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

Gluing Schemes in Mathlib4

A Companion to `Gluing.lean`

1 Introduction

This document provides a natural language companion to the `Gluing.lean` file in Mathlib4. The file develops the theory of gluing schemes together from local pieces, which is fundamental to constructing global objects from local data. This includes both general gluing constructions and specialized "locally directed" gluing for certain well-behaved diagrams.

2 The Glue Data Structure

2.1 Definition of Glue Data

```
structure GlueData extends CategoryTheory.GlueData Scheme where
  f_open :     i j, IsOpenImmersion (f i j)
```

Natural Language: A glue data for schemes consists of:

1. An index set J
2. Schemes U_i for each $i \in J$ (the pieces to be glued)
3. Schemes V_{ij} for each pair (i, j) (the overlaps)
4. Open immersions $f_{ij} : V_{ij} \to U_i$
5. Transition maps $t_{ij} : V_{ij} \to V_{ji}$
6. Compatibility conditions ensuring the gluing is well-defined

The key additional requirement is that all the maps f_{ij} are open immersions, preserving the geometric structure.

2.2 Glue Data Hierarchy

```
abbrev toLocallyRingedSpaceGlueData : LocallyRingedSpace.GlueData :=
  { f_open := D.f_open
    toGlueData :=     .mapGlueData forgetToLocallyRingedSpace }
```

Natural Language: The gluing construction for schemes builds on gluing constructions for locally ringed spaces, sheafed spaces, and presheafed spaces. This hierarchical approach ensures that all the relevant structure is properly preserved.

3 The Glued Scheme Construction

3.1 Implementation Details

```
def gluedScheme : Scheme := by
  apply LocallyRingedSpace.IsOpenImmersion.scheme
    D.toLocallyRingedSpaceGlueData.toGlueData.glued
  intro x
  obtain  i , y,  r f l  := D.toLocallyRingedSpaceGlueData.  _jointly_surjective
      x
  refine  _ , ((D.U i).affineCover.map y).toLRSHom
    D.toLocallyRingedSpaceGlueData.toGlueData.  i, ? _
  ...
```

Natural Language: The glued scheme is constructed by first creating the glued locally ringed space, then verifying that every point has an affine neighborhood. This uses the fact that each piece U_i has an affine cover, and these covers can be transferred to the glued space.

3.2 Main Definition

```
abbrev glued : Scheme :=     .glued
```

Natural Language: The glued scheme $D.glued$ is the result of gluing together all the schemes U_i according to the glue data D.

4 Inclusion Morphisms

4.1 The Inclusion Maps

```
abbrev    (i : D.J) : D.U i     D.glued :=     .   i
```

Natural Language: For each index i, there is a canonical inclusion morphism $\iota_i : U_i \to D.glued$ that embeds the i-th piece into the glued scheme.

4.2 Open Immersion Property

```
instance   _isOpenImmersion   (i : D.J) : IsOpenImmersion (    .   i)
```

Natural Language: Each inclusion $\iota_i : U_i \to D.glued$ is an open immersion, meaning that U_i appears as an open subscheme of the glued space.

4.3 Joint Surjectivity

```
theorem    _jointly_surjective    (x :     .glued.carrier) :
        (i : D.J) (y : (D.U i).carrier), (D.   i).base y = x
```

Natural Language: The inclusion maps are jointly surjective, meaning every point of the glued scheme comes from some piece U_i. This shows that the gluing covers the entire space.

5 Gluing Conditions

5.1 Basic Glue Condition

```
simp (high), reassoc
theorem glue_condition (i j : D.J) : D.t i j     D.f j i     D.   j = D.f i j
     D.   i
```

Natural Language: The fundamental gluing condition states that the two ways of mapping from V_{ij} to the glued space (via U_i or via U_j) agree. This ensures that the overlaps are consistently identified.

5.2 Pullback Characterization

```
def vPullbackCone (i j : D.J) : PullbackCone (D.   i) (D.   j) :=
  PullbackCone.mk (D.f i j) (D.t i j     D.f j i) (by simp)

def vPullbackConeIsLimit (i j : D.J) : IsLimit (D.vPullbackCone i j)
```

Natural Language: The overlap V_{ij} is precisely the intersection $U_i \cap U_j$ in the glued space. This is expressed by saying that V_{ij} with its maps to U_i and U_j forms a pullback square.

6 Topological Properties

6.1 Carrier Isomorphism

```
def isoCarrier : D.glued.carrier     (D_).glued
```

Natural Language: The underlying topological space of the glued scheme is isomorphic to the gluing of the underlying topological spaces of the pieces. This shows that the scheme-theoretic gluing correctly captures the topological gluing.

6.2 Point Identification

```
def Rel (a b :    i, ((D.U i).carrier : Type _)) : Prop :=
      x : (D.V (a.1, b.1)).carrier, (D.f _ _).base x = a.2     (D.t _ _     D.f _
       _).base x = b.2
```

Natural Language: Points $x \in U_i$ and $y \in U_j$ are identified in the glued space if and only if there exists a point in V_{ij} that maps to x under f_{ij} and to y under the composition $t_{ij} \circ f_{ji}$.

6.3 Open Set Characterization

```
theorem isOpen_iff (U : Set D.glued.carrier) : IsOpen U         i, IsOpen ((D.
    i).base      ' U)
```

Natural Language: A subset of the glued space is open if and only if its preimage in each piece U_i is open. This gives a practical criterion for checking openness in glued schemes.

7 Open Covers from Glue Data

7.1 The Natural Cover

```
simps -isSimp
def openCover (D : Scheme.GlueData) : OpenCover D.glued where
  J := D.J
  obj := D.U
  map := D.
```

```
  f x := (D.   _jointly_surjective    x).choose
  covers x :=   _  , (D.   _jointly_surjective    x).choose_spec.choose_spec
```

Natural Language: The pieces U_i naturally form an open cover of the glued scheme $D.glued$, with the inclusion maps serving as the cover morphisms.

8 Gluing from Open Covers

8.1 Cover to Glue Data

```
simps
def gluedCover : Scheme.GlueData.{u} where
  J :=      .J
  U :=      .obj
  V := fun   x  ,  y   => pullback (     .map x) (     .map y)
  f _ _ := pullback.fst _ _
  t _ _ := (pullbackSymmetry _ _).hom
  ...
```

Natural Language: Given an open cover $\mathcal{U} = \{U_i\}$ of a scheme X, we can construct glue data where the overlaps V_{ij} are the pullbacks (intersections) $U_i \times_X U_j$.

8.2 Recovery Isomorphism

```
def fromGlued :      .gluedCover.glued      X
instance : IsIso      .fromGlued
```

Natural Language: The gluing of an open cover of X is canonically isomorphic to X itself. The canonical morphism $\mathcal{U}.gluedCover.glued \to X$ is an isomorphism.

9 Gluing Morphisms

9.1 Compatible Morphism Gluing

```
def glueMorphisms (      : OpenCover.{v} X) {Y : Scheme.{u}} (f :      x,      .obj
    x      Y)
    (hf :      x y, pullback.fst (     .map x) (     .map y)      f x = pullback.snd
        _ _      f y) :
    X      Y
```

Natural Language: Given morphisms $f_i : U_i \to Y$ that agree on overlaps (i.e., are compatible), we can glue them together to get a global morphism $X \to Y$.

9.2 Morphism Extension Property

```
theorem hom_ext (      : OpenCover.{v} X) {Y : Scheme} ( f     f    : X      Y)
    (h :      x,      .map x      f    =      .map x      f   ) :  f    =  f
```

Natural Language: Two morphisms $f_1, f_2 : X \to Y$ are equal if they agree when restricted to each element of an open cover. This is a fundamental "local-to-global" principle.

10 Locally Directed Gluing

10.1 Locally Directed Diagrams

```
variable {J : Type w} [Category.{v} J] (F : J      Scheme.{u})
variable [    {i j} (f : i      j), IsOpenImmersion (F.map f)]
variable [(F      forget).IsLocallyDirected]
```

Natural Language: A diagram $F : J \to$ **Scheme** is locally directed if:

- All morphisms in the diagram are open immersions
- For any schemes $F(i)$, $F(j)$ and points $x \in F(i)$, $y \in F(j)$ that map to the same point in the colimit, there exists $F(k)$ containing both points

10.2 Intersection Construction

```
def V (i j : J) : (F.obj i).Opens :=      (k :     k, (k      i)     (k      j)), (F.
    map k.2.1).opensRange
```

Natural Language: The "intersection" V_{ij} is defined as the union of all schemes $F(k)$ that map to both $F(i)$ and $F(j)$. This captures the overlapping regions in a directed system.

10.3 Glue Data Construction

```
def glueData : Scheme.GlueData where
  J := Shrink.{u} J
  U j := F.obj   j
  V ij := V F   ij  .1   ij  .2
  f i j := Scheme.Opens.    _
  t i j := t F   i    j
  ...
```

Natural Language: For a locally directed diagram, we can automatically construct glue data using the intersection construction. The transition maps are induced by the local directedness property.

10.4 Colimit Construction

```
def cocone : Cocone F where
  pt := (glueData F).glued
    .app j := F.map (eqToHom (by simp))      (glueData F).    (equivShrink _ j)

def isColimit : IsColimit (cocone F)
```

Natural Language: The glued scheme serves as the colimit of the locally directed diagram, with the natural morphisms from each component scheme.

11 Applications and Importance

11.1 General Gluing Applications

The general gluing theory enables:

- Construction of schemes from local affine pieces
- Proof that many constructions preserve the scheme property

- Development of descent theory and faithfully flat covers
- Construction of moduli spaces and parameter spaces

11.2 Locally Directed Applications

The locally directed gluing is particularly useful for:

- Union of schemes indexed by directed sets
- Direct limits in the category of schemes
- Constructions involving increasing sequences of open sets
- Algebraic closures and separable closures of schemes

11.3 Connection to Classical Algebraic Geometry

This gluing theory formalizes several classical constructions:

- Patching together affine charts to build projective varieties
- Constructing schemes from sheaves of rings
- Building global sections from compatible local sections
- Descent for morphisms and other properties

12 Technical Implementation

12.1 Multicoequalizer Construction

The gluing is implemented using multicoequalizers in the category of schemes:

- The coequalizer identifies points that should be the same
- Open immersions ensure the gluing preserves the scheme structure
- Compatibility conditions ensure well-definedness

12.2 Hierarchy of Constructions

The construction builds through several levels:

1. Topological gluing of underlying spaces
2. Presheafed space gluing with compatible structure sheaves
3. Sheafed space gluing with sheaf conditions
4. Locally ringed space gluing with local ring conditions
5. Scheme gluing with additional scheme-theoretic properties

13 Conclusion

The gluing theory in `Gluing.lean` provides:

13.1 Fundamental Constructions

- General framework for gluing schemes from local pieces
- Specialized efficient methods for directed diagrams
- Tools for extending local morphisms to global ones
- Characterizations of topological and algebraic properties

13.2 Theoretical Foundations

- Rigorous treatment of the local-to-global principle
- Proof that schemes can be reconstructed from their open covers
- Foundation for descent theory and faithfully flat morphisms
- Connection between categorical limits and geometric constructions

13.3 Practical Tools

- Concrete algorithms for checking gluing conditions
- Methods for extending local constructions globally
- Tools for working with covers and intersection conditions
- Efficient handling of directed systems and unions

This gluing theory is essential for algebraic geometry, providing both the theoretical framework and practical tools needed to construct and study schemes systematically.