# Lean 4 Code: ValuativeCriterion

Mathlib4

September 6, 2025

## 1 Source Code

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

```
/-
Copyright (c) 2024 Andrew Yang, Qi Ge, Christian Merten. All rights reserved.
Released under Apache 2.0 license as described in the file LICENSE.
Authors: Andrew Yang, Qi Ge, Christian Merten
-/
import Mathlib.AlgebraicGeometry.Morphisms.Immersion
import Mathlib.AlgebraicGeometry.Morphisms.Proper
import Mathlib.RingTheory.RingHom.Injective
import Mathlib.RingTheory.Valuation.LocalSubring

/-!
# Valuative criterion

## Main results

- `AlgebraicGeometry.UniversallyClosed.eq_valuativeCriterion`:
  A morphism is universally closed if and only if
  it is quasi-compact and satisfies the existence part of the valuative criterion.
- `AlgebraicGeometry.IsSeparated.eq_valuativeCriterion`:
  A morphism is separated if and only if
  it is quasi-separated and satisfies the uniqueness part of the valuative
      criterion.
- `AlgebraicGeometry.IsProper.eq_valuativeCriterion`:
  A morphism is proper if and only if
  it is qcqs and of fintite type and satisfies the valuative criterion.

## Future projects
Show that it suffices to check discrete valuation rings when the base is
    Noetherian.

-/

open CategoryTheory CategoryTheory.Limits

namespace AlgebraicGeometry

universe u

/--
A valuative commutative square over a morphism `f : X ⟶ Y` is a square
```
Spec K ⟶ Y
  |        |
```
```

```
Spec R ⟶ X
```
where ‘R‘ is a valuation ring, and ‘K‘ is its ring of fractions.

We are interested in finding lifts ‘Spec R ⟶ Y‘ of this diagram.
-/
structure ValuativeCommSq {X Y : Scheme.{u}} (f : X ⟶ Y) where
  /-- The valuation ring of a valuative commutative square. -/
  R : Type u
  [commRing : CommRing R]
  [domain : IsDomain R]
  [valuationRing : ValuationRing R]
  /-- The field of fractions of a valuative commutative square. -/
  K : Type u
  [field : Field K]
  [algebra : Algebra R K]
  [isFractionRing : IsFractionRing R K]
  /-- The top map in a valuative commutative map. -/
  (i₁ : Spec(K) ⟶ X)
  /-- The bottom map in a valuative commutative map. -/
  (i₂ : Spec(R) ⟶ Y)
  (commSq : CommSq i₁ (Spec.map (CommRingCat.ofHom (algebraMap R K))) f i₂)

namespace ValuativeCommSq

attribute [instance] commRing domain valuationRing field algebra isFractionRing

end ValuativeCommSq

/-- A morphism ‘f : X ⟶ Y‘ satisfies the existence part of the valuative criterion if
every valuative commutative square over ‘f‘ has (at least) a lift. -/
def ValuativeCriterion.Existence : MorphismProperty Scheme :=
  fun _ _ f     ∀ S : ValuativeCommSq f, S.commSq.HasLift

/-- A morphism ‘f : X ⟶ Y‘ satisfies the uniqueness part of the valuative criterion if
every valuative commutative square over ‘f‘ has at most one lift. -/
def ValuativeCriterion.Uniqueness : MorphismProperty Scheme :=
  fun _ _ f     ∀ S : ValuativeCommSq f, Subsingleton S.commSq.LiftStruct

/-- A morphism ‘f : X ⟶ Y‘ satisfies the valuative criterion if
every valuative commutative square over ‘f‘ has a unique lift. -/
def ValuativeCriterion : MorphismProperty Scheme :=
  fun _ _ f     ∀ S : ValuativeCommSq f, Nonempty (Unique (S.commSq.LiftStruct))

variable {X Y : Scheme.{u}} (f : X ⟶ Y)

lemma ValuativeCriterion.iff {f : X ⟶ Y} :
    ValuativeCriterion f ↔ Existence f ∧ Uniqueness f := by
  change (∀ _, _) ↔ (∀ _, _) ∧ (∀ _, _)
  simp_rw [← forall_and, unique_iff_subsingleton_and_nonempty, and_comm, CommSq.HasLift.iff]

lemma ValuativeCriterion.eq :
    ValuativeCriterion = Existence     Uniqueness := by
  ext X Y f
  exact iff
```
```

```lean

lemma ValuativeCriterion.existence {f : X ⟶ Y} (h : ValuativeCriterion f) :
    ValuativeCriterion.Existence f := (iff.mp h).1

lemma ValuativeCriterion.uniqueness {f : X ⟶ Y} (h : ValuativeCriterion f) :
    ValuativeCriterion.Uniqueness f := (iff.mp h).2

namespace ValuativeCriterion.Existence

open IsLocalRing

@[stacks 01KE]
lemma specializingMap (H : ValuativeCriterion.Existence f) :
    SpecializingMap f.base := by
  intro x' y h
  let stalk_y_to_residue_x' : Y.presheaf.stalk y ⟶ X.residueField x' :=
    Y.presheaf.stalkSpecializes h     f.stalkMap x'     X.residue x'
  obtain ⟨A, hA, hA_local⟩ := exists_factor_valuationRing
      stalk_y_to_residue_x'.hom
  let stalk_y_to_A : Y.presheaf.stalk y ⟶ .of A :=
    CommRingCat.ofHom (stalk_y_to_residue_x'.hom.codRestrict _ hA)
  have w : X.fromSpecResidueField x'     f =
      Spec.map (CommRingCat.ofHom (algebraMap A (X.residueField x')))
        Spec.map stalk_y_to_A     Y.fromSpecStalk y := by
    rw [Scheme.fromSpecResidueField, Category.assoc, ←
        Scheme.Spec_map_stalkMap_fromSpecStalk,
      ← Scheme.Spec_map_stalkSpecializes_fromSpecStalk h]
    simp_rw [← Spec.map_comp_assoc]
    rfl
  obtain ⟨l, hl₁, hl₂⟩ := (H { R := A, K := X.residueField x', commSq := ⟨w⟩, ..
      }).exists_lift
  dsimp only at hl₁ hl₂
  refine ⟨l.base (closedPoint A), ?_, ?_⟩
  · simp_rw [← Scheme.fromSpecResidueField_apply x' (closedPoint (X.residueField
      x')), ← hl₁]
    exact (specializes_closedPoint _).map l.base.hom.2
  · rw [← Scheme.comp_base_apply, hl₂]
    simp only [Scheme.comp_coeBase, TopCat.coe_comp, Function.comp_apply]
    have : (Spec.map stalk_y_to_A).base (closedPoint A) = closedPoint
        (Y.presheaf.stalk y) :=
      comap_closedPoint (S := A) (stalk_y_to_residue_x'.hom.codRestrict
          A.toSubring hA)
    rw [this, Y.fromSpecStalk_closedPoint]

instance {R S : CommRingCat} (e : R ≅ S) : IsLocalHom e.hom.hom :=
  isLocalHom_of_isIso _

lemma of_specializingMap (H : (topologically @SpecializingMap).universally f) :
    ValuativeCriterion.Existence f := by
  rintro ⟨R, K, i₁, i₂, ⟨w⟩⟩
  haveI : IsDomain (CommRingCat.of R) :=    _
  haveI : ValuationRing (CommRingCat.of R) :=    _
  letI : Field (CommRingCat.of K) :=    _
  replace H := H (pullback.snd i₂ f) i₂ (pullback.fst i₂ f) (.of_hasPullback i₂ f)
  let lft := pullback.lift (Spec.map (CommRingCat.ofHom (algebraMap R K))) i₁
      w.symm
  obtain ⟨x, h₁, h₂⟩ := @H (lft.base (closedPoint _)) _ (specializes_closedPoint
      (R := R) _)
  let e : CommRingCat.of R ≅ Spec(R).presheaf.stalk ((pullback.fst i₂ f).base x)
```

```
      :=
    (stalkClosedPointIso (.of R)).symm
      Spec(R).presheaf.stalkCongr (.of_eq h₂.symm)
  let α := e.hom     (pullback.fst i₂ f).stalkMap x
  have : IsLocalHom e.hom.hom := isLocalHom_of_isIso e.hom
  have : IsLocalHom α.hom := inferInstanceAs
    (IsLocalHom (((pullback.fst i₂ f).stalkMap x).hom.comp e.hom.hom))
  let β := (pullback i₂ f).presheaf.stalkSpecializes h₁
      Scheme.stalkClosedPointTo lft
  have hαβ : α     β = CommRingCat.ofHom (algebraMap R K) := by
    simp only [CommRingCat.coe_of, Iso.trans_hom, Iso.symm_hom,
        TopCat.Presheaf.stalkCongr_hom,
      Category.assoc, α, e, β, stalkClosedPointIso_inv, StructureSheaf.toStalk]
    change (Scheme.ΓSpecIso (.of R)).inv     Spec(R).presheaf.germ _ _ _     _ = _
    simp only [TopCat.Presheaf.germ_stalkSpecializes_assoc,
        Scheme.stalkMap_germ_assoc]
    -- ‘map_top‘ introduces defeq problems, according to ‘check_compositions‘.
    -- This is probably the cause of the ‘erw‘ needed below.
    simp only [TopologicalSpace.Opens.map_top]
    rw [Scheme.germ_stalkClosedPointTo lft ⊤ trivial]
    erw [← Scheme.comp_app_assoc lft (pullback.fst i₂ f)]
    rw [pullback.lift_fst]
    simp
  have hbij := (bijective_rangeRestrict_comp_of_valuationRing (R := R) (K := K) α
      .hom β.hom
    (CommRingCat.hom_ext_iff.mp hαβ))
  let φ : (pullback i₂ f).presheaf.stalk x ⟶ CommRingCat.of R :=
      CommRingCat.ofHom <|
    (RingEquiv.ofBijective _ hbij).symm.toRingHom.comp β.hom.rangeRestrict
  have hαφ : α     φ =     _ := by ext x; exact (RingEquiv.ofBijective _
      hbij).symm_apply_apply x
  have hαφ' : (pullback.fst i₂ f).stalkMap x     φ = e.inv := by
    rw [← cancel_epi e.hom, ← Category.assoc, hαφ, e.hom_inv_id]
  have hφβ : φ     CommRingCat.ofHom (algebraMap R K) = β :=
    hαβ ▷ CommRingCat.hom_ext (RingHom.ext fun x     congr_arg Subtype.val
      ((RingEquiv.ofBijective _ hbij).apply_symm_apply (β.hom.rangeRestrict x)))
  refine ⟨⟨⟨Spec.map ((pullback.snd i₂ f).stalkMap x     φ)     X.fromSpecStalk
      _, ?_, ?_⟩⟩⟩
  · simp only [← Spec.map_comp_assoc, Category.assoc, hφβ]
    simp only [Spec.map_comp, Category.assoc,
        Scheme.Spec_map_stalkMap_fromSpecStalk,
      Scheme.Spec_map_stalkSpecializes_fromSpecStalk_assoc, β]
    -- This next line only fires as ‘rw‘, not ‘simp‘:
    rw [Scheme.Spec_stalkClosedPointTo_fromSpecStalk_assoc]
    simp [lft]
  · simp only [Spec.map_comp, Category.assoc,
      Scheme.Spec_map_stalkMap_fromSpecStalk,
      ← pullback.condition]
    rw [← Scheme.Spec_map_stalkMap_fromSpecStalk_assoc, ← Spec.map_comp_assoc,
        hαφ']
    simp only [Iso.trans_inv, TopCat.Presheaf.stalkCongr_inv, Iso.symm_inv,
        Spec.map_comp,
      Category.assoc, Scheme.Spec_map_stalkSpecializes_fromSpecStalk_assoc, e]
    rw [← Spec_stalkClosedPointIso, ← Spec.map_comp_assoc,
      Iso.inv_hom_id, Spec.map_id, Category.id_comp]

instance stableUnderBaseChange :
    ValuativeCriterion.Existence.IsStableUnderBaseChange := by
  constructor
  intro Y' X X' Y  Y'_to_Y f X'_to_X f' hP hf commSq
  let commSq' : ValuativeCommSq f :=
  { R := commSq.R
    K := commSq.K
    i₁ := commSq.i₁     X'_to_X
    i₂ := commSq.i₂     Y'_to_Y
    commSq := ⟨by simp only [Category.assoc, hP.w, reassoc_of% commSq.commSq.w]⟩ }
  obtain ⟨l₀, hl₁, hl₂⟩ := (hf commSq').exists_lift
  refine ⟨⟨⟨hP.lift l₀ commSq.i₂ (by simp_all only [commSq']), ?_, hP.lift_snd _
      _ _⟩⟩⟩
  apply hP.hom_ext
  · simpa
  · simp only [Category.assoc]
    rw [hP.lift_snd]
    rw [commSq.commSq.w]

@[stacks 01KE]
protected lemma eq :
    ValuativeCriterion.Existence = (topologically @SpecializingMap).universally
        := by
  ext
  constructor
  · intro _
    apply MorphismProperty.universally_mono
    · apply specializingMap
    · rwa [MorphismProperty.IsStableUnderBaseChange.universally_eq]
  · apply of_specializingMap

end ValuativeCriterion.Existence

/-- The **valuative criterion** for universally closed morphisms. -/
@[stacks 01KF]
lemma UniversallyClosed.eq_valuativeCriterion :
    @UniversallyClosed = ValuativeCriterion.Existence     @QuasiCompact := by
  rw [universallyClosed_eq_universallySpecializing,
      ValuativeCriterion.Existence.eq]

/-- The **valuative criterion** for universally closed morphisms. -/
@[stacks 01KF]
lemma UniversallyClosed.of_valuativeCriterion [QuasiCompact f]
    (hf : ValuativeCriterion.Existence f) : UniversallyClosed f := by
  rw [eq_valuativeCriterion]
  exact ⟨hf,    _   ⟩

section Uniqueness

/-- The **valuative criterion** for separated morphisms. -/
@[stacks 01L0]
lemma IsSeparated.of_valuativeCriterion [QuasiSeparated f]
    (hf : ValuativeCriterion.Uniqueness f) : IsSeparated f where
  diagonal_isClosedImmersion := by
    suffices h : ValuativeCriterion.Existence (pullback.diagonal f) by
      have : QuasiCompact (pullback.diagonal f) :=
        AlgebraicGeometry.QuasiSeparated.diagonalQuasiCompact
      apply IsClosedImmersion.of_isPreimmersion
      apply IsClosedMap.isClosed_range
      apply (topologically @IsClosedMap).universally_le
      exact (UniversallyClosed.of_valuativeCriterion (pullback.diagonal f) h).out
    intro S
    have hc : CommSq S.i₁ (Spec.map (CommRingCat.ofHom (algebraMap S.R S.K)))
        f (S.i₂     pullback.fst f f     f) := ⟨by simp [← S.commSq.w_assoc]⟩
    let S' : ValuativeCommSq f := ⟨S.R, S.K, S.i₁, S.i₂     pullback.fst f f
       f, hc⟩
    have : Subsingleton S'.commSq.LiftStruct := hf S'
    let S'l₁ : S'.commSq.LiftStruct := ⟨S.i₂     pullback.fst f f,
      by simp [S', ← S.commSq.w_assoc], by simp [S']⟩
    let S'l₂ : S'.commSq.LiftStruct := ⟨S.i₂     pullback.snd f f,
      by simp [S', ← S.commSq.w_assoc], by simp [S', pullback.condition]⟩
    have h₁₂ : S'l₁ = S'l₂ := Subsingleton.elim _ _
    constructor
    constructor
    refine ⟨S.i₂     pullback.fst _ _, ?_, ?_⟩
    · simp [← S.commSq.w_assoc]
    · simp
      apply IsPullback.hom_ext (IsPullback.of_hasPullback _ _)
      · simp
      · simp only [Category.assoc, pullback.diagonal_snd, Category.comp_id]
        exact congrArg CommSq.LiftStruct.l h₁₂

@[stacks 01KZ]
lemma IsSeparated.valuativeCriterion [IsSeparated f] :
    ValuativeCriterion.Uniqueness f := by
  intro S
  constructor
  rintro ⟨l₁, hl₁, hl₁'⟩ ⟨l₂, hl₂, hl₂'⟩
  ext : 1
  dsimp at *
  have h := hl₁'.trans hl₂'.symm
  let Z := pullback (pullback.diagonal f) (pullback.lift l₁ l₂ h)
  let g : Z ⟶ Spec(S.R) := pullback.snd _ _
  have : IsClosedImmersion g := MorphismProperty.pullback_snd _ _ inferInstance
  have hZ : IsAffine Z := by
    rw [@HasAffineProperty.iff_of_isAffine @IsClosedImmersion] at this
    exact this.left
  suffices IsIso g by
    rw [← cancel_epi g]
    conv_lhs => rw [← pullback.lift_fst l₁ l₂ h, ← pullback.condition_assoc]
    conv_rhs => rw [← pullback.lift_snd l₁ l₂ h, ← pullback.condition_assoc]
    simp
  suffices h : Function.Bijective (g.appTop) by
    refine (HasAffineProperty.iff_of_isAffine (P := MorphismProperty.isomorphisms
       Scheme)).mpr ?_
    exact ⟨hZ, (ConcreteCategory.isIso_iff_bijective _).mpr h⟩
  constructor
  · let l : Spec(S.K) ⟶ Z :=
      pullback.lift S.i₁ (Spec.map (CommRingCat.ofHom (algebraMap S.R S.K))) (by
        apply IsPullback.hom_ext (IsPullback.of_hasPullback _ _)
        · simpa using hl₁.symm
        · simpa using hl₂.symm)
    have hg : l     g = Spec.map (CommRingCat.ofHom (algebraMap S.R S.K)) :=
      pullback.lift_snd _ _ _
    have : Function.Injective ((l     g).appTop) := by
      rw [hg]
      let e := arrowIsoΓSpecOfIsAffine (CommRingCat.ofHom <| algebraMap S.R S.K)
      let P : MorphismProperty CommRingCat :=
        RingHom.toMorphismProperty <| fun f     Function.Injective f
      have : (RingHom.toMorphismProperty <| fun f     Function.Injective
         f).RespectsIso :=
        RingHom.toMorphismProperty_respectsIso_iff.mp RingHom.injective_respectsIso
      change P _
      rw [← MorphismProperty.arrow_mk_iso_iff (P := P) e]
      exact FaithfulSMul.algebraMap_injective S.R S.K
    rw [Scheme.comp_appTop] at this
    exact Function.Injective.of_comp this
  · rw [@HasAffineProperty.iff_of_isAffine @IsClosedImmersion] at this
    exact this.right

/-- The **valuative criterion** for separated morphisms. -/
lemma IsSeparated.eq_valuativeCriterion :
    @IsSeparated = ValuativeCriterion.Uniqueness     @QuasiSeparated := by
  ext X Y f
  exact ⟨fun _     ⟨IsSeparated.valuativeCriterion f, inferInstance⟩,
    fun ⟨H, _⟩     .of_valuativeCriterion f H⟩

end Uniqueness

/-- The **valuative criterion** for proper morphisms. -/
@[stacks 0BX5]
lemma IsProper.eq_valuativeCriterion :
    @IsProper = ValuativeCriterion     @QuasiCompact     @QuasiSeparated     @LocallyOfFiniteType := by
  rw [isProper_eq, IsSeparated.eq_valuativeCriterion, ValuativeCriterion.eq,
    UniversallyClosed.eq_valuativeCriterion]
  simp_rw [inf_assoc]
  ext X Y f
  change _ ∧ _ ∧ _ ∧ _ ∧ _ ↔ _ ∧ _ ∧ _ ∧ _ ∧ _
  tauto

/-- The **valuative criterion** for proper morphisms. -/
@[stacks 0BX5]
lemma IsProper.of_valuativeCriterion [QuasiCompact f] [QuasiSeparated f] [LocallyOfFiniteType f]
    (H : ValuativeCriterion f) : IsProper f := by
  rw [eq_valuativeCriterion]
  exact ⟨⟨⟨  _  ,   _  ⟩,   _  ⟩,   _  ⟩

end AlgebraicGeometry
```

Listing 1: ValuativeCriterion.lean