**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:

1. **WorkSchool 365 CRM & LMS for applications in e-commerce and e-learning (SurveyQuiz transcripts, EventReview receipts)**
2. **WorkSchool 365 for research applications (MODOS proof-assistant)**
   1. **MODOS proof-assistant possible applications in the computational logic for geometric algorithmics and quantum-fields physics**
3. **Appendix: What is the minimal example of sheaf cohomology? Grammatically**

*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.

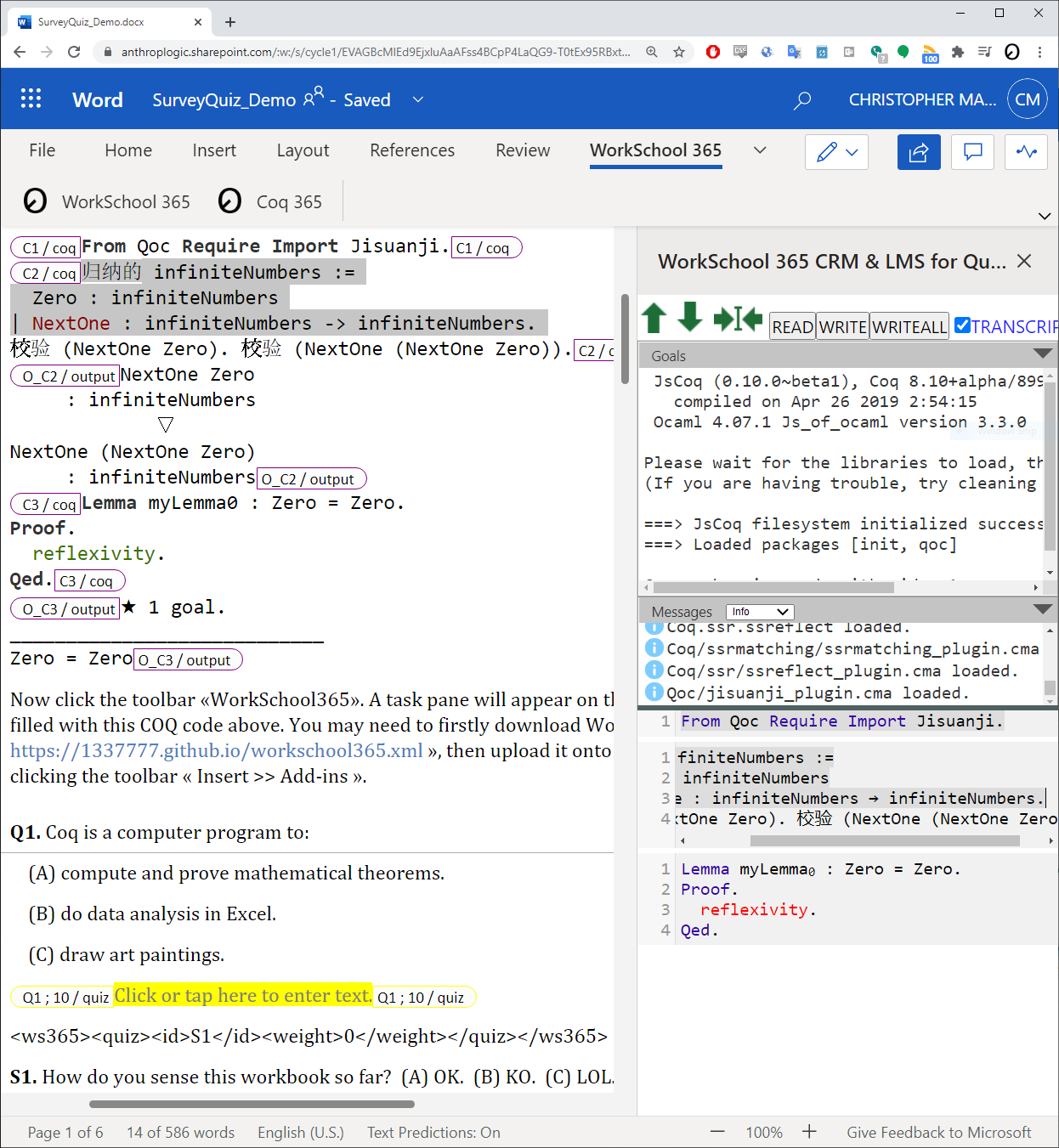
(2.) WorkSchool 365 is ***Open Source Code Secure Microsoft 365 SharePoint Teams application with PayPal + Stripe Payments***. WorkSchool 365 integrates the ***Customer Relationship Management (CRM) + Learning Management System (LMS)*** for your Business School to engage/qualify/educate prospective users 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](http://coq.inria.fr/)

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

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

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

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

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

In contrast to GAP/SINGULAR which does the inner computational-algebra corresponding to the affine-projective aspects of geometry, the MODOS aims at the outer logical/categorial-algebra corresponding to the parameterized-schematic aspects of geometry; this contrast is similar as the OCAML-COQ contrast. In short: MODOS does the computational-logic of the coherent sheaf modules over some base scheme; dually the relative support/spectrum of such sheaf modules/algebras are schemes parameterized over this base scheme (alternatively, the slice topos over this sheaf is étale over the base topos). Links: [https://stacks.math.columbia.edu/tag/01LQ](https://stacks.math.columbia.edu/tag/01L%20Q)

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

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

## **2.1. 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***.

(1 ∘ f) ∘ 1

(Sheaf F) ((1 ∘ f) ∘ 1)

)

1 ∘ f

f

(Sheaf F) (f ∘ 1)

f ∘ 1

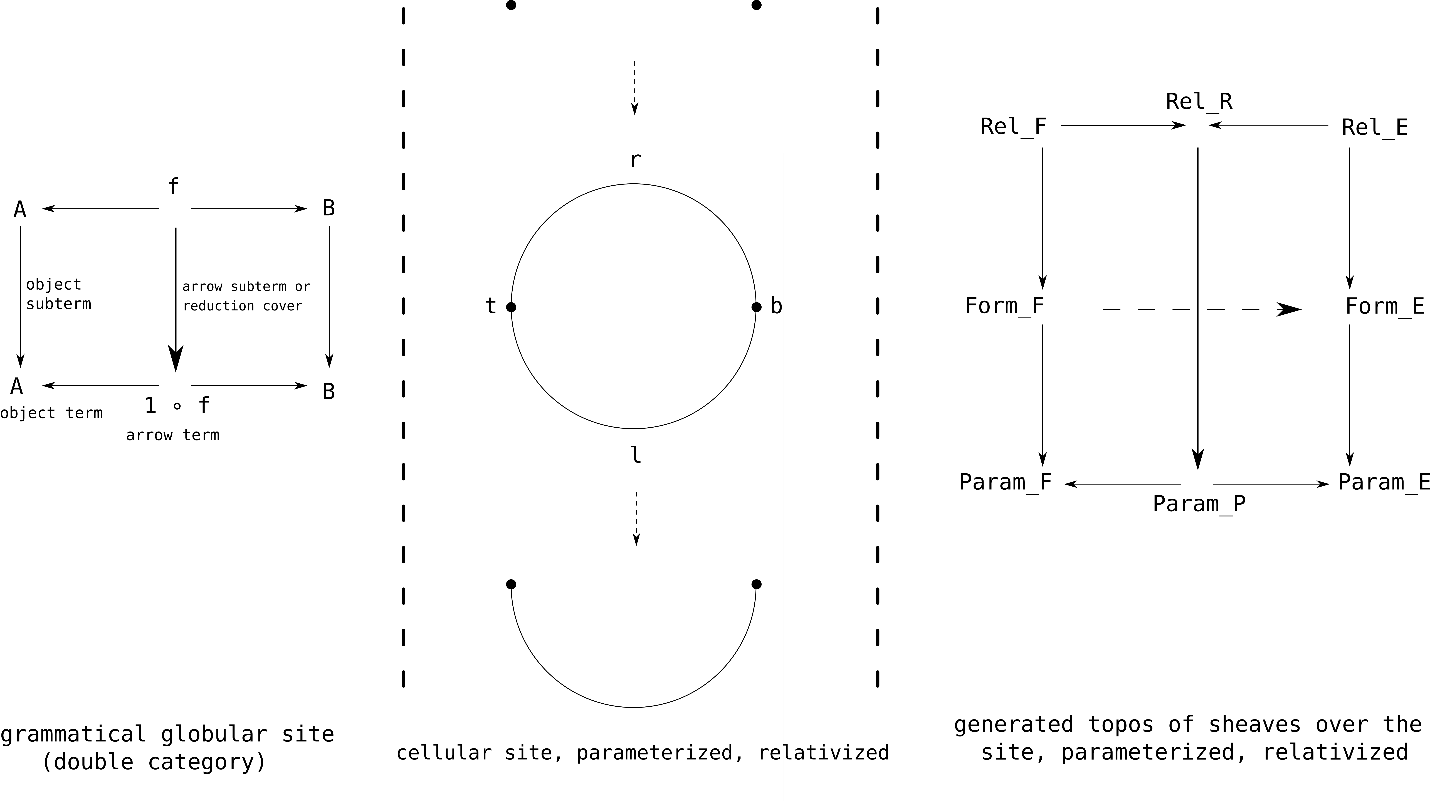
F (f)

element

of shape E

defined by cases over

the constructors of E

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:

**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 ).

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.

j∘i

i

j

f

g

g∘f

f\*i

(f\*i)\_\*j

g\*(f\*i)^

(g∘f)\*(j∘i) :=

(g\*(f\*i)^)\_\*((f\*i)\_\*j)^

constructive-confluence

should be defined

by induction with

universal arrow

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*** ).

dataobjects

= finitely-presentable

sheaves over generators of sets

modos with

dependent products Π,

= sheaves over generators of

diagrams over dataobjects

### The codes for the morphisms:

**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** := **\_**.

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 :

**Inductive** obCoMod : **forall** Sense00\_F (Sense01\_F : Sense01\_def Sense00\_F), **Type** := **\_**

with obData : **forall** Sense00\_F (Sense01\_F : Sense01\_def Sense00\_F), **Type** := **\_**.

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.

G

U

V\_

g\*V\_

g

data f : F(g\*V\_) over

only local-pieces of G

( Viewing F V\_ ) G := Sum (g : G → U) × F(g\*V\_)

sieve V\_ at U

pullback sieve g\*V\_

along g

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”.

G

U

V\_

A\_

elements e : E(A\_) of any E

over local-pieces of G

( Viewed E V\_ ) G := Sum (A\_ sieve at G) × E(A\_) | A\_ contained in V\_

sieve V\_ at U

any sieve A\_ at G

### The constructor elements and algebraic elements of the dataobjects :

**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** := **\_**.

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:

**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" := **\_**.

### The codes for the conversions on the morphisms:

**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 )" := **\_**.

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:

**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 )" := **\_**.

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:

**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" := **\_**.

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.

**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)) \*)

# 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***.

(\*\* # #

#+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.

# # \*\*)

Voila.