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

The “double plus” definition of sheafification says that not-only the outer families-of-families are modulo the germ-equality, but-also the inner families are modulo the germ-equality. This outer-inner contrast is the hint that the “double plus” should be some inductive construction... that grammatical sheaf cohomology exists!  
 And the MODOS proof-assistant 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

.

**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.

Click or tap here to enter text.

**Check** 37:nat. **Goal** 0=0. reflexivity. **Qed**.

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

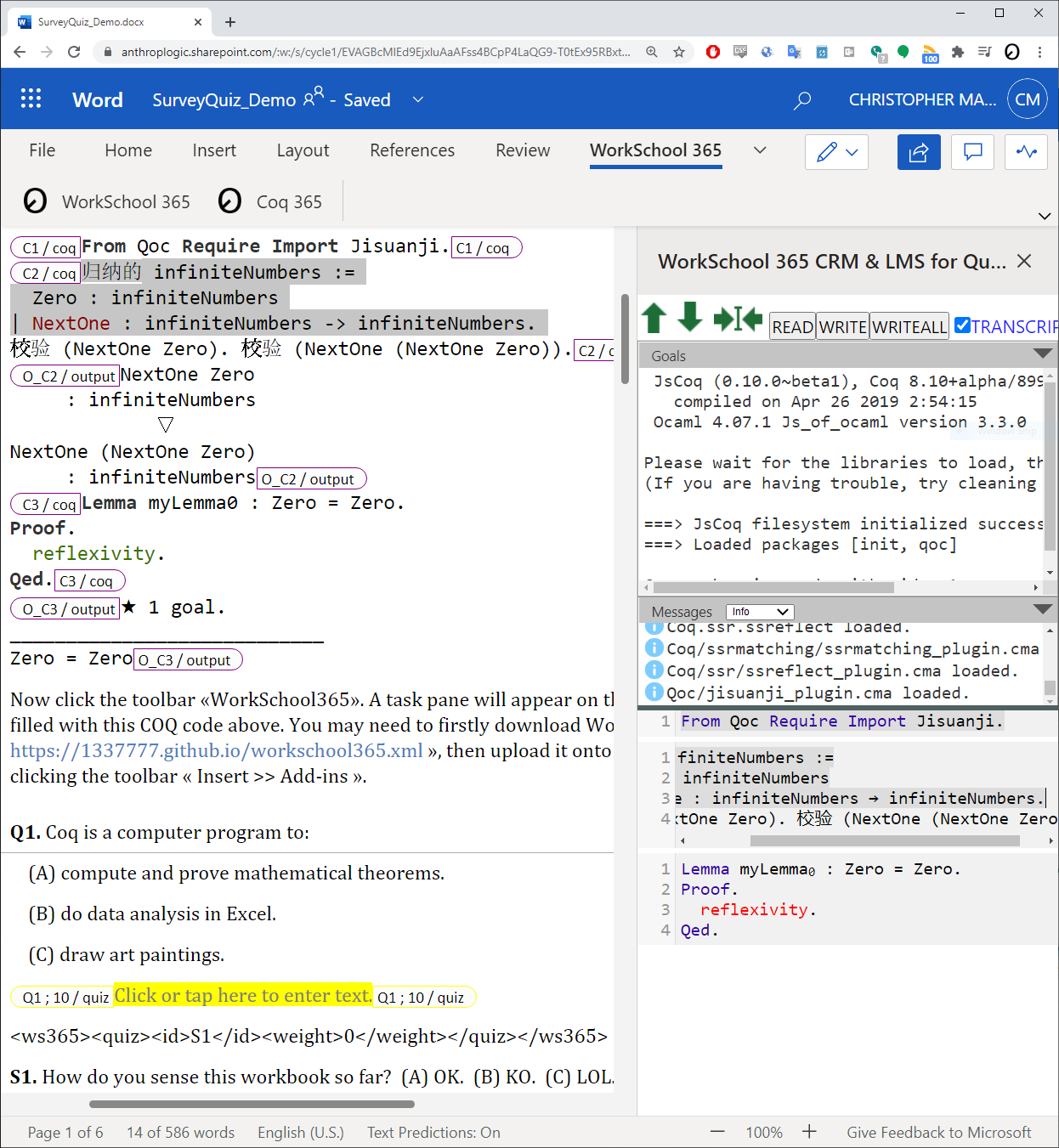
WorkSchool 365 is your education marketplace where the prompting authors pay to get peer reviews of their documents from any ***learning reviewers who pass the quiz*** polls inside the prompting document, with ***shareable transcripts receipts*** of the school work.

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](https://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 ∂∂ = 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***”:

G

U

V\_

g

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\_)

sieve V\_ at U

intersection with

any sieve W\_

pullback sieve g\*V\_

along g

And the definition of the sheafification object is:

G

A\_

elements e : (E A\_) of E

over local-pieces A\_ of G

( Sheafified E ) G := Sum (A\_ : sieve at G) × (E A\_)

any sieve A\_ at G

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⟧ → ⟦f∘Bg⟧ 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.

(1 ∘ f) ∘ 1

(Sheafified E) ((1 ∘ f) ∘ 1)

)

f ∘ 1

f

(Sheafified E) (1 ∘ f)

1 ∘ f

E (f)

element

of shape F

defined by cases over

the constructors of

the dataobject F

**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).

0

ucℤ⊕vcℤ

0

utℤ⊕vtℤ

0

0

(xu∘>uc)ℤ⊕(xv∘>vc)ℤ

in\_uℤ⊗in\_vℤ

(xu∘>uc - xv∘>vc)ℤ

xtℤ

0

\_-\_

1⊕1

\_+\_

(1,-1)

1⊕-1

**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](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](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](https://stacks.math.columbia.edu/tag/01L%20Q)

(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).

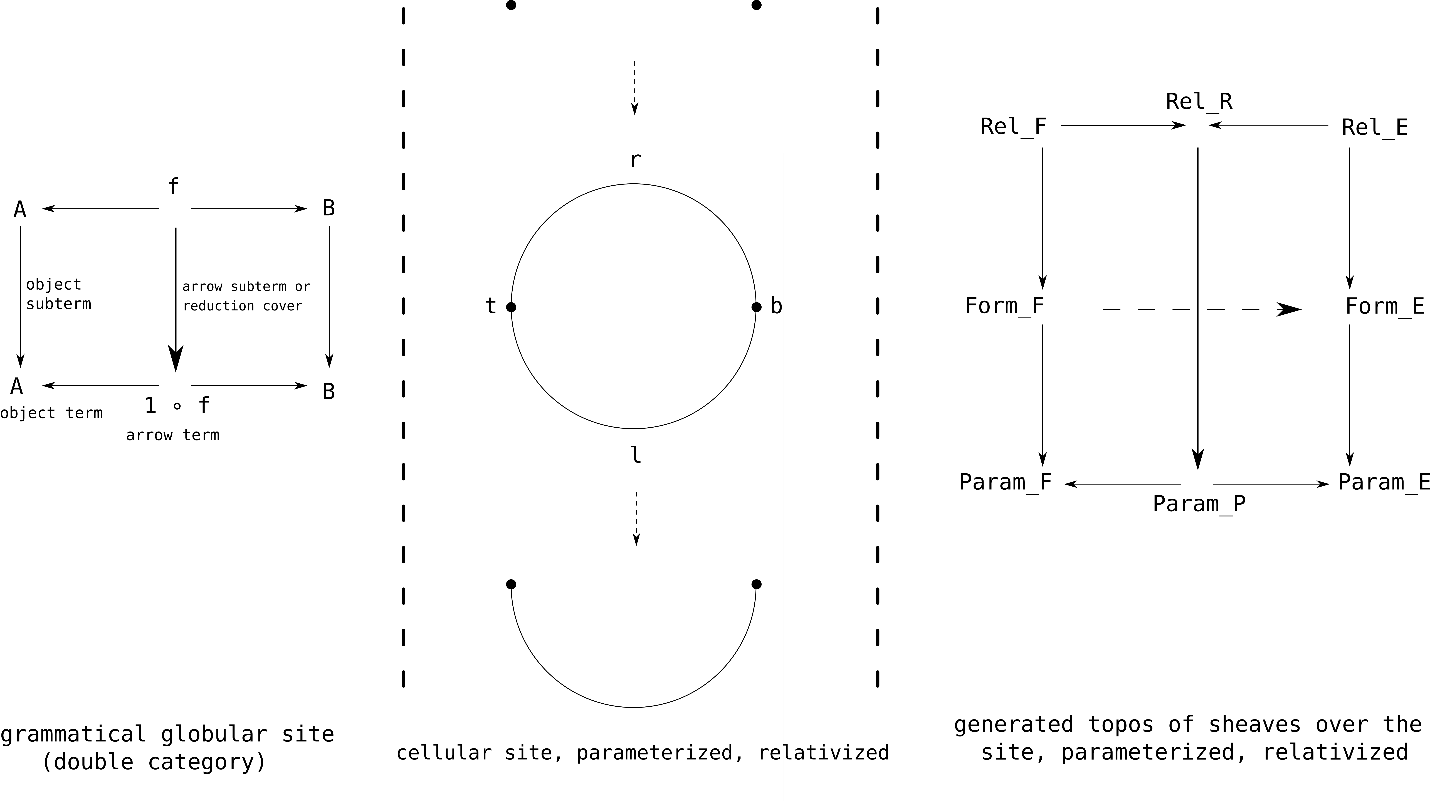
(4.) The MODOS is the homotopical computational logic for ***geometric dataobjects and parsing***, which is some generalization of the constructive-inductive datatypes in logic and the sheaves in geometry.

## 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***.

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>

**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.

Voila.