${}^{-1-1}2\ {}^{opop}2$
$\mathcal{U}\mathcal{U}1\ \nVdash\nVdash$
$:=:=2$

Open Immersions of Schemes in Mathlib4
A Companion to `OpenImmersion.lean`

1 Introduction

This document provides a natural language companion to the `OpenImmersion.lean` file in Mathlib4. The file develops the theory of open immersions of schemes, which are fundamental morphisms that allow us to view open subschemes as schemes in their own right while maintaining the relationship with their ambient scheme.

Open immersions are the scheme-theoretic generalization of open embeddings of topological spaces. They play a crucial role in gluing constructions, descent theory, and the study of non-affine schemes built from affine pieces.

2 Basic Definition

2.1 Open Immersion as Morphism Property

```
abbrev IsOpenImmersion : MorphismProperty (Scheme.{u}) :=
  fun _ _ f      LocallyRingedSpace.IsOpenImmersion f.toLRSHom
```

Natural Language: A morphism of schemes $f : X \to Y$ is an open immersion if it is an open immersion when viewed as a morphism of locally ringed spaces. This means the underlying continuous map is an open embedding and the induced maps on sheaves are isomorphisms on the image.

2.2 Composition Property

```
instance IsOpenImmersion.comp {X Y Z : Scheme.{u}} (f : X      Y) (g : Y      Z)
    [IsOpenImmersion f] [IsOpenImmersion g] : IsOpenImmersion (f      g)
```

Natural Language: The composition of two open immersions is an open immersion. This is a fundamental property that makes open immersions well-behaved under composition.

3 Topological Properties

3.1 Open Range

```
theorem IsOpenImmersion.isOpen_range {X Y : Scheme.{u}} (f : X      Y) [H : IsOpenImmersion f] :
    IsOpen (Set.range f.base)

theorem isOpenEmbedding : IsOpenEmbedding f.base :=
  H.isOpenEmbedding
```

Natural Language: An open immersion $f : X \to Y$ has an open image in Y, and the underlying map $f : X \to Y$ is an open embedding of topological spaces. This captures the "open" part of "open immersion."

3.2 Opens Range

```
def opensRange : Y.Opens :=
  Y.toTopCat.openNhdsOfBase (Set.range f.base) (IsOpenImmersion.isOpen_range f)
```

Natural Language: For an open immersion $f : X \to Y$, we can consider its range as an open subset of Y. This gives us a canonical open set that captures the "image" of X inside Y.

4 Functoriality on Opens

4.1 Image Functor

```
abbrev opensFunctor : X.Opens      Y.Opens :=
  Opens.map f.base

lemma image_le_image_of_le {U V : X.Opens} (e : U      V) : f ^U U      f ^U V :=↩
      by
  exact Opens.map_mono f.base e
```

Natural Language: An open immersion induces a functor from opens of X to opens of Y by taking images. This functor is monotonic: larger open sets have larger images.

4.2 Image Properties

```
lemma image_top_eq_opensRange : f ^U      = f.opensRange := by
  ext; exact Set.image_univ.symm

lemma preimage_image_eq (U : X.Opens) : f ^{-1}U f ^U U = U := by
  ext x; exact  fun    y  , hy,  e   => e     hy, fun h =>  x  , h,  r f l
```

Natural Language: The image of the entire space X equals the range of f. More importantly, taking the preimage of an image gives back the original open set, reflecting the embedding nature of open immersions.

4.3 Injectivity

```
lemma image_injective : Function.Injective (f ^U   ) := by
  intro U V h
  rw [   preimage_image_eq f U, h, preimage_image_eq f V]
```

Natural Language: The image functor is injective: different open sets in X have different images in Y. This is a key property that allows us to recover information about X from its image in Y.

5 Sheaf Isomorphisms

5.1 App Isomorphism on Range

```
lemma isIso_app (V : Y.Opens) (hV : V     f.opensRange) : IsIso (f.app V) := by
  rw [isIso_iff_bijective]
  exact LocallyRingedSpace.IsOpenImmersion.app_bijective f.toLRSHom V hV
```

Natural Language: For open sets $V \subseteq Y$ that are contained in the range of f, the induced map on sections $\Gamma(Y, V) \to \Gamma(X, f^{-1}V)$ is an isomorphism. This captures the "immersion" part of "open immersion."

5.2 Canonical App Isomorphism

```
def appIso (U) :   (Y, f ^U U)       (X, U) :=
  { hom := f.app (f ^U U)     X.presheaf.map (eqToHom (preimage_image_eq f U)).↩
     op
    inv := X.presheaf.map (eqToHom (preimage_image_eq f U).symm).op     (inv (f↩
        .app (f ^U U)))
    -- isomorphism properties }
```

Natural Language: For any open $U \subseteq X$, the sections over its image $f(U) \subseteq Y$ are canonically isomorphic to the sections over U. This is the fundamental isomorphism that makes open immersions behave like embeddings at the sheaf level.

5.3 Naturality

```
theorem appIso_inv_naturality {U V : X.Opens} (i : op U     op V) :
    (f.appIso V).inv     X.presheaf.map i =
    Y.presheaf.map (f.opensFunctor.map i.unop).op     (f.appIso U).inv
```

Natural Language: The app isomorphisms are natural with respect to restriction maps. This ensures that the isomorphisms are compatible with the sheaf structure.

6 Opens Equivalence

6.1 Equivalence of Opens

```
def IsOpenImmersion.opensEquiv {X Y : Scheme.{u}} (f : X     Y) [↩
     IsOpenImmersion f] :
    X.Opens     f.opensRange.Opens :=
  { toFun := fun U =>    (f ^U U).1, image_le_opensRange f  U
    invFun := fun U => f ^{-1}U U.1
    left_inv := preimage_image_eq f
    right_inv := fun U => Subtype.ext (image_preimage_eq_opensRange_inter f U↩
        .1) }
```

Natural Language: An open immersion $f : X \to Y$ induces an equivalence between the opens of X and the opens of the range of f (viewed as an open subset of Y). This shows that X is essentially the same as its image, just viewed in different contexts.

7 Examples of Open Immersions

7.1 Basic Opens in Spec

```
instance basic_open_isOpenImmersion {R : CommRingCat.{u}} (f : R) :
    IsOpenImmersion (Spec.map (CommRingCat.ofHom (algebraMap R (Localization.↩
        Away f))))

instance {R} [CommRing R] (f : R) :
    IsOpenImmersion (Spec.map (algebraMap R R[  f    ]))
```

Natural Language: The canonical map from Spec($R[f^{-1}]$) to Spec(R) (corresponding to the localization $R \to R[f^{-1}]$) is an open immersion. This gives us the basic opens in affine schemes as open subschemes.

7.2 Localization Criterion

```
lemma _root_.AlgebraicGeometry.IsOpenImmersion.of_isLocalization {R S} [↩
    CommRing R] [CommRing S]
    [Algebra R S] [IsLocalization (Algebra.algebraMapSubmonoid R T) S] :
    IsOpenImmersion (Spec.map (algebraMap R S).op)
```

Natural Language: More generally, if S is a localization of R at some multiplicative set, then $\mathrm{Spec}(S) \to \mathrm{Spec}(R)$ is an open immersion. This provides a large class of examples and connects open immersions to classical algebraic concepts.

8 Characterization via Affine Covers

8.1 Local Characterization

```
theorem exists_affine_mem_range_and_range_subset
    {f : X      Y} [IsOpenImmersion f] (x : Y) (hx : x      Set.range f.base) :
        (U : Y.affineOpens), x      U      Set.range f.base      U
```

Natural Language: For an open immersion, every point in the range has an affine open neighborhood that contains the entire range. This provides a way to understand open immersions in terms of affine pieces.

9 Construction of Schemes from Open Immersions

9.1 Scheme Construction

```
def toScheme : Scheme := by
  apply LocallyRingedSpace.IsOpenImmersion.scheme Y.toLocallyRingedSpace
  intro x
  obtain   U  , hxU,  i U   := exists_affine_mem_range_and_range_subset f x (Set.↩
      mem_range_of_surjective h x)
  exact   U  .1, iU,   U  .2
```

Natural Language: Given a locally ringed space Y and a surjective open immersion from affine schemes covering Y, we can construct a scheme structure on Y. This is a fundamental construction principle for schemes.

9.2 Universal Property

```
def toSchemeHom : toScheme Y f      Y :=
    LocallyRingedSpace   .IsOpenImmersion.toSchemeHom Y   f

instance toSchemeHom_isOpenImmersion : AlgebraicGeometry.IsOpenImmersion (↩
    toSchemeHom Y f)
```

Natural Language: The constructed scheme comes with a canonical open immersion into the original locally ringed space, and this morphism is indeed an open immersion of schemes.

10 Pullbacks and Fiber Products

10.1 Pullback Properties

```
instance pullback_snd_of_left : IsOpenImmersion (pullback.snd f g) := by
  have := PullbackCone.isColimit_of_left IsOpenImmersion.isColimitOfLeft
  apply PresheafedSpace.IsOpenImmersion.of_isColimit this
  apply SheafedSpace.IsOpenImmersion.of_isColimit this

instance pullback_fst_of_right : IsOpenImmersion (pullback.fst g f) := by
  rw [    pullbackSymmetry_hom_comp_snd]
  infer_instance
```

Natural Language: In a pullback diagram where one of the morphisms is an open immersion, the projection to the side opposite the open immersion is also an open immersion. This is a fundamental property of pullbacks of open immersions.

10.2 Range of Pullbacks

```
theorem range_pullback_snd_of_left :
    Set.range (pullback.snd f g).base = (g ^{-1}U f.opensRange).1

theorem opensRange_pullback_snd_of_left :
    (pullback.snd f g).opensRange = g ^{-1}U f.opensRange
```

Natural Language: The range of the pullback morphism is exactly the preimage of the range of the original open immersion. This gives us a concrete description of pullbacks in terms of open sets.

11 Base Change Properties

11.1 Stability Under Base Change

```
instance pullback_to_base [IsOpenImmersion g] :
    IsOpenImmersion (limit.   (cospan f g) WalkingCospan.one) := by
  rw [    limit.w (cospan f g) WalkingCospan.Hom.inl]
  infer_instance
```

Natural Language: Open immersions are stable under base change: if $f : X \to Z$ is any morphism and $g : Y \to Z$ is an open immersion, then the pullback $X \times_Z Y \to X$ is also an open immersion.

11.2 Base Change Formulas

```
theorem range_pullback_to_base_of_left :
    Set.range (pullback.fst f g      f).base =
    Set.range f.base      Set.range g.base

theorem range_pullback_to_base_of_right :
    Set.range (pullback.fst g f      g).base =
    Set.range f.base      Set.range g.base
```

Natural Language: The range of the composition from a pullback to the base is the intersection of the ranges of the original morphisms. This provides explicit formulas for understanding pullbacks geometrically.

12 Applications and Examples

12.1 Open Subschemes

Open immersions provide the correct notion of "open subscheme": given a scheme Y and an open subset $U \subseteq Y$, there exists a scheme structure on U and an open immersion $U \hookrightarrow Y$ that makes U an "open subscheme" of Y.

12.2 Gluing Constructions

Open immersions are fundamental to gluing constructions in algebraic geometry:

- Cover a space with affine open pieces
- Each piece is an open immersion into the total space
- Gluing data consists of isomorphisms on overlaps
- The result is a scheme built from affine pieces

12.3 Descent Theory

The stability of open immersions under base change makes them important in descent theory and the study of morphisms of schemes.

13 Technical Lemmas

13.1 Compatibility with Sections

```
lemma appLE_appIso_inv {X Y : Scheme.{u}} (f : X      Y) [IsOpenImmersion f] {U : Y.Opens}
    {V : X.Opens} (e : V      f ^{-1}U U) :
    f.appLE U V e = (f.appIso V).inv
    Y.presheaf.map (homOfLE (show f ^U V      U from image_le_image_of_le f e)).op
```

Natural Language: Technical compatibility results showing how the various section maps interact with the canonical isomorphisms. These are essential for proving functoriality and naturality properties.

14 Conclusion

Open immersions provide a robust framework for understanding open subschemes and embeddings in algebraic geometry. Key insights include:

1. **Local nature**: Open immersions can be checked locally and are stable under composition
2. **Sheaf isomorphisms**: They induce isomorphisms on sections over their range
3. **Equivalence of opens**: They create equivalences between open set lattices
4. **Base change stability**: They behave well under pullbacks and fiber products
5. **Construction principle**: They provide ways to construct schemes from locally ringed spaces

Open immersions are fundamental to:

- Defining open subschemes
- Gluing constructions for non-affine schemes
- Descent theory and étale topology
- Understanding the relationship between schemes and their affine pieces

The theory developed in this file provides the foundation for more advanced constructions in algebraic geometry, particularly those involving non-affine schemes and gluing techniques.