MACLANE PENTAGON COHERENCE IS SOME RECURSIVE COMONADIC DESCENT

1337777.NET

HTTPS://GITHUB.COM/1337777

Abstract

This Coq text shows that Dosen semiassociative coherence covers Maclane associative coherence by some recursive comonadic adjunction (descent?), embedding : SemiAssoc $\leftrightarrows$ Assoc : flattening reflection. This text is some initial attempt to respond to Chlipala "Compositional Computational Reflection" [2], by seeing this "compositional" as truly functorial and by seeing this "computational reflection" as truly comonadic. This text also explains the motivations and future programme, which is related to Yanofsky text "The outer limits of reason".

Categories and Subject Descriptors F.4.1 [*Mathematical Logic And Formal Languages*]: Mechanical Theorem Proving

Keywords Coq, Coherence, Comonadic descent, Automation

1. Contents

Categories [11] study the interaction between reflections and limits. The basic configuration for reflections is :

$$\begin{tikzcd}
\text{CoMod} \arrow[r, "\text{reflection}"] \arrow[d, "\text{identity}"'] \arrow[dr, phantom, "\overset{\text{unit}}{\Longrightarrow}"] & \text{Mod} \arrow[dl, "\text{coreflection}"] \\
\text{CoMod} \arrow[d, "\text{reification}"'] & \\
\text{Modos} &
\end{tikzcd}$$

where, for all reification functor into any Modos category, the map

$$(\,\_\star \text{reflection}) \circ (\text{reification} \star \text{unit})$$

is bijective; or same, for all object M' in CoMod, the polymorphic in M map

$$(\text{coreflection}\ \_\,) \circ \text{unit}_{M'} : \text{Mod}(\text{reflection}\ M', M)$$

$$\to \text{CoMod}(M', \text{coreflection}\ M)$$

is bijective, and being also polymorphic in M' therefore has reverse map of the form $\text{counit}_M \circ (\text{reflection}\ \_\,)$ whose reversal equations is polymorphically determined by

$$\begin{aligned}(\text{coreflection} \star \text{counit}) \circ (\text{unit} \star \text{coreflection}) &= \text{identity}\\ (\text{counit} \star \text{reflection}) \circ (\text{reflection} \star \text{unit}) &= \text{identity}\ .\end{aligned}$$

And it is said that the unit natural/polymorphic/commuting transformation is the *unit of the reflection* and the reflective pair $(\text{reification} \circ \text{coreflection}, \text{reification} \star \text{unit})$ is some *coreflective ("Kan") extension functor* of the reification functor along the reflection functor. This text shows some comonadic adjunction, embedding : SemiAssoc $\leftrightarrows$ Assoc : flattening reflection.

Categories [10] [11] converge to the *descent technique*, this convergence is from both the *functorial semantics technique* with the *monadic adjunctions technique*. Now functorial semantics starts when one attempts to *internalize* the common phrasing of the logician model semantics, and this internalization has as consequence some functionalization/functorialization saturation/normalization of the original theory into some more synthetic theory; note that here the congruence saturation is some instance of postfix function composition and the substitution saturation is some instance of prefix function composition. The "Yoneda"/normalization lemma takes its sense here. And among all the relations between synthetic theories, get the *tensor of theories*, which is some *extension of theories*, and which is the coproduct (disjoint union) of all the operations of the component theories quotiented by extra *commutativity* between any two operations from any two distinct component theories; for example the tensor of two rings with units as synthetic theories gives the bimodules as functorial models.

Now Galois says that any radical extension of all the *symmetric functions* in some indeterminates, which also contains those indeterminates, is abe to be incrementally/resolvably saturated/"algebra" as some further radical extension whose interesting endomorphisms include all the permutations of the indeterminates. And when there are many indeterminates, then some of those permutations are properly preserved down the resolution ... but the resolution vanish any

GCC 1337, January 13–31, 1337, Burgpeters, Dariflo, UAS..
Copyright © 1337 ACM 1-59593-057-4/05/0004. . . $15.00.
http://dx.doi.org/10.1145/

permutation ! In this context of saturated extensions, one then views any polynomial instead as its quotient/ideal of some ring of polynomials and then pastes such quotients into “algebraic algebras” or “spectrums” or “schemes” .. This is Galois descent along Borceux-Janelidze-Tholen [10]

Now *automation programmation* [6] [7] study the interaction between some meta grammar and some base grammar (sense). The critical techniques are :

- Filter-destruct-binder (“clausal function definition”), binder(-expression) which binds itself (“val rec/fix”).
- (parametric) Accumulating continuation function which, for examples, (1) is used as exceptional-value handler, or is used as transformation/composition of some exceptional-value handler, or (2) is used as regexp matcher, or is used as transformation/composition of some regexp matchers, or is used as transformation/recursion onto some regex matchers.
- Polymorphism, multimorphim (“overloading”, “hidden inferred argument”).
- Lists or trees as general form for any datatype, for example the datatype for memorizing/storing the code of the (“parsed”) base grammar text. And the alternative interfaces are commonly for complexity-performance questions such as queues (amortized constant time double-list) or such as log trees (logarithmic time/height balanced binary-search tree where each insertion does traced/flowed-reassociations to rebalance).
- Hidden outer (computational instead of only logical) parameter (“staging the closure”), which is some expression of the form “ fn x : nat => let y := g x : nat in fn z : nat => ... ”. Two particularizations of this technique are : (1) shared address/reference/cell hidden outer parameter for “object-oriented” record of functions, (2) address/reference/cell hidden outer parameter for cached delayed evaluation (“lazy”).
- Code separation, code reuse/instantiation (“parametric modules/functors”), and external-logic (“opaque/abstract”) datatype with refined-to-internal-logic (effectable) operations which consequently preserve (“invariant”) the external-logical properties of the data and also consequently inherit the same sense (correctness).

Esquisse d’un programme : The raw combinatorial (“permutation group”) angle converge to Aigner [9]. Another parallel of the raw combinatorial techniques of Galois is the raw proof-programming techniques of Gentzen that inductive recursive arithmetic cannot well-order some ordinal. One question is whether the descent techniques and the proof-programming techniques can converge.

- The initial item shall be to internalize/functorialize the semantics of Dosen book and do automation programmation so to gather data and examples and experiments .. this is 1 page/day = 1 year memo writing. The automation programming technique has one common form mixing induction or simplification conversion or logical unification or substitution rewriting or repeated heuristic/attempt destructions (“punctuation/pause/insight”) or focused verified recursive grammatical decision-dissolver (“reflective decision procedure”); for example the form behind *crush* of Chlipala CPDT [7] is :

[induction;
(eval beta iota; auto logical unify; auto substitution rewrite);
repeat ((match goal | context match term => destruct);
(eval beta iota; auto logical unify; auto substitution rewrite)
);
congruence; omega; anyreflection]

- The next item shall be to attempt whether the focused verified recursive grammatical decision-dissolver (“compositional computational reflection”) techniques in MirrorShard [2] and the terminology/logic translator (“synthesis/compiler of abstract data types”) techniques in Fiat [3] are amenable to the categorical grammar and functorial and reflection senses. For example, in the diagram above, only in the context of SemiAssoc and Assoc as the senses, then can we put Modos to be Assoc as the meta grammar (maybe augmenting the base letters/constants to collect all the formulas/objects) and do some “calculus of reifications”? Another example is to inquire how the extensible-parser capabilities of the computer together with these terminology-translator techniques (of Fiat) allow to communicate with the computer by using some easier more-human terminology.
- The final item shall be to memo Borceux books 1 and 2 and revisit Gross [1], not only from the simplifying conversion angle or the unification (“logic programming”) angle or the substitution rewrite angle, but also from the focused verified recursive grammatical decision-dissolver and terminology/logic translator angle. This latter angle shall be related to descent techniques for existence (fullness) and identification (failfulness); so this would allow for hidden/inferred arguments to be resolved after descent or for some arguments to be programmed after descent to some easier terminology .. this is 3 pages/day = 1 year memo reading.

Some parallel necessary question is how such activities are simultaneously edited by different authors and effectively universally communicated and tutored/educated. One common obstruction is the non-uniformity of the vocabulary and the raw-expressivity terminology of the computer. Some solution shall be

- to rephrase the basic reference books/encyclopedia/texts into some uniform mathematical vocabulary, for example one may say “shared outer compulogical parameters” instead of “context” or “environment”, and this necessitates to inquire into the contrast/dual math-logic versus dia-para-logic, in the sense of Yanofsky “The outer limits of reason” : for example, what is “noisea”? what is “fool-and-theft”? what is “tradeability”? what is “time”? what is “memory”? ... , and
- to simultaneously edit some web-internet page integrated with local in-browser automation proof-programmation using some easier-expressivity terminology and with hypertextual/cross-references programmation.

The Ur/Web [4] database distributed unified-grammar programming gives some initial tools to attempt [5] such solution.

2. Maclane Associative Coherence

This Coq text shows the semiassociativity completeness and coherence internal some encoding where associative coherence is the meta :

MACLAN PENTAGON IS SOME RECURSIVE SQUARE !! This recursive square *normalize_map_assoc* is the "functorial" parallel to the normalization/flattening of binary trees; and is simply the unit of the reflection for the adjunction.

The associative coherence comes before anything else, before the semiassociative completeness and before the semiassociative coherence :
* The associative coherence, by the recursive square lemma, critically reduce to the classification of the endomorphisms in the semiassociative category.
* This associative coherence do not lack some "Newman-style" diamon lemma. The comonadic adjunction, embedding : *arrows* ⇆ *arrows_assoc* : flattening reflection, which says/subgeres that semiassociative coherence covers (and temporarily comes before) associative coherence is actually done posterior-ly (for want of formality), after it is already known that the simpler *List* subcategory of *arrows* made of endomorphisms is enough to cover (and is equivalent to) *arrows_assoc*.
* The semiassociative coherence do lack some "Newman-style" diamon lemma; and, again posterior-ly, there exists some monadic adjunction, embedding : *List* ⇆ *arrows* : flattening reflection.

The associative category is the meta of the semiassociative category, and the phrasing of semiassociative completeness *lemma_completeness* shows this actuality very clearly. The semiassociative coherence is done in some internal ("first/second order" ??) encoding relative to associative coherence; may be exists some (yet to be found) "higher-order" / Coq Gallina encoding. The semiassociative coherence do lack some "Newman-style" diamon lemma. This diamon lemma is done in somme two-step process : first the codomain object of the diamon is assumed to be in normal/flattened form and this particularized diamon lemma *lemma_directedness* is proved without holding semiassociative completeness; then this assumption is erased and the full diamon lemma *lemma_coherence0* now necessitate the semiassociative completeness.

Dosen book [8] section 4.2 then section 4.3 is written into some computationally false order/precedence. The source of this falsification is the confusion/mixup between "convertible" (definitionality/meta equal) or "propositional equal" where the things are local variables instead of fully constructor-ed explicit terms.

These Coq texts do not necessitate impredicativity and do not necessitate limitation of the flow of information from proof to data and do not necessitate large inductive types with polymorphic constructors and do not necessitate universe polymorphism; therefore no distinction between `Prop` or `Set` or `Type` is made in these Coq texts. Moreover this Coq text does not hesitate to use the very sensible/fragile library *Program.Equality* `dependent destruction` which may introduce extra non-necessary *eq_rect_eq* axioms. It is most possible to erase this *eq_rect_eq* (coherence !! ..) axiom from this Coq texts, otherwise this would be some coherence problem as deep/basic as associative coherence or semiassociative coherence. At the very least it is known that the recursive square *normalize_map_assoc* does not hold *eq_rect_eq* and is closed under the global context.

Noson Yanofsky talks about some Catalan categories to solve associative coherence in hes doctor text, may be the "functorial" normalization/flattening here is related to hes Catalan categories. Also the *AAC tactics* or *CoqMT* which already come with COQ may be related, but the ultimate motivation is different ...

```
Inductive same_assoc : ∀ A B : objects, arrows_assoc A B
→ arrows_assoc A B → Set
 := same_assoc_refl : ∀ (A B : objects) (f : arrows_assoc A
B), f ∼a f
 | ...
 | same_assoc_bracket_left_5 : ∀ A B C D : objects,
    bracket_left_assoc (A /\0 B) C D <oa bracket_left_assoc
A B (C /\0 D) ∼a
    bracket_left_assoc A B C /\1a unitt_assoc D <oa
bracket_left_assoc A (B /\0 C) D <oa
    unitt_assoc A /\1a bracket_left_assoc B C D.
```

```
Inductive normal : objects → Set :=
normal_cons1 : ∀ l : letters, normal (letter l)
| normal_cons2 : ∀ (A : objects) (l : letters), normal A →
normal (A /\0 letter l).
```

```
Fixpoint normalize_aux (Z A : objects) {struct A} : objects
:=
  match A with
    | letter l ⇒ Z /\0 letter l
    | A1 /\0 A2 ⇒ (Z </\0 A1) </\0 A2
  end
where "Z </\0 A" := (normalize_aux Z A).
```

```
Fixpoint normalize (A : objects) : objects :=
  match A with
    | letter l ⇒ letter l
    | A1 /\0 A2 ⇒ (normalize A1) </\0 A2
  end.
```

Roughtly, the lemma *development* takes as input some arrow term (bracket /\ bracket) and output some *developed* arrow term (bracket /\ 1) o (1 /\ bracket) which is ∼*s* convertible (essentially by bifunctoriality of /\) to the input. Now surprisingly, this *development* or factorization lemma necessitate some deep ('well-founded') induction, using some measure *length* which shows that this may be related to arithmetic factorization.

```
Fixpoint length (A B : objects) (f : arrows A B) {struct
f} : nat :=
  match f with
  | unitt _ ⇒ 2
  | bracket_left _ _ _ ⇒ 4
  | up_1 A0 B0 A1 B1 f1 f2 ⇒ length A0 B0 f1 × length
A1 B1 f2
  | com A B C f1 f2 ⇒ length A B f1 + length B C f2
  end.
```

```
Lemma development : ∀ (len : nat) (A B : objects) (f : arrows
A B),
        length f ≤ len → { f' : arrows A B &
                                   developed f' × ((length
f' ≤ length f) × (f ∼s f')) }.
```

```
Fixpoint normalize_aux_unitrefl_assoc Y Z (y : arrows_assoc
Y Z) A
 : arrows_assoc (Y /\0 A) (Z </\0 A) :=
  match A with
    | letter l ⇒ y /\1a unitt_assoc (letter l)
    | A1 /\0 A2 ⇒ ((y </\1a A1) </\1a A2) <oa
bracket_left_assoc Y A1 A2
  end
where "y </\1a A" := (normalize_aux_unitrefl_assoc y A).
```

```
Fixpoint normalize_unitrefl_assoc (A : objects) : arrows_assoc
A (normalize A) :=
  match A with
```

| *letter l* ⇒ *unitt_assoc* (*letter l*)
| *A1* /\0 *A2* ⇒ (*normalize_unitrefl_assoc A1*) </\1a *A2*
end.

Check th151 : ∀ *A* : objects, normal *A* → normalize *A* = *A*.

Aborted th270 : for local variable *A* with *normal A*, although there is the propositional equality *th151*: *normalize* $A = A$, one gets that *normalize A* and *A* are not convertible (definitionally/meta equal); therefore one shall not regard *normalize_unitrefl_assoc A* and *unitt A* as sharing the same domain-codomain indices of *arrows_assoc*.

Check th260 : ∀ *N P* : objects, arrows_assoc *N P* → normalize *N* = normalize *P*.

Aborted lemma_coherence_assoc0 : for local variables *N*, *P* with *arrows_assoc N P*, although there is the propositional equality *th260* : *normalize N* = *normalize P*, one gets that *normalize N* and *normalize P* are not convertible (definitionally/meta equal); therefore some transport other than *eq_rect*, some coherent transport is lacked.

Below *directed y* signify that *y* is in the image of the embedding of *arrows* into *arrows_assoc*.

Check normalize_aux_map_assoc : ∀ (*X Y* : objects) (*x* : arrows_assoc *X Y*)
(*Z* : objects) (*y* : arrows_assoc *Y Z*), directed *y* →
∀ (*A B* : objects) (*f* : arrows_assoc *A B*),
{ *y_map* : arrows_assoc (*Y* </\0 *A*) (*Z* </\0 *B*) &
(*y_map* <oa *x* </\1a *A* ∼a *y* </\1a *B* <oa *x* /\1a *f*) × directed *y_map* }.

Check normalize_map_assoc : ∀ (*A B* : objects) (*f* : arrows_assoc *A B*),
{ *y_map* : arrows_assoc (normalize *A*) (normalize *B*) &
(*y_map* <oa normalize_unitrefl_assoc *A* ∼a
normalize_unitrefl_assoc *B* <oa *f*) × directed *y_map*
}.
Print Assumptions normalize_map_assoc.

Closed under the global context

3. Dosen SemiAssociative Coherence

Inductive *nodes* : *objects* → Set :=
self : ∀ *A* : *objects*, *A*
| *at_left* : ∀ *A* : *objects*, *A* → ∀ *B* : *objects*, *A* /\0 *B*
| *at_right* : ∀ *A B* : *objects*, *B* → *A* /\0 *B*.

Inductive *lt_right* : ∀ *A* : *objects*, *A* → *A* → Set :=
lt_right_cons1 : ∀ (*B* : *objects*) (*z* : *B*) (*C* : *objects*),
self (*C* /\0 *B*) <r *at_right C z*
| *lt_right_cons2* : ∀ (*B C* : *objects*) (*x y* : *B*),
x <r *y* → *at_left x C* <r *at_left y C*
| *lt_right_cons3* : ∀ (*B C* : *objects*) (*x y* : *B*),
x <r *y* → *at_right C x* <r *at_right C y*.

Definition *bracket_left_on_nodes* (*A B C* : *objects*)
(*x* : *nodes* (*A* /\0 (*B* /\0 *C*))) : *nodes* ((*A* /\0 *B*) /\0 *C*).
dependent destruction *x*.
exact (*at_left* (*self* (*A* /\0 *B*)) *C*).
exact (*at_left* (*at_left x B*) *C*).
dependent destruction *x*.
exact (*self* ((*A* /\0 *B*) /\0 *C*)).
exact (*at_left* (*at_right A x*) *C*).
exact (*at_right* (*A* /\0 *B*) *x*). Defined.

Check arrows_assoc_on_nodes : ∀ *A B*, arrows_assoc *A B* → nodes *A* → nodes *B*.

Soundness : (using *nodes* and *arrows_assoc_on_nodes* notations coercions)

Check lemma_soundness : ∀ *A B* (*f* : arrows *A B*) (*x y* : *A*), *f x* <r *f y* → *x* <r *y*.

Completeness : lemma *lem005700* is deep ('well-founded') induction on *lengthn''*, with accumulator/continuation *cumul_letteries*. The prerequisites are 4000 lines of multi-folded/complicated Coq text , and some of which are listed below. And this Coq text is at minimum as complicated as, and certainly less smooth than, the correctness proofs of the regular expression matcher or the balanced binary-search tree presented in Chlipala CPDT [7] or Harper SMLBOOK [6].

Check lemma_completeness : ∀ (*B A* : objects) (*f* : arrows_assoc *B A*),
(∀ *x y* : nodes *B*, *x* <r *y* → (*f x*) <r (*f y*)) → arrows *A B*.

Check lem005700 : ∀ (*B* : objects) (*len* : nat),
∀ (*cumul_letteries* : nodes *B* → bool)
(*H_cumul_letteries_wellform* : cumul_letteries_wellform' *B cumul_letteries*)
(*H_cumul_letteries_satur* : ∀ *y* : nodes *B*, *cumul_letteries y* = true
→ ∀ *z* : nodes *B*, lt_leftorright_eq *y z* → *cumul_letteries z* = true)
(*H_len* : lengthn'' *cumul_letteries H_cumul_letteries_wellform* ≤ *len*),
∀ (*A* : objects) (*f* : arrows_assoc *B A*)
(*H_node_is_lettery* : ∀ *x* : nodes *B*, *cumul_letteries x* = true → node_is_lettery *f x*)
(*H_object_at_node* : ∀ *x* : nodes *B*, *cumul_letteries x* = true

→ object_at_node *x* = object_at_node (*f x*))
(*H_cumul_B* : ∀ *x y* : nodes *B*, *x* <r *y* → (*f x*) <r (*f y*)), arrows *A B*.

Notation *lt_leftorright_eq x y* := (*sum* (*eq x y*) (*sum* (*x* <*l y*) (*x* <*r y*))).

And *nodal_multi_bracket_left_full*, which is some localized/deep *multi_bracket_left* at some internal node, is one of the most complicated/multifolded construction in this Coq text. And *nodal_multi_bracket_left_full* below and later really lack this constructive equality *objects_same*, so that we get some transport map which is coherent, transport map other than *eq_rect*. Maybe it is possible to effectively use the constructive equality *objects_same* at more places instead of *eq*.

Inductive *objects_same* : *objects* → *objects* → Set :=
objects_same_cons1 : ∀ *l* : *letters*, *objects_same* (*letter l*) (*letter l*)
| *objects_same_cons2* : ∀ *A A'* : *objects*, *objects_same A A'* →
∀ *B B'* : *objects*, *objects_same B B'* →
objects_same (*A* /\0 *B*) (*A'* /\0 *B'*).

Fixpoint *foldright* (*A* : *objects*) (*Dlist* : *list objects*) {struct *Dlist*} : *objects* :=
match *Dlist* with

```
      | nil ⇒ A
      | D0 :: Dlist0 ⇒ (A /\\ Dlist0) /\0 D0
where "A /\\ Dlist" := (foldright A Dlist).

Check multi_bracket_left : ∀ (A B C : objects) (Dlist : list
objects),
           arrows (A /\0 (B /\0 C /\\ Dlist)) ((A /\0 B) /\0
C /\\ Dlist).
Fixpoint object_at_node (A : objects) (x : A) {struct x} :
objects :=
   match x with
   | self A0 ⇒ A0
   | at_left A0 x0 _ ⇒ object_at_node A0 x0
   | at_right _ B x0 ⇒ object_at_node B x0
   end.

Inductive object_is_letter : objects → Set :=
      object_is_letter_cons : ∀ l : letters, object_is_letter (let-
ter l).

Notation node_is_letter x := (object_is_letter (object_at_node
x)).

Notation node_is_lettery f w := (prod
( ∀ (x : nodes _), lt_leftorright_eq w x → lt_leftorright_eq (f
w) (f x) )
( ∀ (x : nodes _), lt_leftorright_eq (f w) (f x)
          → lt_leftorright_eq ((rev f) (f w)) ((rev f) (f x)) )).

Notation cumul_letteries_wellform' B cumul_letteries :=
   (∀ x : B, node_is_letter x → eq (cumul_letteries x) true).
```

References

[1] Jason Gross, Adam Chlipala, David I. Spivak. “Experience Implementing a Performant Category-Theory Library in Coq” In: Interactive Theorem Proving. Springer, 2014.

[2] Gregory Malecha, Adam Chlipala, Thomas Braibant. “Compositional Computational Reflection” In: Interactive Theorem Proving. Springer, 2014.

[3] Benjamin Delaware, Clement Pit–Claudel, Jason Gross, Adam Chlipala. “Fiat: Deductive Synthesis of Abstract Data Types in a Proof Assistant” In: Principles of Programming Languages. ACM, 2015.

[4] Adam Chlipala. “Ur/Web: A Simple Model for Programming the Web” In: Principles of Programming Languages. ACM, 2015.

[5] 1337777.org. “1337777.org”
https://github.com/1337777/upo/

[6] Robert Harper. “Programming in Standard ML”
http://www.cs.cmu.edu/~rwh/smlbook/

[7] Adam Chlipala. “Certified Programming with Dependent Types”
http://adam.chlipala.net/cpdt/

[8] Kosta Dosen, Zoran Petric. “Proof-Theoretical Coherence”
http://www.mi.sanu.ac.rs/~kosta/coh.pdf , 2007.

[9] Martin Aigner. “Combinatorial Theory” Springer, 1997

[10] Francis Borceux, George Janelidze. “Galois Theories” Cambridge University Press, 2001.

[11] Francis Borceux. “Handbook of categorical algebra. Volumes 1 2 3” Cambridge University Press, 1994.