title / ereview (Grammatical sheaf cohomology, its MODOS proof-assistant and WorkSchool 365 market for learning reviewers) title / ereview

short / ereview (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 implements the cut-elimination confluence of this inductive construction where the decreasing measure of families-gluing is the restricting covering: | Gluing : (forall (G : Site) (v : Site(G → V | in sieveV)), PreSheaves(Restrict F (sievesW_ v) → Sheafified E)) ⊢ PreSheaves(Restrict F (Sum sievesV_ over sieveU) → Sheafified E). And the separateness-property is expressed via the congruence-conversions clauses. Then the generalization to cohomology beyond 0th (sheaf) is that the grammatical sieves could be programmed such to inductively store the (possibly incompatible) data along with its gluing-differentials: Any list of (semantically-equal) arrows in the grammatical sieve now stores both data (on the singleton lists) and differentials (on the exhaustive ordered listings), and the (inductive) differentials of the outer-gluing of inner-gluings correctly-compute the differentials of the total/sum gluing because ∂∂ = 0... Moreover, the generating topological site has its own cut-elimination confluence of arrow-terms, each arrow-term is covered by its arrow-subterms, and the algebra-operation of composition ⟦f⟧*⟦B⟧*⟦g⟧ → ⟦f ∘_B g⟧ is indeed geometric, is some sheaf condition. Possible applications are the constructive connecting-snake lemma for additive sheaves, or the constructive dependent homotopy types or the constructive geometry of quantum fields in physics.

This research is the fusion of prompts from two expert mathematicians: Kosta Dosen and Pierre Cartier. But should this research be immediately-conclusive and peer-reviewed only by experts in some publishing-market susceptible under falsifications/intoxications? And what sense is peer review of already-computer-verified mathematics? WorkSchool 365 is Your Market for Learning Reviewers. WorkSchool 365 is your education marketplace where the prompting authors pay to get peer reviews of their documents from any learning reviewers who pass the test quiz inside the prompting document, with shareable transcripts receipts of the school work. WorkSchool 365 documents are Word templates with business-logic automation and playable Coq scripts. WorkSchool 365 is free open-source code Microsoft Teams app in the web browser with authentication via only no-password email. Enroll today! WorkSchool365.com) short / ereview

reviewers / ereview (.) reviewers / ereview

Learning Reviewers Quiz Q1. The MODOS end-goal is:
(A) proof-assistant for the computational logic of inductive-constructive-sheafification.
(B) formalization of the correctness of the book “Categories for the Working Mathematician”.
(C) writing pretty vertical formulas in latex.

Q1 ; 50 / quiz (Click or tap here to enter text.)

S0 / coq (`Check 37:nat. Goal 0=0. reflexivity. Qed.`) S0 / coq

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

Outline:

1. **WorkSchool 365 market for learning reviewers**
2. **What is the minimal example of sheaf cohomology? Grammatically**
3. **Interactive outline of the MODOS grammar**

1. WorkSchool 365 market for learning reviewers

What sense is peer review of already-computer-verified mathematics?

WorkSchool 365 is Your Market for Learned Reviews. Marketplace for peer reviews by paid learners qualified via quiz; open-source Word templates with business-logic automation.

"PROMPT: Something which inspires a response, especially a statement or series of questions designed to provoke creative or critical thought from a student."

WorkSchool 365 is your education marketplace where the prompting authors pay to get peer reviews of their documents from any ***learning reviewers who pass the quiz*** polls inside the prompting document, with ***shareable transcripts receipts*** of the school work.

In legalese, «peers» need not be experts but could be learners who are tested, and «publications» need not be conclusive but could be writing prompt documents, and the «market» need not be susceptible under ***falsifications/intoxications*** but could use some immune transcripts-receipts currency. Benefits for learning reviewers: get qualified and paid to share your view. Benefits for prompting authors: get valued by learning reviewers who pass your qualifying quiz.

WorkSchool 365 is free ***open-source code*** Microsoft Teams app (SharePoint with Power Automate) for unlimited users authenticated with only ***no-password*** email (via Microsoft Azure AD); with integration of many popular payments API for the marketplace currency (Stripe with Alipay, WeChat Pay, and PayPal) . The CRM and LMS features within WorkSchool 365 also imply: ⇒ Your MBA for the Classroom, ⇒ Your Form for Event Registrations. Ref: https://github.com/1337777/workschool365

WorkSchool 365 documents are Word templates which also integrate the open-source code ***Coq proof-assistant add-in for Word***, and come with sample content from the Gentle Introduction to the Art of Mathematics textbook (https://giam.southernct.edu). Ref: https://github.com/1337777/1337777.github.io

Enroll today! WorkSchool365.com

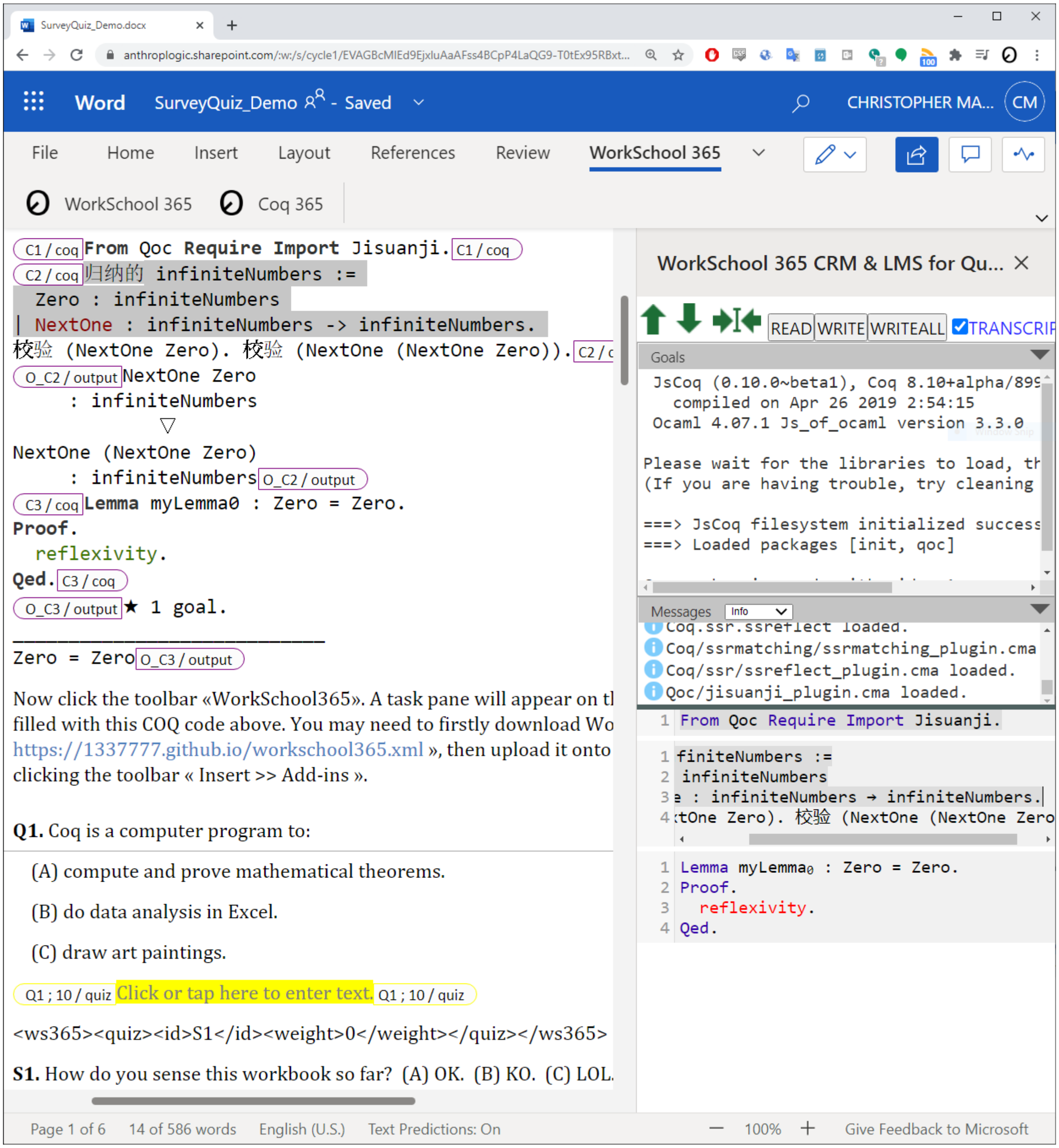


Figure: open-source Word templates
with business-logic automation and integration of the Coq proof-assistant

2. What is the minimal example of sheaf cohomology? Grammatically

2.1. Appendix

Lemma: (The inductively-constructed sheaves have the separateness-property by construction via the congruence-conversions clauses.) Hold the topology site containing one terminal object (the 3-points space) covered by two objects (open sets) U and V which have another intersection object distinct from the initial (empty) object. Then the (sheafified) natural transformation from the coproduct object U + V to the terminal object has surjective image-sheaf, but is not surjective section map at every object (sheaf cohomology). The lemma is that this description can be programmed purely grammatically, in some new categorial computational logic proof-assistant which has cut-elimination confluence (as for proof-theory or type theory), polymorph universal operations (adjunction counits but on generalized elements not only singletons), constructive sheafified dataobjects (generated free term-algebras but via geometry), and fibred objects (dependent types but with no-variables logical-quantifiers).

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!

Indeed, here is some analogy. What is more primitive than appending (flattening) two (a sequence of) lists? Answer: the operation that cons the head with the tail.

And the MODOS proof-assistant implements the cut-elimination confluence of this inductive construction where the decreasing measure of families-gluing is the restricting covering sieves instead of the natural numbers:

```
| Constructing : (G : Site); (u : Site( G ~> U | in sieveU ));
                    (f : F G); (_ : isGene f)
-------------------------------------------------------------------
⊢  Element( G ~> Restrict F sieveU )

| UnitSheafified : (G : Site); (u : Site( G ~> U | in sieveU ));
                    (e : Element( G ~> E )); (ut : Site( U ~> T | in sieveT ))
-------------------------------------------------------------------
⊢  Element( G ~> Sheafified (Restrict E sieveT) )

| RestrictCast :
      (ut : Site( U ~> T | in sieveT ))
-------------------------------------------------------------------
⊢  PreSheaves( Restrict E sieveU ~> Restrict E sieveT )

| SheafifiedMor :
      PreSheaves( F ~> E )
-------------------------------------------------------------------
⊢  PreSheaves( Sheafified F ~> Sheafified E )

| Destructing : (forall (G : Site) (u : Site( G ~> U | in sieveU ))
    (f : F G) (_ : isGene f), Element( G ~> E )); (ut : Site( U ~> T | in sieveT ))
-------------------------------------------------------------------
⊢  PreSheaves( Restrict F sieveU ~> Sheafified (Restrict E sieveT) )

| Gluing : (forall (G : Site) (u : Site( G ~> U | in sieveU )),
                    PreSheaves( Restrict F (sievesV_ u) ~> Sheafified E ))
-------------------------------------------------------------------
⊢  PreSheaves( Restrict F (sum sievesV_ over sieveU) ~> Sheafified E )
```

Lemma: cut-elimination holds. Corollary: grammatical sheaf cohomology exists.

And the separateness-property of the inductively-constructed sheaf is expressed by construction via the congruence-conversions clauses.

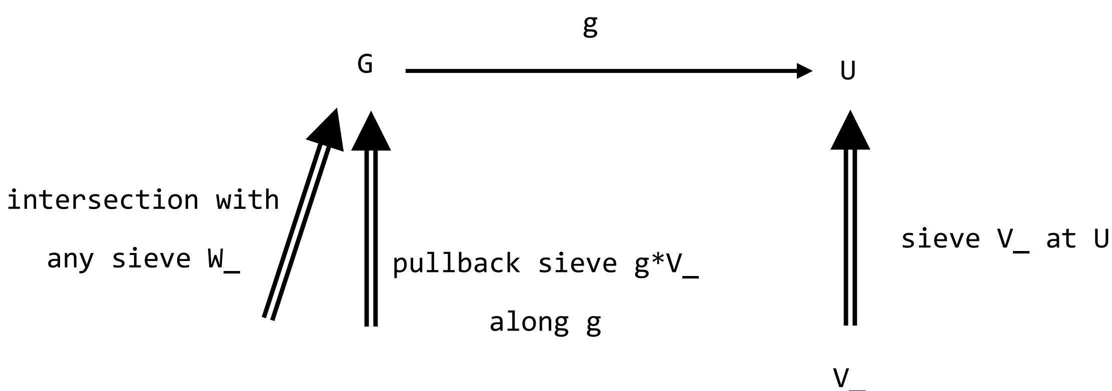
Then the generalization to cohomology beyond 0th (sheaf) is that the grammatical sieves could be programmed such to inductively store the (possibly incompatible) data along with its gluing-differentials: Any list of (semantically-equal) arrows in the grammatical sieve now stores both data (on the singleton lists) and differentials (on the exhaustive ordered listings), and the (inductive) differentials of the outer-gluing of inner-gluings correctly-compute the differentials of the total/sum gluing because $\partial\partial = 0$…

And to express fibred morphisms, the shape of the point is now any “A” instead of the singleton, and the context-extension is polymorph…

for (B over Delta) and for variable (Theta), then

```
Span(Theta ~> (Delta;B))  :<=>  Hom( (x: Gamma; a: A( h(x) )) ~> B( f(x) ) )
with some (f: Gamma -> Delta) and (h: Gamma -> Theta) and (A over Theta)
```

And the definition of the restriction object uses this new style of “***intersection pullback***”:



data f : F(W_ ∩ g*V_) over

only local-pieces of G

```
( Restrict F V_ ) G := Sum (W_ : sieve at G) × (g : G → U) × F(W_ ∩ g*V_)
```

And the definition of the sheafification object is:

$$
\begin{tikzcd}
G \\
A\_ \arrow[u, Rightarrow, "\text{any sieve } A\_ \text{ at } G"']
\end{tikzcd}
$$

elements e : (E A_) of E

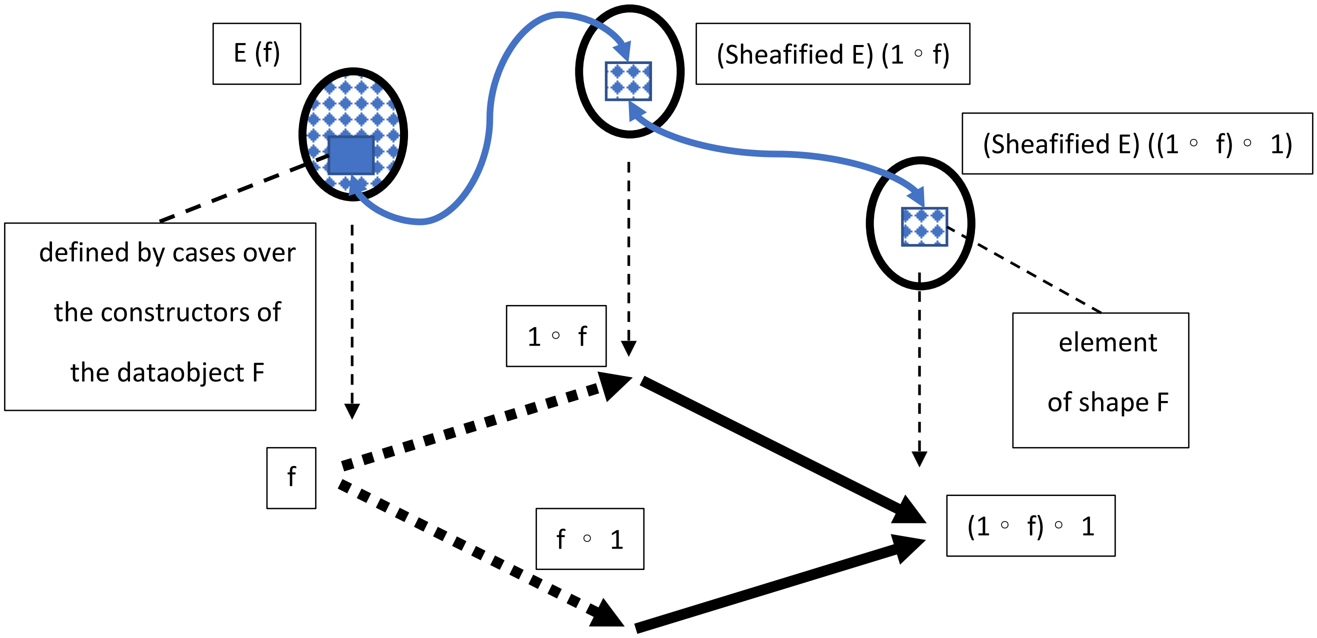
over local-pieces A_ of G

(Sheafified E) G := Sum (A_ : sieve at G) × (E A_)

Contrast the foregoing description with these two well-studied topics: the categorial semantics of type theory syntax and the functorial-semantics of universal algebra syntax. For example: free algebras of some endofunctor which implement datatypes are iteratively constructed as colimits, which themselves are recursively constructed from coequalizers and coproducts; and multi-sorted structures such as any graph with one sort-of-edges and one sort-of-nodes are the covariant sketch models of some coherent theory. Instead, ***Kosta Dosen*** says that categories itself is already some computational logic syntax which has cut-elimination confluence of arrow-terms (in the signature for some adjunction, or comonad, or pairing-product, or 2-category, or proof-net star-autonomous category...). The only difficulty was to discover that the universal arrow (counit) of some adjunction should instead be formulated as some polymorph operation (`f “∘counit” : Left Right P → Q` for any arrow `f : P → Q`). Moreover, remember that the signature for any internal category has one sort-of-objects and one sort-of-arrows, and for any enriched category has many sorts-of-arrows at any source-target objects. Instead, now any arrow-term (such as the product-pairing `<f,g>`) is one sort in the signature which will denote the set of occurrences of this arrow-term in the concrete model (any category with arrow-operations for products). Define any model (in Set) to be some grammatical sheaf (hence globular copresheaf) of (span of) sets over this site. The usual algebra-operations are now constructed via the geometry of coverings (each term is covered by its subterms, indeed the composition $[\![f]\!]*[\![B]\!]*[\![g]\!] \to [\![f \circ_B g]\!]$ is geometric, is some sheaf condition), and the algebra-equations can now be oriented (directed) and are satisfied via the geometry of coverings (each redex term is covered by its contractum). The free algebra datatype construction is now via the geometry of associated-sheafification: starting with some generative presheaf data, then sheafification-restricted-below-any-sieve of this presheaf can be inductively constructed by refinements of the sieves; not merely computationally but with the logical constructing-destructing-refining clauses. Finally, to describe fibred objects with logical-quantifies, it may be assumed some generating cocontinuous adjunction of sites which generates some geometric morphism of topos; for example, any category model is fibred over its pre-order category.

Then what is the categorial semantics of this categorial syntax? The sense mimicks the usual Kripke-Joyal sense, as explicit definitions. The generic model contravariantly sends any object G to the covariant diagram of sets represented by the sheafified G over only the finitely-presentable (data) sheaf-models: G

↦ `Hom(sheafified(Hom(-, G)), fpModelsSet(_))` … and further could be sliced over any (outer/fixed) dataobject.



Proof: Hold the generating topology site containing one terminal object T covered by two objects U (via arrow `ut`) and V (via arrow `vt`) which have another intersection object X (via arrow `xu` to U and arrow `xv` to V, such that `xu◦>ut = xv◦>vt = xt`). Then the sheafified copairing natural transformation `Sheaf[ut|vt]: Sheaf(U+V) → Sheaf(T)` is some epimorphism, but the post-composition (section) map (`_ ◦> Sheaf[ut|vt]`) is not surjective as there is no (global section) morphism in T → `Sheaf(U+V)` which is mapped to the unit. Now the proposition that `Sheaf[ut|vt]` is epic, really is some ("summarized") tautological rephrasing of the congruence-conversion clauses for the constructors of sheaves.

Imprecisely, the goal is to show these two propositions:

```
for any presheaf functor F and cut-free natural transformations
ff1, ff2: PreSheaves(Sheaf(Restrict T sieve{ut,vt}) → Sheaf(F)),
if ((Sheaf[(Constructing ut)|(Constructing vt)]) ◦> ff1) ~
((Sheaf[(Constructing ut)|(Constructing vt)]) ◦> ff2),
then ff1 ~ ff2.
```

```
for any presheaf functor F and elements
ff1, ff2: Element(T → Sheaf(F)),
if ([ut|vt] ◦> ff1) ~ ([ut|vt] ◦> ff2),
then ff1 ~ ff2.
```

The proof is by unfolding/externalizing the sum copairing, then the congruence-conversion clauses for the constructors with sheaf codomain are indeed separateness-properties of the sheafification. Proved.

Note that by induction, oneself proves that every grammatically-constructed presheaf object is separated if it is assumed that any covering sieve is jointly-epic in the site (or that the generating presheaf dataobjects are separated).

Lemma: tentatively, the connecting snake morphism would be programmed constructively (because the equality relation on any constructed sieve was designed to be grammatical/structural, not merely semantical), for the long exact sequence of sheaf cohomology `0 → 0 → 0 → ℤ → ℤ → 0 → 0 → 0` from this short exact sequence (`c` is the coproduct U+V).

$$
\begin{tikzcd}
0 \arrow[r] & 0 \arrow[r] \arrow[d] & uc\mathbb{Z}\oplus vc\mathbb{Z} \arrow[r, "1\oplus 1"] \arrow[d, "1\oplus -1"'] & ut\mathbb{Z}\oplus vt\mathbb{Z} \arrow[r] \arrow[d, "\_-\_"] & 0 \\
0 \arrow[r] & (xu\circ{>}uc\ -\ xv\circ{>}vc)\mathbb{Z} \arrow[r, "{(1,-1)}"] & (xu\circ{>}uc)\mathbb{Z}\oplus(xv\circ{>}vc)\mathbb{Z} \arrow[r, "\_+\_"] & xt\mathbb{Z} \arrow[r] & 0
\end{tikzcd}
$$

Summary: Hold any Dosen-style ***cut-elimination confluence of arrow-terms*** (for some comonad, or pairing-product, or 2-category, or proof-net star-autonomous category,...), and form the (petit) grammatical-globular site (double category) whose objects are the arrow-terms and where any (necessarily finite) covering family of morphisms is either any reduction-conversion linkage or all the (immediate proper, including unit-arrows in cuts) subterms of some redex arrow-term. Define any model (in Set) to be some grammatical sheaf (hence globular copresheaf) of (span of) sets over this site, where each covering family become limit cone (constructively, using compatible families). Now starting with some generative presheaf data, then sheafification-restricted-below-any-sieve of this presheaf can be inductively constructed by refinements of the sieves. Moreover, it may be assumed some generating ***cocontinuous adjunction of sites***; the result is some dependent-constructive-computational-logic of geometric dataobjects (including homotopy-types): ***MODOS***. Now ***globular homology*** of any copresheaf computes the composable occurrences of arrow-terms (cycles from 0 to 1). Also ***grammatical cohomology*** of the sheafification (graded by the nerve of its sieve argument) computes the global solutions of occurrences of all arrow-terms in the model which satisfy the confluence of reductions in the site. Contrast to the covariant sketch models of some coherent theory; but now any globular-covariant (contravariant finite-limit sketch) concrete model is some category with operations on arrows. The sense mimicks the usual Kripke-Joyal sense, as explicit definitions. The ***generic model*** contravariantly sends any object G to the covariant diagram of sets represented by the sheafified G over only the finitely-presentable (data) sheaf-models: `G ↦ Hom(sheafified(Hom(-, G)), fpModelsSet(_))` … and further could be sliced over any (outer/fixed) dataobject.

2.2. Context

(1.) What problem is to be solved? Attempt to formulate some homotopical computational logic for ***geometric dataobjects***, which is some common generalization of the constructive-inductive datatypes in logic and the sheaves in geometry. Also during this process, emphasize the communication-format in which this library of new-mathematics is multi-authored, published and reviewed inside structured-documents which integrate this same computational-logic proof-assistant.

(2.) ***OCAML/COQ*** computer is for reading and writing mathematical computations and proofs. Any collection of elements ("datatype") may be presented constructively and inductively, and thereafter any function ("program") may be defined on such datatype by case-analysis on the constructors and by recursion on this function itself. Links: http://coq.inria.fr

Moreover, the COQ computer extends mere computations (contrasted to OCAML) by allowing any datatype to be parameterized by elements from another datatype, therefore such parameterized datatypes become logical propositions and the programs defined thereon become proofs.

(3.) The computational logic foundation of OCAML/COQ is "type theory", where there is no real grammatical distinction between elements and types as grammatical terms, and moreover only "singleton" terms can be touched/probed. Also, the usual constructive-inductive datatypes of "type theory" generalize the natural-numbers induction to allow structural constructors of the datatype to form expression-trees, but fails to articulate all the possible geometries in the new datatypes.

Type theory was OK for computer-science applications, but is not OK for mathematics (categorial-algebra). A corollary is that (differential cohesive linear) "homotopy type theory" inherits the same flaws. For instance, the algebraic geometry of affine schemes say that "points" (prime ideals) are more than mere singletons: they are morphisms of irreducible closed subschemes into the base scheme.

It is now learned that it was not necessary to retro-grade categorial-algebra into type theory ("categorical-logic" in the sense of Joachim Lambek); but there is instead some alternative reformulation of categorial-algebra as a cut-elimination computational-logic itself (in the sense of ***Kosta Dosen*** and ***Pierre Cartier***), where the generalized elements (arrows) remain internalized/accumulated ("point-as-morphism" / polymorphism) into grammatical-constructors and not become variables/terms as in the usual topos internal-language... Links: http://www.mi.sanu.ac.rs/~kosta ; http://www.ihes.fr/~cartier

(4.) ***GAP/SINGULAR*** computer is for computing in permutation groups and polynomial rings, whenever computational generators are possible, such as for the orbit-stabilizer algorithm ("Schreier generators") or for the multiple-variables multiple-divisors division algorithm ("Euclid/Gauss/Groebner basis"). Links: https://www.gap-system.org

In contrast to GAP/SINGULAR which does the inner computational-algebra corresponding to the affine-projective aspects of geometry, the MODOS aims at the outer logical/categorial-algebra corresponding to the parameterized-schematic aspects of geometry; this contrast is similar as the OCAML-COQ contrast. In short: MODOS does the computational-logic of the coherent sheaf modules over some base scheme; dually the relative support/spectrum of such sheaf modules/algebras are schemes parameterized over this base scheme (alternatively, the slice topos over this sheaf is étale over the base topos). Links: https://stacks.math.columbia.edu/tag/01LQ

(5.) MODOS proof-assistant has solved the critical techniques behind those questions, even if the production-grade engineering is still lacking. Some programming techniques ("cut-elimination", "confluence", "dependent-typed functional programming"...) from computer-science (electrical circuits) generalize to the alternative reformulation of categorial-algebra as a cut-elimination computational-logic ***("adjunctions", "comonads", "products", "enriched categories", "internal categories", "2-categories", "fibred category with local internal products", "associativity coherence", "semi-associativity coherence", "star-autonomous category coherence"***,...). Links: https://github.com/1337777/cartier ; https://github.com/1337777/dosen

(6.) The MODOS is the computational logic for ***geometric dataobjects***, which is some common generalization of the constructive-inductive datatypes in logic and the sheaves in geometry. The MODOS may be the solution to program such questions of the form: how to do the ***geometric parsing*** of some pattern (domain) to enumerate its morphisms/occurrences within/against some language/sheaf geometric dataobject (codomain). The computational logic of those morphisms/occurrences have algebraic operations (such as addition, linear action), and also have geometric operations (such as restriction, gluing). ***At the core, the MODOS has some constructive inductive/refined formulation of the sheafification-operation-restricted by any convering sieve whose refinements are the measure for the induction***.

2.3. Possible applications to geometric algorithmics and quantum-fields physics

(1.) What problem is to be solved? In algorithmics, the usual constructive-inductive datatypes generalize the natural-numbers induction to allow structural constructors of the datatype to form expression-trees, but fails to articulate all the possible geometries in the new datatypes. In physics, Quantum Fields is an attempt to upgrade the mathematics of the 19th century's Maxwell equations of electromagnetism, in particular to clarify the duality between matter particles and light waves. However, those differential geometry methods (even post-Sardanashvily) are still “equational algebra” (from Newton x(t), to Lagrange q(t), to Schrodinger phi(t), up to Feynman psi(x,t)) and fail to upgrade the computational-logic.

(2.) The geometry content of the quantum fields in physics is often in the form of the differential-geometry variational-calculus to find the optimal action defined on the jet-bundles of the field-configurations. This is often formulated in differential, algebraic and even (differential cohesive linear) “homotopy type theory”, of fibered manifolds with equivariance under natural (gauge) symmetries. However, the interdependence between the geometry and the dynamics/momentum data/tensor is still lacking some computational-logic (constructive, mutually-inductive) formulation. Links: https://ncatlab.org/nlab/show/jet+bundle ; https://ncatlab.org/nlab/show/geometry+of+physics

(3.) The computational content of quantum mechanics is often formulated in the substructural-proof technique of dagger compact monoidal categories (linear logic of duality); this computational content should be reformulated ***using the grammatical/syntactical cut-elimination of star-autonomous categories, instead of using the proof-net/string-diagrams graphical normal forms***. Moreover this computational-logic should be ***upgraded to (the sheaves of quantum-states modules over) the jet-bundles of the field-configurations, parameterized over some spacetime manifold***. Now the computational content of the quantum-field is often in the form of the statistics of the correlation at different points of some field-configuration and the statistics of the partition function expressed in the field-configurations modes. A corollary: the point in spacetime is indeed not “singleton” (not even some “string” ...); the field configurations are statistical/thermal/quantum and “uncertain” (the derivative/commutator of some observable along another observable is not zero).

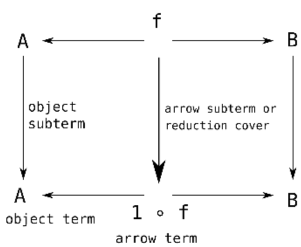
(4.) The MODOS is the homotopical computational logic for ***geometric dataobjects and parsing***, which is some generalization of the constructive-inductive datatypes in logic and the sheaves in geometry.

2.4. The generating site of arrow-terms with confluence

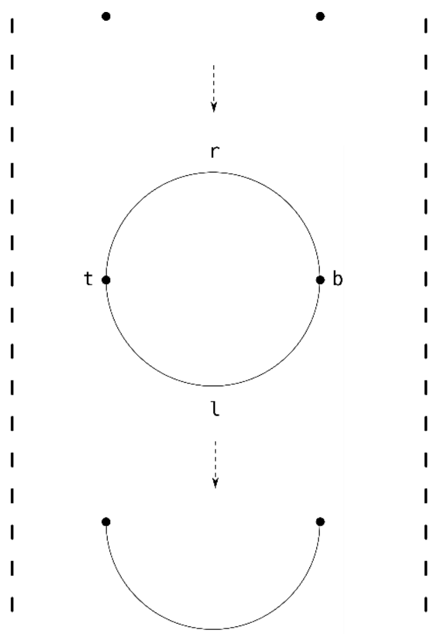
The topos of sheaves is presentable by generators from some site, freely-completed with pullback/substitution distributing over coequalizers-of-kernel-relations and unions-of-subobjects; in contrast to internal methods via Lawvere(-Tierney) geometric modalities. The site is both grammatical/inner (object is syntactic term) and globular/outer (object is span with dimension grading). For example the union of two free-monoid-on-one-generator (as one-object categories) requires sheafification (adding all compositions/cuts across) to become the free-monoid-on-two-generators

Moreover, it may be assumed some generating ***cocontinuous adjunction of sites*** (fibre of any covering sieve is covering), which is some instance of morphism of sites generating some geometric morphism of toposes. Examples of this assumption are: ***the étale map from the circle to the projective space***; or ***the fields-configurations jet-bundle over some spacetime manifold***. In short: ***the site may be parameterized below or relativized above.*** Applications: with proof-net star-autonomous categories, get some constructive alternative to Urs Schreiber's geometry of quantum-fields physics.

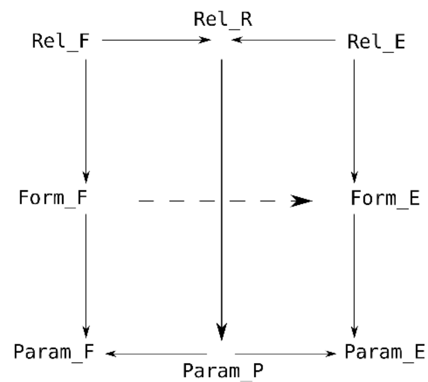
Additive sheaf cohomology over any site may also be formulated in this computational-logic. In short: ***MODOS interfaces the COQ categorial logic of sheaves down to the GAP/SINGULAR algebra of modules***.



grammatical globular site (double category)



cellular site, parameterized, relativized



generated topos of sheaves over the site, parameterized, relativized

Finiteness of the site may be assumed, such as for the site of open subsets of some finite space or finitely generated space or finitely-compact generated space. The “points” of such finite space should be thought of as ordered-by-inclusion “cell faces” (irreducible closed subsets) of another non-finite space. For example, the finite space corresponding to the circle is the “pseudocircle”, whose underlying set has 4 elements { l , r , t , b } (the left arc, right arc, top vertex and bottom vertex of the circle), and whose collection of open subsets is { { l , r , t , b } , { l , r , t } , { l , r , b } , { l , r } , { l } , { r } , { } }.

3. Interactive outline of the MODOS grammar

3.1. What is the end goal?

The end goal is not to verify that the sense is correct; of course, everything here makes sense. The end goal is whether it is possible to formulate some constructive computational logic grammatically. Therefore, this text shall be read first without attention to the sense, then read twice to imagine *some* sense. Ref: https://github.com/1337777/cartier

3.2. Outline:

In this Word document (search the file “WorkSchool365.docx”), click “Insert; Add-ins; WorkSchool 365 Coq” to load and ***play this script interactively***.

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

S1 / coq

```
From Coq Require Import RelationClasses Setoid SetoidClass
     Classes.Morphisms_Prop RelationPairs CRelationClasses CMorphisms.
From mathcomp Require Import ssreflect ssrfun ssrbool eqtype ssrnat fintype.
From Coq Require Lia.

Set Implicit Arguments. Unset Strict Implicit. Unset Printing Implicit
Defensive.
Set Primitive Projections. Set Universe Polymorphism.

Module SHEAF.

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 : Type.
```

```
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.

Global Ltac cbn_ := cbn -[equiv _type_relType _rel_relType _equiv_relType
_objects_functor _arrows_functor functor_ViewOb
                             _arrows_transf transf_ViewObMor
_functor_sieveFunctor _transf_sieveFunctor].
Global Ltac cbn_equiv := unfold _rel_relType, equiv; cbn -[ _arrows_functor
functor_ViewOb
                             _arrows_transf transf_ViewObMor
_functor_sieveFunctor _transf_sieveFunctor].
Global Ltac cbn_view := cbn -[equiv _type_relType _rel_relType _equiv_relType
_objects_functor  _arrows_functor
                             _arrows_transf  _functor_sieveFunctor
_transf_sieveFunctor].
Global Ltac cbn_functor := cbn -[equiv _type_relType _rel_relType
_equiv_relType functor_ViewOb
                               _arrows_transf transf_ViewObMor
_functor_sieveFunctor _transf_sieveFunctor].
Global Ltac cbn_transf := cbn -[equiv _type_relType _rel_relType
_equiv_relType  _arrows_functor functor_ViewOb
                               transf_ViewObMor _functor_sieveFunctor
_transf_sieveFunctor].
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 _transf_sieveFunctor] 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 _transf_sieveFunctor] 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
_transf_sieveFunctor] 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 _transf_sieveFunctor] 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
_transf_sieveFunctor] in H.
Tactic Notation "cbn_sieve" "in" hyp_list(H) := cbn -[equiv _type_relType
_rel_relType _equiv_relType   functor_ViewOb
                                  transf_ViewObMor ] in H.

Definition compatEquiv {U : vertexGene} {UU : sieveFunctor U}  {G} :
crelation ('Sieve( G ~> _ | UU ))
:= fun u u' : 'Sieve( G ~> _ | UU ) => u :>sieve_ == u' :>sieve_ .
Arguments compatEquiv /.

Definition compatEquiv_Equivalence (U : vertexGene) (UU : sieveFunctor U)  G
: Equivalence (@compatEquiv U UU G).
unshelve eexists.
-  abstract(move; intros; move; reflexivity).
-  abstract(move; intros; move; intros; symmetry; assumption).
- abstract(move; intros; move; intros; etransitivity; eassumption).
Qed.

Definition compatRelType (U : vertexGene) (UU : sieveFunctor U)  (G :
vertexGene) : relType.
exists ('Sieve( G ~> _ | UU )) compatEquiv.
exact: compatEquiv_Equivalence.
Defined.

Instance compatEquiv_subrelation (U : vertexGene) (UU : sieveFunctor U)  (G :
vertexGene) :
subrelation (@equiv _ _ (@_equiv_relType _)) (@compatEquiv U UU G).
move. intros u1 u2 Heq. cbn_. rewrite -> Heq. reflexivity.
Qed.

Notation "u ==s v" := (@equiv (* (@_type_relType (compatRelType _ _)) *) _
      (@_rel_relType (compatRelType _ _)) (@_equiv_relType (compatRelType _
_)) u v)
```

```
(at level 70, no associativity) : type_scope.

Definition typeOf_baseSieve (U : vertexGene) (UU : sieveFunctor U) :=
  forall (H : vertexGene) (u u' : 'Sieve( H ~> _ | UU )), u ==s u' -> u ==
u'.

Parameter baseSieve : forall (U : vertexGene) (UU : sieveFunctor U)
  (UU_base : typeOf_baseSieve UU), Type.

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) :>transf_ (_transf_sieveFunctor V2) ==  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.

Section Section2.
Variables (G : vertexGene) (VV : sieveFunctor G)
  (G' : vertexGene) (g : 'Gene( G' ~> G ))
  (UU : sieveFunctor G')
  (G'' : vertexGene) (g' : 'Gene( G'' ~> G' ))(WW : sieveFunctor G'').

Definition interSieve_compos : transf (interSieve VV (g'
o>functor_[functor_ViewOb G] g)
(interSieve UU g' WW) ) (interSieve VV g UU).
Proof. unshelve eexists. intros H. unshelve eexists.
- (* -> *) intros v; unshelve eexists.
    exact: ((_whole_interSieve (_factor_interSieve v)) ).
    exact: (_whole_interSieve v) .
    abstract(do 2 rewrite -> _wholeProp_interSieve;
    rewrite -> _functorialCompos_functor'; simpl; reflexivity).
- (* Proper *) abstract(move; move => f1 f2 Heq;
split; autounfold; simpl; [rewrite -> (whole_interSieve_Proper
(factor_interSieve_Proper Heq)); reflexivity
| rewrite -> (whole_interSieve_Proper Heq); reflexivity]).
- (* typeOf_natural_transf *) abstract (intros H H' h f; autounfold; split;
simpl; reflexivity).
Defined.

Definition pullSieve_compos : transf (pullSieve VV (g'
o>functor_[functor_ViewOb G] g)) (pullSieve VV g).
Proof. unshelve eexists. intros H. unshelve eexists.
- (* -> *) intros v; unshelve eexists.
    exact: ((_factor_interSieve v) o>functor_[functor_ViewOb G'] g').
    exact: (_whole_interSieve v) .
    abstract(rewrite -> _wholeProp_interSieve; rewrite ->
_functorialCompos_functor'; simpl; reflexivity).
- (* Proper *) abstract(move; move => f1 f2 Heq;
split; autounfold; simpl; [rewrite ->  (factor_interSieve_Proper Heq);
reflexivity
| rewrite -> (whole_interSieve_Proper Heq); reflexivity]).
- (* typeOf_natural_transf *) intros H H' h f; autounfold; split; cbn_sieve;
cbn_functor;
[ rewrite -> _functorialCompos_functor'; reflexivity
| reflexivity ].
Defined.
End Section2.

Lemma interSieve_congr G (VV1 VV2 : sieveFunctor G)  (vv: sieveTransf VV1
VV2)
G' (g1 g2 : 'Gene(G' ~> G)) (genEquiv: g1 == g2)
  (UU1 UU2 : sieveFunctor G') (uu: sieveTransf UU1 UU2):
```

```
  sieveTransf (interSieve VV1 g1 UU1) (interSieve VV2 g2 UU2).
Proof. unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
  (* _factor_interSieve *) exact: ((_factor_interSieve v) :>transf_ uu).
  (* _whole_interSieve *) exact: ((_whole_interSieve v) :>transf_ vv).
  (* _wholeProp_interSieve *) abstract(simpl; rewrite  ->
_commute_sieveTransf ,
  _commute_sieveTransf , _wholeProp_interSieve , genEquiv; reflexivity).
  (* _congr_relTransf  *) abstract(move; intros ? ? Heq; split; autounfold;
simpl;
  [ rewrite -> (factor_interSieve_Proper Heq); reflexivity
  | rewrite -> (whole_interSieve_Proper Heq); reflexivity]).
- (*  _natural_transf  *) abstract(intros H' H h v; split; simpl;
  rewrite -> _natural_transf; reflexivity).
- (* _commute_sieveTransf *) abstract(intros H v; simpl; rewrite ->
_commute_sieveTransf; reflexivity).
Defined.

Definition pullSieve_congr G (VV1 VV2 : sieveFunctor G)  (vv: sieveTransf VV1
VV2)
G' (g1 g2 : 'Gene(G' ~> G)) (genEquiv: g1 == g2):
  sieveTransf (pullSieve VV1 g1) (pullSieve VV2 g2)
  := @interSieve_congr G VV1 VV2 vv G' g1 g2 genEquiv _ _ (sieveTransf_Ident
_).

Lemma pullSieve_pullSieve G (VV : sieveFunctor G) G' (g : 'Gene(G' ~> G)) G''
(g' : 'Gene(G'' ~> G')):
sieveTransf (pullSieve (pullSieve VV g) g') (pullSieve VV (g'
o>functor_[functor_ViewOb _] g)).
Proof. unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
  (* _factor_interSieve *) exact (_factor_interSieve v).
  (* _whole_interSieve *) exact: ((_whole_interSieve (_whole_interSieve v))).
  (* _wholeProp_interSieve *)  abstract(rewrite -> _wholeProp_interSieve;
   rewrite -> _functorialCompos_functor';
   setoid_rewrite <- _wholeProp_interSieve at 2; simpl; reflexivity).
  (* _congr_relTransf  *) abstract(move; intros ? ? Heq; split; autounfold;
cbn -[_rel_relType];
   [ rewrite -> (factor_interSieve_Proper Heq); reflexivity
   | rewrite -> (whole_interSieve_Proper (whole_interSieve_Proper Heq));
reflexivity]) .
- (*  _natural_transf  *) abstract(move; split; simpl; reflexivity).
- (* _commute_sieveTransf *) abstract(move; reflexivity).
Defined.

Lemma pullSieve_pullSieve_rev G (VV : sieveFunctor G) G' (g : 'Gene(G' ~> G))
G'' (g' : 'Gene(G'' ~> G')): sieveTransf (pullSieve VV (g'
o>functor_[functor_ViewOb _] g)) (pullSieve (pullSieve VV g) g') .
```

```
Proof. unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
  (* _factor_interSieve *) exact (_factor_interSieve v).
  (* _whole_interSieve *) { unshelve eexists.
        (* _factor_interSieve *) exact (_factor_interSieve v
o>functor_[functor_ViewOb _] g').
        (* _whole_interSieve *) exact: ( _whole_interSieve v).
        (* _wholeProp_interSieve *)  abstract(rewrite ->
_wholeProp_interSieve;
        rewrite -> _functorialCompos_functor'; reflexivity). }
  (* _wholeProp_interSieve *)  abstract(reflexivity).
  (* _congr_relTransf  *) abstract  (move; intros v1 v2; case; autounfold;
cbn_;
  move => Heq_factor Heq_whole; split; autounfold; cbn -[_rel_relType];
  [rewrite -> Heq_factor; reflexivity | ]; split; autounfold; cbn -
[_rel_relType];
  [rewrite -> Heq_factor; reflexivity | rewrite -> Heq_whole; reflexivity ]).
- (*  _natural_transf  *) abstract (move; split; cbn_sieve;
[reflexivity | split; cbn_sieve;
[ rewrite -> _functorialCompos_functor'; reflexivity | reflexivity ]]).
- (* _commute_sieveTransf *) abstract(move; reflexivity).
Defined.

Lemma pullSieve_ident G (VV : sieveFunctor G) : sieveTransf (pullSieve VV
identGene) VV.
Proof. unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. exact: (_whole_interSieve v).
  (* _congr_relTransf  *) abstract (move; move => x y Heq;
   rewrite -> (whole_interSieve_Proper Heq); reflexivity).
- (* _natural_transf *) abstract(move; intros; simpl; reflexivity).
- (* _commute_sieveTransf *) abstract(move; intros; simpl; rewrite ->
_wholeProp_interSieve; simpl;
(* FUNCTOR/TRANSF PROBLEM *) apply: identGene_composGene).
Defined.

Lemma pullSieve_ident_rev G (VV : sieveFunctor G) : sieveTransf VV (pullSieve
VV identGene).
Proof. unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
        exact (v :>sieve_). exact v.
        abstract (cbn_sieve; symmetry; apply: identGene_composGene).
  (* _congr_relTransf  *) abstract(move; move => x y Heq; cbn_transf; split;
cbn_transf; rewrite -> Heq; reflexivity).
- (* _natural_transf *) abstract(move; intros; cbn_sieve; split; cbn_sieve;
            last reflexivity; rewrite -> _natural_transf; reflexivity).
- (* _commute_sieveTransf *) abstract(move; intros; cbn_sieve; reflexivity).
Defined.
```

```
End interSieve.

Existing Instance whole_interSieve_Proper.
Existing Instance factor_interSieve_Proper.

Lemma interSieve_composeOuter G (VV : sieveFunctor G)
G' (g : 'Gene(G' ~> G)) (UU : sieveFunctor G')
 G'' (g' : 'Gene(G'' ~> G')) G''' (g'' : 'Gene(G''' ~> G''))    :
  transf (interSieve    (pullSieve VV (g' o>gene g))  g'' (pullSieve UU (g''
o>gene g')))
   (interSieve VV g UU).
Proof.  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
  (* _factor_interSieve *) exact: ((v :>transf_ (interSieve_projFactor _ _
_))
      :>transf_ (pullSieve_projWhole _ _ ) ).
  (* _whole_interSieve *) exact: ((v :>transf_ (interSieve_projWhole _ _ _))
      :>transf_ (pullSieve_projWhole _ _ ) ).
  (* _wholeProp_interSieve *) abstract (cbn_transf; do 2 rewrite ->
_wholeProp_interSieve;
  rewrite -> (_wholeProp_interSieve v); tac_unsimpl;
   do 3 rewrite <- _functorialCompos_functor';
  reflexivity).
  (* _congr_relTransf  *) abstract (move; intros ? ? Heq; split; cbn_transf;
  rewrite -> Heq; reflexivity).
- (*  _natural_transf  *) abstract (intros H' H h v; split; cbn_sieve;
reflexivity).
Defined.

Lemma interSieve_composeOuter_ident G (VV : sieveFunctor G)
G' (g : 'Gene(G' ~> G)) (UU : sieveFunctor G')
 G'' (g' : 'Gene(G'' ~> G'))  :
  transf (interSieve    (pullSieve VV (g' o>gene g))  (identGene)
(pullSieve UU ( g')))
   (interSieve VV g UU).
Proof. refine (transf_Compos _ (interSieve_composeOuter _ _ _ g' identGene)).
refine (interSieve_congr (sieveTransf_Ident _) (reflexivity _) _).
refine (pullSieve_congr (sieveTransf_Ident _) _).
abstract (symmetry; exact: composGene_identGene).
Defined.

Lemma interSieve_congr_sieveEquiv G (VV1 VV2 : sieveFunctor G)  (vv:
sieveEquiv VV1 VV2)
G' (g1 g2 : 'Gene(G' ~> G)) (genEquiv: g1 == g2)
  (UU1 UU2 : sieveFunctor G') (uu: sieveEquiv UU1 UU2):
  sieveEquiv (interSieve VV1 g1 UU1) (interSieve VV2 g2 UU2).
Proof. unshelve eexists.
exact: (interSieve_congr vv genEquiv uu).
```

```
exact (interSieve_congr (_revSieveTransf_sieveEquiv vv)
  (symmetry genEquiv) (_revSieveTransf_sieveEquiv uu)).
abstract (intros; split; simpl; rewrite -> _injProp_sieveEquiv; reflexivity).
abstract(intros; split; simpl; rewrite -> _surProp_sieveEquiv; reflexivity).
Defined.

Definition pullSieve_congr_sieveEquiv G (VV1 VV2 : sieveFunctor G)  (vv:
sieveEquiv VV1 VV2)
G' (g1 g2 : 'Gene(G' ~> G)) (genEquiv: g1 == g2):
  sieveEquiv (pullSieve VV1 g1) (pullSieve VV2 g2)
  := @interSieve_congr_sieveEquiv G VV1 VV2 vv G' g1 g2 genEquiv _ _
(reflexivity _).

Lemma pullSieve_pullSieve_sieveEquiv G (VV : sieveFunctor G) G' (g : 'Gene(G'
~> G))
G'' (g' : 'Gene(G'' ~> G')): sieveEquiv (pullSieve (pullSieve VV g) g')
  (pullSieve VV (g' o>functor_[functor_ViewOb _] g)).
Proof. unshelve eexists.
exact: pullSieve_pullSieve.
exact: pullSieve_pullSieve_rev.
abstract(intros; split; cbn_transf; reflexivity).
abstract(intros H v; split; cbn_transf; first reflexivity;
last split; cbn_transf; first (rewrite -> (_wholeProp_interSieve v);
reflexivity);
     last reflexivity).
Defined.

Lemma pullSieve_ident_sieveEquiv G (VV : sieveFunctor G) :
  sieveEquiv (pullSieve VV identGene) VV.
Proof. unshelve eexists.
exact: pullSieve_ident.
exact: pullSieve_ident_rev.
abstract(intros; cbn_transf; reflexivity).
abstract(intros H v; split; cbn_transf; last reflexivity;
first rewrite -> _wholeProp_interSieve; apply: identGene_composGene).
Defined.

Definition interSieve_identSieve_sieveEquiv (G G': vertexGene)
(g: 'Gene( G' ~> G )) (WW : sieveFunctor G')
: sieveEquiv (interSieve (identSieve G) g WW) WW.
Proof. unshelve eexists. exact: interSieve_projFactor.
- {  unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
        exact v. exact ((v :>sieve_) :>transf_ (transf_ViewObMor g)).
        abstract (cbn_sieve; reflexivity).
  (* _congr_relTransf  *) abstract(move; move => x y Heq; cbn_transf; split;
        cbn_transf; rewrite -> Heq; reflexivity).
- (* _natural_transf *)
abstract(move; intros; cbn_sieve; split; cbn_sieve; first reflexivity;
```

```
    do 2 rewrite -> _natural_transf; reflexivity).
- (* _commute_sieveTransf *) abstract(move; intros; cbn_sieve; reflexivity).
}
- abstract (intros; cbn_transf; reflexivity).
- abstract (intros H v; cbn_transf; split; cbn_transf; first reflexivity;
  symmetry; apply: (_wholeProp_interSieve v)).
Defined.

(* TODO: REDO: instance of interSieve_identSieve_sieveEquiv *)
Definition pullSieve_identSieve_sieveEquiv (G G': vertexGene)
(g: 'Gene( G' ~> G ))
: sieveEquiv (pullSieve (identSieve G) g) (identSieve G').
Proof. unshelve eexists. exact: interSieve_projFactor.
- { unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
      exact (v :>sieve_). exact (v :>transf_ (transf_ViewObMor g)).
      abstract (cbn_sieve; reflexivity).
  (* _congr_relTransf  *) abstract(move; move => x y Heq;
     cbn_transf; split; cbn_transf; rewrite -> Heq; reflexivity).
- (* _natural_transf *)
  abstract(move; intros; cbn_sieve; split; cbn_sieve;
    first reflexivity; rewrite -> _natural_transf; reflexivity).
- (* _commute_sieveTransf *) abstract(move; intros; cbn_sieve; reflexivity).
}
- abstract (intros; cbn_transf; reflexivity).
- abstract (intros H v; cbn_transf; split; cbn_transf; first reflexivity;
  symmetry; apply: (_wholeProp_interSieve v)).
Defined.

Lemma interSieve_interSieve_rev G (VV : sieveFunctor G) G' (g : 'Gene(G' ~>
G))
(WW : sieveFunctor G')
G'' (g' : 'Gene(G'' ~> G')) (UU : sieveFunctor G'') :
sieveTransf  (interSieve VV (g' o>functor_[functor_ViewOb _] g) (interSieve
WW g' UU))
  (interSieve (interSieve VV g WW) g' UU) .
Proof. unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
  (* _factor_interSieve *) exact (_factor_interSieve (_factor_interSieve v)).
  (* _whole_interSieve *) refine ( v :>transf_  (interSieve_compos _ _ _ _ _)
).
  (* _wholeProp_interSieve *)  abstract (cbn_sieve; rewrite ->
_wholeProp_interSieve; reflexivity).
  (* _congr_relTransf  *) abstract (move; intros v1 v2; case; cbn_sieve;
  move => Heq_factor Heq_whole; split; cbn_sieve;
  [rewrite -> (factor_interSieve_Proper Heq_factor); reflexivity | ]; split;
cbn_sieve;
```

```
  [rewrite -> (whole_interSieve_Proper Heq_factor); reflexivity | rewrite ->
Heq_whole; reflexivity ]).
- (*  _natural_transf  *) abstract (move; split; cbn_sieve;
    first reflexivity;  split; cbn_sieve; reflexivity).
- (* _commute_sieveTransf *) abstract(move; reflexivity).
Defined.

Lemma interSieve_interSieve G (VV : sieveFunctor G) G' (g : 'Gene(G' ~> G))
(WW : sieveFunctor G')
G'' (g' : 'Gene(G'' ~> G')) (UU : sieveFunctor G'') :
sieveTransf  (interSieve (interSieve VV g WW) g' UU)
  (interSieve VV (g' o>functor_[functor_ViewOb _] g) (interSieve WW g' UU)).
Proof. unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
  (* _factor_interSieve *) refine ( v :>transf_  (interSieve_congr
(interSieve_projFactor _ _ _)
      (reflexivity _) (sieveTransf_Ident _)) ).
  (* _whole_interSieve *) exact: ((_whole_interSieve (_whole_interSieve v))).
  (* _wholeProp_interSieve *)  abstract(rewrite -> _wholeProp_interSieve;
   rewrite -> _functorialCompos_functor';
   setoid_rewrite <- _wholeProp_interSieve at 2; simpl; reflexivity).
  (* _congr_relTransf  *) abstract (move; intros ? ? [Heq_outer
Heq_inner];split;  cbn_sieve;
  first (split; cbn_sieve; first (rewrite -> Heq_outer; reflexivity);
        rewrite -> (factor_interSieve_Proper Heq_inner); reflexivity );
  last rewrite -> (whole_interSieve_Proper Heq_inner); reflexivity).
- (*  _natural_transf  *) abstract(move; split; cbn_sieve; first (split;
cbn_sieve; reflexivity);
                          last reflexivity).
- (* _commute_sieveTransf *) abstract(move; reflexivity).
Defined.

Lemma interSieve_interSieve_sieveEquiv  G (VV : sieveFunctor G) G' (g :
'Gene(G' ~> G))
(WW : sieveFunctor G')
G'' (g' : 'Gene(G'' ~> G')) (UU : sieveFunctor G'') :
sieveEquiv  (interSieve (interSieve VV g WW) g' UU)
  (interSieve VV (g' o>functor_[functor_ViewOb _] g) (interSieve WW g' UU)).
Proof. unshelve eexists.
exact: interSieve_interSieve.
exact: interSieve_interSieve_rev.
abstract(intros; split; cbn_transf; first (split; cbn_transf; reflexivity);
reflexivity).
abstract(intros H v; split; cbn_transf; first reflexivity;
last split; cbn_transf; reflexivity).
Defined.

(*  NOT LACKED, SEE GENERAL interSieve_interSieve_rev *)
```

```
Lemma interSieve_pullSieve_rev G (VV : sieveFunctor G) G' (g : 'Gene(G' ~>
G))
G'' (g' : 'Gene(G'' ~> G')) (UU : sieveFunctor G'') :
sieveTransf  (interSieve VV (g' o>functor_[functor_ViewOb _] g) UU)
  (interSieve (pullSieve VV g) g' UU) .
Proof. unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
  (* _factor_interSieve *) exact (_factor_interSieve v).
  (* _whole_interSieve *) { refine ( v :>transf_  _ ).
      refine (transf_Compos (interSieve_congr (sieveTransf_Ident _)
(reflexivity _)
                                    (sieveTransf_sieveFunctor _)) _).
      exact (pullSieve_compos _ _ _). }
  (* _wholeProp_interSieve *)  abstract(reflexivity).
  (* _congr_relTransf  *) abstract  (move; intros v1 v2; case; cbn_sieve;
  move => Heq_factor Heq_whole; split; cbn_sieve;
  [rewrite -> Heq_factor; reflexivity | ]; split; cbn_sieve;
  [rewrite -> Heq_factor; reflexivity | rewrite -> Heq_whole; reflexivity ]).
- (*  _natural_transf  *) abstract (move; split; cbn_sieve;
[reflexivity | split; cbn_sieve;
[ rewrite -> _functorialCompos_functor';
  rewrite -> _natural_transf; reflexivity | reflexivity ]]).
- (* _commute_sieveTransf *) abstract(move; reflexivity).
Defined.

Definition interSieve_image_rev (G : vertexGene)
(UU : sieveFunctor G)
(H : vertexGene) (u : 'Sieve( H ~> _ | UU ))
 (VV : sieveFunctor H)
: sieveTransf (interSieve UU (u :>sieve_)  VV) VV.
Proof. exact: interSieve_projFactor.
Defined.

Definition interSieve_image (G : vertexGene)
(UU : sieveFunctor G)
(H : vertexGene) (u : 'Sieve( H ~> _ | UU ))
 (VV : sieveFunctor H)
: sieveTransf VV (interSieve UU (u :>sieve_)  VV) .
Proof.  unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros K. unshelve eexists.
(* _fun_relTransf *) intros v. unshelve eexists.
      exact v. exact ((v :>sieve_) o>sieve_ u).
      abstract (cbn_sieve; rewrite -> _natural_transf; reflexivity).
(* _congr_relTransf  *) abstract(move; move => x y Heq; cbn_transf; split;
    cbn_transf; rewrite -> Heq; reflexivity).
- (* _natural_transf *)
abstract(move; intros; cbn_sieve; split; cbn_sieve; first reflexivity;
    rewrite <- _natural_transf, <- _functorialCompos_functor' ; reflexivity).
- (* _commute_sieveTransf *) abstract(move; intros; cbn_sieve; reflexivity).
```

```
Defined.

Definition interSieve_image_sieveEquiv (G : vertexGene)
(UU : sieveFunctor G)
(H : vertexGene) (u : 'Sieve( H ~> _ | UU ))
 (VV : sieveFunctor H)
 (UU_base: typeOf_baseSieve UU)
 : sieveEquiv VV (interSieve UU (u :>sieve_)  VV) .
Proof. unshelve eexists.
- exact:  interSieve_image.
-  exact: interSieve_image_rev.
- abstract (intros K v; cbn_transf; split; cbn_transf; first reflexivity;
apply: UU_base; unfold _rel_relType, equiv; simpl; rewrite <-
_natural_transf;
 symmetry; apply: (_wholeProp_interSieve v)).
 - abstract (intros; cbn_transf; reflexivity).
Defined.

Section sumSieve.

Section Section1.
Variables (G : vertexGene) (VV : sieveFunctor G).

Record typeOf_outer_sumSieve :=
  { _object_typeOf_outer_sumSieve :> vertexGene ;
    _arrow_typeOf_outer_sumSieve :> 'Sieve( _object_typeOf_outer_sumSieve ~>
G | VV ) }.

(* higher/congruent structure is possible... *)
Variables (WP_ : forall (object_: vertexGene) (outer_: 'Sieve( object_ ~> G |
VV )),
 sieveFunctor object_).


Record type_sumSieve H :=
  { _object_sumSieve : vertexGene ;
    _outer_sumSieve : 'Sieve( _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 (outer': 'Sieve( _object_sumSieve wv ~> G | VV ))
  (inner': (WP_  outer') H),
  outer' == _outer_sumSieve wv ->
  (* higher/congruent structure is possible... *)
  inner' :>sieve_ == (_inner_sumSieve wv) :>sieve_ ->
  rel_sumSieve wv
  {| _object_sumSieve := _ ;
  _outer_sumSieve := outer' ;
  _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 [outer3 inner3
Heq23 Heq23'];
      destruct Heq12 as [outer2 inner2 Heq12 Heq12']; simpl; constructor;
simpl;
      [ rewrite -> Heq23; simpl; rewrite -> Heq12; simpl; reflexivity
       | 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 => /= [outer_wv2 inner_wv2 Heq12 Heq12']; constructor;
simpl;
    [ rewrite -> Heq12; reflexivity
     | do 2 rewrite <- _natural_transf; rewrite -> Heq_h , Heq12';
reflexivity]).
  - (* typeOf_functorialCompos_functor *) abstract(intros H H' h H'' h'
[object_wv outer_wv inner_wv];
     simpl; constructor; simpl; [ reflexivity | rewrite ->
_functorialCompos_functor; reflexivity]).
  - (* typeOf_functorialIdent_functor *) abstract(intros H [object_wv
outer_wv inner_wv];
  simpl; constructor; simpl; [ reflexivity | 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 :>sieve_)).
    + (* Proper *) abstract(move; autounfold; simpl;
    intros wv1 wv2 Heq; tac_unsimpl;
```

```
    case: wv2 / Heq => /= [outer_wv2 inner_wv2 Heq12 Heq12']; tac_unsimpl;
rewrite -> Heq12;
    rewrite -> Heq12'; reflexivity).
  - (* typeOf_natural_transf *) move. cbn_functor. abstract(move;
cbn_functor; intros H H' h wv;
   rewrite -> _functorialCompos_functor';
  setoid_rewrite -> _natural_transf at 2; reflexivity). }
Defined.

Definition sumSieve_projOuter :
 sieveTransf sumSieve VV.
Proof. unshelve eexists. unshelve eexists.
- intros K. unshelve eexists.
  + (* _fun_relTransf *) intros wv. exact: ((_inner_sumSieve wv :>sieve_)
o>sieve_ (_outer_sumSieve wv)).
  + (* _congr_relTransf *) abstract(move;  intros wv1 wv2 [outer_wv2
inner_wv2 Heq_outer_wv2 Heq_inner_wv2];
  cbn_transf; rewrite -> Heq_outer_wv2, -> Heq_inner_wv2; reflexivity).
- (* _natural_transf *) abstract(move; intros; cbn_sieve;
    rewrite -> _functorialCompos_functor', -> _natural_transf; reflexivity).
- (* _commute_sieveTransf *) abstract(move; intros; simpl; rewrite <-
_natural_transf; reflexivity).
Defined.

End Section1.

Definition sumSieve_sectionPull :
forall (U : vertexGene) (UU : sieveFunctor U)
(VV_ : forall (H: vertexGene) (outer_: 'Sieve( H ~> U | UU )), sieveFunctor
H)
(H: vertexGene)
(u: 'Sieve( H ~> _ | UU )),
 sieveTransf (VV_ H u)
  (pullSieve (sumSieve VV_) (u:>sieve_)) .
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;
  first reflexivity; 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 : sieveFunctor U)
(VV_ : forall (H: vertexGene) (outer_: 'Sieve( H ~> U | UU )), sieveFunctor
H)
(H: vertexGene)
(u: 'Sieve( H ~> _ | UU )),
 transf (VV_ H u)  (sumSieve VV_) .
Proof. intros. exact: (transf_Compos (sumSieve_sectionPull _ _)
(pullSieve_projWhole _ _) ).
Defined.

End sumSieve.
(* Global Hint Unfold compos_sumSieve : poly. *)

Lemma sumSieve_congrTransf (G : vertexGene) (UU1  : sieveFunctor G)
G' ( UU2 : sieveFunctor G')
(uu : transf UU1 UU2)
(VV1_ : forall H : vertexGene, 'Sieve( H ~> _ | UU1 ) -> sieveFunctor H)
(VV2_ : forall H : vertexGene, 'Sieve( H ~> _ | UU2 ) -> sieveFunctor H)
(vv_ : forall (H: vertexGene) (u1: 'Sieve( H ~> _ | UU1 )),
 sieveTransf  (VV1_ _ u1) (VV2_ _ (u1 :>transf_ uu))) :
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: (_outer_sumSieve vu :>transf_ uu).
  (* _inner_sumSieve *) exact: (_inner_sumSieve vu :>transf_ (vv_ _ _)).
  (* _congr_relTransf  *) abstract(move; intros vu1 vu2 [outer_vu2 inner_vu2
Heq_outer_vu2 Heq_inner_vu2];
  simpl; constructor; simpl; [rewrite -> Heq_outer_vu2; reflexivity
  | do 2 rewrite -> _commute_sieveTransf; rewrite -> Heq_inner_vu2;
reflexivity  ]).
- (*  _natural_transf  *) abstract(intros K K' k vvu; cbn_sieve;
  constructor; simpl; [reflexivity | rewrite -> _natural_transf;
reflexivity]).
Defined.

Lemma sumSieve_congr (G : vertexGene) (UU1 UU2 : sieveFunctor G)
(uu : sieveTransf UU1 UU2)
(VV1_ : forall H : vertexGene, 'Sieve( H ~> _ | UU1 ) -> sieveFunctor H)
(VV2_ : forall H : vertexGene, 'Sieve( H ~> _ | UU2 ) -> sieveFunctor H)
(vv_ : forall (H: vertexGene) (u1: 'Sieve( H ~> _ | UU1 )),
sieveTransf  (VV1_ _ u1) (VV2_ _ (u1 :>transf_ uu))) :
sieveTransf  (sumSieve VV1_ ) (sumSieve VV2_).
Proof. unshelve eexists. (* _transf_sieveTransf *) exact:
sumSieve_congrTransf.
```

```
(* _commute_sieveTransf *) abstract(intros K vu; simpl; do 2 rewrite ->
_commute_sieveTransf; reflexivity).
Defined.

Lemma sumSieve_interSieve' (G : vertexGene) (UU : sieveFunctor G)
G' (g : 'Gene(G' ~> G))  (WW : sieveFunctor G')
(VV_ : forall H : vertexGene, 'Sieve( H ~> _ | (interSieve UU g WW) ) ->
sieveFunctor H)
G'' (g' : 'Gene(G'' ~> G'))
(pullVV_ := fun (H : vertexGene) (v : 'Sieve( H ~> _ | (interSieve UU (g'
o>gene g) (pullSieve WW g') ) )) =>
    VV_ _ ( v :>transf_ (interSieve_compos _ g _ g' (identSieve _))) ) :
sieveTransf  (sumSieve pullVV_ ) (pullSieve (sumSieve VV_) g').
Proof. unshelve eexists. unshelve eexists.
intros K. unshelve eexists. intros vu.
{ unshelve eexists. refine (_factor_interSieve (((_inner_sumSieve vu)
:>sieve_)
                                      o>functor_ (_factor_interSieve
(_outer_sumSieve vu)))).
unshelve eexists; cycle 1.
 refine ( (_outer_sumSieve vu):>transf_ (interSieve_compos _ g _ g'
(identSieve _)) ).
 refine (_inner_sumSieve vu).
 abstract (cbn_sieve; rewrite -> _wholeProp_interSieve; rewrite ->
_functorialCompos_functor'; reflexivity).
}

- abstract (subst pullVV_; move; intros vu1 vu2 [outer_vu2 inner_vu2
Heq_outer_vu2 Heq_inner_vu2]; cbn_sieve; split; cbn_sieve;
[rewrite -> Heq_inner_vu2; rewrite -> (factor_interSieve_Proper
(factor_interSieve_Proper Heq_outer_vu2)); reflexivity | ];
constructor; cbn_sieve; [ split; cbn_sieve; [rewrite ->
(whole_interSieve_Proper (factor_interSieve_Proper Heq_outer_vu2));
reflexivity
| rewrite -> (whole_interSieve_Proper Heq_outer_vu2); reflexivity]
| rewrite -> Heq_inner_vu2; reflexivity]).

- abstract(intros K K' k vu; cbn_sieve; split; cbn_sieve;
first (rewrite <- _natural_transf;
rewrite -> _functorialCompos_functor'; reflexivity);
reflexivity).

- abstract(intros K vu; simpl; reflexivity).
Defined.

Lemma sumSieve_pullSieve' (G : vertexGene) (UU : sieveFunctor G)
G' (g : 'Gene(G' ~> G))
(VV_ : forall H : vertexGene, 'Sieve( H ~> _ | (pullSieve UU g) ) ->
sieveFunctor H)
G'' (g' : 'Gene(G'' ~> G'))
```

```
(pullVV_ := fun (H : vertexGene) (v : 'Sieve( H ~> _ | (pullSieve UU (g'
o>gene g)) )) =>
    VV_ _ ( v :>transf_ (pullSieve_compos _ g g')) ) :
sieveTransf  (sumSieve pullVV_ ) (pullSieve (sumSieve VV_) g').
Proof. unshelve eexists. unshelve eexists.
intros K. unshelve eexists. intros vu.
{ unshelve eexists. refine (((_inner_sumSieve vu) :>sieve_) o>functor_
(_factor_interSieve (_outer_sumSieve vu))).
unshelve eexists; cycle 1.
 refine ( (_outer_sumSieve vu):>transf_ (pullSieve_compos _ g g') ).
 refine (_inner_sumSieve vu).
 abstract(cbn_sieve; rewrite -> _functorialCompos_functor'; reflexivity).
}

- abstract (subst pullVV_; move; intros vu1 vu2 [outer_vu2 inner_vu2
Heq_outer_vu2 Heq_inner_vu2]; cbn_sieve; split; cbn_sieve;
[rewrite -> Heq_inner_vu2; rewrite -> (factor_interSieve_Proper
Heq_outer_vu2); reflexivity | ];
constructor; cbn_sieve; [ split; cbn_sieve; [rewrite ->
(factor_interSieve_Proper Heq_outer_vu2); reflexivity
| rewrite -> (whole_interSieve_Proper Heq_outer_vu2); reflexivity]
| rewrite -> Heq_inner_vu2; reflexivity]).

- abstract(intros K K' k vu; cbn_sieve; split; cbn_sieve;
first (rewrite <- _natural_transf;
rewrite -> _functorialCompos_functor'; reflexivity);
reflexivity).

- abstract(intros K vu; simpl; reflexivity).
Defined.

(* sumSieve_pullSieve' ->  sumSieve_pullSieve *)
Lemma sumSieve_pullSieve (G : vertexGene) (UU : sieveFunctor G)
(VV_ : forall H : vertexGene, 'Sieve( H ~> _ | UU ) -> sieveFunctor H)
G' (g : 'Gene(G' ~> G))
(pullVV_ := fun (H : vertexGene) (v : 'Sieve( H ~> _ | (pullSieve UU g) )) =>
    VV_ _ (_whole_interSieve v) ) :
sieveTransf  (sumSieve pullVV_ ) (pullSieve (sumSieve VV_) g).
Proof. unshelve eexists. unshelve eexists.
intros K. unshelve eexists. intros vu.
{ unshelve eexists. refine (((_inner_sumSieve vu) :>sieve_) o>functor_
(_factor_interSieve (_outer_sumSieve vu))).
unshelve eexists; cycle 1.
 refine (_whole_interSieve (_outer_sumSieve vu)).
 refine (_inner_sumSieve vu).
- abstract(cbn_sieve; rewrite -> _wholeProp_interSieve;  rewrite ->
_functorialCompos_functor'; reflexivity). }
- abstract (subst pullVV_; move; intros vu1 vu2 [outer_vu2 inner_vu2
Heq_outer_vu2 Heq_inner_vu2]; cbn_sieve; split; cbn_sieve;
```

```
[rewrite -> Heq_inner_vu2; rewrite -> (factor_interSieve_Proper
Heq_outer_vu2); reflexivity | ];
constructor; cbn_sieve; [rewrite -> (whole_interSieve_Proper Heq_outer_vu2);
reflexivity
| rewrite -> Heq_inner_vu2; reflexivity ]).
- abstract (intros K K' k vu; cbn_sieve; split;
first (cbn_sieve; tac_unsimpl; rewrite <- _natural_transf;
rewrite -> _functorialCompos_functor'; reflexivity);
cbn_sieve;  reflexivity).
- abstract(intros K vu; simpl; reflexivity).
Defined.

(* TODO: KEEEP FOR GENERAL VIEW OBJECT*)
Definition sumSieve_interSieve_image_general
 (U : vertexGene) (UU : sieveFunctor U)
 (H : vertexGene) (u : 'Sieve( H ~> _ | UU ))
 (WW : sieveFunctor H)
(VV_ : forall object_ : vertexGene,
        'Sieve( object_ ~> _ | (interSieve UU (u :>sieve_) WW) ) ->
sieveFunctor object_)
        (K : vertexGene) (w : 'Sieve( K ~> _ | WW )) :
sieveTransf (VV_ _ (w :>transf_ interSieve_image u WW)) (pullSieve (sumSieve
VV_) (w :>sieve_) ) .
Proof. unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros L. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
    (* _factor_interSieve *) exact: (v :>sieve_).
    (* _whole_interSieve *) unshelve eexists.
      * (* _object_sumSieve *) exact: K.
      * (* _outer_sumSieve *) exact (w :>transf_ interSieve_image u WW).
      * (* _inner_sumSieve *) exact: v.
    (* _wholeProp_interSieve *) abstract (cbn_sieve; reflexivity).
  (* _congr_relTransf  *) abstract (move; intros v1 v2 Heq_v; unshelve
eexists; cbn_sieve;
  first (rewrite -> Heq_v; reflexivity);
   split; cbn_sieve; first reflexivity; last (rewrite -> Heq_v;
reflexivity)).
- (*  _natural_transf  *) abstract (move; unshelve eexists; cbn_sieve; first
(rewrite -> _natural_transf; reflexivity);
 reflexivity).
- (* _commute_sieveTransf *) abstract (move; intros; cbn_sieve; reflexivity).
Defined.

Definition sumSieve_interSieve_image
 (U : vertexGene) (UU : sieveFunctor U)
 (H : vertexGene) (u : 'Sieve( H ~> _ | UU ))
(VV_ : forall object_ : vertexGene,
        'Sieve( object_ ~> _ | (pullSieve UU (u :>sieve_) ) ) -> sieveFunctor
object_) :
```

```
sieveTransf (VV_ _ (identGene :>transf_ interSieve_image u (identSieve _)))
(sumSieve VV_) .
Proof. unshelve eexists. (* _transf_sieveTransf *)  unshelve eexists.
- (* _arrows_transf *) intros K. unshelve eexists.
  (* _fun_relTransf *) intros v. unshelve eexists.
    * (* _object_sumSieve *) exact: H.
    * (* _outer_sumSieve *) exact: (identGene :>transf_ interSieve_image u
(identSieve _)).
    * (* _inner_sumSieve *) exact: v.
  (* _congr_relTransf  *) abstract (move; intros v1 v2 Heq_v; unshelve
eexists; cbn_sieve;
  first reflexivity; last (rewrite -> Heq_v; reflexivity)).
- (*  _natural_transf  *) abstract (move; unshelve eexists; cbn_sieve;
reflexivity).
- (* _commute_sieveTransf *) abstract (move; intros; cbn_sieve; (* TODO: HERE
*)
exact: identGene_composGene).
Defined.

Definition imageSieve (U : vertexGene) (UU : sieveFunctor U) : (sieveFunctor
U).
Proof. unshelve eexists.
{ (* functor *) unshelve eexists.
  - (* typeOf_objects_functor *) intros H. exact: (@compatRelType _ UU H).
  - (* _arrows_functor *)  move. intros H H'.
    (* relFunctor *) unshelve eexists.
    + (* -> *)  simpl. intros h u. exact: (h o>sieve_ u).
    + (* Proper *) abstract(move; cbn_transf;
    intros h1 h2 Heq_h u1 u2 Heq; rewrite -> Heq_h; move: Heq; unfold
_rel_relType, equiv;
    simpl; intros Heq; do 2 rewrite <- _natural_transf; rewrite -> Heq;
reflexivity).
  - (* typeOf_functorialCompos_functor *) abstract (intros H H' h H'' h' u;
    unfold _rel_relType, equiv; simpl;
    do 3 rewrite <- _natural_transf; exact: _functorialCompos_functor).
  - (* typeOf_functorialIdent_functor *) abstract(intros H u; unfold
_rel_relType, equiv; simpl;
      rewrite <- _natural_transf; exact:  _functorialIdent_functor). }
{ (* transf *) unshelve eexists.
  - (* typeOf_arrows_transf *) intros H. unshelve eexists.
    + (* -> *) simpl. intros u. exact: (u :>sieve_).
    + (* Proper *) abstract(move; cbn_transf;
                  intros u1 u2 Heq; exact: Heq).
  - (* typeOf_natural_transf *) abstract (move; cbn_transf; intros H H' h u;
    exact: _natural_transf).  }
Defined.

Inductive isCover : forall (U : vertexGene), (sieveFunctor U) -> Type :=
```

```
| BaseSieve_isCover : forall (U : vertexGene) (UU : sieveFunctor U) (UU_base
: typeOf_baseSieve UU ),
    baseSieve UU_base -> isCover UU

| IdentSieve_isCover : forall (G : vertexGene),
 isCover (identSieve G)

| InterSieve_isCover : forall (G : vertexGene) (VV : sieveFunctor G)
    (G' : vertexGene) (g : 'Gene( G' ~> G )) (UU : sieveFunctor G'),
     isCover  VV -> isCover (interSieve VV g UU)

| SumSieve_isCover : forall (G : vertexGene) (VV : sieveFunctor G)
(WP_ : forall (object_: vertexGene) (outer_: 'Sieve( object_ ~> G | VV )),
sieveFunctor object_),
   isCover VV ->
  (forall G' v, isCover (WP_ G' v)) -> isCover (sumSieve WP_).

Record type_Restrict (F : functor) (U : vertexGene) (UU : sieveFunctor U)
(G : vertexGene) : Type :=
{ _indexer_type_Restrict : 'Gene( G ~> U ) ;
  _sieve_type_Restrict : sieveFunctor G;
 _data_type_Restrict :> transf (interSieve UU _indexer_type_Restrict
_sieve_type_Restrict) F;
 _congr_type_Restrict : forall H (u1 u2 : 'Sieve(H ~> _| interSieve UU
_indexer_type_Restrict _sieve_type_Restrict )),
  _factor_interSieve u1 == _factor_interSieve u2 ->
  u1 :>transf_ _data_type_Restrict  ==  u2 :>transf_ _data_type_Restrict }.

Record equiv_Restrict (F : functor) (U : vertexGene) (UU : sieveFunctor U)
(G : vertexGene) (f1_ f2_: type_Restrict F UU G)  :=
{ _indexerEquiv_equiv_Restrict : _indexer_type_Restrict f1_ ==
_indexer_type_Restrict f2_ ;
  _sieveEquiv_equiv_Restrict : sieveEquiv (_sieve_type_Restrict f1_)
(_sieve_type_Restrict f2_) ;
  _dataProp_equiv_Restrict : forall (H : vertexGene)
      (c : 'Sieve( H ~> _ | interSieve UU (_indexer_type_Restrict f1_)
(_sieve_type_Restrict f1_)  )),
  c :>transf_ f1_ ==
  (c :>transf_ interSieve_congr (sieveTransf_Ident UU)
_indexerEquiv_equiv_Restrict _sieveEquiv_equiv_Restrict)
  :>transf_ f2_ }.

Instance equiv_Restrict_Equivalence (F : functor) (U : vertexGene) (UU :
sieveFunctor U)
(G : vertexGene) : Equivalence (@equiv_Restrict F U UU G).
Proof. unshelve eexists.
* abstract(intros f1_ ; exists (reflexivity _) (reflexivity _) ; cbn_transf;
intros K c;
rewrite -> _congr_relTransf; first reflexivity; split; simpl; reflexivity).
```

```
* abstract(intros f1_ f2_ [indexerEquiv_ sieveEquiv_ dataProp_]; exists
(symmetry indexerEquiv_) (symmetry sieveEquiv_);
intros K c; rewrite -> dataProp_;
rewrite -> _congr_relTransf; first reflexivity; split; simpl; first rewrite -
> _injProp_sieveEquiv; reflexivity).
* abstract(intros f1_ f2_ f3_  [indexerEquiv12 sieveEquiv12_ Heq12]
[indexerEquiv23 sieveEquiv23_ Heq23];
exists (transitivity indexerEquiv12 indexerEquiv23)
(transitivity sieveEquiv12_ sieveEquiv23_) ;
intros K c; rewrite -> Heq12, Heq23;
rewrite -> _congr_relTransf; first reflexivity; split; simpl; reflexivity).
Qed.

Definition functor_Restrict (F : functor) (U : vertexGene) (UU : sieveFunctor
U) : functor.
Proof. unshelve eexists.
- (* typeOf_objects_functor *) intros G.  unshelve eexists. exact
(type_Restrict F UU G).
   (* relation *) exact (@equiv_Restrict F U UU G).
   (* Equivalence *) exact: equiv_Restrict_Equivalence.
- (* _arrows_functor *) intros H H'. unshelve eexists.
  (* _fun_relFunctor *) simpl. intros h f_. unshelve eexists.
    (* _indexer_type_Restrict *) exact (h o>functor_[functor_ViewOb U]
(_indexer_type_Restrict f_)).
    (* _sieve_type_Restrict *) exact (pullSieve (_sieve_type_Restrict f_) h).
    (* _data_type_Restrict *) exact (transf_Compos (interSieve_compos _ _ _ _
(identSieve _)) (_data_type_Restrict f_)).
    (* _congr_type_Restrict *) abstract(cbn_transf; intros K u1 u2 Heq_u;
      apply: _congr_type_Restrict; cbn_transf; rewrite ->
(whole_interSieve_Proper Heq_u); reflexivity).
  (* _congr_relFunctor *) abstract(move; cbn_transf; intros h1 h2 Heq_h f1_
f2_  [indexerEquiv sieveEquiv_ Heq];
  unshelve eexists; first (cbn_transf; rewrite -> Heq_h, indexerEquiv;
reflexivity);
  cbn_transf; first (exact: (pullSieve_congr_sieveEquiv sieveEquiv_ Heq_h));
  last intros K c; rewrite -> Heq; rewrite -> _congr_relTransf;
        first reflexivity; split; simpl; reflexivity).
- (* _functorialCompos_functor *) abstract (move; cbn_transf;
intros  G G' g G'' g' f_; unshelve eexists; first(cbn_transf;
rewrite -> _functorialCompos_functor'; reflexivity);
  first (cbn_transf; exact: pullSieve_pullSieve_sieveEquiv);
  last (cbn_transf; intros H c; rewrite -> _congr_relTransf;
        first reflexivity; split; cbn_transf; reflexivity)).
- (* _functorialIdent_functor *) abstract(move; cbn_transf; intros  G f_;
 unshelve eexists; first(cbn_transf;
rewrite -> _functorialIdent_functor; reflexivity);
  first (cbn_transf; exact: pullSieve_ident_sieveEquiv);
  last (cbn_transf; intros H c; rewrite -> _congr_relTransf;
        first reflexivity; split; cbn_transf; reflexivity)).
Defined.
```

```
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_Restrict ?F ?U ?UU ?H ?x ?y] ] =>
 change (@equiv_Restrict F U UU H x y) with
(@equiv _ _ (@_equiv_relType ( (@functor_Restrict F U UU) H )) x y)
end.

Instance indexer_type_Restrict_Proper :
forall [F : functor] [U : vertexGene] [UU : sieveFunctor U] [G : vertexGene],
Proper (equiv ==> equiv) (@_indexer_type_Restrict F U UU G).
Proof.    intros. move. intros f1_ f2_ [indexerEquiv_ sieveEquiv_ Heq_f].
  exact: indexerEquiv_.
Qed.

(* TODO: note that ff and uu are never non-identity at the same time; \
so the grammatical transformation instead should be  transf_RestrictCast  *)
Definition transf_RestrictMor (F E : functor)
(ff : transf F E) (U : vertexGene) (UU VV : sieveFunctor U)
(uu : sieveTransf VV UU) :
transf (functor_Restrict F UU) (functor_Restrict E VV).
Proof. intros. unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  + (* _fun_relTransf *) intros f_. unshelve eexists.
    * (* _indexer_type_Restrict *) exact: (_indexer_type_Restrict f_).
    * (* _sieve_type_Restrict *) exact: (_sieve_type_Restrict f_).
    * (* _data_type_Restrict *) exact (transf_Compos (interSieve_congr uu
(reflexivity _) (sieveTransf_Ident _))
                                          (transf_Compos (_data_type_Restrict
f_) ff)).
    * (* _congr_type_Restrict *) abstract (intros K u1 u2  Heq_u; cbn_transf;
apply: _congr_relTransf;
      apply: _congr_type_Restrict; cbn_transf; rewrite -> Heq_u;
reflexivity).
  + (* _congr_relTransf  *) abstract (move; intros f1_ f2_ [indexerEquiv
sieveEquiv_ Heq]; unshelve eexists; cbn_transf;
  first exact: indexerEquiv; first exact: sieveEquiv_;
  last intros K c;  cbn_transf; apply: _congr_relTransf;
  rewrite -> Heq; apply: _congr_type_Restrict; cbn_transf; reflexivity).
```

```
- (* _natural_transf *) abstract (move; intros; unshelve eexists; cbn_transf;
first exact: (reflexivity _); first exact: (reflexivity _);
cbn_sieve; intros H c; apply: _congr_relTransf;
apply: _congr_type_Restrict; cbn_transf; reflexivity).
Defined.

Definition ident_functor_Restrict G (U : vertexGene) (UU : sieveFunctor U)
(u: 'Sieve( G ~> _ | UU ))
: functor_Restrict (functor_ViewOb G) UU G.
Proof.  unshelve eexists. exact: (u :>sieve_). exact: (identSieve _).
unshelve eexists.
- (* _arrows_transf *)  intros H. unshelve eexists.
  + (* _fun_relTransf *) intros g. exact: (g :>sieve_).
  + (* _congr_relTransf *) abstract(solve_proper).
- (* _natural_transf *) abstract (move; intros; cbn_transf; exact:
_natural_transf).
- (* _congr_type_Restrict *) abstract (intros; cbn_sieve; assumption).
Defined.

Definition ident_functor_Restrict_natural G (U : vertexGene) (UU :
sieveFunctor U)
 (u: 'Sieve( G ~> _ | UU )) G' (g:  'Gene( G' ~> G )):
g o>functor_ ident_functor_Restrict (u) ==
ident_functor_Restrict (g o>sieve_  u)
:>transf_ transf_RestrictMor (transf_ViewObMor g) (sieveTransf_Ident UU).
Proof.    unshelve eexists. cbn_transf; cbn_functor.
  rewrite <- _natural_transf. reflexivity.
  - cbn_sieve. exact: pullSieve_identSieve_sieveEquiv.
  - cbn_transf; intros H c. cbn_sieve. exact: (_wholeProp_interSieve
(_factor_interSieve c)).
Qed.

Instance ident_functor_Restrict_Proper G U UU
: Proper (equiv ==> equiv) (@ident_functor_Restrict G U UU).
Proof. move. intros u1 u2 Heq. unshelve eexists.
 - simpl. rewrite -> Heq; reflexivity.
 - cbn_sieve. reflexivity.
 - intros K c; reflexivity.
Qed.

Definition functor_Restrict_interSieve  (U : vertexGene) (UU : sieveFunctor
U)
(F : functor) G (g : 'Gene(G ~> U)) (VV : sieveFunctor G)
(* (uv: sieveTransf (pullSieve UU g) VV) *) :
transf (functor_Restrict F VV) (functor_Restrict F UU).
Proof.  unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  (* _fun_relTransf *) intros f_. { unshelve eexists.
  - (* _indexer_type_Restrict *) exact: (_indexer_type_Restrict f_
o>functor_[functor_ViewOb _] g).
```

```
  - (* _sieve_type_Restrict *) exact: (interSieve VV (_indexer_type_Restrict
f_) (_sieve_type_Restrict f_) ) .
  - (* _data_type_Restrict *) refine (transf_Compos (interSieve_projFactor _
_ _) (_data_type_Restrict f_)).
  - (* _congr_type_Restrict *) abstract (intros K u1 u2 Heq_u; cbn_transf;
    apply: _congr_type_Restrict; cbn_transf;  rewrite ->
(factor_interSieve_Proper Heq_u); reflexivity). }
   (* _congr_relTransf  *) abstract(move; intros f1_ f2_ [indexerEquiv_
sieveEquiv_ Heq_];
    unshelve eexists; cbn_transf;
    first abstract (rewrite -> indexerEquiv_; reflexivity);
    first exact: (interSieve_congr_sieveEquiv (reflexivity _) indexerEquiv_
sieveEquiv_);
    last intros K c; rewrite -> Heq_; apply: _congr_relTransf;
      split; cbn_transf; reflexivity).
- (*  _natural_transf  *) abstract (intros H' H h f_; unshelve eexists;
cbn_sieve;
first (rewrite -> _functorialCompos_functor'; reflexivity);
first exact: interSieve_interSieve_sieveEquiv;
last intros K c; apply: _congr_relTransf; split; cbn_sieve;  reflexivity).
Defined.

Record type_Sheafified (F : functor)
(G : vertexGene) : Type :=
{ _sieve_type_Sheafified : sieveFunctor G ;
 _data_type_Sheafified :> transf _sieve_type_Sheafified F;
  _compat_type_Sheafified : forall (I : vertexGene), Proper ((@equiv _ _
(@_equiv_relType (compatRelType _ _)))
   ==> (@equiv _ _ (@_equiv_relType _))) (_arrows_transf
_data_type_Sheafified I) }.

Record equiv_Sheafified (F : functor)
(G : vertexGene) (f1_ f2_: type_Sheafified F G)  :=
{ conflSieve_Sheafified : sieveFunctor G  ;
 conflTransf1_Sheafified : sieveTransf conflSieve_Sheafified
(_sieve_type_Sheafified f1_) ;
 conflTransf2_Sheafified : sieveTransf conflSieve_Sheafified
(_sieve_type_Sheafified f2_) ;
 conflEquiv_Sheafified : forall (J : vertexGene) (c : 'Sieve( J ~> _ |
conflSieve_Sheafified )),
  (c :>transf_ conflTransf1_Sheafified) :>transf_ (_data_type_Sheafified f1_)
==
  (c :>transf_ conflTransf2_Sheafified) :>transf_ (_data_type_Sheafified f2_)
}.

Instance equiv_Sheafified_Equivalence (F : functor)
(G : vertexGene) : Equivalence (@equiv_Sheafified F G).
Proof. unshelve eexists.
- abstract (intros f1_ ; eexists (_sieve_type_Sheafified f1_)
(sieveTransf_Ident _) (sieveTransf_Ident _); reflexivity).
```

```
- abstract (intros f1_ f2_ [conflSieve conflTransf1 conflTransf2 Heq];
    exists conflSieve conflTransf2 conflTransf1; symmetry; exact: Heq).
- abstract (intros f1_ f2_ f3_  [conflSieve12 conflTransf1 conflTransf2
Heq12]
[conflSieve23 conflTransf2' conflTransf3 Heq23];
exists (meetSieve conflSieve12 conflSieve23)
 (sieveTransf_Compos (meetSieve_projWhole _ _) conflTransf1)
 (sieveTransf_Compos (meetSieve_projFactor _ _) conflTransf3);
intros H c; cbn_sieve; tac_unsimpl; rewrite -> Heq12; rewrite <- Heq23;
apply _compat_type_Sheafified; move; rewrite -/(equiv _ _); rewrite ->
_commute_sieveTransf; rewrite -> _commute_sieveTransf;
rewrite -> _wholeProp_interSieve; (* FUNCTOR/TRANSF PROBLEM *) exact:
identGene_composGene).
Qed.

Definition functor_Sheafified (F : functor) : functor.
Proof. unshelve eexists.
- (* typeOf_objects_functor *) intros G. unshelve eexists. exact
(type_Sheafified F G).
  + (* relation *) exact (@equiv_Sheafified F G).
  + (* Equivalence *) exact: equiv_Sheafified_Equivalence.
- (* _arrows_functor *) intros H H'. unshelve eexists.
  (* _fun_relFunctor *) simpl. intros h f_. unshelve eexists.
  exact: (pullSieve (_sieve_type_Sheafified f_) h).
  exact (transf_Compos (pullSieve_projWhole _ _) (_data_type_Sheafified f_)).
  abstract(intros I v v' Heq;  cbn_transf; apply: _compat_type_Sheafified;
  move: Heq; unfold  _rel_relType, equiv; simpl; intros Heq;
  do 2 rewrite -> _wholeProp_interSieve; rewrite -> Heq; reflexivity).
  (* _congr_relFunctor *) abstract(move; simpl; intros h1 h2 Heq_h f1_ f2_
  [conflSieve12 conflTransf1 conflTransf2 Heq12]; simpl;
  exists (pullSieve conflSieve12 h1)
    (pullSieve_congr conflTransf1 (reflexivity _) )
    (pullSieve_congr conflTransf2 Heq_h );
  intros K c; cbn -[_rel_relType]; rewrite -> Heq12; reflexivity).
- (* _functorialCompos_functor *) abstract(move; simpl; intros  G G' g G'' g'
f_;
  unshelve eexists;
 first exact (pullSieve (pullSieve ((_sieve_type_Sheafified f_)) g) g');
 first exact  (sieveTransf_Ident _ );
 first (simpl; exact (pullSieve_pullSieve _ _ _));
   intros K c; simpl; tac_unsimpl; reflexivity).
- (* _functorialIdent_functor *) abstract(move; simpl; intros G f_; unshelve
eexists;
 first exact (pullSieve (_sieve_type_Sheafified f_) identGene); simpl;
 first exact  (sieveTransf_Ident _ );
 first exact (pullSieve_ident _ );
  intros K c; simpl; tac_unsimpl; reflexivity).
Defined.

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_Restrict ?F ?U ?UU ?H ?x ?y] ] =>
 change (@equiv_Restrict F U UU H x y) with
(@equiv _ _ (@_equiv_relType ( (@functor_Restrict F U UU) H )) x y)
| [ |- context [@equiv_Sheafified ?F ?U ?UU ?H ?x ?y] ] =>
 change (@equiv_Sheafified F U UU H x y) with
(@equiv _ _ (@_equiv_relType ( (@functor_Sheafified F U UU) H )) x y)
end.

Definition relation_transf (F E : functor) : crelation (transf F E).  (* in
context of assuming congr *)
intros ee1 ee2. exact (forall G (f1 f2 : F G), f1 == f2 -> f1 :>transf_ ee1
== f2 :>transf_ ee2).
Defined.

Instance equiv_transf (F E : functor) : Equivalence (@relation_transf F E).
unshelve eexists;
first (move; intros; move; intros ? ? ? ->; reflexivity);
first (move; intros ? ? Heq; move; intros; symmetry; apply: Heq; symmetry;
assumption);
 move; intros ? ? ? Heq1 Heq2; move; intros; etransitivity;
   [apply:Heq1; eassumption
    | apply: Heq2; reflexivity].
Qed.

Definition rel_transf (F E : functor) : relType.
exists (transf F E) (@relation_transf F E). exact (@equiv_transf F E).
Defined.

Definition transf_RestrictMor_pullSieve
 (U : vertexGene) (UU : sieveFunctor U)  (F : functor) (G : vertexGene)
(f_: functor_Restrict F UU G) (G' : vertexGene) (g: 'Gene( G' ~> G)) :
functor_Restrict F (pullSieve UU (g o>gene _indexer_type_Restrict f_ )) G'.
Proof.
  unshelve eexists.
  - exact: (@identGene G').
  - exact: (pullSieve (_sieve_type_Restrict f_) g).
  -  refine (transf_Compos (interSieve_composeOuter_ident _ _ _ _)
(_data_type_Restrict f_)).
```

```
  - abstract (intros H u1 u2 Heq_u; cbn_transf; apply: _congr_type_Restrict;
     rewrite -> (whole_interSieve_Proper Heq_u); reflexivity).
Defined.

Section Gluing_typeOf.

Variables (U : vertexGene) (UU : sieveFunctor U) (UU_base: typeOf_baseSieve
UU)
(VV_ : forall H : vertexGene, 'Sieve( H ~> _ | UU ) -> sieveFunctor H).

Definition typeOf_sieveCongr :=
  forall (object_ : vertexGene)
  (outer_ outer_' : 'Sieve( object_ ~> _ | UU )),
outer_  == outer_' ->
sieveEquiv (VV_  outer_) (VV_  outer_').

Definition typeOf_sieveNatural :=
  forall (object_ : vertexGene)
  (outer_ : 'Sieve( object_ ~> _ | UU ))
  (K : vertexGene) (w : 'Gene( K ~> object_ )),
(* TODO: sieveEquiv? *) sieveTransf (VV_  (w o>sieve_ outer_))
  (pullSieve (VV_  outer_) w).

Variables (VV_congr : typeOf_sieveCongr)
  (VV_natural : typeOf_sieveNatural)   (F E : functor)
  (ee_ : forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU )),
        transf (functor_Restrict F (VV_ u)) (functor_Sheafified E)).

Definition typeOf_gluingCongr :=
forall (H : vertexGene) (u1 u2 : 'Sieve( H ~> _ | UU ))
  (K : vertexGene) (f1_ : functor_Restrict F (VV_  u1) K)
  (f2_ : functor_Restrict F (VV_  u2) K) (Hequ : u1 == u2)
(Heq_f : f1_ == f2_ :>transf_ transf_RestrictMor (transf_Ident F) (VV_congr
Hequ) ),
  (f1_ :>transf_ ee_  u1) == (f2_ :>transf_ ee_  u2).

Definition typeOf_gluingNatural  :=
forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU ))
  (K : vertexGene) (f_ : functor_Restrict F (VV_ u) K)
  (K' : vertexGene) (k : 'Gene( K' ~> K )),
k o>functor_ (f_ :>transf_ ee_ u) ==
(transf_RestrictMor_pullSieve f_ k
  :>transf_ transf_RestrictMor (transf_Ident F)
            (VV_natural u  (k o>gene _indexer_type_Restrict f_)))
:>transf_ ee_ ((k o>gene _indexer_type_Restrict f_) o>sieve_ u).

Definition typeOf_gluingCompat  :=
forall (H1 : vertexGene) (u1 : 'Sieve( H1 ~> _ | UU ))
(K1 : vertexGene) (f1_ : functor_Restrict F (VV_ u1) K1)
(H2 : vertexGene) (u2 : 'Sieve( H2 ~> _ | UU ))
```

```
(K2 : vertexGene) (f2_ : functor_Restrict F (VV_ u2) K2)
(I : vertexGene)
(w1 : 'Sieve( I ~> K1 | _sieve_type_Sheafified (f1_ :>transf_ ee_ u1) ))
(w2 : 'Sieve( I ~> K2 | _sieve_type_Sheafified (f2_ :>transf_ ee_ u2) ))
(Heq_wu : ((w1 :>sieve_) o>functor_[functor_ViewOb _] _indexer_type_Restrict
f1_) o>functor_ u1
      ==  ((w2 :>sieve_) o>functor_[functor_ViewOb _] _indexer_type_Restrict
f2_) o>functor_ u2 )
(Heq_f_ : ( (transf_RestrictMor_pullSieve f1_ (w1 :>sieve_))
                    :>transf_ transf_RestrictMor (transf_Ident _) (VV_natural
_  _ ) )
      == ( (transf_RestrictMor_pullSieve f2_ (w2  :>sieve_))
              :>transf_ transf_RestrictMor (transf_Ident _) (VV_natural  _ _
) )
          :>transf_ transf_RestrictMor (transf_Ident _) (VV_congr   Heq_wu)
),
    w1 :>transf_ (f1_ :>transf_ ee_  u1) ==
    w2 :>transf_ (f2_ :>transf_ ee_  u2).

Lemma gluingNatural_identGene_of_gluingNatural
(ee_natural : typeOf_gluingNatural) : forall (H : vertexGene) (u : 'Sieve( H
~> _ | UU ))
(K : vertexGene) (f_ : functor_Restrict F (VV_  u) K) ,
(f_ :>transf_ ee_  u) ==
(transf_RestrictMor_pullSieve f_ identGene
:>transf_ transf_RestrictMor (transf_Ident F)
           (VV_natural  u  (identGene o>gene _indexer_type_Restrict f_)))
:>transf_ ee_  ((identGene o>gene _indexer_type_Restrict f_) o>sieve_ u).
Proof. intros. etransitivity. symmetry; apply: _functorialIdent_functor.
etransitivity. apply: ee_natural. reflexivity.
Qed.

End Gluing_typeOf.

Definition transf_Gluing_lemma :
forall (U : vertexGene) (UU : sieveFunctor U)
(VV_ : forall (H: vertexGene) (outer_: 'Sieve( H ~> U | UU )), sieveFunctor
H)
(F : functor)
(G: vertexGene)
(f_: functor_Restrict F (sumSieve VV_) G)
(H: vertexGene)
(u: 'Sieve( H ~> _ | interSieve UU (_indexer_type_Restrict f_)
(_sieve_type_Restrict f_) )),
functor_Restrict F
  (VV_ H (u :>transf_ interSieve_projWhole UU (_indexer_type_Restrict f_)
(_sieve_type_Restrict f_))) H.
Proof. unshelve eexists.
- (* _indexer_type_Restrict *) exact: (@identGene H).
```

```
- (* _sieve_type_Restrict *) exact: (pullSieve (_sieve_type_Restrict f_) (u
:>sieve_) ).
- (* _data_type_Restrict *)
(* transf
  (interSieve (VV_ H (u :>transf_ interSieve_projWhole UU
(_indexer_type_Restrict f_) (_sieve_type_Restrict f_)))
     identGene (pullSieve (_sieve_type_Restrict f_) (u :>sieve_))) F *)
refine (transf_Compos _ (_data_type_Restrict f_)).
refine (transf_Compos (interSieve_congr (sumSieve_sectionPull _ _)
(reflexivity _) (sieveTransf_Ident _)) _).
refine (transf_Compos (interSieve_congr
  (pullSieve_congr (sieveTransf_Ident _) (_wholeProp_interSieve u))
  (reflexivity _) (sieveTransf_Ident _)) _).
exact: interSieve_composeOuter_ident.

- (* _congr_type_Restrict *) abstract (intros K v1 v2 Heq_v;
cbn_transf; apply: _congr_type_Restrict; cbn_transf;
rewrite -> (whole_interSieve_Proper Heq_v); reflexivity).
Defined.

Arguments transf_Gluing_lemma [_ _ _ _ _] f_ [_] u.

Definition transf_Gluing :
forall (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
(VV_ : forall H : vertexGene, 'Sieve( H ~> _ | UU ) -> sieveFunctor H)
(VV_congr : typeOf_sieveCongr VV_)
(VV_natural : typeOf_sieveNatural VV_)
  (F E : functor)
  (ee_ : forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU )),
        transf (functor_Restrict F (VV_ H u)) (functor_Sheafified E))
  (ee_congr : typeOf_gluingCongr VV_congr ee_)
(* ee_natural used in code only not sense *)
(ee_natural : typeOf_gluingNatural VV_natural ee_)
(ee_compat : typeOf_gluingCompat VV_congr VV_natural ee_),
transf (functor_Restrict F (sumSieve VV_)) (functor_Sheafified E).
Proof. unshelve eexists.
- (* _arrows_transf *)  intros G. unshelve eexists.
  + (* _fun_relTransf *) intros f_. unshelve eexists.
    * { (* _sieve_type_Sheafified *)
      - (* sieveFunctor G *) refine (@sumSieve G  (interSieve UU
(_indexer_type_Restrict f_) (_sieve_type_Restrict f_) ) _).
      - (* sieveFunctor H *) intros H u. refine (_sieve_type_Sheafified ( _
:>transf_ ee_ H (u :>transf_ interSieve_projWhole _ _ _) )).
      - (* functor_Restrict F (VV_ H (u :>transf_ interSieve_projWhole UU
(_indexer_type_Restrict f_) (_sieve_type_Restrict f_))) H *)
        exact: transf_Gluing_lemma. }
    * { (* _data_type_Sheafified *) unshelve eexists.
          + (* _arrows_transf *)  intros H. unshelve eexists.
            * (* _fun_relTransf *) intros wu.
```

```
            refine ( (_inner_sumSieve wu) :>transf_ (_data_type_Sheafified (
(transf_Gluing_lemma _ (_outer_sumSieve wu))
                :>transf_ ee_ _ ((_outer_sumSieve wu) :>transf_
interSieve_projWhole _ _ _) )) ).
            * (* _congr_relTransf *) abstract(move; cbn_sieve;
            intros wu1 wu2 [outer_wu2 inner_wu2 Heq_outer_wu2 Heq_inner_wu2];
cbn_sieve;
            unshelve apply: ee_compat;
            first abstract (cbn_; rewrite -> Heq_outer_wu2 , Heq_inner_wu2;
reflexivity);
            cbn_; unshelve eexists; first reflexivity;
            [ cbn_transf; refine (pullSieve_congr_sieveEquiv
(pullSieve_congr_sieveEquiv (reflexivity _) _) _);
              first abstract (rewrite -> Heq_outer_wu2; reflexivity);
              abstract (rewrite -> Heq_inner_wu2; reflexivity)
            |  (* no use _congr_type_Restrict *)  (cbn_transf; intros K c;
              apply: _congr_relTransf; cbn_transf;
              split; cbn_transf; first reflexivity;
              split; cbn_transf; first  (rewrite -> Heq_outer_wu2;
reflexivity);
              rewrite -> _wholeProp_interSieve, transf_interSieve_Eq,
_commute_sieveTransf, _commute_sieveTransf;
              rewrite -> _wholeProp_interSieve, transf_interSieve_Eq,
_commute_sieveTransf;
              rewrite -> Heq_inner_wu2; reflexivity) ]).
          + (* _natural_transf *) abstract(move; intros H H' h u;  cbn_sieve;
rewrite -> _natural_transf; reflexivity). }
    * (* _compat_type_Sheafified *) { abstract(intros I wu1 wu2 Heq_wu;
cbn_transf; unshelve apply: ee_compat;
      [ abstract(apply: UU_base; move: Heq_wu; unfold equiv, _rel_relType,
compatRelType; cbn_sieve; intros Heq_wu;
      (* HERE *) simpl (_ o>functor_[functor_ViewOb _] (@identGene _)); do 2
rewrite -> identGene_composGene;
      do 2 rewrite <- _natural_transf; do 2 rewrite -> _wholeProp_interSieve;
      do 2 rewrite -> _functorialCompos_functor'; rewrite -> Heq_wu;
reflexivity)
      | unshelve eexists; cbn -[equiv _type_relType _rel_relType
_equiv_relType _objects_functor _arrows_functor functor_ViewOb
              transf_ViewObMor _functor_sieveFunctor _transf_sieveFunctor
transf_Gluing_lemma];
        [ abstract (reflexivity)
        | cbn_transf; etransitivity; first exact:
pullSieve_pullSieve_sieveEquiv;
          etransitivity; last (symmetry; exact:
pullSieve_pullSieve_sieveEquiv);
          refine (pullSieve_congr_sieveEquiv (reflexivity _) _); exact:
Heq_wu
        | abstract (cbn_transf;
          intros H c; cbn_transf;
          apply: _congr_type_Restrict; cbn_transf; reflexivity) ] ]). }
```

```
    + (* _congr_relTransf  *) abstract (intros f1_ f2_ [indexerEquiv_
sieveEquiv_ dataProp_]; cbn_transf;
      pose l_  := fun (H : vertexGene)
         (u1 : 'Sieve( H ~> _ | interSieve UU (_indexer_type_Restrict f1_)
(_sieve_type_Restrict f1_) )) =>
         (transf_Gluing_lemma _ u1 :>transf_ ee_ H (_whole_interSieve u1));
      pose r_  := fun (H : vertexGene)
         (u1 : 'Sieve( H ~> _ | interSieve UU (_indexer_type_Restrict f1_)
(_sieve_type_Restrict f1_) )) =>
        (transf_Gluing_lemma _ (u1 :>transf_ interSieve_congr
(sieveTransf_Ident UU) indexerEquiv_ sieveEquiv_)
        :>transf_ ee_ H (_whole_interSieve (u1 :>transf_ interSieve_congr
(sieveTransf_Ident UU) indexerEquiv_ sieveEquiv_)));
      have ee_congr' : forall H u1,
        l_ H u1 == r_ H u1;
      first abstract (intros; unshelve apply: ee_congr; intros;
        [ reflexivity
        | (* HERE LEMMA for transf_Gluing_lemma *) unshelve eexists;
cbn_transf;
          [ reflexivity
          | refine (pullSieve_congr_sieveEquiv sieveEquiv_ _);
            cbn_sieve; rewrite -> _commute_sieveTransf; reflexivity
          | intros K c; rewrite -> dataProp_; apply: _congr_relTransf; split;
cbn_transf;
            [ reflexivity
            | split; cbn_transf; first reflexivity; rewrite ->
_commute_sieveTransf; reflexivity ] ] ]);
      unshelve eexists;
      first exact: (sumSieve (fun H u => conflSieve_Sheafified (ee_congr' H
u)));
      first (cbn_transf;
        refine (sumSieve_congr (uu := sieveTransf_Ident _)
        (VV1_ := (fun H u => conflSieve_Sheafified (ee_congr' H u)))
        (VV2_ := (fun H u => _sieve_type_Sheafified (l_ H u)))
        (fun H u => conflTransf1_Sheafified (ee_congr' H u)) ));
      first (cbn_transf;
        refine (sieveTransf_Compos
          (sumSieve_congr (uu := sieveTransf_Ident _)
          (VV1_ := (fun H u => conflSieve_Sheafified (ee_congr' H u)))
          (VV2_ := (fun H u => _sieve_type_Sheafified (r_ H u)))
          (fun H u => conflTransf2_Sheafified (ee_congr' H u)) ) _ );
        refine (@sumSieve_congr _ _ _
          (interSieve_congr (sieveTransf_Ident _) indexerEquiv_ sieveEquiv_ )
          _ _ ( fun H u1 => sieveTransf_Ident _ ) ));
      abstract(intros J c; cbn_transf; exact: conflEquiv_Sheafified)).

- (* _natural_transf *) abstract(intros G; intros G' g f_; cbn_; cbn_transf;
pose l_  := fun (H : vertexGene) (u : 'Sieve( H ~> _ |
    interSieve UU (_indexer_type_Restrict (g o>functor_ f_))
(_sieve_type_Restrict (g o>functor_ f_)) )) =>
```

```
transf_Gluing_lemma _ u :>transf_ ee_ H (_whole_interSieve u);
pose r_  := fun (H : vertexGene) (u : 'Sieve( H ~> _ |
    interSieve UU (_indexer_type_Restrict (g o>functor_ f_))
(_sieve_type_Restrict (g o>functor_ f_)) )) =>
transf_Gluing_lemma _ (u :>transf_
    interSieve_compos UU (_indexer_type_Restrict f_) (_sieve_type_Restrict
f_) g (identSieve _) )
:>transf_ ee_ H (_whole_interSieve (u :>transf_
  interSieve_compos UU (_indexer_type_Restrict f_) (_sieve_type_Restrict f_)
g (identSieve _) ));

have Heq_inner:  forall (H : vertexGene) (u : 'Sieve( H ~> _ |
    interSieve UU (_indexer_type_Restrict (g o>functor_ f_))
(_sieve_type_Restrict (g o>functor_ f_)) )),
l_ H u == r_ H u;
first  (intros; subst l_ r_; cbn_transf; apply: _congr_relTransf;
(* HERE LEMMA for transf_Gluing_lemma *) unshelve eexists; first (cbn_transf;
reflexivity);
  [ cbn_transf; cbn_sieve;
    etransitivity; first exact: (pullSieve_pullSieve_sieveEquiv (reflexivity
_) _ _);
    refine (pullSieve_congr_sieveEquiv (reflexivity _) _);
      abstract (rewrite -> _wholeProp_interSieve; reflexivity)
  | intros K c; cbn_transf; cbn_sieve; apply: _congr_relTransf;
   split; cbn_sieve; reflexivity]);

unshelve eexists;
first exact: (sumSieve (fun H u => conflSieve_Sheafified (Heq_inner H u)));
only 2: (cbn -[_indexer_type_Restrict functor_Restrict  ];
  refine (sumSieve_congr (uu := sieveTransf_Ident _)
  (VV1_ := (fun H u => conflSieve_Sheafified (Heq_inner H u)))
  (VV2_ := (fun H u => _sieve_type_Sheafified (l_ H u)))
  (fun H u => conflTransf1_Sheafified (Heq_inner H u)) ));
first (cbn -[_indexer_type_Restrict functor_Restrict  ];
refine (sieveTransf_Compos (sumSieve_congr (uu := sieveTransf_Ident _)
                      (VV1_ := (fun H u => conflSieve_Sheafified (Heq_inner H
u)))
                      (VV2_ := (fun H u => _sieve_type_Sheafified (r_ H u)))
                      (fun H u => conflTransf2_Sheafified (Heq_inner H u)) )
_);
simpl (_indexer_type_Restrict _);
exact (sumSieve_interSieve'  _ _  ));
last intros J c; cbn_sieve; subst l_ r_; rewrite -> conflEquiv_Sheafified;
reflexivity).
Defined.

Definition transf_RestrictCast (F E : functor)
 (ff : transf F E) (U : vertexGene) (UU  : sieveFunctor U)
 (UU_base: typeOf_baseSieve UU)
 (V : vertexGene) (vu : 'Sieve(V ~> U | UU)) ( VV : sieveFunctor V)
```

```
 (VV_base: typeOf_baseSieve VV) :
 transf (functor_Restrict F VV) (functor_Restrict E UU).
Proof.    intros. refine (transf_Compos (transf_RestrictMor ff
(sieveTransf_Ident _))
  (functor_Restrict_interSieve _ _ (vu :>sieve_) _)).
(* intros. refine (transf_Compos (transf_RestrictMor ff
(interSieve_projFactor _  (vu :>sieve_) _ ))
(functor_Restrict_interSieve _ _ (vu :>sieve_) _)). *)
Defined.

Definition transf_SheafifiedMor (F E : functor) (ee : transf F E) :
  transf (functor_Sheafified F) (functor_Sheafified E).
Proof. unshelve eexists.
- (* _arrows_transf *) intros H. unshelve eexists.
  + (* _fun_relTransf *) intros f_. unshelve eexists.
    * (* _sieve_type_Sheafified *) exact: (_sieve_type_Sheafified f_).
    * (* _data_type_Sheafified *) exact: (transf_Compos
(_data_type_Sheafified f_) ee).
    * (* _compat_type_Sheafified *) abstract (intros K u1 u2  Heq_u;
cbn_transf; apply: _congr_relTransf;
      apply: _compat_type_Sheafified; cbn_transf; rewrite -> Heq_u;
reflexivity).
  + (* _congr_relTransf  *) abstract (move; intros f1_ f2_
  [conflSieve_ conflTransf1_ conflTransf2_ conflEquiv_];
  unshelve eexists; cbn_transf;
  first exact: conflSieve_; first exact: conflTransf1_;  first exact:
conflTransf2_;
  last intros K c;  cbn_transf; apply: _congr_relTransf;
  rewrite -> conflEquiv_; apply: _compat_type_Sheafified; cbn_transf;
reflexivity).
- (* _natural_transf *) abstract (move; intros; unshelve eexists; cbn_transf;
first shelve; first exact: (sieveTransf_Ident _);  first exact:
(sieveTransf_Ident _);
cbn_sieve; intros H c; apply: _congr_relTransf;
apply: _compat_type_Sheafified; cbn_transf; reflexivity).
Defined.

Section Destructing_typeOf.
Variables (U : vertexGene) (UU : sieveFunctor U).
Variables (UU_base: typeOf_baseSieve UU).
Variables  (F E : functor)
  (ee_ : forall (H : vertexGene) (u : 'Sieve(H ~> _ | UU)),
      F H -> transf  (functor_ViewOb H)  E).

Definition typeOf_destructCongr :=
  forall H, Proper ((@equiv _ _  (@_equiv_relType _)) ==> equiv ==>
   (@equiv _ _  (@_equiv_relType (@rel_transf _ _)))  ) (@ee_ H).

Definition typeOf_destructNatural  :=
```

```
forall (G : vertexGene) (u : 'Sieve(G ~> _ | UU)) (form : F G) (H :
vertexGene)
  (f : (functor_ViewOb G) H)
  (G' : vertexGene) (g : 'Gene( G' ~> G ))
  u' form' f',
  (g o>functor_ u) == u' ->
  (g o>functor_ form) == form' ->
  f == f' :>transf_ (transf_ViewObMor g)  ->
f :>transf_ ee_  u form == f' :>transf_ ee_  u' form'.

End Destructing_typeOf.

Definition transf_Destructing_preCast :
forall (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU )
(F E : functor)
(ee_ : forall (H : vertexGene) (u : 'Sieve(H ~> _ | UU)),
      F H -> transf (functor_ViewOb H) E)
(ee_congr : typeOf_destructCongr ee_)
(ee_natural : typeOf_destructNatural ee_),
transf (functor_Restrict F UU) (functor_Restrict E UU).
Proof.  unshelve eexists.
- (* _arrows_transf *) intros G. unshelve eexists.
  (* _fun_relTransf *) intros f_. { unshelve eexists.
    - (* _indexer_type_Restrict *) exact: (_indexer_type_Restrict f_).
    - (* _sieve_type_Restrict *) exact:  (_sieve_type_Restrict f_)   .
    - { (* _data_type_Restrict *) unshelve eexists.
        + (* _arrows_transf *)  intros H. unshelve eexists.
          * (* _fun_relTransf *) intros u.  exact: (identGene
        :>transf_ ee_ H (u :>transf_ interSieve_projWhole _ _ _) (u :>transf_
f_)).
          * (* _congr_relTransf *) abstract (move; intros u1 u2 Heq;
cbn_transf; cbn_functor;
          rewrite -> ee_congr; first reflexivity;
           first (rewrite -> (whole_interSieve_Proper Heq); reflexivity);
           first (rewrite -> Heq; reflexivity);
           last reflexivity).
          (* abstract(move; intros u1 u2 Heq; cbn_transf; cbn_functor;
          apply: ee_congr; rewrite -> Heq; reflexivity). *)
        + (* _natural_transf *) abstract(move; intros H H' h u; cbn_transf;
        rewrite -> _natural_transf; setoid_rewrite <- ee_natural at 2; first
reflexivity;
        first (cbn_sieve; reflexivity);
        first (rewrite <- _natural_transf; reflexivity); etransitivity;
        first (exact:identGene_composGene ); symmetry; exact:
composGene_identGene). }
    - (* _congr_type_Restrict *) abstract (intros I v v' Heq; cbn_transf;
    have Heq_whole : _whole_interSieve v == _whole_interSieve v';
      first (apply UU_base; move: Heq; unfold  _rel_relType, equiv; simpl;
```

```
        intros Heq; do 2 rewrite -> _wholeProp_interSieve; rewrite -> Heq;
reflexivity);
    apply: ee_congr;
    first (rewrite -> Heq_whole; reflexivity);
    first (apply: _congr_type_Restrict; exact Heq); reflexivity).  }
  (* _congr_relTransf  *) abstract(move; intros f1_ f2_ [indexerEquiv
sieveEquiv_ Heq];
  unshelve eexists; cbn_sieve;
  first (rewrite -> indexerEquiv; reflexivity);
  first exact: sieveEquiv_;
  last intros J c; cbn_sieve; apply: ee_congr; cbn_sieve; first reflexivity;
last reflexivity;
    rewrite -> Heq; apply: _congr_relTransf; split; cbn_sieve; reflexivity).
- (*  _natural_transf  *) abstract(intros H' H h f_; unshelve eexists;
cbn_sieve;
  first reflexivity; first reflexivity;
  first (intros K c; cbn_sieve;
  apply: ee_congr; first reflexivity; last reflexivity;
  apply: _congr_relTransf; split; cbn_sieve;  reflexivity)).
Defined.

Definition transf_UnitSheafified_prePoly_preCast :
forall (F : functor),
transf  F  (functor_Sheafified F).
Proof. unshelve eexists.
- (* _arrows_transf *)  intros G. unshelve eexists.
  + (* _fun_relTransf *) intros f_. unshelve eexists.
    * (* _sieve_type_Sheafified *) exact: (identSieve _).
    * { - (* _data_type_Sheafified *) unshelve eexists.
          + (* _arrows_transf *)  intros H. unshelve eexists.
            * (* _fun_relTransf *) intros u. exact: (u o>functor_ f_).
            * (* _congr_relTransf *)  abstract(move; intros u1 u2 Heq; cbn -
[functor_Restrict];
            tac_unsimpl; rewrite -> Heq; reflexivity).
          + (* _natural_transf *) abstract(move; intros H H' h u;  cbn -
[functor_Restrict];
          tac_unsimpl; rewrite -> _functorialCompos_functor'; reflexivity).
}
      * (* _compat_type_Sheafified *) abstract(intros I v v'; simpl; intros
Heqs; rewrite -> Heqs; reflexivity).
  + (* _congr_relTransf  *) abstract(move; intros f1_ f2_ Heq; unshelve
eexists; cycle 1;
  first exact (sieveTransf_Ident _); first exact (sieveTransf_Ident _);
  intros K c; cbn -[functor_Restrict]; tac_unsimpl; rewrite -> Heq;
reflexivity).
- (* _natural_transf *) abstract(move; intros G G' g f_; cbn_transf;
cbn_functor;
unshelve eexists; cycle 1; first exact (sieveTransf_Ident _); first exact
(sieveTransf_identSieve _);
cbn_transf; cbn_functor; intros K c;  rewrite -> _functorialCompos_functor';
```

```
apply: _congr_relFunctor; last reflexivity;
apply: (_wholeProp_interSieve c)).
Defined.

Definition transf_Destructing
 (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU )
(F E : functor)
(ee_ : forall (H : vertexGene) (u : 'Sieve(H ~> _ | UU)),
      F H -> transf (functor_ViewOb H) E)
(ee_congr : typeOf_destructCongr ee_)
(ee_natural : typeOf_destructNatural ee_)
(V : vertexGene) (VV : sieveFunctor V)
(VV_base: typeOf_baseSieve VV)
(uv : 'Sieve(U ~> V | VV) )  :
transf (functor_Restrict F UU) (functor_Sheafified (functor_Restrict E VV)).
Proof.
  refine (transf_Compos (transf_Destructing_preCast UU_base ee_congr
ee_natural) _).
  refine (transf_Compos (transf_RestrictCast  (transf_Ident _) VV_base uv
UU_base) _).
  exact: (transf_UnitSheafified_prePoly_preCast _).
Defined.

Definition transf_Constructing (* AKA UnitRestrict *)
 (U : vertexGene) (UU : sieveFunctor U)
(F  : functor)
(K : vertexGene) (u : 'Sieve(K ~> _ | UU))
(form : F K) :
transf (functor_ViewOb K)  (functor_Restrict F UU).
Proof.  unshelve eexists.
- (* _arrows_transf *) intros G. unshelve eexists.
  (* _fun_relTransf *) intros f_. { unshelve eexists.
    - (* _indexer_type_Restrict *) exact: ((f_ o>functor_ u) :>sieve_).
    - (* _sieve_type_Restrict *) exact:  (identSieve _).
    - { (* _data_type_Restrict *) unshelve eexists.
        + (* _arrows_transf *)  intros H. unshelve eexists.
          * (* _fun_relTransf *) intros u'. refine (( (_factor_interSieve u')
o>functor_ f_ ) o>functor_ form).
          * (* _congr_relTransf *)
          abstract (move; intros u1 u2 Heq; cbn_sieve; rewrite ->
(factor_interSieve_Proper Heq); reflexivity).
        + (* _natural_transf *) abstract(move; intros H H' h u'; cbn_transf;
        do 2 rewrite -> _functorialCompos_functor';  reflexivity). }
    - (* _congr_type_Restrict *) abstract (intros I v v' Heq; cbn_transf;
    rewrite -> Heq; reflexivity).  }
  (* _congr_relTransf  *) abstract (move; intros f1_ f2_ Heq;
    unshelve eexists; cbn_sieve;
    first (rewrite -> Heq; reflexivity);
    first reflexivity;
```

```
    last intros J c; cbn_sieve; rewrite -> Heq; reflexivity).
- (*  _natural_transf  *) abstract(intros H' H h f_; unshelve eexists;
cbn_sieve;
first (rewrite <- _functorialCompos_functor'; setoid_rewrite <-
_natural_transf at 2; reflexivity);
first exact: pullSieve_identSieve_sieveEquiv;
last intros K0 c; cbn_transf;
apply: _congr_relFunctor; last reflexivity;
 rewrite -> _functorialCompos_functor';
apply: _congr_relFunctor; last reflexivity;
 apply: (_wholeProp_interSieve (_factor_interSieve c))).
Defined.

Definition transf_UnitSheafified
 (U : vertexGene) (UU : sieveFunctor U)
 (UU_base: typeOf_baseSieve UU)
(F  : functor)
(K : vertexGene) (u : 'Sieve(K ~> _ | UU))
(ff: transf  (functor_ViewOb K)  F)
(V : vertexGene) (VV : sieveFunctor V)
(VV_base: typeOf_baseSieve VV)
(uv : 'Sieve(U ~> V | VV) )  :
transf  (functor_ViewOb K)  (functor_Sheafified (functor_Restrict F VV)).
Proof.
    refine (transf_Compos (transf_Constructing u ( identGene :>transf_ ff))
_).
    refine (transf_Compos (transf_RestrictCast  (transf_Ident _) VV_base uv
UU_base) _).
    refine  (transf_UnitSheafified_prePoly_preCast _) .
Defined.

Lemma Constructing_destructNatural
 (U : vertexGene) (UU : sieveFunctor U)
(F  : functor):
typeOf_destructNatural (@transf_Constructing U UU F ).
Proof.  intros; move. intros G u form H f G' g u' form' f' Heq_u Heq_form
Heq_f .
unshelve eexists; cbn_sieve.
- rewrite ->  Heq_f, <- Heq_u, -> _functorialCompos_functor'. reflexivity.
- reflexivity.
- intros K c. rewrite <- Heq_form. rewrite -> _functorialCompos_functor'.
apply: _congr_relFunctor; last reflexivity. rewrite ->  Heq_f. cbn_transf.
rewrite <- _functorialCompos_functor'. reflexivity.
Qed.

Time Inductive elemCode : forall (G: vertexGene) (F : functor)  (ff : transf
(functor_ViewOb G) F), Type :=

| Compos_elemCode : forall (F : functor) ( F''  : vertexGene) (F' : functor)
(ff_ : transf (functor_ViewOb F'') F') (ff' : transf F' F),
```

```
elemCode ff_ -> morCode ff' -> elemCode ( transf_Compos ff_ ff' )

| Constructing_elemCode :
forall (U : vertexGene) (UU : sieveFunctor U)
(F  : functor)
(K : vertexGene) (u : 'Sieve(K ~> _ | UU))
(form : F K),

elemCode (transf_Constructing  u form)

| UnitSheafified_elemCode :
forall  (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
(F  : functor)
(K : vertexGene) (u : 'Sieve(K ~> _ | UU))
(ff: transf  (functor_ViewOb K)  F)
(V : vertexGene) (VV : sieveFunctor V)
(VV_base: typeOf_baseSieve VV)
(uv : 'Sieve(U ~> V | VV) )
(Code_ff : elemCode ff),

elemCode ( transf_UnitSheafified UU_base u ff VV_base uv )

with morCode : forall (E: functor) (F : functor)  (ff : transf E F), Type :=

| Compos_morCode :
forall (F F'' F' : functor) (ff_ : transf F'' F') (ff' : transf F' F),

morCode ff_ -> morCode ff' -> morCode ( transf_Compos ff_ ff' )

| Ident_morCode :
forall (F : functor),

@morCode F F ( transf_Ident F )

| SheafifiedMor_morCode :
forall (F E : functor) (ee : transf F E)
(Code_ee : morCode ee),

morCode (transf_SheafifiedMor ee )

| RestrictCast_morCode :
forall (F : functor)  (U : vertexGene) (UU  : sieveFunctor U)
 (UU_base: typeOf_baseSieve UU)
 (V : vertexGene) (vu : 'Sieve(V ~> U | UU)) ( VV : sieveFunctor V)
  (VV_base: typeOf_baseSieve VV),

morCode (transf_RestrictCast (transf_Ident F) UU_base vu VV_base)
```

```
| Destructing_morCode :
forall (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU )
(F E : functor)
(ee_ : forall (H : vertexGene) (u : 'Sieve(H ~> _ | UU)),
      F H -> transf (functor_ViewOb H) E)
(ee_congr : typeOf_destructCongr ee_)
(ee_natural : typeOf_destructNatural ee_)
(V : vertexGene) (VV : sieveFunctor V)
(VV_base: typeOf_baseSieve VV)
(uv : 'Sieve(U ~> V | VV) ) ,
forall (Code_ee__ : forall (H : vertexGene) (u : 'Sieve(H ~> _ | UU))
    (form: F H) , elemCode (ee_ H u form) ),

morCode (transf_Destructing UU_base ee_congr ee_natural VV_base uv)

| Gluing_morCode :
forall (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
(VV_ : forall H : vertexGene, 'Sieve( H ~> _ | UU ) -> sieveFunctor H)
(VV_congr : typeOf_sieveCongr VV_)
(VV_natural : typeOf_sieveNatural VV_)
  (F E : functor)
  (ee_ : forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU )),
         transf (functor_Restrict F (VV_ H u)) (functor_Sheafified E))
  (ee_congr : typeOf_gluingCongr VV_congr ee_)
(ee_natural : typeOf_gluingNatural VV_natural ee_)
(ee_compat : typeOf_gluingCompat VV_congr VV_natural ee_),
forall (Code_ee : forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU )),
        morCode (ee_ H u)),

morCode (transf_Gluing UU_base ee_congr ee_natural ee_compat).

 (* /!\ LONG TIME /!\
Finished transaction in 534.461 secs (533.984u,0.031s) (successful)
33 sec without Gluing_morCode
 /!\  NOPE  /!\  after delete all polymorphism config leaving only Set
Universe Polymorphism.:
Finished transaction in 0.183 secs (0.171u,0.s) (successful)
Finished transaction in 0.214 secs (0.218u,0.s) (successful) *)

Inductive obCoMod : forall (F : functor), Type :=

| Restrict : forall (F : functor) (U : vertexGene) (UU : sieveFunctor U),
obCoMod (functor_Restrict F UU)

| SheafifiedOb : forall (F : functor),
obCoMod (functor_Sheafified F)

| ViewOb : forall (G : vertexGene),
```

```
obCoMod (functor_ViewOb G).

Notation "u ==1 v" := (@relation_transf _ _ u v)
(at level 70, no associativity) : type_scope.
Tactic Notation "cbn_rel_transf"  :=
cbn_equiv;  unfold  rel_transf, relation_transf.
Tactic Notation "cbn_rel_transf" "in" hyp_list(H) :=
cbn_equiv in H;  unfold  rel_transf, relation_transf in H.

Lemma Congr_Compos_cong :
forall (F F'' F' : functor) (ff_ : transf F'' F') (ff' : transf F' F),
forall (dd_ : transf F'' F') (dd' : transf F' F)
(Congr_congr_ff_ :  ff_ ==1 dd_)
(Congr_congr_ff' :  ff' ==1 dd'),
 (transf_Compos ff_ ff') ==1 (transf_Compos dd_ dd').
Proof. intros. cbn_rel_transf in Congr_congr_ff_ Congr_congr_ff'.
cbn_rel_transf. intros.
apply: ( Congr_congr_ff').  apply: ( Congr_congr_ff_). assumption.
Qed.

(* TODO: keep or erase *)
Instance Congr_Compos_cong' :
forall (F F'' F' : functor),
Proper ( @equiv  _ (@_rel_relType (rel_transf _ _)) (@_equiv_relType
(rel_transf _ _))
 ==> @equiv  _ (@_rel_relType (rel_transf _ _)) (@_equiv_relType (rel_transf
_ _))
 ==> @equiv  _ (@_rel_relType (rel_transf _ _)) (@_equiv_relType (rel_transf
_ _)) )
 (@transf_Compos F F'' F').
Proof. intros. move.  intros ff_  dd_ Congr_congr_ff_  ff' dd'
Congr_congr_ff'.
 cbn_rel_transf in Congr_congr_ff_ Congr_congr_ff'. cbn_rel_transf. intros.
apply: ( Congr_congr_ff').  apply: ( Congr_congr_ff_). assumption.
Qed.

Lemma Congr_Constructing_cong:
forall (U : vertexGene) (UU : sieveFunctor U)
(F  : functor)
(K : vertexGene) (u : 'Sieve(K ~> _ | UU))
(form : F K),
forall (u' : 'Sieve(K ~> _ | UU))
(form' : F K),
forall (Heq_u : u == u')
(Heq_form : form == form'),
 (transf_Constructing  u form) ==1
(transf_Constructing  u' form').
Proof.  intros. cbn_rel_transf. intros G k k' Heq_k . rewrite -> Heq_k.
unshelve eexists; cbn_transf;
first (rewrite -> Heq_u; reflexivity);
```

```
first reflexivity;
last intros H c; cbn_sieve; rewrite -> Heq_form; reflexivity.
Qed.

Definition Congr_UnitSheafified_cong :
forall (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
(F  : functor)
(K : vertexGene) (u : 'Sieve(K ~> _ | UU))
(ff: transf  (functor_ViewOb K)  F)
(V : vertexGene) (VV : sieveFunctor V)
(VV_base: typeOf_baseSieve VV)
(uv : 'Sieve(U ~> V | VV) )
(U' : vertexGene) (UU' : sieveFunctor U')
(UU_base': typeOf_baseSieve UU')

(u' : 'Sieve(K ~> _ | UU'))
(ff': transf  (functor_ViewOb K)  F)
(VV_base': typeOf_baseSieve VV)
(uv' : 'Sieve(U' ~> V | VV) )
(KK : sieveFunctor K)
 (Congr_ff_Sieve: forall (K' : vertexGene) (k : 'Sieve( K' ~> _ | KK )) ,
( (k :>sieve_)) :>transf_ ff == ( (k :>sieve_)) :>transf_ ff')
(Congr_UU_u : sieveEquiv (pullSieve UU (u :>sieve_))
(pullSieve UU' (u' :>sieve_)))
(* (Congr_u : (u :>sieve_) == (u' :>sieve_)) *)
(* MEMO DO NOT USE Congr_Restrict_cast_cong *)
(Congr_u_uv : (u :>sieve_) o>sieve_  uv   == (u' :>sieve_) o>sieve_  uv'  ),
(transf_UnitSheafified UU_base u ff VV_base uv) ==1 (transf_UnitSheafified
UU_base' u' ff' VV_base' uv').
Proof. intros. intros H f f' Heq_f.
unfold transf_UnitSheafified.
cbn -[equiv _type_relType _rel_relType _equiv_relType  _arrows_functor
functor_ViewOb
      transf_ViewObMor _functor_sieveFunctor _transf_sieveFunctor
      transf_Constructing transf_RestrictCast
transf_UnitSheafified_prePoly_preCast].
rewrite -> Heq_f; clear Heq_f.
unshelve eexists; cbn_transf.
exact:  (pullSieve KK f'). exact: sieveTransf_identSieve. exact:
sieveTransf_identSieve.
intros J c; cbn_transf.
unshelve eexists;  cbn_sieve;
first (do 2 rewrite <- _natural_transf, <- _functorialCompos_functor';
setoid_rewrite -> _natural_transf;
rewrite ->  Congr_u_uv; reflexivity).
apply: (pullSieve_congr_sieveEquiv  _ (reflexivity _)).
etransitivity;
 first (apply: (interSieve_congr_sieveEquiv (reflexivity _) _ (reflexivity
_));
```

```
  do 1 rewrite  <- _natural_transf; reflexivity).
etransitivity;
 last (apply: (interSieve_congr_sieveEquiv (reflexivity _) _ (reflexivity
_));
  do 1 rewrite  <- _natural_transf; reflexivity).
etransitivity;
  last apply: pullSieve_pullSieve_sieveEquiv.
etransitivity;
  first (symmetry; apply: pullSieve_pullSieve_sieveEquiv).
apply: (interSieve_congr_sieveEquiv Congr_UU_u  (reflexivity _) (reflexivity
_)).

intros H0 c0. cbn_transf. do 2 rewrite -> _natural_transf. cbn_equiv in c0.
set ll := (X in X :>transf_ _ == X :>transf_ _  ).
have Heq : ll == ( _whole_interSieve (((_factor_interSieve c0) :>sieve_)
o>sieve_ c) ) :>sieve_ ;
last (rewrite -> Heq; apply: Congr_ff_Sieve).
subst ll. etransitivity; first apply: identGene_composGene.
rewrite -> _wholeProp_interSieve. setoid_rewrite <- _natural_transf.
rewrite <- (_wholeProp_interSieve (_factor_interSieve c0 )). reflexivity.
Qed.

Lemma Congr_SheafifiedMor_cong :
forall (F E : functor) (ee : transf F E),
forall (ee' : transf F E)
(Congr_ee :  ee ==1 ee'),
 (transf_SheafifiedMor ee ) ==1 (transf_SheafifiedMor ee' ).
Proof. intros. intros G f f' Heq_f . rewrite -> Heq_f. unshelve eexists;
first shelve;
first exact (sieveTransf_Ident _);
first exact (sieveTransf_Ident _).
abstract (intros H c; cbn_sieve; apply: Congr_ee; reflexivity).
Qed.

Definition Congr_Destructing_cong :
forall (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU )
(F E : functor)
(ee_ : forall (H : vertexGene) (u : 'Sieve(H ~> _ | UU)),
      F H -> transf (functor_ViewOb H) E)
(ee_congr : typeOf_destructCongr ee_)
(ee_natural : typeOf_destructNatural ee_)
(V : vertexGene) (VV : sieveFunctor V)
(VV_base: typeOf_baseSieve VV)
(uv : 'Sieve(U ~> V | VV) ),
forall
(UU_base': typeOf_baseSieve UU)
(dd_ : forall (H : vertexGene) (u : 'Sieve(H ~> _ | UU)),
      F H -> transf  (functor_ViewOb H)  E)
(dd_congr : typeOf_destructCongr dd_)
```

```
(dd_natural : typeOf_destructNatural dd_)
(VV_base': typeOf_baseSieve VV)
 (uv' : 'Sieve(U ~> V | VV) ),

forall (Congr_ee_: forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU )),
  forall (f : F H ), identGene :>transf_ ee_ H u f   == identGene :>transf_
dd_ H u f ) ,
forall  (Congr_uv : uv == uv'),

 (transf_Destructing UU_base ee_congr ee_natural VV_base uv)
==1  (transf_Destructing UU_base' dd_congr dd_natural VV_base' uv').
Proof. intros. intros H f_ f_' Heq_f_.
rewrite -> Heq_f_; clear Heq_f_.
unshelve eexists; cbn_transf.
- exact (identSieve _).
- exact: (sieveTransf_Ident _).
- exact: (sieveTransf_Ident _).
- intros J c.  cbn_transf;  unshelve eexists.
  + abstract (cbn_sieve; rewrite -> Congr_uv; reflexivity).
  + cbn_sieve; reflexivity.
  +  cbn_sieve. intros H0 c0. etransitivity; first apply: Congr_ee_. apply
dd_congr.
     * abstract (reflexivity).
     * apply: _congr_relTransf. unshelve eexists; cbn_transf; reflexivity.
     * reflexivity.
Qed.

Definition Congr_Gluing_cong :
forall (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
(VV_ : forall H : vertexGene, 'Sieve( H ~> _ | UU ) -> sieveFunctor H)
(VV_congr : typeOf_sieveCongr VV_)
(VV_natural : typeOf_sieveNatural VV_)
  (F E : functor)
  (ee_ : forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU )),
         transf (functor_Restrict F (VV_ H u)) (functor_Sheafified E))
  (ee_congr : typeOf_gluingCongr VV_congr ee_)
(* ee_natural used in code only not sense *)
(ee_natural : typeOf_gluingNatural VV_natural ee_)
(ee_compat : typeOf_gluingCompat VV_congr VV_natural ee_),

forall (UU_base': typeOf_baseSieve UU)
(VV_congr' : typeOf_sieveCongr VV_)
(VV_natural' :  typeOf_sieveNatural VV_)
(dd_ : forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU )),
      transf (functor_Restrict F (VV_ H u)) (functor_Sheafified E))
(dd_congr : typeOf_gluingCongr VV_congr' dd_)
(dd_natural : typeOf_gluingNatural VV_natural' dd_)
(dd_compat: typeOf_gluingCompat VV_congr' VV_natural' dd_),
```

```
forall (Congr_ee_: forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU )),
  ee_ H u ==1 dd_ H u) ,

(transf_Gluing UU_base ee_congr ee_natural ee_compat)
==1 (transf_Gluing UU_base' dd_congr dd_natural dd_compat) .
Proof. intros. intros H f_ f_' Heq_f_.
rewrite -> Heq_f_; clear Heq_f_.

have @Congr_ee_': (forall (H0 : vertexGene)
        (u : 'Sieve( H0 ~> _ | interSieve UU (_indexer_type_Restrict f_')
                                (_sieve_type_Restrict f_') )),
        (transf_Gluing_lemma f_' u :>transf_ ee_ H0 (_whole_interSieve u))
        == (transf_Gluing_lemma f_' u :>transf_ dd_ H0 (_whole_interSieve u))
);
first (intros; apply: Congr_ee_;reflexivity).

unshelve eexists; cbn_transf.
- exact: (sumSieve (fun H0 u => (conflSieve_Sheafified (Congr_ee_' H0 u)) )).
- exact: (sumSieve_congr (uu := sieveTransf_Ident _ )
    (fun H0 u => (conflTransf1_Sheafified (Congr_ee_' H0 u)) )).
- exact:  (sumSieve_congr (uu := sieveTransf_Ident _ )
    (fun H0 u => (conflTransf2_Sheafified (Congr_ee_' H0 u)) )).
- abstract (cbn_transf; intros J c;
  apply: (@conflEquiv_Sheafified _ _ _ _ (Congr_ee_' _ _))).
Qed.

Definition Congr_Restrict_cong (F E : functor)
 (ff : transf F E) (U : vertexGene) (UU VV : sieveFunctor U)
 (uu : sieveTransf VV UU) (ff' : transf F E) (uu' : sieveTransf VV UU)
 (Congr_ff : ff ==1 ff')
 (* TODO MEMO (Congr_uu : uu ==1 uu')
 NOT LACKED BECAUSE OF _congr_type_Restrict  *) :
 (transf_RestrictMor ff uu) ==1 (transf_RestrictMor ff' uu').
Proof. cbn_rel_transf in Congr_ff.  intros G f_ f_' [indexerEquiv_
sieveEquiv_ Heq_f_ ].
 unshelve eexists; cbn_transf;
first (exact: indexerEquiv_);
first (exact: sieveEquiv_);
last intros H c; cbn_sieve. apply: Congr_ff.
rewrite -> Heq_f_;
(* TODO: HERE  *) apply: _congr_type_Restrict; cbn_transf; reflexivity.
(*  apply: _congr_relTransf;
unshelve eexists; cbn_transf; first reflexivity; last apply: Congr_uu;
reflexivity.
 *)
Qed.

Definition Congr_Restrict_cast_cong (F E : functor)
(ff : transf F E) (U : vertexGene) (UU  : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
```

```
(V : vertexGene) (vu : 'Sieve(V ~> U | UU)) ( VV : sieveFunctor V)
(VV_base: typeOf_baseSieve VV)
(ff' : transf F E)  (UU_base': typeOf_baseSieve UU) (vu' : 'Sieve(V ~> U |
UU))
 (VV_base': typeOf_baseSieve VV)
 (Congr_ff : ff ==1 ff') (Congr_vu : vu == vu')   :
 (transf_RestrictCast ff UU_base vu VV_base) ==1 (transf_RestrictCast ff'
UU_base' vu' VV_base').
Proof. cbn_rel_transf in Congr_ff.  intros G f_ f_' [indexerEquiv_
sieveEquiv_ Heq_f_ ].
 unshelve eexists; cbn_transf;
first (rewrite -> Congr_vu, -> indexerEquiv_; reflexivity);
first refine (interSieve_congr_sieveEquiv (reflexivity _) indexerEquiv_
sieveEquiv_);
last intros H c; cbn_sieve. apply: Congr_ff.
rewrite -> Heq_f_. (* TODO: HERE POSSIBLE _congr_type_Restrict  *)
 apply: _congr_relTransf. unshelve eexists; cbn_transf;  reflexivity.
Qed.

Definition Congr_Compos_Ident (F E : functor)
(ff : transf F E)  :
(transf_Compos ff  (transf_Ident E))
 ==1 ff.
Proof.  intros G f f' Heq_f. cbn_transf. rewrite -> Heq_f; reflexivity.
Qed.

Definition Congr_Restrict_comp_Restrict (F E : functor)
 (ff : transf F E) (U : vertexGene) (UU VV : sieveFunctor U)
 (uu : sieveTransf VV UU)
 (D : functor)
 (ff' : transf E D) (WW : sieveFunctor U) (vv : sieveTransf WW VV) :
 (transf_Compos (transf_RestrictMor ff uu) (transf_RestrictMor ff' vv))
  ==1 (transf_RestrictMor (transf_Compos ff ff') (sieveTransf_Compos vv uu)).
Proof.  intros G f_ f_' [indexerEquiv_ sieveEquiv_ Heq_f_ ].
 unshelve eexists; cbn_transf;
first (exact: indexerEquiv_);
first (exact: sieveEquiv_);
last intros H c; cbn_sieve.
rewrite -> Heq_f_. do 3 apply: _congr_relTransf.
unshelve eexists; cbn_transf; reflexivity.
Qed.

Definition Congr_Restrict_cast_comp_Restrict_cast (F E : functor)
(ff : transf F E) (U : vertexGene) (UU  : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
(V : vertexGene) (vu : 'Sieve(V ~> U | UU)) ( VV : sieveFunctor V)
(VV_base: typeOf_baseSieve VV)
(D : functor)   (ff' : transf E D) (W : vertexGene) (WW  : sieveFunctor W)
(WW_base: typeOf_baseSieve WW)
(uw : 'Sieve(U ~> W | WW))
```

```
(UU_base': typeOf_baseSieve UU)  :
  (transf_Compos (transf_RestrictCast ff UU_base vu VV_base)
    (transf_RestrictCast ff' WW_base uw UU_base'))
==1 (transf_RestrictCast (transf_Compos ff ff') WW_base ((vu :>sieve_)
o>sieve_ uw ) VV_base).
Proof.  intros G f_ f_' [indexerEquiv_ sieveEquiv_ Heq_f_ ].
 unshelve eexists; cbn_transf;
 first(rewrite -> indexerEquiv_; rewrite <- _functorialCompos_functor', ->
_natural_transf;
  reflexivity).
 - etransitivity. refine (interSieve_congr_sieveEquiv (reflexivity _) _ _).
  (rewrite  -> _natural_transf; reflexivity).
  refine (interSieve_congr_sieveEquiv (reflexivity _) indexerEquiv_
sieveEquiv_).
    symmetry; apply: interSieve_image_sieveEquiv. exact: UU_base.
 - intros H c; cbn_sieve.
  rewrite -> Heq_f_. do 3 apply: _congr_relTransf.
unshelve eexists; cbn_transf; reflexivity.
Qed.

Definition Congr_SheafifiedMor_comp_SheafifiedMor :
 forall (F E : functor) (ee : transf F E),
 forall (D : functor) (dd : transf E D),
 (transf_Compos (transf_SheafifiedMor ee) (transf_SheafifiedMor dd))
==1 (transf_SheafifiedMor (transf_Compos ee dd )).
Proof.  intros. move. intros H f_ f_' Heq_f_. rewrite -> Heq_f_. unshelve
eexists; cbn_transf.
- exact (_sieve_type_Sheafified f_').
- exact: (sieveTransf_Ident _).
- exact: (sieveTransf_Ident _).
- abstract (intros J c; reflexivity).
Qed.

Section Gluing_comp_SheafifiedMor.
Variables (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
(VV_ : forall H : vertexGene, 'Sieve( H ~> _ | UU ) -> sieveFunctor H)
(VV_congr : typeOf_sieveCongr VV_)
(VV_natural : typeOf_sieveNatural VV_)
  (F E : functor)
  (ee_ : forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU )),
         transf (functor_Restrict F (VV_ u)) (functor_Sheafified E)).
Variables  (ee_congr : typeOf_gluingCongr VV_congr ee_)
(ee_natural : typeOf_gluingNatural VV_natural ee_)
(ee_compat : typeOf_gluingCompat VV_congr VV_natural ee_).

Variables (D : functor) (dd : transf E D).

Lemma Gluing_comp_SheafifiedMor_gluingCongr :
```

```
typeOf_gluingCongr VV_congr (fun H u => (transf_Compos (ee_  u)
(transf_SheafifiedMor dd))) .
Proof.     move. intros.
  cbn -[equiv _type_relType _rel_relType _equiv_relType  _arrows_functor
functor_ViewOb
  transf_ViewObMor _functor_sieveFunctor _transf_sieveFunctor
transf_SheafifiedMor].
  apply: _congr_relTransf. apply: ee_congr; eassumption.
Qed.

Lemma Gluing_comp_SheafifiedMor_gluingNatural :
typeOf_gluingNatural VV_natural (fun H u => (transf_Compos (ee_  u)
(transf_SheafifiedMor dd))) .
Proof.     move. intros.
  cbn -[equiv _type_relType _rel_relType _equiv_relType  _arrows_functor
functor_ViewOb
  transf_ViewObMor _functor_sieveFunctor _transf_sieveFunctor
transf_SheafifiedMor transf_RestrictMor].
  rewrite -> _natural_transf.
  apply: _congr_relTransf. apply: ee_natural; eassumption.
Qed.

Lemma Gluing_comp_SheafifiedMor_gluingCompat :
typeOf_gluingCompat VV_congr VV_natural (fun H u => (transf_Compos (ee_  u)
(transf_SheafifiedMor dd))) .
Proof.     move. intros.
  cbn -[equiv _type_relType _rel_relType _equiv_relType  _arrows_functor
functor_ViewOb
  transf_ViewObMor _functor_sieveFunctor _transf_sieveFunctor  ].
  apply: _congr_relTransf. apply: ee_compat; eassumption.
Qed.

Definition Congr_Gluing_comp_SheafifiedMor:
(transf_Compos (transf_Gluing UU_base ee_congr ee_natural ee_compat)
(transf_SheafifiedMor dd) )
==1 (transf_Gluing UU_base (Gluing_comp_SheafifiedMor_gluingCongr)
             (Gluing_comp_SheafifiedMor_gluingNatural)
(Gluing_comp_SheafifiedMor_gluingCompat) ).
Proof. intros. move. intros H f_ f_' Heq_f_. rewrite -> Heq_f_. unshelve
eexists; cbn_transf.
- shelve.
- exact: (sieveTransf_Ident _).
- exact: (sieveTransf_Ident _).
- abstract (intros J c; reflexivity).
Qed.

End Gluing_comp_SheafifiedMor.

Section Destructing_comp_SheafifiedMor.
Variables (U : vertexGene) (UU : sieveFunctor U)
```

```
(UU_base: typeOf_baseSieve UU)
  (F E : functor)
  (ee_ : forall (H : vertexGene) (u : 'Sieve(H ~> _ | UU)),
  F H -> transf  (functor_ViewOb H)  E).
Variables  (ee_congr : typeOf_destructCongr  ee_)
(ee_natural : typeOf_destructNatural  ee_)
(V : vertexGene) (VV : sieveFunctor V)
(VV_base:  typeOf_baseSieve VV)
 (uv : 'Sieve(U ~> V | VV) ) .
Variables (D : functor) (dd : transf E D)
(W : vertexGene)   (WW : sieveFunctor W)
(WW_base:  typeOf_baseSieve WW) (vw : 'Sieve(V ~> W | WW))
(VV_base':  typeOf_baseSieve VV)  .

Lemma Destructing_comp_SheafifiedMor_destructCongr  :
typeOf_destructCongr  (fun H u f => (transf_Compos (@ee_ H u f)  dd)) .
Proof.     do 4 (move; intros).  cbn_transf.
  apply: _congr_relTransf. apply: ee_congr; eassumption.
Qed.

Lemma Destructing_comp_SheafifiedMor_destructNatural :
typeOf_destructNatural  (fun H u f => (transf_Compos (@ee_ H u f)  dd)) .
Proof.     move. intros. cbn_transf.
  apply: _congr_relTransf. apply: ee_natural; eassumption.
Qed.

Definition Congr_Destructing_comp_SheafifiedMor:
(transf_Compos (transf_Destructing UU_base ee_congr ee_natural VV_base uv)
    (transf_SheafifiedMor (transf_RestrictCast dd WW_base vw VV_base' )) )
==1 (transf_Destructing UU_base
(Destructing_comp_SheafifiedMor_destructCongr)
         (Destructing_comp_SheafifiedMor_destructNatural) WW_base  ((uv
:>sieve_) o>sieve_ vw)   ).
Proof. intros. move. intros H f_ f_' Heq_f_. rewrite -> Heq_f_.
clear Heq_f_.  unshelve eexists.
- shelve.
- cbn_sieve. exact: (sieveTransf_Ident _).
- cbn_sieve. exact: (sieveTransf_Ident _).
- intros J c.  unshelve eexists.
  + abstract(cbn_sieve;  rewrite <- _natural_transf;
   do 3 rewrite -> _functorialCompos_functor; reflexivity).
  + cbn_sieve. cbn_sieve in c.
  etransitivity. refine (interSieve_congr_sieveEquiv (reflexivity _) _
(reflexivity _)).
  (rewrite -> _functorialCompos_functor', -> _natural_transf; reflexivity).
    symmetry; apply: interSieve_image_sieveEquiv. exact: VV_base.
  + abstract(intros H0 c0; cbn_sieve; reflexivity).
Qed.

End Destructing_comp_SheafifiedMor.
```

```
Section RestrictCast_comp_Destructing.
Variables (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
  (F E : functor)
  (ee_ : forall (H : vertexGene) (u : 'Sieve(H ~> _ | UU)),
  F H -> transf  (functor_ViewOb H)  E).
Variables  (ee_congr : typeOf_destructCongr  ee_)
(ee_natural : typeOf_destructNatural  ee_)
(V : vertexGene) (VV : sieveFunctor V)
(VV_base:  typeOf_baseSieve VV)
 (uv : 'Sieve(U ~> V | VV) ) .
Variables (D : functor) (dd : transf D F) (UU_base':  typeOf_baseSieve UU)
(W : vertexGene) (wu : 'Sieve(W ~> U | UU))
(WW : sieveFunctor W) (WW_base: typeOf_baseSieve WW) .

Lemma RestrictCast_comp_Destructing_destructCongr  :
typeOf_destructCongr (fun H (w : 'Sieve(H ~> W | WW)) f =>
    @ee_ H ((w :>sieve_) o>sieve_ wu) (f :>transf_ dd)) .
Proof. move. intros H. move. intros w1 w2 Heq_w. move. intros d1 d2 Heq_d.
apply: ee_congr.  rewrite -> Heq_w. reflexivity.
rewrite -> Heq_d. reflexivity.
Qed.

Lemma RestrictCast_comp_Destructing_destructNatural :
typeOf_destructNatural (fun H (w : 'Sieve(H ~> W | WW)) f =>
    @ee_ H ((w :>sieve_) o>sieve_ wu) (f :>transf_ dd)).
Proof.     move. intros G w form H f G' g w' form' f' Heq_w Heq_form Heq_f.
     apply: (ee_natural (g:= g)).
    - rewrite <- Heq_w. rewrite <- _natural_transf.
      rewrite <-  _functorialCompos_functor'. reflexivity.
    - rewrite <- Heq_form. rewrite <- _natural_transf. reflexivity.
    - exact: Heq_f.
Qed.

Definition Congr_RestrictCast_comp_Destructing:
(transf_Compos (transf_RestrictCast dd UU_base' wu WW_base )
  (transf_Destructing UU_base ee_congr ee_natural VV_base uv)   )
==1 (transf_Destructing WW_base (RestrictCast_comp_Destructing_destructCongr)
         (RestrictCast_comp_Destructing_destructNatural) VV_base  ((wu
:>sieve_) o>sieve_ uv)   ).
Proof. intros. move. intros H f_ f_' Heq_f_. rewrite -> Heq_f_.
clear Heq_f_.  unshelve eexists.
- shelve.
- cbn_sieve. exact: (sieveTransf_Ident _).
- cbn_sieve. exact: (sieveTransf_Ident _).
- intros J c.  unshelve eexists.
  + cbn_sieve. rewrite <- _natural_transf.
  do 1 rewrite <- _functorialCompos_functor'. reflexivity.
  + cbn_sieve. cbn_sieve in c.
```

```
  refine (interSieve_congr_sieveEquiv _ (reflexivity _) (reflexivity _)).
  etransitivity. refine (interSieve_congr_sieveEquiv (reflexivity _) _
(reflexivity _)).
  (rewrite -> _natural_transf; reflexivity).
  symmetry; apply: interSieve_image_sieveEquiv. exact: UU_base.
  + intros H0 c0; cbn_sieve. apply: ee_congr.
    * apply: UU_base.  unfold  _rel_relType, equiv; simpl.
    do 2 rewrite -> _wholeProp_interSieve. cbn_sieve.
    rewrite -> _functorialCompos_functor'. do 1 rewrite <- _natural_transf.
reflexivity.
    * do 2 apply: _congr_relTransf. unshelve eexists; cbn_transf;
reflexivity.
    * reflexivity.
Qed.

End RestrictCast_comp_Destructing.

Definition Congr_Constructing_comp_Restrict_cast :
forall (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
(F  : functor) (K : vertexGene) (u : 'Sieve(K ~> _ | UU)) (form : F K),
forall ( E : functor)  (ff : transf F E) (V : vertexGene) (VV  : sieveFunctor
V)
 (VV_base: typeOf_baseSieve VV)
 (uv : 'Sieve(U ~> V | VV)),
(transf_Compos (transf_Constructing u form) (transf_RestrictCast ff VV_base
uv UU_base) )
==1 (transf_Constructing  ((u :>sieve_) o>sieve_ uv) (form :>transf_ ff))  .
Proof. intros. intros G k1 k2 Heq_k.
 unshelve eexists; cbn_transf;
first (rewrite -> Heq_k; rewrite -> _functorialCompos_functor';
do 2 rewrite <- _natural_transf; reflexivity).
 - symmetry; apply: interSieve_image_sieveEquiv. exact: UU_base.
 - intros H c; cbn_sieve.
 rewrite <- _natural_transf. apply: _congr_relFunctor; last reflexivity.
 apply: _congr_relFunctor; last rewrite -> Heq_k; reflexivity.
Qed.

(* /!\ SOLUTION /!\ *)
Definition Congr_Constructing_comp_Destructing :
forall (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU) (F E : functor)
(ee_ : forall (H : vertexGene) (u : 'Sieve(H ~> _ | UU)),
      F H -> transf  (functor_ViewOb H)  E)
(ee_congr : typeOf_destructCongr ee_)
(ee_natural : typeOf_destructNatural ee_)
(V : vertexGene) (VV : sieveFunctor V)  (VV_base: typeOf_baseSieve VV)
(uv : 'Sieve(U ~> V | VV) ),
forall   (K : vertexGene) (u : 'Sieve(K ~> _ | UU)) (form : F K),
 (transf_Compos (transf_Constructing u form)
```

```
              (transf_Destructing UU_base ee_congr ee_natural VV_base uv))
==1 (transf_UnitSheafified UU_base u (ee_ K u  form) VV_base uv).
Proof.  intros. move. intros H h h0 Heq. rewrite -> Heq. unshelve eexists.
- exact (identSieve _).
- exact: (sieveTransf_Ident _).
- exact: (sieveTransf_Ident _).
- intros J c.  unshelve eexists.
  + reflexivity.
  + reflexivity.
  + intros H0 c0. cbn_sieve. cbn_sieve in c0. rewrite -> _natural_transf.
    symmetry; apply: ee_natural.
    * apply: UU_base.  unfold  _rel_relType, equiv; simpl.
    rewrite -> _wholeProp_interSieve. cbn_sieve.
     do 2 rewrite <- _natural_transf.
     rewrite -> _functorialCompos_functor'.
    reflexivity.
    * cbn_transf. reflexivity.
    * cbn_sieve. etransitivity; first exact: identGene_composGene;
    symmetry; exact: composGene_identGene.
Qed.

Definition Congr_Constructing_comp_Gluing :
forall (U : vertexGene) (UU : sieveFunctor U)
(UU_base: typeOf_baseSieve UU)
(VV_ : forall H : vertexGene, 'Sieve( H ~> _ | UU ) -> sieveFunctor H)
(VV_congr : typeOf_sieveCongr VV_)
(VV_natural : typeOf_sieveNatural VV_)
(F E : functor)
(ee_ : forall (H : vertexGene) (u : 'Sieve( H ~> _ | UU )),
        transf (functor_Restrict F (VV_ H u)) (functor_Sheafified E))
(ee_congr : typeOf_gluingCongr VV_congr ee_)
(ee_natural : typeOf_gluingNatural VV_natural ee_)
(ee_compat : typeOf_gluingCompat VV_congr VV_natural ee_),
forall (K : vertexGene) (u : 'Sieve(K ~> _ | (sumSieve VV_))) (form : F K),
   (transf_Compos (transf_Constructing u form)
      (transf_Gluing UU_base ee_congr ee_natural ee_compat))
   ==1   (transf_Compos (transf_Constructing (_inner_sumSieve u) form)
      (ee_ (_object_sumSieve u) (_outer_sumSieve u))).
Proof.  intros. symmetry. move. intros G form'0 form'  Heq. rewrite -> Heq.
clear Heq. etransitivity.
cbn -[equiv _type_relType _rel_relType _equiv_relType   functor_ViewOb
                                 transf_ViewObMor transf_Constructing ].
apply: (gluingNatural_identGene_of_gluingNatural ee_natural).

have @identGene_u : 'Sieve(G ~> _ | interSieve UU
((( form' o>sieve_ _inner_sumSieve u) :>sieve_)
o>functor_ (_outer_sumSieve u :>sieve_)) (identSieve G)) .
  refine (((identGene : 'Sieve(G ~> _ | identSieve G) )
            :>transf_ interSieve_image
                        ((( form' o>sieve_ _inner_sumSieve u) :>sieve_)
```

```
                          o>functor_ _outer_sumSieve u) _)
      :>transf_ (interSieve_congr (sieveTransf_Ident _) _
(sieveTransf_Ident _) )).
  abstract (rewrite <- _natural_transf; reflexivity).

 (* To get this unsimplification, continue and do
  refine (sieveTransf_Compos _ (sumSieve_interSieve_image _  )).
 *)
have Heq_ee: ((transf_RestrictMor_pullSieve (form' :>transf_
transf_Constructing (_inner_sumSieve u) form) identGene
     :>transf_ transf_RestrictMor (transf_Ident F)
                   (VV_natural (_object_sumSieve u) (_outer_sumSieve u) G
                      (identGene o>gene _indexer_type_Restrict (form'
:>transf_ transf_Constructing (_inner_sumSieve u) form))))
  :>transf_ ee_ G  ((identGene o>gene _indexer_type_Restrict (form' :>transf_
transf_Constructing (_inner_sumSieve u) form)) o>sieve_
                   _outer_sumSieve u))
==  (transf_Gluing_lemma (form' :>transf_ transf_Constructing u form)
identGene_u
  :>transf_ ee_ G (identGene_u
                   :>transf_ interSieve_projWhole UU (_indexer_type_Restrict
(form' :>transf_ transf_Constructing u form))
                               (_sieve_type_Restrict (form' :>transf_
transf_Constructing u form)))).
abstract (unshelve apply: ee_congr;
first abstract (cbn_sieve; rewrite -> _functorialCompos_functor';
reflexivity);
last unshelve eexists; cbn_sieve;
  first reflexivity;
  first reflexivity;
  last intros H c; reflexivity).

unshelve eexists.
- exact: (conflSieve_Sheafified Heq_ee) . (*  exact (identSieve _).  *)
- exact: (conflTransf1_Sheafified _).  (* READ Heq_ee HERE *)
- refine (sieveTransf_Compos (conflTransf2_Sheafified _) _).
 refine (sieveTransf_Compos _ (sumSieve_congr
(UU1 := pullSieve UU (((( form' o>sieve_ _inner_sumSieve u) :>sieve_)
o>functor_ _outer_sumSieve u) :>sieve_) )  (fun H0 u0 => sieveTransf_Ident _)
) ).
refine (sieveTransf_Compos _ (sumSieve_interSieve_image _  )).
subst identGene_u. exact: (sieveTransf_Ident _).
apply: (@conflEquiv_Sheafified _ _ _ _ Heq_ee).
Defined.

End COMOD.
End SHEAF.
```

S1 / coq

Voila.