WorkSchool 365 for e-commerce and e-learning with applications to proof-assistants for geometric algorithmics and quantum physics

Short: WorkSchool 365 is the business-school for e-commerce and e-learning with published applications in the Microsoft Commercial Marketplace: open-source code ***WorkSchool 365 CRM & LMS*** with ***SurveyQuiz*** transcripts and ***EventReview*** receipts, and ***MODOS*** proof-assistant of dependent-constructive-computational-logic for geometric algorithmics and quantum physics.

The *SurveyQuiz* and *EventReview* e-commerce and e-learning ***open-source code*** applications are some integration of many popular business software to enable learners/reviewers to share the transcripts/receipts of their (quizzes/reviews) school/work using ***no-password*** user identities with auditability of authorship. The SurveyQuiz are Word documents/forms of large-scale automatically-graded *survey/quizzes* with ***shareable transcripts*** of School by the learners, with integration of the Coq proof-assistant Word add-in and samples from the Gentle Introduction to the Art of Mathematics textbook. The EventReview are Microsoft Teams video meetings with SharePoint databases/calendars of paid/remunerated *review-tasks* for Word documents/events (seminars, conferences, archive papers) with ***shareable receipts*** of Work by the reviewers, whose reviews are appended to the task.

The *MODOS* research application is some library of new-mathematics documents and their proof-assistant, with possible applications in geometric algorithmics and quantum-fields physics. The MODOS proof-assistant is the homotopical computational logic for ***geometric dataobjects and parsing***, which is some common generalization of the constructive-inductive datatypes in logic and the sheaves in geometry. Indeed, the usual datatypes in logic 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. Elsewhere, the usual substructural-proof technique of dagger compact monoidal categories (linear logic of duality) allow to formulate the computational content of quantum mechanics, but fails to articulate the computational-logic content in the differential geometry of quantum-fields jet-bundles parameterized over some spacetime manifold. The MODOS is 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, destruction, gluing).

Outline:

In this Word document, click “Insert ; Add-ins ; WorkSchool 365 Coq” to ***play this script interactively****.*

1. WorkSchool 365 CRM & LMS for applications in e-commerce and e-learning (SurveyQuiz transcripts, EventReview receipts)

(1.) What problem is to be solved? From the legal perspective, as prescribed by many legislative assemblies everywhere, any school is defined by its ability to output the shareable transcripts/receipts records of the ***learning-discovery-engineering-and-teaching/reviewing*** done by users as learners and reviewers (teachers). Links: https://www.ontario.ca/laws/statute/00p36

An ambient legal requirement is that there shall be no ***forced/assault-fool/[intoxicated-by-bad-habits]-and-theft/lie/falsification*** of those transcripts/receipts records. One component of the solution is the authentication of the users without requiring excessive personal information (beyond some email address). Another component is the sharing/authorization of access to the transcripts/receipts records, with auditability of the authorship of the data.

From the commerce perspective, any business is defined by its ability to account for the direct currency (review-assessment, citation, credit, cash money, share certificate, cryptocurrency, …) transactions among all the trading parties (learners-reviewers) without requiring excessive financial information (beyond some payout address) and without relying on indirect government/public currencies.

(2.) WorkSchool 365 is ***Open Source Code Secure Microsoft 365 SharePoint Teams application with PayPal + Stripe Payments***. WorkSchool 365 integrates the ***Customer Relationship Management (CRM) + Learning Management System (LMS)*** for your Business School to engage/qualify/educate prospective users into paying/subscribed/grantee learners/customers or paid/remunerated reviewers/merchants via an integration of PayPal and Stripe shop e-commerce payment (Card, Alipay, WeChat Pay) and Microsoft Marketplace API payment along with Microsoft Business Applications (MBA) for user management (Azure AD), documents database (SharePoint), video conferencing (Teams), and automation from events (Power Automate). Links: https://appsource.microsoft.com/en-us/marketplace/apps?search=WorkSchool%20365

(2.1) WorkSchool 365 SurveyQuiz are Word documents of large-scale automatically-graded survey/quizzes with ***shareable transcripts*** of School by the learners, with integration of the ***open-source code Coq 365 proof-assistant Word add-in*** and samples from the Gentle Introduction to the Art of Mathematics textbook (https://giam.southernct.edu).

(2.2) WorkSchool 365 EventReview are Microsoft Teams video meetings + SharePoint databases/calendars of paid/remunerated review-tasks for Word documents/events with ***shareable receipts*** of Work by the reviewers, whose reviews are appended to the task.

(2.3) WorkSchool 365 UserGraduation are ***no-password*** sign-in/sign-up of unlimited number of users authenticated via Microsoft/Azure or Google or Facebook or Email, and distributed in graduation teams. The users in Cycle 3 (Reviewers) may create their own thematic instances of the SurveyQuiz and EventReview for the free users in Cycle 1 (Learners) or the paying non-free users in Cycles 2 (Seminarians).

(2.4) WorkSchool 365 is ***open source code*** at: https://github.com/1337777/workschool365 and https://github.com/1337777/1337777.github.io with demo instance at https://anthroplogic.WorkSchool365.com

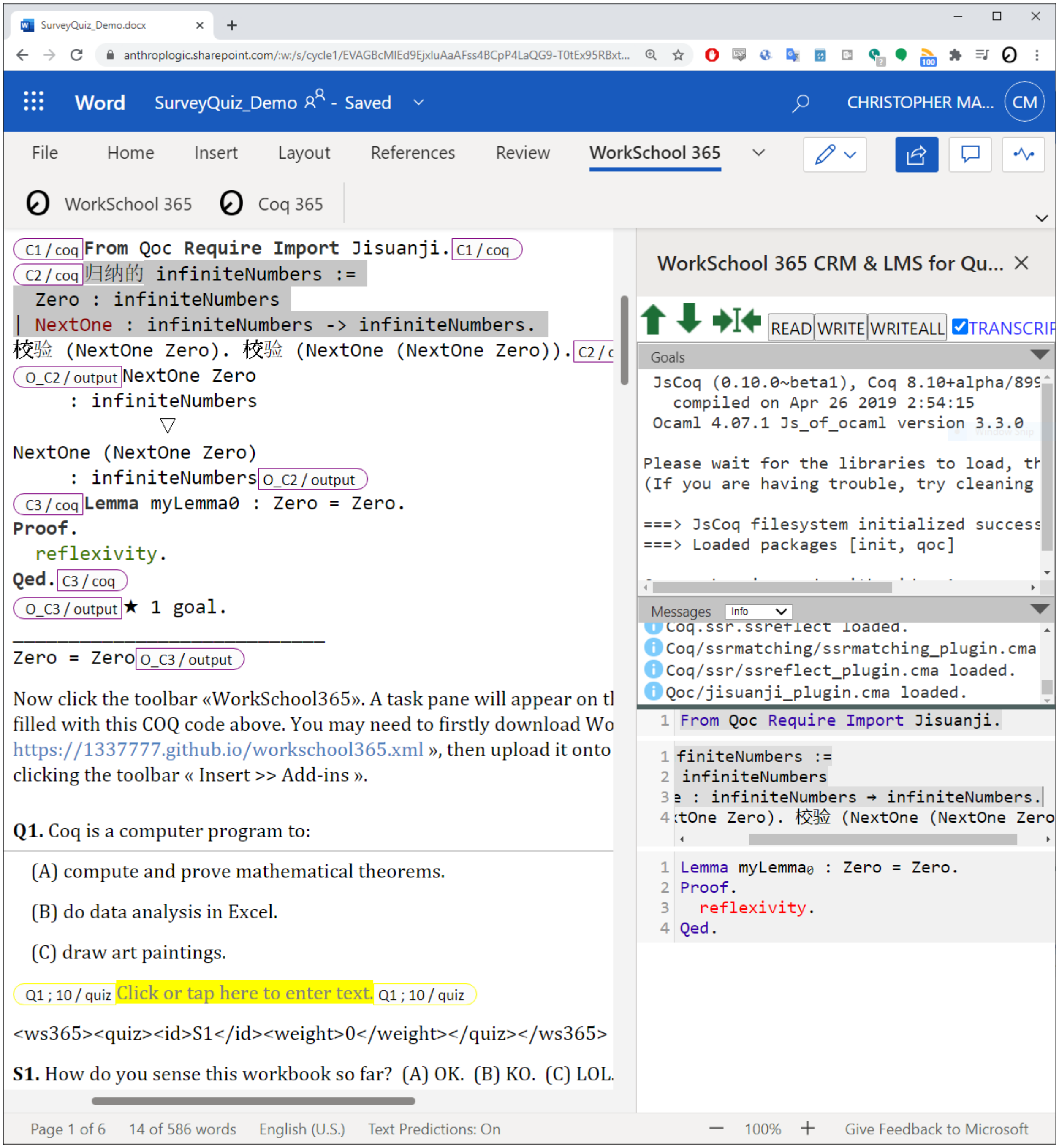


Figure: SurveyQuiz Word document code-ranges being parsed and selected by the Coq add-in in the web browser, with the quiz-ranges responses saved for later automatic grading.

2. WorkSchool 365 for research applications (MODOS proof-assistant)

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

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

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

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

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

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

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

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

(5.) MODOS proof-assistant has solved the critical techniques behind those questions, even if the production-grade engineering is still lacking. Some programming techniques ("cut-elimination", "confluence", "dependent-typed functional programming"...) from computer-science (electrical circuits) generalize to the alternative reformulation of categorial-algebra as a cut-elimination computational-logic ***("adjunctions", "comonads", "products", "enriched categories", "internal categories", "2-***

categories", "fibred category with local internal products", "associativity coherence", "semi-associativity coherence", "star-autonomous category coherence",...). Links: https://github.com/1337777/cartier ; https://github.com/1337777/dosen

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

2.1. MODOS proof-assistant possible applications in the computational logic for 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 phi(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.

3. Appendix: What is the minimal example of sheaf cohomology? Grammatically

Short: Hold any Dosen-style ***cut-elimination 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 the reduction-conversion morphisms) 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 arrows-operations. 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 sheaf-models: G ↦ Hom(sheafified(Hom(− , G)), fpModelsSet(_))

(A.) Morphisms: the shape of the point is now “A” instead of singleton, 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)
```

(B.) Algebraic-geometric dataobjects: the elimination scheme for the dataobjects gives the base of the construction for the sheafification; continued with the refinements/gluing scheme below any sieve...

```
| Destructing_nonRecursiveSignature : forall (F : data diagram) and (E : any
diagram) and (VV : sieve in site),

given the family of morphims (ee_ : forall (U : object in site) (f : F U)
(cons_f : isConstructor F f), Hom( Restrict U VV ~> E )),

then Hom( Restrict F VV ~> Sheafi E VV )

| Refinement : forall (F : data diagram) and (E : any diagram) and (VV :
sieve on V in site) and family of sieves (WW_ : forall V', Site( V' ~> V | in
VV ) -> sieves)),

given the family of morphims (ee_ : forall V' (v : Site( V' ~> V | in VV )),
Hom( Viewing F WW_v ~> Sheafi E WW_v )),

then Hom( Restrict F VV ~> Sheafi E VV ).
```

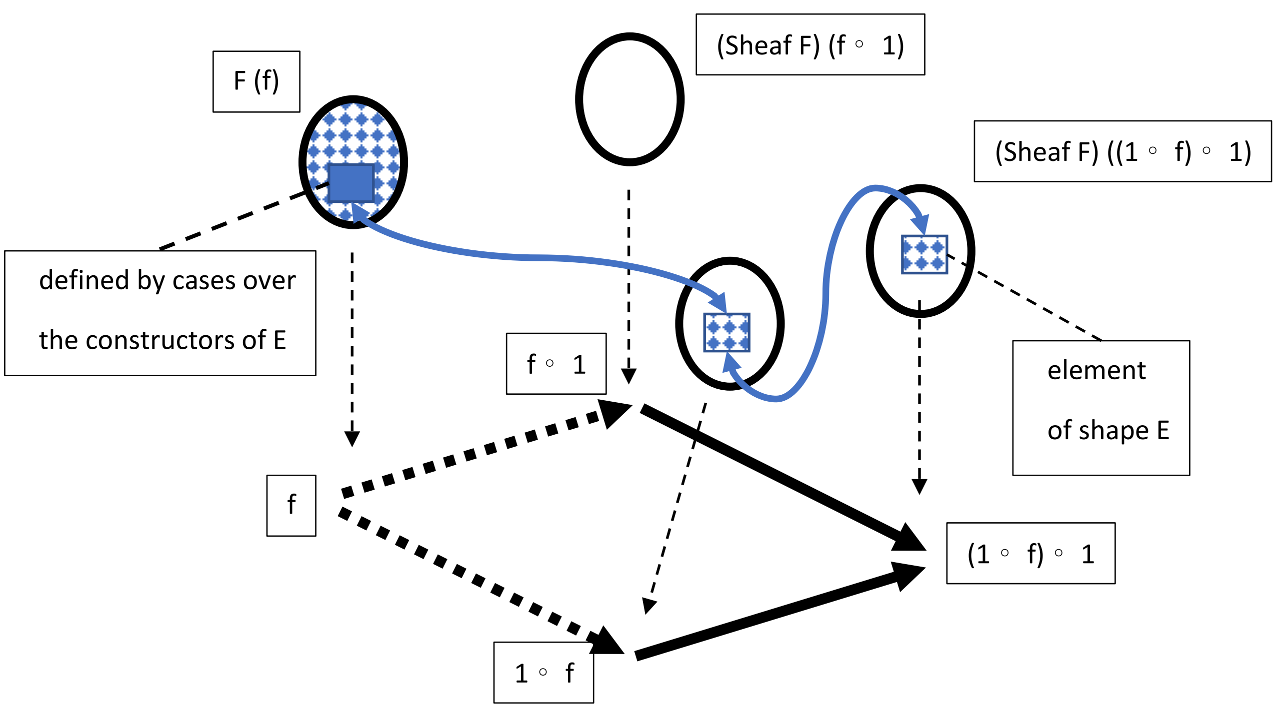
```
Lemma: cut-elimination holds. Corollary: grammatical sheaf cohomology exists.
```

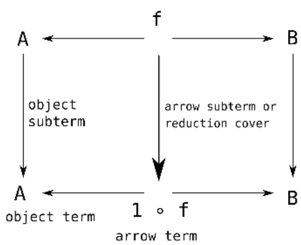
3.1. The generating site of arrow-terms with confluence

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

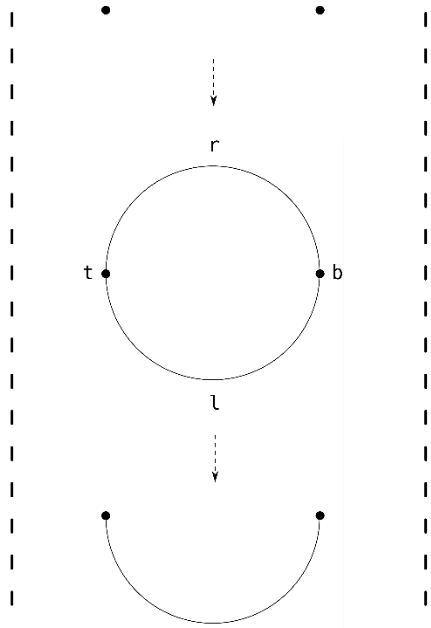
Moreover, it may be assumed some generating ***cocontinuous adjunction of sites*** (fibre of any covering sieve is covering), which is some instance of morphism of sites generating some geometric morphism of toposes. Examples of this assumption are: ***the étale map from the circle to the projective 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-computational-logic alternative to Urs Schreiber's geometry of quantum-fields physics which uses half-axiomatic cohesive-topos.

General sheaf cohomology over any site may also be formulated in this computational-logic, for example: Hold the site of the 3-points space with two open sets U and V which have another non-empty intersection W. Hold M be the sheaf generated by two elements f function on U and g function on V, without any assumption of compatibility over W. Hold N be the sheaf generated by two elements f' function on U and g' function on V and generated by one compatibility relation between f' and g' over W. Hold mn be the transformation of sheaves from M to N which maps f to f' and maps g to g'. Then mn has surjective image-sheaf, but is not surjective map at each open. The lemma is that this description can be written grammatically. In short: ***MODOS interfaces the COQ categorial logic of sheaves down to the GAP/SINGULAR algebra of modules***.

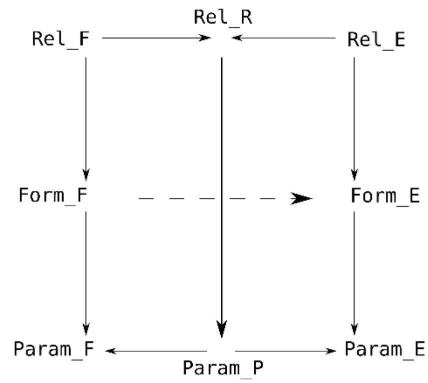




grammatical globular site
(double category)



cellular site, parameterized, relativized



generated topos of sheaves over the
site, parameterized, relativized

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

3.2. 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

3.3. Outline of the grammar

The generating site:

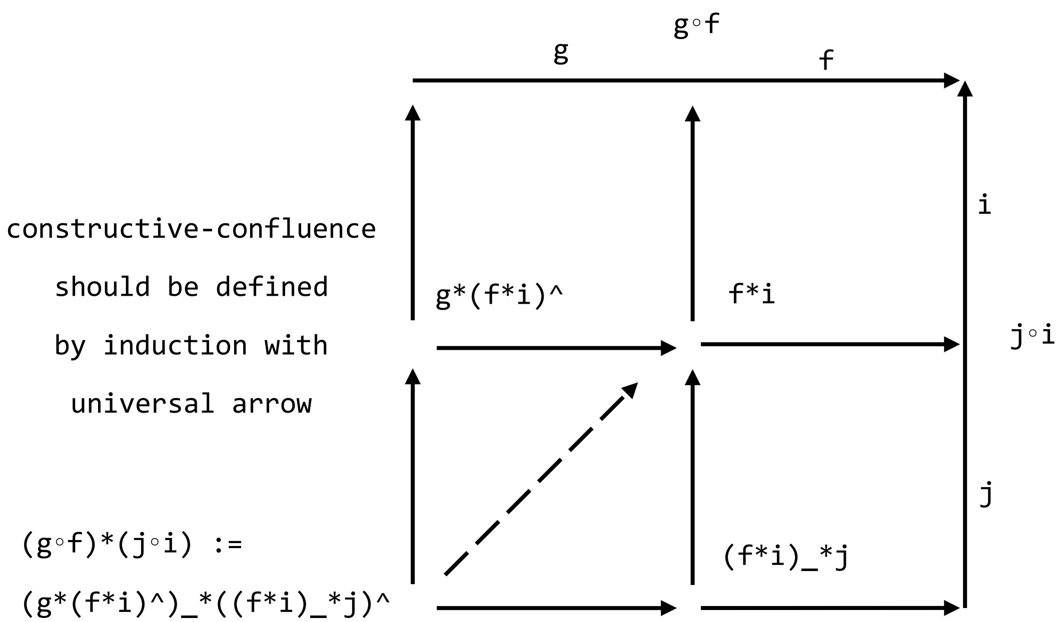
C1_format / coq

```
Parameter obGenerator : eqType.

Parameter morGenerator : obGenerator -> obGenerator -> Type.
Notation "''Generator' ( V ~> U )" := (@morGenerator V U)
  (at level 0, format "''Generator' (  V  ~>  U  )") : poly_scope.

Parameter polyGenerator :
  forall U V, 'Generator( V ~> U ) -> forall W, 'Generator( W ~> V ) ->
'Generator( W ~> U ).
Notation "wv o>Generator vu" := (@polyGenerator _ _ vu _ wv)
  (at level 40, vu at next level) : poly_scope.
```

```
Parameter unitGenerator : forall {U : obGenerator}, 'Generator( U ~> U ).
```

C1_format / coq

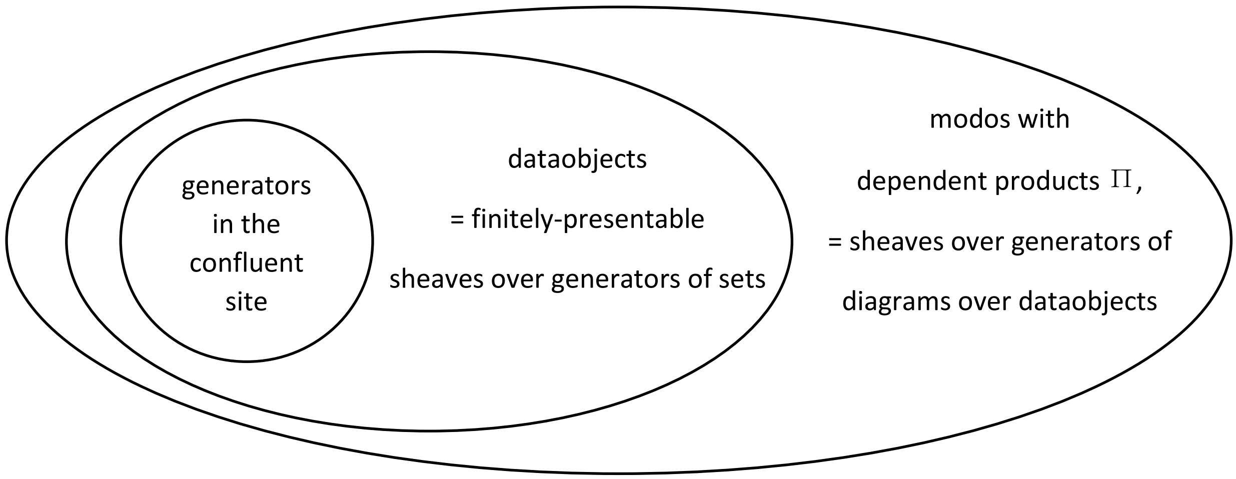
The objects of this generating site are arrow-terms of another grammar, and the morphisms of the site are reductions or subterms; and these define the covering sieves. Also it is assumed that the site has been completely solved already: some form of constructive-confluence is available. Any significant example, such as the pairing-product of arrow-terms, would involve some ***constructive-confluence*** with explicit universal/polymorphic morphisms, as outlined in the earlier file ***dosenSolution101.v*** . For the pullbacks example, this says that the universal morphism from `(g∘f)*(i)` to `f*i` is constructive.



In fact, these generators are the most basic dataobjects of the generated modos. Indeed, each generator object has some (polymorph) singleton constructor element (the unit morphism). More complex dataobjects are the finitely-presentable objects in the modos. Reminder that the modos (grammar) is whatever grammatical description is possible to express relative over the classifying topos (sense). And the classifying topos consists of the covariant diagrams of sets over the finitely-presentable contravariant Set-sheaf models; moreover the ***generic model*** is

```
G ↦ Hom(sheafified(Hom(-, G)), fpModelsSet(_))
```

At present, the expressiveness of the modos is limited, for example any diagram of sets over the dataobjects is only constant diagram of some set. And in the example below, any sieve is generated only by some singleton morphism; but the formulation would extend easily to general sieves (ref the file ***cartierSolution8.v***).



The codes for the morphisms:

C2_format / coq

```
Inductive morCode : forall (Sense00_E : obGenerator -> Type)
(Sense01_E : Sense01_def Sense00_E), forall (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F), Sense1_def Sense01_E Sense01_F -> Type
:= _
with morCode_Family : forall U V (vu : 'Generator( V ~> U )) (Sense00_F :
obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F) (Sense00_E :
obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E) (Sense1_ee__ :
forall (G : obGenerator)  (form : Sense00_F G), Sense1_def (Sense01_Viewing
(Sense01_ViewOb G) vu) Sense01_E) (Sense1_ee_morphism : Morphism_prop
Sense01_F Sense1_ee__), Type := _.
```

C2_format / coq

Because the objects may depent on the morphisms (such as for the pi-product object), this dependence must be indirect via pseudo codes for morphisms. Here [Sense01_def] specifies some action and its functoriality, and [Sense1_def] specifies some transformation and its naturality.

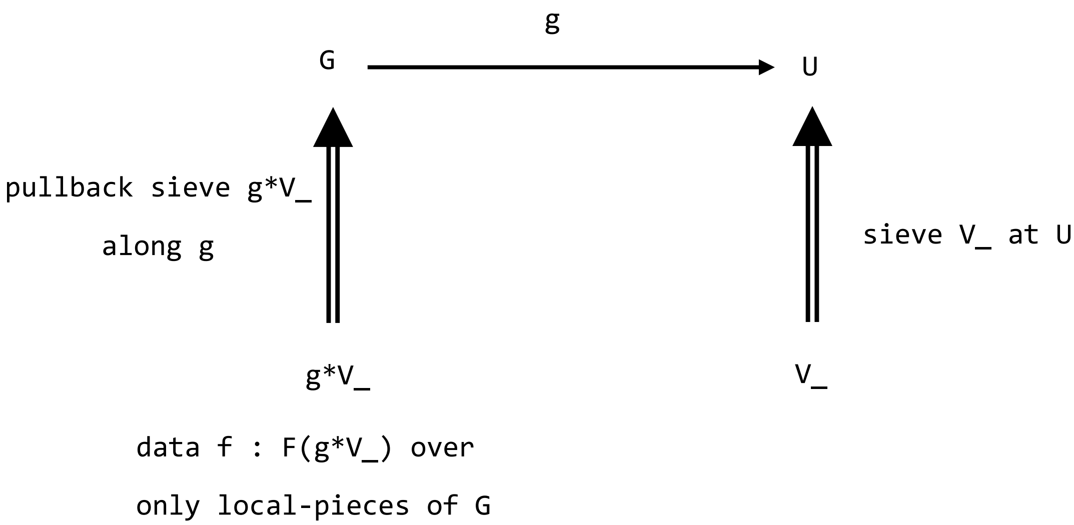
The objects and the data-objects :

C3_format / coq

```
Inductive obCoMod : forall Sense00_F (Sense01_F : Sense01_def
Sense00_F), Type := _
with obData : forall Sense00_F (Sense01_F : Sense01_def Sense00_F), Type :=
_.
```

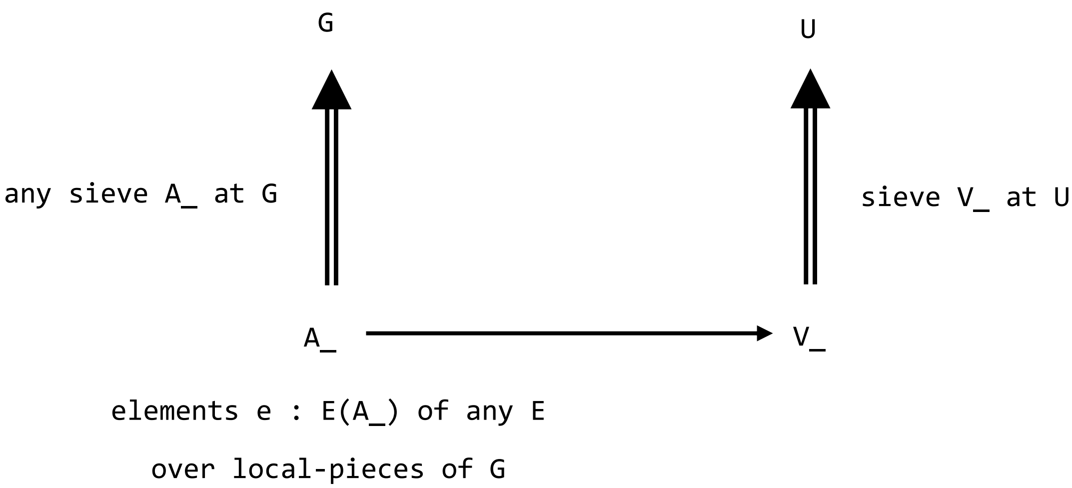
C3_format / coq

The grammatical entry [obData] is for only the data (=finitely-presentable) objects, and the entry [obCoMod] is for any object. Moreover there is some constructor [Viewing] from [obData] to [obCoMod], which takes some sieve as parameter and view/restrict the dataobject below the sieve (the pullback of the sieve to this dataobject). Now any function out of this dataobject to some other object is in fact some local-function (family) over the viewing-pieces of the dataobject to the other object.



(Viewing F V_) G := Sum (g : G → U) × F(g*V_)

The constructor [ViewedOb] is sheafification, such that, tautologically, it can absord any family (local-function) of its elements again as some single element. Clearly, the reductions-conversions relations on the grammatical(=very-nonunique) morphisms ensure the separateness (“uniqueness” condition), here therefore sheafification is essentially the “plus construction”.



(Viewed E V_) G := Sum (A_ sieve at G) × E(A_) | A_ contained in V_

The constructor elements and algebraic elements of the dataobjects :

C4_format / coq

```
Inductive elConstruct : forall (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F) (F : obData Sense01_F), forall (G :
obGenerator)  (form : Sense00_F G), Type := _
```

```
with elAlgebra : forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F) (F : obData Sense01_F), forall (G : obGenerator) (form : Sense00_F G), Type := _.
```

C4_format / coq

These grammatical entries are only for dataobjects [obData]. The algebra such as the restriction operation or the zero-plus operations extend from the actual constructors. Given that each constructor should be polymorph, then the restriction operation [Restrict_elAlgebra] can be eliminated/accumulated onto the constructors in the solution.

The algebraic conversions (equations) on the algebraic elements:

C5_format / coq

```
Inductive convElAlgebra : forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F) (F : obData Sense01_F), forall (G : obGenerator) (form : Sense00_F G) (cons_form : elAlgebra F form ), forall (form' : Sense00_F G) (cons_form' : elAlgebra F form'), ElCongr_def form form' -> Type := _
where "cons_f0  [ Congr_f_f0 ]<==  cons_f" := _.
```

C5_format / coq

The codes for the conversions on the morphisms:

C6_format / coq

```
Inductive congrMorCode : forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E), forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F), forall (Sense1_ff : Sense1_def Sense01_E Sense01_F) (Code_ff : morCode Sense1_ff), forall (Sense1_ff' : Sense1_def Sense01_E Sense01_F) (Code_ff' : morCode Sense1_ff'), forall (Congr_congr_ff : Congr_def Sense1_ff Sense1_ff'), Type := _
where "''CoMod$' (  Code_ff  ~>  Code_ff'  @_ Congr_congr_ff  )" := _.
```

C6_format / coq

The reductions-conversions on the morphisms may change the codes of the morphisms, therefore this change must be tracked explicitly via other codes for the conversions. Any grammatical operation which takes some morphism as argument shall carry the code for the conversions of this morphism, and accumulate the changes into this code.

The morphisms:

C7_format / coq

```
Inductive morCoMod : forall Sense00_E Sense01_E (E : @obCoMod Sense00_E Sense01_E), forall Sense00_F Sense01_F (F : @obCoMod Sense00_F Sense01_F), forall (Sense1_ff : Sense1_def Sense01_E Sense01_F) (Code_ff : morCode Sense1_ff), Type := _
where "''CoMod' ( E ~> F @_ Code_ff )" := _.
```

C7_format / coq

The elimination scheme [Constructing] for the dataobjects gives the base of the construction for the sheafification; continued with the refinements/gluing scheme [Refinement] below any sieve... Note that [Constructing] is some hidden Yoneda.

The conversions on the morphisms:

C8_format / coq
```
Inductive convCoMod : forall Sense00_E Sense01_E (E : @obCoMod
Sense00_E Sense01_E), forall Sense00_F Sense01_F (F : @obCoMod Sense00_F
Sense01_F), forall (Sense1_ff : Sense1_def Sense01_E Sense01_F) (Code_ff :
morCode Sense1_ff ) (ff : 'CoMod( E ~> F @_ Code_ff )), forall (Sense1_ff0 :
Sense1_def Sense01_E Sense01_F) (Code_ff0 : morCode Sense1_ff0 ) (ff0 :
'CoMod( E ~> F @_ Code_ff0 )), forall (Congr_congr_ff : Congr_def Sense1_ff
Sense1_ff0) (congr_ff : 'CoMod$( Code_ff ~> Code_ff0 @_ Congr_congr_ff  )),
Prop := _
where "ff0 [  congr_ff  ]<~~ ff" := _.
```
C8_format / coq

Those reductions-conversions are of 4 kinds: the reflexivity and transitivity reductions, the congruence reductions, the (top-most) cut-reductions, and any noncut conversion. Here the cancellation (evaluation) reduction-conversion is [Constructing_comp_Destructing].

The example over the natural numbers site:

The example at the end of the playable script in the next section assumes that the generating site is the natural numbers.

C9_format / coq
```
Declare Module Import CoMod : (COMOD NatGenerator).

Parameter (GFixed : obGenerator).

Definition example_morphism :
{ Sense1_ff : Sense1_def _ _ &
{ Code_ff : morCode Sense1_ff &
'CoMod( Viewing (ViewOb GFixed) (eq_refl _ : 2 <= 3) ~>
 ViewedOb (Viewing (ViewOb GFixed) (eq_refl _ : 0 <= 0)) (eq_refl _ : 2 <= 3)
@_ Code_ff ) }}.
Proof.
repeat eexists.
eapply Refinement with (vu := (eq_refl _ : 2 <= 3)) (2 := Refl_congrMorCode).
eapply Refinement with (vu := (eq_refl _ : 1 <= 2)) (2 := Refl_congrMorCode).
eapply Refinement with (vu := (eq_refl _ : 0 <= 1)) (2 := Refl_congrMorCode).
eapply Destructing with (vu := (eq_refl _ : 0 <= 0)).
intros. eapply Compos.
- apply Constructing, ElConstruct_elAlgebra, (ViewOb_elConstruct
unitGenerator).
- move: (elConstruct_obDataViewObP GFixed cons_form).
elim (eq_comparable GFixed GFixed) => [ /=  ?  cons_form_P | // ].
destruct cons_form_P.
apply Constructing, ElConstruct_elAlgebra, (ViewOb_elConstruct g).

Unshelve. all: intros; try apply Congr_AtMember_Compos_morCode_Family;
try apply AtMember_Compos_morCode_Family_congrMorCode.
Defined.

Definition example_reduction:
{ Sense1_ff : Sense1_def _ _ &
```

```
{ Code_ff : morCode Sense1_ff &
{ ff : 'CoMod( _ ~> _ @_ Code_ff ) &
{ Congr_congr_ff : Congr_def _ _ &
{ congr_ff : 'CoMod$( _ ~> _ @_ Congr_congr_ff ) &
( ff )  [ congr_ff ]<~~
 ((Constructing (eq_refl _ : 2 <= 3) (ElConstruct_elAlgebra
(ViewOb_elConstruct unitGenerator)))
   o>CoMod (projT2 (projT2 example_morphism)))
}}}}}.
Proof.
repeat eexists.  simpl.
eapply convCoMod_Trans.
eapply Constructing_comp_Refinement.
eapply convCoMod_Trans.
eapply Refinement_cong, Constructing_comp_Refinement.
eapply convCoMod_Trans.
eapply Refinement_cong, Refinement_cong, Constructing_comp_Refinement.
eapply convCoMod_Trans.
eapply Refinement_cong, Refinement_cong, Refinement_cong,
Constructing_comp_Destructing.
simpl. destruct (eq_comparable GFixed GFixed); last by []; simpl.
eapply convCoMod_Trans.
eapply Refinement_cong, Refinement_cong, Refinement_cong, UnitViewedOb_cong,
Constructing_comp_Constructing.
exact: convCoMod_Refl.
Unshelve. all: try apply Refl_congrMorCode.
Defined.
Eval simpl in (projT1 (projT2 (projT2 example_reduction))).
(*
= Refinement (eqxx (2 - 3))
    (Refinement (eqxx (1 - 2))
      (Refinement (eqxx (0 - 1))
          (UnitViewedOb
            (Constructing (eqxx (0 - 0))
                (Restrict_elAlgebra
                  (ElConstruct_elAlgebra
                      (ViewOb_elConstruct unitGenerator)) unitGenerator)))
          Refl_congrMorCode) Refl_congrMorCode) Refl_congrMorCode
: 'CoMod( Viewing (ViewOb GFixed) (eqxx (2 - 3) : 1 < 3) ~>
  ViewedOb (Viewing (ViewOb GFixed) (eqxx (0 - 0) : 0 <= 0))
    (eqxx (2 - 3) : 1 < 3) @_ projT1 (projT2 example_reduction)) *)
```

C9_format / coq

3.4. Example

In this Word document (search "WorkSchool365.docx"), click on "Insert ; Add-ins" and search "WorkSchool 365 Coq". Next click "Coq" to load and ***play this script interactively***.

S1 / coq
```
(** # #
#+TITLE: cartierSolution0.v
```

Proph

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

shows the general outline of the solutions to some question of CARTIER which is
 how to program the MODOS proof-assistant for
 « dependent constructive computational logic for geometric dataobjects »
 (including homotopy types) ...

OUTLINE ::

* Generating site, its cut-elimination and confluence

* Generated modos, its cut-elimination and confluence

* Example

-----

* Generating site, its cut-elimination and confluence

```
#+BEGIN_SRC coq :exports both :results silent # # **)
From mathcomp Require Import ssreflect ssrfun ssrbool eqtype ssrnat.
From Coq Require Lia.

Module SHEAF.

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

Notation "''sval'" := (@proj1_sig _ _).

Declare Scope poly_scope. Delimit Scope poly_scope with poly. Open Scope poly.

Module Type GENERATOR.

Parameter obGenerator : eqType.

Parameter morGenerator : obGenerator -> obGenerator -> Type.
Notation "''Generator' ( V ~> U )" := (@morGenerator V U)
(at level 0, format "''Generator' (  V  ~>  U  )") : poly_scope.

Parameter polyGenerator :
forall U V, 'Generator( V ~> U ) -> forall W, 'Generator( W ~> V ) -> 'Generator( W ~> U ).
Notation "wv o>Generator vu" := (@polyGenerator _ _ vu _ wv)
(at level 40, vu at next level) : poly_scope.

Parameter unitGenerator : forall {U : obGenerator}, 'Generator( U ~> U ).

Parameter polyGenerator_morphism :
forall (U V : obGenerator) (vu : 'Generator( V ~> U ))
        (W : obGenerator) (wv : 'Generator( W ~> V )),
forall X (xw : 'Generator( X ~> W )),
  xw o>Generator ( wv o>Generator vu ) = ( xw o>Generator wv ) o>Generator vu.
Parameter polyGenerator_unitGenerator :
forall (U V : obGenerator) (vu : 'Generator( V ~> U )),
  vu = ((@unitGenerator V) o>Generator vu ).
Parameter unitGenerator_polyGenerator :
forall (U : obGenerator), forall (W : obGenerator) (wv : 'Generator( W ~> U )),
    wv = ( wv o>Generator (@unitGenerator U)).
```

```
Parameter ConflVertex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )), obGenerator.
Parameter ConflProject :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
  'Generator( ConflVertex projecter indexer ~> IndexerVertex ).
Parameter ConflIndex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
  'Generator( ConflVertex projecter indexer ~> ProjecterVertex ).
Parameter ConflCommuteProp :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
  ConflProject projecter indexer o>Generator indexer
  = ConflIndex projecter indexer o>Generator projecter.

Parameter ConflMorphismIndex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
  'Generator( ConflVertex projecter (preIndexer o>Generator indexer) ~>
                          ConflVertex projecter indexer ).
Parameter ConflMorphismIndexCommuteProp :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
ConflProject projecter (preIndexer o>Generator indexer) o>Generator preIndexer
= ConflMorphismIndex projecter indexer preIndexer o>Generator ConflProject projecter indexer
/\  ConflIndex projecter (preIndexer o>Generator indexer)
    = ConflMorphismIndex projecter indexer preIndexer o>Generator ConflIndex projecter
indexer.

Parameter ConflProp_ComposIndex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
{ CommonConflVertex : obGenerator &
{ CommonConfl1 : 'Generator( CommonConflVertex ~> (ConflVertex (ConflProject projecter
indexer) preIndexer )) &
{ CommonConfl2 : 'Generator( CommonConflVertex ~> (ConflVertex projecter (preIndexer
o>Generator indexer ))) |
  CommonConfl1 o>Generator (ConflProject (ConflProject projecter indexer) preIndexer )
  = CommonConfl2 o>Generator (ConflProject projecter (preIndexer o>Generator indexer ))
  /\  CommonConfl1 o>Generator ((ConflIndex (ConflProject projecter indexer) preIndexer ))
      = CommonConfl2 o>Generator (ConflMorphismIndex projecter indexer preIndexer )
} } }.

Parameter ConflProp_AssocIndex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
forall PrePreIndexerVertex (prePreIndexer : 'Generator( PrePreIndexerVertex ~>
PreIndexerVertex )),
{ CommonConflVertex : obGenerator &
{ CommonConfl1 : 'Generator( CommonConflVertex ~>
 (ConflVertex projecter (prePreIndexer o>Generator (preIndexer o>Generator indexer)))) &
{ CommonConfl2 : 'Generator( CommonConflVertex ~>
 (ConflVertex projecter ((prePreIndexer o>Generator preIndexer) o>Generator indexer))) |
```

```
  CommonConfl1 o>Generator (ConflProject projecter (prePreIndexer o>Generator (preIndexer
o>Generator indexer)))
  = CommonConfl2 o>Generator (ConflProject projecter ((prePreIndexer o>Generator preIndexer)
o>Generator indexer))
  /\  CommonConfl1 o>Generator ((ConflMorphismIndex projecter (preIndexer o>Generator indexer)
prePreIndexer)
                                  o>Generator (ConflMorphismIndex projecter indexer
preIndexer))
      = CommonConfl2 o>Generator (ConflMorphismIndex projecter indexer (prePreIndexer
o>Generator preIndexer))
} } }.

Parameter ConflProp_MorphismIndexRelativeProject :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
{ CommonConflVertex : obGenerator &
{ CommonConfl1 : 'Generator( CommonConflVertex ~> ConflVertex projecter
                            (ConflMorphismIndex projecter (indexer) preIndexer
                            o>Generator (ConflProject projecter (indexer)
                                          o>Generator indexer))) &
{ CommonConfl2 : 'Generator( CommonConflVertex ~> ConflVertex projecter
                                (ConflProject projecter (preIndexer o>Generator indexer)
                                o>Generator (preIndexer o>Generator indexer))) |
CommonConfl1 o>Generator ConflProject projecter (ConflMorphismIndex projecter (indexer)
preIndexer
o>Generator (ConflProject projecter (indexer) o>Generator indexer))
= CommonConfl2 o>Generator ConflProject projecter
(ConflProject projecter (preIndexer o>Generator indexer) o>Generator (preIndexer o>Generator
indexer))
/\  CommonConfl1 o>Generator (ConflMorphismIndex projecter (ConflProject projecter (indexer)
o>Generator indexer)
(ConflMorphismIndex projecter (indexer) preIndexer)
  o>Generator ConflMorphismIndex projecter (indexer) (ConflProject projecter (indexer)))
= CommonConfl2 o>Generator (ConflMorphismIndex projecter (preIndexer o>Generator indexer)
                              (ConflProject projecter (preIndexer o>Generator indexer))
        o>Generator ConflMorphismIndex projecter (indexer) preIndexer)
} } }.

Parameter ConflProp_ComposRelativeIndex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall PreProjecterVertex (preProjecter : 'Generator( PreProjecterVertex ~> ProjecterVertex
)),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
{ CommonConflVertex : obGenerator &
{ CommonConfl1 : 'Generator( CommonConflVertex ~>
ConflVertex preProjecter (ConflIndex projecter (preIndexer o>Generator indexer))) &
{ CommonConfl2 : 'Generator( CommonConflVertex ~> ConflVertex preProjecter
                     (ConflMorphismIndex projecter indexer preIndexer
                      o>Generator ConflIndex projecter indexer)) |
CommonConfl1 o>Generator ConflProject preProjecter (ConflIndex projecter (preIndexer
o>Generator indexer))
= CommonConfl2 o>Generator ConflProject preProjecter (ConflMorphismIndex projecter indexer
preIndexer
                                                          o>Generator ConflIndex projecter
indexer)
/\  CommonConfl1 o>Generator (ConflProject preProjecter (ConflIndex projecter (preIndexer
o>Generator indexer))
o>Generator ConflMorphismIndex projecter indexer preIndexer)
= CommonConfl2 o>Generator (ConflMorphismIndex preProjecter (ConflIndex projecter indexer)
```

```
  (ConflMorphismIndex projecter indexer preIndexer)
o>Generator ConflProject preProjecter (ConflIndex projecter indexer))
} } }.

Parameter ConflProp_MixIndexProject_1 :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
forall PreProjecterVertex (preProjecter : 'Generator( PreProjecterVertex ~> ConflVertex
projecter indexer )),
{ CommonConflVertex : obGenerator &
{ CommonConfl1 : 'Generator( CommonConflVertex ~>
 ConflVertex (preProjecter o>Generator ConflProject projecter indexer) preIndexer ) &
{ CommonConfl2 : 'Generator( CommonConflVertex ~>
 ConflVertex preProjecter (ConflMorphismIndex projecter indexer preIndexer)) |
  CommonConfl1 o>Generator ConflProject (preProjecter o>Generator ConflProject projecter
indexer) preIndexer
  = CommonConfl2 o>Generator (ConflProject preProjecter (ConflMorphismIndex projecter indexer
preIndexer)
                                          o>Generator ConflProject projecter (preIndexer
o>Generator indexer))
  /\  CommonConfl1 o>Generator (ConflIndex (preProjecter o>Generator ConflProject projecter
indexer) preIndexer)
      = CommonConfl2 o>Generator (ConflIndex preProjecter (ConflMorphismIndex projecter
indexer preIndexer))
} } }.

End GENERATOR.

Module Type COMOD (Generator : GENERATOR).
Import Generator.
(** # #
#+END_SRC

* Generated modos, its cut-elimination and confluence

#+BEGIN_SRC coq :exports both :results silent # # **)
Definition Sense01_action (Sense00 : obGenerator -> Type)
(Sense01 : forall G G' : obGenerator, 'Generator( G' ~> G ) -> Sense00 G -> Sense00 G')
          G G' (g : 'Generator( G' ~> G)) (x : Sense00 G)
:= (Sense01 G G' g x).

Notation "g o>Generator_ [ Sense01 ] x" := (@Sense01_action _ Sense01 _ _ g x)
 (at level 40, x at next level) : poly_scope.
Notation "g o>Generator_ x" := (@Sense01_action _ _ _ _ g x)
(at level 40, x at next level) : poly_scope.

Definition Sense01_functor (Sense00 : obGenerator -> Type)
 (Sense01 : forall G G' : obGenerator, 'Generator( G' ~> G ) -> Sense00 G -> Sense00 G') :
Prop :=
( forall G G' (g : 'Generator( G' ~> G)) G'' (g' : 'Generator( G'' ~> G')) x,
    g' o>Generator_[Sense01] (g o>Generator_[Sense01] x)
    = (g' o>Generator g) o>Generator_[Sense01] x ) /\
( forall G x, x = (@unitGenerator G) o>Generator_[Sense01] x ).

Definition Sense01_def (Sense00 : obGenerator -> Type)
:= { Sense01 : ( forall G G' : obGenerator, 'Generator( G' ~> G ) -> Sense00 G -> Sense00 G' )
|
      Sense01_functor Sense01 }.

Definition Sense1_natural Sense00_F (Sense01_F : Sense01_def Sense00_F)
```

```
 Sense00_E (Sense01_E : Sense01_def Sense00_E) (Sense1 : forall G : obGenerator, Sense00_F G -
> Sense00_E G) : Prop :=
forall G G' (g : 'Generator( G' ~> G )) (f : Sense00_F G),
  g o>Generator_[sval Sense01_E] (Sense1 G f)
  = Sense1 G' (g o>Generator_[sval Sense01_F] f).

Definition Sense1_def Sense00_F (Sense01_F : Sense01_def Sense00_F) Sense00_E (Sense01_E :
Sense01_def Sense00_E)
:=  { Sense1 : ( forall G : obGenerator, Sense00_F G -> Sense00_E G ) |
      Sense1_natural Sense01_F Sense01_E Sense1 }.

Notation "''exists'  x  ..." := (exist _ x _) (at level 10, x at next level) : poly_scope.
Notation "[<  data  |  ...  >]" := (@existT _ (fun data => @sigT _ _) data _) (at level 0) :
poly_scope.

Lemma Sense00_ViewOb : forall (G : obGenerator), (obGenerator -> Type).
Proof. intros G. refine (fun H => 'Generator( H ~> G )). Defined.

Lemma Sense01_ViewOb : forall (G : obGenerator), Sense01_def (Sense00_ViewOb G).
Proof.
intros. unshelve eexists.
- intros H H' h. refine (fun g => h o>Generator g).
- abstract (split; [intros; exact: polyGenerator_morphism
                    | intros; exact: polyGenerator_unitGenerator]).
Defined.

Record Sense00_Viewing Sense00_F (Sense01_F : Sense01_def Sense00_F)
      U V (vu : 'Generator( V ~> U )) (G: obGenerator) : Type :=
{ getIndexerOfViewing : 'Generator( G ~> U ) ;
  getDataOfViewing : Sense00_F (ConflVertex vu getIndexerOfViewing)
}.

Axiom Sense00_Viewing_quotient :
 forall Sense00_F (Sense01_F : Sense01_def Sense00_F)
  U V (vu : 'Generator( V ~> U )),
forall G : obGenerator, forall (f1 f2 : Sense00_Viewing Sense01_F vu G),
forall (CommonConflVertex : obGenerator)
(CommonConfl1 : 'Generator( CommonConflVertex ~> (ConflVertex vu (getIndexerOfViewing f1))))
(CommonConfl2 : 'Generator( CommonConflVertex ~> (ConflVertex vu (getIndexerOfViewing f2)))),
CommonConfl1 o>Generator (ConflProject vu (getIndexerOfViewing f1))
= CommonConfl2 o>Generator (ConflProject vu (getIndexerOfViewing f2)) ->
CommonConfl1 o>Generator_[sval Sense01_F] (getDataOfViewing f1)
= CommonConfl2 o>Generator_[sval Sense01_F] (getDataOfViewing f2)
-> f1 = f2.

Definition Sense01_Viewing Sense00_F (Sense01_F : Sense01_def Sense00_F)
          U V (vu : 'Generator( V ~> U ))
: Sense01_def (Sense00_Viewing Sense01_F vu ).
Proof.
intros. unshelve eexists.
- intros G G' g f. exists ( g o>Generator (getIndexerOfViewing f)).
exact: ((ConflMorphismIndex vu (getIndexerOfViewing f) g)
            o>Generator_[sval Sense01_F] (getDataOfViewing f)).
- abstract (split; simpl;
[ intros G G' g G'' g' f;
move: (ConflProp_AssocIndex vu (getIndexerOfViewing f) g g' ) =>
  [CommonConflVertex [CommonConfl1 [CommonConfl2 [HeqProject HeqIndex]]]];
    unshelve eapply Sense00_Viewing_quotient; simpl;
    [
    | exact CommonConfl1
    | exact CommonConfl2
```

```
    | assumption
    |
    ];
    do 2 rewrite [LHS](proj1 (proj2_sig Sense01_F));
        rewrite [RHS](proj1 (proj2_sig Sense01_F));
        congr( _ o>Generator_ _);
        rewrite -polyGenerator_morphism;
        assumption
  | intros G f;
    unshelve eapply Sense00_Viewing_quotient; simpl;
    [
    | exact (ConflMorphismIndex vu (getIndexerOfViewing f) unitGenerator)
    | exact unitGenerator
    | rewrite -(proj1 (ConflMorphismIndexCommuteProp vu (getIndexerOfViewing f)
unitGenerator));
      rewrite -[RHS]polyGenerator_unitGenerator -[LHS]unitGenerator_polyGenerator; reflexivity
    | rewrite [RHS](proj1 (proj2_sig Sense01_F));
      congr( _ o>Generator_ _);
      rewrite -[RHS]polyGenerator_unitGenerator; reflexivity
  ]]).
Defined.

Record Sense00_ViewedOb Sense00_F (Sense01_F : Sense01_def Sense00_F)
      U V (vu : 'Generator( V ~> U )) (G: obGenerator) : Type :=
{ getProjectVertexOfViewed : obGenerator ;
  getProjectOfViewed : 'Generator( getProjectVertexOfViewed ~> G ) ;
  getDataOfViewed : Sense00_F getProjectVertexOfViewed ;
  getConditionOfViewed : 'Generator( getProjectVertexOfViewed ~> V )
}.

Axiom Sense00_ViewedOb_quotient :
forall Sense00_F (Sense01_F : Sense01_def Sense00_F)
        U V (vu : 'Generator( V ~> U )) (G: obGenerator),
forall (f1 f2 : Sense00_ViewedOb Sense01_F vu G),
forall (CommonConflVertex : obGenerator)
    (CommonConfl1 : 'Generator( CommonConflVertex ~> getProjectVertexOfViewed f1 ))
    (CommonConfl2 : 'Generator( CommonConflVertex ~> getProjectVertexOfViewed f2 )),
  CommonConfl1 o>Generator (getProjectOfViewed f1)
  = CommonConfl2 o>Generator (getProjectOfViewed f2) ->
  CommonConfl1 o>Generator_[sval Sense01_F] (getDataOfViewed f1)
  = CommonConfl2 o>Generator_[sval Sense01_F] (getDataOfViewed f2)
  -> f1 = f2.

Definition Sense01_ViewedOb Sense00_F (Sense01_F : Sense01_def Sense00_F)
          U V (vu : 'Generator( V ~> U ))
: Sense01_def (Sense00_ViewedOb Sense01_F vu).
Proof.
intros. unshelve eexists.
- intros G G' g f. exact
{| getProjectVertexOfViewed :=(ConflVertex (getProjectOfViewed f) g) ;
  getProjectOfViewed := (ConflProject (getProjectOfViewed f) g) ;
  getDataOfViewed := ((ConflIndex (getProjectOfViewed f) g)
                    o>Generator_[sval Sense01_F] (getDataOfViewed f)) ;
  getConditionOfViewed := ((ConflIndex (getProjectOfViewed f) g)
                    o>Generator (getConditionOfViewed f))
|}.
- abstract (split; simpl;
[ intros G G' g G'' g' f;
move: (ConflProp_ComposIndex (getProjectOfViewed f) g g' ) =>
[CommonConflVertex [CommonConfl1 [CommonConfl2 [HeqProject HeqIndex]]]];
unshelve eapply Sense00_ViewedOb_quotient; simpl;
```

```
[
| exact CommonConfl1
| exact CommonConfl2
| assumption
|
];
do 2 rewrite [LHS](proj1 (proj2_sig Sense01_F));
rewrite [RHS](proj1 (proj2_sig Sense01_F));
congr( _ o>Generator_ _); rewrite HeqIndex; rewrite -polyGenerator_morphism;
rewrite -(proj2 (ConflMorphismIndexCommuteProp _ _ _ )); reflexivity
| intros G f;
  unshelve eapply Sense00_ViewedOb_quotient; simpl;
  [
  | exact (ConflIndex (getProjectOfViewed f) unitGenerator)
  | exact unitGenerator
  | rewrite -(ConflCommuteProp (getProjectOfViewed f) unitGenerator);
    rewrite -polyGenerator_unitGenerator -unitGenerator_polyGenerator; reflexivity
  | rewrite [RHS](proj1 (proj2_sig Sense01_F));
    congr( _ o>Generator_ _);
    rewrite -polyGenerator_unitGenerator; reflexivity
]]).
Defined.

Definition element_to_polyelement : forall Sense00_F (Sense01_F : Sense01_def Sense00_F) G,
  Sense00_F G -> Sense1_def (Sense01_ViewOb G) Sense01_F.
Proof.
intros ? ? G f. unshelve eexists.
apply: (fun G' g => g o>Generator_[sval Sense01_F] f).
abstract (move; simpl; intros G' G'' g' g;
          rewrite -(proj1 (proj2_sig Sense01_F)); reflexivity).
Defined.

Definition Sense1_Compos :
forall (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)
(Sense00_F' : obGenerator -> Type)
(Sense01_F' : Sense01_def Sense00_F')
(Sense1_ff' : Sense1_def Sense01_F' Sense01_F)
(Sense00_F'' : obGenerator -> Type)
(Sense01_F'' : Sense01_def Sense00_F'')
(Sense1_ff_ : Sense1_def Sense01_F'' Sense01_F'),
Sense1_def Sense01_F'' Sense01_F.
Proof.
intros. unshelve eexists.
- intros G dataIn.
apply: (sval Sense1_ff' G (sval Sense1_ff_ G dataIn)).
- abstract (move; simpl; intros; rewrite [LHS](proj2_sig Sense1_ff');
          rewrite (proj2_sig Sense1_ff_); reflexivity).
Defined.

Definition Sense1_Constructing_default :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F),

forall (G : obGenerator) (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) (Sense01_Viewing Sense01_F vu).
Proof.
intros. unshelve eexists.
- intros H h. exact
{|
```

```
getIndexerOfViewing := getIndexerOfViewing h;
  getDataOfViewing := getDataOfViewing h o>Generator_[sval Sense01_F] form
|}.
- abstract (move; simpl; intros; unshelve eapply Sense00_Viewing_quotient; simpl;
[
| exact unitGenerator
| exact unitGenerator
| reflexivity
| rewrite -(proj1 (proj2_sig Sense01_F)); reflexivity
]).
Defined.

Definition Sense1_ViewObMor :
forall (G : obGenerator) (H : obGenerator) (g : 'Generator( H ~> G )),
Sense1_def (Sense01_ViewOb H) (Sense01_ViewOb G).
Proof.
intros G H hg. unshelve eexists.
- intros G0 h. exact: (  h o>Generator hg ).
- abstract (move; simpl; intros ; rewrite /Sense01_action /= ; exact: polyGenerator_morphism).
Defined.

Definition Sense1_Viewing Sense00_F (Sense01_F : Sense01_def Sense00_F)
U V (vu : 'Generator( V ~> U ))
Sense00_E (Sense01_E : Sense01_def Sense00_E)
(Sense1_ff : Sense1_def Sense01_F Sense01_E) :
Sense1_def (Sense01_Viewing Sense01_F vu) (Sense01_Viewing Sense01_E vu).
Proof.
intros. unshelve eexists.
- intros G f. exact
{|
getIndexerOfViewing := getIndexerOfViewing f;
  getDataOfViewing :=
    sval Sense1_ff (ConflVertex vu (getIndexerOfViewing f))
        (getDataOfViewing f)
|}.
- abstract (move; intros; simpl;
unshelve eapply Sense00_Viewing_quotient; simpl;
[ | exact unitGenerator
  | exact unitGenerator
  | reflexivity
  | rewrite (proj2_sig Sense1_ff); reflexivity ] ).
Defined.

Definition Morphism_prop
U V (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(Sense1_ee__ : forall (G : obGenerator) (form : Sense00_F G),
  Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E) :=
forall (G : obGenerator)  (form : Sense00_F G),
forall (G' : obGenerator) (g : 'Generator( G' ~> G )) ,
forall (H : obGenerator) (f0 : (Sense00_Viewing (Sense01_ViewOb G') vu) H) f,
(*  pb (g'o>g) V = V = pb (g) V *)
f =  (sval (Sense1_Viewing vu (Sense1_ViewObMor g)) H f0) ->
(sval (Sense1_ee__ G form) H f) =
(sval (Sense1_ee__ G' (g o>Generator_[sval Sense01_F] form)) H f0).

Lemma Morphism_Constructing
: forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
Morphism_prop Sense01_F (@Sense1_Constructing_default _ _ vu _ Sense01_F ).
```

```
Proof.
intros; move; intros; subst; unshelve eapply Sense00_Viewing_quotient; simpl;
[ | exact unitGenerator
| exact unitGenerator
| reflexivity
| congr ( _ o>Generator_ _);
  rewrite (proj1 (projT2 Sense01_F)); reflexivity
].
Qed.

Definition Sense1_Destructing :
forall U V (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(Sense1_ee : forall (G : obGenerator) (form : Sense00_F G),
  Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee),
Sense1_def (Sense01_Viewing Sense01_F vu ) (Sense01_ViewedOb Sense01_E vu).
Proof.
intros. unshelve eexists.
- intros G f. exact
{|
getProjectVertexOfViewed := ConflVertex vu (getIndexerOfViewing f);
getProjectOfViewed := ConflProject vu (getIndexerOfViewing f);
getDataOfViewed :=
sval
  (Sense1_ee (ConflVertex vu (getIndexerOfViewing f))
             (getDataOfViewing f)) (ConflVertex vu (getIndexerOfViewing f))
  {|
    getIndexerOfViewing :=
      ConflProject vu (getIndexerOfViewing f)
                  o>Generator getIndexerOfViewing f;
    getDataOfViewing :=
      ConflMorphismIndex vu (getIndexerOfViewing f)
                        (ConflProject vu (getIndexerOfViewing f))
  |};
getConditionOfViewed := ConflIndex vu (getIndexerOfViewing f)
|}.
- abstract (move; simpl; intros G G' g' f;
move: (ConflProp_ComposIndex vu (getIndexerOfViewing f) g' )
=> [CommonConflVertex [CommonConfl1 [CommonConfl2 [HeqProject HeqIndex]]]];
unshelve eapply Sense00_ViewedOb_quotient; simpl;
[
| exact CommonConfl1
| exact CommonConfl2
| assumption
|
];
do 1 rewrite [LHS](proj1 (proj2_sig Sense01_E));
rewrite HeqIndex;
do 1 rewrite -[LHS](proj1 (proj2_sig Sense01_E));
congr (_ o>Generator_ _);
do 1 rewrite [in LHS](proj2_sig (Sense1_ee _ _));
apply: Sense1_ee_morphism;
have Heq: (ConflMorphismIndex vu (getIndexerOfViewing f) g')
o>Generator (ConflProject vu (getIndexerOfViewing f))
= (ConflProject vu (g' o>Generator getIndexerOfViewing f) o>Generator g');
first (by rewrite (proj1 (ConflMorphismIndexCommuteProp _ _ _ )); reflexivity);
move: (ConflProp_MorphismIndexRelativeProject vu (getIndexerOfViewing f) g')
=> [CommonConflVertex' [CommonConfl1' [CommonConfl2' [HeqProject' HeqIndex']]]];
unshelve eapply Sense00_Viewing_quotient; simpl;
```

```
[
| exact CommonConfl1'
| exact CommonConfl2'
| assumption
| assumption
]).
Defined.

Definition Sense1_UnitViewedOb
U V (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)
(* V = pb vu G *)
(G : obGenerator)
(Sense1_ff: Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_F) :
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) (Sense01_ViewedOb Sense01_F vu).
Proof.
intros; unshelve eexists.
- intros H h. exact
{|
getProjectVertexOfViewed := ConflVertex vu (getIndexerOfViewing h);
getProjectOfViewed := ConflProject vu (getIndexerOfViewing h);
getDataOfViewed :=
ConflProject vu (getIndexerOfViewing h)
              o>Generator_[sval Sense01_F] sval Sense1_ff H h;
getConditionOfViewed := ConflIndex vu (getIndexerOfViewing h)
|}.
- abstract (move; simpl; intros H H' h' f;
move: (ConflProp_ComposIndex vu (getIndexerOfViewing f) h' ) =>
[CommonConflVertex [CommonConfl1 [CommonConfl2 [HeqProject HeqIndex]]]];
unshelve eapply Sense00_ViewedOb_quotient; simpl;
[
| exact CommonConfl1
| exact CommonConfl2
| assumption
|
];
do 3 rewrite [in LHS](proj2_sig Sense1_ff);
do 2 rewrite [in RHS](proj2_sig Sense1_ff);
congr (sval Sense1_ff _ _);
do 2 rewrite [in RHS](proj1 (proj2_sig ( Sense01_Viewing (Sense01_ViewOb G) vu))) ;
do 2 rewrite [in LHS](proj1 (proj2_sig ( Sense01_Viewing (Sense01_ViewOb G) vu))) ;
congr ( _ o>Generator_ _);
rewrite -[in RHS]HeqProject;
rewrite -[in LHS]polyGenerator_morphism;
rewrite -[in RHS]polyGenerator_morphism;
congr (CommonConfl1 o>Generator _);
rewrite ConflCommuteProp; reflexivity).
Defined.

Definition lem_Viewing_Refinement :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall W (wv : 'Generator( W ~> V(*nope, not pb*))),
{ lem: forall G (g_f : (Sense00_Viewing Sense01_F vu) G ),
(Sense00_Viewing Sense01_F wv) (ConflVertex vu (getIndexerOfViewing g_f)) |
forall G H (hg : 'Generator( H ~> G )) (g_f : (Sense00_Viewing Sense01_F vu) G ),
lem H (hg o>Generator_[sval (Sense01_Viewing Sense01_F vu)] g_f)  =
(ConflMorphismIndex vu (getIndexerOfViewing g_f) hg)
o>Generator_[sval (Sense01_Viewing Sense01_F wv)]
            lem G g_f }.
```

```
Proof.
intros. unshelve eexists.
- intros.  exact
{|
getIndexerOfViewing := ConflIndex vu (getIndexerOfViewing g_f);
getDataOfViewing :=
  ConflProject wv
               (ConflIndex vu (getIndexerOfViewing g_f))
               o>Generator_[sval Sense01_F] getDataOfViewing g_f
|}.
- abstract (intros; simpl;
move: (ConflProp_ComposRelativeIndex vu wv (getIndexerOfViewing g_f) hg )
=> [CommonConflVertex [CommonConfl1 [CommonConfl2 [HeqProject HeqIndex]]]];
unshelve eapply Sense00_Viewing_quotient; simpl;
[
| exact CommonConfl1
| exact CommonConfl2
| assumption
|
];
do 2 rewrite [in RHS](proj1 (proj2_sig ( Sense01_F)));
do 2 rewrite [in LHS](proj1 (proj2_sig ( Sense01_F)));
congr (_ o>Generator_ _);
rewrite -[in LHS]polyGenerator_morphism;
rewrite -[in RHS]polyGenerator_morphism;
exact HeqIndex).
Defined.

Definition Sense1_Refinement :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E),
forall W (wv : 'Generator( W ~> V(*nope, not pb*))),
forall (Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
Sense1_def (Sense01_Viewing Sense01_F vu) (Sense01_ViewedOb Sense01_E vu).
Proof.
intros. unshelve eexists.
- intros G g_f.
pose lem1 : (Sense00_ViewedOb Sense01_E wv) (ConflVertex vu (getIndexerOfViewing g_f)) :=
(sval Sense1_ee (ConflVertex vu (getIndexerOfViewing g_f))
      (proj1_sig (lem_Viewing_Refinement vu Sense01_F wv ) _ g_f)).
exact {|
  getProjectVertexOfViewed := getProjectVertexOfViewed lem1;
  getProjectOfViewed :=
    getProjectOfViewed lem1
                      o>Generator ConflProject vu (getIndexerOfViewing g_f);
  getDataOfViewed := getDataOfViewed lem1;
  getConditionOfViewed := getConditionOfViewed lem1 o>Generator wv
|}.
- abstract (move; intros G H hg g_f;
rewrite [in RHS](proj2_sig (lem_Viewing_Refinement _ _ _ ));
rewrite -[in RHS](proj2_sig Sense1_ee);
simpl;
set getProjectOfViewed_ee := (getProjectOfViewed (sval Sense1_ee _ _));
move: @getProjectOfViewed_ee;
set getDataOfViewed_ee := (getDataOfViewed (sval Sense1_ee _ _));
move: @getDataOfViewed_ee;
set getConditionOfViewed_ee := (getConditionOfViewed (sval Sense1_ee _ _));
move: @getConditionOfViewed_ee;
set getProjectVertexOfViewed_ee := (getProjectVertexOfViewed (sval Sense1_ee _ _));
```

```
set getIndexerOfViewing_g_f := (getIndexerOfViewing g_f);
move => getConditionOfViewed_ee getDataOfViewed_ee getProjectOfViewed_ee;

move: (@ConflProp_MixIndexProject_1 _ _ vu _ getIndexerOfViewing_g_f _ hg _
getProjectOfViewed_ee)
=> [CommonConflVertex [CommonConfl1 [CommonConfl2 [HeqProject HeqIndex]]]];
unshelve eapply Sense00_ViewedOb_quotient; simpl;
[
| exact CommonConfl1
| exact CommonConfl2
| exact HeqProject
|
];
do 1 rewrite [in RHS](proj1 (proj2_sig ( Sense01_E))) ;
do 1 rewrite [in LHS](proj1 (proj2_sig ( Sense01_E))) ;
congr ( _ o>Generator_ _);
exact HeqIndex).
Defined.

Definition Sense1_ViewedMor :
forall (U V : obGenerator) (vu : 'Generator( V ~> U ))
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F),
Sense1_def (Sense01_ViewedOb Sense01_E vu)
(Sense01_ViewedOb Sense01_F vu).
Proof.
intros. unshelve eexists.
- intros G e_. exact
{|
getProjectVertexOfViewed := getProjectVertexOfViewed e_;
getProjectOfViewed := getProjectOfViewed e_;
getDataOfViewed :=
sval Sense1_ff (getProjectVertexOfViewed e_) (getDataOfViewed e_);
getConditionOfViewed := getConditionOfViewed e_
|}.
- abstract (move; intros; unshelve eapply Sense00_ViewedOb_quotient; simpl;
[ | exact: unitGenerator
| exact: unitGenerator
| reflexivity
|  ];
congr (_ o>Generator_ _ );
rewrite (proj2_sig Sense1_ff); reflexivity).
Defined.

Definition Sense1_Unit:
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
Sense1_def Sense01_F Sense01_F.
Proof.
intros. exists (fun G => id).
abstract (intros; move; intros; reflexivity).
Defined.

Definition Morphism_Compos_morCode_Family :
forall U V (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(Sense1_ee__ : forall (G : obGenerator)  (form : Sense00_F G),
  Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__),
```

```
forall (Sense00_D : obGenerator -> Type) (Sense01_D : Sense01_def Sense00_D),
forall (Sense1_dd : Sense1_def Sense01_E Sense01_D),

Morphism_prop Sense01_F (fun (G : obGenerator)  (form : Sense00_F G) =>
                            Sense1_Compos Sense1_dd (Sense1_ee__ G form)).
Proof.
intros. move; simpl; intros.
congr (sval Sense1_dd _ _ ). exact: Sense1_ee_morphism.
Qed.

Inductive morCode
: forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E) ,
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
Sense1_def Sense01_E Sense01_F -> Type :=

| AtMember :
forall U V (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(Sense1_ee__ : forall (G : obGenerator)  (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__)
(Code_ee : morCode_Family Sense1_ee_morphism),

forall (G : obGenerator)  (form : Sense00_F G),
morCode (Sense1_ee__ G form)

| Compos_morCode :
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_F' : obGenerator -> Type) (Sense01_F' : Sense01_def Sense00_F')
(Sense1_ff' : Sense1_def Sense01_F' Sense01_F),
morCode Sense1_ff' ->
forall (Sense00_F'' : obGenerator -> Type) (Sense01_F'' : Sense01_def Sense00_F'')
(Sense1_ff_ : Sense1_def Sense01_F'' Sense01_F' ),
morCode Sense1_ff_ -> morCode ( Sense1_Compos Sense1_ff' Sense1_ff_ )

| Unit_morCode :
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
morCode ( Sense1_Unit Sense01_F )

| Destructing_morCode :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E),
forall (Sense1_ee__ : forall (G : obGenerator) (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__),
forall (Code_ee__ : morCode_Family Sense1_ee_morphism),
morCode (Sense1_Destructing Sense1_ee_morphism)

| Refinement_morCode :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E),
forall W (wv : 'Generator( W ~> V )),
forall (Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_ee : morCode Sense1_ee),
morCode (Sense1_Refinement vu Sense1_ee)

| UnitViewedOb_morCode :
```

```
forall U V (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)
(G : obGenerator)
(Sense1_ff: Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_F)
(Code_ff : morCode Sense1_ff) ,
morCode ( Sense1_UnitViewedOb Sense1_ff )

| ViewedMor_morCode :
forall (U V : obGenerator) (vu : 'Generator( V ~> U ))
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F)
(Code_ff : morCode Sense1_ff),
morCode (Sense1_ViewedMor vu Sense1_ff )

with morCode_Family :
forall U V (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(Sense1_ee__ : forall (G : obGenerator)  (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__), Type :=

| Constructing_morCode_Family :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),

morCode_Family (@Morphism_Constructing _ _ vu _ Sense01_F )

| Compos_morCode_Family :
forall U V (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(Sense1_ee__ : forall (G : obGenerator)  (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__),

forall (Sense00_D : obGenerator -> Type) (Sense01_D : Sense01_def Sense00_D),
forall (Sense1_dd : Sense1_def Sense01_E Sense01_D)
(Code_dd : morCode Sense1_dd),

morCode_Family Sense1_ee_morphism ->

morCode_Family (Morphism_Compos_morCode_Family Sense1_ee_morphism Sense1_dd).

Inductive obCoMod : forall Sense00_F (Sense01_F : Sense01_def Sense00_F), Type :=

| Viewing :
forall Sense00_F Sense01_F
(F: @obData Sense00_F Sense01_F)
U V (vu : 'Generator( V ~> U )),
@obCoMod (Sense00_Viewing Sense01_F vu) (Sense01_Viewing Sense01_F vu)

| ViewedOb :
forall Sense00_F (Sense01_F : Sense01_def Sense00_F)
(F: @obCoMod Sense00_F Sense01_F)
U V (vu : 'Generator( V ~> U )),
@obCoMod (Sense00_ViewedOb Sense01_F vu) (Sense01_ViewedOb Sense01_F vu)

with obData : forall Sense00_F (Sense01_F : Sense01_def Sense00_F), Type :=
```

```
(* | UnaryDataOb : obData Sense01_UnaryDataOb *)

| ViewOb : forall G : obGenerator, @obData (Sense00_ViewOb G) (Sense01_ViewOb G).

Inductive elConstruct :
forall (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F) ,
forall (G : obGenerator)  (form : Sense00_F G), Type :=

| ViewOb_elConstruct : forall G : obGenerator,
forall (G' : obGenerator) (g : 'Generator( G' ~> G )) ,
elConstruct (ViewOb G) g

(* with elConstruct_OneRecursiveArg _ : forall _, Type := *)

with elAlgebra :
forall (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F) ,
forall (G : obGenerator) (form : Sense00_F G), Type :=

| ElConstruct_elAlgebra :
forall (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F) ,

forall (G : obGenerator)  (form : Sense00_F G),
forall (cons_form : elConstruct F form),
elAlgebra F form

| Restrict_elAlgebra (*NOT in solution*):
forall (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F) ,

forall (G : obGenerator)  (form : Sense00_F G),
forall (cons_form : elAlgebra F form),

forall (G' : obGenerator) (g' : 'Generator( G' ~> G )),
elAlgebra F (g' o>Generator_[sval Sense01_F ] form )
(* | Zero : ... | Plus : ... *)  .

Module Inversion_elConstruct_obDataViewOb.
Inductive elConstruct GFixed : forall (G : obGenerator)
(form: Sense00_ViewOb GFixed G)
(cons_form: elConstruct (ViewOb GFixed) form), Type :=
| ViewOb_elConstruct :
forall (G' : obGenerator) (g : 'Generator( G' ~> GFixed )) ,
elConstruct (ViewOb_elConstruct g).
End Inversion_elConstruct_obDataViewOb.

Lemma elConstruct_obDataViewObP (GFixed : obGenerator) : forall (Sense00_F : obGenerator ->
Type)
(Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F) ,
forall (G : obGenerator)  (form : Sense00_F G) (cons_form: elConstruct F form),
ltac:(destruct F as [ GF]; [
destruct (eq_comparable GFixed GF);
[refine (Inversion_elConstruct_obDataViewOb.elConstruct cons_form)
```

```
| refine True]]).
Proof.
intros. destruct cons_form.
- intros eq. destruct eq as [Heq |].
+ apply: Inversion_elConstruct_obDataViewOb.ViewOb_elConstruct.
+ apply I.
Defined.

Inductive Solution_elConstruct : Type :=
with Solution_elAlgebra : Type :=
(* ELIMINATE
| Restrict_elAlgebra : *).

Section ElCongr_def.

Variables (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F) (F : obData
Sense01_F)
(G : obGenerator) (form : Sense00_F G) (cons_form : elAlgebra F form )
(form' : Sense00_F G) (cons_form' : elAlgebra F form' ).

Definition ElCongr_def : Type := form' = form.

End ElCongr_def.

Lemma ElCongr_Trans_convElAlgebra :
forall (Sense00_F : obGenerator -> Type) ,
forall (G : obGenerator) (form : Sense00_F G) ,
forall (form' : Sense00_F G),
ElCongr_def form form' ->
forall (form'' : Sense00_F G) ,
ElCongr_def form' form'' ->
ElCongr_def form form''.
Proof.
etransitivity; eassumption.
Qed.

Lemma ElCongr_Restrict_Restrict:
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(G : obGenerator) (form : Sense00_F G)
(G' : obGenerator) (g' : 'Generator( G' ~> G )) (G'0 : obGenerator)
(g'0 : 'Generator( G'0 ~> G' )),
ElCongr_def (g'0 o>Generator_[sval Sense01_F] (g' o>Generator_[sval Sense01_F] form))
((g'0 o>Generator g') o>Generator_[sval Sense01_F] form).
Proof.
intros. move. rewrite (proj1 (proj2_sig Sense01_F)). reflexivity.
Qed.

Lemma ElCongr_Restrict_ViewOb:
forall (G G' : obGenerator) (g : 'Generator( G' ~> G ))
(G'0 : obGenerator) (g'0 : 'Generator( G'0 ~> G' )),
ElCongr_def (g'0 o>Generator_[sval (Sense01_ViewOb G)] g) (g'0 o>Generator g).
Proof.
reflexivity.
Qed.

Reserved Notation "cons_f0  [ Congr_f_f0 ]<==  cons_f" (at level 10 ,  Congr_f_f0, cons_f at
level 40).

Inductive convElAlgebra :
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F) (F : obData
Sense01_F) ,
```

```
forall (G : obGenerator) (form : Sense00_F G) (cons_form : elAlgebra F form ),
forall (form' : Sense00_F G) (cons_form' : elAlgebra F form' ), ElCongr_def form form' -> Type
:=

| Trans_convElAlgebra :
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F) (F : obData
Sense01_F) ,
forall (G : obGenerator) (form : Sense00_F G) (cons_form : elAlgebra F form ),
forall (form' : Sense00_F G) (cons_form' : elAlgebra F form' ),
forall (Congr_form_form' : ElCongr_def form form' ),
cons_form'  [Congr_form_form' ]<== cons_form ->
forall (form'' : Sense00_F G) (cons_form'' : elAlgebra F form'' ),
forall (Congr_form'_form'' : ElCongr_def form' form'' ),
cons_form''  [Congr_form'_form'']<== cons_form' ->
cons_form''
[ElCongr_Trans_convElAlgebra Congr_form_form' Congr_form'_form'']<==
cons_form

| Restrict_Restrict (*NOT in solution*):
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F) (F : obData
Sense01_F) ,
forall (G : obGenerator)  (form : Sense00_F G),
forall (cons_form : elAlgebra F form),
forall (G' : obGenerator) (g' : 'Generator( G' ~> G )),
forall (G'0 : obGenerator) (g'0 : 'Generator( G'0 ~> G' )),

(Restrict_elAlgebra cons_form (g'0 o>Generator g'))
[ElCongr_Restrict_Restrict Sense01_F form g' g'0]<==
(Restrict_elAlgebra (Restrict_elAlgebra cons_form g') g'0)

| Restrict_ViewOb (*NOT in solution*):
forall (G : obGenerator), forall (G' : obGenerator) (g : 'Generator( G' ~> G )),
forall (G'0 : obGenerator) (g'0 : 'Generator( G'0 ~> G' )),

(ElConstruct_elAlgebra (ViewOb_elConstruct (g'0 o>Generator g)))
[ElCongr_Restrict_ViewOb g g'0]<==
(Restrict_elAlgebra (ElConstruct_elAlgebra (ViewOb_elConstruct g)) g'0)

where "cons_f0  [ Congr_f_f0 ]<==  cons_f" := (@convElAlgebra _ _ _ _ _ cons_f _ cons_f0
Congr_f_f0 ).

Section Congr_def.
Variables (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense1_ff : Sense1_def Sense01_E Sense01_F)
(Code_ff : morCode Sense1_ff)
(Sense1_ff' : Sense1_def Sense01_E Sense01_F)
(Code_ff' : morCode Sense1_ff').

Definition Congr_def : Type :=
forall (G' : obGenerator), forall form'  form'0 ,
forall Heq : form'0 =  form',
(sval Sense1_ff' G' form'0) =  (sval Sense1_ff G' form').

End Congr_def.

Lemma Congr_Trans:
forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense1_ff : Sense1_def Sense01_E Sense01_F)
(Sense1_ff' : Sense1_def Sense01_E Sense01_F),
```

```
forall (Congr_congr_ff : Congr_def Sense1_ff Sense1_ff' ),
forall (Sense1_ff'' : Sense1_def Sense01_E Sense01_F ),
forall (Congr_congr_ff' : Congr_def Sense1_ff' Sense1_ff''),
Congr_def Sense1_ff Sense1_ff''.
Proof.
intros. move; intros. subst.
etransitivity. apply: Congr_congr_ff'. reflexivity.
apply: Congr_congr_ff. reflexivity.
Qed.

Definition Congr_Constructing_comp_Destructing :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),

forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)

(Sense1_ee__ : forall (G : obGenerator) (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__),

forall (G : obGenerator) (form : Sense00_F G) ,

Congr_def (Sense1_Compos (Sense1_Destructing Sense1_ee_morphism)
(Sense1_Constructing_default vu Sense01_F form))
(Sense1_UnitViewedOb (Sense1_ee__ G form)).
Proof.
intros. move. intros H h h0 Heq; subst.
unshelve eapply Sense00_ViewedOb_quotient; simpl;
[
| exact unitGenerator
| exact unitGenerator
| subst; reflexivity
|
].
congr ( _ o>Generator_ _). subst.
etransitivity; first last.
apply Sense1_ee_morphism. reflexivity. simpl.
rewrite (proj2_sig (Sense1_ee__ _ _)). reflexivity.
Qed.

Definition Congr_UnitViewedOb_cong
U V (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)
(G : obGenerator)
(Sense1_ff: Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_F)
(Sense1_ff0: Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_F)
(Congr_ff: Congr_def Sense1_ff Sense1_ff0) :
Congr_def (Sense1_UnitViewedOb Sense1_ff) (Sense1_UnitViewedOb Sense1_ff0).
Proof.
intros. move. intros. subst.
unshelve eapply Sense00_ViewedOb_quotient; simpl;
[
| exact unitGenerator
| exact unitGenerator
| subst; reflexivity
|
].
congr(_ o>Generator_ _). congr(_ o>Generator_ _). apply: Congr_ff. reflexivity.
Qed.
```

```
Definition Congr_Constructing_comp_Refinement :
forall (U V : obGenerator) (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(W : obGenerator) (wv : 'Generator( W ~> V ))
(Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv)
(Sense01_ViewedOb Sense01_E wv))
(G : obGenerator) (form : Sense00_F G),
Congr_def
(Sense1_Compos (Sense1_Refinement vu Sense1_ee)
(Sense1_Constructing_default vu Sense01_F form))
(Sense1_Refinement vu
(Sense1_Compos Sense1_ee
(Sense1_Constructing_default wv Sense01_F form))).
Proof.
intros. move. intros H h h0 Heq. subst. simpl.
rewrite (proj1 (proj2_sig Sense01_F)).
unshelve eapply Sense00_ViewedOb_quotient; simpl;
[
| exact unitGenerator
| exact unitGenerator
| reflexivity
| reflexivity
].
Qed.

Definition Congr_Refinement_comp_ViewedMor:
forall (U V : obGenerator) (vu : 'Generator( V ~> U )) (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)  (Sense00_E : obGenerator -> Type)
(Sense01_E : Sense01_def Sense00_E) (W : obGenerator)
(wv : 'Generator( W ~> V ))
(Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv)
      (Sense01_ViewedOb Sense01_E wv)),
forall (Sense00_D : obGenerator -> Type) (Sense01_D : Sense01_def Sense00_D)
(Sense1_dd : Sense1_def Sense01_E Sense01_D),
Congr_def (Sense1_Compos (Sense1_ViewedMor vu Sense1_dd) (Sense1_Refinement vu Sense1_ee))
(Sense1_Refinement vu (Sense1_Compos (Sense1_ViewedMor wv Sense1_dd) Sense1_ee)).
Proof.
intros. move. intros H h h0 Heq. subst.
unshelve eapply Sense00_ViewedOb_quotient; simpl;
[ | exact unitGenerator
| exact unitGenerator
| reflexivity
| reflexivity].
Qed.

Lemma Congr_Constructing_cong:
forall (U V : obGenerator) (vu : 'Generator( V ~> U )) (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)  (G : obGenerator)
(form : Sense00_F G)  (form' : Sense00_F G)
(ElCong_form_form' : ElCongr_def form form'),

Congr_def (Sense1_Constructing_default vu Sense01_F form)
(Sense1_Constructing_default vu Sense01_F form').
Proof.
intros. move; intros. subst. rewrite ElCong_form_form'.
unshelve eapply Sense00_Viewing_quotient; simpl;
[ | exact unitGenerator
| exact unitGenerator
| reflexivity
| reflexivity ].
```

```
Qed.

Lemma congr_Compos_cong :
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_F' : obGenerator -> Type) (Sense01_F' : Sense01_def Sense00_F')
(Sense1_ff' : Sense1_def Sense01_F' Sense01_F) (Sense00_F'' : obGenerator -> Type)
(Sense01_F'' : Sense01_def Sense00_F'')
(Sense1_ff_ : Sense1_def Sense01_F'' Sense01_F')
(Sense1_ee' : Sense1_def Sense01_F' Sense01_F )
(Congr_congr_ff' : Congr_def Sense1_ff' Sense1_ee' ),
forall (Sense1_ee_ : Sense1_def Sense01_F'' Sense01_F' )
(Congr_congr_ff_ : Congr_def Sense1_ff_ Sense1_ee_ ),
Congr_def (Sense1_Compos Sense1_ff' Sense1_ff_) (Sense1_Compos Sense1_ee' Sense1_ee_).
Proof.
intros; move; intros; simpl.
apply: ( Congr_congr_ff').  apply: ( Congr_congr_ff_). assumption.
Qed.

Lemma Congr_Refl : forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def
Sense00_E),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F),
Congr_def Sense1_ff Sense1_ff.
Proof.
intros. move; intros. subst; reflexivity.
Qed.

Definition Congr_AtMember_Compos_morCode_Family :
forall (U V: obGenerator)
(vu: 'Generator( V ~> U ))
(Sense00_F: obGenerator -> Type)
(Sense00_E: obGenerator -> Type)
(Sense01_E: Sense01_def Sense00_E)
(Sense1_ee__: forall G : obGenerator,
Sense00_F G -> Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense00_D: obGenerator -> Type)
(Sense01_D: Sense01_def Sense00_D)
(Sense1_dd: Sense1_def Sense01_E Sense01_D)
(G: obGenerator)
(form: Sense00_F G),
Congr_def (Sense1_Compos Sense1_dd (Sense1_ee__ G form))
(Sense1_Compos Sense1_dd (Sense1_ee__ G form)).
Proof.
intros. move; intros; subst; reflexivity.
Qed.

Definition Congr_Destructing_comp_ViewedMor :
forall (U V: obGenerator)
(vu: 'Generator( V ~> U ))
(Sense00_F: obGenerator -> Type)
(Sense01_F: Sense01_def Sense00_F)
(Sense00_E: obGenerator -> Type)
(Sense01_E: Sense01_def Sense00_E)
(Sense1_ee__: forall G : obGenerator,
Sense00_F G -> Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism: Morphism_prop Sense01_F Sense1_ee__)
(Sense00_D: obGenerator -> Type)
(Sense01_D: Sense01_def Sense00_D)
(Sense1_dd: Sense1_def Sense01_E Sense01_D),
Congr_def (Sense1_Compos (Sense1_ViewedMor vu Sense1_dd) (Sense1_Destructing
Sense1_ee_morphism))
```

```
(Sense1_Destructing (Morphism_Compos_morCode_Family Sense1_ee_morphism Sense1_dd)).
Proof.
intros. move; simpl; intros; subst.
unshelve eapply Sense00_ViewedOb_quotient; simpl;
[
| exact unitGenerator
|  exact unitGenerator
|  reflexivity
|  reflexivity
].
Qed.

Lemma Congr_Refinement_cong :
forall (U V: obGenerator)
(vu: 'Generator( V ~> U ))
(Sense00_F: obGenerator -> Type)
(Sense01_F: Sense01_def Sense00_F)
(Sense00_E: obGenerator -> Type)
(Sense01_E: Sense01_def Sense00_E)
(W: obGenerator)
(wv: 'Generator( W ~> V ))
(Sense1_ee Sense1_dd: Sense1_def (Sense01_Viewing Sense01_F wv)
            (Sense01_ViewedOb Sense01_E wv))
(Congr_congr_eedd : Congr_def Sense1_ee Sense1_dd),
(Congr_def (Sense1_Refinement vu Sense1_ee)
        (Sense1_Refinement vu Sense1_dd)).
intros. move. intros; subst; simpl.
set sval_Sense1_dd_ := (sval Sense1_dd _ _).
set sval_Sense1_ee_ := (sval Sense1_ee _ _).
have -> : sval_Sense1_dd_ = sval_Sense1_ee_ by
 apply: Congr_congr_eedd.
 unshelve eapply Sense00_ViewedOb_quotient; simpl;
 [
 | exact unitGenerator
 |  exact unitGenerator
 |  reflexivity
 |  reflexivity
 ].
Qed.

Lemma Congr_Rev : forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def
Sense00_E),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F),
forall (Sense1_ff' : Sense1_def Sense01_E Sense01_F),
forall (Congr_congr_ff : Congr_def Sense1_ff Sense1_ff'),
Congr_def Sense1_ff' Sense1_ff.
Proof.
intros; move; intros; subst;  symmetry;  apply: Congr_congr_ff; reflexivity.
Qed.

Definition Congr_Constructing_comp_Constructing :
forall (U V : obGenerator) (vu : 'Generator( V ~> U ))
 (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
 (G : obGenerator) (form : Sense00_F G)
 (H : obGenerator)
 (h : Sense00_ViewOb G H),
Congr_def
(Sense1_Compos (Sense1_Constructing_default vu Sense01_F form)
(Sense1_Constructing_default vu (Sense01_ViewOb G) h))
(Sense1_Constructing_default vu Sense01_F (h o>Generator_[sval Sense01_F] form)).
```

```
Proof.
intros. move; intros;  subst;  simpl.
unshelve eapply Sense00_Viewing_quotient; simpl;
[
| exact unitGenerator
| exact unitGenerator
| reflexivity
| rewrite -(proj1 (proj2_sig Sense01_F)); reflexivity
].
Qed.

Reserved Notation "''CoMod$' (  Code_ff  ~>  Code_ff'  @_ Congr_congr_ff  )" (at level 0).
Inductive congrMorCode : forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def
Sense00_E),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F)
(Code_ff : morCode Sense1_ff),
forall (Sense1_ff' : Sense1_def Sense01_E Sense01_F)
(Code_ff' : morCode Sense1_ff'),
forall (Congr_congr_ff : Congr_def Sense1_ff Sense1_ff'), Type :=

| Trans_congrMorCode : forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def
Sense00_E),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F)
(Code_ff : morCode Sense1_ff),
forall (Sense1_ff' : Sense1_def Sense01_E Sense01_F)
(Code_ff' : morCode Sense1_ff'),
forall (Congr_congr_ff : Congr_def Sense1_ff Sense1_ff')
(congr_ff : 'CoMod$( Code_ff ~> Code_ff' @_ Congr_congr_ff )),
forall (Sense1_ff'' : Sense1_def Sense01_E Sense01_F )
(Code_ff'' : morCode Sense1_ff''),
forall (Congr_congr_ff' : Congr_def Sense1_ff' Sense1_ff'' )
(congr_ff' : 'CoMod$( Code_ff' ~> Code_ff'' @_ Congr_congr_ff' )),
'CoMod$( Code_ff ~> Code_ff'' @_ Congr_Trans Congr_congr_ff Congr_congr_ff'  )

| Refl_congrMorCode : forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def
Sense00_E),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F)
(Code_ff : morCode Sense1_ff),
'CoMod$( Code_ff ~> Code_ff @_ Congr_Refl Sense1_ff )

| Rev_congrMorCode : forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def
Sense00_E),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F)
(Code_ff : morCode Sense1_ff),
forall (Sense1_ff' : Sense1_def Sense01_E Sense01_F)
(Code_ff' : morCode Sense1_ff'),
forall (Congr_congr_ff : Congr_def Sense1_ff Sense1_ff')
(congr_ff : 'CoMod$( Code_ff ~> Code_ff' @_ Congr_congr_ff )),
'CoMod$( Code_ff' ~> Code_ff @_ Congr_Rev Congr_congr_ff )

| Constructing_comp_Destructing_congrMorCode :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F),

forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)

(Sense1_ee__ : forall (G : obGenerator) (form : Sense00_F G),
```

```
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__)
(Code_ee__ : morCode_Family Sense1_ee_morphism),

forall (G : obGenerator) (form : Sense00_F G) ,
'CoMod$( Compos_morCode (Destructing_morCode Code_ee__)
(AtMember (Constructing_morCode_Family vu Sense01_F)  form) ~>
(UnitViewedOb_morCode (AtMember Code_ee__ form)) @_
Congr_Constructing_comp_Destructing Sense1_ee_morphism form )

| UnitViewedOb_cong_congrMorCode
U V (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)
(G : obGenerator)
(Sense1_ff: Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_F)
(Code_ff : morCode Sense1_ff)
(Sense1_ff0: Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_F)
(Code_ff0 : morCode Sense1_ff0)
(Congr_ff: Congr_def Sense1_ff Sense1_ff0)
(congr_ff : 'CoMod$( Code_ff ~> Code_ff0 @_ Congr_ff )) :
'CoMod$( UnitViewedOb_morCode Code_ff ~>
  UnitViewedOb_morCode Code_ff0 @_
  Congr_UnitViewedOb_cong Congr_ff )

| Constructing_comp_Refinement_congrMorCode :
forall (U V : obGenerator) (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(W : obGenerator) (wv : 'Generator( W ~> V ))
(Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv)
                      (Sense01_ViewedOb Sense01_E wv))
(Code_ee : morCode Sense1_ee)
(G : obGenerator) (form : Sense00_F G),
'CoMod$ (Compos_morCode (Refinement_morCode vu Code_ee)
(AtMember (Constructing_morCode_Family vu Sense01_F) form) ~>
Refinement_morCode vu (Compos_morCode Code_ee
(AtMember (Constructing_morCode_Family wv Sense01_F) form))
@_ (Congr_Constructing_comp_Refinement Sense1_ee form))

| Refinement_comp_ViewedMor_congrMorCode:
forall (U V : obGenerator) (vu : 'Generator( V ~> U )) (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)  (Sense00_E : obGenerator -> Type)
(Sense01_E : Sense01_def Sense00_E) (W : obGenerator)
(wv : 'Generator( W ~> V ))
(Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv)
                  (Sense01_ViewedOb Sense01_E wv)) ,
forall (Code_ee : morCode Sense1_ee),
forall (Sense00_D : obGenerator -> Type) (Sense01_D : Sense01_def Sense00_D)
(Sense1_dd : Sense1_def Sense01_E Sense01_D)
(Code_dd : morCode Sense1_dd),
'CoMod$( Compos_morCode (ViewedMor_morCode vu Code_dd) (Refinement_morCode vu Code_ee) ~>
  Refinement_morCode vu (Compos_morCode (ViewedMor_morCode wv Code_dd) Code_ee)
  @_ Congr_Refinement_comp_ViewedMor Sense1_ee Sense1_dd )

| Constructing_cong_congrMorCode :
forall (U V : obGenerator) (vu : 'Generator( V ~> U )) (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F)  (G : obGenerator)
(form : Sense00_F G)  (form' : Sense00_F G)
(ElCong_form_form' : ElCongr_def form form'),
```

```
'CoMod$( AtMember (Constructing_morCode_Family vu Sense01_F) form ~>
      AtMember (Constructing_morCode_Family vu Sense01_F) form' @_
      (Congr_Constructing_cong Sense01_F ElCong_form_form' ))

| Compos_cong_congrMorCode :
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_F' : obGenerator -> Type) (Sense01_F' : Sense01_def Sense00_F')

(Sense1_ff' : Sense1_def Sense01_F' Sense01_F) (Code_ff' : morCode Sense1_ff')
(Sense00_F'' : obGenerator -> Type) (Sense01_F'' : Sense01_def Sense00_F'')

(Sense1_ff_ : Sense1_def Sense01_F'' Sense01_F' ) (Code_ff_ : morCode Sense1_ff_)
(Sense1_ee' : Sense1_def Sense01_F' Sense01_F ) (Code_ee' : morCode Sense1_ee')
(Congr_congr_ff' : Congr_def Sense1_ff' Sense1_ee' ),
'CoMod$( Code_ff' ~> Code_ee' @_ Congr_congr_ff' ) ->
forall (Sense1_ee_ : Sense1_def Sense01_F'' Sense01_F' ) (Code_ee_ : morCode Sense1_ee_)
(Congr_congr_ff_ : Congr_def Sense1_ff_ Sense1_ee_ ),
'CoMod$( Code_ff_ ~> Code_ee_ @_ Congr_congr_ff_ ) ->
'CoMod$( Compos_morCode Code_ff' Code_ff_ ~> Compos_morCode Code_ee'  Code_ee_ @_
                  congr_Compos_cong Congr_congr_ff' Congr_congr_ff_ )

| AtMember_Compos_morCode_Family_congrMorCode :
forall
(U V: obGenerator)
(vu: 'Generator( V ~> U ))
(Sense00_F: obGenerator -> Type)
(Sense01_F: Sense01_def Sense00_F)
(Sense00_E: obGenerator -> Type)
(Sense01_E: Sense01_def Sense00_E)
(Sense1_ee__: forall G : obGenerator,
Sense00_F G -> Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism: Morphism_prop Sense01_F Sense1_ee__)
(Code_ee__: morCode_Family Sense1_ee_morphism)
(Sense00_D: obGenerator -> Type)
(Sense01_D: Sense01_def Sense00_D)
(Sense1_dd: Sense1_def Sense01_E Sense01_D)
(Code_dd: morCode Sense1_dd)
(G: obGenerator)
(form: Sense00_F G),
'CoMod$( AtMember (Compos_morCode_Family Code_dd Code_ee__) form ~>
  Compos_morCode Code_dd (AtMember Code_ee__ form) @_
  (Congr_AtMember_Compos_morCode_Family Sense1_ee__ Sense1_dd form ))

| Destructing_comp_ViewedMor_congrMorCode :
forall (U V: obGenerator)
(vu: 'Generator( V ~> U ))
(Sense00_F: obGenerator -> Type)
(Sense01_F: Sense01_def Sense00_F)
(Sense00_E: obGenerator -> Type)
(Sense01_E: Sense01_def Sense00_E)
(Sense1_ee__: forall G : obGenerator,
  Sense00_F G -> Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism: Morphism_prop Sense01_F Sense1_ee__)
(Code_ee__: morCode_Family Sense1_ee_morphism)
(Sense00_D: obGenerator -> Type)
(Sense01_D: Sense01_def Sense00_D)
(Sense1_dd: Sense1_def Sense01_E Sense01_D)
(Code_dd: morCode Sense1_dd),
'CoMod$( Compos_morCode (ViewedMor_morCode vu Code_dd) (Destructing_morCode Code_ee__) ~>
  Destructing_morCode (Compos_morCode_Family Code_dd Code_ee__) @_
  Congr_Destructing_comp_ViewedMor Sense1_ee_morphism Sense1_dd )
```

```
| Refinement_cong_congrMorCode :
forall (U V : obGenerator) (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(Sense00_E : obGenerator -> Type)
(Sense01_E : Sense01_def Sense00_E)
(W : obGenerator) (wv : 'Generator( W ~> V ))
(Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv)
        (Sense01_ViewedOb Sense01_E wv))
(Code_ee : morCode Sense1_ee)
(Sense1_dd : Sense1_def (Sense01_Viewing Sense01_F wv)
        (Sense01_ViewedOb Sense01_E wv))
(Code_dd : morCode Sense1_dd)
(Congr_congr_eedd : Congr_def Sense1_ee Sense1_dd)
(congr_eedd : 'CoMod$( Code_ee ~> Code_dd @_ Congr_congr_eedd )),

'CoMod$( Refinement_morCode vu Code_ee ~>
Refinement_morCode vu Code_dd @_ Congr_Refinement_cong Congr_congr_eedd)

| Constructing_comp_Constructing_congrMorCode :
forall (U V : obGenerator) (vu : 'Generator( V ~> U ))
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F) (G : obGenerator) (form : Sense00_F G)
(cons_form : elAlgebra F form) (H : obGenerator)
(h : Sense00_ViewOb G H) (cons_h : elAlgebra (ViewOb G) h),
'CoMod$( Compos_morCode
    (AtMember (Constructing_morCode_Family vu Sense01_F) form)
    (AtMember (Constructing_morCode_Family vu (Sense01_ViewOb G)) h)
~> AtMember (Constructing_morCode_Family vu Sense01_F)
    (h o>Generator_ form) @_ Congr_Constructing_comp_Constructing Sense01_F form h  )

where "''CoMod$' (  Code_ff  ~>  Code_ff'  @_ Congr_congr_ff  )" :=
(@congrMorCode _ _ _ _ _ Code_ff _ Code_ff' Congr_congr_ff) : poly_scope.

Notation "congr_ff_ o>$ congr_ff'" :=
  (@Trans_congrMorCode _ _ _ _ _ _ _ _ _ congr_ff_ _ _ _ congr_ff')
  (at level 40 , congr_ff' at next level , left associativity) : poly_scope.
Arguments Refl_congrMorCode {_ _ _ _ _ _}.

Reserved Notation "''CoMod' (  E  ~>  F  @_ Code_ff  )" (at level 0).

Inductive morCoMod : forall Sense00_E Sense01_E
(E : @obCoMod Sense00_E Sense01_E ),
forall Sense00_F Sense01_F
(F : @obCoMod Sense00_F Sense01_F ),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F)
(Code_ff : morCode Sense1_ff ), Type :=

| Compos : forall Sense00_F Sense01_F
(F : @obCoMod Sense00_F Sense01_F ),
forall Sense00_F' Sense01_F'
(F' : @obCoMod Sense00_F' Sense01_F' ) Sense1_ff'
(Code_ff' : morCode Sense1_ff')
(ff' : 'CoMod( F' ~> F @_ Code_ff' )),

forall Sense00_F'' Sense01_F''
(F'' : @obCoMod Sense00_F'' Sense01_F''),
forall Sense1_ff_ (Code_ff_ : morCode Sense1_ff_)
(ff_ : 'CoMod( F'' ~> F' @_ Code_ff_ )),

'CoMod( F'' ~> F @_ (Compos_morCode Code_ff'  Code_ff_))
```

```
| Unit :
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F : @obCoMod Sense00_F Sense01_F ),
'CoMod( F ~> F @_ (Unit_morCode Sense01_F))

| Constructing :
forall U V (vu : 'Generator( V ~> U )), forall (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F) (F : obData Sense01_F),

forall (G : obGenerator) (form : Sense00_F G) (cons_form : elAlgebra F form ),

'CoMod( Viewing (ViewOb G) vu ~> Viewing F vu
  @_ (AtMember (Constructing_morCode_Family vu Sense01_F) form))

| Destructing :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F),

forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(E : @obCoMod Sense00_E Sense01_E)

(Sense1_ee__ : forall (G : obGenerator)  (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__)
(Code_ee__ : morCode_Family Sense1_ee_morphism)

(Sense1_ee0__ : forall (G : obGenerator)
          (form : Sense00_F G) (cons_form : elConstruct F form ),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Code_ee0__ : forall (G : obGenerator)
        (form : Sense00_F G) (cons_form : elConstruct F form),
morCode (Sense1_ee0__ G form cons_form))
(ee__ : forall (G : obGenerator)
  (form : Sense00_F G) (cons_form : elConstruct F form),
'CoMod( Viewing (ViewOb G) vu ~> E @_ (Code_ee0__ G form cons_form))),

forall (Congr_congr_ee__ : forall (G : obGenerator)
              (form : Sense00_F G) (cons_form : elConstruct F form),
Congr_def ((Sense1_ee__ G form)) (Sense1_ee0__ G form cons_form))
(congr_ee__ : forall (G : obGenerator)
        (form : Sense00_F G) (cons_form : elConstruct F form),
'CoMod$( (AtMember Code_ee__ form) ~> (Code_ee0__ G form cons_form)
                          @_ Congr_congr_ee__ G form cons_form )),

'CoMod( Viewing F vu ~> ViewedOb E vu @_ (Destructing_morCode Code_ee__))

| Refinement :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F: @obData Sense00_F Sense01_F),
forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(E: @obCoMod Sense00_E Sense01_E),
forall W (wv : 'Generator( W ~> V )),
forall (Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_ee : morCode Sense1_ee),
forall (Sense1_ee0 : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_ee0 : morCode Sense1_ee0),
```

```
forall (ee: 'CoMod( Viewing F wv ~> ViewedOb E wv @_ Code_ee0 )),
forall (Congr_congr_ee : Congr_def Sense1_ee Sense1_ee0)
(congr_ee : 'CoMod$( Code_ee ~> Code_ee0 @_ Congr_congr_ee )),

'CoMod( Viewing F vu ~> ViewedOb E vu @_ (Refinement_morCode vu Code_ee))

| UnitViewedOb :
forall U V (vu : 'Generator( V ~> U )),
forall Sense00_F (Sense01_F : Sense01_def Sense00_F)
(F: @obCoMod Sense00_F Sense01_F)
(G : obGenerator)
(Sense1_ff : Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu)  Sense01_F)
(Code_ff : morCode Sense1_ff) (ff : 'CoMod(  Viewing (ViewOb G) vu ~> F @_ Code_ff )),
'CoMod( Viewing (ViewOb G) vu ~> ViewedOb F vu @_ UnitViewedOb_morCode Code_ff )

| ViewedMor :
forall (U V : obGenerator) (vu : 'Generator( V ~> U ))
(Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(E: @obCoMod Sense00_E Sense01_E)
(Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F: @obCoMod Sense00_F Sense01_F),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F)
(Code_ff : morCode Sense1_ff)
(ff : 'CoMod( E ~> F @_ Code_ff )),
'CoMod( ViewedOb E vu ~> ViewedOb F vu @_ ViewedMor_morCode vu Code_ff   )

where "''CoMod' ( E ~> F @_ Code_ff )" := (@morCoMod _ _ E _ _ F _ Code_ff) : poly_scope.

Notation "ff_ o>CoMod ff'" := (@Compos _ _ _ _ _ _ _ _ ff' _ _ _ _ _ ff_ )
(at level 40, left associativity) : poly_scope.

Reserved Notation "ff0  [  congr_ff  ]<~~  ff" (at level 10 ,  congr_ff , ff at level 40).

Inductive convCoMod : forall Sense00_E Sense01_E (E : @obCoMod Sense00_E Sense01_E ),
forall Sense00_F Sense01_F (F : @obCoMod Sense00_F Sense01_F ),
forall (Sense1_ff : Sense1_def Sense01_E Sense01_F)
(Code_ff : morCode Sense1_ff ) (ff : 'CoMod( E ~> F @_ Code_ff )),
forall (Sense1_ff0 : Sense1_def Sense01_E Sense01_F)
(Code_ff0 : morCode Sense1_ff0 ) (ff0 : 'CoMod( E ~> F @_ Code_ff0 )),
forall (Congr_congr_ff : Congr_def Sense1_ff Sense1_ff0)
(congr_ff : 'CoMod$( Code_ff ~> Code_ff0 @_ Congr_congr_ff )), Prop :=

| Constructing_comp_Destructing :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F),

forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(E : @obCoMod Sense00_E Sense01_E)

(Sense1_ee__ : forall (G : obGenerator)  (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__)
(Code_ee__ : morCode_Family Sense1_ee_morphism)

(Sense1_ee0__ : forall (G : obGenerator)
        (form : Sense00_F G) (cons_form : elConstruct F form ),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Code_ee0__ : forall (G : obGenerator)
      (form : Sense00_F G) (cons_form : elConstruct F form),
morCode (Sense1_ee0__ G form cons_form))
```

```
(ee__ : forall (G : obGenerator)
(form : Sense00_F G) (cons_form : elConstruct F form),
'CoMod( Viewing (ViewOb G) vu ~> E @_ (Code_ee0__ G form cons_form))),

forall (Congr_congr_ee__ : forall (G : obGenerator)
            (form : Sense00_F G) (cons_form : elConstruct F form),
Congr_def ((Sense1_ee__ G form)) (Sense1_ee0__ G form cons_form))
(congr_ee__ : forall (G : obGenerator)
      (form : Sense00_F G) (cons_form : elConstruct F form),
'CoMod$( (AtMember Code_ee__ form) ~> (Code_ee0__ G form cons_form)
                          @_ Congr_congr_ee__ G form cons_form )),

forall (G : obGenerator) (form : Sense00_F G) (cons_form : elConstruct F form ),

(UnitViewedOb ( ee__ G form cons_form ))

[ (Constructing_comp_Destructing_congrMorCode Code_ee__ form)
o>$ (UnitViewedOb_cong_congrMorCode (congr_ee__ G form cons_form)) ]<~~

(( Constructing vu (ElConstruct_elAlgebra cons_form))
  o>CoMod ( Destructing ee__ congr_ee__ ))

| Destructing_comp_ViewedMor :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F),

forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(E : @obCoMod Sense00_E Sense01_E)

(Sense1_ee__ : forall (G : obGenerator)  (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__)
(Code_ee__ : morCode_Family Sense1_ee_morphism)

(Sense1_ee0__ : forall (G : obGenerator)
                (form : Sense00_F G) (cons_form : elConstruct F form ),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Code_ee0__ : forall (G : obGenerator)
              (form : Sense00_F G) (cons_form : elConstruct F form),
morCode (Sense1_ee0__ G form cons_form))
(ee__ : forall (G : obGenerator)
        (form : Sense00_F G) (cons_form : elConstruct F form),
'CoMod( Viewing (ViewOb G) vu ~> E @_ (Code_ee0__ G form cons_form))),

forall (Congr_congr_ee__ : forall (G : obGenerator)
          (form : Sense00_F G) (cons_form : elConstruct F form),
Congr_def ((Sense1_ee__ G form)) (Sense1_ee0__ G form cons_form))
(congr_ee__ : forall (G : obGenerator)
    (form : Sense00_F G) (cons_form : elConstruct F form),
'CoMod$( (AtMember Code_ee__ form) ~> (Code_ee0__ G form cons_form)
                        @_ Congr_congr_ee__ G form cons_form )),

forall (Sense00_D : obGenerator -> Type) (Sense01_D : Sense01_def Sense00_D)
(D: @obCoMod Sense00_D Sense01_D),
forall (Sense1_dd : Sense1_def Sense01_E Sense01_D)
(Code_dd : morCode Sense1_dd)
(dd : 'CoMod( E ~> D @_ Code_dd )),

( Destructing (fun (G : obGenerator) (form : Sense00_F G) (cons_form : elConstruct F form) =>
        ((ee__ G form cons_form) o>CoMod dd))
```

```
    (fun (G : obGenerator) (form : Sense00_F G) (cons_form : elConstruct F form) =>
        (AtMember_Compos_morCode_Family_congrMorCode Code_ee__ Code_dd form)
        o>$ Compos_cong_congrMorCode (Refl_congrMorCode) (congr_ee__ G form cons_form)))

[ Destructing_comp_ViewedMor_congrMorCode Code_ee__ Code_dd ]<~~

(  ( Destructing ee__ congr_ee__ ) o>CoMod ( ViewedMor vu dd ))
(*MEMO: The type of this term is a product while it is expected to be (morCode_Family
?Sense1_ee_morphism). *)

| Constructing_comp_Refinement :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
  (F: @obData Sense00_F Sense01_F),
forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
  (E: @obCoMod Sense00_E Sense01_E),
forall W (wv : 'Generator( W ~> V )),
forall (Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_ee : morCode Sense1_ee),
forall (Sense1_ee0 : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_ee0 : morCode Sense1_ee0),
forall (ee: 'CoMod( Viewing F wv ~> ViewedOb E wv @_ Code_ee0 )),
forall (Congr_congr_ee : Congr_def Sense1_ee Sense1_ee0)
(congr_ee : 'CoMod$( Code_ee ~> Code_ee0 @_ Congr_congr_ee )),

forall (G : obGenerator) (form : Sense00_F G) (cons_form : elAlgebra F form ),
(Refinement vu ((Constructing wv cons_form) o>CoMod ee )
  (Compos_cong_congrMorCode congr_ee (Refl_congrMorCode)))

[ Constructing_comp_Refinement_congrMorCode vu Code_ee form ]<~~

(( Constructing vu cons_form ) o>CoMod ( Refinement vu ee congr_ee))

| Refinement_comp_ViewedMor :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F: @obData Sense00_F Sense01_F),
forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(E: @obCoMod Sense00_E Sense01_E),
forall W (wv : 'Generator( W ~> V )),
forall (Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_ee : morCode Sense1_ee),
forall (Sense1_ee0 : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_ee0 : morCode Sense1_ee0),
forall (ee: 'CoMod( Viewing F wv ~> ViewedOb E wv @_ Code_ee0 )),
forall (Congr_congr_ee : Congr_def Sense1_ee Sense1_ee0)
(congr_ee : 'CoMod$( Code_ee ~> Code_ee0 @_ Congr_congr_ee )),

forall (Sense00_D : obGenerator -> Type) (Sense01_D : Sense01_def Sense00_D)
(D: @obCoMod Sense00_D Sense01_D),
forall (Sense1_dd : Sense1_def Sense01_E Sense01_D)
(Code_dd : morCode Sense1_dd)
(dd : 'CoMod( E ~> D @_ Code_dd )),

( Refinement vu ( ee o>CoMod ( ViewedMor wv dd ))
(Compos_cong_congrMorCode (Refl_congrMorCode) congr_ee ))
```

```
[ Refinement_comp_ViewedMor_congrMorCode vu Code_ee Code_dd ]<~~

(( Refinement vu ee congr_ee) o>CoMod ( ViewedMor vu dd ))

| Constructing_comp_Constructing :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F),
forall (G : obGenerator) (form : Sense00_F G) (cons_form : elAlgebra F form ),

forall (H : obGenerator) (h : (Sense00_ViewOb G) H) (cons_h : elAlgebra (ViewOb G) h),
( Constructing vu (Restrict_elAlgebra cons_form h))

[ Constructing_comp_Constructing_congrMorCode vu cons_form cons_h ]<~~

(( Constructing vu cons_h ) o>CoMod ( Constructing vu cons_form ))

| Compos_cong : forall Sense00_F Sense01_F
        (F : @obCoMod Sense00_F Sense01_F ),
forall Sense00_F' Sense01_F'
(F' : @obCoMod Sense00_F' Sense01_F' ) Sense1_ff'
(Code_ff' : morCode Sense1_ff')
(ff' : 'CoMod( F' ~> F @_ Code_ff' )),

forall Sense00_F'' Sense01_F''
(F'' : @obCoMod Sense00_F'' Sense01_F''),
forall Sense1_ff_ (Code_ff_ : morCode Sense1_ff_)
(ff_ : 'CoMod( F'' ~> F' @_ Code_ff_ )),

forall Sense1_ee'
(Code_ee' : morCode Sense1_ee')
(ee' : 'CoMod( F' ~> F @_Code_ee' )),
forall Congr_congr_ff' (congr_ff' : 'CoMod$( Code_ff' ~> Code_ee' @_ Congr_congr_ff' )),
ee' [ congr_ff' ]<~~ ff' ->

forall Sense1_ee_ (Code_ee_ : morCode Sense1_ee_)
(ee_ : 'CoMod( F'' ~> F' @_ Code_ee_ )),
forall Congr_congr_ff_ (congr_ff_ : 'CoMod$( Code_ff_ ~> Code_ee_ @_ Congr_congr_ff_ )),
ee_ [ congr_ff_ ]<~~ ff_ ->

( ee_ o>CoMod ee' )
[ Compos_cong_congrMorCode congr_ff'  congr_ff_ ]<~~
( ff_ o>CoMod ff' )

| Constructing_cong :
forall U V (vu : 'Generator( V ~> U )), forall (Sense00_F : obGenerator -> Type)
(Sense01_F : Sense01_def Sense00_F) (F : obData Sense01_F),

forall (G : obGenerator) (form : Sense00_F G) (cons_form : elAlgebra F form ),
forall (form' : Sense00_F G) (cons_form' : elAlgebra F form' )
(ElCong_form_form': ElCongr_def form form') ,
( cons_form'  [ ElCong_form_form' ]<==  cons_form )  ->

( Constructing vu cons_form' )
[ Constructing_cong_congrMorCode vu Sense01_F ElCong_form_form' ]<~~
( Constructing vu cons_form )

| Destructing_cong :
(*SIMPLE CONGRUENCE, possible is congruence
with different Code_dd__ and manual coherence conversions in 'CoMod$ *)
forall U V (vu : 'Generator( V ~> U )),
```

```
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F : obData Sense01_F),

forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(E : @obCoMod Sense00_E Sense01_E)

(Sense1_ee__ : forall (G : obGenerator)  (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Sense1_ee_morphism : Morphism_prop Sense01_F Sense1_ee__)
(Code_ee__ : morCode_Family Sense1_ee_morphism)

(Sense1_ee0__ : forall (G : obGenerator)
        (form : Sense00_F G) (cons_form : elConstruct F form ),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Code_ee0__ : forall (G : obGenerator)
      (form : Sense00_F G) (cons_form : elConstruct F form),
morCode (Sense1_ee0__ G form cons_form))
(ee__ : forall (G : obGenerator)
(form : Sense00_F G) (cons_form : elConstruct F form),
'CoMod( Viewing (ViewOb G) vu ~> E @_ (Code_ee0__ G form cons_form))),

forall (Congr_congr_ee__ : forall (G : obGenerator)
            (form : Sense00_F G) (cons_form : elConstruct F form),
Congr_def ((Sense1_ee__ G form)) (Sense1_ee0__ G form cons_form))
(congr_ee__ : forall (G : obGenerator)
      (form : Sense00_F G) (cons_form : elConstruct F form),
'CoMod$( (AtMember Code_ee__ form) ~> (Code_ee0__ G form cons_form)
                          @_ Congr_congr_ee__ G form cons_form )),
forall (Sense1_dd__ : forall (G : obGenerator)  (form : Sense00_F G),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E
:= Sense1_ee__)
(Sense1_dd_morphism : Morphism_prop Sense01_F Sense1_dd__
:= Sense1_ee_morphism)
(Code_dd__ : morCode_Family Sense1_dd_morphism
:= Code_ee__)

(Sense1_dd0__ : forall (G : obGenerator)
        (form : Sense00_F G) (cons_form : elConstruct F form ),
Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_E)
(Code_dd0__ : forall (G : obGenerator)
      (form : Sense00_F G) (cons_form : elConstruct F form),
morCode (Sense1_dd0__ G form cons_form))
(dd__ : forall (G : obGenerator)
(form : Sense00_F G) (cons_form : elConstruct F form),
'CoMod( Viewing (ViewOb G) vu ~> E @_ (Code_dd0__ G form cons_form))),

forall (Congr_congr_dd__ : forall (G : obGenerator)
            (form : Sense00_F G) (cons_form : elConstruct F form),
Congr_def ((Sense1_dd__ G form)) (Sense1_dd0__ G form cons_form))
(congr_dd__ : forall (G : obGenerator)
      (form : Sense00_F G) (cons_form : elConstruct F form),
'CoMod$( (AtMember Code_dd__ form) ~> (Code_dd0__ G form cons_form)
                          @_ Congr_congr_dd__ G form cons_form )),

forall (Congr_congr_eedd0__ : forall (G : obGenerator) (form : Sense00_F G) (cons_form :
elConstruct F form ),
Congr_def (Sense1_ee0__ G form cons_form) (Sense1_dd0__ G form cons_form))
(congr_eedd0__ : forall (G : obGenerator) (form : Sense00_F G) (cons_form : elConstruct F form
),
'CoMod$( (Code_ee0__ G form cons_form) ~>
  (Code_dd0__ G form cons_form) @_ (Congr_congr_eedd0__ G form cons_form))),
```

```
forall (conv_eedd0__ : forall (G : obGenerator) (form : Sense00_F G) (cons_form : elConstruct
F form ),
  (dd__ G form cons_form) [ (congr_eedd0__ G form cons_form) ]<~~ (ee__ G form cons_form)),

( Destructing dd__ (fun (G : obGenerator) (form : Sense00_F G) (cons_form : elConstruct F form
)
                => (congr_ee__ G form cons_form) o>$ (congr_eedd0__ G form cons_form)))

[ Refl_congrMorCode (*SIMPLE CONGRUENCE *) ]<~~

 ( Destructing ee__ congr_ee__ )

| Refinement_cong :
forall U V (vu : 'Generator( V ~> U )),
forall (Sense00_F : obGenerator -> Type) (Sense01_F : Sense01_def Sense00_F)
(F: @obData Sense00_F Sense01_F),
forall (Sense00_E : obGenerator -> Type) (Sense01_E : Sense01_def Sense00_E)
(E: @obCoMod Sense00_E Sense01_E),
forall W (wv : 'Generator( W ~> V )),
forall (Sense1_ee : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_ee : morCode Sense1_ee),
forall (Sense1_ee0 : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_ee0 : morCode Sense1_ee0),
forall (ee: 'CoMod( Viewing F wv ~> ViewedOb E wv @_ Code_ee0 )),
forall (Congr_congr_ee : Congr_def Sense1_ee Sense1_ee0)
(congr_ee : 'CoMod$( Code_ee ~> Code_ee0 @_ Congr_congr_ee )),

forall (Sense1_dd : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_dd : morCode Sense1_dd),
forall (Sense1_dd0 : Sense1_def (Sense01_Viewing Sense01_F wv) (Sense01_ViewedOb Sense01_E
wv)),
forall (Code_dd0 : morCode Sense1_dd0),
forall (dd: 'CoMod( Viewing F wv ~> ViewedOb E wv @_ Code_dd0 )),
forall (Congr_congr_dd : Congr_def Sense1_dd Sense1_dd0)
(congr_dd : 'CoMod$( Code_dd ~> Code_dd0 @_ Congr_congr_dd )),

forall (Congr_congr_eedd0 : Congr_def Sense1_ee0 Sense1_dd0)
(congr_eedd0 : 'CoMod$( Code_ee0 ~> Code_dd0 @_ Congr_congr_eedd0 )),

forall (conv_eedd0 : dd [ congr_eedd0 ]<~~ ee),

( Refinement vu dd congr_dd )
[ Refinement_cong_congrMorCode vu (congr_ee o>$ congr_eedd0 o>$ (Rev_congrMorCode congr_dd))
]<~~
( Refinement vu ee congr_ee )

| UnitViewedOb_cong :
forall U V (vu : 'Generator( V ~> U )),
forall Sense00_F (Sense01_F : Sense01_def Sense00_F) (F: @obCoMod Sense00_F Sense01_F) (G :
obGenerator)
(Sense1_ff : Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu)  Sense01_F)
(Code_ff : morCode Sense1_ff) (ff : 'CoMod(  Viewing (ViewOb G) vu ~> F @_ Code_ff )),
forall (Sense1_ff0: Sense1_def (Sense01_Viewing (Sense01_ViewOb G) vu) Sense01_F)
(Code_ff0 : morCode Sense1_ff0)
(ff0 : 'CoMod(  Viewing (ViewOb G) vu ~> F @_ Code_ff0 ))
(Congr_ff: Congr_def Sense1_ff Sense1_ff0)
(congr_ff : 'CoMod$( Code_ff ~> Code_ff0 @_ Congr_ff )),
```

```
  ff0 [ congr_ff ]<~~ ff ->
(  UnitViewedOb ff0 )
[ UnitViewedOb_cong_congrMorCode congr_ff ]<~~
( UnitViewedOb ff )

| convCoMod_Trans : forall Sense00_E Sense01_E
(E : @obCoMod Sense00_E Sense01_E),
forall Sense00_F Sense01_F
(F : @obCoMod Sense00_F Sense01_F),
forall Sense1_ff (Code_ff : morCode Sense1_ff) (ff : 'CoMod( E ~> F @_ Code_ff )),
forall Sense1_ff0 (Code_ff0 : morCode Sense1_ff0) (ff0 : 'CoMod( E ~> F @_ Code_ff0 )),
forall (Congr_congr_ff : Congr_def Sense1_ff Sense1_ff0 )
(congr_ff : 'CoMod$( Code_ff ~> Code_ff0 @_ Congr_congr_ff )),
ff0 [ congr_ff ]<~~ ff ->
forall Sense1_ff0' (Code_ff0' : morCode Sense1_ff0') (ff0' : 'CoMod( E ~> F @_ Code_ff0' )),
forall (Congr_congr_ff0 : Congr_def Sense1_ff0 Sense1_ff0')
(congr_ff0 : 'CoMod$( Code_ff0 ~> Code_ff0' @_ Congr_congr_ff0 )),

ff0' [ congr_ff0 ]<~~ ff0 ->
ff0' [ congr_ff o>$ congr_ff0 ]<~~ ff

| convCoMod_Refl : forall Sense00_E Sense01_E
(E : @obCoMod Sense00_E Sense01_E),
forall Sense00_F Sense01_F
(F : @obCoMod Sense00_F Sense01_F),
forall Sense1_ff (Code_ff : morCode Sense1_ff) (ff : 'CoMod( E ~> F @_ Code_ff )),

ff [ Refl_congrMorCode ]<~~ ff

where "ff0 [  congr_ff  ]<~~ ff" := (@convCoMod _ _ _ _ _ _ _ _ ff _ _ ff0 _ congr_ff).
(* forall (YKK2 : Congr_def _ _) (KK2 : 'CoMod$( _ ~> _ @_ YKK2 )),    *)

Global Hint Constructors convCoMod : core.

End COMOD.
(** # #
#+END_SRC

* Example

#+BEGIN_SRC coq :exports both :results silent # # **)
Module NatGenerator <: GENERATOR.

Definition obGenerator : eqType := nat_eqType.
Definition morGenerator : obGenerator -> obGenerator -> Type.
intros n m. exact (n <= m).
Defined.
Notation "''Generator' ( V ~> U )" := (@morGenerator V U)
(at level 0, format "''Generator' (  V  ~>  U  )") : poly_scope.
Definition polyGenerator :
forall U V, 'Generator( V ~> U ) -> forall W, 'Generator( W ~> V ) -> 'Generator( W ~> U ).
intros U V a W vu. eapply (leq_trans); eassumption.
Defined.

Notation "wv o>Generator vu" := (@polyGenerator _ _ vu _ wv)
(at level 40, vu at next level) : poly_scope.

Definition unitGenerator : forall {U : obGenerator}, 'Generator( U ~> U ) := leqnn.

Definition ConflVertex :
```

```
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )), obGenerator.
intros. exact (minn ProjecterVertex IndexerVertex).
Defined.
Definition ConflProject :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
  'Generator( ConflVertex projecter indexer ~> IndexerVertex ).
intros. exact: geq_minr.
Defined.
Definition ConflIndex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
  'Generator( ConflVertex projecter indexer ~> ProjecterVertex ).
intros. exact: geq_minl.
Defined.
Definition ConflCommuteProp :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
  ConflProject projecter indexer o>Generator indexer
  = ConflIndex projecter indexer o>Generator projecter.
intros. apply: bool_irrelevance.
Qed.

Definition ConflMorphismIndex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
  'Generator( ConflVertex projecter (preIndexer o>Generator indexer) ~>
                      ConflVertex projecter indexer ).
Proof.
unfold morGenerator. intros. rewrite leq_min. rewrite geq_minl.  simpl.
unfold ConflVertex. eapply leq_trans. exact: geq_minr. assumption.
Defined.

Definition ConflMorphismIndexCommuteProp :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
  ConflProject projecter (preIndexer o>Generator indexer) o>Generator preIndexer
  = ConflMorphismIndex projecter indexer preIndexer o>Generator ConflProject projecter indexer
  /\  ConflIndex projecter (preIndexer o>Generator indexer)
      = ConflMorphismIndex projecter indexer preIndexer o>Generator ConflIndex projecter
indexer.
intros. split; apply: bool_irrelevance.
Qed.

Parameter polyGenerator_morphism :
forall (U V : obGenerator) (vu : 'Generator( V ~> U ))
        (W : obGenerator) (wv : 'Generator( W ~> V )),
forall X (xw : 'Generator( X ~> W )),
  xw o>Generator ( wv o>Generator vu ) = ( xw o>Generator wv ) o>Generator vu.
Parameter polyGenerator_unitGenerator :
forall (U V : obGenerator) (vu : 'Generator( V ~> U )),
  vu = ((@unitGenerator V) o>Generator vu ).
Parameter unitGenerator_polyGenerator :
forall (U : obGenerator), forall (W : obGenerator) (wv : 'Generator( W ~> U )),
    wv = ( wv o>Generator (@unitGenerator U)).

Parameter ConflProp_ComposIndex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
```

```
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
{ CommonConflVertex : obGenerator &
{ CommonConfl1 : 'Generator( CommonConflVertex ~> (ConflVertex (ConflProject projecter
indexer) preIndexer )) &
{ CommonConfl2 : 'Generator( CommonConflVertex ~> (ConflVertex projecter (preIndexer
o>Generator indexer ))) |
  CommonConfl1 o>Generator (ConflProject (ConflProject projecter indexer) preIndexer )
  = CommonConfl2 o>Generator (ConflProject projecter (preIndexer o>Generator indexer ))
  /\  CommonConfl1 o>Generator ((ConflIndex (ConflProject projecter indexer) preIndexer ))
      = CommonConfl2 o>Generator (ConflMorphismIndex projecter indexer preIndexer )
} } }.

Parameter ConflProp_AssocIndex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
forall PrePreIndexerVertex (prePreIndexer : 'Generator( PrePreIndexerVertex ~>
PreIndexerVertex )),
{ CommonConflVertex : obGenerator &
{ CommonConfl1 : 'Generator( CommonConflVertex ~>
 (ConflVertex projecter (prePreIndexer o>Generator (preIndexer o>Generator indexer)))) &
{ CommonConfl2 : 'Generator( CommonConflVertex ~>
 (ConflVertex projecter ((prePreIndexer o>Generator preIndexer) o>Generator indexer))) |
  CommonConfl1 o>Generator (ConflProject projecter (prePreIndexer o>Generator (preIndexer
o>Generator indexer)))
  = CommonConfl2 o>Generator (ConflProject projecter ((prePreIndexer o>Generator preIndexer)
o>Generator indexer))
  /\  CommonConfl1 o>Generator ((ConflMorphismIndex projecter (preIndexer o>Generator indexer)
prePreIndexer)
                                  o>Generator (ConflMorphismIndex projecter indexer
preIndexer))
      = CommonConfl2 o>Generator (ConflMorphismIndex projecter indexer (prePreIndexer
o>Generator preIndexer))
} } }.

Parameter ConflProp_MorphismIndexRelativeProject :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
{ CommonConflVertex : obGenerator &
{ CommonConfl1 : 'Generator( CommonConflVertex ~> ConflVertex projecter
                    (ConflMorphismIndex projecter (indexer) preIndexer
                    o>Generator (ConflProject projecter (indexer)
                                  o>Generator indexer))) &
{ CommonConfl2 : 'Generator( CommonConflVertex ~> ConflVertex projecter
              (ConflProject projecter (preIndexer o>Generator indexer)
              o>Generator (preIndexer o>Generator indexer))) |
CommonConfl1 o>Generator ConflProject projecter (ConflMorphismIndex projecter (indexer)
preIndexer
o>Generator (ConflProject projecter (indexer) o>Generator indexer))
= CommonConfl2 o>Generator ConflProject projecter
(ConflProject projecter (preIndexer o>Generator indexer) o>Generator (preIndexer o>Generator
indexer))
/\  CommonConfl1 o>Generator (ConflMorphismIndex projecter (ConflProject projecter (indexer)
o>Generator indexer)
(ConflMorphismIndex projecter (indexer) preIndexer)
  o>Generator ConflMorphismIndex projecter (indexer) (ConflProject projecter (indexer)))
= CommonConfl2 o>Generator (ConflMorphismIndex projecter (preIndexer o>Generator indexer)
                              (ConflProject projecter (preIndexer o>Generator indexer))
        o>Generator ConflMorphismIndex projecter (indexer) preIndexer)
```

```
} } }.

Parameter ConflProp_ComposRelativeIndex :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall PreProjecterVertex (preProjecter : 'Generator( PreProjecterVertex ~> ProjecterVertex
)),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
{ CommonConflVertex : obGenerator &
{ CommonConfl1 : 'Generator( CommonConflVertex ~>
 ConflVertex preProjecter (ConflIndex projecter (preIndexer o>Generator indexer))) &
{ CommonConfl2 : 'Generator( CommonConflVertex ~> ConflVertex preProjecter
                      (ConflMorphismIndex projecter indexer preIndexer
                       o>Generator ConflIndex projecter indexer)) |
CommonConfl1 o>Generator ConflProject preProjecter (ConflIndex projecter (preIndexer
o>Generator indexer))
= CommonConfl2 o>Generator ConflProject preProjecter (ConflMorphismIndex projecter indexer
preIndexer
                                                          o>Generator ConflIndex projecter
indexer)
/\  CommonConfl1 o>Generator (ConflProject preProjecter (ConflIndex projecter (preIndexer
o>Generator indexer))
o>Generator ConflMorphismIndex projecter indexer preIndexer)
= CommonConfl2 o>Generator (ConflMorphismIndex preProjecter (ConflIndex projecter indexer)
  (ConflMorphismIndex projecter indexer preIndexer)
o>Generator ConflProject preProjecter (ConflIndex projecter indexer))
} } }.

Parameter ConflProp_MixIndexProject_1 :
forall BaseVertex ProjecterVertex (projecter : 'Generator( ProjecterVertex ~> BaseVertex )),
forall IndexerVertex (indexer : 'Generator( IndexerVertex ~> BaseVertex )),
forall PreIndexerVertex (preIndexer : 'Generator( PreIndexerVertex ~> IndexerVertex )),
forall PreProjecterVertex (preProjecter : 'Generator( PreProjecterVertex ~> ConflVertex
projecter indexer )),
{ CommonConflVertex : obGenerator &
{ CommonConfl1 : 'Generator( CommonConflVertex ~>
 ConflVertex (preProjecter o>Generator ConflProject projecter indexer) preIndexer ) &
{ CommonConfl2 : 'Generator( CommonConflVertex ~>
 ConflVertex preProjecter (ConflMorphismIndex projecter indexer preIndexer)) |
  CommonConfl1 o>Generator ConflProject (preProjecter o>Generator ConflProject projecter
indexer) preIndexer
  = CommonConfl2 o>Generator (ConflProject preProjecter (ConflMorphismIndex projecter indexer
preIndexer)
                                          o>Generator ConflProject projecter (preIndexer
o>Generator indexer))
  /\  CommonConfl1 o>Generator (ConflIndex (preProjecter o>Generator ConflProject projecter
indexer) preIndexer)
      = CommonConfl2 o>Generator (ConflIndex preProjecter (ConflMorphismIndex projecter
indexer preIndexer))
} } }.
End NatGenerator.

Import NatGenerator.
Declare Module Import CoMod : (COMOD NatGenerator).

Parameter (GFixed : obGenerator).

Definition example_morphism :
{ Sense1_ff : Sense1_def _ _ &
{ Code_ff : morCode Sense1_ff &
'CoMod( Viewing (ViewOb GFixed) (eq_refl _ : 2 <= 3) ~>
```

```
 ViewedOb (Viewing (ViewOb GFixed) (eq_refl _ : 0 <= 0)) (eq_refl _ : 2 <= 3) @_ Code_ff ) }}.
Proof.
repeat eexists.
eapply Refinement with (vu := (eq_refl _ : 2 <= 3)) (2 := Refl_congrMorCode).
eapply Refinement with (vu := (eq_refl _ : 1 <= 2)) (2 := Refl_congrMorCode).
eapply Refinement with (vu := (eq_refl _ : 0 <= 1)) (2 := Refl_congrMorCode).
eapply Destructing with (vu := (eq_refl _ : 0 <= 0)).
intros. eapply Compos.
- apply Constructing, ElConstruct_elAlgebra, (ViewOb_elConstruct unitGenerator).
- move: (elConstruct_obDataViewObP GFixed cons_form).
elim (eq_comparable GFixed GFixed) => [ /=  ?  cons_form_P | // ].
destruct cons_form_P.
apply Constructing, ElConstruct_elAlgebra, (ViewOb_elConstruct g).

Unshelve. all: intros; try apply Congr_AtMember_Compos_morCode_Family;
try apply AtMember_Compos_morCode_Family_congrMorCode.
Defined.

Definition example_reduction:
{ Sense1_ff : Sense1_def _ _ &
{ Code_ff : morCode Sense1_ff &
{ ff : 'CoMod( _ ~> _ @_ Code_ff ) &
{ Congr_congr_ff : Congr_def _ _ &
{ congr_ff : 'CoMod$( _ ~> _ @_ Congr_congr_ff ) &
( ff )  [ congr_ff ]<~~
 ((Constructing (eq_refl _ : 2 <= 3) (ElConstruct_elAlgebra (ViewOb_elConstruct
unitGenerator)))
   o>CoMod (projT2 (projT2 example_morphism)))
}}}}}.
Proof.
repeat eexists.  simpl.
eapply convCoMod_Trans.
eapply Constructing_comp_Refinement.
eapply convCoMod_Trans.
eapply Refinement_cong, Constructing_comp_Refinement.
eapply convCoMod_Trans.
eapply Refinement_cong, Refinement_cong, Constructing_comp_Refinement.
eapply convCoMod_Trans.
eapply Refinement_cong, Refinement_cong, Refinement_cong, Constructing_comp_Destructing.
simpl. destruct (eq_comparable GFixed GFixed); last by []; simpl.
eapply convCoMod_Trans.
eapply Refinement_cong, Refinement_cong, Refinement_cong, UnitViewedOb_cong,
Constructing_comp_Constructing.
exact: convCoMod_Refl.
Unshelve. all: try apply Refl_congrMorCode.
Defined.
Eval simpl in (projT1 (projT2 (projT2 example_reduction))).
(*
= Refinement (eqxx (2 - 3))
    (Refinement (eqxx (1 - 2))
      (Refinement (eqxx (0 - 1))
          (UnitViewedOb
            (Constructing (eqxx (0 - 0))
                (Restrict_elAlgebra
                  (ElConstruct_elAlgebra
                      (ViewOb_elConstruct unitGenerator)) unitGenerator)))
          Refl_congrMorCode) Refl_congrMorCode) Refl_congrMorCode
: 'CoMod( Viewing (ViewOb GFixed) (eqxx (2 - 3) : 1 < 3) ~>
  ViewedOb (Viewing (ViewOb GFixed) (eqxx (0 - 0) : 0 <= 0))
    (eqxx (2 - 3) : 1 < 3) @_ projT1 (projT2 example_reduction)) *)
End SHEAF.
```

```
(** # #
#+END_SRC

Voila.
# # **)
```

S1 / coq

Voila.