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

Pullbacks of Schemes in Mathlib4

A Companion to `Pullbacks.lean`

1 Introduction

This document provides a natural language companion to the `Pullbacks.lean` file in Mathlib4. The file constructs fibered products (pullbacks) of schemes using gluing techniques, following Hartshorne's Theorem 3.3. The construction shows how to build pullbacks for arbitrary schemes by reducing to the affine case through open covers.

2 Main Construction Strategy

2.1 The Gluing Approach

The fundamental idea is to construct pullbacks via gluing:

- Given schemes X, Y, Z and morphisms $f : X \to Z$, $g : Y \to Z$
- For an open cover $\{U_i\}$ of X, if pullbacks $U_i \times_Z Y$ exist
- Then we can glue these pullbacks to construct $X \times_Z Y$

This reduces the problem to the affine case, where pullbacks are constructed via tensor products of rings.

3 The Pullback Namespace

3.1 Intersection Schemes

```
def v (i j :      .J) : Scheme :=
  pullback ((pullback.fst (     .map i      f) g)          .map i) (     .map j)
```

Natural Language: For indices i, j in an open cover, V_{ij} represents the intersection of $U_i \times_Z Y$ and $U_j \times_Z Y$ over X. This is constructed as the pullback of the natural maps.

3.2 Transition Maps

```
def t (i j :      .J) : v       f g i j      v       f g j i
```

Natural Language: The transition map $t_{ij} : V_{ij} \to V_{ji}$ is the canonical isomorphism given by the symmetry and associativity of pullbacks. This ensures that the intersections can be consistently glued.

3.3 Properties of Transition Maps

```
theorem t_id (i :      .J) : t       f g i i =       _
```

Natural Language: The transition map from V_{ii} to itself is the identity morphism.

4 The Gluing Data Structure

4.1 The Main Gluing Construction

```
def gluing : Scheme.GlueData.{u} where
  J :=      .J
  U i := pullback (     .map i      f) g
  V := fun   i  ,  j   => v       f g i j
  f _ _ := pullback.fst _ _
  t i j := t       f g i j
  cocycle i j k := cocycle       f g i j k
```

Natural Language: This constructs the gluing data needed to create the fibered product. The schemes $U_i = U_i \times_Z Y$ are glued together using the intersection schemes V_{ij} and transition maps t_{ij}.

4.2 Cocycle Condition

```
theorem cocycle (i j k :      .J) : t'       f g i j k      t'       f g j k i      t'
          f g k i j =       _
```

Natural Language: The composition of transition maps around a triple satisfies the cocycle condition, ensuring that the gluing is well-defined.

5 Projection Morphisms

5.1 First Projection

```
def p1 : (gluing       f g).glued      X
```

Natural Language: The first projection $p_1 : X \times_Z Y \to X$ is obtained by gluing the natural projections from each $U_i \times_Z Y$ to $U_i \subseteq X$.

5.2 Second Projection

```
def p2 : (gluing       f g).glued      Y
```

Natural Language: The second projection $p_2 : X \times_Z Y \to Y$ is obtained by gluing the natural projections from each $U_i \times_Z Y$ to Y.

5.3 Pullback Square Property

```
theorem p_comm : p1       f g      f = p2       f g      g
```

Natural Language: The glued scheme forms a pullback square: $p_1 \circ f = p_2 \circ g$, confirming that we have constructed the fibered product correctly.

6 Universal Property

6.1 Lifting Property

```
def gluedLift : s.pt      (gluing       f g).glued
```

Natural Language: Given any pullback cone s for the diagram $X \leftarrow Z \to Y$, there exists a unique morphism from the apex of s to the glued pullback.

6.2 Main Theorem: Glued Pullback is a Limit

```
def gluedIsLimit : IsLimit (PullbackCone.mk _ _ (p_comm      f g))
```

Natural Language: The glued construction satisfies the universal property of pullbacks, making it the categorical pullback (fibered product) in the category of schemes.

7 Special Cases and Applications

7.1 Affine-Affine Pullbacks

```
instance affine_hasPullback {A B C : CommRingCat}
    (f : Spec A     Spec C)
    (g : Spec B     Spec C) : HasPullback f g
```

Natural Language: When all schemes are affine, pullbacks exist and are computed using the tensor product construction in commutative rings.

7.2 General Existence

```
instance : HasPullbacks Scheme
```

Natural Language: The category of schemes has all pullbacks. This is proven using the gluing construction applied to affine covers.

8 Spectrum Tensor Product Isomorphism

8.1 The Main Isomorphism

```
def pullbackSpecIso :
    pullback (Spec.map (CommRingCat.ofHom (algebraMap R S)))
      (Spec.map (CommRingCat.ofHom (algebraMap R T)))     Spec(S    [R] T)
```

Natural Language: For commutative rings R, S, T with algebra structures, the pullback of $\mathrm{Spec}(S) \to \mathrm{Spec}(R) \leftarrow \mathrm{Spec}(T)$ is isomorphic to $\mathrm{Spec}(S \otimes_R T)$.

8.2 Projection Formulas

```
lemma pullbackSpecIso_inv_fst :
    (pullbackSpecIso R S T).inv     pullback.fst _ _ = Spec.map (ofHom
       includeLeftRingHom)
```

Natural Language: The first projection corresponds to the ring homomorphism $s \mapsto s \otimes 1$ from S to $S \otimes_R T$.

9 Open Covers of Pullbacks

9.1 Cover by Left Components

```
def openCoverOfLeft (     : OpenCover X) (f : X     Z) (g : Y     Z) : OpenCover
    (pullback f g)
```

Natural Language: Given an open cover $\{U_i\}$ of X, the pullback $X \times_Z Y$ is covered by the schemes $U_i \times_Z Y$.

9.2 Cover by Both Components

```
def openCoverOfLeftRight (  X    : X.OpenCover) (  Y    : Y.OpenCover) (f : X     Z
    ) (g : Y     Z) :
    (pullback f g).OpenCover
```

Natural Language: Given open covers of both X and Y, the pullback is covered by all pairwise products $U_i \times_Z V_j$.

10 Geometric Properties

10.1 Preservation of Affine Property

```
instance isAffine_of_isAffine_isAffine_isAffine {X Y Z : Scheme}
    (f : X     Z) (g : Y     Z) [IsAffine X] [IsAffine Y] [IsAffine Z] :
    IsAffine (pullback f g)
```

Natural Language: The pullback of affine schemes over an affine base is affine. This follows from the tensor product construction for affine schemes.

10.2 Empty Pullbacks

```
theorem _root_.AlgebraicGeometry.Scheme.isEmpty_pullback
    {X Y S : Scheme.{u}} (f : X     S) (g : Y     S)
    (H : Disjoint (Set.range f.base) (Set.range g.base)) : IsEmpty    (Limits.
        pullback f g)
```

Natural Language: If the images of f and g are disjoint in S, then the pullback is empty.

11 Applications to Cartesian Monoidal Structure

11.1 Cartesian Structure on Over Categories

```
instance : CartesianMonoidalCategory (Over S)
```

Natural Language: The existence of pullbacks gives the category of S-schemes a cartesian monoidal structure, where the tensor product is the pullback over S.

12 Conclusion

The pullback construction in `Pullbacks.lean` provides:

- A systematic way to construct fibered products of arbitrary schemes
- Reduction of the general case to the well-understood affine case
- Complete proof that the category of schemes has all pullbacks
- Explicit constructions for various special cases and covers
- Foundation for cartesian monoidal structure in algebraic geometry

This construction is fundamental to many areas of algebraic geometry, including base change, families of schemes, and moduli theory.