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

Short: AnthropLOGIC.com WorkSchool 365 is some legal business-university for e-commerce and e-learning with published applications in the Microsoft Commercial Marketplace: ***SurveyQuiz*** transcripts, ***EventReview*** receipts, ***MODOS*** proof-assistant for geometric algorithmics and quantum physics.

The *SurveyQuiz* and *EventReview* e-commerce and e-learning 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 SharePoint databases/calendars of paid/remunerated *review-tasks* for Word documents/events (seminars, conferences, archive papers, journal papers) with ***shareable receipts*** of Work by the reviewers, whose reviews are appended to the task.

The *MODOS* e-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, gluing).

Outline:

1. WorkSchool 365 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 the 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 (learner-reviewer) without requiring excessive financial information (beyond some payout address) and without relying on indirect government/public currencies.

(2.) WorkSchool 365 integrates the ***Customer Relationship Management (CRM) + Learning Management System (LMS)*** for your Business or University to engage/qualify/educate prospective users into paying/subscribed/grantee learners/customers or paid/remunerated reviewers/merchants via an integration of Stripe.com e-commerce payment (Card, Alipay, WeChat Pay, PayPal) + Microsoft.com Business Applications MBA (Azure AD, SharePoint Teams, 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 Coq 365 proof-assistant Word add-in and samples from the Gentle Introduction to the Art of Mathematics textbook. Demo instance: http://giam.southernct.edu ; https://SurveyQuiz.WorkSchool365.com

(2.2) WorkSchool 365 EventReview are 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. Demo instance: https://EventReview.WorkSchool365.com

(2.3) WorkSchool 365 UserGraduation are ***no-password*** sign-in/sign-up 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). Demo instance: https://WorkSchool365.com

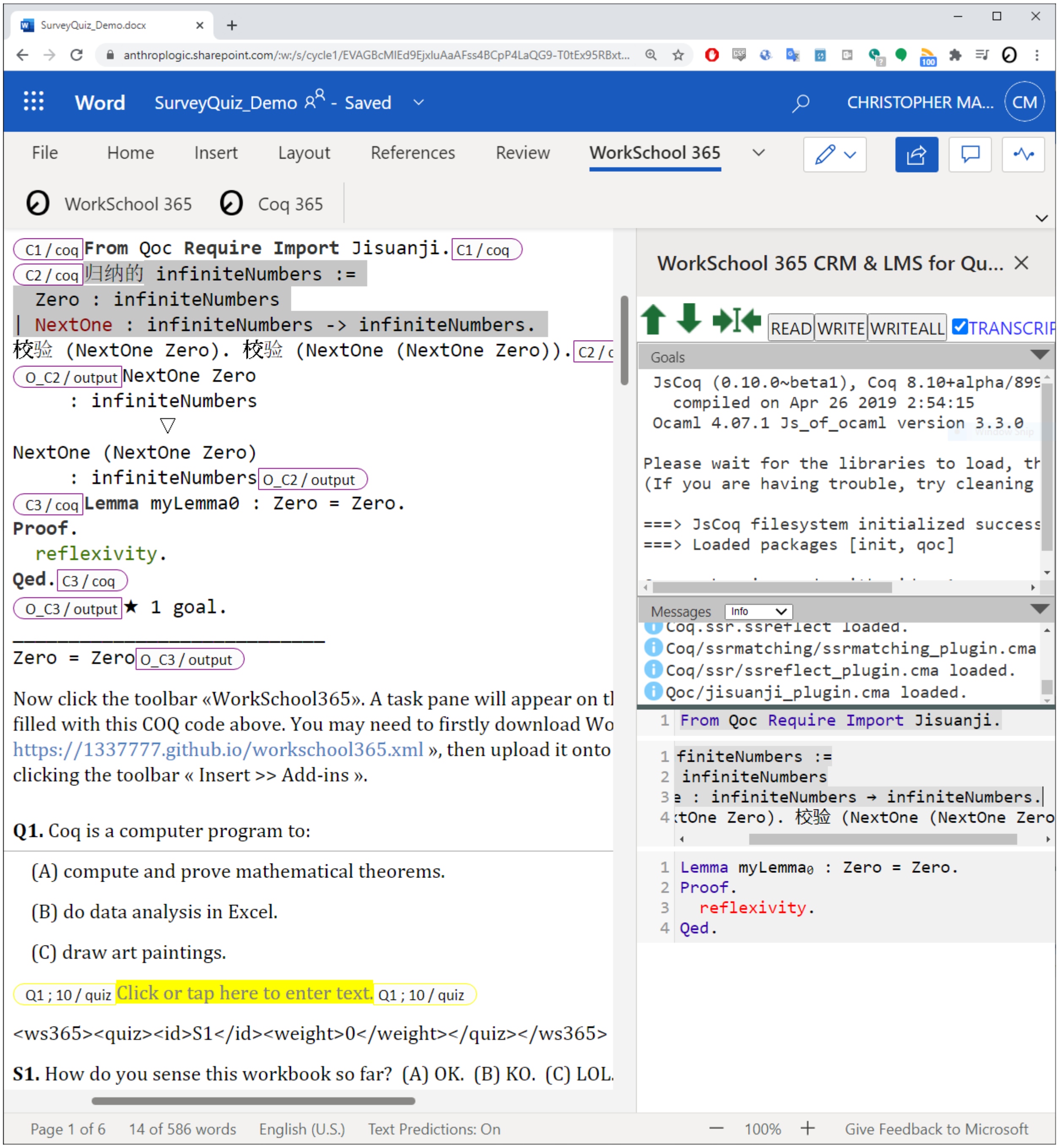


Figure: SurveyQuiz Word document code-ranges being parsed and selected by the Coq add-in in the web browser, with the quiz-ranges responses saved for later automatic grading.

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

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

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

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

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

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

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

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

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

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

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

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

2.1. MODOS proof-assistant possible applications in the computational logic for geometric algorithmics and quantum-fields physics

(1.) What problem is to be solved? In algorithmics, the usual constructive-inductive datatypes generalize the natural-numbers induction to allow structural constructors of the datatype to form expression-trees, but fails to articulate all the possible geometries in the new datatypes. In physics, Quantum Fields is an attempt to upgrade the mathematics of the 19th century's Maxwell equations of electromagnetism, in particular to clarify the duality between matter particles and light waves. However, those differential geometry methods (even post-Sardanashvily) are still "equational algebra" and fail to upgrade the computational-logic content.

(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

(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 geometrically constructored dataobjects gives the construction scheme for the sheafification-restricted below any sieve...

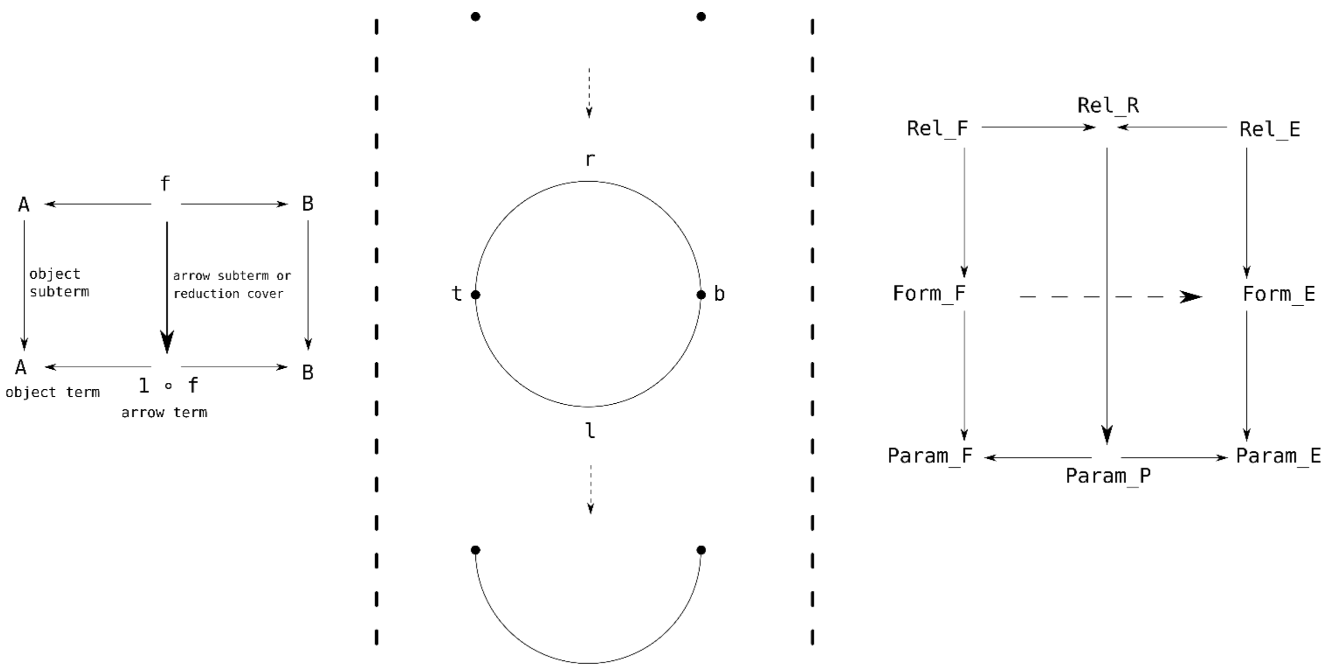
```
| Base_clause_nonRecursiveSignature: for (VV : Sieve), and for the family

(forall (U : open) (f : F U), isConstructor VV f -> Hom( U  ~>  E )), then

Hom( Restrict_VV F  ~>  SheafiRestricted_VV E )

| Refinement_clause_nonRecursiveSignature:

for (VV : Sieve) (WW : Sieve), (WW contained in VV) -> and for the family

(forall (U : open) (f : F U), asConstructor VV f ->

Hom( Restrict_WW U  ~>  SheafiRestricted_WW E )), then

Hom( Restrict_VV F  ~>  SheafiRestricted_VV E ) .

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

1. The generating site of arrow-terms with reduction-links

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

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

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



grammatical globular site (double category)

cellular site, parameterized, relativized

generated topos of sheaves over the site, parameterized, relativized

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

C1 / coq
```
Parameter Parameter0 : obGenerator -> obParametrizator.
Parameter Parameter1 :  forall A A' : obGenerator,
'Generator(A~>A') -> 'Parametrizator(Parameter0 A ~> Parameter0 A').
```
C1 / coq

0. 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 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

2. The data-objects, mutually-inductive with the pseudo codes for the morphisms

First, the codes for the generating data objects are collected; for example, the codes of the binary true-false data object are:

C2 / coq
```
Inductive obData_Param : forall _, Type :=
| Binary_Param : obData_Param _ .

Inductive obData : forall _, Type :=
| Binary_Form : obData _.
```
C2 / coq

Now the grammatical entry for the data-objects is mutually recursive with the grammatical entry for the pseudo codes of the morphisms. Here the clause [ViewOb] is the Yoneda embedding of the generators from the site to the topos. And the clause [Viewing] takes from any data-object together with some viewing (sieve) to produce the general logic-objects. And the clause [ViewedOb] is the sheafification operation. And the clause [FormatOb] is polymorph context-extension which takes any form object to produce some parameter object.

C3 / coq
```
Inductive obCoMod_Param : forall _, Type :=
| ViewOb_Param :
  forall P : obParametrizator,
obCoMod_Param _

| Viewing_Param_default :
  forall (Param_SubstF : obCoMod_Param _)
  (* (VV : forall G, Sieve) *),
obCoMod_Param _
```

```
| ViewedOb_Param_default :
  forall (Param_F : obCoMod_Param _)
(* (VV : forall G, Sieve) *),
obCoMod_Param _

| FormatOb :
  forall (F: obCoMod _) (Param_F : obCoMod_Param _),
obCoMod_Param _

with obCoMod :
  forall _, Type :=

| ViewOb :
  forall G : obGenerator,
obCoMod _

| Viewing_Form_default :
  forall (F : obCoMod  Param_SubstF)
  (* (VV : forall G, Sieve)(WW : forall param, Sieve) *),
obCoMod _

| ViewedOb_default :
  forall (F: obCoMod _) (Param_F : obCoMod_Param _)
  (* (VV : forall G, Sieve)(WW : forall param, Sieve) *),
obCoMod _

| Functions :
  forall (F: obCoMod _) (Param_F : obCoMod_Param _),
  forall (Param_FunctionsF : obCoMod_Param _),
  forall (PParam_SubstF : morCode_PParam Sense1_Param_proj
Sense1_Param_subst),
obCoMod _

with morCode_PParam :
  forall (Sense1_Param_subst_ff : Sense1_Param_def Sense01_Param_EF
Sense01_Param_F), Type :=

with morCode :
  forall (Sense1_Form_ff : Sense1_Form_def Sense1_FormParam_E
Sense1_FormParam_F
  Sense1_Param_proj_ff Sense1_Param_subst_ff), Type :=  .
```

C3 / coq

3. The elements of the data-objects, as constructors or as algebraic operations

The elements of some data object sheaf can be either generating constructors or algebraic-geometric (restriction, zero, addition, linear action, coboundary). Memo that the formulation of the generating constructors is such that they accumulate restrictions by arrows belonging to some viewing (sieve); this enables to eliminate the algebraic-restrictions from the solution.

```
C4 / coq (Inductive elAlgebra_Param (* (VV : forall G, Sieve) *):
  forall (Param_SubstF: obCoMod_Param _)
  (G : obGenerator) (paramlocal : Sense00_Param_SumSubstF G)
  (P : obParametrizator) (inFibre_P : inFibre G P), Type :=

with elConstruct_Form (* (VV : forall G, Sieve)(WW : forall param, Sieve) *):
  forall (Param_SubstF : obCoMod_Param _) (F : obCoMod Param_SubstF)
  param (cons_paramlocal : c.elConstruct_Param (* VV *) Param_SubstF
            (sval Sense1_Param_subst G param) (InFibre G))
  form, Type :=

| True_Binary_Form :
  forall (G' : obGenerator) (g : 'Generator( G' ~> GTop (* | WW true *) )),
elConstruct_Form (* WW *)  _

| ViewOb_Refine_El_Form :
(* (UU : forall param, Sieve) <= WW *)
  forall _, elConstruct_Form (* UU *) _ ~> elConstruct_Form (* WW *) _

with elAlgebra_Form (* (VV : forall G, Sieve)(WW : forall param, Sieve) *):
  forall (Param_SubstF : obCoMod_Param _) (F : obCoMod Param_SubstF)
  param (cons_paramlocal : c.elAlgebra_Param (* VV *) Param_SubstF
            (sval Sense1_Param_subst G param) (InFibre G))
  form, Type :=

| ElConstruct_elAlgebra_Form :
  forall _,
elConstruct_Form _  -> elAlgebra_Form _

| Restrict_elAlgebra_Form :
  forall param cons_paramlocal form (cons_form : elAlgebra_Form form),
  forall (G' : obGenerator) (g : 'Generator( G' ~> G (* | WW param *) )),
elAlgebra_Form ( g o> form )

(* | Zero :  forall _ , elAlgebra_Form _
| Addition :  forall _ , elAlgebra_Form _ -> elAlgebra_Form _ ->
elAlgebra_Form _
| Coboundary :  forall _ , elAlgebra_Form (n) _  -> elAlgebra_Form (n+1) _
*).) C4 / coq
```

4. Conversions equations in the algebra of the elements of the data-objects

As an example, it is possible to formulate that two grammatically-distinct constructors [True] and [Yes] with the same sense are in fact convertible (equal) in the algebra. Those equations on the data are the result of the resolution from the conversions in the logic. The final solution to decide these equations is left to some external solver such as GAP/SINGULAR.

C5 / coq

```
Inductive convElAlgebra_Param : (* VV *)
  forall (cons_paramlocal : elAlgebra_Param Param_SubstF paramlocal),
  forall (cons_paramlocal' : elAlgebra_Param Param_SubstF paramlocal'), Type
:=
with convElAlgebra_Form : (* VV WW *)
  forall (cons_form : elAlgebra_Form F cons_paramlocal form ),
  forall (cons_form' : elAlgebra_Form F cons_paramlocal' form' ), Type :=

| TrueYes_convElAlgebra_Form :
convElAlgebra_Form (True_Binary_Form unitGenerator)
(Yes_Binary_Form unitGenerator).
```

C5 / coq

5. Morphisms, whose conversions are to be logically decided or resolved to the algebra of the elements of data-objects

The clause [Constructing] is the embedding of data-objects elements as morphisms. The clause [Destructing_Fine] is the elimination from data-objects, which says that the generating constructors (supported on the given viewing/sieve) are sufficient to define some morphism from the data-object. The clause [Destructing_Refine] is the refinement of the sieve which restrict the sheafification operation. In short: [Destructing_Fine] and Destructing_Refine] are the constructive inductive/refined formulation of the sheafification-operation-restricted by any convering sieve whose refinements are the measure for the induction. The clauses [Hold] and [Eval] are the usual abstraction/discharge and evaluation/substitution of the lambda calculus on functions. The clause [FormatMor] is context-extension from form-morphisms to parameter-morphisms. The clause [AdditionMor] is the algebra in the general logic-objects to be reduced into the algebra [Addition] of the data-objects elements.

C6 / coq

```
Inductive morCoMod :
  forall E F (PParam_EF : morCode_PParam Sense1_Param_proj_ff
Sense1_Param_subst_ff) (Form_ff : morCode Sense1_Form_ff ), Type :=

| Compos :
  forall F' (ff' : 'CoMod( F' ~> F @_ PParam_F'F @^ Form_ff' )),
  forall (ff_ : 'CoMod( F'' ~> F' @_ PParam_F''F' @^ Form_ff_ )),
'CoMod( F'' ~> F @_ (Compos_morCode_PParam PParam_F'F PParam_F''F')
                 @^ (Compos_morCode  Form_ff'  Form_ff_) )

| Unit :
  forall F, 'CoMod( F ~> F @_ _ @^ _ )

| ViewObMor :
  forall (G H : obGenerator) (g : 'Generator( G ~> H )),
'CoMod( ViewOb G ~> ViewOb H @_ _ @^ _ )

| Constructing_default :
  forall _ (* (VV : forall G, Sieve)(WW : forall param, Sieve) *),
  forall (F : obCoMod Sense1_FormParam_F Sense1_Param_proj Param_SubstF),
  forall (G : obGenerator) (param : Sense00_Param_SubstF G)
```

```
  (cons_paramlocal : c.elAlgebra_Param (* VV *) Param_SubstF  (sval
Sense1_Param_subst G param) (InFibre G))
  (form : Sense00_AtParam_ Sense1_FormParam_F Sense1_Param_proj param)
  (cons_form : elAlgebra_Form (* WW *) F  cons_paramlocal form ),
'CoMod( ViewOb G ~> Viewing_Form_default F (* VV WW *)  @_ _ @^ _ )

| ViewedMor_default :
  forall  (* (WW : forall param, Sieve) *)
  (* (UU : forall param, Sieve) <= WW *)
  (ff : 'CoMod( E ~> F @_ PParam_EF @^ Form_ff )),
  forall (param_ff : 'CoMod__( Param_E ~> Param_F @_ PParam_EF' )),
'CoMod( ViewedOb_default E Param_E (* WW *)  ~> ViewedOb_default F Param_F (*
UU *) @_ _  @^ _ )

| UnitViewedOb_default :
  forall (* (VV : forall G, Sieve)(WW : forall param, Sieve) *)
  (ff : 'CoMod ( E ~> F @_ PParam_EF @^ Form_ff )),
'CoMod ( E ~> ViewedOb_default F Param_F (* VV WW *)  @_ _  @^ _ )

| Destructing_Fine_default :
  forall _ (* (VV : forall G, Sieve)(WW : forall param, Sieve) *),
  forall (param_ee_ :
    forall (G : obGenerator) (paramlocal : Sense00_Param_SumSubstF G)
    (P : obParametrizator) (inFibre_P : inFibre G P)
    (cons_paramlocal : c.elConstruct_Param (* VV *) Param_SubstF paramlocal
inFibre_P ),
    'CoMod__( ViewOb_Param P ~> Param_E @_ _ )),
  forall (ee__ :
    forall (G : obGenerator) (param : Sense00_Param_SubstF G)
    (cons_paramlocal : c.elConstruct_Param (* VV *) Param_SubstF (sval
Sense1_Param_subst G param) (InFibre G))
    (form : Sense00_AtParam_ Sense1_FormParam_F Sense1_Param_proj param)
    (cons_form : elConstruct_Form (* WW *) F cons_paramlocal form ),
    'CoMod( ViewOb G ~> E @_ _  @^ _)),
'CoMod( Viewing_Form_default F (* VV WW *) ~> ViewedOb_default E Param_E (*
VV WW *) @_ _ @^ _ )

| Destructing_Refine_default :
  forall _ (* (VV : forall G, Sieve)(WW : forall param, Sieve) *)
  (* (UU : forall param, Sieve) <= WW *),
  forall _ (ee__ :
    forall (G : obGenerator) (param : Sense00_Param_SubstF G)
    (cons_paramlocal : c.elConstruct_Param (* VV *) Param_SubstF (sval
Sense1_Param_subst G param) (InFibre G))
    (form : Sense00_AtParam_ Sense1_FormParam_F Sense1_Param_proj param)
    (cons_form : elConstruct_Form (* WW *) F cons_paramlocal form ),
    'CoMod( Viewing_Form_default F (* VV UU *) ~> ViewedOb_default E Param_E
(* VV UU *)  @_ _  @^ _)),
'CoMod( Viewing_Form_default F (* VV WW *) ~> ViewedOb_default E Param_E (*
VV WW *)  @_ _ @^ _ )
```

```
| Eval :
  forall (param_eval : 'CoMod__( Param_FunctionsF ~> Param_F @_
PParam_SubstF )),
  forall E (ee : 'CoMod( F ~> E @_  PParam_FE @^ Form_ee )),
'CoMod( Functions F Param_F Param_FunctionsF PParam_SubstF ~> E  @_ _ @^ _ )

| Hold :
  forall (param_eval : 'CoMod__( Param_FunctionsF ~> Param_F @_
PParam_SubstF ) ),
  forall L (ll : 'CoMod( L ~> F @_ PParam_LF @^ Form_ll )),
'CoMod( L ~> Functions F Param_F Param_FunctionsF PParam_SubstF @_ _ @^ _ )

where "''CoMod' ( E ~> F @_ PParam_EF @^ Form_ff )" := (morCoMod _ )

(* | AdditionMor: _ *)

with morCoMod_PParam :
  forall Param_E Param_F
  (PParam_EF : morCode_PParam Sense1_Param_proj_ff Sense1_Param_subst_ff),
Type :=

| FormatMor :
  forall (ff : 'CoMod( E ~> F @_  PParam_EF @^ Form_ff ))
  (param_ff : 'CoMod__( Param_E ~> Param_F @_ PParam_EF' )),
  forall Param_D (param_ee : 'CoMod__( Param_D ~> Param_E @_ PParam_DE )),
'CoMod__( Param_D ~> FormatOb F Param_F @_ _ )

where "''CoMod__' (  Param_E  ~>  Param_F  @_ PParam_EF )" :=
(morCoMod_PParam _).
```
C6 / coq

6. Conversions, both logical conversions and resolution to the algebra-conversions

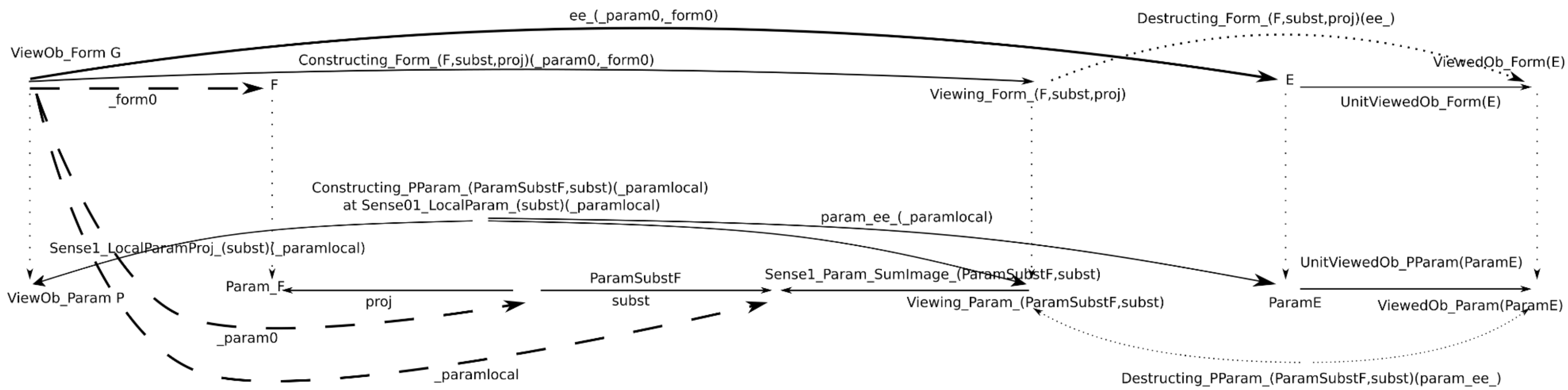
The clause [Constructing_cong] is the exit passage from logic to algebra, whose final decision is by some external solver such as GAP/SINGULAR. The clauses [Constructing_comp_Destructing] and [Hold_comp_Eval] are the usual cancellation conversions. Memo that any conversions in the forms-morphisms above may change the parameter-morphisms below; that is, conversions cause reparameterization which must be tracked explicitly. Such reparameterization include structural reparameterizations such as the associativity of bracketing and its coherence.

C7 / coq

```
Inductive convCoMod :
  forall (PParam_EF : morCode_PParam _) (Form_ff : morCode _)
  (ff : 'CoMod( E ~> F @_ PParam_EF @^ Form_ff ))
  (ff0 : 'CoMod( E ~> F @_ PParam_EF0 @^ Form_ff0 )), Prop :=

| Constructing_default_cong :
  forall G param form  (cons_form : elConstruct_Form F param form),
```

```
  forall param' form'  (cons_form' : elConstruct_Form F param' form'),

  convElAlgebra_Form cons_form cons_form' ->

(Constructing_default cons_form') [ _ @^ _  ]<~~ (Constructing_default
cons_form)

| Constructing_default_comp_Destructing_default :
forall G param cons_paramlocal form cons_form ,

(UnitViewedOb_default Param_E ( ee__ G param cons_paramlocal form cons_form
))
  [ (Assoc_reparam _ _ _)   o>_$ _
  @^ (Assoc_congrMorCode _ _ _)  o>$ _ ]<~~
((Constructing_default (ElConstruct_elAlgebra_Form cons_form))
      o>CoMod ( Destructing_default param_ee_  ee__  ))

| Hold_comp_Eval :
forall (F : @obCoMod Sense00_Form_F Sense01_Form_F Sense00_Param_F
            Sense01_Param_F Sense1_FormParam_F)
Param_F Param_FunctionsF,
forall (param_eval : 'CoMod__(Param_FunctionsF ~> Param_F @_ PParam_SubstF)),
forall L (ll : 'CoMod( L ~> F @_ PParam_LF @^ Form_ll )),
forall E (ee : 'CoMod( F ~> E @_  PParam_FE @^ Form_ee )),

   ( ll o>CoMod ee )
     [  (Assoc_reparam _ _ _) o>_$ _
        @^ (Assoc_congrMorCode  _  _  _)  o>$ _ ]<~~
   ( (Hold param_eval ll) o>CoMod (Eval param_eval' ee) )

where "ff0 [  reparam_rr @^ congr_ff  ]<~~ ff" := (convCoMod _)

with convCoMod_PParam :
  forall (PParam_EF : morCode_PParam _)
  (param_ff : 'CoMod__( Param_E ~> Param_F @_ PParam_EF ))
  (param_ff0 : 'CoMod__( Param_E ~> Param_F @_  PParam_EF0 )), Prop :=

| Constructing_PParam_default_comp_Destructing_PParam_default : _

where "param_ff0  [  reparam_rr  ]<~~__ param_ff" := (convCoMod_PParam
_).
```
C7 / coq

Another angle of view:



((UnitViewedOb_Form(E)) "<o" (ee_(_param0,_form0)))   [isos]<~~
((Destructing_Form_(F,subst,proj)(ee_)) <o (Constructing_Form_(F,subst,proj)(_param0,_form0)))