Grammatical sheaf cohomology, its MODOS proof-assistant and WorkSchool 365 market for learning reviewers

The “double plus” definition of sheafification says that not-only the outer families-of-families are modulo the germ-equality, but-also the inner families are modulo the germ-equality. This outer-inner contrast is the hint that the “double plus” should be some inductive construction... that grammatical sheaf cohomology exists! And the MODOS proof-assistant is its cut-elimination confluence. The key technique is that the grammatical sieves (nerve) could be programmed such to inductively store both the (possibly incompatible) glued-data along with its differentials (incompatibilities) of the gluing. The significance of studying “family of families” grammatically instead of the semantic geometry-gluing is similar as the earlier significance of studying “equality of equalities” grammatically instead of the semantic homotopy-paths. And such research programme, prompted from the mathematicians Kosta Dosen and Pierre Cartier, would require some new WorkSchool365.com education market for paid tested learning peer reviewers.

.

U₁

U₂

U₃

U₁’

U₂’

U₁’’

U₁’

f: F(V₃)

f∩U₁’’

⋮

⋯

**Diagram 1.** Each nested basic sieve is some refinement of the fixed cover . The grammatical total/sum sieve is no longer one-to-one (mono) into the actual arrows of the site.

**Lemma 03AS** (<https://stacks.math.columbia.edu/tag/03AS>). Let be a category. Let be a family of morphisms with fixed target such that all fibre products exist in . Consider the chain complex of abelian presheaves

where the last nonzero term is placed in degree 0 and where the map

is given by times the canonical map. Then there is an isomorphism

functorial in

Note that any of the products may be empty. So how is the usual nerve modelled? Via the contravariant structure sheaf of the compactly-supported continuous functions, which is in fact also some covariant co-sheaf. Therefore, instead of

**Lemma 03F5.** Let be a presheaf of rings on . The chain complex

is exact in positive degrees

Oneself could dualize any co-sheaf through the complex of the elementary projective sheaves (instead of the generators)

with the boundary maps

Note that this resulting complex would be the same as the linear dual of the through the dualizing complex of co-sheaves (Verdier dual)… The observation is that this duality is inevitable, so that homology of one (structure) co-sheaf and cohomology of another (coefficients) sheaf would be constructed simultaneously. Now how does the computational construction relate to the logical definition? This is Lemma 03AU and Lemma 03F7.

**Lemma 03AU.** For abelian presheaves only, not sheaves, there is a functorial quasi-isomorphism

where the right-hand side indicates the derived functor

of the left exact functor

**Lemma 03F7.** Let any abelian sheaf . Assume that for all , all and all . Then

And both lemmas rely on the technique of moving into the total complex of some double complex such as or the Cartan-Eilenberg resolution (Lemma 015I) for some injective resolution . Anyway, to sense the idea, remember that for any acyclic resolution , the long exact sequence allows to move through the double complex such as:

In other words, the cellular degree can be inductively decreased at the cost of increasing the coefficients degree. And for the injective resolution of some presheaf, this increase in the coefficients degree signifies that the coefficients are more complicated such as “family of families of base values” (or “superposition of superpositions”, in the case of the co-presheaf). This suggests to construct some single storage container both for the gluing and for the differentials (incompatibilities) of this gluing, as sketched in the *Diagram 1*. Memo that the grammatical total/sum sieve is no longer one-to-one (mono) into the actual arrows of the site; any actual arrow may be factorized via many (the cell degree) codes.

For the benefit of the lazy reader, *Diagram 2*is some instance of *Diagram* *1* where all the refinements from the fixed top cover-sieve are identities. And the Coq *Code C1* is the nerve-sieve inductive type for such limited instances.

U₁

f₂: F(U₂)

f₂∩U₁ - f₁’∩U₂

U₁

U₂

U₂

U₁

f₁: F(U₁)

U

U

U₁∩U₂

f₂’: F(U₂)

U₂

U₁

f₁’: F(U₁)

U

U₂

**Diagram 2.** Instance of *Diagram 1* where all the refinements from the fixed top cover-sieve are identities.

**Variable** nerveStruct : seq nat -> bool.

**Variable** topSieve : nat.

**Inductive** nerveSieve: **forall** (dimCoef: nat) (dimCell: nat) (cell: seq nat), **Type** :=

| Diff\_nerveSieve: **forall** (dimCoef: nat) (dimCell: nat) (cell: seq nat)

(outer\_: Fin.t (S dimCell) -> Fin.t (S topSieve))

(innerCell\_: **forall** (outerIndex: Fin.t (S dimCell)), seq nat)

(cell\_eq: **forall** (outerIndex: Fin.t (S dimCell)),

seq.perm\_eq cell ((to\_nat (outer\_ outerIndex): nat) :: (innerCell\_ outerIndex)))

(cell\_nerveStruct: nerveStruct cell),

**forall** (inner\_nerveSieve: **forall** (outerIndex: Fin.t (S dimCell)),

nerveSieve dimCoef dimCell (innerCell\_ outerIndex)),

nerveSieve (S dimCoef) (S dimCell) cell

| Glue\_nerveSieve: **forall** (dimCoef: nat) (dimCell: nat) (cell: seq nat),

**forall** (inner\_nerveSieve: **forall** (outerIndex: Fin.t (S topSieve)),

nerveSieve dimCoef dimCell cell),

nerveSieve (S dimCoef) dimCell cell

| Unit\_nerveSieve:

nerveSieve 0 0 [:: ].

Together with its “differential gluing” elimination-scheme into any (sheafified) sheaf:

**Definition** diffGluing: **forall** (dimCoef: nat),

**forall** (inner\_coef: **forall** (outerIndex: Fin.t (S topSieve)),

**forall** (dimCell: nat) (cell: seq nat),

nerveSieve dimCoef dimCell cell -> sheafiCoef cell),

**forall** (dimCell: nat) (cell: seq nat),

nerveSieve (S dimCoef) dimCell cell -> sheafiCoef cell.

And concretely the codes below show one computed example of such sieve values in this nerve type, together with one computed example of such coefficient values in this sheafified type.

Glue\_nerveSieve (two\_cases

(Diff id (two\_cases

(Diff (one\_cases (Fin.FS Fin.F1))

(one\_cases (Unit\_nerveSieve xpredT 1)))

(Diff (one\_cases Fin.F1) (one\_cases (Unit\_nerveSieve xpredT 1)))))

(Diff (two\_cases (Fin.FS Fin.F1) Fin.F1) (two\_cases

(Diff (one\_cases Fin.F1) (one\_cases (Unit\_nerveSieve xpredT 1)))

(Diff (one\_cases (Fin.FS Fin.F1))

(one\_cases (Unit\_nerveSieve xpredT 1))))))

: nerveSieve xpredT 1 3 2 [:: 0; 1]

glue\_shfyCoef

[< restrict\_shfyCoef [:: 1; 0; 1]

(diff\_shfyCoef

[< restrict\_shfyCoef [:: 0; 1] (congr\_shfyCoef dd) ;;

restrict\_shfyCoef [:: 0; 1] (congr\_shfyCoef bb) >]) ;;

restrict\_shfyCoef [:: 0; 0; 1]

(diff\_shfyCoef

[< restrict\_shfyCoef [:: 0; 1] (congr\_shfyCoef aa) ;;

restrict\_shfyCoef [:: 0; 1] (congr\_shfyCoef cc) >]) >]

: sheafiCoef [:: 0; 1]

Now returning to the general situation of *Diagram 1*, the pseudo-*Code C4* shows the outline of the nerve-sieve inductive type:

**Inductive** nerveSieve: **forall** K (UU : K → **Type** sieve at U) (**\_** : UU refines topCover along arrow u : U → topCoverUnion), **forall** (G : open **where** data will be stored) (**\_** : G ⊆ U), **forall** (dim: nat) (diffCell: **forall** i : {0, 1, …, dim-1}, topCoverOpens), **Type** :=

| NerveSieve\_Diff (\* at cell dim +1, at coeffiecients degree +1 \*) :

**forall** K UU G dim diffCell,

**forall** (famSieve\_ : **forall** Uk : UU, sieve at some open famVertex\_Uk along some pull arrow famPullArrow\_Uk : Uk → famVertex\_Uk and refining the topCover),

**forall** (outerFactor\_ : **forall** i : {0, 1, …, dim+1}, is some open Uk : UU with G ⊆ outerFactor\_(i) ⊆ U),

**forall** (inner\_nerveSieve : **forall** i : {0, 1, …, dim+1},

nerveSieve (famSieve\_(outerFactor\_ i))

(the generator open V\_i of famSieve\_(outerFactor\_ i) **where** G factorizes)

(**fun** j : {0, 1, …, dim} => outerFactor\_(**if** j<i **then** j **else** j+1) as topCoverOpens)),

**forall** (G\_weight : structCoSheaf G),

nerveSieve (SumSieve famSieve\_ over UU) G

(**fun** i : {0, 1, …, dim+1} => topCoverOpen generator of outerFactor\_i)

| NerveSieve\_Gluing (\* at same cell dim >= 0, at coefficients degree +1 \*) : …

| NerveSieve\_Base (\* at cell dim = 0, at coeffiecients degree = 0 \*) : …

Finally, below are the draft forms of the remaining *Code C5* for the particular *Diagram 2* and *Code C6* for the general *Diagram 1*:

<https://github.com/1337777/cartier>

and such research programme would require some new WorkSchool 365 education market for paid tested learning peer reviewers (sign-in via the Microsoft Marketplace):

<https://workschool365.com>

<https://appsource.microsoft.com/en-us/product/office/WA200003598>

**Learning Reviewers Qualification Quiz:** ❓ Q1. The MODOS end-goal is:

(A) proof-assistant for the computational logic of “family of families”.

(B) formalization of the correctness of the book “Categories for the Working Mathematician”.

(C) writing vertical pretty formulas in latex.

Click or tap here to enter text.

*In Word, “Insert; Add-ins; WorkSchool 365” to play this Coq script or sign-in for learning reviewers. WorkSchool365.com*

**Module** **Example**.

**From** Coq **Require** Lia Vectors.Vector.

**From** mathcomp **Require** **Import** ssreflect ssrfun ssrbool eqtype ssrnat seq fintype tuple finfun.

**Set** **Implicit** **Arguments**. **Unset** Strict **Implicit**. **Unset** **Printing** **Implicit** Defensive.

**Section** tools.

**Definition** to\_nat := **fun** m => **fun** i : Fin.t (S m) => (proj1\_sig (Fin.to\_nat i)).

**Definition** one\_cases : **forall** (P : Fin.t (S 0) -> **Type**)

(P1 : P Fin.F1) (p : Fin.t (S 0)), P p.

**Proof**. intros. apply: Fin.caseS'. apply P1.

clear p; intros p. pattern p. apply: Fin.case0.

**Defined**.

**Definition** two\_cases : **forall** (P : Fin.t (S 1) -> **Type**)

(P1 : P Fin.F1) (P2 : P (Fin.FS Fin.F1)) (p : Fin.t (S 1)), P p.

**Proof**. intros. apply: Fin.caseS'. apply P1.

apply: one\_cases. apply P2.

**Defined**.

**Variable** dimCell : nat.

**Variable** typeAt\_ : Fin.t (S dimCell) -> **Type**.

**Inductive** ilist : nat -> **Type** :=

| Nil\_ilist : ilist O

| Cons\_ilist : **forall** n (H : (n < (S dimCell))%coq\_nat),

typeAt\_ ( Fin.of\_nat\_lt H) -> ilist n -> ilist (S n).

**Definition** to\_ilist (t: **forall** i : Fin.t (S dimCell), typeAt\_ i) n (H : (n <= S dimCell)%coq\_nat) : ilist n.

**Proof**. induction n.

- apply: Nil\_ilist.

- apply: Cons\_ilist.

+ exact: (t (Fin.of\_nat\_lt H)).

+ apply IHn. abstract(Lia.lia).

**Defined**.

**End** tools.

**Section** nerveSieve.

**Variable** nerveStruct : seq nat -> bool.

**Variable** topSieve : nat.

**Inductive** nerveSieve: **forall** (dimCoef: nat) (dimCell: nat) (cell: seq nat), **Type** :=

| Diff\_nerveSieve: **forall** (dimCoef: nat) (dimCell: nat) (cell: seq nat)

(outer\_: Fin.t (S dimCell) -> Fin.t (S topSieve))

(innerCell\_: **forall** (outerIndex: Fin.t (S dimCell)), seq nat)

(cell\_eq: **forall** (outerIndex: Fin.t (S dimCell)),

seq.perm\_eq cell ((to\_nat (outer\_ outerIndex): nat) :: (innerCell\_ outerIndex)))

(cell\_nerveStruct: nerveStruct cell),

**forall** (inner\_nerveSieve: **forall** (outerIndex: Fin.t (S dimCell)),

nerveSieve dimCoef dimCell (innerCell\_ outerIndex)),

nerveSieve (S dimCoef) (S dimCell) cell

| Glue\_nerveSieve: **forall** (dimCoef: nat) (dimCell: nat) (cell: seq nat),

**forall** (inner\_nerveSieve: **forall** (outerIndex: Fin.t (S topSieve)),

nerveSieve dimCoef dimCell cell),

nerveSieve (S dimCoef) dimCell cell

| Unit\_nerveSieve:

nerveSieve 0 0 [:: ].

**Parameter** sheafiCoef : **forall** (cell: seq nat), **Type**.

**Parameter** restrict\_shfyCoef : **forall** (cell0: seq nat), **forall** (cell: seq nat) (i : nat),

perm\_eq cell (i :: cell0) -> sheafiCoef cell0 -> sheafiCoef cell.

**Parameter** diff\_shfyCoef : **forall** (dimCell: nat) (cell: seq nat),

ilist (**fun** (outerIndex: Fin.t (S dimCell)) => sheafiCoef cell) (S dimCell) -> sheafiCoef cell.

**Parameter** glue\_shfyCoef : **forall** (cell: seq nat),

ilist (**fun** (outerIndex: Fin.t (S topSieve)) => sheafiCoef ((to\_nat outerIndex) :: cell)) (S topSieve) -> sheafiCoef cell.

**Parameter** congr\_shfyCoef : **forall** (cell0: seq nat), **forall** (cell: seq nat),

perm\_eq cell cell0 -> sheafiCoef cell0 -> sheafiCoef cell.

**Definition** diffGluing: **forall** (dimCoef: nat),

**forall** (inner\_coef: **forall** (outerIndex: Fin.t (S topSieve)), **forall** (dimCell: nat) (cell: seq nat),

nerveSieve dimCoef dimCell cell -> sheafiCoef cell),

**forall** (dimCell: nat) (cell: seq nat),

nerveSieve (S dimCoef) dimCell cell -> sheafiCoef cell.

**Proof**. intros ? ? ? ? ns. inversion ns; subst.

{ (\* Diff\_nerveSieve \*)

(\* apply: (diff\_shfyCoef (fun i : 'I\_(\_.+1) =>

restrict\_shfyCoef (cell\_eq i)

(inner\_coef (outer\_ i) \_ (innerCell\_ i) (inner\_nerveSieve i)))). \*)

eapply diff\_shfyCoef. refine (to\_ilist **\_** (le\_n **\_**) ). intros i. apply: restrict\_shfyCoef.

+ apply: (cell\_eq i).

+ eapply (inner\_coef (outer\_ i)). apply: inner\_nerveSieve. }

{ (\* Glue\_nerveSieve \*)

(\* apply: glue\_shfyCoef (fun i : 'I\_topSieve =>

inner\_coef i dimCell cell (inner\_nerveSieve i)). \*)

apply glue\_shfyCoef. refine (to\_ilist **\_** (le\_n **\_**) ). intros i. eapply restrict\_shfyCoef.

+ exact: perm\_refl.

+ eapply (inner\_coef i). apply: (inner\_nerveSieve i). }

**Defined**.

**Definition** diffGluing\_unit:

**forall** (unit\_coef: sheafiCoef [:: ]),

**forall** (dimCell: nat) (cell: seq nat),

nerveSieve 0 dimCell cell -> sheafiCoef cell.

**Proof**. intros ? ? ? ns. inversion ns; subst.

{ (\* Unit\_nerveSieve \*) exact: unit\_coef. }

**Defined**.

**End** nerveSieve.

**Section** example1\_sheafiCoef.

**Definition** example1\_nerveSieve: nerveSieve (**fun** **\_** => true) 1 2 1 [:: 0].

**Proof**. apply: Glue\_nerveSieve. apply: two\_cases.

{ unshelve eapply Diff\_nerveSieve.

apply: one\_cases. exact: (@Fin.F1 1).

apply: one\_cases. exact: [:: ].

apply: one\_cases. reflexivity.

reflexivity.

apply: one\_cases. apply Unit\_nerveSieve. }

{ unshelve eapply Diff\_nerveSieve.

apply: one\_cases. exact: (@Fin.F1 1).

apply: one\_cases. exact: [:: ].

apply: one\_cases. reflexivity.

reflexivity.

apply: one\_cases. apply Unit\_nerveSieve. }

**Defined**.

**Notation** Diff y x := (@Diff\_nerveSieve **\_** **\_** **\_** **\_** **\_** y **\_** **\_** **\_** x).

**Print** example1\_nerveSieve.

(\* Glue\_nerveSieve

(two\_cases (Diff (one\_cases Fin.F1) (one\_cases (Unit\_nerveSieve xpredT 1)))

(Diff (one\_cases Fin.F1) (one\_cases (Unit\_nerveSieve xpredT 1))))

: nerveSieve xpredT 1 2 1 [:: 0] \*)

**Variable** aa bb cc dd : sheafiCoef [:: ].

**Definition** example1\_sheafiCoef: sheafiCoef [:: 0].

**Proof**. apply: (diffGluing **\_** example1\_nerveSieve). apply: two\_cases.

{ apply: diffGluing.

{ apply: two\_cases.

exact: (diffGluing\_unit aa). exact: (diffGluing\_unit bb). } }

{ apply: diffGluing.

{ apply: two\_cases.

exact: (diffGluing\_unit cc). exact: (diffGluing\_unit dd). } }

**Defined**.

**Notation** "[< x2 ;; .. ;; xn >]" := (Cons\_ilist x2 .. (Cons\_ilist xn (@Nil\_ilist **\_** **\_**)) ..)

(at level 0, format "[< '[' x2 ;; '/' .. ;; '/' xn ']' >]" ) .

**Arguments** Nil\_ilist {**\_** **\_**}.

**Arguments** restrict\_shfyCoef [**\_**] cell [**\_** **\_**] **\_**.

**Arguments** congr\_shfyCoef {**\_** **\_** **\_**} **\_**.

**Eval** compute in example1\_sheafiCoef.

(\* = glue\_shfyCoef

[< restrict\_shfyCoef [:: 1; 0]

(diff\_shfyCoef [< restrict\_shfyCoef [:: 0] cc >]) ;;

restrict\_shfyCoef [:: 0; 0]

(diff\_shfyCoef [< restrict\_shfyCoef [:: 0] aa >]) >]

: sheafiCoef [:: 0] \*)

**End** example1\_sheafiCoef.

**Section** example2\_sheafiCoef.

**Example** example2\_nerveSieve: nerveSieve (**fun** **\_** => true) 1 3 2 [:: 0; 1].

**Proof**. apply: Glue\_nerveSieve. apply: two\_cases.

{ unshelve eapply Diff\_nerveSieve.

exact: id.

{ apply: two\_cases. exact: [:: 1]. exact: [:: 0]. }

{ apply: two\_cases. reflexivity. reflexivity. }

reflexivity.

{ apply: two\_cases.

{ unshelve eapply Diff\_nerveSieve.

apply: one\_cases. exact: (Fin.FS (@Fin.F1 0)).

apply: one\_cases. exact: [::].

apply: one\_cases. reflexivity.

reflexivity.

apply: one\_cases. apply: Unit\_nerveSieve. }

{ unshelve eapply Diff\_nerveSieve.

apply: one\_cases. exact: (@Fin.F1 1).

apply: one\_cases. exact: [::].

apply: one\_cases. reflexivity.

reflexivity.

apply: one\_cases. apply: Unit\_nerveSieve. } } }

{ unshelve eapply Diff\_nerveSieve.

{ (\* permute, not id inclusion \*)

apply: two\_cases. exact: (Fin.FS (@Fin.F1 0)). exact: (@Fin.F1 1). }

{ apply: two\_cases. exact: [:: 0]. exact: [:: 1]. }

{ apply: two\_cases. reflexivity. reflexivity. }

reflexivity.

{ apply: two\_cases.

{ unshelve eapply Diff\_nerveSieve.

apply: one\_cases. exact: (@Fin.F1 1).

apply: one\_cases. exact: [::].

apply: one\_cases. reflexivity.

reflexivity.

apply: one\_cases. apply: Unit\_nerveSieve. }

{ unshelve eapply Diff\_nerveSieve.

apply: one\_cases. exact: (Fin.FS (@Fin.F1 0)).

apply: one\_cases. exact: [::].

apply: one\_cases. reflexivity.

reflexivity.

apply: one\_cases. apply: Unit\_nerveSieve. } } }

**Defined**.

**Notation** Diff y x := (@Diff\_nerveSieve **\_** **\_** **\_** **\_** **\_** y **\_** **\_** **\_** x).

**Print** example2\_nerveSieve.

(\*Glue\_nerveSieve (two\_cases

(Diff id (two\_cases

(Diff (one\_cases (Fin.FS Fin.F1))

(one\_cases (Unit\_nerveSieve xpredT 1)))

(Diff (one\_cases Fin.F1) (one\_cases (Unit\_nerveSieve xpredT 1)))))

(Diff (two\_cases (Fin.FS Fin.F1) Fin.F1) (two\_cases

(Diff (one\_cases Fin.F1) (one\_cases (Unit\_nerveSieve xpredT 1)))

(Diff (one\_cases (Fin.FS Fin.F1))

(one\_cases (Unit\_nerveSieve xpredT 1))))))

: nerveSieve xpredT 1 3 2 [:: 0; 1] \*)

**Variable** aa bb : sheafiCoef [:: 0].

**Variable** cc dd : sheafiCoef [:: 1].

**Definition** example2\_sheafiCoef: sheafiCoef [:: 0; 1].

**Proof**. apply: (diffGluing **\_** example2\_nerveSieve). apply: two\_cases.

{ apply: diffGluing. intros **\_**.

{ intros ? ? ns. inversion ns; subst. (\* **TODO:** for case-analysis, everywhere should use destructible ilist instead of function,

together with conversion operation ilist -> function, similar as ssreflect finfun \*)

{ move: (outer\_ (@Fin.F1 dimCell0)) (innerCell\_ (@Fin.F1 dimCell0))(cell\_eq (@Fin.F1 dimCell0)) (inner\_nerveSieve (@Fin.F1 dimCell0)).

apply: two\_cases.

{ move => innerCell\_0 cell\_eq0 inner\_nerveSieve0. inversion inner\_nerveSieve0; subst. apply: (congr\_shfyCoef cell\_eq0).

exact: aa. }

{ move => innerCell\_0 cell\_eq0 inner\_nerveSieve0. inversion inner\_nerveSieve0; subst. apply: (congr\_shfyCoef cell\_eq0).

exact: cc. } }

{ move: (inner\_nerveSieve (@Fin.F1 1)). move => inner\_nerveSieve0. inversion inner\_nerveSieve0; subst.

apply: glue\_shfyCoef. refine (to\_ilist **\_** (le\_n 2) ). apply: two\_cases. exact: aa. exact: cc. } } }

{ apply: diffGluing. intros **\_**.

{ intros ? ? ns. inversion ns; subst.

{ move: (outer\_ (@Fin.F1 dimCell0)) (innerCell\_ (@Fin.F1 dimCell0))(cell\_eq (@Fin.F1 dimCell0)) (inner\_nerveSieve (@Fin.F1 dimCell0)).

apply: two\_cases.

{ move => innerCell\_0 cell\_eq0 inner\_nerveSieve0. inversion inner\_nerveSieve0; subst. apply: (congr\_shfyCoef cell\_eq0).

exact: bb. }

{ move => innerCell\_0 cell\_eq0 inner\_nerveSieve0. inversion inner\_nerveSieve0; subst. apply: (congr\_shfyCoef cell\_eq0).

exact: dd. } }

{ move: (inner\_nerveSieve (@Fin.F1 1)). move => inner\_nerveSieve0. inversion inner\_nerveSieve0; subst.

apply: glue\_shfyCoef. refine (to\_ilist **\_** (le\_n 2) ). apply: two\_cases. exact: bb. exact: dd. } } }

**Defined**.

**Notation** "[< x2 ;; .. ;; xn >]" := (Cons\_ilist x2 .. (Cons\_ilist xn (@Nil\_ilist **\_** **\_**)) ..)

(at level 0, format "[< '[' x2 ;; '/' .. ;; '/' xn ']' >]" ) .

**Arguments** Nil\_ilist {**\_** **\_**}.

**Arguments** restrict\_shfyCoef [**\_**] cell [**\_** **\_**] **\_**.

**Arguments** congr\_shfyCoef {**\_** **\_** **\_**} **\_**.

**Eval** compute in example2\_sheafiCoef. (\* note that because of the permute instruction in example2\_nerveSieve,

then the order dd,bb is permuted as contrasted to aa,cc \*)

(\* = glue\_shfyCoef

[< restrict\_shfyCoef [:: 1; 0; 1]

(diff\_shfyCoef

[< restrict\_shfyCoef [:: 0; 1] (congr\_shfyCoef dd) ;;

restrict\_shfyCoef [:: 0; 1] (congr\_shfyCoef bb) >]) ;;

restrict\_shfyCoef [:: 0; 0; 1]

(diff\_shfyCoef

[< restrict\_shfyCoef [:: 0; 1] (congr\_shfyCoef aa) ;;

restrict\_shfyCoef [:: 0; 1] (congr\_shfyCoef cc) >]) >]

: sheafiCoef [:: 0; 1]

\*)

**End** example2\_sheafiCoef.

**End** **Example**.

And for the general situation:

(\*\* # #

#+TITLE: cartierSolution0.v

https://github.com/1337777/cartier/blob/master/cartierSolution0.v

QUICK START FROM Inductive nerveSieve

-----

#+BEGIN\_SRC coq :exports both :results silent # # \*\*)

**From** Coq **Require** Lia.

**From** Coq **Require** **Import** RelationClasses **Setoid** SetoidClass

**Classes**.Morphisms\_Prop RelationPairs CRelationClasses CMorphisms.

**From** mathcomp **Require** **Import** ssreflect ssrfun ssrbool eqtype ssrnat seq fintype tuple finfun.

**Set** **Implicit** **Arguments**. **Unset** Strict **Implicit**. **Unset** **Printing** **Implicit** Defensive.

**Set** Primitive **Projections**. **Set** **Universe** Polymorphism.

**Close** **Scope** bool. **Declare** **Scope** poly\_scope. **Delimit** **Scope** poly\_scope with poly. **Open** **Scope** poly.

**Module** **Type** GENE.

**Class** relType : **Type** := RelType

{ \_type\_relType : **Type**;

\_rel\_relType : crelation \_type\_relType;

\_equiv\_relType :> Equivalence \_rel\_relType }.

**About** relType.

**Coercion** \_type\_relType : relType >-> Sortclass.

**Definition** equiv {A: **Type**} {R: crelation A} `{Equivalence A R} : crelation A := R.

(\* **TODO:** keep or comment \*)

**Arguments** \_rel\_relType : simpl never.

**Arguments** \_equiv\_relType : simpl never.

**Arguments** equiv : simpl never.

**Notation** " x == y " := (@equiv (\* (@\_type\_relType \_) \*) **\_** (@\_rel\_relType **\_**) (@\_equiv\_relType **\_**) x y)

(at level 70, no associativity) : type\_scope.

**Notation** LHS := (**\_** : **fun** XX => XX == **\_**).

**Notation** RHS := (**\_** : **fun** XX => **\_** == XX).

**Notation** "[| x ; .==. |]" := (exist (**fun** t => (**\_** == **\_**)) x **\_**) (at level 10, x at next level) : poly\_scope.

**Notation** "[| x ; .=. |]" := (exist (**fun** t => (**\_** = **\_**)) x **\_**) (at level 10, x at next level) : poly\_scope.

**Parameter** vertexGene : eqType.

**Parameter** arrowGene : vertexGene -> vertexGene -> relType.

**Notation** "''Gene' ( V ~> U )" := (@arrowGene U V)

(at level 0, format "''Gene' ( V ~> U )") : poly\_scope.

**Parameter** composGene :

**forall** U, **forall** V W, 'Gene( W ~> V ) -> 'Gene( V ~> U ) -> 'Gene( W ~> U ).

**Notation** "wv o:>gene vu" := (@composGene **\_** **\_** **\_** wv vu)

(at level 40, vu at next level) : poly\_scope.

**Declare** **Instance** composGene\_Proper: **forall** U V W, Proper (equiv ==> equiv ==> equiv) (@composGene U V W).

**Parameter** identGene : **forall** {U : vertexGene}, 'Gene( U ~> U ).

**Parameter** composGene\_compos :

**forall** (U V : vertexGene) (vu : 'Gene( V ~> U ))

(W : vertexGene) (wv : 'Gene( W ~> V )),

**forall** X (xw : 'Gene( X ~> W )),

xw o:>gene ( wv o:>gene vu ) == ( xw o:>gene wv ) o:>gene vu.

**Parameter** composGene\_identGene :

**forall** (U V : vertexGene) (vu : 'Gene( V ~> U )),

(@identGene V) o:>gene vu == vu .

**Parameter** identGene\_composGene :

**forall** (U : vertexGene), **forall** (W : vertexGene) (wv : 'Gene( W ~> U )),

wv o:>gene (@identGene U) == wv.

**Notation** typeOf\_objects\_functor := (vertexGene -> relType).

**Class** relFunctor (F : typeOf\_objects\_functor) (G G' : vertexGene) : **Type** := RelFunctor

{ \_fun\_relFunctor : 'Gene( G' ~> G ) -> F G -> F G' ;

\_congr\_relFunctor :> Proper (equiv ==> @equiv **\_** **\_** (@\_equiv\_relType ( F G ))

==> @equiv **\_** **\_** (@\_equiv\_relType ( F G'))) \_fun\_relFunctor ; }.

**Coercion** \_fun\_relFunctor : relFunctor >-> Funclass.

**Definition** typeOf\_arrows\_functor (F : typeOf\_objects\_functor)

:= **forall** G G' : vertexGene, relFunctor F G G' .

**Definition** fun\_arrows\_functor\_ViewOb := composGene.

**Notation** "wv o>gene vu" := (@fun\_arrows\_functor\_ViewOb **\_** **\_** **\_** wv vu)

(at level 40, vu at next level) : poly\_scope.

**Definition** fun\_transf\_ViewObMor (G H: vertexGene) (g: 'Gene( H ~> G )) (H': vertexGene) :

'Gene(H' ~> H) -> 'Gene(H' ~> G) .

**Proof**. exact: ( **fun** h => h o:>gene g ). **Defined**.

(\* **TODO:** REDO GENERAL fun\_transf\_ViewObMor\_Proper \*)

**Global** **Instance** fun\_transf\_ViewObMor\_Proper G H g H' : Proper (equiv ==> equiv) (@fun\_transf\_ViewObMor G H g H').

**Proof**. move. intros ? ? Heq. unfold fun\_transf\_ViewObMor. rewrite -> Heq; reflexivity.

**Qed**.

**Notation** "wv :>gene vu" := (@fun\_transf\_ViewObMor **\_** **\_** vu **\_** wv)

(at level 40, vu at next level) : poly\_scope.

**Definition** typeOf\_functorialCompos\_functor (F : typeOf\_objects\_functor)

(F\_ : typeOf\_arrows\_functor F) :=

**forall** G G' (g : 'Gene( G' ~> G)) G'' (g' : 'Gene( G'' ~> G')) (f : F G),

F\_ **\_** **\_** g' (F\_ **\_** **\_** g f) ==

F\_ **\_** **\_** ( g' o>gene g (\*? or g' :>gene g or g' o:>gene g ?\*) ) f.

**Definition** typeOf\_functorialIdent\_functor (F : typeOf\_objects\_functor)

(F\_ : typeOf\_arrows\_functor F) :=

**forall** G (f : F G), F\_ **\_** **\_** (@identGene G) f == f.

**Record** functor := Functor

{ \_objects\_functor :> typeOf\_objects\_functor ;

\_arrows\_functor :> (\* :> ??? \*) typeOf\_arrows\_functor \_objects\_functor;

\_functorialCompos\_functor : typeOf\_functorialCompos\_functor \_arrows\_functor;

\_functorialIdent\_functor : typeOf\_functorialIdent\_functor \_arrows\_functor;

}.

**Notation** "g o>functor\_ [ F ] f" := (@\_arrows\_functor F **\_** **\_** g f)

(at level 40, f at next level) : poly\_scope.

**Notation** "g o>functor\_ f" := (@\_arrows\_functor **\_** **\_** **\_** g f)

(at level 40, f at next level) : poly\_scope.

**Definition** equiv\_rel\_functor\_ViewOb (G H : vertexGene) : crelation 'Gene( H ~> G ).

**Proof**. exact: equiv.

**Defined**.

(\* (\* no lack for now, unless want uniformity of the (opaque) witness... \*)

**Arguments** equiv\_rel\_functor\_ViewOb /.

\*)

**Definition** functor\_ViewOb (G : vertexGene) : functor.

**Proof**. unshelve eexists.

- (\* typeOf\_objects\_functor \*) intros H. exact: 'Gene( H ~> G ).

- (\* typeOf\_arrows\_functor \*) intros H H'. exists (@fun\_arrows\_functor\_ViewOb G H H').

abstract (typeclasses eauto).

- (\* typeOf\_functorialCompos\_functor \*) abstract (move; intros; exact: composGene\_compos).

- (\* typeOf\_functorialIdent\_functor \*) abstract (move; intros; exact: composGene\_identGene).

**Defined**.

**Definition** \_functorialCompos\_functor' {F : functor} :

**forall** G G' (g : 'Gene( G' ~> G)) G'' (g' : 'Gene( G'' ~> G')) (f : F G),

g' o>functor\_ [ F ] (g o>functor\_ [ F ] f)

== (g' o>functor\_ [ functor\_ViewOb G ] g) o>functor\_ [ F ] f

:= @\_functorialCompos\_functor F.

**Class** relTransf (F E : typeOf\_objects\_functor) (G : vertexGene) : **Type** := RelTransf

{ \_fun\_relTransf : F G -> E G ;

\_congr\_relTransf :> Proper (@equiv **\_** **\_** (@\_equiv\_relType ( F G ))

==> @equiv **\_** **\_** (@\_equiv\_relType ( E G))) \_fun\_relTransf ; }.

**Coercion** \_fun\_relTransf : relTransf >-> Funclass.

**Notation** typeOf\_arrows\_transf F E

:= (**forall** G : vertexGene, relTransf F E G) .

**Definition** typeOf\_natural\_transf (F E : functor)

(ee : typeOf\_arrows\_transf F E) :=

**forall** G G' (g : 'Gene( G' ~> G )) (f : F G),

g o>functor\_[E] (ee G f) == ee G' (g o>functor\_[F] f).

**Record** transf (F : functor) (E : functor) := Transf

{ \_arrows\_transf :> typeOf\_arrows\_transf F E ;

\_natural\_transf : typeOf\_natural\_transf \_arrows\_transf;

}.

**Notation** "f :>transf\_ [ G ] ee" := (@\_arrows\_transf **\_** **\_** ee G f)

(at level 40, ee at next level) : poly\_scope.

**Notation** "f :>transf\_ ee" := (@\_arrows\_transf **\_** **\_** ee **\_** f)

(at level 40, ee at next level) : poly\_scope.

**Definition** transf\_ViewObMor (G : vertexGene) (H : vertexGene) (g : 'Gene( H ~> G )) :

transf (functor\_ViewOb H) (functor\_ViewOb G).

**Proof**. unshelve eexists.

- (\* \_arrows\_transf \*) unshelve eexists.

+ (\* \_fun\_relTransf \*) exact: (fun\_transf\_ViewObMor g).

+ (\* \_congr\_relTransf \*) exact: fun\_transf\_ViewObMor\_Proper.

- (\* \_natural\_transf \*)abstract (move; simpl; intros; exact: composGene\_compos).

**Defined**.

**Definition** \_functorialCompos\_functor'' {F : functor} :

**forall** G G' (g : 'Gene( G' ~> G)) G'' (g' : 'Gene( G'' ~> G')) (f : F G),

g' o>functor\_ [ F ] (g o>functor\_ [ F ] f)

== (g' :>transf\_ (transf\_ViewObMor g)) o>functor\_ [ F ] f

:= @\_functorialCompos\_functor F.

**Record** sieveFunctor (U : vertexGene) : **Type** :=

{ \_functor\_sieveFunctor :> functor ;

\_transf\_sieveFunctor : transf \_functor\_sieveFunctor (functor\_ViewOb U) ; }.

**Lemma** transf\_sieveFunctor\_Proper (U : vertexGene) (UU : sieveFunctor U) H:

Proper (equiv ==> equiv) (\_transf\_sieveFunctor UU H).

apply: \_congr\_relTransf.

**Qed**.

**Notation** "''Sieve' ( G' ~> G | VV )" := (@\_functor\_sieveFunctor G VV G')

(at level 0, format "''Sieve' ( G' ~> G | VV )") : poly\_scope.

**Notation** "h o>sieve\_ v " := (h o>functor\_[@\_functor\_sieveFunctor **\_** **\_**] v)

(at level 40, v at next level, format "h o>sieve\_ v") : poly\_scope.

**Notation** "v :>sieve\_" := (v :>transf\_ (\_transf\_sieveFunctor **\_**)) (at level 40) : poly\_scope.

**Record** preSieve (U : vertexGene) : **Type** :=

{ \_functor\_preSieve :> vertexGene -> **Type**;

\_transf\_preSieve : **forall** G : vertexGene, (\_functor\_preSieve G) -> (functor\_ViewOb U G) ; }.

**Arguments** \_transf\_preSieve {**\_** **\_** **\_**} .

**Notation** "''preSieve' ( G' ~> G | VV )" := (@\_functor\_preSieve G VV G')

(at level 0, format "''preSieve' ( G' ~> G | VV )") : poly\_scope.

**Notation** "v :>preSieve\_" := (@\_transf\_preSieve **\_** **\_** **\_** v) (at level 40) : poly\_scope.

**Global** **Ltac** cbn\_ := cbn -[equiv \_type\_relType \_rel\_relType \_equiv\_relType \_objects\_functor \_arrows\_functor functor\_ViewOb

\_arrows\_transf transf\_ViewObMor \_functor\_sieveFunctor \_functor\_preSieve \_transf\_sieveFunctor \_transf\_preSieve].

**Global** **Ltac** cbn\_equiv := unfold \_rel\_relType, equiv; cbn -[ \_arrows\_functor functor\_ViewOb

\_arrows\_transf transf\_ViewObMor \_functor\_sieveFunctor \_functor\_preSieve \_transf\_sieveFunctor \_transf\_preSieve].

**Global** **Ltac** cbn\_view := cbn -[equiv \_type\_relType \_rel\_relType \_equiv\_relType \_objects\_functor \_arrows\_functor

\_arrows\_transf \_functor\_sieveFunctor \_functor\_preSieve \_transf\_sieveFunctor].

**Global** **Ltac** cbn\_functor := cbn -[equiv \_type\_relType \_rel\_relType \_equiv\_relType functor\_ViewOb

\_arrows\_transf transf\_ViewObMor \_functor\_sieveFunctor \_functor\_preSieve \_transf\_sieveFunctor \_transf\_preSieve].

**Global** **Ltac** cbn\_transf := cbn -[equiv \_type\_relType \_rel\_relType \_equiv\_relType \_arrows\_functor functor\_ViewOb

transf\_ViewObMor \_functor\_sieveFunctor \_functor\_preSieve \_transf\_sieveFunctor \_transf\_preSieve].

**Global** **Ltac** cbn\_sieve := cbn -[equiv \_type\_relType \_rel\_relType \_equiv\_relType functor\_ViewOb

transf\_ViewObMor ].

**Tactic** **Notation** "cbn\_" "in" hyp\_list(H) := cbn -[equiv \_type\_relType \_rel\_relType \_equiv\_relType \_objects\_functor \_arrows\_functor functor\_ViewOb

\_arrows\_transf transf\_ViewObMor \_functor\_sieveFunctor \_functor\_preSieve \_transf\_sieveFunctor \_transf\_preSieve] in H.

**Tactic** **Notation** "cbn\_equiv" "in" hyp\_list(H) := unfold \_rel\_relType, equiv in H; cbn -[ \_arrows\_functor functor\_ViewOb

\_arrows\_transf transf\_ViewObMor \_functor\_sieveFunctor \_functor\_preSieve \_transf\_sieveFunctor \_transf\_preSieve] in H.

**Tactic** **Notation** "cbn\_view" "in" hyp\_list(H) := cbn -[equiv \_type\_relType \_rel\_relType \_equiv\_relType \_objects\_functor \_arrows\_functor

\_arrows\_transf \_functor\_sieveFunctor \_functor\_preSieve \_transf\_sieveFunctor \_transf\_preSieve] in H.

**Tactic** **Notation** "cbn\_functor" "in" hyp\_list(H) := cbn -[equiv \_type\_relType \_rel\_relType \_equiv\_relType functor\_ViewOb

\_arrows\_transf transf\_ViewObMor \_functor\_sieveFunctor \_functor\_preSieve \_transf\_sieveFunctor \_transf\_preSieve] in H.

**Tactic** **Notation** "cbn\_transf" "in" hyp\_list(H) := cbn -[equiv \_type\_relType \_rel\_relType \_equiv\_relType \_arrows\_functor functor\_ViewOb

transf\_ViewObMor \_functor\_sieveFunctor \_functor\_preSieve \_transf\_sieveFunctor \_transf\_preSieve] in H.

**Tactic** **Notation** "cbn\_sieve" "in" hyp\_list(H) := cbn -[equiv \_type\_relType \_rel\_relType \_equiv\_relType functor\_ViewOb

transf\_ViewObMor ] in H.

**End** GENE.

**Module** **Type** COMOD (Gene : GENE).

**Import** Gene.

**Ltac** tac\_unsimpl := repeat

lazymatch goal with

| [ |- context [@fun\_transf\_ViewObMor ?G ?H ?g ?H' ?h] ] =>

change (@fun\_transf\_ViewObMor G H g H' h) with

(h :>transf\_ (transf\_ViewObMor g))

| [ |- context [@fun\_arrows\_functor\_ViewOb ?U ?V ?W ?wv ?vu] ] =>

change (@fun\_arrows\_functor\_ViewOb U V W wv vu) with

(wv o>functor\_[functor\_ViewOb U] vu)

(\* no lack\*)

| [ |- context [@equiv\_rel\_functor\_ViewOb ?G ?H ?x ?y] ] =>

change (@equiv\_rel\_functor\_ViewOb G H x y) with

(@equiv **\_** **\_** (@\_equiv\_relType ( (functor\_ViewOb G) H )) x y)

(\* | [ |- context [@equiv\_rel\_arrowSieve ?G ?G' ?g ?H ?x ?y] ] =>

change (@equiv\_rel\_arrowSieve G G' g H x y) with

(@equiv \_ (@\_rel\_relType ( (arrowSieve g) H )) x y) \*)

**end**.

**Definition** transf\_Compos :

**forall** (F F'' F' : functor) (ff\_ : transf F'' F') (ff' : transf F' F),

transf F'' F.

**Proof**. intros. unshelve eexists.

- intros G. unshelve eexists. intros f. exact:((f :>transf\_ ff\_) :>transf\_ ff').

abstract(solve\_proper).

(\* exists (Basics.compose (ff' G) (ff\_ G) ). abstract(typeclasses eauto). \*)

- abstract (move; cbn\_; intros; (\* unfold Basics.compose; \*)

rewrite -> \_natural\_transf , \_natural\_transf; reflexivity).

**Defined**.

**Definition** transf\_Ident :

**forall** (F : functor), transf F F.

**Proof**. intros. unshelve eexists.

- intros G. exists id.

abstract(simpl\_relation).

- abstract (move; cbn\_; intros; reflexivity).

**Defined**.

**Definition** typeOf\_commute\_sieveTransf

(G : vertexGene) (V1 V2 : sieveFunctor G) (vv : transf V1 V2) : **Type** :=

**forall** (H : vertexGene) (v : 'Sieve( H ~> G | V1 )),

(v :>transf\_ vv) :>sieve\_ == v :>sieve\_ .

**Record** sieveTransf G (V1 V2 : sieveFunctor G) : **Type** :=

{ \_transf\_sieveTransf :> transf V1 V2 ;

\_commute\_sieveTransf : typeOf\_commute\_sieveTransf \_transf\_sieveTransf} .

**Instance** fun\_transf\_ViewObMor\_measure {G H: vertexGene} {g: 'Gene( H ~> G )} {H': vertexGene}:

@Measure 'Gene(H' ~> H) 'Gene(H' ~> G) (@fun\_transf\_ViewObMor G H g H') := { }.

**Definition** sieveTransf\_Compos :

**forall** U (F F'' F' : sieveFunctor U) (ff\_ : sieveTransf F'' F') (ff' : sieveTransf F' F),

sieveTransf F'' F.

**Proof**. intros. unshelve eexists.

- exact: (transf\_Compos ff\_ ff').

- abstract(move; intros; cbn\_transf; autounfold; do 2 rewrite -> \_commute\_sieveTransf; reflexivity).

**Defined**.

**Definition** sieveTransf\_Ident :

**forall** U (F : sieveFunctor U) , sieveTransf F F.

**Proof**. intros. unshelve eexists.

- exact: (transf\_Ident F).

- abstract(move; intros; reflexivity).

**Defined**.

**Definition** identSieve (G: vertexGene) : sieveFunctor G.

unshelve eexists.

exact: (functor\_ViewOb G).

exact: (transf\_Ident (functor\_ViewOb G)).

**Defined**.

**Definition** sieveTransf\_identSieve :

**forall** U (F : sieveFunctor U) , sieveTransf F (identSieve U).

**Proof**. intros. unshelve eexists.

- exact: (\_transf\_sieveFunctor F).

- abstract(move; intros; reflexivity).

**Defined**.

(\* TODO MERE WITH sieveTransf\_identSieve \*)

**Lemma** sieveTransf\_sieveFunctor G (VV : sieveFunctor G) :

sieveTransf VV (identSieve G).

**Proof**. unshelve eexists. exact: \_transf\_sieveFunctor.

- (\* \_commute\_sieveTransf \*) abstract(move; reflexivity).

**Defined**.

**Record** sieveEquiv G (V1 V2 : sieveFunctor G) : **Type** :=

{ \_sieveTransf\_sieveEquiv :> sieveTransf V1 V2 ;

\_revSieveTransf\_sieveEquiv : sieveTransf V2 V1 ;

\_injProp\_sieveEquiv : **forall** H v, (v :>transf\_[H] \_revSieveTransf\_sieveEquiv )

:>transf\_ \_sieveTransf\_sieveEquiv == v ;

\_surProp\_sieveEquiv : **forall** H v, (v :>transf\_[H] \_sieveTransf\_sieveEquiv )

:>transf\_ \_revSieveTransf\_sieveEquiv == v } .

**Definition** rel\_sieveEquiv G : crelation (sieveFunctor G) := **fun** V1 V2 => sieveEquiv V1 V2.

**Instance** equiv\_sieveEquiv G: Equivalence (@rel\_sieveEquiv G ).

unshelve eexists.

- intros V1. unshelve eexists. exact (sieveTransf\_Ident **\_**). exact (sieveTransf\_Ident **\_**).

abstract (reflexivity). abstract (reflexivity).

- intros V1 V2 Hseq. unshelve eexists.

exact (\_revSieveTransf\_sieveEquiv Hseq). exact (\_sieveTransf\_sieveEquiv Hseq).

abstract(intros; rewrite -> \_surProp\_sieveEquiv; reflexivity).

abstract(intros; rewrite -> \_injProp\_sieveEquiv; reflexivity).

- intros V1 V2 V3 Hseq12 Hseq23. unshelve eexists. exact (sieveTransf\_Compos Hseq12 Hseq23).

exact (sieveTransf\_Compos (\_revSieveTransf\_sieveEquiv Hseq23) (\_revSieveTransf\_sieveEquiv Hseq12)).

abstract(intros; cbn\_transf; rewrite -> \_injProp\_sieveEquiv; rewrite -> \_injProp\_sieveEquiv; reflexivity).

abstract(intros; cbn\_transf; rewrite -> \_surProp\_sieveEquiv; rewrite -> \_surProp\_sieveEquiv; reflexivity).

**Defined**.

**Section** interSieve.

**Section** Section1.

**Variables** (G : vertexGene) (VV : sieveFunctor G)

(G' : vertexGene) (g : 'Gene( G' ~> G ))

(UU : sieveFunctor G').

**Record** type\_interSieve H :=

{ \_factor\_interSieve : 'Sieve( H ~> **\_** | UU ) ;

\_whole\_interSieve : 'Sieve( H ~> **\_** | VV ) ;

\_wholeProp\_interSieve : \_whole\_interSieve :>sieve\_

== (\_factor\_interSieve :>sieve\_) o>functor\_[functor\_ViewOb G] g }.

**Definition** rel\_interSieve H : crelation (type\_interSieve H).

intros v v'. exact (((\_factor\_interSieve v == \_factor\_interSieve v') \*

(\_whole\_interSieve v == \_whole\_interSieve v')) %type ).

**Defined**.

**Instance** equiv\_interSieve H : Equivalence (@rel\_interSieve H).

abstract(unshelve eexists;

[ (move; intros; move; split; reflexivity)

| (move; intros ? ? [? ?]; move; intros; split; symmetry; assumption)

| (move; intros ? ? ? [? ?] [? ?]; move; intros; split; etransitivity; eassumption)]).

**Qed**.

**Definition** interSieve : sieveFunctor G'.

**Proof**. unshelve eexists.

{ (\* functor \*) unshelve eexists.

- (\* typeOf\_objects\_functor \*) intros H.

+ (\* relType \*) unshelve eexists. exact (type\_interSieve H).

exact (@rel\_interSieve H).

exact (@equiv\_interSieve H).

- (\* typeOf\_arrows\_functor \*) unfold typeOf\_arrows\_functor. intros H H'.

+ (\* relFunctor \*) unshelve eexists.

\* (\* -> \*) cbn\_. intros h vg'. unshelve eexists.

exact: (h o>sieve\_ (\_factor\_interSieve vg')).

exact: (h o>sieve\_ (\_whole\_interSieve vg')).

abstract(cbn\_; tac\_unsimpl; rewrite <- 2!\_natural\_transf;

rewrite -> \_wholeProp\_interSieve, \_functorialCompos\_functor'; reflexivity).

\* (\* Proper \*) abstract(move; autounfold;

intros h1 h2 Heq\_h vg'1 vg'2; case => /= Heq\_vg' Heq\_vg'0;

split; cbn\_; rewrite -> Heq\_h; [rewrite -> Heq\_vg' | rewrite -> Heq\_vg'0]; reflexivity).

- (\* typeOf\_functorialCompos\_functor \*) abstract(move; intros; autounfold; split; cbn\_;

rewrite -> \_functorialCompos\_functor; reflexivity).

- (\* typeOf\_functorialIdent\_functor \*) abstract(move; intros; autounfold; split; cbn\_;

rewrite -> \_functorialIdent\_functor; reflexivity). }

{ (\* transf \*) unshelve eexists.

- (\* typeOf\_arrows\_transf \*) intros H. unshelve eexists.

+ (\* -> \*) cbn\_; intros vg'. exact: ((\_factor\_interSieve vg') :>sieve\_).

+ (\* Proper \*) abstract(move; autounfold; cbn\_;

intros vg'1 vg'2; case => /= Heq0 Heq; rewrite -> Heq0; reflexivity).

- (\* typeOf\_natural\_transf \*) abstract(move; cbn -[\_arrows\_functor]; intros;

rewrite -> \_natural\_transf; reflexivity). }

**Defined**.

**Lemma** transf\_interSieve\_Eq H (v : 'Sieve(H ~> **\_** | interSieve )) :

((\_factor\_interSieve v) :>sieve\_ ) == (v :>sieve\_ ) .

**Proof**. reflexivity.

**Qed**.

**Global** **Instance** whole\_interSieve\_Proper H : Proper (equiv ==> equiv)

(@\_whole\_interSieve H : 'Sieve( H ~> **\_** | interSieve ) -> 'Sieve( H ~> **\_** | VV )).

**Proof**. move. cbn\_. intros v1 v2 [Heq Heq']. exact Heq'.

**Qed**.

**Global** **Instance** factor\_interSieve\_Proper H : Proper (equiv ==> equiv)

(@\_factor\_interSieve H : 'Sieve( H ~> **\_** | interSieve ) -> 'Sieve( H ~> **\_** | UU )).

**Proof**. move. cbn\_. intros v1 v2 [Heq Heq']. exact Heq.

**Qed**.

**Definition** interSieve\_projWhole : transf interSieve VV.

**Proof**. unshelve eexists. unshelve eexists.

- (\* -> \*) exact: \_whole\_interSieve.

- (\* Proper \*) exact: whole\_interSieve\_Proper. (\* abstract (typeclasses eauto). \*)

- (\* typeOf\_natural\_transf \*) abstract(intros H H' h f; cbn\_; reflexivity).

**Defined**.

**Definition** interSieve\_projFactor : sieveTransf interSieve UU.

**Proof**. unshelve eexists. unshelve eexists. unshelve eexists.

- (\* -> \*) exact: \_factor\_interSieve.

- (\* Proper \*) exact: factor\_interSieve\_Proper. (\* abstract (typeclasses eauto). \*)

- (\* typeOf\_natural\_transf \*) abstract(intros H H' h f; cbn\_; reflexivity).

- (\* \_commute\_sieveTransf \*) abstract(move; cbn\_; intros; reflexivity).

**Defined**.

**End** Section1.

**Definition** pullSieve G VV G' g := @interSieve G VV G' g (identSieve G').

**Definition** meetSieve G VV UU := @interSieve G VV G (@identGene G) UU.

**Definition** pullSieve\_projWhole G VV G' g :

transf (@pullSieve G VV G' g) VV

:= (@interSieve\_projWhole G VV G' g (identSieve G')).

**Definition** pullSieve\_projFactor G VV G' g :

sieveTransf (@pullSieve G VV G' g) (identSieve G')

:= (@interSieve\_projFactor G VV G' g (identSieve G')).

**Definition** meetSieve\_projFactor G VV UU :

sieveTransf (@meetSieve G VV UU) UU := @interSieve\_projFactor G VV G (@identGene G) UU .

**Definition** meetSieve\_projWhole G VV UU :

sieveTransf (@meetSieve G VV UU) VV.

exists (interSieve\_projWhole **\_** **\_** **\_**).

intros H v; simpl. rewrite -> \_wholeProp\_interSieve.

(\* HERE \*) abstract(exact: identGene\_composGene).

**Defined**.

**End** interSieve.

**Existing** **Instance** whole\_interSieve\_Proper.

**Existing** **Instance** factor\_interSieve\_Proper.

**Section** sumSieve.

**Section** Section1.

**Variables** (G : vertexGene) (VV : preSieve G).

**Record** typeOf\_outer\_sumSieve :=

{ \_object\_typeOf\_outer\_sumSieve :> vertexGene ;

\_arrow\_typeOf\_outer\_sumSieve :> 'preSieve( \_object\_typeOf\_outer\_sumSieve ~> G | VV ) }.

(\* higher/congruent structure is possible... \*)

**Variables** (WP\_ : **forall** (object\_: vertexGene) (outer\_: 'preSieve( object\_ ~> G | VV )),

sieveFunctor object\_).

(\*

(\* higher/congruent structure is possible... \*)

**Definition** typeOf\_sieveCongr :=

**forall** (object\_ : vertexGene)

(outer\_ outer\_' : 'preSieve( object\_ ~> **\_** | VV )),

outer\_ == outer\_' ->

sieveEquiv (WP\_ outer\_) (WP\_ outer\_').

**Variables** WP\_congr : typeOf\_sieveCongr. \*)

**Record** type\_sumSieve H :=

{ \_object\_sumSieve : vertexGene ;

\_outer\_sumSieve : 'preSieve( \_object\_sumSieve ~> G | VV ) ;

\_inner\_sumSieve : 'Sieve( H ~> **\_** | WP\_ \_outer\_sumSieve ) }.

**Inductive** rel\_sumSieve H (wv : type\_sumSieve H) : type\_sumSieve H -> **Type** :=

| Rel\_sumSieve : **forall**

(inner': (WP\_ (\_outer\_sumSieve wv)) H),

(\* higher/congruent structure is possible... \*)

(\* inner' :>transf\_ (WP\_congr Heq\_outer) == (\_inner\_sumSieve wv) -> \*)

inner' == (\_inner\_sumSieve wv) ->

rel\_sumSieve wv

{| \_object\_sumSieve := **\_** ;

\_outer\_sumSieve := **\_** ;

\_inner\_sumSieve := inner' |}.

Instance rel\_sumSieve\_Equivalence H : Equivalence (@rel\_sumSieve H).

abstract(unshelve eexists;

[ (intros [object\_wv outer\_wv inner\_wv]; constructor; reflexivity)

| (\* intros wv1 wv2 []. \*) (intros [object\_wv1 outer\_wv1 inner\_wv1] [object\_wv2 outer\_wv2 inner\_wv2] [];

constructor; symmetry; assumption)

| (intros wv1 wv2 wv3 Heq12 Heq23; destruct Heq23 as [ inner3 Heq23'];

destruct Heq12 as [ inner2 Heq12']; simpl; constructor; simpl;

rewrite -> Heq23'; simpl; rewrite -> Heq12'; simpl; reflexivity)]).

**Qed**.

(\* **TODO:** sumSieve\_projOuter : sumSieve -> UU \*)

**Definition** sumSieve : sieveFunctor G.

**Proof**. unshelve eexists.

{ (\* functor \*) unshelve eexists.

- (\* typeOf\_objects\_functor \*) intros H.

+ (\* relType \*) unshelve eexists. exact (type\_sumSieve H).

+ (\* Setoid \*) exact (@rel\_sumSieve H).

(\* exists (equiv @@ (@compos\_sumSieve H))%signature. \*)

+ (\* Equivalence \*) exact: rel\_sumSieve\_Equivalence.

- (\* typeOf\_arrows\_functor \*) move. intros H H'.

(\* relFunctor \*) unshelve eexists.

+ (\* -> \*) simpl. intros h wv. unshelve eexists.

exact: (\_object\_sumSieve wv). exact: (\_outer\_sumSieve wv).

exact: (h o>sieve\_ \_inner\_sumSieve wv).

+ (\* Proper \*) abstract(move; autounfold; simpl;

intros h1 h2 Heq\_h [object\_wv1 outer\_wv1 inner\_wv1] wv2 Heq; tac\_unsimpl;

case: wv2 / Heq => /= [ inner\_wv2 Heq12']; constructor; simpl;

rewrite -> Heq\_h , Heq12'; reflexivity).

- (\* typeOf\_functorialCompos\_functor \*) abstract(intros H H' h H'' h' [object\_wv outer\_wv inner\_wv];

simpl; constructor; simpl; rewrite -> \_functorialCompos\_functor; reflexivity).

- (\* typeOf\_functorialIdent\_functor \*) abstract(intros H [object\_wv outer\_wv inner\_wv];

simpl; constructor; simpl; rewrite -> \_functorialIdent\_functor; reflexivity). }

{ (\* transf \*) unshelve eexists.

- (\* typeOf\_arrows\_transf \*) intros H. unshelve eexists.

+ (\* -> \*) simpl; intros wv. exact: ((\_inner\_sumSieve wv :>sieve\_) o>functor\_ (\_outer\_sumSieve wv :>preSieve\_)).

+ (\* Proper \*) abstract(move; autounfold; simpl;

intros wv1 wv2 Heq; tac\_unsimpl;

case: wv2 / Heq => /= [ inner\_wv2 Heq12']; tac\_unsimpl;

rewrite -> Heq12'; reflexivity).

- (\* typeOf\_natural\_transf \*) abstract(move; cbn\_functor; move; cbn\_functor; intros H H' h wv;

rewrite -> \_functorialCompos\_functor'; rewrite -> \_natural\_transf; reflexivity). }

**Defined**.

**End** Section1.

**Section** genSieve.

**Definition** genSieve (U : vertexGene) (UU : preSieve U)

:= (sumSieve (**fun** (object: vertexGene) (outer: 'preSieve( object ~> U | UU )) => identSieve object ) ).

**Definition** preSieveTransf\_unit (U : vertexGene) (UU : preSieve U) :

**forall** G (outer: 'preSieve( G ~> U | UU )), 'Sieve( G ~> U | (genSieve UU) ) .

**Proof**. intros. exists **\_** outer. exact: (identGene). **Defined**.

**Definition** transf\_of\_preSieveTransf

(U : vertexGene) (UU : preSieve U) (V : vertexGene) (VV : sieveFunctor V)

(ff : **forall** G, 'preSieve( G ~> U | UU ) -> 'Sieve( G ~> V | VV) ) :

transf (genSieve UU) VV .

**Proof**. unshelve eexists. unshelve eexists.

- (\* -> \*) intros u. exact ( (\_inner\_sumSieve u) o>functor\_ (ff **\_** (\_outer\_sumSieve u)) ).

- (\* Proper \*) abstract(move; move => u1 u2 [inner\_u Heq]; cbn\_transf; rewrite -> Heq; reflexivity).

- (\* typeOf\_natural\_transf \*) abstract(intros H H' h u; cbn\_sieve; rewrite -> \_functorialCompos\_functor'; reflexivity).

**Defined**.

**Definition** preSieveTransf\_of\_transf (U : vertexGene) (UU : preSieve U) (V : vertexGene) (VV : sieveFunctor V)

(ff : transf (genSieve UU) VV ) := (**fun** G (outer: 'preSieve( G ~> U | UU )) =>

((preSieveTransf\_unit outer) :>transf\_ ff) ).

**Lemma** transf\_of\_preSieveTransf\_surj (U : vertexGene) (UU : preSieve U) (V : vertexGene) (VV : sieveFunctor V)

(ff : transf (genSieve UU) VV ) :

**forall** G (outer: 'Sieve( G ~> U | (genSieve UU) )),

outer :>transf\_ ff == outer :>transf\_ transf\_of\_preSieveTransf (preSieveTransf\_of\_transf ff) .

**Proof**. intros . unfold preSieveTransf\_of\_transf. cbn\_sieve.

rewrite -> \_natural\_transf. apply: \_congr\_relTransf. split. cbn\_sieve. apply: identGene\_composGene.

**Qed**.

**Definition** typeOf\_commute\_presieveTransfArrow

(U : vertexGene) (UU : preSieve U) (V : vertexGene) (VV : sieveFunctor V) (uv : 'Gene( U ~> V))

(ff : **forall** G, 'preSieve( G ~> U | UU ) -> 'Sieve( G ~> V | VV) ) : **Type** :=

**forall** (H : vertexGene) (u : 'preSieve( H ~> U | UU )),

(ff **\_** u ) :>sieve\_ == (u :>preSieve\_) o>functor\_[functor\_ViewOb **\_**] uv .

**Record** presieveTransfArrow

(U : vertexGene) (UU : preSieve U) (V : vertexGene) (VV : sieveFunctor V) (uv : 'Gene( U ~> V)) : **Type** :=

{ \_transf\_presieveTransfArrow :> **forall** G, 'preSieve( G ~> U | UU ) -> 'Sieve( G ~> V | VV);

\_commute\_presieveTransfArrow : typeOf\_commute\_presieveTransfArrow uv \_transf\_presieveTransfArrow} .

**Definition** typeOf\_commute\_sieveTransfArrow

(G1 : vertexGene) (V1: sieveFunctor G1) (G2 : vertexGene) (V2: sieveFunctor G2)

(g12 : 'Gene( G1 ~> G2)) (vv : transf V1 V2) : **Type** :=

**forall** (H : vertexGene) (v : 'Sieve( H ~> G1 | V1 )),

(v :>transf\_ vv) :>sieve\_ == (v :>sieve\_) o>functor\_[functor\_ViewOb **\_**] g12.

**Record** sieveTransfArrow (G1 : vertexGene) (V1: sieveFunctor G1) (G2 : vertexGene) (V2: sieveFunctor G2)

(g12 : 'Gene( G1 ~> G2)) : **Type** :=

{ \_transf\_sieveTransfArrow :> transf V1 V2 ;

\_commute\_sieveTransfArrow : typeOf\_commute\_sieveTransfArrow g12 \_transf\_sieveTransfArrow} .

**Definition** sieveTransfArrow\_of\_preSieveTransf

(U : vertexGene) (UU : preSieve U) (V : vertexGene) (VV : sieveFunctor V) (uv : 'Gene( U ~> V))

(ff : presieveTransfArrow UU VV uv) : sieveTransfArrow (genSieve UU) VV uv.

**Proof**. exists (transf\_of\_preSieveTransf ff).

abstract(move; intros; cbn\_sieve; rewrite <- \_functorialCompos\_functor', <- \_natural\_transf;

rewrite -> \_commute\_presieveTransfArrow; reflexivity).

**Defined**.

**Definition** preSieveTransf\_of\_sieveTransfArrow

(U : vertexGene) (UU : preSieve U) (V : vertexGene) (VV : sieveFunctor V) (uv : 'Gene( U ~> V))

(ff : sieveTransfArrow (genSieve UU) VV uv) : presieveTransfArrow UU VV uv.

**Proof**. exists (preSieveTransf\_of\_transf ff).

abstract(move; intros; unfold preSieveTransf\_of\_transf;

rewrite -> \_commute\_sieveTransfArrow; cbn\_sieve; rewrite -> \_functorialIdent\_functor; reflexivity).

**Defined**.

**Definition** sieveTransfArrow\_Compos :

**forall** U U' U'' F F'' F' (u\_ : 'Gene( U'' ~> U')) (ff\_ : sieveTransfArrow F'' F' u\_)

(u' : 'Gene( U' ~> U)) (ff' : sieveTransfArrow F' F u'),

sieveTransfArrow F'' F (u\_ o>gene u').

**Proof**. intros. unshelve eexists.

- exact: (transf\_Compos ff\_ ff').

- abstract(move; intros; cbn\_transf; do 2 rewrite -> \_commute\_sieveTransfArrow;

rewrite <- \_functorialCompos\_functor'; reflexivity).

**Defined**.

**End** genSieve.

**Definition** sumSieve\_projOuter :

**forall** (U : vertexGene) (UU : preSieve U)

(VV\_ : **forall** (H: vertexGene) (outer\_: 'preSieve( H ~> U | UU )), sieveFunctor H),

sieveTransf (sumSieve VV\_) (genSieve UU).

**Proof**. unshelve eexists. unshelve eexists.

- intros K. unshelve eexists.

+ (\* \_fun\_relTransf \*) intros wv. eexists. exact (\_outer\_sumSieve wv). exact (\_inner\_sumSieve wv :>sieve\_).

+ (\* \_congr\_relTransf \*) abstract(move; intros wv1 wv2 [ inner\_wv2 Heq\_inner\_wv2];

cbn\_transf; split;cbn\_transf; rewrite -> Heq\_inner\_wv2; reflexivity).

- (\* \_natural\_transf \*) abstract(move; intros; cbn\_sieve; split; cbn\_sieve; rewrite -> \_natural\_transf; reflexivity ).

- (\* \_commute\_sieveTransf \*) abstract(move; intros; simpl; reflexivity).

**Defined**.

**Definition** sumSieve\_sectionPull :

**forall** (U : vertexGene) (UU : preSieve U)

(VV\_ : **forall** (H: vertexGene) (outer\_: 'preSieve( H ~> U | UU )), sieveFunctor H)

(H: vertexGene)

(u: 'preSieve( H ~> **\_** | UU )),

sieveTransf (VV\_ H u)

(pullSieve (sumSieve VV\_) (u:>preSieve\_)) .

**Proof**. unshelve eexists. unshelve eexists.

- intros K. unshelve eexists.

+ (\* \_fun\_relTransf \*) intros v. unshelve eexists.

\* (\* \_factor\_interSieve \*)exact: ((v :>sieve\_) ).

(\* \_whole\_interSieve \*) unshelve eexists.

\* (\* \_object\_sumSieve \*) exact: H.

\* (\* \_outer\_sumSieve \*) exact: u.

\* (\* \_inner\_sumSieve \*) exact: v.

\* (\* \_wholeProp\_interSieve \*) abstract(simpl; reflexivity).

+ (\* \_congr\_relTransf \*) abstract(move; intros v1 v2 Heq\_v; split; autounfold; simpl;

first (rewrite -> Heq\_v; reflexivity); split; autounfold; simpl;

rewrite -> Heq\_v; reflexivity).

- (\* \_natural\_transf \*) abstract(move; intros; split; cbn\_transf; last reflexivity;

cbn\_sieve; rewrite -> \_natural\_transf; reflexivity).

- (\* \_commute\_sieveTransf \*) abstract(move; intros; simpl; reflexivity).

**Defined**.

**Definition** sumSieve\_section:

**forall** (U : vertexGene) (UU : preSieve U)

(VV\_ : **forall** (H: vertexGene) (outer\_: 'preSieve( H ~> U | UU )), sieveFunctor H)

(H: vertexGene)

(u: 'preSieve( H ~> **\_** | UU )),

transf (VV\_ H u) (sumSieve VV\_) .

**Proof**. intros. exact: (transf\_Compos (sumSieve\_sectionPull **\_** **\_**) (pullSieve\_projWhole **\_** **\_**) ).

**Defined**.

**End** sumSieve.

**Section** sumPreSieve.

**Section** Section1.

**Variables** (G : vertexGene) (VV : preSieve G).

**Record** typeOf\_outer\_sumPreSieve :=

{ \_object\_typeOf\_outer\_sumPreSieve :> vertexGene ;

\_arrow\_typeOf\_outer\_sumPreSieve :> 'preSieve( \_object\_typeOf\_outer\_sumPreSieve ~> G | VV ) }.

(\* higher/congruent structure is possible... \*)

**Variables** (WP\_ : **forall** (object\_: vertexGene) (outer\_: 'preSieve( object\_ ~> G | VV )),

preSieve object\_).

**Record** type\_sumPreSieve H :=

{ \_object\_sumPreSieve : vertexGene ;

\_outer\_sumPreSieve : 'preSieve( \_object\_sumPreSieve ~> G | VV ) ;

\_inner\_sumPreSieve : 'preSieve( H ~> **\_** | WP\_ \_outer\_sumPreSieve ) }.

**Inductive** rel\_sumPreSieve H (wv : type\_sumPreSieve H) : type\_sumPreSieve H -> **Type** :=

| Rel\_sumPreSieve : **forall**

(inner': (WP\_ (\_outer\_sumPreSieve wv)) H),

inner' :>preSieve\_ == (\_inner\_sumPreSieve wv) :>preSieve\_ ->

rel\_sumPreSieve wv

{| \_object\_sumPreSieve := **\_** ;

\_outer\_sumPreSieve := **\_** ;

\_inner\_sumPreSieve := inner' |}.

Instance rel\_sumPreSieve\_Equivalence H : Equivalence (@rel\_sumPreSieve H).

abstract(unshelve eexists;

[ (intros [object\_wv outer\_wv inner\_wv]; constructor; reflexivity)

| (\* intros wv1 wv2 []. \*) (intros [object\_wv1 outer\_wv1 inner\_wv1] [object\_wv2 outer\_wv2 inner\_wv2] [];

constructor; symmetry; assumption)

| (intros wv1 wv2 wv3 Heq12 Heq23; destruct Heq23 as [ inner3 Heq23'];

destruct Heq12 as [ inner2 Heq12']; simpl; constructor; simpl;

rewrite -> Heq23'; simpl; rewrite -> Heq12'; simpl; reflexivity)]).

**Qed**.

(\* **TODO:** sumPreSieve\_projOuter : sumPreSieve -> UU \*)

**Definition** sumPreSieve : preSieve G.

**Proof**.

unshelve eexists.

- (\* typeOf\_objects\_functor \*) intros H.

+ exact (type\_sumPreSieve H).

- (\* typeOf\_arrows\_transf \*) intros H.

+ (\* -> \*) simpl; intros wv. exact: ((\_inner\_sumPreSieve wv :>preSieve\_) o>functor\_ (\_outer\_sumPreSieve wv :>preSieve\_)).

**Defined**.

**Definition** sumPreSieve\_projOuter : presieveTransfArrow (sumPreSieve) (genSieve VV) (identGene).

**Proof**. unshelve eexists.

- intros H uv. exists **\_** (\_outer\_sumPreSieve uv). exact ((\_inner\_sumPreSieve uv) :>preSieve\_).

- abstract(move; intros; cbn\_sieve; rewrite <- \_functorialCompos\_functor';

apply: \_congr\_relFunctor; first reflexivity; symmetry; exact: identGene\_composGene).

**Defined**.

**End** Section1.

**End** sumPreSieve.

**Section** sumPullSieve.

**Section** Section1.

**Variables** (G : vertexGene) (VV : preSieve G).

(\* \*)

**Variables** (famVertex\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

vertexGene).

**Variables** (famArrow\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

'Gene( object ~> famVertex\_ outer )).

**Variables** (famSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

sieveFunctor (famVertex\_ outer)).

**Variables** (famInterPreSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

preSieve object).

**Definition** sumPullSieve := @sumSieve G VV (**fun** object outer => interSieve (famSieve\_ outer) (famArrow\_ outer) (genSieve (famInterPreSieve\_ outer)) ).

**Definition** sumPullSieve\_projSumPreSieve :

sieveTransf sumPullSieve (genSieve (sumPreSieve famInterPreSieve\_)).

**Proof**. unshelve eexists. unshelve eexists.

- intros K. unshelve eexists.

+ (\* \_fun\_relTransf \*) intros wv. eexists.

\* { unshelve eexists; cycle 1. exact (\_outer\_sumSieve wv). exact (\_outer\_sumSieve (\_factor\_interSieve (\_inner\_sumSieve wv))). }

simpl. exact (\_inner\_sumSieve (\_factor\_interSieve (\_inner\_sumSieve wv))).

+ (\* \_congr\_relTransf \*) abstract (move; intros wv1 wv2 [ inner\_wv2 [[outer\_factor\_inner\_wv2 Heq\_inner\_factor\_inner\_wv2\_] Heq\_whole\_inner\_wv2]];

cbn\_transf; split; cbn\_transf; rewrite -> Heq\_inner\_factor\_inner\_wv2\_; reflexivity).

- (\* \_natural\_transf \*) abstract(move; intros; cbn\_sieve; split; cbn\_sieve; reflexivity).

- (\* \_commute\_sieveTransf \*) abstract(move; intros; cbn\_sieve; rewrite -> \_functorialCompos\_functor'; reflexivity).

**Defined**.

**End** Section1.

**End** sumPullSieve.

**Definition** typeOf\_commute\_preSieveTransf

(G : vertexGene) (V1 V2 : preSieve G) (vv : **forall** G : vertexGene, V1 G -> V2 G) : **Type** :=

**forall** (H : vertexGene) (v : 'preSieve( H ~> G | V1 )),

(vv **\_** v ) :>preSieve\_ == v :>preSieve\_ .

**Record** preSieveTransf G (V1 V2 : preSieve G) : **Type** :=

{ \_transf\_preSieveTransf :> **forall** G : vertexGene, V1 G -> V2 G ;

\_commute\_preSieveTransf : typeOf\_commute\_preSieveTransf \_transf\_preSieveTransf} .

**Notation** "f :>preSieveTransf\_ ee" := (@\_transf\_preSieveTransf **\_** **\_** **\_** ee **\_** f)

(at level 40, ee at next level) : poly\_scope.

**Lemma** sumSieve\_congrTransf (G : vertexGene) (UU1 : preSieve G)

G' ( UU2 : preSieve G')

(uu : **forall** G : vertexGene, UU1 G -> UU2 G)

(VV1\_ : **forall** H : vertexGene, 'preSieve( H ~> **\_** | UU1 ) -> sieveFunctor H)

(VV2\_ : **forall** H : vertexGene, 'preSieve( H ~> **\_** | UU2 ) -> sieveFunctor H)

(vv\_ : **forall** (H: vertexGene) (u1: 'preSieve( H ~> **\_** | UU1 )),

sieveTransf (VV1\_ **\_** u1) (VV2\_ **\_** (uu **\_** u1 ))) :

transf (sumSieve VV1\_ ) (sumSieve VV2\_).

**Proof**. unshelve eexists.

- (\* \_arrows\_transf \*) intros K. unshelve eexists.

(\* \_fun\_relTransf \*) intros vu. unshelve eexists.

(\* \_object\_sumSieve \*) exact: (\_object\_sumSieve vu).

(\* \_outer\_sumSieve \*) exact: (uu **\_** (\_outer\_sumSieve vu ) ).

(\* \_inner\_sumSieve \*) exact: (\_inner\_sumSieve vu :>transf\_ (vv\_ **\_** **\_**)).

(\* \_congr\_relTransf \*) abstract(move; intros vu1 vu2 [ inner\_vu2 Heq\_inner\_vu2];

simpl; constructor; simpl; rewrite -> Heq\_inner\_vu2; reflexivity).

- (\* \_natural\_transf \*) abstract(intros K K' k vvu; cbn\_sieve;

constructor; simpl; rewrite -> \_natural\_transf; reflexivity).

**Defined**.

**Lemma** sumSieve\_congr (G : vertexGene) (UU1 UU2 : preSieve G)

(uu : preSieveTransf UU1 UU2)

(VV1\_ : **forall** H : vertexGene, 'preSieve( H ~> **\_** | UU1 ) -> sieveFunctor H)

(VV2\_ : **forall** H : vertexGene, 'preSieve( H ~> **\_** | UU2 ) -> sieveFunctor H)

(vv\_ : **forall** (H: vertexGene) (u1: 'preSieve( H ~> **\_** | UU1 )),

sieveTransf (VV1\_ **\_** u1) (VV2\_ **\_** (uu **\_** u1))) :

sieveTransf (sumSieve VV1\_ ) (sumSieve VV2\_).

**Proof**. unshelve eexists. (\* \_transf\_sieveTransf \*) exact: sumSieve\_congrTransf.

(\* \_commute\_sieveTransf \*) abstract(intros K vu; simpl; rewrite -> \_commute\_sieveTransf; rewrite -> \_commute\_preSieveTransf; reflexivity).

**Defined**.

**Definition** typeOf\_basePreSieve (U : vertexGene) (UU : preSieve U) :=

**forall** (H : vertexGene) (u u' : 'preSieve( H ~> **\_** | UU )), u :>preSieve\_ == u' :>preSieve\_ -> u = u'.

**Parameter** basePreSieve : **forall** (U : vertexGene) (UU : preSieve U)

(UU\_base : typeOf\_basePreSieve UU) , **Type**.

**Inductive** isCover : **forall** (U : vertexGene) (UU\_pre : preSieve U) (UU : sieveFunctor U), sieveTransf UU (genSieve UU\_pre) -> **Type** :=

| BasePreSieve\_isCover : **forall** (U : vertexGene) (UU : preSieve U) (UU\_base : typeOf\_basePreSieve UU),

basePreSieve UU\_base -> @isCover **\_** UU (genSieve UU) (sieveTransf\_Ident **\_**)

(\*TODO | IdentSieve\_isCover : forall (G : vertexGene),

isCover (identSieve G) (identGene G ...) \*)

| InterSieve\_isCover : **forall** (G : vertexGene) (VV\_pre : preSieve G) (VV : sieveFunctor G) (VV\_transf : sieveTransf VV (genSieve VV\_pre))

(G' : vertexGene) (g : 'Gene( G' ~> G )) (UU\_pre : preSieve G') (UU : sieveFunctor G') (UU\_transf : sieveTransf UU (genSieve UU\_pre)),

@isCover **\_** VV\_pre VV VV\_transf -> @isCover **\_** UU\_pre UU UU\_transf -> @isCover **\_** UU\_pre (interSieve VV g UU) (sieveTransf\_Compos (interSieve\_projFactor **\_** **\_** **\_**) UU\_transf)

| SumSieve\_isCover : **forall** (G : vertexGene) (VV : preSieve G) (VV\_base : typeOf\_basePreSieve VV)

(VV\_base\_cover : basePreSieve VV\_base),

**forall** (famVertex\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

vertexGene)

(famPreSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

preSieve (famVertex\_ object outer))

(famSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

sieveFunctor (famVertex\_ object outer))

(famSieveTransf\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

sieveTransf (famSieve\_ object outer) (genSieve (famPreSieve\_ object outer)))

(famIsCover\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

@isCover **\_** (famPreSieve\_ object outer) (famSieve\_ object outer) (famSieveTransf\_ object outer))

(famPullArrow\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

'Gene( object ~> famVertex\_ object outer ))

(famPullPreSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

preSieve object)

(famPullPreSieveTransf\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> G | VV )),

sieveTransfArrow (genSieve (famPullPreSieve\_ object outer)) (genSieve (famPreSieve\_ object outer)) (famPullArrow\_ object outer)),

@isCover **\_** (sumPreSieve famPullPreSieve\_) (sumPullSieve famPullArrow\_ famSieve\_ famPullPreSieve\_)

(sumPullSieve\_projSumPreSieve famPullArrow\_ famSieve\_ famPullPreSieve\_).

**Section** nerveSieve.

**Variables** (topPreSieveVertexes: vertexGene) (topPreSieve: preSieve topPreSieveVertexes) (structCoSheaf: typeOf\_objects\_functor).

**Inductive** nerveSieve: **forall** (U : vertexGene) (UU\_pre : (preSieve U)) (UU : sieveFunctor U) (UU\_transf: sieveTransf UU (genSieve UU\_pre)) (UU\_isCover : isCover UU\_transf),

**forall** (u\_arrowTop : 'Gene( U ~> topPreSieveVertexes)) (UU\_transfTop : presieveTransfArrow UU\_pre (genSieve topPreSieve) u\_arrowTop),

**forall** (G : vertexGene) (g\_sense : 'Gene( G ~> U)),

**forall** (dim: nat) (diffPreSieveVertexes: 'I\_(dim) -> vertexGene )

(diffPreSieve: **forall** i : 'I\_(dim), 'preSieve( (diffPreSieveVertexes i) ~> **\_** | topPreSieve )), **Type** :=

| NerveSieve\_Diff (\* at cell dim +1 , at coeffiecients degree +1 \*) :

**forall** (U : vertexGene) (UU\_pre : (preSieve U)) (UU\_pre\_base : typeOf\_basePreSieve UU\_pre) (UU\_pre\_cover : basePreSieve UU\_pre\_base),

**forall** (u\_arrowTop : 'Gene( U ~> topPreSieveVertexes)) (UU\_transfTop : presieveTransfArrow UU\_pre (genSieve topPreSieve) u\_arrowTop),

**forall** (famVertex\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), vertexGene)

(famPreSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), preSieve (famVertex\_ object outer))

(famSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), sieveFunctor (famVertex\_ object outer))

(famSieveTransf\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

sieveTransf (famSieve\_ object outer) (genSieve (famPreSieve\_ object outer)))

(famIsCover\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

isCover (famSieveTransf\_ object outer))

(famTopArrow\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), 'Gene( (famVertex\_ object outer) ~> topPreSieveVertexes ) )

(famTransfTop\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), presieveTransfArrow (famPreSieve\_ object outer) (genSieve topPreSieve) (famTopArrow\_ object outer))

(famPullArrow\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), 'Gene( object ~> famVertex\_ object outer ))

(famPullPreSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

preSieve object)

(famPullPreSieveTransf\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

sieveTransfArrow (genSieve (famPullPreSieve\_ object outer)) (genSieve (famPreSieve\_ object outer)) (famPullArrow\_ object outer))

(famHeqArrow\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

(\* (outer :>sieve\_) o>functor\_[functor\_ViewOb \_] u\_arrowTop :=: \*) (UU\_transfTop **\_** outer) :>sieve\_

== (famPullArrow\_ object outer) o>functor\_[functor\_ViewOb **\_**] (famTopArrow\_ object outer) ),

**forall** (G : vertexGene) (g\_sense : 'Gene( G ~> U)),

**forall** (dim: nat) (object: 'I\_(S dim) -> vertexGene),

**forall** (outer: **forall** i : 'I\_(S dim), 'preSieve( object i ~> U | UU\_pre )),

**forall** (inner: **forall** i : 'I\_(S dim),

'Sieve( G ~> **\_** | interSieve (famSieve\_ (object i) (outer i)) (famPullArrow\_ (object i) (outer i))

(genSieve (famPullPreSieve\_ (object i) (outer i))) ) ),

**forall** (inner\_nerveSieve: **forall** i : 'I\_(S dim),

nerveSieve (famIsCover\_ (object i) (outer i))

(famTransfTop\_ (object i) (outer i))

(\* ((\_outer\_sumSieve (((inner i) :>transf\_ (interSieve\_projWhole \_ \_ \_)) :>transf\_ (famSieveTransf\_ (object i) (outer i)) )) :>preSieve\_ ) \*)

((\_outer\_sumSieve ( (famPullPreSieveTransf\_ (object i) (outer i)) **\_** ((inner i) :>transf\_ (interSieve\_projFactor **\_** **\_** **\_**)) )) :>preSieve\_ )

(**fun** j : 'I\_(dim) => \_outer\_sumSieve (UU\_transfTop **\_** (outer (lift i j))))),

**forall** (inner\_senseCompat : **forall** i : 'I\_(S dim), ((inner i) :>sieve\_) o>functor\_[functor\_ViewOb **\_**] ((outer i) :>preSieve\_) == g\_sense ),

**forall** (G\_weight : structCoSheaf G),

nerveSieve (SumSieve\_isCover UU\_pre\_cover famIsCover\_ famPullPreSieveTransf\_)

(preSieveTransf\_of\_sieveTransfArrow (sieveTransfArrow\_Compos (sieveTransfArrow\_of\_preSieveTransf (sumPreSieve\_projOuter famPullPreSieve\_))

(sieveTransfArrow\_of\_preSieveTransf UU\_transfTop))) g\_sense

(**fun** i : 'I\_(S dim) => \_outer\_sumSieve (UU\_transfTop **\_** (outer i)))

| NerveSieve\_Gluing (\* at same cell dim >= 0, at coefficients degree +1 \*) :

**forall** (U : vertexGene) (UU\_pre : (preSieve U)) (UU\_pre\_base : typeOf\_basePreSieve UU\_pre) (UU\_pre\_cover : basePreSieve UU\_pre\_base),

**forall** (u\_arrowTop : 'Gene( U ~> topPreSieveVertexes)) (UU\_transfTop : presieveTransfArrow UU\_pre (genSieve topPreSieve) u\_arrowTop),

**forall** (famVertex\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), vertexGene)

(famPreSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), preSieve (famVertex\_ object outer))

(famSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), sieveFunctor (famVertex\_ object outer))

(famSieveTransf\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

sieveTransf (famSieve\_ object outer) (genSieve (famPreSieve\_ object outer)))

(famIsCover\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

isCover (famSieveTransf\_ object outer))

(famTopArrow\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), 'Gene( (famVertex\_ object outer) ~> topPreSieveVertexes ) )

(famTransfTop\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), presieveTransfArrow (famPreSieve\_ object outer) (genSieve topPreSieve) (famTopArrow\_ object outer))

(famPullArrow\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )), 'Gene( object ~> famVertex\_ object outer ))

(famPullPreSieve\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

preSieve object)

(famPullPreSieveTransf\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

sieveTransfArrow (genSieve (famPullPreSieve\_ object outer)) (genSieve (famPreSieve\_ object outer)) (famPullArrow\_ object outer))

(famHeqArrow\_ : **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

(\* (outer :>sieve\_) o>functor\_[functor\_ViewOb \_] u\_arrowTop :=: \*) (UU\_transfTop **\_** outer) :>sieve\_

== (famPullArrow\_ object outer) o>functor\_[functor\_ViewOb **\_**] (famTopArrow\_ object outer) ),

**forall** (G : vertexGene) (g\_sense : 'Gene( G ~> U)),

**forall** (dim: nat) (diffPreSieveVertexes: 'I\_(dim) -> vertexGene )

(diffPreSieve: **forall** i : 'I\_(dim), topPreSieve (diffPreSieveVertexes i)),

**forall** (fam\_nerveSieve: **forall** (object: vertexGene) (outer: 'preSieve( object ~> U | UU\_pre )),

nerveSieve (famIsCover\_ object outer)

(famTransfTop\_ object outer)

(famPullArrow\_ object outer)

diffPreSieve),

**forall** (G\_weight : structCoSheaf G),

nerveSieve (SumSieve\_isCover UU\_pre\_cover famIsCover\_ famPullPreSieveTransf\_)

(preSieveTransf\_of\_sieveTransfArrow (sieveTransfArrow\_Compos (sieveTransfArrow\_of\_preSieveTransf (sumPreSieve\_projOuter famPullPreSieve\_))

(sieveTransfArrow\_of\_preSieveTransf UU\_transfTop))) g\_sense diffPreSieve

| NerveSieve\_Base (\* at cell dim = 0, at coeffiecients degree = 0 \*) :

**forall** (U : vertexGene) (UU\_pre : preSieve U) (UU\_pre\_base : typeOf\_basePreSieve UU\_pre) (UU\_pre\_isBase: basePreSieve UU\_pre\_base ),

**forall** (u\_arrowTop : 'Gene( U ~> topPreSieveVertexes)) (UU\_transfTop : presieveTransfArrow UU\_pre (genSieve topPreSieve) u\_arrowTop),

**forall** (G : vertexGene) (g\_sense : 'preSieve( G ~> **\_** | UU\_pre)),

nerveSieve (BasePreSieve\_isCover UU\_pre\_isBase) UU\_transfTop (g\_sense :>preSieve\_ )

(**fun** i : 'I\_0 => (ffun0 (card\_ord 0) : **forall** i : 'I\_(0), 'preSieve( ((ffun0 (card\_ord 0)) i) ~> **\_** | topPreSieve )) i).

**End** nerveSieve.

**End** COMOD.

(\*\* # #

#+END\_SRC

Voila.

# # \*\*)