A computational logic (coinductive) interface for schemes in algebraic geometry.

    [updated] https://github.com/1337777/cartier/blob/master/cartierSolution16.lp

This is the continuation of an ongoing research programme of discovering a truly computational logic (Lambdapi type theory) for categories, profunctors, fibred categories, univalence, polynomial functors, sites, sheaves and schemes, in the style of Kosta Dosen [1].

Firstly, a glue operation for any sheaf `S` over the sheafification modality `mod_smod` is declared:

```
    constant symbol glue : Π [A B : cat] (A_site : site A)  [S : smod A_site B]
[I : cat] [G : func I B] (L : mod A I),
transf L Id_func (smod_mod S) G
→  transf (smod_mod (mod_smod A_site L)) Id_func (smod_mod S) G;
```

From which a glue operation for any sheaf `S` over the sieve-closure modality `sieve_ssieve` is defined, where `sieve` is the presheaf (profunctor …) which classifies sieves:

```
    symbol glue_sieve_mod_def : Π [A B : cat] (A_site : site A) [I : cat] [F :
func I A]  [S : smod A_site B]  [G : func I B] [D : cat] [K : func I D] (ff : hom
F (sieve A D) K),
transf (sieve_mod ff) Id_func (smod_mod S) G)
→  transf (smod_mod (ssieve_smod ((ff '∘ (sieve_ssieve A_site _))))) Id_func
(smod_mod S) G ≔ glue …;
```

Now a transformation (predicate) into the sieves-classifier (truth-values) `sieve` corresponds to a subprofunctor (fibred/dependent profunctor), such as the maximal sieve or the intersection sieve, but also a novel “sub sieve” construction.

And when the sieve-closure `sieve_ssieve` happens to evaluate to the maximal sieve, then this glue operation is indeed the expected inverse operation of the Yoneda action by sieve-elements. In short: Nicolas Tabareau [2] is only a formalization of the semantics of sheafification, not an actual computational logic; and moreover, its Definition 5.2 causes flaws: instead of “a subobject of E is dense in E if …”, it should be “a subobject P of E is dense in another subobject Q of E if …”.

Next, there are also (substructural) versions of this story in the presence of context: tensor-product context A⊗B ⊢ C, subtype (sum type) context Σ(a:A),P(a) ⊢ C, and dependent-type context (x:A)|B(x) ⊢ C(x). It is not necessary to use multi-categories and hyperdoctrines (categories with families …) to manage contexts, don’t be indoctrinated by Mikey [3] lol … Why? Because a question which is rarely entertained by logicians is:

    “Who are my end-users?”

For dependent-type contexts, the answer lies in a (computational) implementation of the (usually semantic) context-extension operation:

```
    injective symbol Context_cat : Π [X : cat], catd X → cat;
injective symbol Context_proj_func : Π [X : cat] (A : catd X), func (Context_cat
A) X;
injective symbol Context_intro_func : Π [Y : cat] [B : catd Y] [X : cat] (xy :
func X Y), funcd (Terminal_catd X) xy B → func X (Context_cat B);
injective symbol Context_func : Π [X Y : cat] [A : catd X] [B : catd Y] [xy : func
X Y], funcd A xy B → func (Context_cat A) (Context_cat B);
```

For subtype contexts, the answer already lies in an implementation of sieves/ subobject classifiers/universes as explained in the preceding paragraphs, and in the implementation of the Sigma-sum (along fibrations) and the Pi-product (along opfibrations …) of fibred categories/profunctors (with beck-chevalley-commutation along fibres):

```
    injective symbol Sigma_catd : Π [X : cat] (F : catd X) (Z : catd (Context_cat
F)), catd X;
injective symbol Sigma_elim_funcd : Π [X : cat] (F : catd X) [Z : catd
(Context_cat F)] [X'] (G : func X X') [C : catd X'], funcd Z ((Context_proj_func
F) ∘> G) C → funcd (Sigma_catd F Z) G C;
injective symbol Pi_elim_funcd : Π [X : cat] [F : catd X] [Z : catd (Context_cat
F)] [X'] [x : func X' X] [E : catd X'], funcd E x (Pi_catd F Z) → funcd
(Fibre_catd E (Context_proj_func (Fibre_catd F x))) (Context_func
(Fibre_elim_funcd F x)) Z;
```

For tensor-product contexts, the answer lies in reformulating the elimination rule for the tensor product in multi-categories which says: if a:A, b:B ⊢ C then A⊗B ⊢ C. Each time that this rule would be used, instead its intention can be simulated by the Lambdapi end-user by manually (via macros …) adding a rewrite rule which outputs the intended body content of A⊗B ⊢ C when applied to a constructor ⟨a, b⟩. In short: multi-categories are to help the dummy end-user.

Next, the presentation of affine schemes is such that it exposes a logical interface/specification which computes; for example the structure sheaf `ascheme_mod_ring_loc` localized away from r : R has restrictions along D(s·r) ⊆ D(r) (and, substructurally, along explicit radicals D(r) ⊆ D(r^n) …) which compute (by the usual recasting of any sheaf element `f/r^n` as `(s^n)·f/(s·r)^n`):

```
    rule (@ascheme_mult_hom $Ml $Af _ $U $s $r) '∘ (_) _'∘> (ring_loc_intro
(ascheme_mod_ring_loc $Af $r) $f $n)
↪ ring_loc_intro (ascheme_mod_ring_loc $Af (ring_mult (mod_loc_ring $Ml $U) $s
$r)) (ring_mult (mod_loc_ring $Ml $U) (ring_exp (mod_loc_ring $Ml $U) $s $n) $f)
$n;
```

And similar computations have been implemented for formal (colimits) joins D(f)∨D(g) of basic opens. But Joyal’s covering axiom D(s+r) ⊆ D(s)∪D(r) of the basic open D(s+r), has been reformulated approximately as a cover by the inclusions D(s·√(a·s + b·r)) ⊆ D(√(a·s + b·r)) (similarly for r) such to simultaneously also handle the unimodular (that is, ⟨r, s⟩ = 1) cover of a basic open. By the way, this functorial double-category framework is justified even for (the limit of) any functor (diagram) of rings, rather than a single ring, because localization is left-exact …

Then, there is an implementation of locally ringed sites and schemes. But the traditional definition of schemes via isomorphisms to affine schemes won’t work computationally; instead, one has to declare that the slice-category sites of the base site satisfy the interface/specification of an affine scheme. But what is a slice-category site? How does its glue operation relate to the glue operation of the base site? This has required subtle reformulations of a continuous(-and-cocontinuous) morphism of sites with a continuous right adjoint [4] (with “technical lemma 7.20.4 00XM”):

```
    rule site_morph_mod_adjL $Sm $R (glue $r_)
↪ glue (adj_mod_adjL (site_morph_adj $Sm) (smod_mod $R) $r_);
```

But then, what is the invertibility-support D(–) for a locally ringed site and how does it relate to each affine-scheme’s invertibility-support D(–) in the slice-categories. The author of cartierSolution16.lp claims that D(f) is intended to be the SIEVE (generated by a singleton when restricted to a slice affine-scheme …) under U of all opens V where the restriction of the function f: O(U) becomes invertible, together with (a computational reformulation of) the limit condition:

```
    lim_{V:D(f)} O(V)  =  O(U)[1/f]
```

In short, some structured data is being transferred from a base scheme to its slice-categories where they are required to satisfy the affine-scheme interface.

Moreover, the affine-scheme interface is coinductive (self-reference), meaning that its slice-categories are also required to satisfy the affine-scheme interface. This slice coinduction/self-reference is deeply intrinsic and unavoidable; so that all the constructions are always relativized (in the slice-category under an object). That is, even the affine-scheme invertibility-support D(–) constructors `ascheme_inv_slice_func`, which are the basic objects which generate the topology, are also relativized by-definition:

```
    constant symbol ascheme_inv_slice_func : Π [Ml : struct_mod_loc] (Af : ascheme
Ml), Π [I] [U : func I (mod_loc_cat Ml)] (r : hom U (smod_mod (mod_loc_smod Ml))
(Terminal_func _)), func I (slice_cat U);

    constant symbol scheme_slice_ascheme : Π [Ml : struct_mod_loc] [Cs :
struct_cov_sieve (mod_loc_site Ml)] (Sc : scheme Ml Cs), Π [U : func
(cov_sieve_cat Cs) (mod_loc_cat Ml)] (u : hom U (sieve_mod (cov_sieve_hom Cs))
Id_func),
ascheme (@Struct_mod_loc (slice_cat U) (slice_site (mod_loc_site Ml) U)
  (site_morph_pullback_smod (scheme_site_morph Sc U) (mod_loc_smod Ml))
  (mod_pullb_mod_ring (mod_loc_mod_ring Ml) (slice_proj_func U))
  (mod_pullb_mod_loc (mod_loc_mod_loc Ml) (slice_proj_func U)));
```

A consequence of this formulation is that these various structure sheaves are now canonically related, beyond the mere usual knowledge that for any ring `R`:

```
    R[1/fg]  ≅   R[1/f][1/g]
```

MAX [5] recently defended an equivalence between functorial schemes `X` and locally-ringed-lattice schemes `Y`, approximately:

```
    LRDL^op( X ⇒ Spec , Y )  ≅   ( X ⇒ LRDL^op( Spec( – ), Y ) )
```

where (X ⇒ X') := Fun(CommRing , Set)(X , X') and Spec: Fun(CommRing^op, LRDL^op). But the author of cartierSolution16.lp claims that their profunctor framework should allow a hybrid handling of schemes as locally ringed sites together with their functor-of-points semantics. And most importantly, it is not necessary to try and express those things using an internal logic (not truly-computational …) within the Zarisky topos, in the style of Thierry Coquand [6].

Ultimately, it is a conjecture that this new framework could solve the open problem of discovering a (graded) differential linear logic formulation for the algebraic-geometry’s cohomology differentials via the profunctorial semantics of linear logic in the context of sieves as profunctors…

It is also a conjecture that this new framework could solve the search of a hybrid framework· combining polynomial functors (good algebra) “depending” on analytic functors (good logic) as motivated by Ehrhard’s [7, section 3.1.1, page 44] outrageous definition of the composition of polynomials by the use of differentials instead of by elementary algebra.

Another open question: would such an algebra-independent computational-logical interface for commutative (affine) schemes be able to also specify schemes of (noncommutative) associative algebras; for example, in the sense of Siqveland Arvid [8]?

But wait, here is a down-to-Earth-fresh-air sanity-check challenge (“contexte et préalables d’un débat”) for ∞-cosmonauts who may be “offended”/arguing in the vacuum against the preceding paragraphs, lol …

```
    https://github.com/1337777/cartier/blob/master/cartierSolution14.lp
    https://github.com/1337777/cartier/blob/master/Kosta_Dosen_2pages.pdf
```

where this author shows that 1+2=3 via 3 different methods: the category of natural-numbers as a higher inductive type; the natural-numbers object inside any fixed category; and the colimits inside the category of finite sets/numbers.

Indeed, this new functorial programming language, also referred as Dosen’s « m– » or « emdash » or « modos », is able to express the usual logic such as the tensor and internal-hom of profunctors, the sigma-sum and pi-product of fibred categories/ profunctors; but is also able to express the concrete and inductively-constructed categories/profunctors, to express the adjunctions such as the product-and-exponential or the constant-diagram-and-limit adjunctions within any fixed category, to express contravariance and duality such as a computational-proof that right-adjoints preserve limits from the dual statement, to express groupoids and univalent universes, to express polynomial functors as bicomodules in the double category of categories, functors, cofunctors and profunctors, etc…

Such a multi-year research programme requires new tools and frameworks to articulate the speedy circulation of (papers/apps) reviews as a currency; and as for any currency, to articulate the unavoidable questions of “theft”, “falsification”, “intoxication”, “assault” beyond the traditional context of a cash-bank-robbery-by-a-gang …

An attempt at such a new platform/marketplace is this author’s « re365.net » open-source Microsoft 365 app, developed by a community of 3,000+ contributors [9], which is a tiktok-style AI-assisted calendar overlay to standardize the contacting of (VIP) researchers/businesses by reviewers whose side-goals are to co-author or by-product more intelligent (AI) interfaces for their papers/apps API.

[1] Kosta Dosen. “Cut-elimination in categories”
[2] Nicolas Tabareau. “Lawvere-Tierney sheafification in homotopy type theory”
[3] Mikey. “Categorical logic from a categorical point of view (for dummies)”
[4] The Stacks Project. “Section 7.22 00XW: Cocontinuous functors which have a right adjoint”
[5] MAX Zeuner. “Univalent Foundations of Constructive Algebraic Geometry”
[6] Thierry Coquand. “A foundation for synthetic algebraic geometry”
[7] Thomas Ehrhard. “An introduction to differential linear logic: proof-nets, models and antiderivatives”
[8] Siqveland Arvid. “Schemes of Associative algebras”
[9] < https://meetup.com/dubai-ai > < https://dailyReviews.link >