor reflects the contravariant relationship between algebra and geometry.

10.2. Applications

These constructions enable many advanced topics in algebraic geometry:

- **Fiber Products**: Combined with the terminal object, finite limits provide fiber products, essential for intersection theory.
- **Base Change**: The limit constructions provide the foundation for base change operations.
- **Moduli Problems**: Coproducts and limits are essential for constructing moduli spaces and parameter spaces.
- **Descent Theory**: The categorical properties established here are fundamental for descent theory and stack constructions.

The systematic development of limits and colimits thus provides both the theoretical foundation and practical tools needed for advanced algebraic geometry in the Mathlib4 framework.