cartierSolution7.v

Table of Contents

Proph

https://gitee.com/OOO1337777/cartier/blob/master/cartierSolution7.v
https://github.com/1337777/cartier/blob/master/cartierSolution7.v.pdf

solves half of some question of Cartier which is how to program grammatical polymorph generated-functor-along-reindexing ("Kan extension") …

SHORT ::

The ends is to do start with some given generating-functor from some given reindexer-graph into some given generator-graph and then to generate some other extended functor from some given extended indexer-graph into some extension of the given generator-graph ; but where are those outputs of the generated-functor at the indexes ? Oneself could get them as metafunctors over this generator-graph , as long as oneself grammatically-distinguishes whatever-is-interesting .

Memo that the sense of this generated-functor (« colimits ») really-is the colimit(-simultaneously) of multiple diagrams , instead of the multiple colimits of each diagram ("pointwise") (I.3.7.2) … Moreover memo that here the generator-graph is some non-quantified outer/global parameter , instead of some innerly-quantified local argument which is carried around by all the grammatical constructors , in some « polygeneration » (functorial) form , as for some presentation of grammatical right-adjunction (I.3.7.6) … Elsewhere memo that the generated-functor is similar as some existential-quantification functor (left adjoint to some preimage functor of the generating-functor) , therefore oneself may now think of adding logical-connectives to form some external-logic of modos and to attempt polymorph (relative-)quantifier-elimination …

Now the conversion-relation shall convert across two morphisms whose sense-decoding datatype-indexes/arguments are not syntactically/grammatically-the-same . But oneself does show that , by logical-deduction , these two sense-decodings are indeed propositionally equal ("soundness lemma") .

Finally , some linear total/asymptotic grade is defined on the morphisms and the tactics-automated degradation lemma shows that each of the conversion indeed degrades the redex morphism . But to facilitate the COQ automatic-arithmetic during the degradation lemma , here oneself assumes some finiteness-property on the graph of reindexer elements of each index along the reindexing-functor and also assumes some other finiteness-property on the indexer-graph . Clearly this ongoing COQ program and deduction will still-proceed when those things are confined less than any regular cardinal .

For instant first impression , the sense-decoding ("Yoneda") of the generated-functor-along-reindexing , is written as :

```
Definition Yoneda00_Generated (I : obIndexer) (G : obGenerator) :=
  { R : { R : obReIndexer & 'Indexer( ReIndexing0 R |- I ) }
        & 'Generator( G ~> Generating0 (projT1 R) ) }.
Axiom Yoneda00_Generated_quotient :
  forall (I : obIndexer) (G : obGenerator),
  forall (R S : obReIndexer) (rs : 'ReIndexer( R |- S ))
    (si : 'Indexer( ReIndexing0 S |- I ))
    (gr : 'Generator( G ~> Generating0 R )),
    ( existT _ (existT _ S si) (gr o>Generator Generating1 rs) )
    = ( existT _ (existT _ R (ReIndexing1 rs o>Indexer si)) gr
        : Yoneda00_Generated I G ).
```

KEYWORDS :: 1337777.OOO ; COQ ; cut-elimination ; polymorph generated-functor-along-reindexing ; polymorph metafunctors-grammar ; modos

OUTLINE ::

- Indexer metalogic , generating-views data
- Grammatical presentation of objects
  - Sense-decodings of the objects
  - Grammar of the objects , which carry the sense-decodings
- Grammatical presentation of morphisms
  - Sense-decodings of the morphisms
  - Grammar of the morphisms , which carry the sense-decodings
- Solution morphisms
  - Solution morphisms without polymorphism
  - Destruction of morphisms with inner-instantiation of object-indexes
- Grammatical conversions of morphisms , which infer the same sense-decoding
  - Grammatical conversions of morphisms
  - Same sense-decoding for convertible morphisms
  - Linear total/asymptotic grade and the degradation lemma
- Polymorphism/cut-elimination by computational/total/asymptotic/reduction/(multi-step) resolution

---

HINT :: free master-engineering ; program this grammatical polymorph viewed-metafunctor-along-views-data ("sheafification") :

```
| PolyElement :
  Transformations( ( S : SubViewOfGeneratingView G ) ~> F )
  -> Transformations( G ~> ViewedMetaFunctor F )
```

BUY MOM RECURSIVE T-SQUARE :: paypal.me/1337777 1337777.OOO@gmail.com ; 微信支付 2796386464@qq.com ; eth 0x54810dcb93b37DBE874694407f78959Fa222D920 ; amazon amazon.com/hz/wishlist/ls/28SN02HR4EB8W ; irc #OOO1337777

---

1 Indexer metalogic , generating-views data

The ends is to do start with some given generating-functor from some given reindexer-graph into some given generator-graph and then to generate some other extended functor from some given extended indexer-graph into some extension of the given generator-graph ; but where are those outputs of the generated-functor at the indexes ? Oneself could get them as metafunctors over this generator-graph , as long as oneself grammatically-distinguishes whatever-is-interesting .

Memo that the sense of this generated-functor (« colimits ») really-is the colimit(-simultaneously) of multiple diagrams , instead of the multiple colimits of each diagram ("pointwise") … This is because , in this ongoing COQ program , the input object [(I : obIndexer)] is always innerly-quantified (inner/local argument instead of outer/global parameter) . Therefore , if oneself wants to change this into some outer-quantification , then oneself will get , for multiple outer-parameters [(I : obIndexer)] , the grammatical colimit of the diagram (over the graph of the reindexer elements of [I] along the reindexing-functor) determined by [I] (I.3.7.2) .

Moreover memo that , in this ongoing COQ program , the generator-graph is some non-quantified outer/global parameter , instead of some innerly-quantified local argument which is carried around by all the grammatical constructors , in some « polygeneration » (functorial) form , as for some presentation of right adjunction . Therefore , if oneself wants to change this into some polygeneration inner-quantification , then oneself will get some grammatical (right) adjoint/coreflection (in the polygeneration formulation) (I.3.7.6) .

1.1 Indexer metalogic

As common , some reindexing-functor [Parameter ReIndexing0 : obReIndexer -> obIndexer] is given from some reindexer graph to some indexer graph .

In contrast , to facilitate the COQ automatic-arithmetic during the degradation lemma , here oneself shall present the predicate [Inductive is_MorIndexer12_ (I : obIndexer) : forall R : obReIndexer, 'Indexer(ReIndexing0 R |- I) -> Type] such to force/assume [Axiom is_MorIndexer12_allP] the finiteness of this graph [{ R : obReIndexer & 'Indexer(ReIndexing0 R |- (I : obIndexer)) }] of common-interest (« graph of reindexer elements of some index [I] along the reindexing-functor » , in other words : « the preimage of some index [I] along the reindexing-functor ») ; also some other finiteness is forced/assumed [Axiom is_ObIndexer12_allP] on the indexes of the indexer graph [obIndexer] . Clearly this ongoing COQ program and deduction will still-proceed when those things are confined less than any regular cardinal .

```
From mathcomp
    Require Import ssreflect ssrfun ssrbool eqtype ssrnat seq choice fintype tuple.
Require Psatz.

Module GENERATEDFUNCTOR.

Set Implicit Arguments.
Unset Strict Implicit.
Unset Printing Implicit Defensive.
```

```
Delimit Scope poly_scope with poly.
Open Scope poly.

Parameter obIndexer : Type.
Parameter morIndexer : obIndexer -> obIndexer -> Type.
Parameter polyIndexer :
  forall A A', morIndexer A' A -> forall A'', morIndexer A'' A' -> morIndexer A'' A .
Parameter unitIndexer : forall {A : obIndexer}, morIndexer A A.

Notation "''Indexer' ( A' |- A )" :=
 (@morIndexer A' A) (at level 0, format "''Indexer' (  A'  |-  A  )") : poly_scope.
Notation "_@ A'' o>Indexer a'" := (@polyIndexer _ _ a' A'')
          (at level 40, A'' at next level, left associativity,
           format "_@ A''  o>Indexer  a'") : poly_scope.
Notation "a_ o>Indexer a'" := (@polyIndexer _ _ a' _ a_)
                  (at level 40, a' at next level, left associativity) : poly_scope.

Axiom polyIndexer_morphism :
  forall (A A' : obIndexer) (a' : 'Indexer( A' |- A ))
    (A'' : obIndexer) (a_ : 'Indexer( A'' |- A' )),
  forall B (b : 'Indexer( B |- A'' )),
      b o>Indexer ( a_ o>Indexer a' ) = ( b o>Indexer a_ ) o>Indexer a' .

Axiom polyIndexer_unitIndexer :
  forall (A A' : obIndexer) (a' : 'Indexer( A' |- A )),
    a' = ( (@unitIndexer A') o>Indexer a' ) .

Axiom unitIndexer_polyIndexer :
  forall (A : obIndexer), forall (A'' : obIndexer) (a_ : 'Indexer( A'' |- A )),
    a_ = ( a_ o>Indexer (@unitIndexer A) ) .

Parameter obReIndexer : Type.
Parameter morReIndexer : obReIndexer -> obReIndexer -> Type.
Parameter polyReIndexer :
  forall A A', morReIndexer A' A -> forall A'', morReIndexer A'' A' -> morReIndexer A'' A .
Parameter unitReIndexer : forall {A : obReIndexer}, morReIndexer A A.

Notation "''ReIndexer' ( A' |- A )" := (@morReIndexer A' A)
                  (at level 0, format "''ReIndexer' (  A'  |-  A  )") : poly_scope.
Notation "_@ A'' o>ReIndexer a'" := (@polyReIndexer _ _ a' A'')
          (at level 40, A'' at next level, left associativity,
           format "_@ A''  o>ReIndexer  a'") : poly_scope.
Notation "a_ o>ReIndexer a'" := (@polyReIndexer _ _ a' _ a_)
                  (at level 40, a' at next level, left associativity) : poly_scope.

Axiom polyReIndexer_morphism :
  forall (A A' : obReIndexer) (a' : 'ReIndexer( A' |- A ))
    (A'' : obReIndexer) (a_ : 'ReIndexer( A'' |- A' )),
  forall B (b : 'ReIndexer( B |- A'' )),
      b o>ReIndexer ( a_ o>ReIndexer a' ) = ( b o>ReIndexer a_ ) o>ReIndexer a' .

Axiom polyReIndexer_unitReIndexer :
  forall (A A' : obReIndexer) (a' : 'ReIndexer( A' |- A )),
    a' = ( (@unitReIndexer A') o>ReIndexer a' ) .

Axiom unitReIndexer_polyReIndexer :
  forall (A : obReIndexer), forall (A'' : obReIndexer) (a_ : 'ReIndexer( A'' |- A )),
    a_ = ( a_ o>ReIndexer (@unitReIndexer A) ) .

Parameter ReIndexing0 : obReIndexer -> obIndexer.
Parameter ReIndexing1 : forall A A' : obReIndexer,
    'ReIndexer( A |- A' ) -> 'Indexer( ReIndexing0 A |- ReIndexing0 A') .

Axiom ReIndexing_morphism :
  forall (A A' : obReIndexer) (a' : 'ReIndexer( A' |- A ))
    (A'' : obReIndexer) (a_ : 'ReIndexer( A'' |- A' )),
ReIndexing1 ( a_ o>ReIndexer a' ) = ( ReIndexing1 a_ ) o>Indexer ( ReIndexing1 a' ).

Axiom ReIndexing_unitReIndexer :
  forall (A : obReIndexer),
    (@unitIndexer (ReIndexing0 A)) = ( ReIndexing1 (@unitReIndexer A) ) .

Parameter ObReIndexer1_ : obIndexer -> obReIndexer.
Parameter ObReIndexer2_ : obIndexer -> obReIndexer.
Parameter MorIndexer1_ :
  forall I : obIndexer, 'Indexer( ReIndexing0 (ObReIndexer1_ I) |- I ).
```

```
Parameter MorIndexer2_ :
  forall I : obIndexer, 'Indexer( ReIndexing0 (ObReIndexer2_ I) |- I ).

Inductive is_MorIndexer12_ (I : obIndexer) :
  forall R : obReIndexer, 'Indexer( ReIndexing0 R |- I ) -> Type :=
| Is_MorIndexer12_MorIndexer1_ : is_MorIndexer12_ (MorIndexer1_ I)
| Is_MorIndexer12_MorIndexer2_ : is_MorIndexer12_ (MorIndexer2_ I) .

Axiom is_MorIndexer12_allP : forall (I : obIndexer),
    forall (R : obReIndexer) (ri : 'Indexer( ReIndexing0 R |- I )), is_MorIndexer12_ ri.

Parameter ObIndexer1 : obIndexer.
Parameter ObIndexer2 : obIndexer.

Inductive is_ObIndexer12 : obIndexer -> Type :=
| Is_ObIndexer12_ObIndexer1 : is_ObIndexer12 (ObIndexer1)
| Is_ObIndexer12_ObIndexer2 : is_ObIndexer12 (ObIndexer2) .

Axiom is_ObIndexer12_allP : forall (I : obIndexer), is_ObIndexer12 I.
```

1.2 Generating-views data

As common , some generating functor [Parameter Generating0 : obReIndexer -> obGenerator] is given from some reindexer graph to some generator graph . Each output of the generated-functor at some index is some grammatically-distinguished ("new") metafunctor over this generator graph .

```
Parameter obGenerator : Type.
Parameter morGenerator : obGenerator -> obGenerator -> Type.
Parameter polyGenerator :
  forall A A', morGenerator A' A -> forall A'', morGenerator A'' A' -> morGenerator A'' A .
Parameter unitGenerator : forall {A : obGenerator}, morGenerator A A.

Notation "''Generator' ( A' ~> A )" := (@morGenerator A' A)
                  (at level 0, format "''Generator' (  A'  ~>  A  )") : poly_scope.
Notation "_@ A'' o>Generator a'" := (@polyGenerator _ _ a' A'')
          (at level 40, A'' at next level, left associativity,
           format "_@ A''  o>Generator  a'") : poly_scope.
Notation "a_ o>Generator a'" := (@polyGenerator _ _ a' _ a_)
                 (at level 40, a' at next level, left associativity) : poly_scope.

Axiom polyGenerator_morphism :
  forall (A A' : obGenerator) (a' : 'Generator( A' ~> A ))
    (A'' : obGenerator) (a_ : 'Generator( A'' ~> A' )),
  forall B (b : 'Generator( B ~> A'' )),
      b o>Generator ( a_ o>Generator a' ) = ( b o>Generator a_ ) o>Generator a' .

Axiom polyGenerator_unitGenerator :
  forall (A A' : obGenerator) (a' : 'Generator( A' ~> A )),
    a' = ( (@unitGenerator A') o>Generator a' ) .

Axiom unitGenerator_polyGenerator :
  forall (A : obGenerator), forall (A'' : obGenerator) (a_ : 'Generator( A'' ~> A )),
    a_ = ( a_ o>Generator (@unitGenerator A) ) .

Parameter Generating0 : obReIndexer -> obGenerator.
Parameter Generating1 : forall A A' : obReIndexer,
    'ReIndexer( A |- A' ) -> 'Generator( Generating0 A ~> Generating0 A') .

Axiom Generating_morphism :
  forall (A A' : obReIndexer) (a' : 'ReIndexer( A' |- A ))
    (A'' : obReIndexer) (a_ : 'ReIndexer( A'' |- A' )),
Generating1 ( a_ o>ReIndexer a' ) = ( Generating1 a_ ) o>Generator (Generating1 a').

Axiom Generating_unitReIndexer :
  forall (A : obReIndexer),
    (@unitGenerator (Generating0 A)) = ( Generating1 (@unitReIndexer A) ) .
```

2 Grammatical presentation of objects

The sense-decoding of any object is some metafunctor . The sense-decoding of any morphism is some metatransformation . The grammatical objects are simultaneously-mutually presented with their sense-decoding ; this could be done via some common inductive-recursive presentation or alternatively by inferring the sense-decoding computation into extra indexes of the type-family of the objects . This same comment holds for the presentation of grammatical morphisms .

While the common choice would be to use the inductive-recursive presentation, it is true that the extra-indexes presentation enable the easy sharing of indexes among grammatical objects and grammatical morphisms ; therefore this extra-indexes presentation avoids the need for manipulating extra propositional-equalities which express these sharings .

Memo that these sense-decodings may be held for two ends : (1) to express the cocone logical-condition on any input cocone data as held by the reflector-constructor ("universality-morphism" , copairing) ; (2) to express the dependence of the output limit-object on the morphisms contained in some given input diagram . In the ongoing COQ program , the description (2) will not be necessary because the morphisms contained in the input diagrams are touched indirectly and uniformly (essentially-fixed) .

2.1 Sense-decodings of the objects

The elements at some generator [G] of the metafunctor over the generator-graph which is the sense-decoding of the output of the generated-functor at some index [I] is : the reindexer elements [R] of this index [I] along the reindexing-functor which are also above this generator [G] along the generating-functor ; modulo some polyarrowing in the choice of the reindex .

```
Definition Yoneda00_Generated (I : obIndexer) (G : obGenerator) :=
  { R : { R : obReIndexer & 'Indexer( ReIndexing0 R |- I ) }
        & 'Generator( G ~> Generating0 (projT1 R) ) }.
```

Memo that in the common formulation , if the reindexing-functor is flat and the generator-graph is locally presentable , then oneself will get this ongoing formulation ; but in reality oneself may start as in this ongoing formulation where it is relaxed (less-requirements) that the universality/limitativeness property of this construction holds simultaneously (inner-quantification) at all the indexes . Moreover memo that in the case that the generator-graph is [Set] of sets , this presentation will give the common definition .

Elsewhere : oneself may see the graph [{ R : obReIndexer & 'Indexer(ReIndexing0 R |- (I : obIndexer)) }] of the reindexer elements of [I] along the reindexing-functor as some subset/predicate over the whole reindexer-graph , and see the output of the generated-functor at some index [Yoneda00_Generated (I : obIndexer)] as some predicate over the generator-graph . Then the generated-functor is similar as some existential-quantification functor (left adjoint to some preimage functor of the generating-functor) , oneself may now think of adding logical-connectives to form some external-logic of modos and to attempt polymorph (relative-)quantifier-elimination …

```
Definition Yoneda01_functor (Yoneda00 : obGenerator -> Type)
           (Yoneda01 : forall G G' : obGenerator,
               'Generator( G' ~> G ) -> Yoneda00 G -> Yoneda00 G') : Prop :=
  ( (* binary/composing functoriality *)
    ( forall G G' (g : 'Generator( G' ~> G)) G'' (g' : 'Generator( G'' ~> G')) x,
        Yoneda01 _ _ g' (Yoneda01 _ _ g x) = Yoneda01 _ _ (g' o>Generator g) x ) /\
    ( (* empty/unit functoriality is held only in PolyElement_Pairing *)
      forall G x, x = Yoneda01 _ _ (@unitGenerator G) x ) ) .

Definition Yoneda10_natural
           Yoneda00_F (Yoneda01_F : { Yoneda01 : _ | Yoneda01_functor Yoneda01 })
           Yoneda00_E (Yoneda01_E : { Yoneda01 : _ | Yoneda01_functor Yoneda01 })
           (Yoneda10 : forall G : obGenerator, Yoneda00_F G -> Yoneda00_E G) : Prop :=
  forall G G' (g : 'Generator( G' ~> G )) (f : Yoneda00_F G),
    (proj1_sig Yoneda01_E) _ _ g (Yoneda10 G f)
    = Yoneda10 G' ((proj1_sig Yoneda01_F) _ _ g f) .

Notation "<< R ; i ; g >>" := (existT _ (existT _ R i) g)
                                (at level 0, format "<<  R  ;  i  ;  g  >>").

Section Senses_obCoMod.

Lemma Yoneda00_View : forall (B : obGenerator), (obGenerator -> Type).
Proof. intros B. refine (fun A => 'Generator( A ~> B ) ). Defined.

Lemma Yoneda01_View : forall (B : obGenerator),
    {Yoneda01 : ( forall A A' : obGenerator,
          'Generator( A' ~> A ) -> (Yoneda00_View B) A -> (Yoneda00_View B) A' ) |
     Yoneda01_functor Yoneda01} .
Proof.
  intros. exists (fun A A' a x => a o>Generator x).
  abstract (split; [intros; exact: polyGenerator_morphism
                   | intros; exact: polyGenerator_unitGenerator]).
Defined.

Definition Yoneda00_Generated (I : obIndexer) (G : obGenerator) :=
  { R : { R : obReIndexer & 'Indexer( ReIndexing0 R |- I ) }
        & 'Generator( G ~> Generating0 (projT1 R) ) }.

Axiom Yoneda00_Generated_quotient :
```

```
  forall (I : obIndexer) (G : obGenerator),
  forall (R S : obReIndexer) (rs : 'ReIndexer( R |- S ))
    (si : 'Indexer( ReIndexing0 S |- I ))
    (gr : 'Generator( G ~> Generating0 R )),
    ( existT _ ( existT _ S (si) ) (gr o>Generator (Generating1 rs)) )
    = ( existT _ ( existT _ R ((ReIndexing1 rs) o>Indexer si) ) (gr)
        : Yoneda00_Generated I G ).

Lemma Yoneda01_Generated :
  forall (I : obIndexer),
    { Yoneda01 : ( forall G G' : obGenerator,
    'Generator( G' ~> G ) -> Yoneda00_Generated I G -> Yoneda00_Generated I G' ) |
      Yoneda01_functor Yoneda01 }.
Proof.
  unshelve eexists.
  - intros G G' g ii.
    refine (existT _ (existT _ (projT1 (projT1 ii)) (projT2 (projT1 ii)))
                   (g o>Generator (projT2 ii))) .
  - abstract (split; [ intros; rewrite -polyGenerator_morphism; reflexivity
                     | intros G ii;
                       rewrite -polyGenerator_unitGenerator;
                       destruct ii as [[? ?] ?]; reflexivity ]) .
Defined.

Lemma Yoneda01_Generated_PolyIndexer :
  forall (I : obIndexer) (J : obIndexer) (i : 'Indexer( I |- J )),
 {Yoneda10 : forall G : obGenerator, Yoneda00_Generated I G -> Yoneda00_Generated J G |
   Yoneda10_natural (Yoneda01_Generated I) (Yoneda01_Generated J) Yoneda10} .
  intros. unshelve eexists.
  refine (fun G gi => existT _ (existT _ (projT1 (projT1 gi))
                                      ((projT2 (projT1 gi)) o>Indexer i))
                            (projT2 gi) )  .
  abstract(intros; move; reflexivity).
Defined.

End Senses_obCoMod.
```

2.2 Grammar of the objects, which carry the sense-decodings

As common , the [View] constructor is the (covariant) Yoneda-embedding (therefore [View G] is some contravariant metafunctor) .

In contrast , the polymorphism/cut-elimination resolution will require the destruction of some morphism which is constrained by the structure of its domain/codomain objects . Therefore it is necessary , to grammatically-distinguish those singleton objects which in-reality came from some indexing/family of many objects ; for example , the output-object of the generated-functor at some index is such object which shall be grammatically distinguished . Now this grammatically-distinguishing is done by using two mutually-inductive datatypes ; more-precisely the datatype for indexed/family objects [obCoMod_indexed] is nested (non-recursively , for grammatically-remembering-only …) within the datatype for singleton objects [obCoMod] .

Moreover in contrast , during the above destruction , oneself wants some additional data to be instantiated/shared , beyond the domain/codomain objects : (the sense-decoding [Yoneda01_Generated_PolyIndexer] of) the indexer-arrow (functorial-)actions across the indexed/family objects . Therefore oneself shall make the grammatical presentation of the indexed/family objects carry this additional data via some extra type-index/argument .

```
Inductive obCoMod : forall Yoneda00 : obGenerator -> Type,
    { Yoneda01 : ( forall G G' : obGenerator,
                   'Generator( G' ~> G ) -> Yoneda00 G -> Yoneda00 G' ) |
                 Yoneda01_functor Yoneda01 } -> Type :=

| AtIndexOb : forall (Yoneda00_F_ : obIndexer -> _) (Yoneda01_F_ : forall I : obIndexer, _)
               (Yoneda01_F_Poly : forall I J : obIndexer, 'Indexer( I |- J ) -> _),
    (@obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly) ->
    forall I : obIndexer, @obCoMod (Yoneda00_F_(I)) (Yoneda01_F_(I))

| View : forall G : obGenerator, @obCoMod (Yoneda00_View G) (Yoneda01_View G)

with obCoMod_indexed (**memo: non-recursive **) :
       forall Yoneda00_ : obIndexer -> obGenerator -> Type,
       forall Yoneda01_ : (forall I : obIndexer, { Yoneda01 : ( forall G G' : obGenerator,
                   'Generator( G' ~> G ) -> Yoneda00_ I G -> Yoneda00_ I G' ) |
                                        Yoneda01_functor Yoneda01 }),
       forall Yoneda01_Poly : (forall I J : obIndexer, 'Indexer( I |- J ) ->
         {Yoneda10_Poly_i : forall G : obGenerator, Yoneda00_ I G -> Yoneda00_ J G |
          Yoneda10_natural (Yoneda01_ I) (Yoneda01_ J) Yoneda10_Poly_i}), Type :=
```

```
| ObCoMod_indexed :  forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly,
    (forall I : obIndexer, @obCoMod (Yoneda00_F_(I)) (Yoneda01_F_(I))) ->
    @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly

| Generated : @obCoMod_indexed Yoneda00_Generated Yoneda01_Generated
                               Yoneda01_Generated_PolyIndexer .
```

3 Grammatical presentation of morphisms

3.1 Sense-decodings of the morphisms

The sense-decoding of any morphism is some metatransformation . Memo that these sense-decodings will be held in the constructor [Reflector] to express the cocone logical-condition on any input cocone data as held by the output reflector-constructor ("universality-morphism" , copairing) .

Memo that the quotient relation [Yoneda00_Generated_quotient] on the elements of the generated-functor at some index at some generator will be used only once to show the lemma [Yoneda10_CoUnitGenerated_form_morphismReIndexer_morphismIndexer] that the counit (section/injection) of the generated-functor is polyarrowing across the indexer and is polyarrowing across the reindexer .

```
Section Senses_morCoMod.

Lemma Yoneda10_PolyCoMod :
  forall Yoneda00_F1 Yoneda01_F1 Yoneda00_F2 Yoneda01_F2
   (Yoneda10_ff_ : {Yoneda10 : forall A : obGenerator, Yoneda00_F1 A -> Yoneda00_F2 A |
                    Yoneda10_natural Yoneda01_F1 Yoneda01_F2 Yoneda10 })
    Yoneda00_F2' Yoneda01_F2'
  (Yoneda10_ff' : {Yoneda10 : forall A : obGenerator, Yoneda00_F2 A -> Yoneda00_F2' A |
                  Yoneda10_natural Yoneda01_F2 Yoneda01_F2' Yoneda10}),
    {Yoneda10 : ( forall A : obGenerator, Yoneda00_F1 A -> Yoneda00_F2' A ) |
     Yoneda10_natural Yoneda01_F1 Yoneda01_F2' Yoneda10}.
Proof.
  intros. exists (fun A => (proj1_sig Yoneda10_ff') A \o (proj1_sig Yoneda10_ff_) A ).
  abstract (intros; move; intros; simpl; rewrite (proj2_sig Yoneda10_ff')
                                          (proj2_sig Yoneda10_ff_); reflexivity).
Defined.

Lemma Yoneda10_UnitCoMod :
  forall Yoneda00_F Yoneda01_F,
    {Yoneda10 : ( forall A : obGenerator, Yoneda00_F A -> Yoneda00_F A ) |
     Yoneda10_natural Yoneda01_F Yoneda01_F Yoneda10 } .
Proof.
  intros. exists (fun A => id).
  abstract (intros; move; intros; reflexivity).
Defined.

Lemma Yoneda10_PolyElement :
  forall Yoneda00_F Yoneda01_F (B : obGenerator) (f : Yoneda00_F B),
    {Yoneda10 : ( forall A : obGenerator, Yoneda00_View B A -> Yoneda00_F A ) |
     Yoneda10_natural (Yoneda01_View B) Yoneda01_F Yoneda10} .
Proof.
  intros. exists (fun A b => proj1_sig Yoneda01_F _ _  b f) .
  abstract (intros; move; intros; apply: (proj1 (proj2_sig Yoneda01_F))).
Defined.

Lemma Yoneda10_PolyTransf :
  forall Yoneda00_F Yoneda01_F Yoneda00_G  Yoneda01_G
    (transf : {transf : ( forall A : obGenerator, Yoneda00_F A -> Yoneda00_G A ) |
              Yoneda10_natural Yoneda01_F Yoneda01_G transf })
    (A : obGenerator)
    (Yoneda10_ff : {Yoneda10 : forall A0 : obGenerator,
                        'Generator( A0 ~> A ) -> Yoneda00_F A0 |
                   Yoneda10_natural (Yoneda01_View A) Yoneda01_F Yoneda10 }),
    {Yoneda10 : ( forall A0 : obGenerator, 'Generator( A0 ~> A ) -> Yoneda00_G A0 ) |
     Yoneda10_natural (Yoneda01_View A) Yoneda01_G Yoneda10 } .
Proof.
  intros. exists (fun A' => (proj1_sig transf) A' \o (proj1_sig Yoneda10_ff) A' ).
  abstract (intros; move; intros; simpl in *;
            rewrite (proj2_sig transf) (proj2_sig Yoneda10_ff); reflexivity).
Defined.

Lemma Yoneda10_CoUnitGenerated :
  forall (I : obIndexer), forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
      forall Yoneda00_F Yoneda01_F,
```

```
      forall Yoneda10_rr : {Yoneda10 : forall G : obGenerator,
                            Yoneda00_F G -> Yoneda00_View (Generating0 R) G |
            Yoneda10_natural Yoneda01_F (Yoneda01_View (Generating0 R)) Yoneda10},
        { Yoneda10 : forall G : obGenerator, Yoneda00_F G -> Yoneda00_Generated (I) G |
          Yoneda10_natural Yoneda01_F (Yoneda01_Generated I) Yoneda10}.
Proof.
  intros. unshelve eexists.
  refine (fun G ff => sval (Yoneda01_Generated_PolyIndexer i) G
                        (existT _ (existT _ R (@unitIndexer (ReIndexing0 R)))
                                ((proj1_sig Yoneda10_rr) G ff))).
  abstract (intros; move; intros; rewrite -(proj2_sig Yoneda10_rr); reflexivity).
Defined.

Lemma Yoneda10_Reflector :
  forall (Yoneda00_F_ : obIndexer -> _)
    (Yoneda01_F_ : forall I : obIndexer, _)
    (Yoneda10_ff_ : forall (I : obIndexer),
        forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
          {Yoneda10_ff_i : _ |
 Yoneda10_natural (Yoneda01_View (Generating0 R)) (Yoneda01_F_(I)) Yoneda10_ff_i}),
  forall (I : obIndexer),
    {Yoneda10 : forall G : obGenerator, Yoneda00_Generated I G -> Yoneda00_F_ I G |
     Yoneda10_natural (Yoneda01_Generated I) (Yoneda01_F_ I) Yoneda10} .
Proof.
  intros. unshelve eexists.
  - intros G ii. refine (sval (Yoneda10_ff_ _ _
                                              (projT2 (projT1 ii))) G (projT2 ii)) .
  - abstract (intros G G' g ii;
             rewrite [in LHS](proj2_sig (Yoneda10_ff_ _ _ _ )); reflexivity).
Defined.

Definition Yoneda10_functorIndexer (Yoneda00_F_ : obIndexer -> _)
          (Yoneda01_F_ : forall I : obIndexer, _)
          (Yoneda01_F_Poly : forall I J : obIndexer, 'Indexer( I |- J ) ->
      {Yoneda01_F_Poly_i : forall G : obGenerator, Yoneda00_F_ I G -> Yoneda00_F_ J G |
       Yoneda10_natural (Yoneda01_F_ I) (Yoneda01_F_ J) Yoneda01_F_Poly_i})
  := ( forall (I J : obIndexer) (i : 'Indexer( I |- J ))
         (K : obIndexer) (j : 'Indexer( J |- K )),
         forall (G : obGenerator),
           sval (Yoneda01_F_Poly J K j) G \o sval (Yoneda01_F_Poly I J i) G
           =1 sval (Yoneda01_F_Poly I K (i o>Indexer j)) G ) /\
     ( forall (I : obIndexer),
         forall (G : obGenerator),
           id =1 sval (Yoneda01_F_Poly I I (@unitIndexer I)) G ) .

Section Section1.
Variables (Yoneda00_F_ : obIndexer -> _)
          (Yoneda01_F_ : forall I : obIndexer, _)
          (Yoneda01_F_Poly : forall I J : obIndexer, 'Indexer( I |- J ) ->
      {Yoneda01_F_Poly_i : forall G : obGenerator, Yoneda00_F_ I G -> Yoneda00_F_ J G |
       Yoneda10_natural (Yoneda01_F_ I) (Yoneda01_F_ J) Yoneda01_F_Poly_i})
          (Yoneda10_ff_ : forall (I : obIndexer),
              forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
                {Yoneda10_ff_i : forall G : obGenerator,
                    Yoneda00_View (Generating0 R) G -> Yoneda00_F_(I) G |
 Yoneda10_natural (Yoneda01_View (Generating0 R)) (Yoneda01_F_(I)) Yoneda10_ff_i}).

Definition Yoneda10_morphismReIndexer_morphismIndexer :=
  forall (I : obIndexer),
  forall (R : obReIndexer) (ri : 'Indexer( ReIndexing0 R |- I )),
  forall (S : obReIndexer) (sr : 'ReIndexer( S |- R )),
  forall (J : obIndexer) (ij : 'Indexer( I |- J )),
  forall (G : obGenerator),
    ( sval (Yoneda10_ff_  ((ReIndexing1 sr o>Indexer ri) o>Indexer ij)) G )
    =1 (sval (Yoneda01_F_Poly ij) G \o
             (sval (Yoneda10_ff_ ri) G \o
 sval (Yoneda10_PolyElement (Yoneda01_View (Generating0 R)) (Generating1 sr)) G)).

Definition Yoneda10_morphismIndexerOnly
  := (  forall (I : obIndexer),
         forall (R : obReIndexer) (ri : 'Indexer( ReIndexing0 R |- I )),
         forall (J : obIndexer) (ij : 'Indexer( I |- J )),
         forall (G : obGenerator),
           ( sval (Yoneda10_ff_  (ri o>Indexer ij)) G )
           =1 ( sval (Yoneda01_F_Poly ij) G \o
                  (sval (Yoneda10_ff_ ri) G) )) .
```

```
Lemma Yoneda10_morphismReIndexer_morphismIndexer_to_Yoneda10_morphismIndexerOnly :
  Yoneda10_morphismReIndexer_morphismIndexer
  -> Yoneda10_morphismIndexerOnly .
Proof.
  intros H. move. intros. move. intros x.
  move => /(_  I R ri _ (unitReIndexer) J ij G) in H.
  rewrite -ReIndexing_unitReIndexer in H.
  rewrite -polyIndexer_unitIndexer in H.
  rewrite -Generating_unitReIndexer in H.
  move => /(_ x) in H. rewrite /= -unitGenerator_polyGenerator in H.
  exact: H.
Qed.

Definition Yoneda10_naturalIndexer
           (Yoneda00_E_ : obIndexer -> _)
           (Yoneda01_E_ : forall I : obIndexer, _)
           (Yoneda01_E_Poly : forall I J : obIndexer, 'Indexer( I |- J ) ->
    {Yoneda01_E_Poly_i : forall G : obGenerator, Yoneda00_E_ I G -> Yoneda00_E_ J G |
     Yoneda10_natural (Yoneda01_E_ I) (Yoneda01_E_ J) Yoneda01_E_Poly_i})
           (Yoneda10_ee_ : forall I : obIndexer, {Yoneda10_ee_I : forall G : obGenerator,
                                         Yoneda00_F_(I) G -> Yoneda00_E_(I) G |
                 Yoneda10_natural (Yoneda01_F_(I)) (Yoneda01_E_(I)) Yoneda10_ee_I})
  := forall (I J : obIndexer) (ij : 'Indexer( I |- J )),
    forall (G : obGenerator),
      ( sval (Yoneda10_ee_ J) G  \o
             sval (Yoneda01_F_Poly ij) G  )
      =1 ( sval (Yoneda01_E_Poly _ _ ij) G \o
                (sval (Yoneda10_ee_ I) G  )) .

End Section1.

Lemma Yoneda10_Reflector_naturalIndexer_ALT :
  forall (Yoneda00_F_ : obIndexer -> _)
    (Yoneda01_F_ : forall I : obIndexer, _)
    (Yoneda01_F_Poly : forall I J : obIndexer, 'Indexer( I |- J ) ->
     {Yoneda01_F_Poly_i : forall G : obGenerator, Yoneda00_F_ I G -> Yoneda00_F_ J G |
      Yoneda10_natural (Yoneda01_F_ I) (Yoneda01_F_ J) Yoneda01_F_Poly_i})
    (Yoneda10_ff_ : forall (I : obIndexer),
        forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
          {Yoneda10_ff_i : forall G : obGenerator,
              Yoneda00_View (Generating0 R) G -> Yoneda00_F_(I) G |
 Yoneda10_natural (Yoneda01_View (Generating0 R)) (Yoneda01_F_(I)) Yoneda10_ff_i}),
  forall (Yoneda10_ff_morphismReIndexer_morphismIndexer :
       Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
    Yoneda10_naturalIndexer Yoneda01_Generated_PolyIndexer Yoneda01_F_Poly
                            (Yoneda10_Reflector Yoneda10_ff_).
Proof.
  intros. rewrite /Yoneda10_naturalIndexer.
  intros; move. intros i.
 apply: (Yoneda10_morphismReIndexer_morphismIndexer_to_Yoneda10_morphismIndexerOnly
           Yoneda10_ff_morphismReIndexer_morphismIndexer).
Qed.

End Senses_morCoMod.
```

3.2 Grammar of the morphisms , which carry the sense-decodings

As common , the [PolyElement] constructor inputs some element of any functor and changes its format and outputs some generator-morphisms-(functorial-)action ("Yoneda") . Also the [PolyTransf] constructor inputs some (sense) transformation of elements across two metafunctors and changes its format and outputs some (grammatical) transformation of generator-morphisms-(functorial-)actions ("Yoneda") . Memo that both cut-constructors [PolyCoMod] and [PolyTransf] shall be erased/eliminated .

As common , the [CoUnitGenerated] constructor is the counit (section/injection) obtained from seeing the generated-functor functor as left adjoint of the precomposition-by-the-reindexing-functor functor . But there are 3 contrasts : (1) this counit (which commonly-is some family-over-the-reindexer of morphisms) is polyarrowing in the indexer along the reindexing-functor ; (2) this constructor is presented in the polymorphic formulation , and therefore accumulates some pre-composed morphism-argument ; (3) this constructor is grammatically-distinguished from the [PolyElement] constructor , instead of being defined via the [PolyElement] constructor .

As common , the [Reflector] constructor expresses the universality/limitativeness ("universality-morphism" , copairing) of the above adjunction . In contrast , this universality/limitativeness is relaxed (less-requirements) for multiple diagrams simultaneously , instead of the multiple universalities/limitativenesses of each diagram ("pointwise") . Indeed , in this ongoing COQ program , the input object [(I : obIndexer)] is always innerly-quantified (inner/local argument instead of outer/global parameter) .

Moreover in contrast , to express the grammatical conversion-relation [Reflector_morphism] that the [Reflector] constructor is polymorphic , it is necessary to grammatically-distinguish those singleton morphisms which in-reality came from some indexing/family of many morphisms ; for example , the input-morphism of this polymorphism conversion-relation is such morphism which shall be grammatically distinguished , also the output-morphism of the reflector-constructor at some index is such morphism which shall be grammatically distinguished . Now this grammatically-distinguishing is done by using two mutually-inductive datatypes ; more-precisely the datatype for indexed/family morphisms [morCoMod_indexed] is nested (non-recursively , for grammatically-remembering-only …) within the datatype for singleton morphisms [morCoMod] .

```
Reserved Notation "''CoMod' ( E ~> F @ Yoneda10 )"
         (at level 0, format "''CoMod' (  E  ~>  F  @  Yoneda10  )").
Reserved Notation "''CoMod_' ( E_ ~> F_ @ Yoneda10_ )"
         (at level 0, format "''CoMod_' (  E_  ~>  F_  @  Yoneda10_  )").

Inductive morCoMod : forall Yoneda00_E Yoneda01_E,
    @obCoMod Yoneda00_E Yoneda01_E ->
    forall Yoneda00_F Yoneda01_F,
      @obCoMod Yoneda00_F Yoneda01_F ->
      { Yoneda10 : ( forall G : obGenerator, Yoneda00_E G -> Yoneda00_F G ) |
                   Yoneda10_natural Yoneda01_E Yoneda01_F Yoneda10 } -> Type :=

(** ----outer-structural (solution) morphisms---- **)

| AtIndexMor : forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
                 (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly),
    forall (Yoneda10_ff_ : forall I : obIndexer, _ ),
      'CoMod_( E_ ~> F_ @ Yoneda10_ff_ ) ->
 forall (I : obIndexer), 'CoMod( AtIndexOb E_(I) ~> AtIndexOb F_(I) @ Yoneda10_ff_(I) )

(** -----cuts to be eliminated----- **)

| PolyCoMod : forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
                Yoneda00_F' Yoneda01_F' (F' : @obCoMod Yoneda00_F' Yoneda01_F')
                Yoneda10_ff' , 'CoMod( F' ~> F @ Yoneda10_ff' ) ->
            forall Yoneda00_F'' Yoneda01_F'' (F'' : @obCoMod Yoneda00_F'' Yoneda01_F''),
            forall Yoneda10_ff_ , 'CoMod( F'' ~> F' @ Yoneda10_ff_ ) ->
              'CoMod( F'' ~> F @ Yoneda10_PolyCoMod Yoneda10_ff_ Yoneda10_ff' )

| PolyTransf : forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
                 Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E)
         (transf : {transf : ( forall G : obGenerator, Yoneda00_F G -> Yoneda00_E G ) |
                    Yoneda10_natural Yoneda01_F Yoneda01_E transf})
         (G : obGenerator) Yoneda10_ff ,
    'CoMod( View G ~> F @ Yoneda10_ff ) ->
    'CoMod( View G ~> E @ Yoneda10_PolyTransf transf Yoneda10_ff )

(** ----solution morphisms---- **)

| UnitCoMod : forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
    'CoMod( F ~> F @ Yoneda10_UnitCoMod Yoneda01_F )

| PolyElement : forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
                  (G : obGenerator) (f : Yoneda00_F G),
    'CoMod( View G ~> F @ Yoneda10_PolyElement Yoneda01_F f )

| CoUnitGenerated : forall (I : obIndexer),
    forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F) Yoneda10_rr,
      'CoMod( F ~> View (Generating0 R) @ Yoneda10_rr ) ->
     'CoMod( F ~> AtIndexOb Generated(I) @ Yoneda10_CoUnitGenerated i Yoneda10_rr )


where "''CoMod' ( F' ~> F @ Yoneda10 )" :=
        (@morCoMod _ _ F' _ _ F Yoneda10) : poly_scope

with morCoMod_indexed (**memo: non-recursive **)
     : forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly,
    @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly ->
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly,
      @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly ->
      (forall I : obIndexer, { Yoneda10 : forall G : obGenerator,
                               Yoneda00_E_(I) G -> Yoneda00_F_(I) G |
         Yoneda10_natural (Yoneda01_E_(I)) (Yoneda01_F_(I)) Yoneda10 }) -> Type :=
```

```
(** ----outer-structural (solution) morphisms---- **)

| MorCoMod_indexed :
    forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
      (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly),
    forall (Yoneda10_ff_ : forall I : obIndexer, _ ),
      (forall (I : obIndexer),
          'CoMod( AtIndexOb E_(I) ~> AtIndexOb F_(I) @ Yoneda10_ff_(I) )) ->
      'CoMod_( E_ ~> F_ @ Yoneda10_ff_ )

(** ----solution morphisms---- **)

| Reflector :
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
      (Yoneda10_ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
            {Yoneda10_ff_i : forall G : obGenerator,
                Yoneda00_View (Generating0 R) G -> Yoneda00_F_(I) G |
  Yoneda10_natural (Yoneda01_View (Generating0 R)) (Yoneda01_F_(I)) Yoneda10_ff_i})
      (ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
     'CoMod( View (Generating0 R) ~> AtIndexOb F_(I) @ (Yoneda10_ff_(I)(R)(i)) ))
      (**memo: Yoneda01_F_Poly_functorIndexer and Yoneda10_ff_morphismReIndexerOnly not used in to show convCoMod
      (Yoneda01_F_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_F_Poly)
      (Yoneda10_ff_morphismReIndexer_morphismIndexer :
         Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
      'CoMod_( Generated ~> F_ @ Yoneda10_Reflector Yoneda10_ff_ )

where "''CoMod_' ( E_ ~> F_ @ Yoneda10_ )"
        := (@morCoMod_indexed _ _ _ E_ _ _ _ F_ Yoneda10_) : poly_scope .

Notation "''CoMod' ( F' ~> F )" := (@morCoMod _ _ F' _ _ F _)
       (at level 0, only parsing, format "''CoMod' (  F'  ~>  F  )") : poly_scope.
Notation "''CoMod_' ( E_ ~> F_ )" := (@morCoMod_indexed _ _ _ E_ _ _ _ F_ _)
                  (at level 0, format "''CoMod_' (  E_  ~>  F_  )") : poly_scope.

Notation "''AtIndexMor' ff_ I" := (@AtIndexMor _ _ _ _ _ _ _ _ _ ff_ I)
                   (at level 10, ff_ at next level, I at next level) : poly_scope.

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

Notation "ff o>Transf_ transf @ G" := (@PolyTransf _ _ _ _ _ _ G transf _ _ ff)
 (at level 3, transf at next level, G at level 0, left associativity) : poly_scope.

Notation "ff o>Transf_ transf" := (@PolyTransf _ _ _ _ _ _ _ transf _ _ ff)
                                    (at level 3, transf at next level) : poly_scope.

Notation "@ ''UnitCoMod' F" := (@UnitCoMod _ _ F)
                                  (at level 10, only parsing) : poly_scope.

Notation "''UnitCoMod'" := (@UnitCoMod _ _ _) (at level 0) : poly_scope.

Notation "''PolyElement' F f" := (@PolyElement _ _ F _ f)
                       (at level 10, F at next level, f at next level) : poly_scope.

Notation "rr o>CoMod 'CoUnitGenerated @ i" := (@CoUnitGenerated _ _ i _ _ _ _ rr)
                  (at level 4, i at next level, right associativity) : poly_scope.

Notation "''MorCoMod_indexed' ff_" := (@MorCoMod_indexed _ _ _ _ _ _ _ _ _ _ ff_)
                                     (at level 10, ff_ at next level) : poly_scope.

Notation "[[ ff_ @ Yoneda01_F_Poly_functorIndexer , Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_" :=
  (@Reflector _ _ _ _ _ ff_ Yoneda01_F_Poly_functorIndexer
             Yoneda10_ff_morphismReIndexer_morphismIndexer)
    (at level 4, Yoneda01_F_Poly_functorIndexer at next level,
     Yoneda10_ff_morphismReIndexer_morphismIndexer at next level,
     format "[[  ff_  @  Yoneda01_F_Poly_functorIndexer  ,  Yoneda10_ff_morphismReIndexer_morphismIndexer  ]]_" )

Notation "[[ ff_ ]]_" := (@Reflector _ _ _ _ _ ff_ _ _ )
                          (at level 4, format "[[  ff_  ]]_" ) : poly_scope.

Scheme morCoMod_morCoMod_indexed_ind := Induction for morCoMod Sort Prop
 with  morCoMod_indexed_morCoMod_ind := Induction for morCoMod_indexed Sort Prop.
```

```
Combined Scheme morCoMod_morCoMod_indexed_mutind from
         morCoMod_morCoMod_indexed_ind, morCoMod_indexed_morCoMod_ind.
Scheme  morCoMod_morCoMod_indexed_rect := Induction for morCoMod Sort Type
 with  morCoMod_indexed_morCoMod_rect := Induction for morCoMod_indexed Sort Type.
Definition morCoMod_morCoMod_indexed_mutrect P P0 a b c d e f g h :=
  pair (@morCoMod_morCoMod_indexed_rect P P0 a b c d e f g h)
       (@morCoMod_indexed_morCoMod_rect P P0 a b c d e f g h ).
```

4 Solution morphisms

As common, the purely-grammatical polymorphism cut-constructor [PolyCoMod] is not part of the solution terminology .

In contrast, there is one additional cut-constructor [PolyTransf] which inputs some (sense) transformation of elements across two metafunctors and changes its format and outputs some (grammatical) transformation of generator-morphisms-(functorial-)actions ("Yoneda") . Memo that both cut-constructors [PolyCoMod] and [PolyTransf] shall be erased/eliminated .

4.1 Solution morphisms without polymorphism

```
Module Sol.
Section Section1.
Delimit Scope sol_scope with sol.

Inductive morCoMod : forall Yoneda00_E Yoneda01_E,
    @obCoMod Yoneda00_E Yoneda01_E ->
    forall Yoneda00_F Yoneda01_F,
      @obCoMod Yoneda00_F Yoneda01_F ->
      { Yoneda10 : ( forall G : obGenerator, Yoneda00_E G -> Yoneda00_F G ) |
                   Yoneda10_natural Yoneda01_E Yoneda01_F Yoneda10 } -> Type :=

| AtIndexMor : forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
                 (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly),
    forall (Yoneda10_ff_ : forall I : obIndexer, _),
      'CoMod_( E_ ~> F_ @ Yoneda10_ff_ ) -> forall (I : obIndexer),
        'CoMod( AtIndexOb E_(I) ~> AtIndexOb F_(I) @ (Yoneda10_ff_(I)) )

| UnitCoMod : forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
    'CoMod( F ~> F @ Yoneda10_UnitCoMod Yoneda01_F )

| PolyElement : forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
                  (G : obGenerator) (f : Yoneda00_F G),
    'CoMod( View G ~> F @ Yoneda10_PolyElement Yoneda01_F f )

| CoUnitGenerated : forall (I : obIndexer),
    forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F) Yoneda10_rr,
      'CoMod( F ~> View (Generating0 R) @ Yoneda10_rr ) ->
     'CoMod( F ~> AtIndexOb Generated(I) @ Yoneda10_CoUnitGenerated i Yoneda10_rr )

where "''CoMod' ( F' ~> F @ Yoneda10 )" :=
        (@morCoMod _ _ F' _ _ F Yoneda10) : sol_scope

with morCoMod_indexed
     : forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly,
    @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly ->
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly,
      @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly ->
      (forall I : obIndexer, { Yoneda10 : forall G : obGenerator,
                          Yoneda00_E_(I) G -> Yoneda00_F_(I) G |
         Yoneda10_natural (Yoneda01_E_(I)) (Yoneda01_F_(I)) Yoneda10 }) -> Type :=

| MorCoMod_indexed :
    forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
      (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly),
    forall (Yoneda10_ff_ : forall I : obIndexer, _ ),
      (forall (I : obIndexer),
          'CoMod( AtIndexOb E_(I) ~> AtIndexOb F_(I) @ (Yoneda10_ff_(I)) ) ) ->
      'CoMod_( E_ ~> F_ @ Yoneda10_ff_ )

| Reflector :
```

```
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
      (Yoneda10_ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
            {Yoneda10_ff_i : forall G : obGenerator,
                Yoneda00_View (Generating0 R) G -> Yoneda00_F_(I) G |
  Yoneda10_natural (Yoneda01_View (Generating0 R)) (Yoneda01_F_(I)) Yoneda10_ff_i})
      (ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
      'CoMod( View (Generating0 R) ~> AtIndexOb F_(I) @ (Yoneda10_ff_ _ _ (i)) ))
      (Yoneda01_F_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_F_Poly)
      (Yoneda10_ff_morphismReIndexer_morphismIndexer :
         Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
      'CoMod_( Generated ~> F_ @ Yoneda10_Reflector Yoneda10_ff_ )

where "''CoMod_' ( E_ ~> F_ @ Yoneda10_ )" :=
        (@morCoMod_indexed _ _ _ E_ _ _ _ F_ Yoneda10_) : sol_scope .

End Section1.

Module Export Ex_Notations.
Delimit Scope sol_scope with sol.

Notation "''CoMod' ( F' ~> F @ Yoneda10 )" := (@morCoMod _ _ F' _ _ F Yoneda10)
          (at level 0, format "''CoMod' (  F'  ~>  F  @  Yoneda10  )") : sol_scope.

Notation "''CoMod' ( F' ~> F )" := (@morCoMod _ _ F' _ _ F _)
         (at level 0, only parsing, format "''CoMod' (  F'  ~>  F  )") : sol_scope.

Notation  "''CoMod_' ( E_ ~> F_ @ Yoneda10_ )" :=
  (@morCoMod_indexed _ _ _ E_ _ _ _ F_ Yoneda10_)
    (at level 0, format "''CoMod_' (  E_  ~>  F_  @  Yoneda10_  )") : sol_scope.

Notation "''CoMod_' ( E_ ~> F_ )" := (@morCoMod_indexed _ _ _ E_ _ _ _ F_ _)
                     (at level 0, format "''CoMod_' (  E_  ~>  F_  )") : sol_scope.

Notation "''AtIndexMor' ff_ I" := (@AtIndexMor _ _ _ _ _ _ _ _ _ _ ff_ I)
                      (at level 10, ff_ at next level, I at next level) : sol_scope.

Notation "@ ''UnitCoMod' F" := (@UnitCoMod _ _ F)
                                  (at level 10, only parsing) : sol_scope.

Notation "''UnitCoMod'" := (@UnitCoMod _ _ _) (at level 0) : sol_scope.

Notation "''PolyElement' F f" := (@PolyElement _ _ F _ f)
                        (at level 10, F at next level, f at next level) : sol_scope.

Notation "rr o>CoMod 'CoUnitGenerated @ i" := (@CoUnitGenerated _ _ i _ _ _ _ rr)
                     (at level 4, i at next level, right associativity) : sol_scope.

Notation "''MorCoMod_indexed' ff_" := (@MorCoMod_indexed _ _ _ _ _ _ _ _ _ _ ff_)
                                       (at level 10, ff_ at next level) : sol_scope.

Notation "[[ ff_ @ Yoneda01_F_Poly_functorIndexer , Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_" :=
  (@Reflector _ _ _ _ _ ff_ Yoneda01_F_Poly_functorIndexer
              Yoneda10_ff_morphismReIndexer_morphismIndexer)
    (at level 4, Yoneda01_F_Poly_functorIndexer at next level,
     Yoneda10_ff_morphismReIndexer_morphismIndexer at next level,
     format "[[  ff_  @  Yoneda01_F_Poly_functorIndexer  ,  Yoneda10_ff_morphismReIndexer_morphismIndexer  ]]_" )

Notation "[[ ff_ ]]_" := (@Reflector _ _ _ _ _ ff_ _ _ )
                            (at level 4, format "[[  ff_  ]]_" ) : sol_scope.

End Ex_Notations.

Scheme morCoMod_morCoMod_indexed_ind := Induction for morCoMod Sort Prop
  with  morCoMod_indexed_morCoMod_ind := Induction for morCoMod_indexed Sort Prop.
Combined Scheme morCoMod_morCoMod_indexed_mutind from
         morCoMod_morCoMod_indexed_ind, morCoMod_indexed_morCoMod_ind.
Scheme morCoMod_morCoMod_indexed_rect := Induction for morCoMod Sort Type
  with  morCoMod_indexed_morCoMod_rect := Induction for morCoMod_indexed Sort Type.
Definition morCoMod_morCoMod_indexed_mutrect P P0 a b c d e f :=
  pair (@morCoMod_morCoMod_indexed_rect P P0 a b c d e f )
       (@morCoMod_indexed_morCoMod_rect P P0 a b c d e f ).

Definition toPolyMor_mut :
  (forall Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E)
```

```
    Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
    Yoneda10_ff (ff : 'CoMod( E ~> F @ Yoneda10_ff ) %sol ),
      'CoMod( E ~> F @ Yoneda10_ff ) %poly ) *
  ( forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
      (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly)
      Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
      Yoneda10_ff_ (ff_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff_ ) %sol ),
      'CoMod_( E_ ~> F_ @ Yoneda10_ff_ ) %poly ) .
Proof.
  apply morCoMod_morCoMod_indexed_mutrect.
  - (* AtIndexMor *) intros ? ? ? ? ? ? ? ? ?  ff_ IHff_ I .
    exact: ('AtIndexMor IHff_ I)%poly.
  - (* UnitCoMod *) intros ? ? F .
    exact: ( @'UnitCoMod F ) %poly.
  - (* PolyElement *) intros ? ? F ? f .
    exact: ( 'PolyElement F f ) %poly.
  - (* CoUnitGenerated *) intros ? ? ? ? ? ? ? rr IHrr.
    exact: ( IHrr o>CoMod 'CoUnitGenerated @ i )%poly.
  - (* MorCoMod_indexed *) intros ? ? ? ? ? ? ? ? ? ff_ IHff_ .
    exact: ( 'MorCoMod_indexed (fun I : obIndexer => IHff_(I)) )%poly.
  - (* Reflector *) intros ? ? ? F_ ? ff_ IHff_ Yoneda01_F_Poly_functorIndexer
                          Yoneda10_ff_morphismReIndexer_morphismIndexer.
    exact: ( [[ ( fun (I : obIndexer)
                    (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )) =>
                    (IHff_ I R i) ) @ Yoneda01_F_Poly_functorIndexer ,
                Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_ )%poly.
Defined.

Definition toPolyMor := fst toPolyMor_mut.
Definition toPolyMor_indexed := snd toPolyMor_mut.
Arguments toPolyMor : simpl nomatch.
Arguments toPolyMor_indexed : simpl nomatch.

Lemma toPolyMor_mut_AtIndexMor :
  forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
         (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly)
         Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
         (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
         Yoneda10_ff_ (ff_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff_ )%sol),
  forall I : obIndexer,
    toPolyMor (AtIndexMor ff_ I) = ('AtIndexMor (toPolyMor_indexed ff_) I)%poly.
Proof. reflexivity. Qed.

Lemma toPolyMor_mut_UnitCoMod :
  forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
    toPolyMor (@'UnitCoMod F)%sol = (@'UnitCoMod F)%poly.
Proof. reflexivity. Qed.

Lemma toPolyMor_mut_PolyElement :
  forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F) (G : obGenerator)
         (f : Yoneda00_F G),
    toPolyMor ( 'PolyElement F f )%sol = ( 'PolyElement F f ) %poly.
Proof. reflexivity. Qed.

Lemma toPolyMor_mut_CoUnitGenerated :
 forall (I : obIndexer) (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I ))
        Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
        Yoneda10_rr (rr: 'CoMod( F ~> View (Generating0 R) @ Yoneda10_rr )%sol),
   toPolyMor (rr o>CoMod 'CoUnitGenerated @ i)%sol = ((toPolyMor rr) o>CoMod 'CoUnitGenerated @ i)%poly.
Proof. reflexivity. Qed.

Lemma toPolyMor_mut_MorCoMod_indexed :
  forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
         (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly)
         Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
         (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
         Yoneda10_ff_ (ff_: (forall I : obIndexer,
          'CoMod( AtIndexOb E_ I ~> AtIndexOb F_ I @ Yoneda10_ff_ I )%sol)),
    toPolyMor_indexed ('MorCoMod_indexed ff_ )%sol
    = ( 'MorCoMod_indexed (fun I : obIndexer => toPolyMor (ff_(I))) )%poly.
Proof. reflexivity. Qed.

Lemma toPolyMor_mut_Reflector :
  forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
    (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
    Yoneda10_ff_ (ff_ : (forall (I : obIndexer)
```

```
                          (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
       'CoMod( View (Generating0 R) ~> AtIndexOb F_ I @ Yoneda10_ff_ I R i )%sol))
        (Yoneda01_F_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_F_Poly)
        (Yoneda10_ff_morphismReIndexer_morphismIndexer :
        Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
    toPolyMor_indexed ([[ ff_ @ Yoneda01_F_Poly_functorIndexer ,
                         Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_ )%sol
    = ( [[ ( fun (I : obIndexer) (R : obReIndexer)
               (i : 'Indexer( ReIndexing0 R |- I )) => toPolyMor (ff_(I)(R)(i)) )
             @ Yoneda01_F_Poly_functorIndexer ,
           Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_ )%poly.
Proof. reflexivity. Qed.

Definition toPolyMor_mut_rewrite :=
  (toPolyMor_mut_AtIndexMor, toPolyMor_mut_UnitCoMod, toPolyMor_mut_PolyElement,
   toPolyMor_mut_CoUnitGenerated, toPolyMor_mut_MorCoMod_indexed,
   toPolyMor_mut_Reflector).
```

4.2 Destruction of morphisms with inner-instantiation of object-indexes

As common , the polymorphism/cut-elimination resolution will require the destruction of some morphism which is constrained by the structure of its domain/codomain objects . In contrast , during the above destruction , oneself wants some additional data to be instantiated/shared , beyond the domain/codomain objects : (the sense-decoding [Yoneda01_Generated_PolyIndexer] of) the indexer-arrow (functorial-)actions across the indexed/family objects .

Regardless the (nested) mutually-inductive presentation of the datatypes and regardless the extra-indexes in the datatype-families , oneself easily still-gets the common dependent-destruction of morphisms with inner-instantiation of object-indexes .

Moreover some contrast is during the polymorphism/cut-elimination resolution . In the earlier COQ programs for limits , it was better to start by general-destructing the prefix-argument [f_] of the composition [(f_ o>CoMod f')] and then to constrained-destruct the postfix-parameter [f'] such to use the general-polymorphism of the projection (unit of adjunction) and the instantiated-polymorphism of the pairing ; the alternative would use the instantiated-polymorphism of the projection and general-polymorphism of the pairing . In this ongoing COQ program for colimits , it is better to start by general-destructing the postfix-parameter [f'] of the composition [(f_ o>CoMod f')] and then to constrained-destruct the prefix-argument [f_] such to use the general-polymorphism of the counit (section/injection) and the instantiated-polymorphism of the reflector (copairing) ; the alternative would be the same but with more case-analyses .

```
Module Destruct_codomView.

Inductive morCoMod_codomView
: forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F ),
    forall (G : obGenerator), forall Yoneda10_ff,
        'CoMod( F ~> (View G) @ Yoneda10_ff ) %sol -> Type :=

| UnitCoMod :  forall B : obGenerator,
    morCoMod_codomView ( ( @'UnitCoMod (View B) )%sol )

| PolyElement : forall (G G' : obGenerator) (f : Yoneda00_View G G'),
    morCoMod_codomView ( ( 'PolyElement (View G) f )%sol ) .

Lemma morCoMod_codomViewP
  : forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
    forall Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G),
    forall Yoneda10_gg (gg : 'CoMod( F ~> G @ Yoneda10_gg ) %sol ),
      ltac:( destruct G; [ refine (unit : Type) | ];
                refine (morCoMod_codomView gg) ).
Proof.
  intros. case: _ _ F _ _ G Yoneda10_gg / gg.
  - intros; exact: tt.
  - destruct F; [intros; exact: tt | ].
    constructor 1.
  - destruct F; [intros; exact: tt | ].
    constructor 2.
  - intros; exact: tt.
Defined.

End Destruct_codomView.

Module Destruct_codomAtIndexObGenerated.

Inductive morCoMod_codomAtIndexObGenerated
: forall Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E ),
    forall (I : obIndexer), forall Yoneda10_ee,
```

```
      'CoMod( E ~> (AtIndexOb Generated I) @ Yoneda10_ee ) %sol -> Type :=

| UnitCoMod : forall (I : obIndexer),
    morCoMod_codomAtIndexObGenerated ( @'UnitCoMod (AtIndexOb Generated I) )%sol

| PolyElement : forall (I : obIndexer) (G : obGenerator) (f : Yoneda00_Generated I G),
    morCoMod_codomAtIndexObGenerated (PolyElement (AtIndexOb Generated I) f )%sol

| CoUnitGenerated : forall (I : obIndexer),
    forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F) Yoneda10_rr
      (rr : 'CoMod( F ~> View (Generating0 R) @ Yoneda10_rr ) %sol),
        morCoMod_codomAtIndexObGenerated ( rr o>CoMod 'CoUnitGenerated @ i )%sol

| MorCoMod_indexed :
    forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
      (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
    forall (Yoneda10_ff_ : forall I : obIndexer, _)
      (ff_ : (forall (I : obIndexer),
   'CoMod( AtIndexOb E_(I) ~> AtIndexOb Generated(I) @ (Yoneda10_ff_(I)) ) %sol)),
    forall (J : obIndexer),  morCoMod_codomAtIndexObGenerated
                       (AtIndexMor ( MorCoMod_indexed ff_ ) J)%sol

| Reflector :
    forall (Yoneda10_ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )), _ )
      (ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
            'CoMod( View (Generating0 R) ~> AtIndexOb Generated(I)
                        @ (Yoneda10_ff_ _ _ (i)) ) %sol)
      (Yoneda01_Generated_PolyIndexer_functorIndexer :
         Yoneda10_functorIndexer Yoneda01_Generated_PolyIndexer)
      (Yoneda10_ff_morphismReIndexer_morphismIndexer :
         Yoneda10_morphismReIndexer_morphismIndexer
           Yoneda01_Generated_PolyIndexer Yoneda10_ff_),
    forall (J : obIndexer),
      morCoMod_codomAtIndexObGenerated
        (AtIndexMor [[ ff_ @ Yoneda01_Generated_PolyIndexer_functorIndexer ,
                      Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_ J)%sol .

Lemma morCoMod_codomAtIndexObGeneratedP
  : forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
    forall Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G),
    forall Yoneda10_gg (gg : 'CoMod( F ~> G @ Yoneda10_gg ) %sol ),
      ltac:( destruct G as [? ? ? F_ I | ]; [ | refine (unit : Type) ];
               destruct F_; [ refine (unit : Type) | ];
                 refine (morCoMod_codomAtIndexObGenerated gg) ).
Proof.
  intros. case: _ _ F _ _ G Yoneda10_gg / gg.
  - intros ? ? ? E_ ? ? ? F_ ? ff_ J.
    destruct ff_ .
    + destruct F_; [ intros; exact: tt | ].
      constructor 4.
    + destruct F_; [ intros; exact: tt | ].
      constructor 5.
  - destruct F as [? ? ? F_ I | ]; [ | intros; exact: tt ].
    destruct F_; [ intros; exact: tt | ];
      constructor 1.
  - destruct F as [? ? ? F_ I | ]; [ | intros; exact: tt ].
    destruct F_; [ intros; exact: tt | ];
      constructor 2.
  - constructor 3.
Defined.

End Destruct_codomAtIndexObGenerated.

End Sol.
```

5 Grammatical conversions of morphisms , which infer the same sense-decoding

As common , the grammatical conversions are classified into : the total/(multi-step) conversions , and the congruences (recursive) conversions , and the constant (non-recursive) conversions which are used in the polymorphism/cut-elimination lemma , and the constant conversions which are only for the wanted sense of generated-functor-along-reindexing , and the constant conversions which are only for the confluence lemma , and the constant conversions which are derivable by using the finished cut-elimination lemma .

In contrast , because of the embedded sense-decoding extra-indexes/arguments in the datatype-families [morCoMod] [morCoMod_indexed] of the morphisms , the conversion-relation shall convert across two morphisms whose sense-decoding datatype-indexes/arguments are not syntactically/grammatically-the-same . But oneself does show that , by logical-deduction [convCoMod_sense] , these two sense-decodings are indeed propositionally equal ("soundness lemma") .

Regardless the mutually-inductive presentation of the singleton conversion-relation [convCoMod] and the indexed conversion-relation [convCoMod_indexed] , it is possible to avoid this extra predicate [convCoMod_indexed] by blending/substituting/nesting it within the constructor [AtIndexMor_cong] of the predicate [convCoMod] : for example , the new conclusion of the constructor [Reflector_cong] would be [(forall J , AtIndexMor [[ff0_]]_ J <~~ AtIndexMor [[ff_]]_ J)] .

Now memo the conversion-for-morphisms constructor [AtIndexMor'MorCoMod_indexed] which says that [grammatically collecting/familiarize many morphisms and then grammatically selecting some singleton morphism from this collection/family at some index] is convertible to [applying/substituting this index in the original collection/family/function] . This conversion-relation will be held during the polymorphism/cut-elimination resolution . One question is whether such similar conversion-for-objects (instead of for-morphisms) across singleton-objects and indexed-objects would be useful ?

Finally , some linear total/asymptotic grade is defined on the morphisms and the tactics-automated degradation lemma shows that each of the conversion indeed degrades the redex morphism .

5.1 Grammatical conversions of morphisms

```
Section Senses_convCoMod.

Definition PolyTransf_morphismPolyTransf_transf :
  forall Yoneda00_F Yoneda01_F Yoneda00_G Yoneda01_G
    (transf : {transf : ( forall A : obGenerator, Yoneda00_F A -> Yoneda00_G A ) |
              Yoneda10_natural Yoneda01_F Yoneda01_G transf})
    Yoneda00_H Yoneda01_H
    (transf' : {transf : ( forall A : obGenerator, Yoneda00_G A -> Yoneda00_H A ) |
               Yoneda10_natural Yoneda01_G Yoneda01_H transf}),
    {transf0 : forall A : obGenerator, Yoneda00_F A -> Yoneda00_H A |
     Yoneda10_natural Yoneda01_F Yoneda01_H transf0}.
Proof.
  intros. unshelve eexists.
  refine (fun G => sval transf' G \o sval transf G). intros. move. intros. simpl.
  rewrite (proj2_sig transf') (proj2_sig transf) . reflexivity.
Defined.

Definition Yoneda10_ViewGenerating_PolyReIndexer_form :
  forall (R S : obReIndexer) (sr : 'ReIndexer( S |- R )),
    {Yoneda10 : forall G : obGenerator,
        Yoneda00_View (Generating0 S) G -> Yoneda00_View (Generating0 R) G |
     Yoneda10_natural (Yoneda01_View (Generating0 S))
                      (Yoneda01_View (Generating0 R)) Yoneda10} .
Proof.
  intros. unshelve eexists.
  refine (fun G s => (sval (Yoneda01_View (Generating0 R))
                        (Generating0 S) G s (Generating1 sr))).
  abstract (intros; move; intros; simpl; exact: polyGenerator_morphism).
Defined.

Lemma Yoneda01_Generated_PolyIndexer_functorIndexer :
  ( forall (I J : obIndexer) (i : 'Indexer( I |- J ))
      (K : obIndexer) (j : 'Indexer( J |- K )),
      forall (G : obGenerator),
        sval (Yoneda01_Generated_PolyIndexer j) G \o
             sval (Yoneda01_Generated_PolyIndexer  i) G
        =1 sval (Yoneda01_Generated_PolyIndexer  (i o>Indexer j)) G )
  /\ ( forall (I : obIndexer), forall (G : obGenerator),
          id =1 sval (Yoneda01_Generated_PolyIndexer (@unitIndexer I)) G ) .
Proof.
  split.
  - intros. move. intros g. simpl.
    rewrite -[in LHS]polyIndexer_morphism . reflexivity.
  - intros. move. intros g. simpl.
    rewrite -[in RHS]unitIndexer_polyIndexer . case: g => - [? ?] ?. reflexivity.
Qed.

Lemma Yoneda10_CoUnitGenerated_form :
  forall (I : obIndexer), forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
```

```
      { Yoneda10 : _ |
        Yoneda10_natural (Yoneda01_View (Generating0 R))
                         (Yoneda01_Generated (I)) Yoneda10}.
Proof.
  intros. unshelve eexists.
  refine (fun G r => sval (Yoneda01_Generated_PolyIndexer i) G
                     (existT _ (existT _ R (@unitIndexer (ReIndexing0 R))) r)).
  abstract (intros; move; reflexivity).
Defined.

Lemma Yoneda10_CoUnitGenerated_form_morphismReIndexer_morphismIndexer :
  forall (I : obIndexer), forall (R : obReIndexer) (ri : 'Indexer( ReIndexing0 R |- I )),
      forall (S : obReIndexer) (sr : 'ReIndexer( S |- R )),
      forall (J : obIndexer) (ij : 'Indexer( I |- J )),
      forall (G : obGenerator),
        ( sval (Yoneda10_CoUnitGenerated_form
                  ((ReIndexing1 sr o>Indexer ri) o>Indexer ij)) G )
        =1 ( sval (Yoneda01_Generated_PolyIndexer ij) G \o
                  (sval (Yoneda10_CoUnitGenerated_form ri) G \o
sval (Yoneda10_PolyElement (Yoneda01_View (Generating0 R)) (Generating1 sr)) G) ).
Proof.
  intros. move. intros g. simpl.
  rewrite -[in LHS]polyIndexer_unitIndexer.
  rewrite -[in RHS]polyIndexer_unitIndexer.
  rewrite -[in LHS]polyIndexer_morphism.
  symmetry. apply: Yoneda00_Generated_quotient.
Qed.

Definition Yoneda10_CoUnitGenerated_form_morphismIndexerOnly
  := Yoneda10_morphismReIndexer_morphismIndexer_to_Yoneda10_morphismIndexerOnly
       Yoneda10_CoUnitGenerated_form_morphismReIndexer_morphismIndexer .

Lemma Yoneda10_CoUnitGenerated_form_morphismIndexerOnly_ALT : forall (I : obIndexer),
    forall (R : obReIndexer) (ri : 'Indexer( ReIndexing0 R |- I )),
    forall (J : obIndexer) (ij : 'Indexer( I |- J )),
    forall (G : obGenerator),
      ( sval (Yoneda10_CoUnitGenerated_form ( ri o>Indexer ij )) G )
      =1 ( sval (Yoneda01_Generated_PolyIndexer ij) G \o
                (sval (Yoneda10_CoUnitGenerated_form ri) G ) ) .
Proof.
  intros. move. intros g. simpl.
  rewrite -[in LHS]polyIndexer_unitIndexer.
  rewrite -[in RHS]polyIndexer_unitIndexer.
  reflexivity.
Qed.

Lemma Reflector_morphism_morphismReIndexer_morphismIndexer :
  forall (Yoneda00_F_ : obIndexer -> _)
    (Yoneda01_F_ : forall I : obIndexer, _)
    (Yoneda10_ff_ : forall (I : obIndexer),
        forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
          {Yoneda10_ff_i : _ |
 Yoneda10_natural (Yoneda01_View (Generating0 R)) (Yoneda01_F_(I)) Yoneda10_ff_i})
    (Yoneda01_F_Poly : forall I J : obIndexer, 'Indexer( I |- J ) -> _)
  (* (Yoneda01_F_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_F_Poly) *)
    (Yoneda10_ff_morphismReIndexer_morphismIndexer :
       Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
  forall (Yoneda00_E_ : obIndexer -> _ )
    (Yoneda01_E_ : forall I : obIndexer, _ ),
  forall (Yoneda10_ee_ : forall I : obIndexer, {Yoneda10_ee_I : forall G : obGenerator,
                                         Yoneda00_F_(I) G -> Yoneda00_E_(I) G |
               Yoneda10_natural (Yoneda01_F_(I)) (Yoneda01_E_(I)) Yoneda10_ee_I}),
  forall (Yoneda01_E_Poly : forall I J : obIndexer, 'Indexer( I |- J ) -> _)
  (* (Yoneda01_E_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_E_Poly) *)
    (Yoneda10_ee_naturalIndexer :
       Yoneda10_naturalIndexer Yoneda01_F_Poly Yoneda01_E_Poly Yoneda10_ee_ ),
    Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_E_Poly
  (fun (I : obIndexer) (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )) =>
     Yoneda10_PolyCoMod (Yoneda10_ff_ I R i) (Yoneda10_ee_ I)) .
Proof.
  intros. move. intros. simpl. move.  intros gs. simpl.
  rewrite Yoneda10_ff_morphismReIndexer_morphismIndexer.
  simpl. rewrite [LHS]Yoneda10_ee_naturalIndexer. reflexivity.
Qed.

End Senses_convCoMod.
```

```
Reserved Notation "ff0 <~~ ff" (at level 70).
Reserved Notation "ff0_ <~~_ ff_" (at level 70).

Inductive convCoMod : forall Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E),
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
    forall Yoneda10_ff ( ff : 'CoMod( E ~> F @ Yoneda10_ff ) ),
    forall Yoneda10_ff0 ( ff0 : 'CoMod( E ~> F @ Yoneda10_ff0 ) ), Prop :=

(**  ----- the total/(multi-step) conversions -----  **)

| convCoMod_Refl :
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
      Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G)
      Yoneda10_gg (gg : 'CoMod( F ~> G @ Yoneda10_gg ) ),
      gg <~~ gg

| convCoMod_Trans :
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
    forall Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G),
    forall Yoneda10_gg (gg : 'CoMod( F ~> G @ Yoneda10_gg ) ),
    forall Yoneda10_uTrans (uTrans : 'CoMod( F ~> G @ Yoneda10_uTrans ) ),
      ( uTrans <~~ gg ) ->
      forall Yoneda10_gg0 (gg0 : 'CoMod( F ~> G @ Yoneda10_gg0 ) ),
        ( gg0 <~~ uTrans ) -> ( gg0 <~~ gg )

(**  ----- the congruences (recursive) conversions for singleton morphisms -----  **)

| AtIndexMor_cong : forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
                (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
         (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly),
    forall Yoneda10_ff_ (ff_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff_ )),
    forall Yoneda10_ff0_ (ff0_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff0_ )),
      ff0_ <~~_ ff_ ->
      forall (I : obIndexer), (AtIndexMor ff0_(I)) <~~ (AtIndexMor ff_(I))

| PolyCoMod_cong :
    forall Yoneda00_F Yoneda01_F' (F' : @obCoMod Yoneda00_F Yoneda01_F')
      Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
      Yoneda10_ff' (ff' : 'CoMod( F' ~> F @ Yoneda10_ff' ))
      Yoneda00_F' Yoneda01_F'' (F'' : @obCoMod Yoneda00_F' Yoneda01_F'')
      Yoneda10_ff_ (ff_ : 'CoMod( F'' ~> F' @ Yoneda10_ff_ ))
      Yoneda10_ff_0 (ff_0 : 'CoMod( F'' ~> F' @ Yoneda10_ff_0 ))
      Yoneda10_ff'0 (ff'0 : 'CoMod( F' ~> F @ Yoneda10_ff'0 )),
      ff_0 <~~ ff_ -> ff'0 <~~ ff' -> ( ff_0 o>CoMod ff'0 ) <~~ ( ff_ o>CoMod ff' )

| PolyTransf_cong :
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
      Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G)
      (transf : {transf : forall A : obGenerator, Yoneda00_F A -> Yoneda00_G A |
                 Yoneda10_natural Yoneda01_F Yoneda01_G transf})
      (A : obGenerator)
      Yoneda10_ff (ff : 'CoMod( View A ~> F @ Yoneda10_ff ))
      Yoneda10_ff0 (ff0 : 'CoMod( View A ~> F @ Yoneda10_ff0 )),
      ff0 <~~ ff -> ( ff0 o>Transf_ transf @ G ) <~~ ( ff o>Transf_ transf @ G )

| CoUnitGenerated_cong :
    forall (I : obIndexer), forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F) Yoneda10_rr
      (rr : 'CoMod( F ~> View (Generating0 R) @ Yoneda10_rr )),
    forall Yoneda10_rr0 (rr0 : 'CoMod( F ~> View (Generating0 R) @ Yoneda10_rr0 )),
      rr0 <~~ rr ->
      (rr0 o>CoMod 'CoUnitGenerated @ i) <~~ (rr o>CoMod 'CoUnitGenerated @ i)

(** ----- the constant (non-recursive) conversions which are used during the
PolyTransf polymorphism elimination ----- **)

| PolyTransf_'PolyElement :
    forall Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E)
      Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
   transf (G : obGenerator) Yoneda10_ff (ff: 'CoMod( View G ~> E @ Yoneda10_ff )),
   ( PolyElement F (proj1_sig transf G (sval Yoneda10_ff G (@unitGenerator G))) )
     <~~ ( ff o>Transf_ transf @ F
         : 'CoMod( View G ~> F @ _ ) )

(** ----- the constant conversions which are used during the PolyCoMod
polymorphism elimination ----- **)
```

```
| AtIndexMor'MorCoMod_indexed :
    forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
      (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly),
    forall (Yoneda10_ff_ : forall I : obIndexer, _ )
      (ff_ : forall (I : obIndexer),
          'CoMod( AtIndexOb E_(I) ~> AtIndexOb F_(I) @ (Yoneda10_ff_(I)) )),
    forall (J : obIndexer),
      (ff_(J) : 'CoMod( AtIndexOb E_ J ~> AtIndexOb F_ J @ Yoneda10_ff_ J ) )
        <~~ (AtIndexMor (MorCoMod_indexed ff_) J
           : 'CoMod( AtIndexOb E_ J ~> AtIndexOb F_ J @ Yoneda10_ff_ J ) )

| PolyCoMod'UnitCoMod :
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
      Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G)
      Yoneda10_gg (gg : 'CoMod( F ~> G @ Yoneda10_gg )),
      gg <~~ ( gg o>CoMod ('UnitCoMod) )

| PolyCoMod_UnitCoMod :
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
      Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G)
      Yoneda10_gg (gg : 'CoMod( F ~> G @ Yoneda10_gg )),
      gg <~~ ( ('UnitCoMod) o>CoMod gg )

| PolyCoMod_PolyElement :
    forall Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E)
      Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
      Yoneda10_ff (ff : 'CoMod( E ~> F @ Yoneda10_ff )),
    forall (G : obGenerator) (e : Yoneda00_E G),
      (PolyElement F (sval Yoneda10_ff G e))
        <~~ ((PolyElement E e) o>CoMod ff
           : 'CoMod( View G ~> F @ _ ) )

| CoUnitGenerated_morphism :
    forall (I : obIndexer), forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
      forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F) Yoneda10_rr
        (rr : 'CoMod( F ~> View (Generating0 R) @ Yoneda10_rr )),
      forall Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E) Yoneda10_ff
        (ff : 'CoMod( E ~> F @ Yoneda10_ff )),
        ( (ff o>CoMod rr) o>CoMod 'CoUnitGenerated @ i )
          <~~ ( ff o>CoMod (rr o>CoMod 'CoUnitGenerated @ i) )

| Reflector_morphism :
    forall (Yoneda00_F_ : obIndexer -> _ )
      (Yoneda01_F_ : forall I : obIndexer, _ ) Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
      (Yoneda10_ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )), _ )
      (ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
      'CoMod( View (Generating0 R) ~> AtIndexOb F_(I) @ Yoneda10_ff_(I)(R)(i) ))
      (Yoneda01_F_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_F_Poly)
      (Yoneda10_ff_morphismReIndexer_morphismIndexer :
         Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
    forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
      (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
    forall Yoneda10_ee_ (ee_ : 'CoMod_( F_ ~> E_ @ Yoneda10_ee_ ) ),
    forall (Yoneda01_E_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_E_Poly)
      (Yoneda10_ee_naturalIndexer :
         Yoneda10_naturalIndexer Yoneda01_F_Poly Yoneda01_E_Poly Yoneda10_ee_ ),
    forall (J : obIndexer),
      ( AtIndexMor [[ (fun (I : obIndexer)
                          (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I ))
                        => ff_(I)(R)(i) o>CoMod AtIndexMor ee_(I) )
                         @ Yoneda01_E_Poly_functorIndexer
                       , (Reflector_morphism_morphismReIndexer_morphismIndexer
                            Yoneda10_ff_morphismReIndexer_morphismIndexer
                            Yoneda10_ee_naturalIndexer) ]]_(J) )
        <~~ ( AtIndexMor [[ ff_ @ Yoneda01_F_Poly_functorIndexer ,
   Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_(J) o>CoMod AtIndexMor ee_(J)
            : 'CoMod( AtIndexOb Generated(J) ~> AtIndexOb E_(J) @ _ ) )

| Reflector_CoUnitGenerated :
 forall (Yoneda00_F_ : obIndexer -> _ ) (Yoneda01_F_ : forall I : obIndexer, _ )
 (Yoneda01_F_Poly : forall I J : obIndexer, 'Indexer( I |- J ) -> _ )
```

```
 (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
 (Yoneda10_ff_ : forall I : obIndexer,
     forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )), _ )
 (ff_ : forall I : obIndexer, forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
       'CoMod( View (Generating0 R) ~> AtIndexOb F_(I) @ Yoneda10_ff_(I)(R)(i) ))
 (Yoneda01_F_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_F_Poly)
 (Yoneda10_ff_morphismReIndexer_morphismIndexer :
    Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
 forall I : obIndexer, forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
     forall Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E) Yoneda10_rr
       (rr : 'CoMod( E ~> View (Generating0 R) @ Yoneda10_rr )),
       ( rr o>CoMod ff_(I)(R)(i) )
         <~~ ( (rr o>CoMod 'CoUnitGenerated @ i)
                o>CoMod AtIndexMor [[ ff_ @ Yoneda01_F_Poly_functorIndexer ,
                             Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_(I)
                : 'CoMod( E ~> AtIndexOb F_(I) @ Yoneda10_PolyCoMod
                            (Yoneda10_CoUnitGenerated i Yoneda10_rr)
                            (Yoneda10_Reflector Yoneda10_ff_ I) ) )

(** ----- the constant conversions which are only for the wanted sense of
generated-functor-along-reindexing grammar ----- **)

| UnitCoModView_'PolyElement : forall (G : obGenerator),
    (PolyElement (View G) ((@unitGenerator G) : 'Generator( G ~> G)))
      <~~ (@'UnitCoMod (View G)
           : 'CoMod( View G ~> View G @ _ ) )

| CoUnitGenerated'PolyElement :
    forall (I : obIndexer) (R : obReIndexer) (i : 'Indexer( (ReIndexing0 R) |- I )),
    forall (G : obGenerator) (f : Yoneda00_View (Generating0 R) G),
      (PolyElement (AtIndexOb Generated(I))
                   (sval (Yoneda10_CoUnitGenerated_form i) G f))
        <~~ ( (PolyElement (View (Generating0 R )) f) o>CoMod 'CoUnitGenerated @ i
              : 'CoMod( View G ~> AtIndexOb Generated(I) @ _ ) )

(**MEMO: Yoneda10_CoUnitGenerated_form_morphismReIndexer_morphismIndexer is
 already present and will be masqued , such to get some more-general constructor **)
| Reflector'CoUnitGenerated : forall (I : obIndexer),
    forall Yoneda10_CoUnitGenerated_form_morphismReIndexer_morphismIndexer,
      (@'UnitCoMod (AtIndexOb Generated(I)))
        <~~ ( AtIndexMor [[ (fun (I : obIndexer)
                  (R : obReIndexer) (ri : 'Indexer( ReIndexing0 R |- I ))
          => (@'UnitCoMod (View (Generating0 R))) o>CoMod 'CoUnitGenerated @ ri)
                             @ Yoneda01_Generated_PolyIndexer_functorIndexer ,
         Yoneda10_CoUnitGenerated_form_morphismReIndexer_morphismIndexer ]]_(I)
           : 'CoMod( AtIndexOb Generated(I) ~> AtIndexOb Generated(I) @ _ ) )

(** ----- the constant conversions which are only for the confluence lemma ---- **)

(** none **)

(** ----- the constant symmetrized-conversions which are symmetrized-derivable by
using the finished cut-elimination lemma ----- TODO: COMMENT ALL THIS SECTION
----- **)


(** (**MEMO: commented now so that it non-prevent the degradation lemma *)
| PolyCoMod_morphism :
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
      Yoneda00_F' Yoneda01_F' (F' : @obCoMod Yoneda00_F' Yoneda01_F')
      Yoneda10_ff' (ff' : 'CoMod( F' ~> F @ Yoneda10_ff' )),
      forall Yoneda00_F'' Yoneda01_F'' (F'' : @obCoMod Yoneda00_F'' Yoneda01_F'')
        Yoneda10_ff_ (ff_ : 'CoMod( F'' ~> F' @ Yoneda10_ff_ )),
      forall Yoneda00_F''' Yoneda01_F''' (F''' : @obCoMod Yoneda00_F''' Yoneda01_F''')
        Yoneda10_ff__ (ff__ : 'CoMod( F''' ~> F'' @ Yoneda10_ff__ )),
        ((ff__ o>CoMod ff_) o>CoMod ff')
          <~~ (ff__ o>CoMod (ff_ o>CoMod ff'))  **)

(**MEMO: this is some lemma for the more-general [View_'PolyElement] below **)
| ViewView_'PolyElement :
 forall (H G : obGenerator) Yoneda01_ff (ff : 'CoMod( View G ~> View H @ Yoneda01_ff)),
   (PolyElement (View H)
                (sval Yoneda01_ff G (@unitGenerator G) : 'Generator( G ~> H)))
     <~~ ( ff : 'CoMod( View G ~> View H @ _ ) )

| PolyTransf_morphism :
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
```

```
      Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G)
      (transf : {transf : forall A : obGenerator, Yoneda00_F A -> Yoneda00_G A |
                 Yoneda10_natural Yoneda01_F Yoneda01_G transf})
      (A : obGenerator)
      Yoneda10_ff (ff : 'CoMod( View A ~> F @ Yoneda10_ff )),
    forall A' Yoneda10_aa (aa : 'CoMod( View A' ~> View A @ Yoneda10_aa )),
      ( (aa o>CoMod ff) o>Transf_ transf @ G )
        <~~ ( aa o>CoMod (ff o>Transf_ transf @ G) )

| PolyTransf_morphismPolyTransf :
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
      Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G)
      (transf : {transf : ( forall A : obGenerator, Yoneda00_F A -> Yoneda00_G A ) |
                 Yoneda10_natural Yoneda01_F Yoneda01_G transf})
      (A : obGenerator) Yoneda10_ff
      (ff : 'CoMod( View A ~> F @ Yoneda10_ff ))
      Yoneda00_H Yoneda01_H (H : @obCoMod Yoneda00_H Yoneda01_H)
      (transf' : {transf : ( forall A : obGenerator, Yoneda00_G A -> Yoneda00_H A ) |
                  Yoneda10_natural Yoneda01_G Yoneda01_H transf}),
      (ff o>Transf_ (PolyTransf_morphismPolyTransf_transf transf transf') @ H)
        <~~ ((ff o>Transf_ transf @ G) o>Transf_ transf' @ H)

| View_'PolyElement :
    forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
      (G : obGenerator) Yoneda01_ff (ff : 'CoMod( View G ~> F @ Yoneda01_ff)),
      (PolyElement F (sval Yoneda01_ff G (@unitGenerator G) : Yoneda00_F G))
        <~~ (ff : 'CoMod( View G ~> F @ _ ))

| CoUnitGenerated_morphismReIndexer_morphismIndexer :
  forall (I : obIndexer) (R : obReIndexer) (ri : 'Indexer( (ReIndexing0 R) |- I )),
  forall (S : obReIndexer) (sr : 'ReIndexer( S |- R )),
  forall (J : obIndexer) (ij : 'Indexer( I |- J )),
  forall (G : obGenerator) Yoneda10_ff
    (ff : 'CoMod( View G ~> View (Generating0 S) @ Yoneda10_ff)),
    ( ff o>CoMod 'CoUnitGenerated
                 @ ((ReIndexing1 sr o>Indexer ri) o>Indexer ij) )
    <~~ ( ( ( ff o>CoMod (PolyElement (View (Generating0 R)) (Generating1 sr)) )
            o>CoMod 'CoUnitGenerated @ ri )
          o>Transf_ (Yoneda01_Generated_PolyIndexer ij)
        : 'CoMod( View G ~> AtIndexOb Generated(J) @ _ ) )

| Reflector_naturalIndexer :
    forall (Yoneda00_F_ : obIndexer -> _ )
      (Yoneda01_F_ : forall I : obIndexer, _) Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
      (Yoneda10_ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )), _ )
      (ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
     'CoMod( View (Generating0 R) ~> AtIndexOb F_(I) @ (Yoneda10_ff_(I)(R)(i)) ))
      (Yoneda01_F_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_F_Poly)
      (Yoneda10_ff_morphismReIndexer_morphismIndexer :
         Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
    forall (I J : obIndexer) (j : 'Indexer( I |- J )),
    forall (G : obGenerator) Yoneda10_ii
      (ii : 'CoMod( View G ~> AtIndexOb Generated(I) @ Yoneda10_ii )),
      ( ( ii o>CoMod AtIndexMor [[ ff_ @ Yoneda01_F_Poly_functorIndexer ,
                   Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_(I) )
          o>Transf_ (Yoneda01_F_Poly _ _ j) @ (AtIndexOb F_(J)) )
        <~~ ( ( ii o>Transf_
                   (Yoneda01_Generated_PolyIndexer j) @ (AtIndexOb Generated(J)) )
              o>CoMod AtIndexMor [[ ff_ @ Yoneda01_F_Poly_functorIndexer ,
                   Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_(J)
            : 'CoMod( View G ~> AtIndexOb F_(J) @ _ ) )

where "gg0 <~~ gg" := (@convCoMod _ _ _ _ _ _ _ gg _ gg0)

with convCoMod_indexed (**memo: non-recursive **) :
       forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
         (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
       forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
         (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly),
       forall Yoneda10_ff_ ( ff_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff_ ) ),
       forall Yoneda10_ff0_ ( ff0_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff0_ ) ), Prop :=

(**  ----- the congruences conversions for indexed morphisms -----  **)
```

```
| MorCoMod_indexed_cong (**memo: some form of extensionality *):
    forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
      (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
    forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly),
    forall (Yoneda10_ff_ : forall I : obIndexer, _ )
      (ff_ : forall (I : obIndexer),
          'CoMod( AtIndexOb E_(I) ~> AtIndexOb F_(I) @ (Yoneda10_ff_(I)) ) ),
    forall (Yoneda10_ff0_ : forall I : obIndexer, _ )
      (ff0_ : forall (I : obIndexer),
          'CoMod( AtIndexOb E_(I) ~> AtIndexOb F_(I) @ (Yoneda10_ff0_(I)) ) ),
      (forall I : obIndexer, ff0_(I) <~~ ff_(I)) ->
      ( MorCoMod_indexed ff0_ ) <~~_ ( MorCoMod_indexed ff_ )

| Reflector_cong :
    forall (Yoneda00_F_ : obIndexer -> _ )
      (Yoneda01_F_ : forall I : obIndexer, _ ) Yoneda01_F_Poly
      (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
      (Yoneda10_ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )), _ )
      (ff_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
      'CoMod( View (Generating0 R) ~> AtIndexOb F_(I) @ (Yoneda10_ff_(I)(R)(i)) ))
      (Yoneda01_F_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_F_Poly)
      (Yoneda10_ff_morphismReIndexer_morphismIndexer :
         Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
    forall (Yoneda10_ff0_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )), _ )
      (ff0_ : forall (I : obIndexer),
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
    'CoMod( View (Generating0 R) ~> AtIndexOb F_(I) @ (Yoneda10_ff0_(I)(R)(i)) ))
      (Yoneda10_ff0_morphismReIndexer_morphismIndexer :
         Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff0_),
      ( forall (I : obIndexer)
        (**memo: conversion is allowed at every [I] simultaneously **) ,
          forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
            ff0_(I)(R)(i) <~~ ff_(I)(R)(i) ) ->
      ( [[ ff0_ @ Yoneda01_F_Poly_functorIndexer ,
           Yoneda10_ff0_morphismReIndexer_morphismIndexer ]]_ )
        <~~_ ( [[ ff_ @ Yoneda01_F_Poly_functorIndexer ,
                Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_ )

where "gg0_ <~~_ gg_" := (@convCoMod_indexed _ _ _ _ _ _ _ _ _ gg_ _ gg0_).

Hint Constructors convCoMod.
Hint Constructors convCoMod_indexed.

Scheme convCoMod_convCoMod_indexed_ind :=
  Induction for convCoMod Sort Prop
  with convCoMod_indexed_convCoMod_ind :=
    Induction for convCoMod_indexed Sort Prop.
Combined Scheme convCoMod_convCoMod_indexed_mutind from
         convCoMod_convCoMod_indexed_ind, convCoMod_indexed_convCoMod_ind.

Section SomeInstances.

(** this lemma formulation is only for some PolyElement input .. no generalization
, which would be derivable by using the finished cut-elimination lemma **)
Lemma CoUnitGenerated_morphismReIndexer_morphismIndexer_PolyElement_ALT :
  forall (I : obIndexer) (R : obReIndexer) (ri : 'Indexer( (ReIndexing0 R) |- I )),
  forall (S : obReIndexer) (sr : 'ReIndexer( S |- R )),
  forall (J : obIndexer) (ij : 'Indexer( I |- J )),
  forall (G : obGenerator) (f : Yoneda00_View (Generating0 S) G),
    ( PolyElement (AtIndexOb Generated(J))
                  (sval (Yoneda01_Generated_PolyIndexer ij) G
                        (sval (Yoneda10_CoUnitGenerated_form ri) G
                   (sval (Yoneda10_ViewGenerating_PolyReIndexer_form sr) G f))) )
      <~~ ( (PolyElement (View (Generating0 S)) f)
            o>CoMod 'CoUnitGenerated
                   @ ((ReIndexing1 sr o>Indexer ri) o>Indexer ij)
          : 'CoMod( View G ~> AtIndexOb Generated(J) @ _ ) ) .
Proof.
  intros. eapply convCoMod_Trans; first by exact: CoUnitGenerated'PolyElement.
  rewrite Yoneda10_CoUnitGenerated_form_morphismReIndexer_morphismIndexer.
  exact: convCoMod_Refl.
Qed.
```

```
Hypothesis Hyp_convCoMod_Sym :
  forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
  forall Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G),
  forall Yoneda10_gg (gg : 'CoMod( F ~> G @ Yoneda10_gg ) ),
  forall Yoneda10_gg0 (gg0 : 'CoMod( F ~> G @ Yoneda10_gg0 ) ),
    ( gg0 <~~ gg ) -> ( gg <~~ gg0 ).

(** memo: the more-general (than [(PolyElement (View (Generating0 S)) f)] input)
lemma which will be derivable by using the finished cut-elimination lemma **)
Lemma CoUnitGenerated_morphismReIndexer_morphismIndexer_PolyElement :
  forall (I : obIndexer) (R : obReIndexer) (ri : 'Indexer( (ReIndexing0 R) |- I )),
  forall (S : obReIndexer) (sr : 'ReIndexer( S |- R )),
  forall (J : obIndexer) (ij : 'Indexer( I |- J )),
  forall (G : obGenerator) (f : Yoneda00_View (Generating0 S) G),
    ( (PolyElement (View (Generating0 S)) f)
        o>CoMod 'CoUnitGenerated
                @ ((ReIndexing1 sr o>Indexer ri) o>Indexer ij) )
      <~~ ( ( ( (PolyElement (View (Generating0 S)) f)
                o>CoMod (PolyElement (View (Generating0 R)) (Generating1 sr)) )
              o>CoMod 'CoUnitGenerated @ ri )
            o>Transf_ (Yoneda01_Generated_PolyIndexer ij)
          : 'CoMod( View G ~> AtIndexOb Generated(J) @ _ ) ) .
Proof.
  intros. eapply convCoMod_Trans.
  - { eapply convCoMod_Trans;
      first by eapply PolyTransf_cong, CoUnitGenerated_cong, PolyCoMod_PolyElement.
      eapply convCoMod_Trans;
        first by eapply PolyTransf_cong, CoUnitGenerated'PolyElement.
      exact: PolyTransf_'PolyElement. }
  - apply: Hyp_convCoMod_Sym.
    { eapply convCoMod_Trans; first by exact: CoUnitGenerated'PolyElement.
      rewrite Yoneda10_CoUnitGenerated_form_morphismReIndexer_morphismIndexer.
      simpl. rewrite -polyGenerator_unitGenerator. exact: convCoMod_Refl. }
Qed.

(** memo: the more-general (than [(PolyElement (View (Generating0 S)) f)] input)
lemma will be derivable by using the finished cut-elimination lemma **)
Lemma Reflector_naturalIndexer_PolyElement :
  forall (Yoneda00_F_ : obIndexer -> _)
    (Yoneda01_F_ : forall I : obIndexer, _) Yoneda01_F_Poly
    (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
    (Yoneda10_ff_ : forall (I : obIndexer),
        forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )), _ )
    (ff_ : forall (I : obIndexer),
        forall (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
      'CoMod( View (Generating0 R) ~> AtIndexOb F_(I) @ (Yoneda10_ff_ _ _ (i)) ))
    (Yoneda01_F_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_F_Poly)
    (Yoneda10_ff_morphismReIndexer_morphismIndexer :
      Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
  forall (I J : obIndexer) (j : 'Indexer( I |- J )),
  forall (G : obGenerator) (ii : Yoneda00_Generated(I) G),
    ( ( (PolyElement (AtIndexOb Generated(I)) ii)
          o>CoMod AtIndexMor [[ ff_ @ Yoneda01_F_Poly_functorIndexer ,
                          Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_ I )
        o>Transf_ (Yoneda01_F_Poly _ _ j) @ (AtIndexOb F_(J)) )
      <~~ ( ( (PolyElement (AtIndexOb Generated(I)) ii)
        o>Transf_ (Yoneda01_Generated_PolyIndexer j) @ (AtIndexOb Generated(J)) )
            o>CoMod AtIndexMor [[ ff_ @ Yoneda01_F_Poly_functorIndexer ,
                          Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_ J
          : 'CoMod( View G ~> AtIndexOb F_(J) @ _ ) ).
Proof.
  intros. eapply convCoMod_Trans.
  - { eapply convCoMod_Trans; first by eapply PolyCoMod_cong;
      [exact: PolyTransf_'PolyElement | exact: convCoMod_Refl].
      eapply convCoMod_Trans; first by exact: PolyCoMod_PolyElement.
      simpl. rewrite
      (Yoneda10_morphismReIndexer_morphismIndexer_to_Yoneda10_morphismIndexerOnly
         Yoneda10_ff_morphismReIndexer_morphismIndexer). exact: convCoMod_Refl.
    }
  - apply: Hyp_convCoMod_Sym.
    { eapply convCoMod_Trans;
        first by eapply PolyTransf_cong, PolyCoMod_PolyElement.
      simpl. rewrite -(proj2_sig (Yoneda10_ff_ _ _ _)).
      exact: PolyTransf_'PolyElement.
    }
Qed.
```

```
End SomeInstances.
```

5.2 Same sense-decoding for convertible morphisms

Because of the embedded sense-decoding extra-indexes/arguments in the datatype-families [morCoMod] [morCoMod_indexed] of the morphisms , the conversion-relation shall convert across two morphisms whose sense-decoding datatype-indexes/arguments are not syntactically/grammatically-the-same . But oneself does show that , by logical-deduction [convCoMod_sense] , these two sense-decodings are indeed propositionally equal ("soundness lemma") .

Memo that the lemma [convCoMod_sense] will only be used during the polymorphism/cut-elimination resolution to show the property [Yoneda10_morphismReIndexer_morphismIndexer] (polyarrowing of some cocone across the reindexer and across the indexer) of the proposed output solution-morphisms , in the 2 cases when the input morphism is [((AtIndexMor [[ggSol_]]_ J) o>CoMod (AtIndexMor [[ffSol_]]_ J))] or the input morphism is [([[ff_ @ I]])] .

```
(**memo: none such [Yoneda01_F_Poly_functorIndexer] or [Yoneda10_ff_morphismReIndexer]
   are used to show [convCoMod_sense_mut] **)
Lemma convCoMod_sense_mut :
  (forall Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E),
      forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
      forall Yoneda10_ff (ff : 'CoMod( E ~> F @ Yoneda10_ff )),
      forall Yoneda10_ff0 (ff0 : 'CoMod( E ~> F @ Yoneda10_ff0 )),
        ff0 <~~ ff -> forall G' : obGenerator,
          (proj1_sig Yoneda10_ff G') =1 (proj1_sig Yoneda10_ff0 G')) /\
  (forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
     (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly)
     Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
     (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
     Yoneda10_ff_ (ff_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff_ ))
     Yoneda10_ff0_ (ff0_ : 'CoMod_( E_ ~> F_  @ Yoneda10_ff0_ )),
      ff0_ <~~_ ff_ -> forall I : obIndexer, forall G' : obGenerator,
          (proj1_sig (Yoneda10_ff_(I)) G') =1 (proj1_sig (Yoneda10_ff0_(I)) G')).
Proof.
  apply convCoMod_convCoMod_indexed_mutind.

  (**  ----- the total/(multi-step) conversions -----  **)
  - (* convCoMod_Refl *)  intros. move. intros f. reflexivity.
  - (* convCoMod_Trans *)  intros until 1. intros gg_eq .
    intros until 1. intros uTrans_eq.
    intros. move. intros f. rewrite gg_eq uTrans_eq . reflexivity.

  (**  ----- the congruences (recursive) conversions for singleton morphisms -----  **)
  - (* AtIndexMor_cong *) intros until 1. intros Heq. exact: Heq.
  - (* PolyCoMod_cong *)  intros until 1. intros ff__eq .
    intros ? ff'_eq ? . move. intros f'.
    rewrite /Yoneda10_PolyCoMod /= . rewrite ff__eq ff'_eq. reflexivity.
  - (* PolyTransf_cong *)  intros until 2. intros ff_eq . intros. move. intros a.
    simpl. (* rewrite /Yoneda10_PolyTransf /= . *) rewrite ff_eq. reflexivity.
  - (* CoUnitGenerated_cong *)  intros until 1. intros rr_eq .
    intros. move. intros f.  simpl. rewrite rr_eq. reflexivity.

  (** ----- the constant (non-recursive) conversions which are used during the
  PolyTransf polymorphism elimination ----- **)
  - (* PolyTransf_'PolyElement *) intros. move. intros g. simpl.
    rewrite [RHS](proj2_sig transf).
    rewrite [in RHS](proj2_sig Yoneda10_ff). simpl.
    rewrite -[in RHS]unitGenerator_polyGenerator. reflexivity.

  (** ----- the constant conversions which are used during the PolyCoMod
  polymorphism elimination ----- **)
  - (* AtIndexMor'MorCoMod_indexed *)
    intros. move. intros j. simpl. reflexivity.
  - (* PolyCoMod'UnitCoMod *) intros. move. intros f. simpl.  reflexivity.
  - (* PolyCoMod_UnitCoMod *) intros. move. intros f. simpl.  reflexivity.
  - (* PolyCoMod_PolyElement *) intros. move. intros g. simpl.
    symmetry. exact: (proj2_sig Yoneda10_ff).
  - (* CoUnitGenerated_morphism *) intros. move. intros e. simpl. reflexivity.
  - (* Reflector_morphism *) intros. move. intros jj. simpl. reflexivity.
  - (* Reflector_CoUnitGenerated *) intros. move. intros e. simpl.
    rewrite -[in LHS]polyIndexer_unitIndexer . reflexivity.

  (** ----- the constant conversions which are only for the wanted sense of
  generated-functor-along-reindexing grammar ----- **)
```

```
  - (* UnitCoModView_'PolyElement *)
    intros. move. intros g. simpl. exact: unitGenerator_polyGenerator.
  - (* CoUnitGenerated'PolyElement  *)
    intros. move. intros g. simpl. reflexivity.
  - (* Reflector'CoUnitGenerated *)
    intros. move. intros i. simpl. rewrite -[in LHS]polyIndexer_unitIndexer.
    destruct i as [ [? ?] ?]; reflexivity.

  (** ----- the constant symmetrized-conversions which are symmetrized-derivable
  by using the finished cut-elimination lemma ----- **)
  (**  - (* PolyCoMod_morphism *) intros. move. intros f.
    reflexivity (* associativity of function composition *). **)
  - (* ViewView_'PolyElement *)
    intros. move. intros g. simpl.
    rewrite [g in LHS]unitGenerator_polyGenerator.
    rewrite -[in LHS](proj2_sig Yoneda01_ff). simpl. reflexivity.
  - (* PolyTransf_morphism *)  intros. move. intros g. reflexivity.
  - (* PolyTransf_morphismPolyTransf *) intros. move. intros g. reflexivity.
  - (* View_'PolyElement *)
    intros. move. intros g. simpl. rewrite [g in LHS]unitGenerator_polyGenerator.
    rewrite -[in LHS](proj2_sig Yoneda01_ff). simpl. reflexivity.
  - (* CoUnitGenerated_morphismReIndexer_morphismIndexer *)
    intros. move. intros g. simpl.
    rewrite [RHS]Yoneda10_CoUnitGenerated_form_morphismReIndexer_morphismIndexer.
    simpl. reflexivity.
  - (* Reflector_naturalIndexer *)
    intros. move. intros g. simpl. exact:
   (Yoneda10_morphismReIndexer_morphismIndexer_to_Yoneda10_morphismIndexerOnly
      Yoneda10_ff_morphismReIndexer_morphismIndexer).

  (**  ----- the congruences conversions for indexed morphisms -----  **)
  - (* MorCoMod_indexed_cong *) intros until 1. intros Heq. exact: Heq.
  - (*  Reflector_cong *) intros until 4. intros ff_eq . intros. move. intros ii.
    simpl. exact: ff_eq.
Qed.

Definition convCoMod_sense := proj1 convCoMod_sense_mut.
Definition convCoMod_sense_indexed := proj2 convCoMod_sense_mut.
```

5.3 Linear total/asymptotic grade and the degradation lemma

Memo that the grade of the reflector-constructor [Reflector] is defined as the maximum of all the section-morphisms of the input cocone ; therefore this maximum is indeed taken over all the indexer-indexes and reindexer-reindexes which are refering/indexing to these (section-)morphisms .

Moreover to facilitate the COQ automatic-arithmetic during the degradation lemma , here oneself has presented the predicate [Inductive is_MorIndexer12_] such to force/assume [Axiom is_MorIndexer12_allP] the finiteness of this graph [{ R : obReIndexer & 'Indexer(ReIndexing0 R |- (I : obIndexer)) }] of common-interest ; also some other finiteness is forced/assumed [Axiom is_ObIndexer12_allP] on the indexes of the indexer graph [obIndexer] . Clearly this ongoing COQ program and deduction will still-proceed when those things are confined less than any regular cardinal .

Elsewhere , memo that if the conversion-relation constructor [convCoMod_Refl] was absent , then oneself would get some degradation lemma with tight/strict less-than : [(grade ff0 < grade ff)] ; this is the tight/strict-degrading which will occur during the polymorphism/cut-elimination resolution (by the automatic-arithmetic-tactic calls therein) .

```
Notation max m n := ((Nat.add m (Nat.sub n m))%coq_nat).
Arguments Nat.sub : simpl nomatch.
Arguments Nat.add : simpl nomatch.

Definition grade_mut :
  (forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
     Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G)
     Yoneda10_gg (gg : 'CoMod( F ~> G @ Yoneda10_gg ) ), nat ) *
  (forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
     (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
     Yoneda00_G_ Yoneda01_G_ Yoneda01_G_Poly
     (G_ : @obCoMod_indexed Yoneda00_G_ Yoneda01_G_ Yoneda01_G_Poly)
     Yoneda10_gg_ (gg_ : 'CoMod_( F_ ~> G_ @ Yoneda10_gg_ ) ), nat ).
Proof.
  apply morCoMod_morCoMod_indexed_mutrect.
  - (* AtIndexMor *) intros ? ? ? ? ? ? ? ? ? ?  ff_ IHff_ (*I*) _ .
    exact: (S IHff_).
  - (* PolyCoMod *) intros ? ? F ? ? F' ? ff' IHff' ? ? F'' ? ff_ IHff_ .
    exact: ( 2 * (S (IHff' + IHff_)%coq_nat ) )%coq_nat .
```

```
  - (* PolyTransf *) intros ? ?  F ? ? G transf A ? ff IHff .
    exact: (S IHff).
  - (* UnitCoMod *) intros ? ? F .
    exact: (S ( (* gradeOb F = *) O )).
  - (* PolyElement *) intros ? ? F ? f .
    exact: (S ( O (* = grade? *) )).
  - (* CoUnitGenerated *) intros ? ? ? ? ? ? ? rr IHrr.
    exact: (S (S IHrr)).
  - (* MorCoMod_indexed *) intros ? ? ? ? ? ? ? ? ? ff_ IHff_ .
    exact: (S (max (IHff_ ObIndexer1) (IHff_ ObIndexer2) )).
  - (* Reflector *) intros ? ? ? F_ ? ff_ IHff_ ? ? .
    exact: (S (S (max
    (max (IHff_(ObIndexer1)(ObReIndexer1_ ObIndexer1)(MorIndexer1_ ObIndexer1))
         (IHff_(ObIndexer1)(ObReIndexer2_ ObIndexer1)(MorIndexer2_ ObIndexer1)))
    (max (IHff_(ObIndexer2)(ObReIndexer1_ ObIndexer2)(MorIndexer1_ ObIndexer2))
        (IHff_(ObIndexer2)(ObReIndexer2_ ObIndexer2)(MorIndexer2_ ObIndexer2)))))).
Defined.

Definition grade := fst grade_mut.
Definition grade_indexed := snd grade_mut.
Arguments grade : simpl nomatch.
Arguments grade_indexed : simpl nomatch.

Lemma grade_mut_AtIndexMor :
  forall Yoneda00_E_  Yoneda01_E_ Yoneda01_E_Poly
    (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly)
    Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
    (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
    Yoneda10_ff_ (ff_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff_ )),
    forall I : obIndexer, grade (AtIndexMor ff_ I) = S (grade_indexed ff_).
Proof. reflexivity. Qed.

Lemma grade_mut_PolyCoMod :
    forall Yoneda00_F Yoneda01_F  (F : @obCoMod Yoneda00_F Yoneda01_F)
      Yoneda00_F' Yoneda01_F' (F' : @obCoMod Yoneda00_F' Yoneda01_F')
      Yoneda10_ff' (ff' : 'CoMod( F' ~> F @ Yoneda10_ff' ))
      Yoneda00_F'' Yoneda01_F'' (F'' : @obCoMod Yoneda00_F'' Yoneda01_F'')
      Yoneda10_ff_ (ff_ : 'CoMod( F'' ~> F' @ Yoneda10_ff_ )),
  grade (ff_ o>CoMod ff') = ( 2 * (S (grade ff' + grade ff_)%coq_nat ) )%coq_nat.
Proof. reflexivity. Qed.

Lemma grade_mut_PolyTransf :
  forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
    Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G) transf,
    forall (A : obGenerator) Yoneda10_ff (ff : 'CoMod( View A ~> F @ Yoneda10_ff )),
      grade (ff o>Transf_ transf @ G) = (S (grade ff) )%coq_nat .
Proof. reflexivity. Qed.

Lemma grade_mut_UnitCoMod :
  forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
    grade (@'UnitCoMod F) = (S (O) )%coq_nat .
Proof. reflexivity. Qed.

Lemma grade_mut_PolyElement :
  forall Yoneda00_F Yoneda01_F (F: @obCoMod Yoneda00_F Yoneda01_F)
         (G : obGenerator) (f : Yoneda00_F G),
    grade (PolyElement F f) = (S (O) )%coq_nat .
Proof. reflexivity. Qed.

Lemma grade_mut_CoUnitGenerated :
 forall (I : obIndexer) (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
 forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
        Yoneda10_rr (rr : 'CoMod( F ~> View (Generating0 R) @ Yoneda10_rr )),
    grade (rr o>CoMod 'CoUnitGenerated @ i) = (S (S (grade rr) ))%coq_nat .
Proof. reflexivity. Qed.

Lemma grade_mut_MorCoMod_indexed :
  forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
    (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly)
    Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
    (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly) Yoneda10_ff_
    (ff_ : forall I : obIndexer,
        'CoMod( AtIndexOb E_ I ~> AtIndexOb F_ I @ Yoneda10_ff_ I )),
    grade_indexed (MorCoMod_indexed ff_)
    = (S (max (grade(ff_ ObIndexer1)) (grade(ff_ ObIndexer2)))) .
Proof. reflexivity. Qed.
```

```
Lemma grade_mut_Reflector :
  forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
    (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly) Yoneda10_ff_
    (ff_ : (forall (I : obIndexer)
             (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
         'CoMod( View (Generating0 R) ~> AtIndexOb F_ I @ Yoneda10_ff_ I R i ))),
  forall (Yoneda01_F_Poly_functorIndexer : Yoneda10_functorIndexer Yoneda01_F_Poly)
    (Yoneda10_ff_morphismReIndexer_morphismIndexer :
       Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ff_),
    grade_indexed ([[ ff_ @ Yoneda01_F_Poly_functorIndexer ,
                      Yoneda10_ff_morphismReIndexer_morphismIndexer ]]_)
 = (S (S (max
 (max (grade (ff_(ObIndexer1)(ObReIndexer1_ ObIndexer1)(MorIndexer1_ ObIndexer1)))
      (grade (ff_(ObIndexer1)(ObReIndexer2_ ObIndexer1)(MorIndexer2_ ObIndexer1))))
 (max (grade (ff_(ObIndexer2)(ObReIndexer1_ ObIndexer2)(MorIndexer1_ ObIndexer2)))
  (grade (ff_(ObIndexer2)(ObReIndexer2_ ObIndexer2)(MorIndexer2_ ObIndexer2))))))).
Proof. reflexivity. Qed.

Definition grade_rewrite :=
  (grade_mut_AtIndexMor, grade_mut_PolyCoMod, grade_mut_PolyTransf,
   grade_mut_UnitCoMod, grade_mut_PolyElement, grade_mut_CoUnitGenerated,
   grade_mut_MorCoMod_indexed, grade_mut_Reflector).

Ltac tac_indexed_all :=
  repeat match goal with
         | [ ri : 'Indexer( ReIndexing0 _ |- ?I )
             |- context [max _ _] ] => destruct (is_ObIndexer12_allP I);
                                     destruct (is_MorIndexer12_allP ri)
         | (* after above match *)
         [ I : obIndexer
           |- context [max _ _] ] => destruct (is_ObIndexer12_allP I)
         | [ Hgrade : (forall (I : obIndexer),
                          ( _ <= _ )%coq_nat) |- context [max _ _] ] =>
           move: {Hgrade} (Hgrade ObIndexer2) (Hgrade ObIndexer1);
           rewrite ?grade_rewrite
         | [ Hgrade : (forall (I : obIndexer)
                         (R : obReIndexer) (i : 'Indexer( ReIndexing0 R |- I )),
                          ( _ <= _ )%coq_nat) |- context [max _ _] ] =>
           move: {Hgrade} (Hgrade ObIndexer2 _ (MorIndexer2_ ObIndexer2))
                          (Hgrade ObIndexer2 _ (MorIndexer1_ ObIndexer2))
                          (Hgrade ObIndexer1 _ (MorIndexer2_ ObIndexer1))
                          (Hgrade ObIndexer1 _ (MorIndexer1_ ObIndexer1));
           rewrite ?grade_rewrite
         end.

Lemma grade_mut_gt0 :
  (forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
     Yoneda00_G Yoneda01_G (G : @obCoMod Yoneda00_G Yoneda01_G)
     Yoneda10_gg (gg : 'CoMod( F ~> G @ Yoneda10_gg ) ),
      ((S O) <= (grade gg))%coq_nat ) /\
  (forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
     (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
     Yoneda00_G_ Yoneda01_G_ Yoneda01_G_Poly
     (G_ : @obCoMod_indexed Yoneda00_G_ Yoneda01_G_ Yoneda01_G_Poly)
     Yoneda10_gg_ (gg_ : 'CoMod_( F_ ~> G_ @ Yoneda10_gg_ ) ),
      ((S O) <= (grade_indexed gg_))%coq_nat ).
Proof.
  apply morCoMod_morCoMod_indexed_mutind;
    intros; rewrite ?grade_rewrite; tac_indexed_all; intros; abstract Psatz.lia.
Qed.

Definition grade_gt0 := fst grade_mut_gt0.
Definition grade_indexed_gt0 := snd grade_mut_gt0.

Ltac tac_grade_gt0 :=
  match goal with
  | [ gg1 : 'CoMod( _ ~> _ @ _ ) ,
            gg2 : 'CoMod( _ ~> _ @ _ ) ,
                  gg3 : 'CoMod( _ ~> _ @ _ ) ,
                        gg4 : 'CoMod( _ ~> _ @ _ ) |- _ ] =>
    move : (@grade_gt0 _ _ _ _ _ _ _ gg1) (@grade_gt0 _ _ _ _ _ _ _ gg2)
                                          (@grade_gt0 _ _ _ _ _ _ _ gg3)
                                          (@grade_gt0 _ _ _ _ _ _ _ gg4)
  | [ gg1 : 'CoMod( _ ~> _ @ _ ) ,
            gg2 : 'CoMod( _ ~> _ @ _ ) ,
                  gg3 : 'CoMod( _ ~> _ @ _ ) |- _ ] =>
    move : (@grade_gt0 _ _ _ _ _ _ _ gg1) (@grade_gt0 _ _ _ _ _ _ _ gg2)
```

```
                                      (@grade_gt0 _ _ _ _ _ _ _ gg3)
  | [ gg1 : 'CoMod( _ ~> _ @ _ ) ,
            gg2 : 'CoMod( _ ~> _ @ _ )  |- _ ] =>
    move : (@grade_gt0 _ _ _ _ _ _ _ gg1) (@grade_gt0 _ _ _ _ _ _ _ gg2)
  | [ gg1 : 'CoMod( _ ~> _ @ _ )  |- _ ] =>
    move : (@grade_gt0 _ _ _ _ _ _ _ gg1)
  end.

Ltac tac_grade_indexed_gt0 :=
  match goal with
  | [ gg1 : 'CoMod_( _ ~> _ @ _ ) ,
            gg2 : 'CoMod_( _ ~> _ @ _ ) ,
                  gg3 : 'CoMod_( _ ~> _ @ _ ) ,
                        gg4 : 'CoMod_( _ ~> _ @ _ ) |- _ ] =>
    move : (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ gg1)
             (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ gg2)
             (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ gg3)
             (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ gg4)
  | [ gg1 : 'CoMod_( _ ~> _ @ _ ) ,
            gg2 : 'CoMod_( _ ~> _ @ _ ) ,
                  gg3 : 'CoMod_( _ ~> _ @ _ ) |- _ ] =>
    move : (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ gg1)
             (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ gg2)
             (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ gg3)
  | [ gg1 : 'CoMod_( _ ~> _ @ _ ) ,
            gg2 : 'CoMod_( _ ~> _ @ _ )  |- _ ] =>
    move : (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ gg1)
             (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ gg2)
  | [ gg1 : 'CoMod_( _ ~> _ @ _ )  |- _ ] =>
    move : (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ gg1)
  end.

Ltac tac_grade_indexed_gt0_indexing :=
  match goal with
  | [ gg1 : (forall I : obIndexer, 'CoMod_( _ ~> _ @ _ )) ,
            gg2 : (forall I : obIndexer, 'CoMod_( _ ~> _ @ _ ))  |- _ ] =>
    move : (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ (gg1 ObIndexer1))
             (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ (gg1 ObIndexer2))
             (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ (gg2 ObIndexer1))
             (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ (gg2 ObIndexer2))
  | [ gg1 : (forall I : obIndexer, 'CoMod_( _ ~> _ @ _ ))  |- _ ] =>
    move : (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ (gg1 ObIndexer1))
             (@grade_indexed_gt0 _ _ _ _ _ _ _ _ _ (gg1 ObIndexer2))
  end.

Ltac tac_grade_gt0_indexing :=
match goal with
| [ gg1 : (forall I R (i : 'Indexer( ReIndexing0 R |- I )), 'CoMod( _ ~> _ @ _ )),
          gg2 : (forall I R (i : 'Indexer( ReIndexing0 R |- I )), 'CoMod( _ ~> _ @ _ ))
    |- _ ] => move:
  (@grade_gt0 _ _ _ _ _ _ _
           (gg1(ObIndexer1)(ObReIndexer1_ ObIndexer1)(MorIndexer1_ ObIndexer1)))
    (@grade_gt0 _ _ _ _ _ _ _
           (gg1(ObIndexer1)(ObReIndexer2_ ObIndexer1)(MorIndexer2_ ObIndexer1)))
    (@grade_gt0 _ _ _ _ _ _ _
           (gg1(ObIndexer2)(ObReIndexer1_ ObIndexer2)(MorIndexer1_ ObIndexer2)))
    (@grade_gt0 _ _ _ _ _ _ _
           (gg1(ObIndexer2)(ObReIndexer2_ ObIndexer2)(MorIndexer2_ ObIndexer2)))
    (@grade_gt0 _ _ _ _ _ _ _
           (gg2(ObIndexer1)(ObReIndexer1_ ObIndexer1)(MorIndexer1_ ObIndexer1)))
    (@grade_gt0 _ _ _ _ _ _ _
           (gg2(ObIndexer1)(ObReIndexer2_ ObIndexer1)(MorIndexer2_ ObIndexer1)))
    (@grade_gt0 _ _ _ _ _ _ _
           (gg2(ObIndexer2)(ObReIndexer1_ ObIndexer2)(MorIndexer1_ ObIndexer2)))
    (@grade_gt0 _ _ _ _ _ _ _
           (gg2(ObIndexer2)(ObReIndexer2_ ObIndexer2)(MorIndexer2_ ObIndexer2)))
| [ gg1 : (forall I R (i : 'Indexer( ReIndexing0 R |- I )), 'CoMod( _ ~> _ @ _ ))
    |- _ ] => move :
    (@grade_gt0 _ _ _ _ _ _ _
    (gg1(ObIndexer1)(ObReIndexer1_ ObIndexer1)(MorIndexer1_ ObIndexer1)))
    (@grade_gt0 _ _ _ _ _ _ _
    (gg1(ObIndexer1)(ObReIndexer2_ ObIndexer1)(MorIndexer2_ ObIndexer1)))
    (@grade_gt0 _ _ _ _ _ _ _
    (gg1(ObIndexer2)(ObReIndexer1_ ObIndexer2)(MorIndexer1_ ObIndexer2)))
    (@grade_gt0 _ _ _ _ _ _ _
    (gg1(ObIndexer2)(ObReIndexer2_ ObIndexer2)(MorIndexer2_ ObIndexer2)))
end.
```

```
Lemma degrade_mut :
  ( forall Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E),
      forall Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F),
      forall Yoneda10_ff ( ff : 'CoMod( E ~> F @ Yoneda10_ff ) ),
      forall Yoneda10_ff0 ( ff0 : 'CoMod( E ~> F @ Yoneda10_ff0 ) ),
        ff0 <~~ ff -> ( grade ff0 <= grade ff )%coq_nat ) /\
  ( forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
           (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly),
      forall Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
            (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly),
      forall Yoneda10_ff_ ( ff_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff_ ) ),
      forall Yoneda10_ff0_ ( ff0_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff0_ ) ),
        ff0_ <~~_ ff_ -> ( grade_indexed ff0_ <= grade_indexed ff_ )%coq_nat ).
Proof.
  Time apply convCoMod_convCoMod_indexed_mutind;
    try solve [intros; rewrite ?grade_rewrite;
               try tac_grade_gt0; try tac_grade_indexed_gt0;
                 try tac_grade_gt0_indexing; try tac_grade_indexed_gt0_indexing;
                   tac_indexed_all;
                   intros; abstract Psatz.lia].
Qed. (* /!\ LONG TIME 47s *)

Definition degrade := proj1 degrade_mut.
Definition degrade_indexed := proj2 degrade_mut.

Ltac tac_degrade_mut H_grade :=
  intuition idtac;
  repeat match goal with
         | [ Hred : ( _ <~~ _ )%poly |- _ ] =>
           move : (degrade Hred) ; clear Hred
         | [ Hred : ( _ <~~_ _ )%poly |- _ ] =>
           move : (degrade_indexed Hred) ; clear Hred
         | [ Hred : (forall (I : obIndexer),
                       ( _ <~~ _ )%poly) |- _ ] =>
           move: {Hred} (degrade (Hred ObIndexer2)) (degrade (Hred ObIndexer1))
         | [ Hred : (forall I R (i : 'Indexer( ReIndexing0 R |- I )),
                          ( _ <~~ _ )%poly) |- _ ] =>
           move: {Hred} (degrade (Hred ObIndexer2 _ (MorIndexer2_ ObIndexer2)))
                          (degrade (Hred ObIndexer2 _ (MorIndexer1_ ObIndexer2)))
                          (degrade (Hred ObIndexer1 _ (MorIndexer2_ ObIndexer1)))
                          (degrade (Hred ObIndexer1 _ (MorIndexer1_ ObIndexer1)))
         end;
  move: H_grade; clear; rewrite ?(Sol.toPolyMor_mut_rewrite, grade_rewrite);
  intros; try tac_grade_gt0; try tac_grade_indexed_gt0;
  try tac_grade_gt0_indexing; try tac_grade_indexed_gt0_indexing;
  intros; Psatz.lia.
```

6 Polymorphism/cut-elimination by computational/total/asymptotic/reduction/(multi-step) resolution

As common , this resolution is not programmed by morphisms-structural recursion but instead is programmed by grade-structural recursion . Moreover , this resolution presents two (nested) mutually-recursive functions : one function for the singleton-resolution of the singleton-morphisms and one function for the indexed-resolution of the indexed/family-morphisms .

In contrast , this resolution also computes the sense-decoding datatype-index/argument of the resolved morphism , this datatype-index/argument is inferred as metavariable from the actual resolved morphism via the [eexists] tactic . The technical progress of this resolution does require the earlier lemma [convCoMod_sense] , which will only be used to show the property [Yoneda10_morphismReIndexer_morphismIndexer] (polyarrowing of some cocone across the reindexer and across the indexer) of the proposed output solution-morphisms , in the 2 cases when the input morphism is [((AtIndexMor [[ggSol_]]_ J) o>CoMod (AtIndexMor [[ffSol_]]_ J))] or the input morphism is [([[ff_ @ I]])] .

This COQ program and deduction is mostly-automated ; but memo that COQ lacks inductive-recursive presentations and memo that here the automation-tactics use only logical eauto-resolution because COQ lacks some more-efficient heterogeneous-rewriting tactics , because the conversion-relation do convert across two morphisms whose sense-decoding indexes are not syntactically/grammatically-the-same .

```
Module Resolve.
Export Sol.Ex_Notations.

Ltac tac_simpl := rewrite ?grade_rewrite; rewrite ?Sol.toPolyMor_mut_rewrite;
                  cbn -[grade grade_indexed Sol.toPolyMor Sol.toPolyMor_indexed].
Ltac tac_reduce := tac_simpl; intuition eauto.
```

```
Fixpoint solveCoMod len {struct len} :
  forall Yoneda00_E Yoneda01_E (E : @obCoMod Yoneda00_E Yoneda01_E)
         Yoneda00_F Yoneda01_F (F : @obCoMod Yoneda00_F Yoneda01_F)
         Yoneda10_ff (ff : 'CoMod( E ~> F @ Yoneda10_ff )),
  forall grade_ff : (grade ff <= len)%coq_nat,
    { ffSol : { Yoneda10_ffSol : _ & 'CoMod( E ~> F @ Yoneda10_ffSol )%sol}
    | (Sol.toPolyMor (projT2 ffSol)) <~~ ff }

with solveCoMod_indexed len {struct len} :
  forall Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly
    (E_ : @obCoMod_indexed Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly)
    Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly
    (F_ : @obCoMod_indexed Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly)
    Yoneda10_ff_ (ff_ : 'CoMod_( E_ ~> F_ @ Yoneda10_ff_ )),
  forall grade_ff_ : (grade_indexed ff_ <= len)%coq_nat,
    { ffSol_ : { Yoneda10_ffSol_ : _ & 'CoMod_( E_ ~> F_ @ Yoneda10_ffSol_ )%sol}
    | (Sol.toPolyMor_indexed (projT2 ffSol_)) <~~_ ff_ } .
Proof.
{ (** solveCoMod **)
case : len => [ | len ].

(** len is O **)
- ( move => ? ? E ? ? F ? ff grade_ff ); exfalso;
    clear - grade_ff; abstract tac_degrade_mut grade_ff.

(** len is (S len) **)
- move => ? ? E ? ? F Yoneda10_ff ff; case : _ _ E _ _ F Yoneda10_ff / ff =>
  [ Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly E_
            Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly F_
            Yoneda10_ff_ ff_ I (** AtIndexMor ff_ I **)
  | Yoneda00_F Yoneda01_F F Yoneda00_F' Yoneda01_F' F'
           Yoneda10_ff' ff' Yoneda00_F'' Yoneda01_F'' F''
           Yoneda10_ff_ ff_  (** ff_ o>CoMod ff' **)
  | Yoneda00_E Yoneda01_E E Yoneda00_F Yoneda01_F F
           transf A Yoneda10_ff ff (** ff o>Transf_ transf @ F **)
  | Yoneda00_F Yoneda01_F F (** @'UnitCoMod F **)
  | Yoneda00_F Yoneda01_F F A f (** PolyElement F f **)
  | I R i Yoneda00_F Yoneda01_F F Yoneda10_rr rr
      (** rr o>CoMod 'CoUnitGenerated @ i **)
  ] grade_ff .

  (** ff is AtIndexMor ff_ I **)
  + have [:blurb] ffSol_prop :=
      (proj2_sig (solveCoMod_indexed len _ _ _ _ _ _ _ _ _ _ ff_ blurb));
        first by clear -grade_ff; abstract tac_degrade_mut grade_ff.
    move: (projT1 (sval (solveCoMod_indexed len _ _ _ _ _ _ _ _ _ _ ff_ blurb)))
  (projT2 (sval (solveCoMod_indexed len _ _ _ _ _ _ _ _ _ _ ff_ blurb))) ffSol_prop
    => Yoneda10_ffSol_ ffSol_ ffSol_prop .

    unshelve eexists. eexists. refine ( 'AtIndexMor ffSol_ I )%sol.
    move: ffSol_prop; clear; abstract tac_reduce.

  (** ff is ff_ o>CoMod ff' *)
  + all: cycle 1.

  (** ff is ff o>Transf_ transf @ G **)
  + unshelve eexists. eexists.
    refine ( 'PolyElement F (proj1_sig transf _
                              (sval Yoneda10_ff _ (@unitGenerator _))) )%sol.
    clear; abstract exact: PolyTransf_'PolyElement.

  (** gg is @'UnitCoMod F **)
  + unshelve eexists. eexists. refine (@'UnitCoMod F)%sol.
    clear; abstract exact: convCoMod_Refl.

  (** gg is PolyYoneda00 F f **)
  + unshelve eexists. eexists. refine ('PolyElement F f)%sol.
    clear; abstract exact: convCoMod_Refl.

  (** ff is rr o>CoMod 'CoUnitGenerated @ i **)
  + have [:blurb] rrSol_prop :=
      (proj2_sig (solveCoMod len _ _ _ _ _ _ _ rr blurb));
        first by clear -grade_ff; abstract tac_degrade_mut grade_ff.
    move: (projT1 (sval (solveCoMod len _ _ _ _ _ _ _ rr blurb)))
            (projT2 (sval (solveCoMod len _ _ _ _ _ _ _ rr blurb))) rrSol_prop
    => Yoneda10_rrSol rrSol rrSol_prop .
```

```
  unshelve eexists. eexists. refine ( rrSol o>CoMod 'CoUnitGenerated @ i )%sol.
  move: rrSol_prop; clear; abstract tac_reduce.

(** ff is ff_ o>CoMod ff' *)
+ have [:blurb] ff'Sol_prop :=
    (proj2_sig (solveCoMod len _ _ _ _ _ _ _ ff' blurb));
      first by clear -grade_ff; abstract tac_degrade_mut grade_ff.
  move: (projT1 (sval (solveCoMod len _ _ _ _ _ _ _ ff' blurb)))
          (projT2 (sval (solveCoMod len _ _ _ _ _ _ _ ff' blurb))) ff'Sol_prop
  => Yoneda10_ff'Sol ff'Sol ff'Sol_prop .
  have [:blurb] ff_Sol_prop :=
    (proj2_sig (solveCoMod len _ _ _ _ _ _ _ ff_ blurb));
      first by clear -grade_ff; abstract tac_degrade_mut grade_ff.
  move: (projT1 (sval (solveCoMod len _ _ _ _ _ _ _ ff_ blurb)))
          (projT2 (sval (solveCoMod len _ _ _ _ _ _ _ ff_ blurb))) ff_Sol_prop
  => Yoneda10_ff_Sol ff_Sol ff_Sol_prop .

  (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  **)
  destruct ff'Sol as
      [ Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly E_
                    Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly F_
                    Yoneda10_ff'Sol_ ff'Sol_ I (** AtIndexMor ff'Sol_ I **)
      | Yoneda00_F Yoneda01_F F (** @'UnitCoMod F **)
      | Yoneda00_F Yoneda01_F F A f (** PolyElement F f **)
      | I R i Yoneda00_F Yoneda01_F F Yoneda10_rr rrSol
          (** rrSol o>CoMod 'CoUnitGenerated @ i **) ].

  (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol) ,
      is (ff_Sol o>CoMod (AtIndexMor ff'Sol_ I) ) **)
  * { destruct ff'Sol_ as
          [ Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly E_
                        Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly F_
                        Yoneda10_ff'Sol_ ff'Sol_ (** MorCoMod_indexed ff'Sol_ **)
          | Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly F_ Yoneda10_ffSol_ ffSol_
                        Yoneda01_F_Poly_functorIndexer
                        Yoneda10_ffSol_morphismReIndexer_morphismIndexer
                         (** [[ ffSol_ ]]_ **) ].

      (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  ,
          is (ff_Sol o>CoMod (AtIndexMor ( MorCoMod_indexed ff'Sol_ ) I) ) **)
      - have [:blurb] ff_Sol_o_ff'Sol_I_prop :=
          (proj2_sig (solveCoMod len _ _ _ _ _ _ _
               (Sol.toPolyMor ff_Sol o>CoMod Sol.toPolyMor (ff'Sol_ I)) blurb));
            first by clear -grade_ff ff_Sol_prop ff'Sol_prop;
            abstract(destruct (is_ObIndexer12_allP I); tac_degrade_mut grade_ff).
        move: (projT1 (sval (solveCoMod len _ _ _ _ _ _ _
                (Sol.toPolyMor ff_Sol o>CoMod Sol.toPolyMor (ff'Sol_ I)) blurb)))
                (projT2 (sval (solveCoMod len _ _ _ _ _ _ _
                         (Sol.toPolyMor ff_Sol o>CoMod Sol.toPolyMor (ff'Sol_ I))
                         blurb))) ff_Sol_o_ff'Sol_I_prop
        => Yoneda10_ff_Sol_o_ff'Sol_I ff_Sol_o_ff'Sol_I ff_Sol_o_ff'Sol_I_prop.

        unshelve eexists. eexists. refine ( ff_Sol_o_ff'Sol_I )%sol.
        move: ff_Sol_prop ff'Sol_prop ff_Sol_o_ff'Sol_I_prop; clear; tac_simpl.
        abstract (tac_simpl; intros; eapply convCoMod_Trans with
              (uTrans := (Sol.toPolyMor ff_Sol) o>CoMod
     ('AtIndexMor ( 'MorCoMod_indexed (fun I0 => Sol.toPolyMor (ff'Sol_ I0))) I));
                   tac_reduce).

      (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  ,
          is (ff_Sol o>CoMod (AtIndexMor [[ ffSol_ ]]_ I) ) **)
      - move:
(Sol.Destruct_codomAtIndexObGenerated.morCoMod_codomAtIndexObGeneratedP ff_Sol)
        => ff_Sol_codomAtIndexObGeneratedP.
        { destruct ff_Sol_codomAtIndexObGeneratedP as
              [ J (** ( @'UnitCoMod (AtIndexOb Generated J) ) **)
              | J G f (** (PolyElement (AtIndexOb Generated J) f ) **)
              | J R j Yoneda00_F Yoneda01_F F Yoneda10_rrSol rrSol
              (** rrSol o>CoMod 'CoUnitGenerated @ j **)
              | Yoneda00_E_ Yoneda01_E_ Yoneda01_Poly E_
                            Yoneda10_ggSol_ ggSol_ J
              (** AtIndeMor (MorCoMod_indexed ggSol_) **)
              | Yoneda10_ggSol_ ggSol_
                            Yoneda01_Generated_PolyIndexer_functorIndexer'
                            Yoneda10_ggSol_morphismReIndexer_morphismIndexer J
               (** AtIndexMor [[ ffSol_' ]]_ J **) ].
```

```
          (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  , is
(( @'UnitCoMod (AtIndexOb Generated J) ) o>CoMod (AtIndexMor [[ ffSol_ ]]_ J) ) **)
          - unshelve eexists. eexists.
            refine ('AtIndexMor [[ ffSol_ @ Yoneda01_F_Poly_functorIndexer,
                   Yoneda10_ffSol_morphismReIndexer_morphismIndexer  ]]_ J)%sol.
            move: ff_Sol_prop ff'Sol_prop; clear;
              abstract (tac_simpl; intros; eapply convCoMod_Trans with
        (uTrans := ( 'UnitCoMod ) o>CoMod (Sol.toPolyMor ('AtIndexMor [[ ffSol_
                                 @ Yoneda01_F_Poly_functorIndexer,
      Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ J)%sol)); tac_reduce).

          (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  , is
((PolyElement (AtIndexOb Generated J) f ) o>CoMod (AtIndexMor [[ ffSol_ ]]_ J) ) **)
          - unshelve eexists. eexists.
            refine ('PolyElement (AtIndexOb F_(J))
             (sval (Yoneda10_ffSol_ _ _ (projT2 (projT1 f))) G (projT2 f)))%sol.
            move: ff_Sol_prop ff'Sol_prop; clear;
            abstract (rewrite ?Sol.toPolyMor_mut_rewrite; intros;
             eapply convCoMod_Trans with
             (uTrans := ('PolyElement (AtIndexOb Generated J) f)
                          o>CoMod (Sol.toPolyMor ('AtIndexMor [[ ffSol_
                          @ Yoneda01_F_Poly_functorIndexer,
                  Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ J)%sol));
             rewrite ?Sol.toPolyMor_mut_rewrite; by eauto).

          (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  , is
(rrSol o>CoMod 'CoUnitGenerated @ j) o>CoMod (AtIndexMor [[ ffSol_ ]]_ J) **)
          - have [:blurb] rrSol_o_ffSol_prop :=
              (proj2_sig (solveCoMod len _ _ _ _ _ _ _
         (Sol.toPolyMor rrSol o>CoMod Sol.toPolyMor (ffSol_(J)(R)(j))) blurb));
                first by clear -grade_ff ff_Sol_prop ff'Sol_prop;
                abstract(destruct (is_ObIndexer12_allP J);
                 destruct (is_MorIndexer12_allP j); tac_degrade_mut grade_ff).
            move: (projT1 (sval (solveCoMod len _ _ _ _ _ _ _
         (Sol.toPolyMor rrSol o>CoMod Sol.toPolyMor (ffSol_(J)(R)(j))) blurb)))
                    (projT2 (sval (solveCoMod len _ _ _ _ _ _ _
          (Sol.toPolyMor rrSol o>CoMod Sol.toPolyMor (ffSol_(J)(R)(j))) blurb)))
   rrSol_o_ffSol_prop => Yoneda10_rrSol_o_ffSol rrSol_o_ffSol rrSol_o_ffSol_prop.

            unshelve eexists. eexists. refine ( rrSol_o_ffSol )%sol.
            move: ff_Sol_prop ff'Sol_prop rrSol_o_ffSol_prop; clear;
            abstract (rewrite ?Sol.toPolyMor_mut_rewrite; intros;
                      eapply convCoMod_Trans with
                (uTrans := ((Sol.toPolyMor rrSol) o>CoMod 'CoUnitGenerated @ j)
   o>CoMod (Sol.toPolyMor ('AtIndexMor [[ ffSol_ @ Yoneda01_F_Poly_functorIndexer,
                Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ J)%sol));
                      rewrite ?Sol.toPolyMor_mut_rewrite; by eauto).

          (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  , is
('AtIndexMor ('MorCoMod_indexed ggSol_) J) o>CoMod (AtIndexMor [[ ffSol_ ]]_ J) **)
          - have [:blurb] ggSol_J_o_ff'Sol_prop :=
              (proj2_sig (solveCoMod len _ _ _ _ _ _ _
       (Sol.toPolyMor (ggSol_ J) o>CoMod Sol.toPolyMor ('AtIndexMor [[ ffSol_
                    @ Yoneda01_F_Poly_functorIndexer ,
          Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ J)%sol) blurb));
                first by clear -grade_ff ff_Sol_prop ff'Sol_prop;
           abstract(destruct (is_ObIndexer12_allP J); tac_degrade_mut grade_ff).
            move: (projT1 (sval (solveCoMod len _ _ _ _ _ _ _
     (Sol.toPolyMor (ggSol_ J) o>CoMod Sol.toPolyMor ('AtIndexMor [[ ffSol_
                                   @  Yoneda01_F_Poly_functorIndexer ,
         Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ J)%sol) blurb)))
                    (projT2 (sval (solveCoMod len _ _ _ _ _ _ _
     (Sol.toPolyMor (ggSol_ J) o>CoMod Sol.toPolyMor ('AtIndexMor [[ ffSol_
                   @  Yoneda01_F_Poly_functorIndexer ,
      Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ J)%sol) blurb)))
                    ggSol_J_o_ff'Sol_prop
            => Yoneda10_ggSol_J_o_ff'Sol ggSol_J_o_ff'Sol ggSol_J_o_ff'Sol_prop.

            unshelve eexists. eexists. refine ( ggSol_J_o_ff'Sol )%sol.
           move: ff_Sol_prop ff'Sol_prop ggSol_J_o_ff'Sol_prop; clear; tac_simpl.
           abstract (tac_simpl; intros; eapply convCoMod_Trans with
       (uTrans := ('AtIndexMor
          ('MorCoMod_indexed (fun I : obIndexer => Sol.toPolyMor (ggSol_ I))) J)
  o>CoMod (Sol.toPolyMor ('AtIndexMor [[ ffSol_ @ Yoneda01_F_Poly_functorIndexer,
    Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ J)%sol)); tac_reduce).
```

```
            (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  , is
(AtIndexMor [[ ggSol_ ]]_ J) o>CoMod (AtIndexMor [[ ffSol_ ]]_ J) **)
            - have [:blurb_] ggSol_o_ffSol_prop I0 R0
                   (i0 : 'Indexer( ReIndexing0 R0 |- I0 )) :=
                (proj2_sig (solveCoMod len _ _ _ _ _ _ _
                      ((Sol.toPolyMor (ggSol_ I0 R0 i0)) o>CoMod
 AtIndexMor (Sol.toPolyMor_indexed ( [[ ffSol_ @ Yoneda01_F_Poly_functorIndexer,
        Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ %sol))
 (I0)) (blurb_ I0 R0 i0)));
                  first by clear -grade_ff ff_Sol_prop ff'Sol_prop;
                  abstract((move => I0 R0 i0); destruct (is_ObIndexer12_allP I0);
                    destruct (is_MorIndexer12_allP i0); tac_degrade_mut grade_ff).
              have @Yoneda10_ggSol_o_ffSol_ :=
                (fun I0 R0 (i0 : 'Indexer( ReIndexing0 R0 |- I0 )) =>
                (projT1 (sval (solveCoMod len _ _ _ _ _ _ _
                      ((Sol.toPolyMor (ggSol_ I0 R0 i0)) o>CoMod
   AtIndexMor (Sol.toPolyMor_indexed ( [[ ffSol_ @ Yoneda01_F_Poly_functorIndexer,
                     Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ %sol))
                     (I0)) (blurb_ I0 R0 i0))))).
              have @ggSol_o_ffSol_ : forall I0 R0 i0,
                  'CoMod( View (Generating0 R0) ~> AtIndexOb F_ I0
                               @ Yoneda10_ggSol_o_ffSol_ I0 R0 i0 )%sol
                := (fun I0 R0 (i0 : 'Indexer( ReIndexing0 R0 |- I0 )) =>
                      (projT2 (sval (solveCoMod len _ _ _ _ _ _ _
                      ((Sol.toPolyMor (ggSol_ I0 R0 i0)) o>CoMod
   AtIndexMor (Sol.toPolyMor_indexed ( [[ ffSol_ @ Yoneda01_F_Poly_functorIndexer,
                    Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ %sol))
                                                  (I0)) (blurb_ I0 R0 i0))))).
              have {ggSol_o_ffSol_prop}: (forall I0 R0 i0,
                 Sol.toPolyMor (ggSol_o_ffSol_(I0)(R0)(i0)) <~~
                              ((Sol.toPolyMor (ggSol_ I0 R0 i0)) o>CoMod
 AtIndexMor (Sol.toPolyMor_indexed ( [[ ffSol_ @ Yoneda01_F_Poly_functorIndexer,
           Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ %sol))
 (I0))) := ggSol_o_ffSol_prop.
              move: Yoneda10_ggSol_o_ffSol_ ggSol_o_ffSol_ =>
              Yoneda10_ggSol_o_ffSol_ ggSol_o_ffSol_ ggSol_o_ffSol_prop.
              clear solveCoMod solveCoMod_indexed.

              (**memo: convCoMod_sense is really necessary here **)
              have Yoneda10_ggSol_o_ffSol_morphismReIndexer_morphismIndexer :
                Yoneda10_morphismReIndexer_morphismIndexer
                  Yoneda01_F_Poly Yoneda10_ggSol_o_ffSol_ .
              { clear - Yoneda10_ffSol_morphismReIndexer_morphismIndexer
                        Yoneda10_ggSol_morphismReIndexer_morphismIndexer
                        ggSol_o_ffSol_prop;
                  move : (fun I0 R0 i0 =>
         convCoMod_sense (ggSol_o_ffSol_prop I0 R0 i0)) => ggSol_o_ffSol_prop_eq.
                rewrite /Yoneda10_morphismReIndexer_morphismIndexer.
              intros. move. intros s. simpl. do 2 rewrite - ggSol_o_ffSol_prop_eq.
              exact: (Reflector_morphism_morphismReIndexer_morphismIndexer
                        Yoneda10_ggSol_morphismReIndexer_morphismIndexer
                        (Yoneda10_Reflector_naturalIndexer_ALT
                           Yoneda10_ffSol_morphismReIndexer_morphismIndexer )).
              }

              unshelve eexists. eexists.
              refine ( 'AtIndexMor [[ ( fun I0 R0 i0 => ggSol_o_ffSol_(I0)(R0)(i0) )
                                        @ Yoneda01_F_Poly_functorIndexer ,
             Yoneda10_ggSol_o_ffSol_morphismReIndexer_morphismIndexer ]]_ J )%sol.
              move: ff_Sol_prop ff'Sol_prop ggSol_o_ffSol_prop; clear;
                abstract( rewrite ?Sol.toPolyMor_mut_rewrite;
                          (*invisible*) progress simpl; intros;
                          eapply convCoMod_Trans with
     (uTrans := (AtIndexMor [[ (fun I0 R0 i0 => Sol.toPolyMor (ggSol_ I0 R0 i0))
                              @ Yoneda01_Generated_PolyIndexer_functorIndexer',
                 Yoneda10_ggSol_morphismReIndexer_morphismIndexer ]]_ J)
               o>CoMod ( 'AtIndexMor (Sol.toPolyMor_indexed [[ ffSol_
                            @ Yoneda01_F_Poly_functorIndexer,
                Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ %sol) J));
                          first (by rewrite ?Sol.toPolyMor_mut_rewrite; eauto);
                          eapply convCoMod_Trans with
          (uTrans := (AtIndexMor [[ (fun I0 R0 i0 => Sol.toPolyMor (ggSol_ I0 R0 i0)
                    o>CoMod ( 'AtIndexMor (Sol.toPolyMor_indexed [[ ffSol_
                                  @ Yoneda01_F_Poly_functorIndexer,
               Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ %sol) I0))
                              @ Yoneda01_F_Poly_functorIndexer,
           (Reflector_morphism_morphismReIndexer_morphismIndexer
```

```
    Yoneda10_ggSol_morphismReIndexer_morphismIndexer
               (Yoneda10_Reflector_naturalIndexer_ALT
                 Yoneda10_ffSol_morphismReIndexer_morphismIndexer )) ]]_ J));
                        rewrite ?Sol.toPolyMor_mut_rewrite; by eauto).
          }
      }

    (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  ,
is (ff_Sol o>CoMod (@'UnitCoMod F)) **)
    * unshelve eexists. eexists. refine (ff_Sol)%sol.
      move:ff_Sol_prop ff'Sol_prop; clear;
      abstract (tac_simpl; intros; eapply convCoMod_Trans with
                                  (uTrans := ff_ o>CoMod ('UnitCoMod)); tac_reduce).

    (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  ,
is (ff_Sol o>CoMod (PolyElement F f)) **)
    * move:
        (Sol.Destruct_codomView.morCoMod_codomViewP ff_Sol) => ff_Sol_codomViewP.
      { destruct ff_Sol_codomViewP as
            [ G (** @'UnitCoMod (View G) **)
            | G G' g (** PolyElement (View G) g **) ].

        - unshelve eexists. eexists. refine ('PolyElement F f)%sol.
          move: ff_Sol_prop ff'Sol_prop; clear;
          abstract (tac_simpl; intros; eapply convCoMod_Trans with
                (uTrans := ('UnitCoMod) o>CoMod ('PolyElement F f)); tac_reduce).

        - unshelve eexists. eexists.
          refine ('PolyElement F (proj1_sig Yoneda01_F G G' g f) )%sol.
          move: ff_Sol_prop ff'Sol_prop; clear;
          abstract (tac_simpl; intros; eapply convCoMod_Trans with
   (uTrans := ('PolyElement (View G) g) o>CoMod ('PolyElement F f)); tac_reduce).
      }

    (** gg is (ff_ o>CoMod ff') , to (ff_Sol o>CoMod ff'Sol)  ,
is (ff_Sol o>CoMod (rrSol o>CoMod 'CoUnitGenerated @ i)) **)
    * have [:blurb] ff_Sol_o_rrSol_prop :=
        (proj2_sig (solveCoMod len _ _ _ _ _ _ _
                   (Sol.toPolyMor ff_Sol o>CoMod Sol.toPolyMor rrSol) blurb));
          first by clear -grade_ff ff_Sol_prop ff'Sol_prop;
          abstract tac_degrade_mut grade_ff.
      move: (projT1 (sval (solveCoMod len _ _ _ _ _ _ _
                 (Sol.toPolyMor ff_Sol o>CoMod Sol.toPolyMor rrSol) blurb)))
              (projT2 (sval (solveCoMod len _ _ _ _ _ _ _
  (Sol.toPolyMor ff_Sol o>CoMod Sol.toPolyMor rrSol) blurb))) ff_Sol_o_rrSol_prop
      => Yoneda10_ff_Sol_o_rrSol ff_Sol_o_rrSol ff_Sol_o_rrSol_prop .

      unshelve eexists. eexists.
      refine ( ff_Sol_o_rrSol o>CoMod 'CoUnitGenerated @ i )%sol.
      move: ff_Sol_prop ff'Sol_prop ff_Sol_o_rrSol_prop; clear;
      abstract (tac_simpl; intros; eapply convCoMod_Trans with
         (uTrans := (Sol.toPolyMor ff_Sol) o>CoMod
            ((Sol.toPolyMor rrSol) o>CoMod 'CoUnitGenerated @ i)); tac_reduce).
}
{ (** solveCoMod_indexed **)
clear solveCoMod_indexed. (**memo: non-recursive **)
case : len => [ | len ].

(** len is O **)
- ( move => ? ? ? E_ ? ? ? F_ ? ff_ grade_ff_ ); exfalso;
    clear - grade_ff_; abstract tac_degrade_mut grade_ff_.

(** len is (S len) **)
- move => ? ? ? E_ ? ? ? F_ Yoneda10_ff_ ff_;
          case : _ _ _ E_ _ _ _ F_ Yoneda10_ff_ / ff_ =>
  [ Yoneda00_E_ Yoneda01_E_ Yoneda01_E_Poly E_
                Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly F_
                Yoneda10_ff_ ff_ (** MorCoMod_indexed ff_ **)
  | Yoneda00_F_ Yoneda01_F_ Yoneda01_F_Poly F_ Yoneda10_ff_ ff_
                Yoneda01_F_Poly_functorIndexer
                Yoneda10_ff_morphismReIndexer_morphismIndexer (** [[ ff_ ]]_ **)
  ] grade_ff_ .

  (** ff is MorCoMod_indexed ff_ **)
  + have [:blurb_] ffSol_prop (I : obIndexer) :=
      (proj2_sig (solveCoMod len _ _ _ _ _ _ _ (ff_(I)) (blurb_ I)));
        first by clear -grade_ff_;
```

```
 abstract(move => I; destruct (is_ObIndexer12_allP I); tac_degrade_mut grade_ff_).
    have @Yoneda10_ffSol_ := (fun (I : obIndexer) =>
               projT1 (sval (solveCoMod len _ _ _ _ _ _ _ (ff_(I)) (blurb_ I)))).
    have @ffSol_ : (forall I, 'CoMod( AtIndexOb E_ I ~> AtIndexOb F_ I @ Yoneda10_ffSol_ I )%sol)
      := (fun (I : obIndexer) =>
            projT2 (sval (solveCoMod len _ _ _ _ _ _ _ (ff_(I)) (blurb_ I)))).
    have {ffSol_prop}: (forall (I : obIndexer), Sol.toPolyMor (ffSol_(I)) <~~ ff_ I)
      := ffSol_prop.
      move: Yoneda10_ffSol_ ffSol_ => Yoneda10_ffSol_ ffSol_ ffSol_prop.

      unshelve eexists. eexists. refine ( 'MorCoMod_indexed ffSol_ )%sol.
      move: ffSol_prop; clear; abstract tac_reduce.

  (** ff is [[ ff_ @ I ]] **)
  + have [:blurb_] ffSol_prop I R (i : 'Indexer( ReIndexing0 R |- I )) :=
      (proj2_sig (solveCoMod len _ _ _ _ _ _ _ (ff_(I)(R)(i)) (blurb_ I R i)));
        first by clear -grade_ff_;
        abstract((move => I R i); destruct (is_ObIndexer12_allP I);
                 destruct (is_MorIndexer12_allP i); tac_degrade_mut grade_ff_).

    have @Yoneda10_ffSol_ := (fun I R i =>
     projT1 (sval (solveCoMod len _ _ _ _ _ _ _ (ff_(I)(R)(i)) (blurb_ I R i)))).
    have @ffSol_ : (forall I R i,
   'CoMod( View (Generating0 R) ~> AtIndexOb F_ I @ Yoneda10_ffSol_ I R i ) %sol)
      := (fun I R i => projT2 (sval (solveCoMod len _ _ _ _ _ _ _
                                              (ff_(I)(R)(i)) (blurb_ I R i)))) .
    have {ffSol_prop}: (forall I R i,
               Sol.toPolyMor (ffSol_(I)(R)(i)) <~~ ff_(I)(R)(i)) := ffSol_prop.
    move: Yoneda10_ffSol_ ffSol_ => Yoneda10_ffSol_ ffSol_ ffSol_prop.
    clear solveCoMod.

    (**memo: convCoMod_sense is really necessary here **)
    have Yoneda10_ffSol_morphismReIndexer_morphismIndexer :
      Yoneda10_morphismReIndexer_morphismIndexer Yoneda01_F_Poly Yoneda10_ffSol_ .
    { clear - Yoneda10_ff_morphismReIndexer_morphismIndexer ffSol_prop;
        move : (fun I R i => convCoMod_sense (ffSol_prop I R i)) => ffSol_prop_eq.
      rewrite /Yoneda10_morphismReIndexer_morphismIndexer.
      intros. move. intros. simpl. do 2 rewrite - ffSol_prop_eq.
      apply Yoneda10_ff_morphismReIndexer_morphismIndexer.
    }

    unshelve eexists. eexists. refine
     ( [[ ffSol_ @ Yoneda01_F_Poly_functorIndexer ,
          Yoneda10_ffSol_morphismReIndexer_morphismIndexer ]]_ )%sol.
    move: ffSol_prop; clear; abstract tac_reduce.
}
Defined.
End Resolve.
End GENERATEDFUNCTOR.
```

Voila.