Lean 4 Code: PointsPi

Mathlib4

September 6, 2025

1 Source Code

The following is the Lean 4 source code from `PointsPi.lean`:

```
/-
Copyright (c) 2024 Andrew Yang. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
Authors: Andrew Yang
-/
import Mathlib.AlgebraicGeometry.Morphisms.Immersion

/-!

# `   R_i`-Points of Schemes

We show that the canonical map `X(   R_i) ⟶    X(R_i)` (`AlgebraicGeometry.pointsPi`)
is injective and surjective under various assumptions

-/

open CategoryTheory Limits PrimeSpectrum

namespace AlgebraicGeometry

universe u v

variable {ι : Type u} (R : ι → CommRingCat.{u})

lemma Ideal.span_eq_top_of_span_image_evalRingHom
    {ι} {R : ι → Type*} [∀ i, CommRing (R i)] (s : Set (   i, R i))
    (hs : s.Finite) (hs' : ∀ i, Ideal.span (Pi.evalRingHom (R ·) i '' s) = ⊤) :
    Ideal.span s = ⊤ := by
  simp only [Ideal.eq_top_iff_one, ← Subtype.range_val (s := s), ← Set.range_comp,
    Finsupp.mem_ideal_span_range_iff_exists_finsupp] at hs'
  choose f hf using hs'
  have : Fintype s := hs.fintype
  refine ⟨Finsupp.equivFunOnFinite.symm fun i x     f x i, ?_⟩
  ext i
  simpa [Finsupp.sum_fintype] using hf i

lemma eq_top_of_sigmaSpec_subset_of_isCompact
    (U : Spec(   i, R i).Opens) (V : Set Spec(   i, R i))
    (hV :    (sigmaSpec R).opensRange ⊆ V)
    (hV' : IsCompact (X := Spec(   i, R i)) V)
    (hVU : V ⊆ U) : U = ⊤ := by
  obtain ⟨s, hs⟩ := (PrimeSpectrum.isOpen_iff _).mp U.2
  obtain ⟨t, hts, ht, ht'⟩ : ∃ t ⊆ s, t.Finite ∧ V ⊆     i ∈ t, (basicOpen i).1
      := by
    obtain ⟨t, ht⟩ := hV'.elim_finite_subcover
      (fun i : s     (basicOpen i.1).1) (fun _     (basicOpen _).2)
      (by simpa [← Set.compl_iInter, ← zeroLocus_iUnion₂ (κ := (· ∈ s)), ← hs])
    exact ⟨t.map (Function.Embedding.subtype _), by simp, Finset.finite_toSet _,
        by simpa using ht⟩
  replace ht' : V ⊆ (zeroLocus t)    := by
    simpa [← Set.compl_iInter, ← zeroLocus_iUnion₂ (κ := (· ∈ t))] using ht'
  have (i : _) : Ideal.span (Pi.evalRingHom (R ·) i '' t) = ⊤ := by
    rw [← zeroLocus_empty_iff_eq_top, zeroLocus_span, ←
        preimage_comap_zeroLocus,
      ← Set.compl_univ_iff, ← Set.preimage_compl, Set.preimage_eq_univ_iff]
    trans (Sigma.ι _ i     sigmaSpec R).opensRange.1
    · simp; rfl
    · rw [Scheme.Hom.opensRange_comp]
      exact (Set.image_subset_range _ _).trans (hV.trans ht')
  have : Ideal.span s = ⊤ := top_le_iff.mp
    ((Ideal.span_eq_top_of_span_image_evalRingHom _ ht this).ge.trans
        (Ideal.span_mono hts))
  simpa [← zeroLocus_span s, zeroLocus_empty_iff_eq_top.mpr this] using hs

lemma eq_bot_of_comp_quotientMk_eq_sigmaSpec (I : Ideal (   i, R i))
    (f : (    fun i     Spec (R i)) ⟶ Spec((   i, R i)     I))
    (hf : f     Spec.map (CommRingCat.ofHom (Ideal.Quotient.mk I)) = sigmaSpec R)
        :
    I = ⊥ := by
  refine le_bot_iff.mp fun x hx     ?_
  ext i
  simpa [← Category.assoc, Ideal.Quotient.eq_zero_iff_mem.mpr hx] using
    congr((Spec.preimage (Sigma.ι (Spec <| R ·) i     $hf)).hom x).symm

/-- If `V` is a locally closed subscheme of `Spec (   Rᵢ)` containing `    Spec
    Rᵢ`, then
`V = Spec (   Rᵢ)`. -/
lemma isIso_of_comp_eq_sigmaSpec {V : Scheme}
    (f : (    fun i     Spec (R i)) ⟶ V) (g : V ⟶ Spec(   i, R i))
    [IsImmersion g] [CompactSpace V]
    (hU' : f     g = sigmaSpec R) : IsIso g := by
  have : g.coborderRange = ⊤ := by
    apply eq_top_of_sigmaSpec_subset_of_isCompact (hVU := subset_coborder)
    · simpa only [← hU'] using Set.range_comp_subset_range f.base g.base
    · exact isCompact_range g.base.hom.2
  have : IsClosedImmersion g := by
    have : IsIso g.coborderRange.ι := by rw [this, ← Scheme.topIso_hom];
        infer_instance
    rw [← g.liftCoborder_ι]
    infer_instance
  obtain ⟨I, e, rfl⟩ := IsClosedImmersion.Spec_iff.mp this
  obtain rfl := eq_bot_of_comp_quotientMk_eq_sigmaSpec R I (f     e.hom) (by rwa
      [Category.assoc])
  convert_to IsIso (e.hom     Spec.map (RingEquiv.quotientBot
      _).toCommRingCatIso.inv)
  infer_instance

variable (X : Scheme)

/-- The canonical map `X(   Rᵢ) ⟶    X(Rᵢ)`.
```

```
This is injective if ‘X‘ is quasi-separated, surjective if ‘X‘ is affine,
or if ‘X‘ is compact and each ‘Rᵢ‘ is local. -/
noncomputable
def pointsPi : (Spec(   i, R i) ⟶ X) →     i, Spec (R i) ⟶ X :=
  fun f i     Spec.map (CommRingCat.ofHom (Pi.evalRingHom (R ·) i))     f

lemma pointsPi_injective [QuasiSeparatedSpace X] : Function.Injective (pointsPi R X) := by
  rintro f g e
  have := isIso_of_comp_eq_sigmaSpec R (V := equalizer f g)
    (equalizer.lift (sigmaSpec R) (by ext1 i; simpa using congr_fun e i))
    (equalizer.ι f g) (by simp)
  rw [← cancel_epi (equalizer.ι f g), equalizer.condition]

lemma pointsPi_surjective_of_isAffine [IsAffine X] : Function.Surjective (pointsPi R X) := by
  rintro f
  refine ⟨Spec.map (CommRingCat.ofHom
    (Pi.ringHom fun i     (Spec.preimage (f i     X.isoSpec.hom)).1))     X.isoSpec.inv, ?_⟩
  ext i : 1
  simp only [pointsPi, ← Spec.map_comp_assoc, Iso.comp_inv_eq]
  exact Spec.map_preimage _

lemma pointsPi_surjective [CompactSpace X] [∀ i, IsLocalRing (R i)] :
    Function.Surjective (pointsPi R X) := by
  intro f
  let 𝒰 : X.OpenCover := X.affineCover.finiteSubcover
  have (i : _) : IsAffine (𝒰.obj i) := isAffine_Spec _
  have (i : _) : ∃ j, Set.range (f i).base ⊆ (𝒰.map j).opensRange := by
    refine ⟨𝒰.f ((f i).base (IsLocalRing.closedPoint (R i))), ?_⟩
    rintro _ ⟨x, rfl⟩
    exact ((IsLocalRing.specializes_closedPoint x).map (f i).base.hom.2).mem_open
      (𝒰.map _).opensRange.2 (𝒰.covers _)
  choose j hj using this
  have (j₀ : _) := pointsPi_surjective_of_isAffine (ι := { i // j i = j₀ }) (R ·) (𝒰.obj j₀)
    (fun i     IsOpenImmersion.lift (𝒰.map j₀) (f i.1) (by rcases i with ⟨i, rfl⟩; exact hj i))
  choose g hg using this
  simp_rw [funext_iff, pointsPi] at hg
  let R' (j₀) := CommRingCat.of (   i : { i // j i = j₀ }, R i)
  let e : (   i, R i) ≃+*     j₀, R' j₀ :=
  { toFun f _ i := f i
    invFun f i := f _ ⟨i, rfl⟩
    right_inv _ := funext₂ fun j₀ i     by rcases i with ⟨i, rfl⟩; rfl
    map_mul' _ _ := rfl
    map_add' _ _ := rfl }
  refine ⟨Spec.map (CommRingCat.ofHom e.symm.toRingHom)     inv (sigmaSpec R')
    Sigma.desc fun j₀     g j₀     𝒰.map j₀, ?_⟩
  ext i : 1
  have : (Pi.evalRingHom (R ·) i).comp e.symm.toRingHom =
    (Pi.evalRingHom _ ⟨i, rfl⟩).comp (Pi.evalRingHom (R' ·) (j i)) := rfl
  rw [pointsPi, ← Spec.map_comp_assoc, ← CommRingCat.ofHom_comp, this, CommRingCat.ofHom_comp,
    Spec.map_comp_assoc, ← ι_sigmaSpec R', Category.assoc, IsIso.hom_inv_id_assoc,
    Sigma.ι_desc, ← Category.assoc, hg, IsOpenImmersion.lift_fac]

```

```
end AlgebraicGeometry
```

Listing 1: PointsPi.lean