A new implementation of dependent types via Dosen's substructural categorial programming: example of the Yoneda lemma for fibrations

Cut-elimination in the double category of fibred profunctors with inner cut-eliminated adjunctions:

https://github.com/1337777/cartier/blob/master/cartierSolution13.lp

Inductive datatypes or computation of concrete data limits via the abstract prover grammar:

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

Outline.

- ✓ Categories, functors, profunctors, hom-arrows, transformations.
- ✓ Composition is Yoneda "lemma".
- ✓ Outer cut-elimination or functorial lambda calculus.
- ✓ Inner cut-elimination or decidable adjunctions.
- ✓ Synthetic fibred Yoneda.
- ✓ Substructural fibred Yoneda.
- ✓ Comma elimination ("J-rule arrow induction").
- ✓ Cut-elimination for fibred arrows.
- ✓ Pi-category-of-fibred-functors and Sigma-category.
- ✓ And why profunctors (of sets)?
- ✓ What is a fibred profunctor anyway?
- ✓ Higher inductive types, the interval type, concrete categories.
- ✓ Universe, universal fibration.
- ✓ Weighted limits.
- ✓ Duality Op, covariance vs contravariance.
- ✓ Grammatical topology.
- ✓ Applications: concrete datatypes and limits of finite sets.
- ✓ References.

• This closes the open problem of *implementing a dependent-types computer for category theory*, where types are categories and dependent types are fibrations of categories. The basis for this implementation are the ideas and techniques from *Kosta Dosen's book « Cut-elimination in categories » (1999)*, which essentially is about the substructural logic of category theory, in particular about how some *good substructural formulation of the Yoneda lemma* allows for computation and automatic-decidability of categorial equations.

• The core of dependent types/fibrations in category theory is the Lawvere's comma/slice construction and the corresponding Yoneda lemma for fibrations (https://stacks.math.columbia.edu/tag/0GWH), thereby its implementation essentially closes this open problem also investigated by Cisinski's directed

types or Garner's 2-dimensional types. What qualifies as a solution is subtle and **the thesis here is that Dosen's substructural techniques cannot be bypassed**.

• In summary, this text implements, using *Blanqui's LambdaPi metaframework software tool*, an (outer) cut-elimination in the double category of fibred profunctors with (inner) cut-eliminated adjunctions.

• This implementation of the outer cut-elimination essentially is a *new functorial lambda calculus via the « dinaturality » of evaluation* and the monoidal bi-closed structure of profunctors, without need for multicategories because (outer) contexts are expressed via dependent types.

• This text also implements *(higher) inductive datatypes* such as the join-category (interval simplex), with its introduction/elimination/computation rules.

- This text also implements *Sigma-categories/types* and categories-of-functors and more generally *Pi-categories-of-functors*, but an alternative more-intrinsic formulation using functors fibred *over spans or over Kock's polynomial-functors* will be investigated.

- This text also implements *dualizing Op operations*, and it can computationally-prove that left-adjoint functors preserve profunctor-weighted colimits from the proof that *right-adjoint functors preserve profunctor-weighted limits*.

- This text also implements a *grammatical (univalent) universe* and the universal fibration classifying small fibrations, together with the dual universal opfibration.

• Finally, there is an experimental implementation of *covering (co)sieves towards grammatical sheaf cohomology* and towards a description of algebraic geometry's schemes in their formulation as *locally affine ringed sites* (structured topos), instead of via their Coquand's formulation as underlying topological space...

Motivation 1.

- Composition of functions and associativity

$$e \circ f \circ g \circ h$$

CHOICE A? $\qquad ((e \circ f) \circ g) \circ h$

CHOICE B? $\qquad e \circ (f \circ (g \circ h))$

- Problem with computation rules, for pairing/projections, for case/injection

$$\text{projectFirst} \circ \text{pair}\,(x, y) = x$$

$$\text{case}[s|t] \circ \text{inLeft}(z) = s(z)$$

- Attempt for CHOICE A:

$$((e \circ \text{projectFirst}) \circ \text{pair}\,) \circ f \qquad \text{KO}$$

$$\big((e \circ \text{case}[s|t]) \circ \text{inLeft}\big) \circ f \qquad \text{OK}$$

- Attempt for CHOICE B:

$$e \circ (\text{projectFirst} \circ (\text{pair}\, \circ f)) \qquad \text{OK}$$

$$e \circ (\text{case}[s|t] \circ (\text{inLeft} \circ f)) \qquad \text{KO}$$

Motivation 2.

• How to write the (co)unit ϵ of an adjunction between left adjoint $F: D \to C$ and right adjoint $G: C \to D$ (Note: the notion of adjoint functions is a generalization of the notion of inverse functions)?

CHOICE A: $\quad \epsilon_X: C(FGX, X)$ where X is variable

CHOICE B: $\quad \epsilon_X^F: C[F, -](GX, X)$ where $C[F, -] : D \times C \to Set$
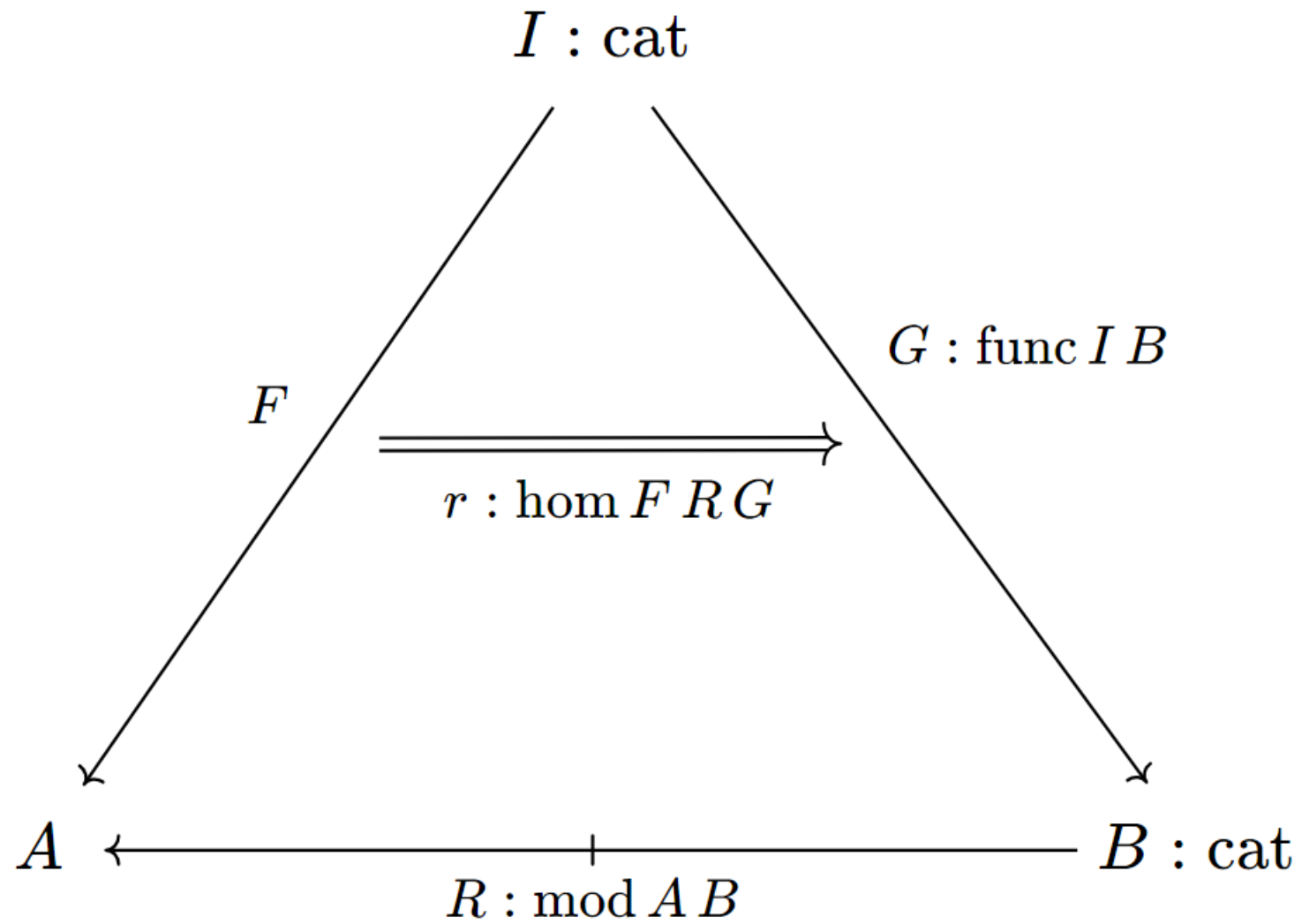
CHOICE C: $\quad \epsilon_X: C[FG, -](X, X)$

BAD CHOICE: $\quad \epsilon_X: C[FGX, -](-, X)$

Categories, functors, profunctors, hom-arrows, transformations.

• These organize into a *double category* of (fibred) profunctors, where

categories are basic and manipulated from the outside via functors $F : I \to C$

instead of via their usual objects "$F : \mathrm{Ob}(C)$".

```
constant symbol cat : TYPE;

constant symbol func : Π (A B : cat), TYPE;

constant symbol mod : Π (A B : cat), TYPE;

constant symbol hom_Set : Π [I A B : cat], func I A → mod A B → func I B → Set;

injective symbol hom [I A B : cat] (F : func I A) (R : mod A B) (G : func I B): TYPE
  ≔ τ (@hom_Set I A B F R G );

injective symbol transf [A' B' A B: cat] (R' : mod A' B') (F : func A' A) (R : mod A
B) (G : func B' B) : TYPE ≔ τ (@transf_Set A' B' A B R' F R G);
```

$$I : \mathrm{cat}$$

F

$G : \mathrm{func}\, I\, B$

$$r : \mathrm{hom}\, F\, R\, G$$

$$A \xleftarrow{\hspace{3cm}} B : \mathrm{cat}$$

$$R : \mathrm{mod}\, A\, B$$

$$A' \xleftarrow{\quad R' : \mathrm{mod}\ A'\ B' \quad} B' : \mathrm{cat}$$

$$F \Bigg\downarrow \qquad\qquad \Longrightarrow \qquad\qquad \Bigg\downarrow G : \mathrm{func}\ I\ B$$

$$rr : \mathrm{transf}\ R'\ F\ R\ G$$

$$A \xleftarrow{\quad R : \mathrm{mod}\ A\ B \quad} B : \mathrm{cat}$$

Composition is Yoneda "lemma".

• There are the usual compositions/whiskering and their units/identities.

```
symbol ∘> : Π [A B C: cat], func A B → func B C → func A C;


symbol ∘>> : Π [X B C: cat], func C X → mod X B → mod C B;


constant symbol ⊗ : Π [A B X : cat], mod A B → mod B X → mod A X;


symbol ∘↓ : Π [I A B I' : cat] [R : mod A B] [F : func I A] [G : func I B], hom F R
G → Π (X : func I' I), hom (X ∘> F) R (G <∘ X);


symbol '∘ : Π [A B' B I : cat] [S : mod A B'] [T : mod A B] [X : func I A] [Y : func
I B']  [G : func B' B],
 hom X S Y → transf S Id_func T G → hom X T (G <∘ Y);


symbol ''∘ [ B'' B' A B : cat] [R : mod A B''] [S : mod A B'] [T : mod A B] [Y :
func B'' B'] [G : func B' B] :
   transf R Id_func S Y → transf S Id_func T G →  transf R Id_func T (G <∘ Y);
```

- But the usual inner composition/cut inside categories

$$\forall A\ B\ C : \mathrm{Ob}(R),\ \hom_R(B,C) \to \hom_R(A,B) \to \hom_R(A,C),$$

instead, is assumed directly as the *Yoneda "lemma"*, *by reordering quantifiers*:

$$\forall B\ C : \mathrm{Ob}(R),\ \hom_R(B,C) \to \big(\forall\ A : \mathrm{Ob}(R), \hom_R(A,B) \to \hom_R(A,C)\big),$$

and using the *unit category-profunctor* so that any *hom-element/arrow* becomes, via this Yoneda "lemma", also a *transformation* from the unit profunctor.

```
constant symbol Unit_mod : ⊓ [X A B : cat], func A X → func B X → mod A B;


injective symbol _'∘> : ⊓ [I A B J : cat] [F : func I A] [R : mod A B] [G : func I
B], ⊓ (M : func J A),
  hom F R G → transf (Unit_mod M F) Id_func (M ∘>> R) G;


injective symbol ∘>'_ : ⊓ [I A B J : cat] [F : func I A] [R : mod A B] [G : func I
B],
  hom F R G → ⊓ (N: func J B), transf (Unit_mod G N) F (R <<∘ N) Id_func;
```

$$I : \mathrm{cat}$$

$$A(M, F)$$

$$F$$

$$r : \mathrm{hom}\, F\, R\, G$$

$$G : \mathrm{func}\, I\, B$$

$$r \circ \_ : \mathrm{transf}\, A(M, F)\, 1_J\, R(M, -)\, G$$

$$J$$

$$M$$

$$A$$

$$R : \mathrm{mod}\, A\, B$$

$$B : \mathrm{cat}$$
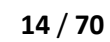
Outer cut-eliminatin or functorial lambda calculus.

• This implementation of the outer cut-elimination essentially is a *new functorial lambda calculus via the « dinaturality » of evaluation* and the monoidal bi-closed structure of profunctors. The conjunction bifunctor $\_\otimes\_$ has right adjoint implication bifunctor $\_\Rightarrow\_$ via the lambda/eval bijection of hom-sets. Then dinaturality is used to accumulate the argument-component of the eval operation instead on its function-component:

$$\text{``eval}_{B,O} \circ B \otimes (g)\text{''} \circ (x \otimes f)$$

$$= \text{``eval}_{A,O} \circ A \otimes \big((x \Rightarrow O) \circ (g \circ f)\big)\text{''}, \qquad x \colon A \to B$$

```
constant symbol ⊗ : Π [A B X : cat], mod A B → mod B X → mod A X;

constant symbol ⇐ : Π [A B X : cat], mod A B → mod X B → mod A X;

constant symbol ⇒ : Π [A B X : cat], mod A X → mod A B → mod X B;


injective symbol Eval_cov_transf : Π [A B  X A'  : cat] [P : mod A B] [Q : mod B X]
[O : mod A' X] [F : func A A'] ,
 transf P                        F (O ⇐ Q) Id_func →
 transf (P ⊗ Q)  F O                      Id_func;


constant symbol Tensor_cov_transf : Π [A' I I' X' A X: cat] [P' : mod A' I'] [Q' :
mod I' X'] [P : mod A I] [Q : mod I X] [F : func A' A]  [G : func X' X] , Π (M :
func I' I),

transf P' F (P <<∘ M) Id_func →  transf ( Q') Id_func (M ∘>> Q) G →

transf ((⊗) P' Q') F ((⊗) P Q) G;


rule  (Eval_cov_transf $pq_o) ∘'' (Tensor_cov_transf $M $p'p $q'q)

↪  Eval_cov_transf ((Imply_cov_transf (Id_transf _) $q'q) ∘'' (($pq_o <<∘1 $M) ∘''
$p'p));
```

$$R = (T \otimes S)(\_, -) = \Sigma_{x:X} \; T(\_, x) \times S(x, -)$$

$$X$$

$$T \qquad S = T \Rightarrow R$$

$$\epsilon_R^T : T \otimes (T \Rightarrow R) \to R$$

$$D \xleftarrow{\quad R \quad} C$$
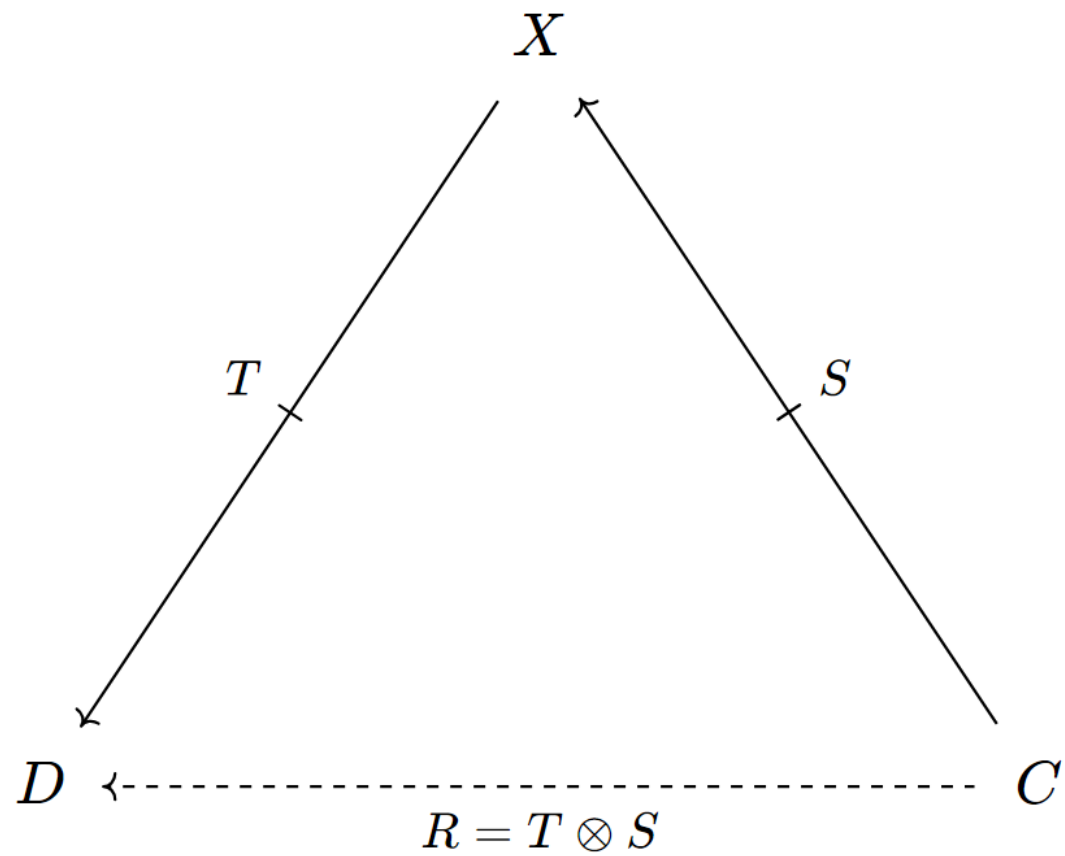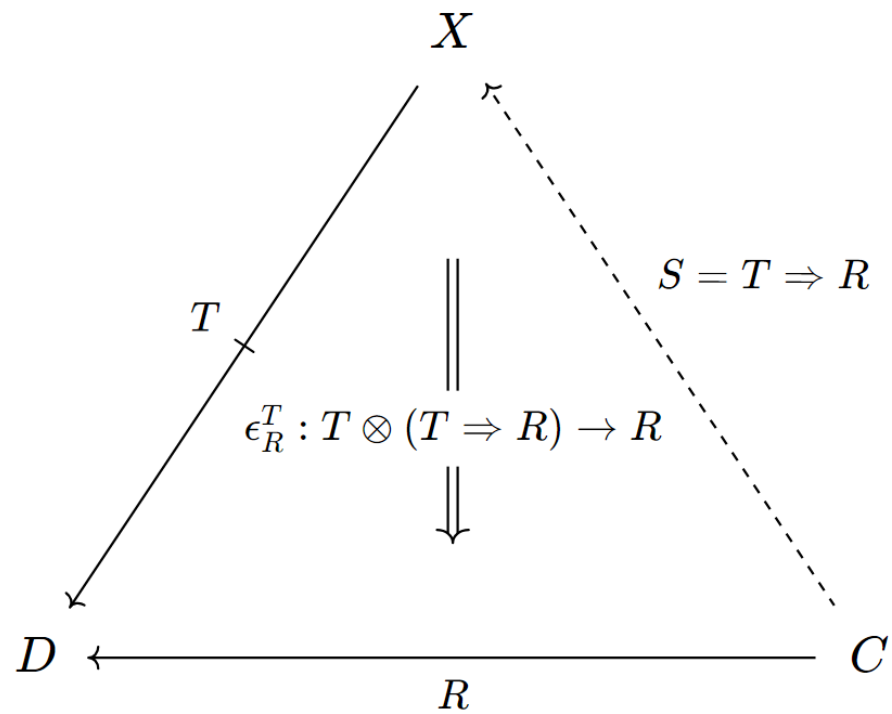
$$S = (T \Rightarrow R)(\_, -) = \Pi_{d:D} \ T(d, \_) \to R(d, -)$$

$$T \otimes - \dashv T \Rightarrow - \quad \text{contravariant imply (``right lifting'')}$$

$$X$$

$$T = R \Leftarrow S$$

$$S$$

$$\epsilon_R^S : (R \Leftarrow S) \otimes S \to R$$

$$D \xleftarrow{\quad R \quad} C$$
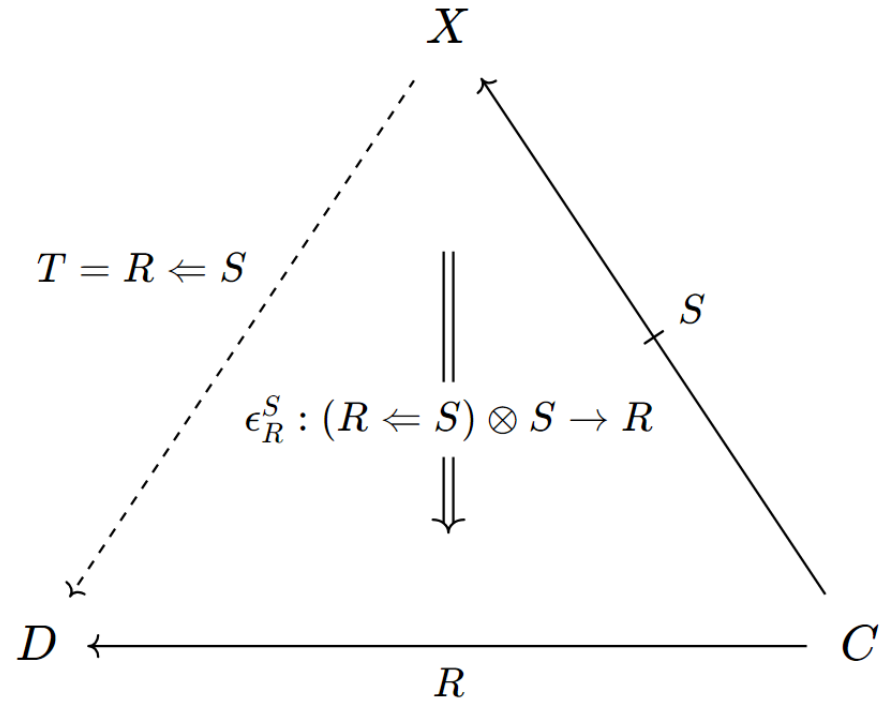
$$T = (R \Leftarrow S)(\_, -) = \Pi_{c:C} \; R(\_, c) \leftarrow S(-, c)$$

$$- \otimes S \dashv - \Leftarrow S \quad \text{covariant imply (``right extension'')}$$

$$X'$$

$$R \otimes F^* = \mathrm{Lan}_F R$$

$$\eta_R^{F^*} : R \to (R \otimes F^* \Leftarrow F^*)$$

$$F_* \dashv \qquad \dashv \qquad \dashv F^*$$

$$- \Leftarrow F^* =$$
$$-(\_, F) =$$
$$- \otimes F_*$$

$$R \Leftarrow F_* = \mathrm{Ran}_F R$$

$$\epsilon_R^{F_*} : (R \Leftarrow F_*) \otimes F_* \to R$$

$$D \qquad\qquad R \qquad\qquad X$$

$$- \otimes F^* \dashv - \Leftarrow F^* = -(\_, F) = - \otimes F_* \dashv - \Leftarrow F_*$$

X'

$F_* \otimes R = \mathrm{Lan}_{F^{\mathrm{op}}} R$

F_* $\dashv$ F^*

$F_* \Rightarrow - = \\ -(F, \_) = \\ F^* \otimes -$

$\eta_R^{F_*} : \\ R \to (F_* \Rightarrow (F_* \otimes R))$

$F^* \Rightarrow R = \mathrm{Ran}_{F^{\mathrm{op}}} R$

$\epsilon_R^{F^*} : \\ F^* \otimes (F^* \Rightarrow R) \to R$

X $\qquad R \qquad$ C

$F_* \otimes - \dashv F_* \Rightarrow - = -(F, \_) = F^* \otimes - \dashv F^* \Rightarrow -$

$$
\begin{array}{ccccc}
A & \xleftarrow{\quad P \quad} & B & \xleftarrow{\quad Q \quad} & X \\
\Big\downarrow{F} & & & & \Big\downarrow{L} \\
A' & \xleftarrow{\quad O \quad} & & & X'
\end{array}
$$

P

Q

A

B

X

$\lambda f : \mathrm{transf}\, P\, F\, (O \Leftarrow_L Q)\, 1_B$

$O \Leftarrow_L Q$

F

L

$f : \mathrm{transf}\, (P \otimes Q)\, F\, O\, L \Longrightarrow$

A'

O

X'
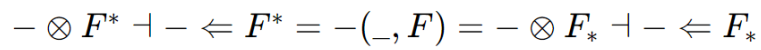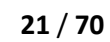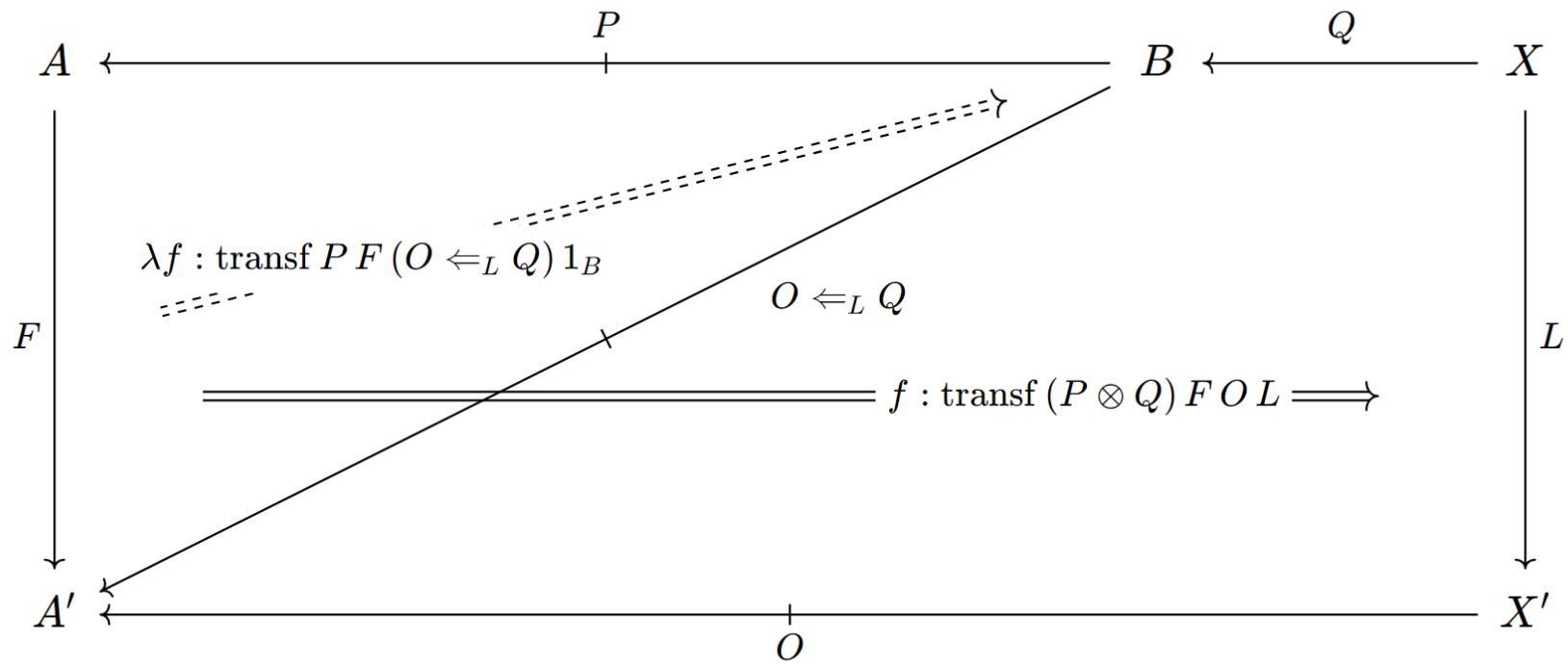
Inner cut-elimination or decidable adjunctions.

• The *inner* cuts/compositions/actions within categories must also be eliminated/admissible/computational in a confluent/convergent manner in order to obtain the automatic-decidability of the categorial equations.

• Here "cut" is synonymous with either of

  o non-fibred composition of arrows (or action by arrows on a profunctor/module), or

  o fibred composition of fibred arrows (or action by fibred arrows on a fibred profunctor), or

  o fibred transport (action by non-fibred arrows on a fibred profunctor).

- *For reference, §4.1.5 in Kosta Dosen's book* says that an adjunction with left adjoint functor F: catB → catA, right adjoint functor G: catA → catB, counit transformation ϕ_A: $F\ G\ A \to A$ (where among many formulations, A could be seen as a *parameter* functor exposing a *variable X* with ϕ_X^A: $F\ (G\ A\ X) \to A\ (X)$, that is ϕ_X^A: $\text{hom}_{\text{catA}(F,A)}\ (G\ A\ X)\ (X))$, and unit transformation γ_B: $B \to G\ F\ B$, is formulated as rewrite rules from any redex outer cut on the left-side to the contractum containing some *smaller* inner cut, where f_1, g_1 are the (fixed) *parameters* and f_2, g_2 are the *natural variables*. Those rules are classified as:

  o *Accumulation rules* (those accumulation equations can be formulated once generically for all such transformations):

$$f_2 \circ (f_1)``A \circ \phi \circ F" = (f_2 \circ f_1)``A \circ \phi \circ F"$$

$$``A \circ \phi \circ F"(g_1)`` \circ F" \circ g_2 = ``A \circ \phi \circ F"(g_1 \circ g_2)$$

  o *Naturality rules*:

$$(f_1)``A \circ \phi \circ F" \circ ((f_2)``GA \circ 1") = ((f_1)``1 \circ 1" \circ f_2)``A \circ \phi \circ F"$$

$$f_2 \circ ``A \circ \phi \circ F"(g_1) = ``A \circ \phi \circ F"((f_2)``GA \circ 1" \circ g_1)$$

o And similarly, for the naturality of the adjunction unit transformation, and for the *functoriality/naturality* (besides the generic accumulation rules) of every functor:

$$(\text{"}1 \circ FB\text{"}(g_2)) \circ \text{"}G \circ \gamma \circ B\text{"}(g_1) = \text{"}G \circ \gamma \circ B\text{"}(g_2 \circ \text{"}1 \circ 1\text{"}(g_1))$$

$$(\text{"}1 \circ B\text{"}(g_2)) \circ \text{"}1 \circ B\text{"}(g_1) = \text{"}1 \circ B\text{"}(g_2 \circ \text{"}1 \circ 1\text{"}(g_1))$$

o *Beta-cancellation conversion (or rewrite) rules* (i.e., $(\lambda-.C[-]) \cdot \_ = C[\_]$; the other half, *Eta-cancellation* $\lambda-.(g \cdot -) = g$ is similar):

$$(f_1)\text{"}A \circ \phi \circ F\text{"}((f_2)\text{"}G \circ \gamma \circ B\text{"}(g_1)) = (f_1)\text{"}A \circ 1\text{"}((f_2)\text{"}1 \circ FB\text{"}(g_1))$$

With (substructural) variations such as:

$$\text{"}A \circ \phi \circ F\text{"}((f_2)\text{"}G \circ \gamma \circ B\text{"}) = f_2$$

$$\text{"}1 \circ \phi \circ F\text{"}(\text{"}G \circ \gamma \circ B\text{"}g_2) = \text{"}1 \circ FB\text{"}g_2$$

where indeed the functions on arrows $F-$ and $G-$ of those functors are not primitive but are themselves the (Yoneda) "antecedental/consequential transformation" formulations "$1 \circ F-$" or "$G- \circ 1$" of the identity arrows on applied-functor objects...

symbol **Func_con_hom** : Π [A B A' : cat] (Z : func A A') (F : func B A),
  hom F (Unit_mod Z Id_func) (Z <∘ F);

constant symbol **Adj_con_hom** : Π [L R : cat] [LAdj_func : func R L] [RAdj_func : func
L R] (aj : adj LAdj_func RAdj_func), Π [I] (Z : func I R) [J] (N : func J I),
  hom N (Unit_mod Z RAdj_func) (N ∘> (Z ∘> LAdj_func));

assert [C D : cat] [F : func D C] [G : func C D] [aj : adj F G]   [C'] [N : func C'
C] [I] [X : func I C']   [C''] [N' : func C'' C'] [I'] [X' : func I' I] [Z : func I'
C''] (f : hom X' (Unit_mod X N') Z)   [D'] [M : func D' D] [J] [Y : func J D']
[D''] [M' : func D'' D'] [J'] [Y' : func J' J] [W : func J' D''] (g : hom W
(Unit_mod M' Y) Y')   ⊢ eq_refl _ : π (

 ((g '∘ (M')_'∘> (Adj_con_hom aj M Y)) ∘>'_(N <∘ X <∘ X'))

     ''∘ ((M' ∘> M)_'∘> ((Adj_cov_hom aj N X) ∘>'_(N') ∘' f))
 = ((g '∘ (M')_'∘> (Func_con_hom (M ∘> F) Y)) ∘>'_(N <∘ X <∘ X'))

     ''∘ ((M' ∘> M ∘> F)_'∘> ((Func_cov_hom N X) ∘>'_(N') ∘' f)));
// : transf (Unit_mod (Y' ∘> (Y ∘> (M ∘> F))) (N <∘ X <∘ X'))
          W (Unit_mod (M' ∘> M ∘> F) (N <∘ N')) Z

$$J$$

$$I$$

$$Y$$

$$X$$

$$D'$$

$$C'$$

$$Y \qquad \eta_Y^M : \hom_{D(M,G)} Y \ (FMY) \qquad M$$

$$N \qquad \epsilon_X^N : \hom_{C(F,N)} \ (GNX) \ X \qquad X$$

$$D$$

$$C$$

$$F$$

$$G$$

$$D' \xrightarrow{\quad M \quad} D \xleftarrow{\quad G \quad} C$$

$$D \xrightarrow{\quad F \quad} C \xleftarrow{\quad N \quad} C'$$

I

Y

W $\quad\Longrightarrow\quad t : \hom_R W\ X \quad\Longrightarrow\quad$ X

J

$g : \hom_{D(M,W)} Z\ Y$

$t \circ_{ML} -:$
transf $D(ML,W)\ \mathbf{1}\ R(ML,-)\ X$

$g \circ_L -:$
transf $E(L,Z)\ \mathbf{1}\ D(ML,W)\ Y$

Z

$D \longleftarrow \qquad\qquad R \qquad\qquad C$

$E(L,Z)$

M

$R(ML,-)$

E

L

E'
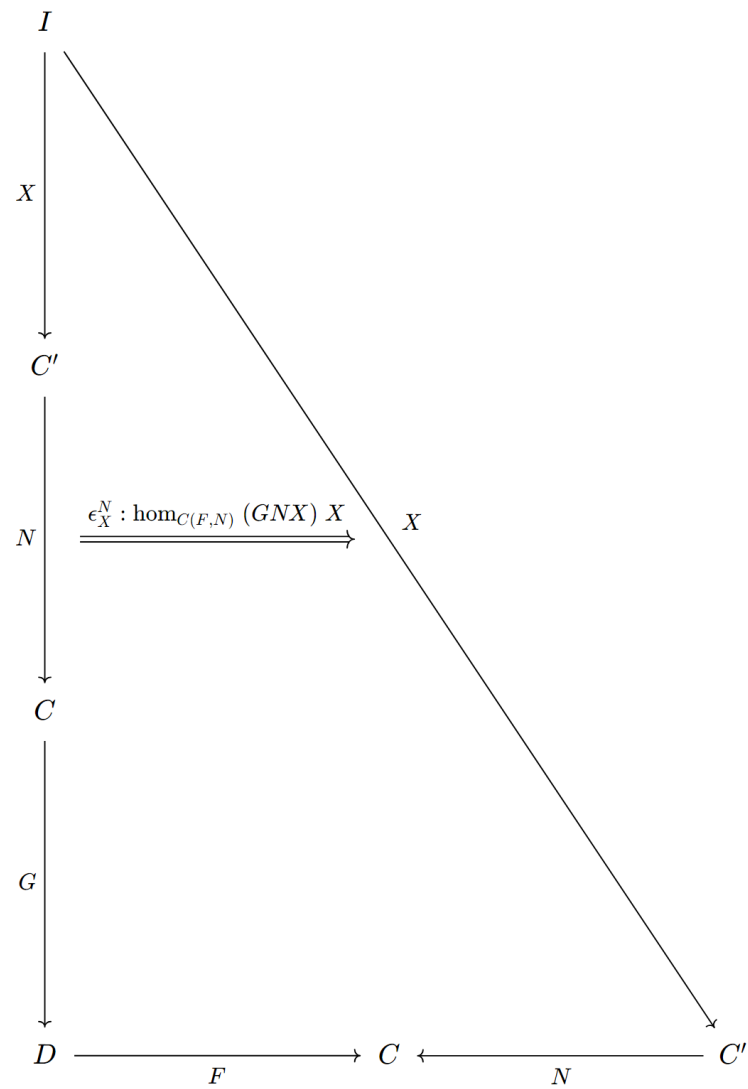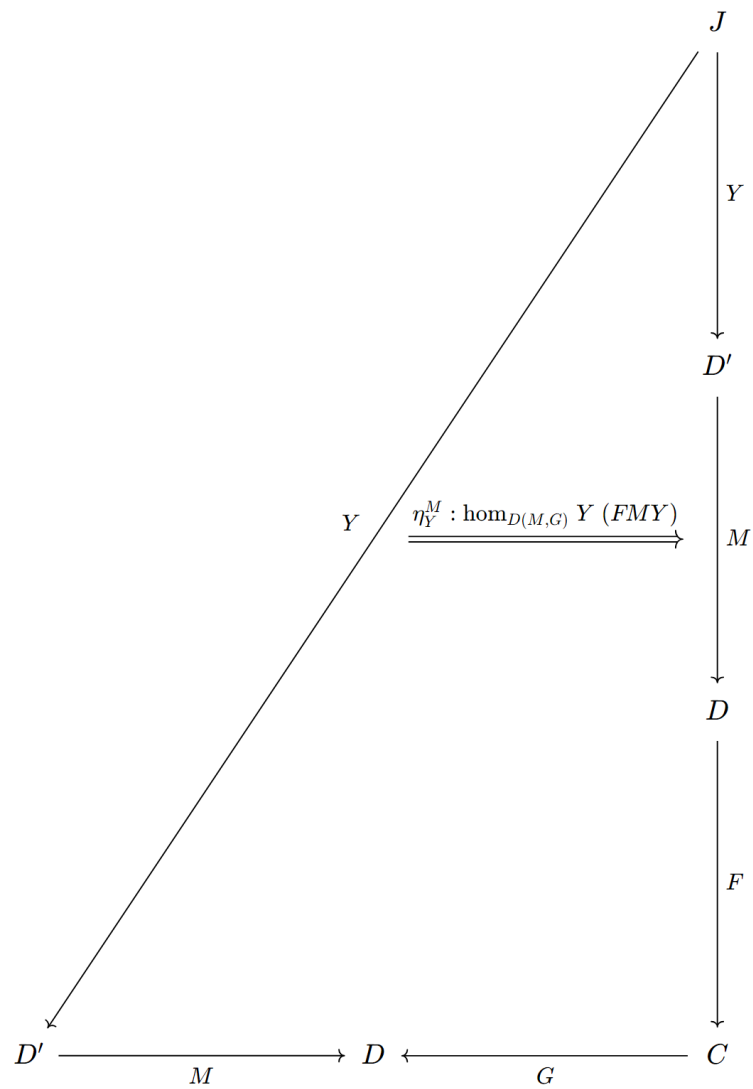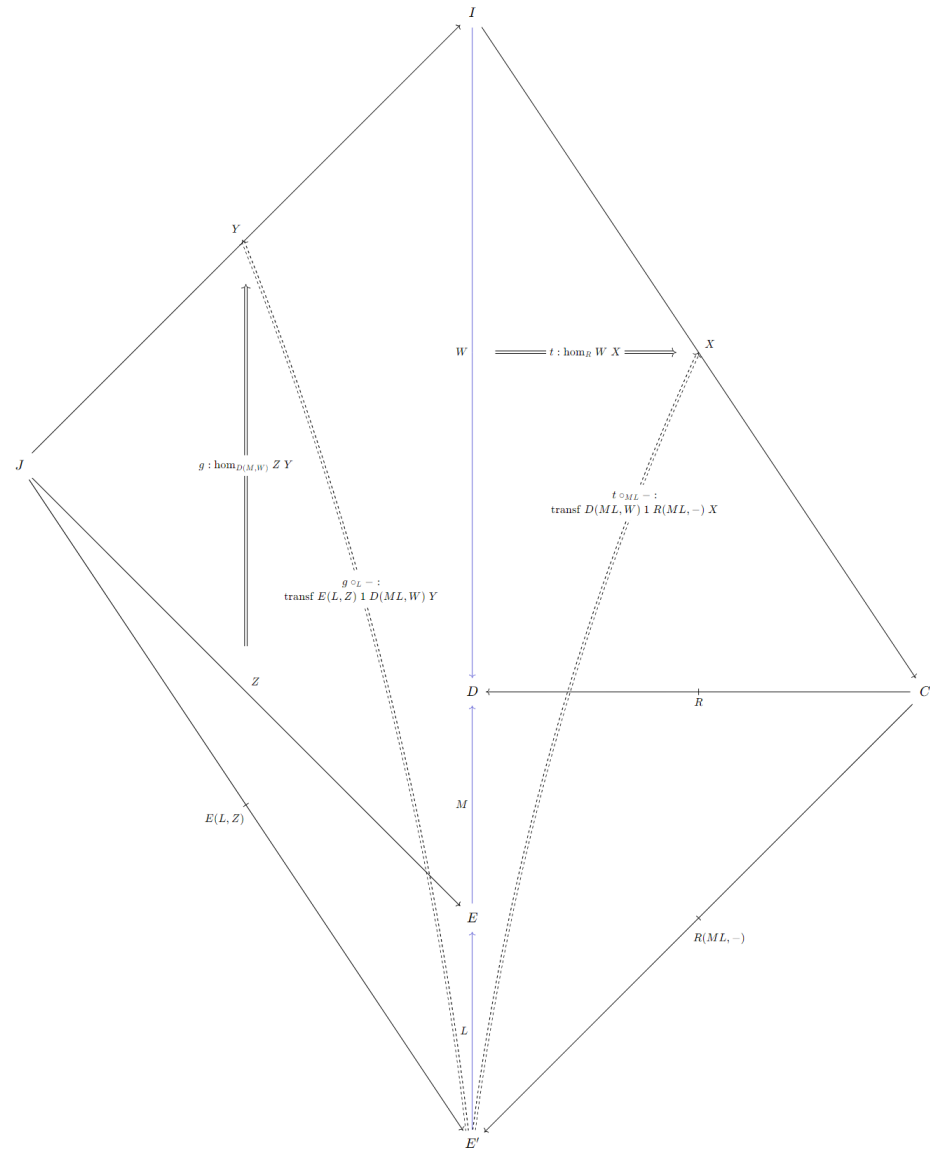
accumulation (contravariant) : $\quad (t \circ_M \underline{\ g\ }) \circ_L - \quad = \quad t \circ_{ML} (g \circ_L -) \quad :$ transf $E(L,Z)\ \mathbf{1}\ R(ML,-)\ (XY)$

I'

X'

I

$f:$
$\hom_{C'(X,X')} X' Z$

X

C'

Z

$f \circ_{N'} e_X^X$
$: \hom_{C(F,NX')} (GNXX') Z$

N

$\Longrightarrow e_X^X : \hom_{C(F,N)} (GNX) X \Longrightarrow X$

$C(FMYY',NXX')$

$\circ_{NXX'} (\eta_Y^H \circ_{M'} g):$
transf $C(FMYY',NXX') W D(MM',GNXX') 1$

J' Y'' J Y D' M D F C

G

$\eta_Y^H:$
$\hom_{D(M,G)} Y (FMY)$

D F C N C' N' C''

$g:$
$\hom_{C(M',Y)} W Y''$

$\eta_Y^H \circ_{M'} g:$
$\hom_{D(MM',G)} W (FMYY')$

Y

$(f \circ_{N'} e_X^X) \circ_{MM'} \quad :$
transf $D(MM',GNXX') 1 C(FMM',NN') Z$

M

W

$C(FMM',NN')$

D'

M'

D''

beta cancellation: $(\underline{f} \circ_{N'} e_X^X) \circ_{MM'} (- \circ_{NXX'} (\eta_Y^H \circ_{M'} g))$ $=$ $"N\underline{f} \circ_{N'} 1" \circ_{FMM'} (- \circ_{NXX'} "1 \circ_{M'} FM\underline{g}")$ $: C(FMYY',NXX') W C(FMM',NN') Z$

J

Y

I

X

C'

N

C

G

D

$C'(W, XY)$

W

$f : \hom_{C'(X,L)} Y\ Z$

Z

$- \circ_{XY} 1^W_{D(1,GN)}$
$= \text{``}GN -_W \circ_{XY} 1\text{''}$
$: \operatorname{transf} C(W,XY)\ (GNW)\ D(1,GNXY)\ 1$

$\underline{f} \circ_L \epsilon^N_X$
$: \hom_{C(F,NL)} (GNXY)\ Z$

$\epsilon^N_X : \hom_{C(F,N)} (GNX)\ X$

X

$(\underline{f} \circ_L \epsilon^N_X) \circ_1 -$
$: \operatorname{transf} D(1,GNXY)\ 1\ C(F,NL)\ Z$

$1^W_{D(1,GN)} : \hom_{D(1,GN)} (GNW)\ W$

$K \xrightarrow{\quad W \quad} C' \xrightarrow{\quad N \quad} C \xrightarrow{\quad G \quad} D \xrightarrow{\quad F \quad} C \xleftarrow{\quad N \quad} C'$

L

E

naturality (counit, contravariant eq, left of eq): $(\underline{f} \circ_L \epsilon^N_X) \circ_1 (\text{``}GN -_W \circ_{XY} 1\text{''})$  $=$  $((\underline{f} \circ_L \text{``}1^X_{C(1,1)} \circ_W 1 -_{XY}\text{''}) \circ_L \epsilon^N_W)$ $: \operatorname{transf} C(W,XY)\ (GNW)\ C(F,NL)\ Z$

$$K \xleftarrow{\quad C'(W,XY) \quad} J$$

W

C'

N

C

G

$D \xrightarrow{\quad F \quad} C \xleftarrow{\quad N \quad} C'$

Y

$f : \hom_{C'(X,L)} Y\, Z$

Z

X

L

E

$(\underline{f} \circ_L \text{``}1^X_{C'(1,1)} \circ_W 1 -_{XY}\text{''})$
$: \text{transf}\ C(W,XY)\ 1\ C'(W,L)\ Z$

$\epsilon^N_W : \hom_{C(F,N)} (GNW)\ W$

W

$- \circ_L \epsilon^N_W$
$: \text{transf}\ C'(W,L)\ (GNW)\ C(F,NL)\ 1$

$\underline{f} \circ_L 1^X_{C'(1,1)}$
$: \hom_{C'(1,L)} (XY)\ Z$

naturality (counit, contravariant eq, right of eq):
$(\underline{f} \circ_L \epsilon^N_X) \circ_1 (\text{``}GN -_W \circ_{XY} 1\text{''}) \quad = \quad ((\underline{f} \circ_L \text{``}1^X_{C'(1,1)} \circ_W 1 -_{XY}\text{''}) \circ_L \epsilon^N_W) \quad : \text{transf}\ C(W,XY)\ (GNW)\ C(F,NL)\ Z$

J

Y

$g : \hom_{D(M,GNX)} Z\ Y$

I

$C''(XY,L)$

$(\epsilon^N_X \circ_M g) : \hom_{C(FM,N)} Z\ (XY)$

X

Z

C'

N $== \epsilon^N_X : \hom_{C(F,N)} (GNX)\ X \Longrightarrow$ X $- \circ_L (\epsilon^N_X \circ_M g)$
$: \mathrm{transf}\ C''(XY,L)\ Z\ C(FM,NL)\ 1$

C C'''

G $\epsilon^N_X \circ_M -$
$: \mathrm{transf}\ D(M,GNX)\ 1\ C(FM,N)\ X$

L

D $\xrightarrow{\ \ F\ \ }$ C $\xleftarrow{\ \ N\ \ }$ C''
      $(F \dashv G)$

M

$C(FM,N)$

E

naturality (counit, covariant eq, left of eq): $- \circ_L(\epsilon^N_X \circ_M g)$ $=$ $\epsilon^{NL}_Y \circ_M (\text{``}GN-_X \circ_L 1\text{''} \circ_M g)$: transf $C''(XY,L)\ Z\ C(FM,NL)\ 1$

$$C''$$

$$C'(XY,L)$$

$$L$$

$$- \circ_L (1^X_{D(1,GN)} \circ_M \_g\_)$$
$$= ``GN -_X \circ_L 1" \circ_M \_g\_$$
$$: \text{transf } C'(XY,L) \ Z \ D(M,GNL) \ 1$$

$$J \xleftarrow{\quad Y \quad} I \xrightarrow{\quad X \quad} C'$$

$$X$$

$$N$$

$$\epsilon^{NL}_1 : \hom_{C(F,NL)} (GNL) \ 1$$

$$1$$

$$C'$$

$$g : \hom_{D(M,GNX)} Z \ Y$$

$$C$$

$$\epsilon^{NL}_1 \circ_M (``GN - \circ_L 1" \circ_M \_g\_)$$
$$: \text{transf } C'(XY,L) \ Z \ C(FM,NL) \ 1$$

$$N$$

$$C$$

$$Z$$

$$G$$

$$G$$

$$\epsilon^{NL}_1 \circ_M - : \text{transf } D(M,GNL) \ 1 \ C(FM,NL) \ 1$$

$$D \xrightarrow[\substack{F \\ (F \dashv G)}]{} C \xleftarrow{\quad N \quad} C' \xleftarrow{\quad L \quad} C''$$

$$M$$

$$C(FM,NL)$$

$$E$$

naturality (counit, covariant eq, right of eq): $\quad - \circ_L(\epsilon^N_X \circ_M \_g\_) \quad = \quad \epsilon^{NL}_1 \circ_M (``GN -_X \circ_L 1" \circ_M \_g\_) \quad : \text{transf } C'(XY,L) \ Z \ C(FM,NL) \ 1$

$J \xrightarrow{\quad Y \quad} D' \xrightarrow{\quad M \quad} D \xrightarrow{\quad F \quad} C$

$\eta_Y^M : \hom_{D(M,G)} Y\ (FMY)$

Y

$D'(Z,Y)$

$\eta_Y^M \circ_Z -$
$: \text{trans } D'(Z,Y)\ 1\ D(MZ,G)\ (FMY)$

$G \quad \Longrightarrow \quad \epsilon_1^1 : \hom_{C(F,1)} G\ 1 \quad \Longrightarrow \quad 1$

$D \xrightarrow{\quad F \quad} C$

M

$\epsilon_1^1 \circ_{MZ} -$
$: \text{transf } D(MZ,G)\ 1\ C(FMZ,1)\ 1$

D'

$C(FMZ,1)$

Z

D''

beta cancellation (contravariant, instance): $\qquad \epsilon_1^1 \circ_{MZ} (\eta_Y^M \circ_Z -) \quad = \quad \text{``}1 \circ_Z FM -_Y\text{''} \qquad : \text{transf } D'(Z,Y)\ 1\ C(FMZ,1)\ (FMY)$

I

X

C'

$C(FMY, NX)$

N $\epsilon_X^N : \hom_{C(F,N)} (GNX)\, X$ X

$-\circ_{NX} \eta_Y^M :$
transf $C(FMY, NX)\, Y\, D(M, GNX)\, 1$

$J \xrightarrow{\;Y\;} D' \xrightarrow{\;M\;} D \xrightarrow{\;F\;} C$

G

$\epsilon_X^N \circ_M - :$
transf $D(M, GNX)\, 1\, C(FM, N)\, X$

$\eta_Y^M : \hom_{D(M,G)}\, Y\, (FMY)$

$D \xrightarrow{\;F\;} C \xleftarrow{\;N\;} C'$

Y

M

$C(FM, N)$

D'
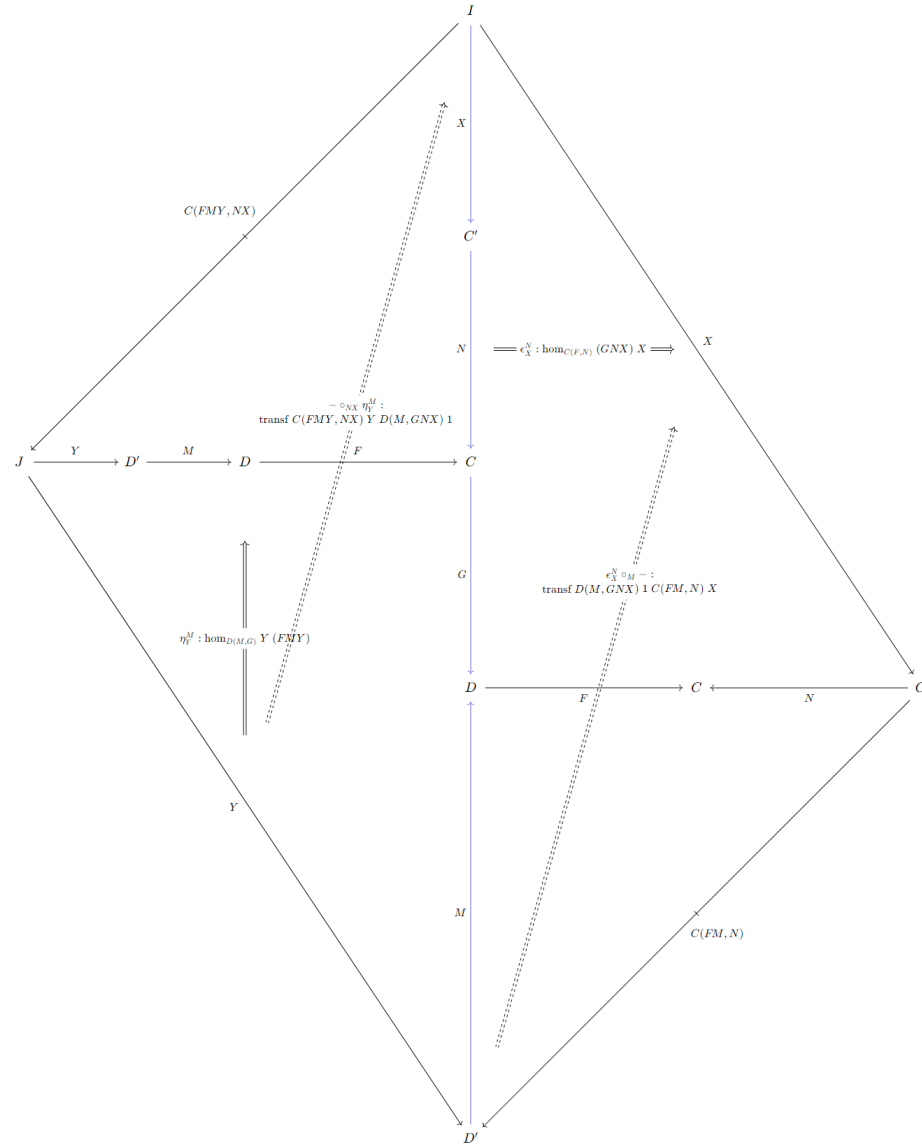
beta cancellation (instance): $\epsilon_X^N \circ_M (-\circ_{NX} \eta_Y^M)$ =   id_cast   : $C(FMY, NX)\, Y\, C(FM, N)\, X$

Synthetic fibred Yoneda.

Lemma 4.41.1 (2-Yoneda lemma for fibred categories). Let C be a category. Let $S \to C$ be a fibred category over C. Let $U \in Ob(C)$. The functor

$$Mor_{Fib/C}\,(C/U\,,S) \longrightarrow S_U$$

given by $G \mapsto G(id_U)$ is an equivalence, whose inverse:

$$S_U \longrightarrow Mor_{Fib/C}\,(C/U\,,S)$$

is such that given $x \in Ob(S_U)$ the associated functor is:

- on objects: $(f : V \to U) \mapsto f^*x$, and

- on morphisms: the arrow $(g : V'/U \to V/U)$ maps to the universality morphism $(f \circ g)^*x \to f^*x$ fibred over the arrow g.

=]

$$y^* f'$$

$$(f')^* y \in S_{V'} \dashrightarrow \exists! \dashrightarrow f^* x \in S_V \xrightarrow{\quad x^* f \quad} x \in S_U \xleftarrow{\quad E \quad} y \in S_{U'}$$

$$V' \xrightarrow{\quad g \quad} V \xrightarrow{\quad f \in (C/U)_V \quad} U \xleftarrow{\quad e \quad} U'$$

$$f' \in (C/U')_{V'} \quad , \quad e \circ f' = f \circ g$$
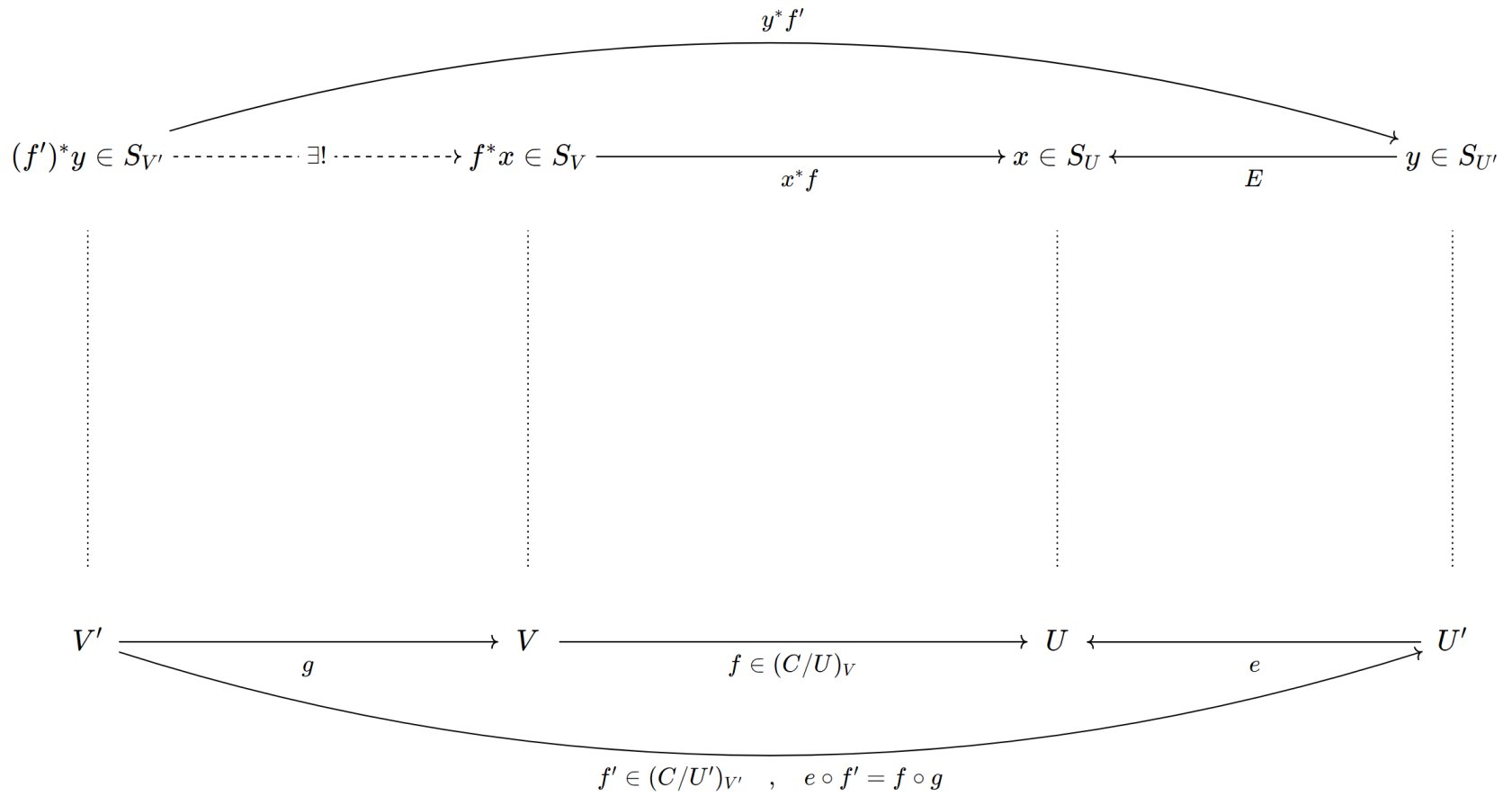
Substructural fibred Yoneda.

• Expressed as a blend of the *generalized (above-any-arrow) universality/introduction rule* of the transported/pulledback objects, via the (fibred) Yoneda formulation, inside fibred categories:

```
  symbol Fibration_con_funcd : Π [I X X'  : cat] [x'x : func X' X] [G : func X I]
[JJ : catd X'] [F : func X' I] [II : catd I] (II_isf : isFibration_con II),

Π (FF : funcd JJ F II) (f : hom x'x (Unit_mod G Id_func) F),
funcd JJ x'x (Fibre_catd II G);


  constant symbol Fibration_con_intro_homd : Π [I X X'  : cat] [x'x : func X' X] [G
: func X I] [JJ : catd X'] [F : func X' I] [II : catd I] (II_isf : isFibration_con
II) (FF : funcd JJ F II) (f : hom x'x (Unit_mod G Id_func) F) [X'0 : cat] [x'0x :
func X'0 X] [X'' : cat] [x''x : func X'' X] [x''x'0 : func X'' X'0] (x'0x' :  hom
x''x'0 (Unit_mod x'0x x'x) x''x') [KK : catd X'0] (GG : funcd KK x'0x (Fibre_catd II
G)) [HH : funcd (Fibre_catd JJ x''x') x''x'0 KK],

homd ((x'0x' '∘ ((x'0x)_'∘> f))) HH (Unit_modd (GG ∘>d (Fibre_elim_funcd II G))
Id_funcd) ((Fibre_elim_funcd JJ (x''x')) ∘>d FF) →
homd x'0x' HH (Unit_modd GG (Fibration_con_funcd II_isf FF f)) (Fibre_elim_funcd JJ
(x''x'));
```

- blended together with the *(covariant) composition operation*, via the

(indexed) Yoneda formulation, inside fibred categories:

```
  constant symbol ∘>d'_ : Π [X Y I: cat] [F : func I X] [R : mod X Y] [G : func I Y]
[r : hom F R G] [A : catd X] [B : catd Y] [II] [FF : funcd II F A] [RR : modd A R B]
[GG : funcd II G B],

homd r FF RR GG →
Π [J: cat] [M : func J Y] [JJ : catd J] (MM : funcd JJ M B),
transfd ( r ∘>'_(M) ) (Unit_modd GG MM) FF (RR d<<∘ MM ) Id_funcd;
```

- Thereby this blend allows to express the outer (first) functorial-action by S_U

or the inner functorial-action by C/U (or both simultaneously) in the mapping:

$$S_U \longrightarrow Mor_{Fib/C}\,(C/U\,,S)$$

Comma elimination ("J-rule arrow induction").

• The above intrinsic/structural universality formulation comes with a corresponding *reflected/internalized algebra formulation*, which is the comma category where the J-rule elimination ("equality/path/arrow induction") occurs.

```
constant symbol Comma_con_intro_funcd : ⊓ [A B I : cat] [R : mod A B] (BB : catd B)
[x : func I A] [y : func I B] (r : hom x R y),

funcd (Fibre_catd BB y) x (Comma_con_catd R BB);


constant symbol Comma_con_elim_funcd : ⊓ [I X : cat] (G : func X I) [II : catd I]
(II_isf : isFibration_con II) [JJ : catd I] (FF : funcd JJ Id_func II),

funcd (Comma_con_catd (Unit_mod G Id_func) JJ) G II;
```

• Similarly, pullbacks have a universal formulation (fibre of fibration), an algebraic formulation (composition of spans), or mixed (product of fibration-objects in the slice category).

$$I(G,-) \downarrow KK \xleftarrow{\quad \text{intro}_f \quad} KK \xrightarrow{\quad \forall FF \quad} II$$

$$\exists \, \text{elim}(FF)$$

$$\text{elim}(FF) \circ \text{intro}_f = f^*FF$$

fib

fib

$$X \xleftarrow{\quad x \quad} X' \xrightarrow{\quad F \quad} I$$

$$f$$

$$G$$
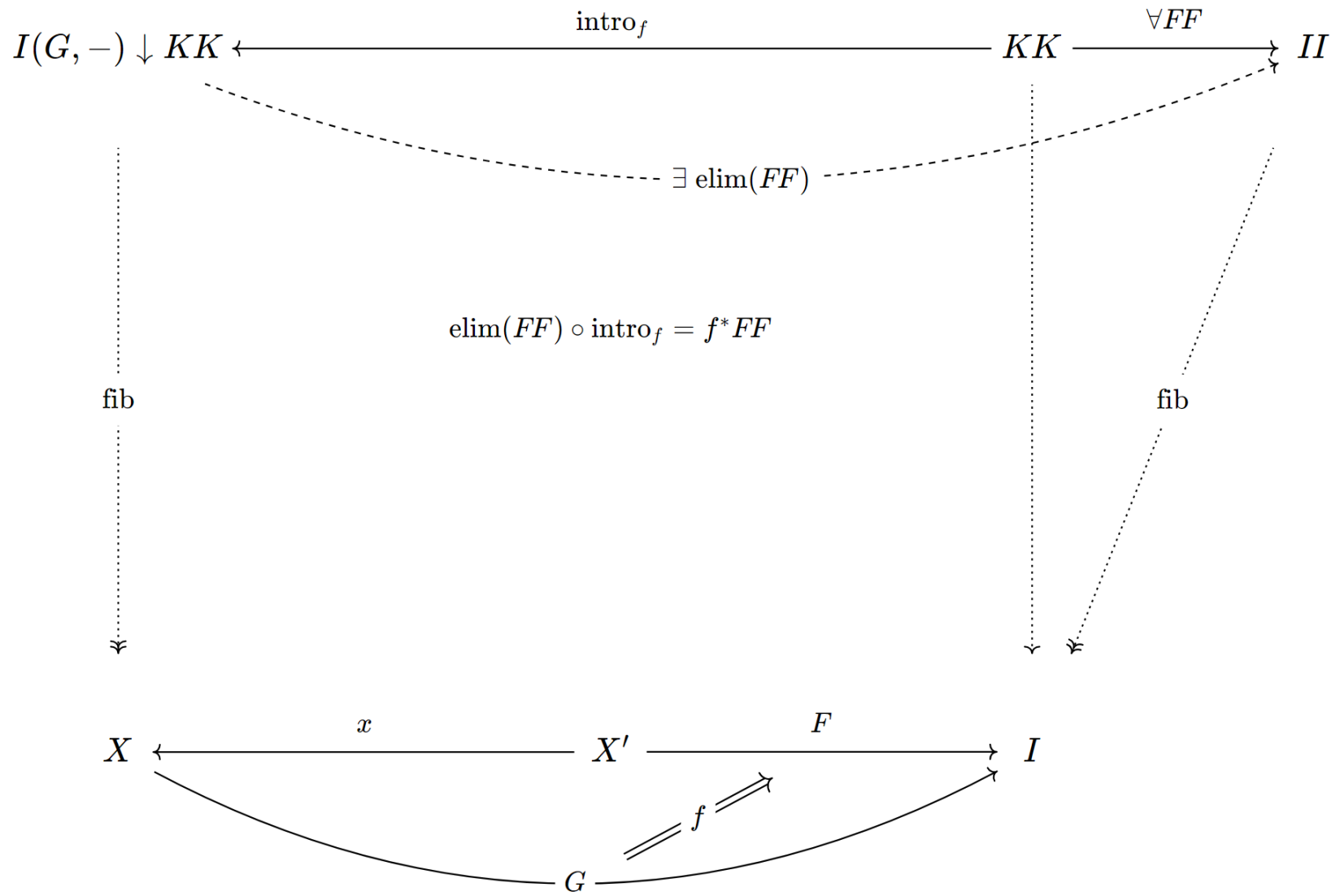
Cut-elimination for fibred arrows.

- *Non-fibred* composition cut-elimination only considers pairs of arrows:

$$p: X \to \boldsymbol{Y} \ \text{ then } \ q: \boldsymbol{Y} \to Z$$

- *Fibred* composition cut-elimination should also consider pairs of arrows:

$$p: A \to f^* \boldsymbol{B} \ \text{ then } \ q: g^* \boldsymbol{B} \to C$$

- Therefore, *grammatically* any fibred arrow should be fibred over a span-of-arrows, instead of over one object (the identity arrow), or more generally should be fibred over a *polynomial-functor* with intrinsic *distributivity* ($\Pi\Sigma = \Sigma\Pi\varepsilon^*$):

$$r: g^* A \longrightarrow f^* Z$$

- All these *intrinsic* structures are reflected/internalized as an *explicit substitution/pullback-type-former* for any fibration.

$$A \xrightarrow{\quad p \circ q := (g'^* p) \circ (f'^* q) \, : \, g'^* A \to f'^* C \quad} C$$

$$A \xrightarrow{\quad p : A \to f^* B \quad} B \xrightarrow{\quad q : g^* B \to C \quad} C$$

$$X \xrightarrow{\quad f \quad} Y \xleftarrow{\quad g \quad} Z$$

$$g' := f^* g \qquad\qquad f' := g^* f$$

$$X \times_Y Z$$
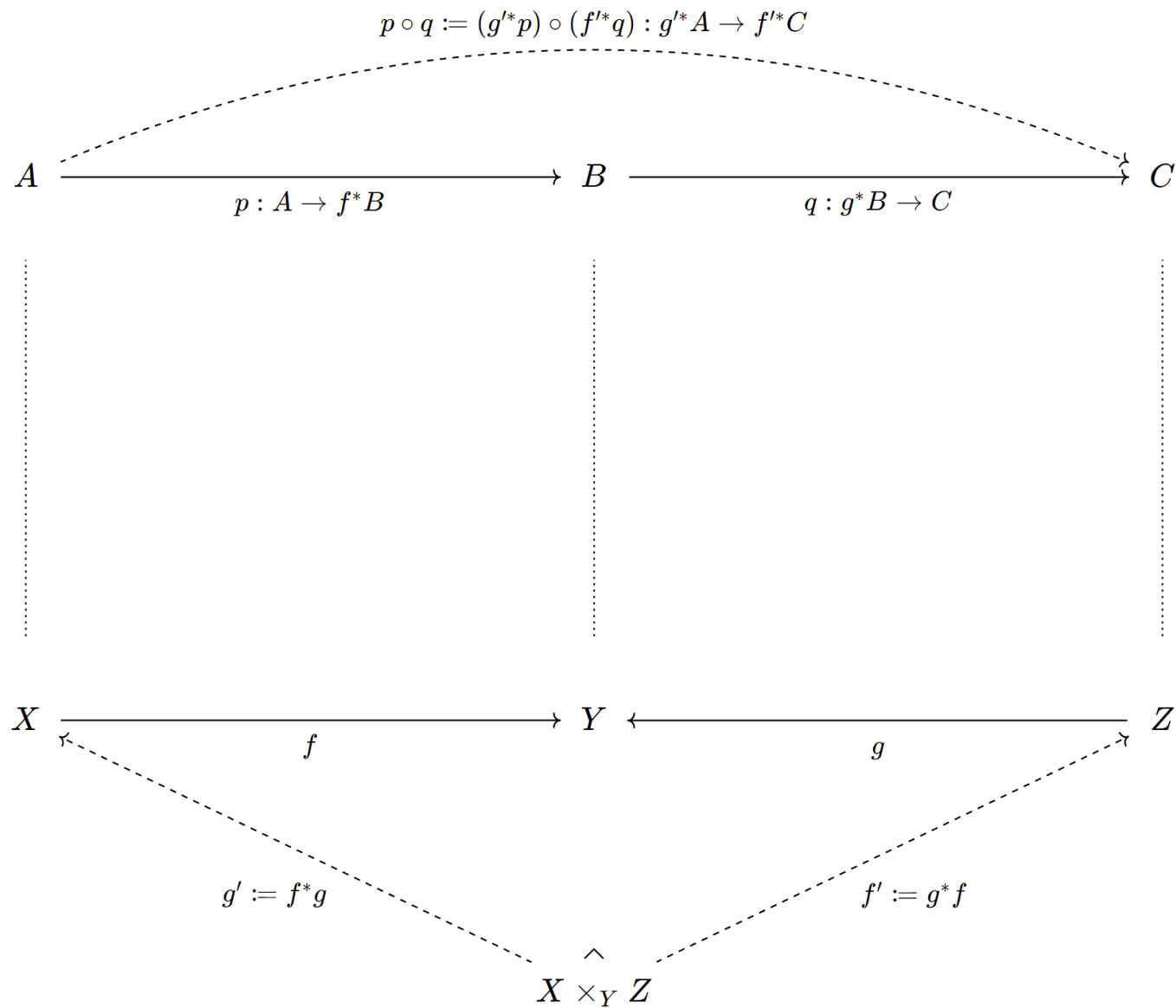
Pi-category-of-fibred-functors and Sigma-category.

• Ordinary Pi/product-types consider only the sections of some fibration, but Pi-categories should consider all the category-of-fibred-functors to some fibration, this leads to the new construction of *Pi-category-of-functors*.

• This preliminary implementation does not yet use the more-intrinsic formulation using *functors fibred-above-a-span of functors*. But *Sigma/sum-categories* can be already intrinsically-implemented using only functors fibred-forward-above a single functor.

```
constant symbol Pi_intro_funcd : Π [X' X Y : cat] (K : func X X') (A : catd X)  (F :
func X Y) (B : catd Y) [X''] (D : catd X'') (L : func X'' X')  (FF : funcd
(Productd_catd (Fibre_catd A (Pullback_snd_func L K)) (Fibre_catd D
(Pullback_fst_func L K))) ((Pullback_snd_func L K) ◦> F) B),
funcd D L (Pi_catd K A F B);



constant symbol Pi_elim_funcd : Π [X' X Y : cat] (K : func X X') (A : catd X)  (F :
func X Y) (B : catd Y),
funcd (Productd_catd A (Fibre_catd (Pi_catd K A F B) K)) F B;
```

$$\forall\, FF \text{ over } L^*K, (K^*L)^*A, (F \circ K^*L)$$

$$D \dashrightarrow \Pi_{K,A,F} B \xrightarrow{\hspace{3cm}} B$$

$$\exists\, \mathrm{intro}(FF) \qquad\qquad \mathrm{elim}_{K,A,F}(B) \text{ over } K, A, F$$

$$(K^*L)^*A$$

$$A$$

$$X'' \times_{X'} X$$

$$L^*K \qquad\qquad K^*L$$

$$X'' \xleftarrow{\hspace{2cm}} X' \xleftarrow{\hspace{2cm}} X \xrightarrow{\hspace{2cm}} Y$$

$$L \qquad\qquad K \qquad\qquad F$$
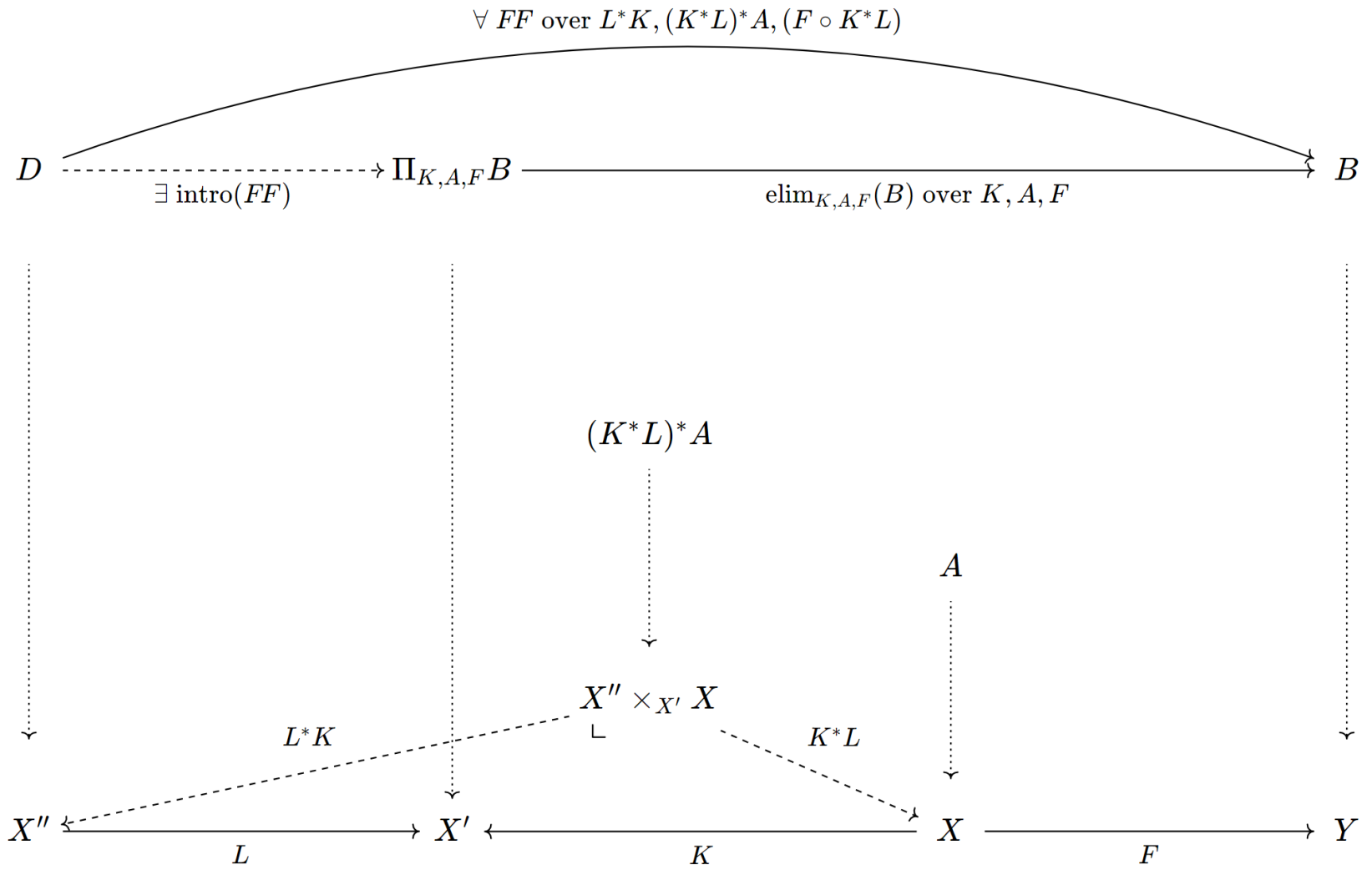
And why profunctors (of sets)?

• The primary motivation is that they form a monoidal bi-closed double category (*functorial lambda calculus*).

• Another motivation is that the subclass of fibrations called *discrete/groupoidal fibrations can only be computationally-recognized*/expressed instead via (indexed) presheaves/profunctors of sets. And the comma construction is how to recover the intended discrete fibration.

• Ultimately profunctors enriched in preorders/quantales instead of mere sets could be investigated.

• Visualizing the comma/slice category as a fibred category of *triangles of arrows fibred by their base*:

X'

$x'x$

$t \circ x'x$

X $\xrightarrow{\hspace{4cm}}$ Y

$t \in {}_R/Y$

X'

$x'x \in A(X', X)$

X
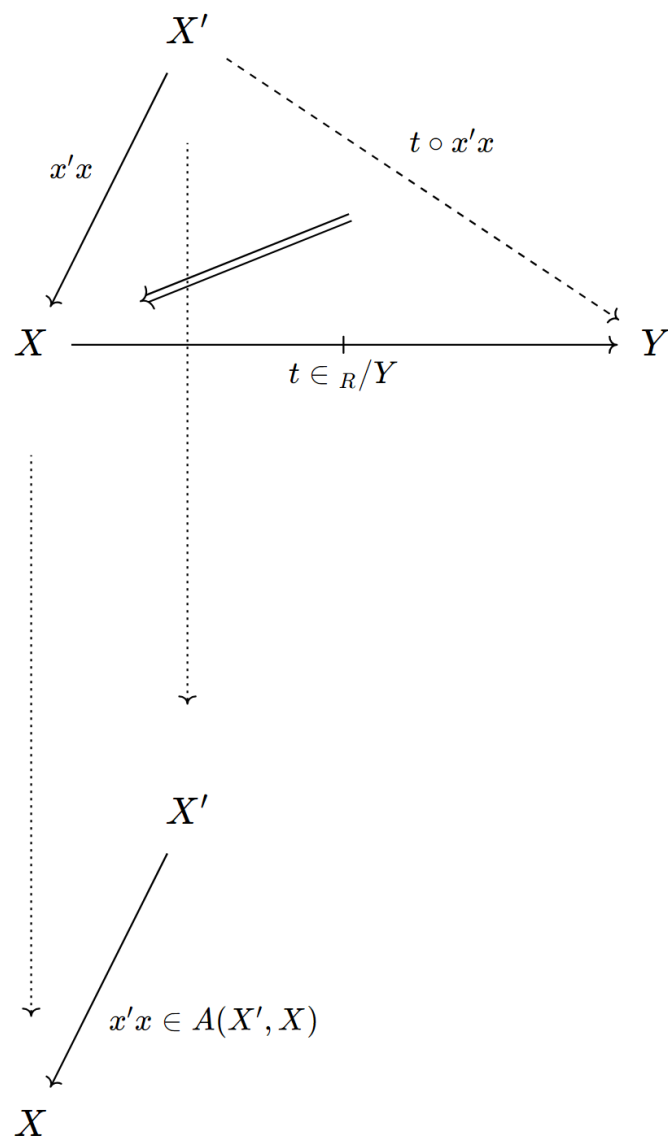
What is a fibred profunctor anyway?

• The comma/slice categories are only fibred categories (of triangles of arrows fibred by their base), not really fibred profunctors. One example of fibred profunctor from the coslice category to the slice category is *the set of squares fibred by their diagonal* which witnesses that this square is constructed by pasting two triangles.

```
constant symbol Comma_homd : Π [A B I : cat] (R : mod A B) [x : func I A] [y : func
I B], Π [J0] [F : func J0 A] [x' : func I J0] (x'x : hom x' (Unit_mod F x) Id_func),
Π [J1] [G : func J1 B] [y' : func I J1] (yy' : hom Id_func (Unit_mod y G) y'), Π (s
: hom x' (F ∘>> R) y) (t : hom x (R <<∘ G) y') (r : hom x' (F ∘>> (R <<∘ G)) y'),

π (( x'x '∘  ((F)_'∘> t) ) = r) → π (( (s ∘>'_(G)) ∘' yy' ) = r) →
homd r (Comma_con_intro_funcd (Triv_catd B) s)
   ((Comma_con_comp_funcd R (Triv_catd B) F) ∘>>d ((Comma_modd (Triv_catd A) R
(Triv_catd B)) d<<∘ (Comma_cov_comp_funcd (Triv_catd A) R G)))
   (Comma_cov_intro_funcd (Triv_catd A) t);
```

• This text implements such fibred profunctor of (cubical) squares (thereby validating the hypothesis that computational-cubes should have connections/diagonals...).

• For witnessing the (no-computational-content) pasting along the diagonal, this implementation uses for the first time the *LambdaPi-metaframework's equality predicate* which internally-reflects all the conversion-rules; in particular the implementation uses here *the categorial-associativity equation axiom, which is a provable metatheorem* which must *not* be added as a rewrite rule!

X''

$x''x'$

$s \in X'/_R, \ \in_R/Y$

X' ————————→ Y

$x'x \in A(X', X)$

$=$

$t \circ x'x$

$yy' \in B(Y, Y')$

X ————————→ Y'

$t \in_R/Y'$

X'

$r \in R : B \nrightarrow A$

$t \circ x'x = r = yy' \circ s$ Y'
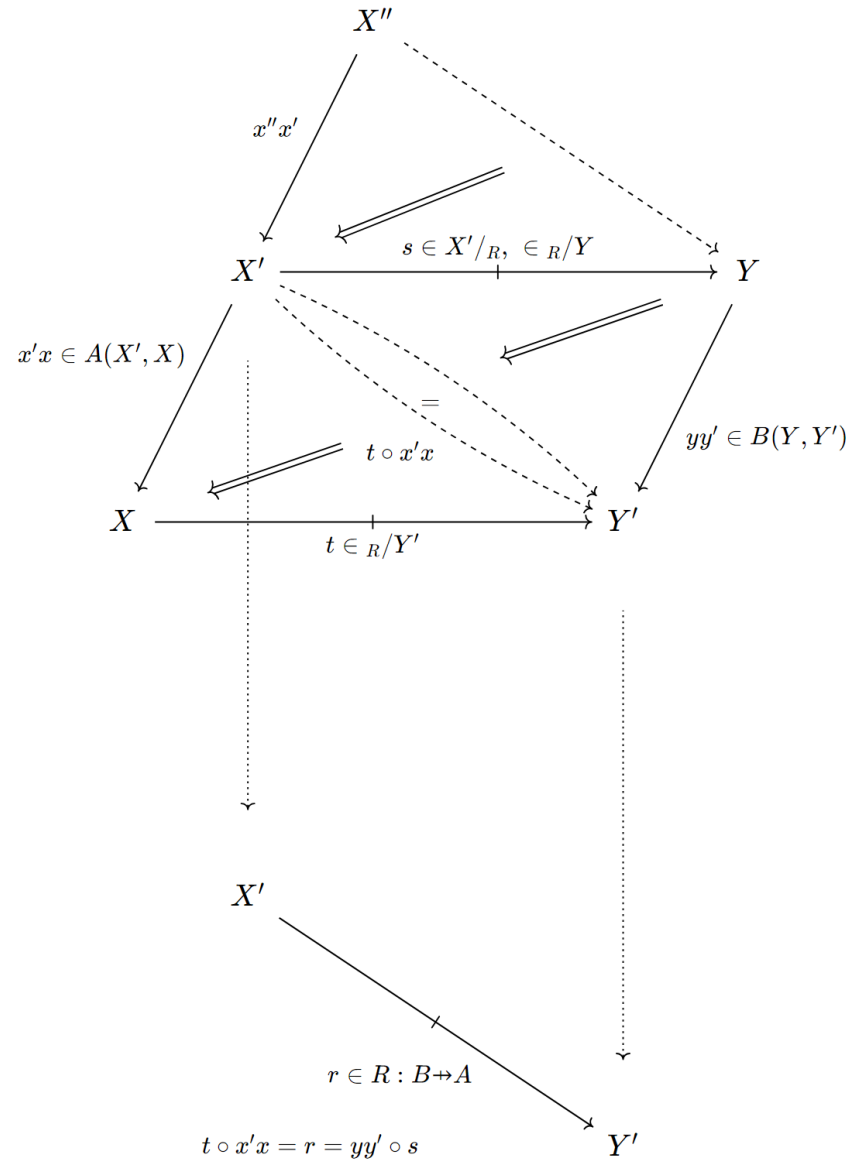
Higher inductive types, the interval type, concrete categories.

• This text implements (higher) inductive datatypes such as the *join-category* (interval simplex) and *concrete categories* (ref. the section "Applications"), with their introduction/elimination/computation rules. The (3-dimensional) *naturality cone conditions*, which relate the (2-dimensional) arrows introduced by the introduction-rules, are expressed using the LambdaPi metaframework conversion rewrite rules.

```
symbol join_cat : Π (A B : cat), cat;

symbol join_fst_func : Π (A B : cat), func A (join_cat A B);

symbol join_snd_func : Π (A B : cat), func B (join_cat A B);

symbol join_hom :  Π (A B : cat) [I : cat] (a : func I A) (b : func I B),
hom a (Unit_mod (join_fst_func A B) (join_snd_func A B)) b;


rule @'∘ _ _ _ _ _ _ _ $a' Id_func _ _ $r (( Id_func ) _'∘> (join_hom $A $B $a $b))
↪ (join_hom $A $B $a' $b);
```

- *But for the elimination rules*, those (3-dimensional) naturality cone conditions are expressed using the LambdaPi equality predicate = (which internally-reflects its rewrites rules), to relate the (2-dimensional) arrows arguments. Note that the naturality cone conditions *carry no computational content* and are only logical consistency checks.
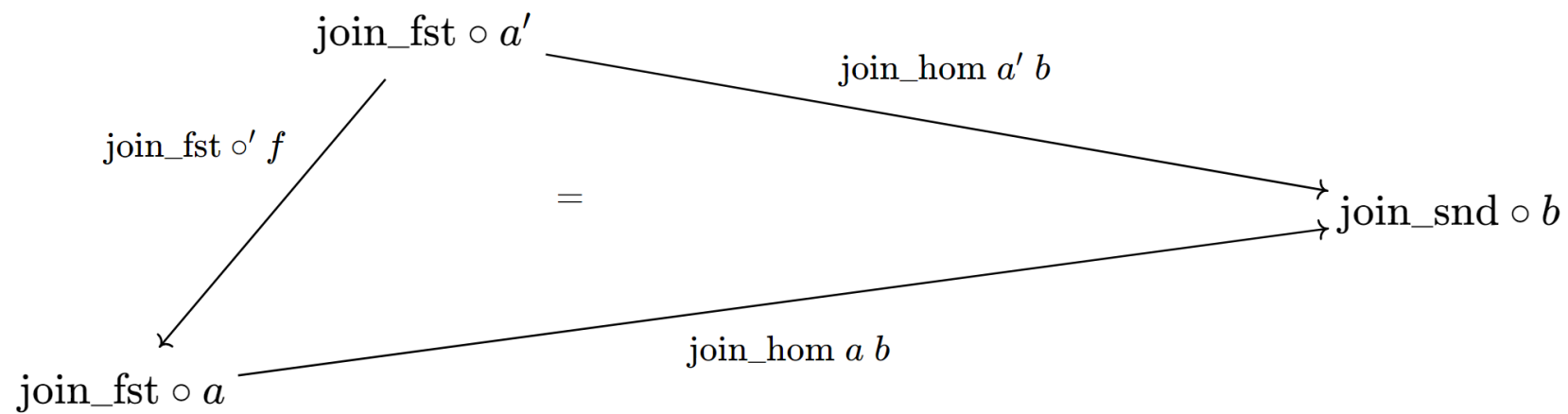
```
symbol join_elim_con_func : Π (A B : cat) [E : cat] (first_func : func A E)
(second_func : func B E) (one_hom : Π (I : cat) (a : func I A) (b : func I B), hom a
(Unit_mod (first_func) Id_func) (second_func <∘ b))

(natural_eq : Π [I : cat] (a : func I A) (b : func I B) [a'] (r : hom a' (Unit_mod
Id_func a) Id_func),
      π (r  '∘ (( _ ) _'∘> (one_hom I a b)) = (one_hom I a' b))),
   func (join_cat A B) E;



rule join_fst_func $A $B ∘> (join_elim_con_func $A $B $F0 $F1 $r _) ↪ $F0
with  join_snd_func $A $B ∘> (join_elim_con_func $A $B $F0 $F1 $r _) ↪ $F1;


rule ((join_hom $A $B $a $b) '∘ ((join_fst_func $A $B) _'∘>
 (Func_con_hom (join_elim_con_func $A $B $first_func $second_func $one_hom _)
              (join_snd_func $A $B)))) ↪ $one_hom _ $a $b ;
```

$\text{join\_fst} \circ a'$

$\text{join\_fst} \circ' f$

$\text{join\_hom } a' \ b$

$=$

$\text{join\_snd} \circ b$
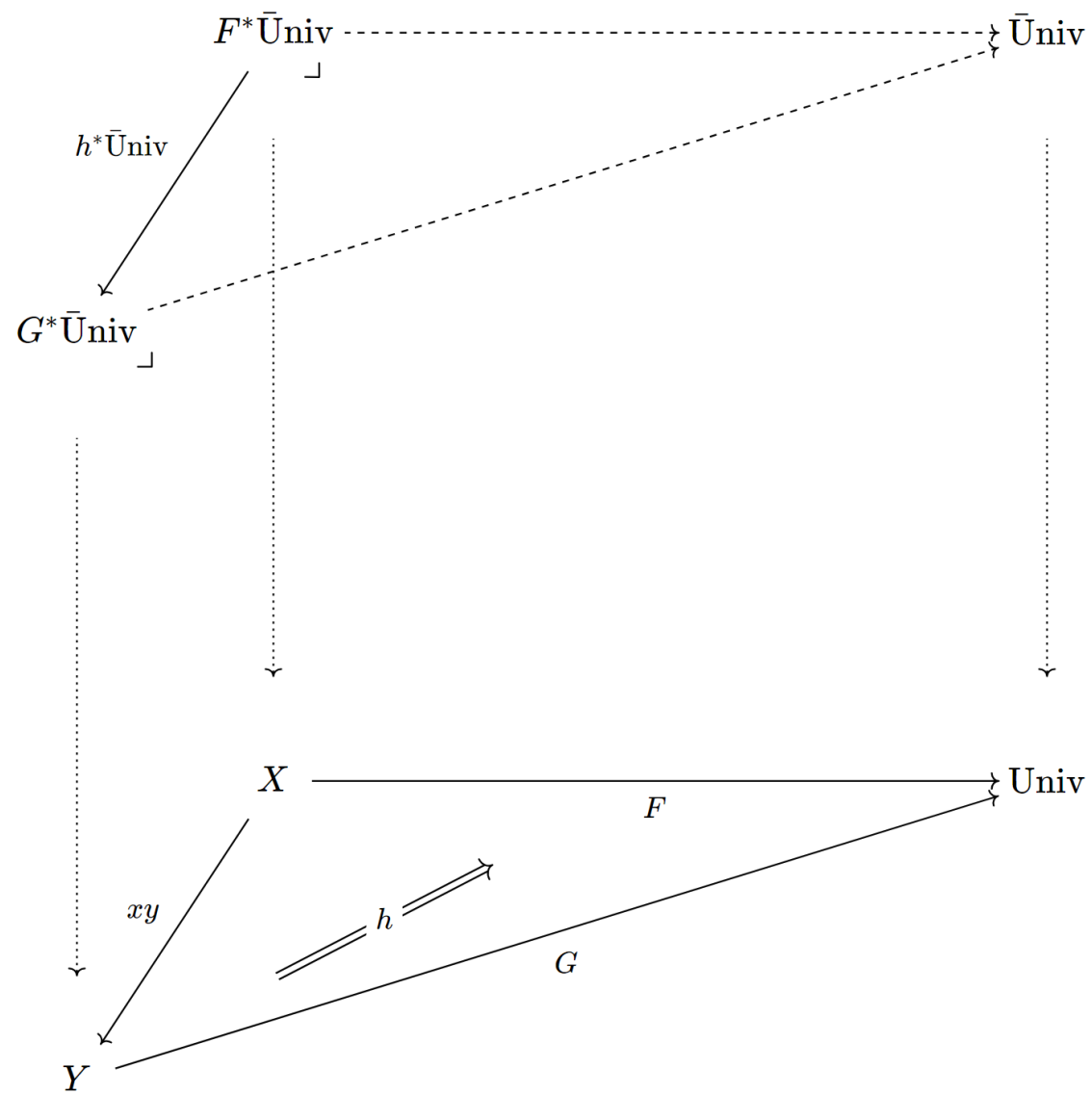
$\text{join\_fst} \circ a$

$\text{join\_hom } a \ b$

Universe, universal fibration.

• This text implements a *grammatical (univalent) universe* and the universal fibration classifying small fibrations, together with the dual universal opfibration.

• This universe is made grammatical (univalent) by declaring an inverse to the fibrational-transport inside the universe fibration.

```
constant symbol Universe_con_cat : cat;

constant symbol Universe_con_catd : catd Universe_con_cat;

symbol Universe_con_func : Π [X : cat] (A : catd X) (A_isf : isFibration_con A),
func  X Universe_con_cat;

symbol Universe_con_funcd : Π [X : cat] (A : catd X) (A_isf : isFibration_con A),
funcd A (Universe_con_func A A_isf) Universe_con_catd;


injective symbol Universe_Fibration_con_funcd_inv : Π [X Y: cat] (F : func X
Universe_con_cat) (G : func Y Universe_con_cat) [xy : func X Y],

funcd (Fibre_catd Universe_con_catd F) xy (Fibre_catd Universe_con_catd G) →
hom xy (Unit_mod G Id_func) F;
```

$$F^*\bar{\mathrm{U}}\mathrm{niv} \dashrightarrow \bar{\mathrm{U}}\mathrm{niv}$$

$$h^*\bar{\mathrm{U}}\mathrm{niv}$$

$$G^*\bar{\mathrm{U}}\mathrm{niv}$$

$$X \xrightarrow{F} \mathrm{Univ}$$

$$xy \qquad h \qquad G$$

$$Y$$

Weighted limits.

- This text implements *profunctor-weighted limits* (that the *right-extension* $\mathrm{Hom}(-, F) \Leftarrow W$ is representable as $\mathrm{Hom}(-, \lim^W F)$) and *profunctor-weighted colimits* (that the *right-lifting* $W \Rightarrow \mathrm{Hom}(F, -)$ is representable as $\mathrm{Hom}(\mathrm{colim}^W F, -)$).

- And it can *computationally-prove* that left-adjoint functors preserve profunctor-weighted colimits from its computational-proof that *right-adjoint functors preserve profunctor-weighted limits.*

- A computational-proof is to be contrasted from a logical deduction which uses the reflected/internalized LambdaPi propositional equality.

```
constant symbol limit_cov : Π [B J0 J J' : cat] (K : func J J0) (F : func J0 B) (W :
mod J' J) (F_⇐_W : func J' B), TYPE;



injective symbol limit_cov_univ_transf : Π  [B J J' : cat] [W : mod J' J] [F : func
J B] [F_⇐_W : func J' B]

 (isl : limit_cov F W F_⇐_W), Π [I : cat] (M : func I B),

 transf (((Unit_mod M F)) ⇐ W) Id_func (Unit_mod M F_⇐_W) Id_func;



symbol righ_adjoint_preserves_limit_cov [B J J' A : cat] [W : mod J' J] [F : func J
B] [F_⇐_W : func J' B] (isl : limit_cov F W F_⇐_W) [R : func B A] [L : func A B]
(isa : adj L R) [I : cat] (M : func I A) :

  transf ((Unit_mod M (F ∘> R)) ⇐ W) Id_func (Unit_mod M R <<∘ F_⇐_W) Id_func

  ≔ ((Adj_con_hom isa M Id_func) ∘>'_(F_⇐_W)) ∘''
            ((limit_cov_univ_transf isl (M ∘> L)) ∘''
             (Imply_cov_transf ((M)_'∘> Adj_cov_hom isa F Id_func) (Id_transf W)));
```
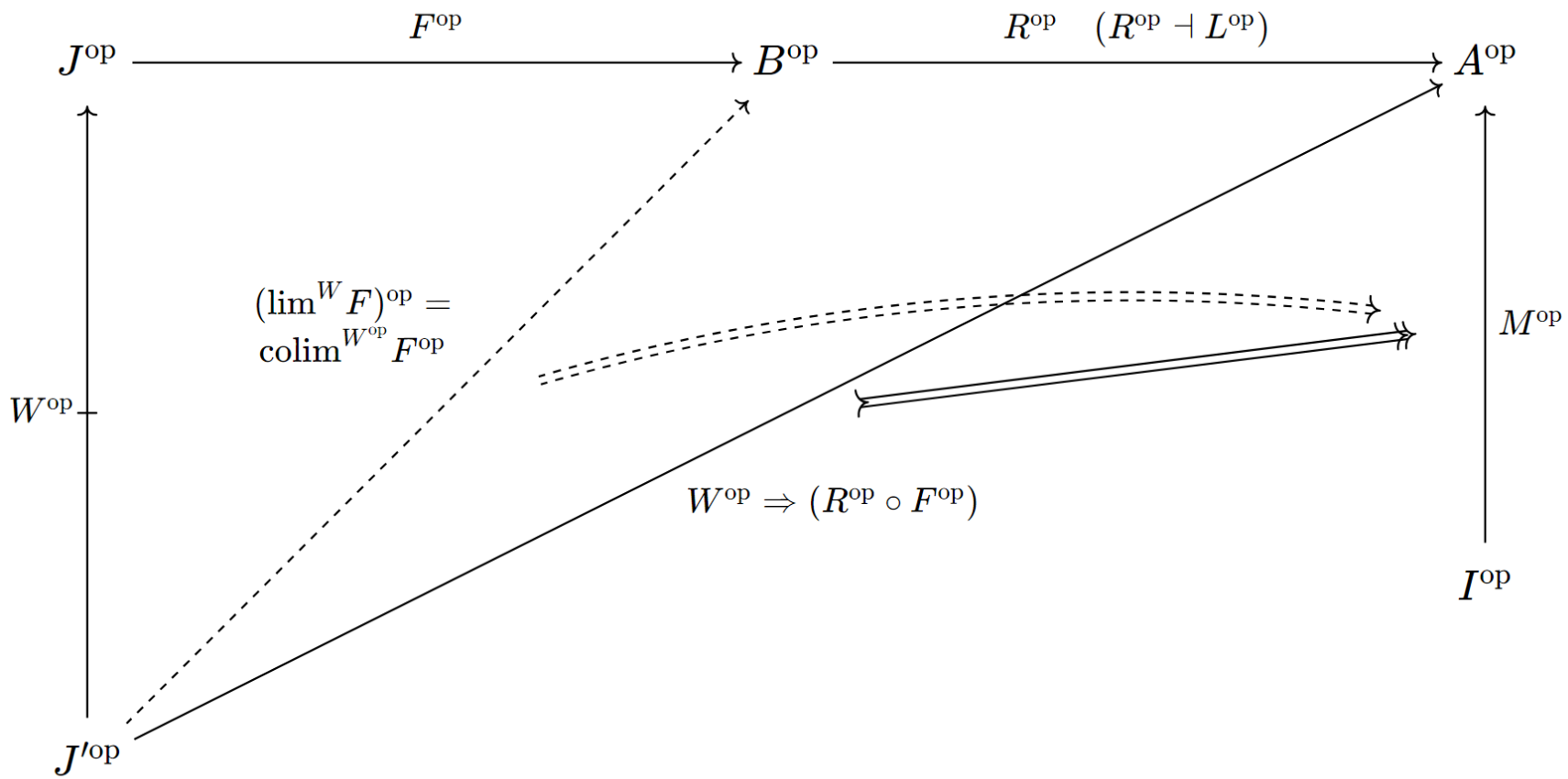
I

M

$\phi \in A(M, R \circ F) \Leftarrow W$

$(R \circ F) \Leftarrow W$

J'

W

$\lim^W F$

A

$R \quad (L \dashv R)$

B

F

J
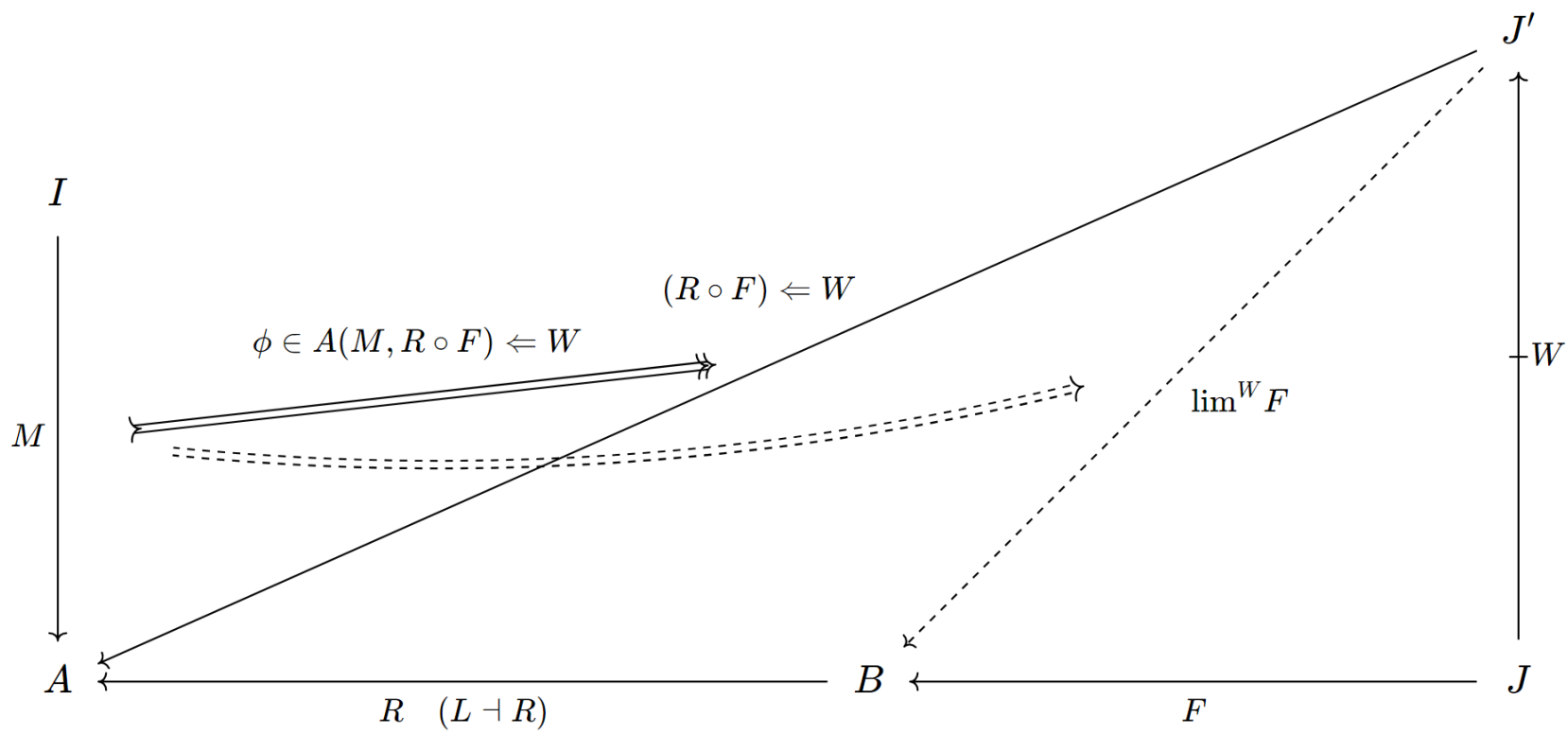
Duality Op, covariance vs contravariance.

• This text implements the *dualizing Op* operations for categories, functors, profunctors/modules, hom-arrows, transformations, adjunctions, limits, fibrations… which are used to computationally-prove that left-adjoint functors preserve profunctor-weighted colimits from the proof that right-adjoint functors preserve profunctor-weighted limits.

```
symbol left_adjoint_preserves_limit_con [B J J' A : cat] [W : mod J J'] [F : func J
B] [W_⊗_F : func J' B] (isl : limit_con F W W_⊗_F) [R : func A B] [L : func B A]
(isa : adj L R) [I : cat] (M : func I A) :

  transf (W ⇒ (Unit_mod (F ∘> L) M)) Id_func (W_⊗_F ∘>> Unit_mod L M) Id_func

  ≔ Op_transf (righ_adjoint_preserves_limit_cov (Op_limit_cov isl) (Op_adj isa)
(Op_func M));
```

$$J^{\mathrm{op}} \xrightarrow{\quad F^{\mathrm{op}} \quad} B^{\mathrm{op}} \xrightarrow{\quad R^{\mathrm{op}} \quad (R^{\mathrm{op}} \dashv L^{\mathrm{op}}) \quad} A^{\mathrm{op}}$$

$$(\lim{}^{W} F)^{\mathrm{op}} = \mathrm{colim}^{W^{\mathrm{op}}} F^{\mathrm{op}}$$

$$W^{\mathrm{op}}$$

$$M^{\mathrm{op}}$$

$$W^{\mathrm{op}} \Rightarrow (R^{\mathrm{op}} \circ F^{\mathrm{op}})$$

$$I^{\mathrm{op}}$$

$$J'^{\mathrm{op}}$$

Grammatical topology.

• Finally, there is an experimental implementation of *covering (co)sieves towards grammatical sheaf cohomology* and towards a description of algebraic geometry's schemes in their formulation as *locally affine ringed sites* (structured topos)... The implementation of the covering (co)sieve *predicate* uses ideas from the local modifier $j: \Omega \to \Omega$ where $\Omega(A)$ is the classifier of (sub-)objects (sieves) $\mathcal{U}$ of the object A, and where $j_A(\mathcal{U})(f: X \to A) := $ "$f^*\mathcal{U} \in J(X)$" is the (opaque) set of witnesses that the pullback-sieve $f^*\mathcal{U}$ is covering. Then the transitivity axiom: $\mathcal{U} \in J(A)$ if $\big(\forall f : \text{some cover of } A, \ j_A(\mathcal{U})(f)\big)$, iff $j_A(j_A(\mathcal{U}))$, becomes expressible.

constant symbol **covering** : Π [X Y I : cat] [A : catd X] [R : mod X Y] [B : catd Y]
(RR : modd A R B) [y : func I Y], funcd (Triv_catd I) y B → TYPE ;

constant symbol **coveringu** : Π [I : cat] [A : catd I] [B : catd I] (RR : moddu A B),
funcd (Triv_catd I) Id_func B → TYPE ;

constant symbol **Total_covering** : Π [X Y I : cat] [A : catd X] [R : mod X Y] [B :
catd Y] [RR : modd A R B] [G : func I Y] [H : funcd (Triv_catd I) G B],
  (Π [F : func I X] (r : hom F R G), **coveringu** (Fibre_hom_moddu RR r)
(Fibre_intro_funcd _ G Id_func H)) →
  covering RR H ;

constant symbol **Glue_transfd** : Π [X Y X' Y': cat] [A' : catd X'] [A : catd X] [B' :
catd Y'] [B: catd Y] [xx' : func X X'] [yy' : func Y Y'] [R' : mod X' Y'] [R : mod X
Y] [rr' : transf R xx' R' yy'] [RR : modd A R B] [FF : funcd A xx' A'] [RR' : modd
A' R' B'] [GG : funcd B yy' B'], Π [I : cat] [G : func I Y] [H : funcd (Triv_catd
I) G B] , **covering** RR H →
(Π [F : func I X] (r : hom F R G), **transfduu** (Fibre_hom_moddu RR r)
  (Fibre_intro_funcd _ (F ∘> xx') Id_func (Fibre_elim_funcd A F ∘>d FF))
  (Fibre_hom_moddu (Sheaf_con_modd RR') (r '∘ rr'))
  (Fibre_intro_funcd _ (G ∘> yy') Id_func (Fibre_elim_funcd B G ∘>d GG))) →
transfd rr' RR FF (Sheaf_con_modd RR') GG;

$$F^*A \xrightarrow{\hspace{4cm} A^*F \hspace{4cm}} A$$

r^*T

T

$$G^*B \xrightarrow{\hspace{5cm}} B$$

B^*G

H

$\mathrm{intro}_G(H)$
if covers r^*T

then covers T

X

$\forall\, F$

R

$\overset{\forall\, r}{=\!\!=\!\Rightarrow}$

$$I \xrightarrow{\hspace{6cm} G \hspace{2cm}} Y$$
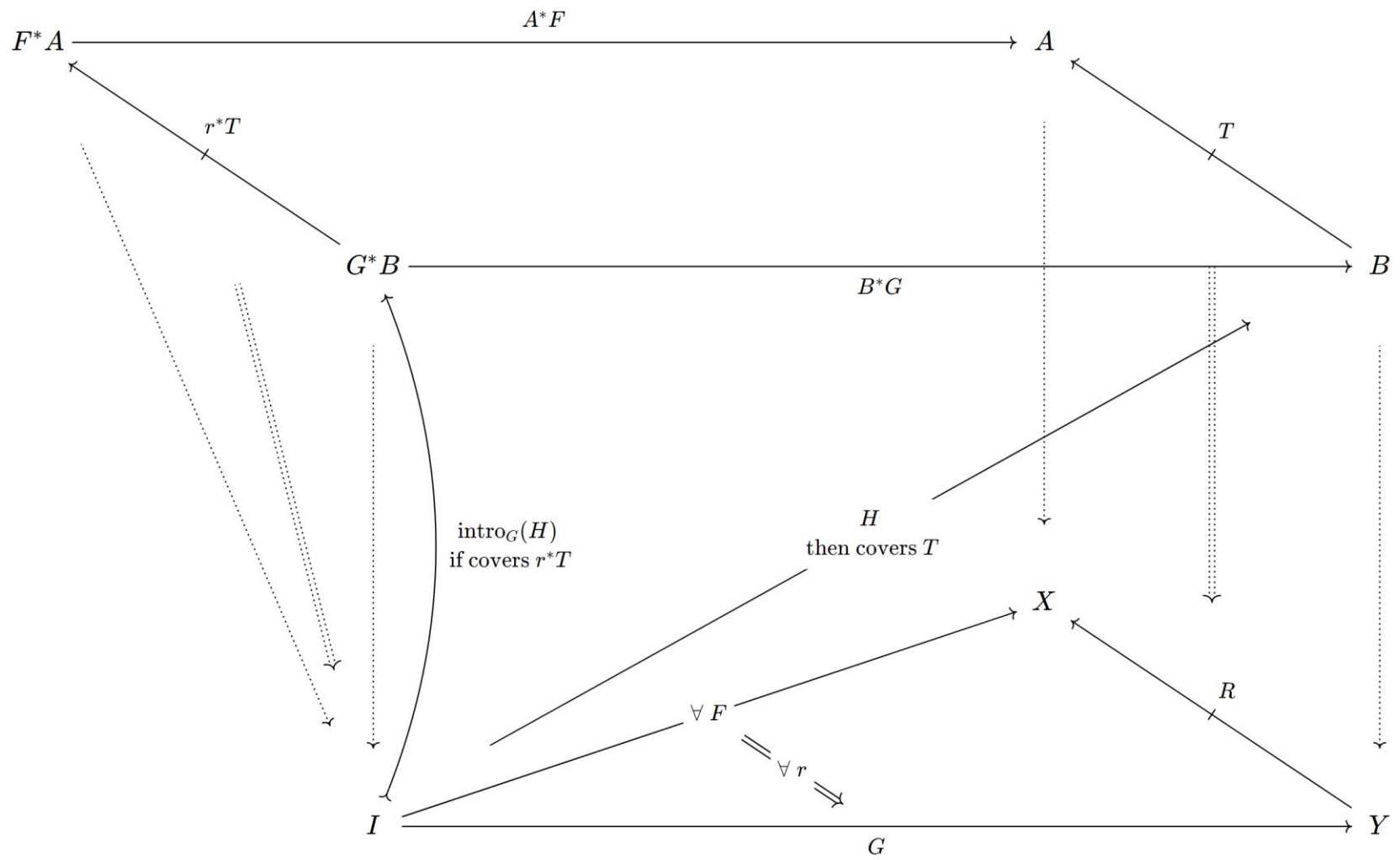
• A sheaf is data defined over some topology, and sheaf cohomology is linear algebra with data defined over some topology. A closer inspection reveals that there is some intermediate formulation which is computationally-better than Cech cohomology: at least for the standard simplexes (line, triangle, etc.), then intersections of opens could be internalized as primitive/generating opens for the cover and become points in the nerve of this cover (as suggested by the barycentric subdivision). This redundant storage space for functions defined over the topology is what allows (semantically-)possibly-incompatible functions to be (grammatically/formally)-glued regardless, and to prove the acyclicity for the standard simplex (and to compute how this acyclicity fails in the presence of holes in the nerve). For example, this sheaf data type, gives the gluing operation:

$$F(U0) := \text{sum over the slice U0 U01} = \mathbb{Z} \oplus \mathbb{Z};$$

$$F(U1) := \mathbb{Z} \oplus \mathbb{Z}; F(U01) := \mathbb{Z}$$

$$F(U) = \text{kan extension} = \mathbb{Z} \oplus \mathbb{Z} \oplus \mathbb{Z};$$

$$\text{gluing:} \, F(U0) \oplus F(U1) \oplus F(U01) \rightarrow F(U)$$

$$\big((f0, f01), (g1, g01), (h01)\big) \mapsto (f0, g1, f01 + g01 - h01)$$

where the signed sum generalizes to higher degrees because the Euler

characteristic is 1 (or inclusion–exclusion principle).

Applications: concrete datatypes and limits of finite sets.

• This text also implements (higher) inductive datatypes such as the *join-category* (interval simplex) and *concrete categories*, with their introduction/elimination/computation rules. (In the updated file, https://github.com/1337777/cartier/blob/master/cartierSolution14.lp)

• *Concrete categories*, such as the category of finite sets, are presented within the *abstract prover grammar* via *datatypes*. Now datatypes are higher types because they allow constructors for arrows, besides constructors for objects.

• The key to discover the correct formulation is to understand the terminal category also as a datatype, and thereafter use this *terminal datatype*'s primitives to formulate the other more-complex datatypes.

• The goal of this section is to demo that it is possible to do a *roundtrip* between the concrete data structures and the abstract prover grammar; this is very subtle. This new approach allows, not only to compute with concrete data, but also to do so via a *grammatical interface* which is more strongly-specified/typed and which enables the theorem proving/programming of the

correctness-by-construction of the algorithms, such as the usual *algorithm to inductively compute general finite limits/colimits* from the equalizer limits, product limits and terminal limits.

• This new approach is to be contrasted for example from the AlgebraicJulia library package, which is an attempt to add "functional language" features to the Julia numerical computing language, via category theory. This applied category theory on concrete data structures allows to achieve some amount of compositionality (function-based) features onto ordinary numerical computing. The AlgebraicJulia implementation essentially hacks and reimplements some pseudo-dependent-types domain-specific-language embedded within Julia.

• This demo now successfully works generically, including on this silly example: the limit/equalizer of a (inductive) diagram when the (inductive-hypothesis) product cone **[12;11]×[22;21]×[33;32;31]** now is given an extra constant arrow **[22;21] → [33;32;31] onto 31**, besides its old discrete base diagram.

   o The output limit cone's apex object:

```
compute obj_category_Obj ((category_Obj_obj One) ∘>o (sigma_Fst
(construct_inductively_limit_instance_liset _ example_graph_isf example_diagram)));

// (0,13,21,31) :: (0,13,22,31) :: (0,12,21,31) :: (0,12,22,31) :: (0,11,21,31) ::
(0,11,22,31) :: nil

//Pair_natUniv (Pair_natUniv (Pair_natUniv (Base_natUniv 0) (Base_natUniv 13))
(Base_natUniv 21)) (Base_natUniv 31) :: (Pair_natUniv (Pair_natUniv (Pair_natUniv
(Base_natUniv 0) (Base_natUniv 13)) (Base_natUniv 22)) (Base_natUniv 31) ::
(Pair_natUniv (Pair_natUniv (Pair_natUniv (Base_natUniv 0) (Base_natUniv 12))
(Base_natUniv 21)) (Base_natUniv 31) :: (Pair_natUniv (Pair_natUniv (Pair_natUniv
(Base_natUniv 0) (Base_natUniv 12)) (Base_natUniv 22)) (Base_natUniv 31) ::
(Pair_natUniv (Pair_natUniv (Pair_natUniv (Base_natUniv 0) (Base_natUniv 11))
(Base_natUniv 21)) (Base_natUniv 31) :: (Pair_natUniv (Pair_natUniv (Pair_natUniv
(Base_natUniv 0) (Base_natUniv 11)) (Base_natUniv 22)) (Base_natUniv 31) :: □)))))
```

o The output limit cone's side arrows:

```
compute arr_category_Arr ((Eval_cov_hom_transf ((sigma_Snd
(construct_inductively_limit_instance_liset _ example_graph_isf example_diagram))₁
)) ∘a' ( (@weightprof_Arr_arr _ _ _ (category_Obj_obj One) (graph_Obj_obj (Some
(Some None))) One)) );

// λ x, natUniv_snd (natUniv_fst (natUniv_fst x))
```

o The output limit cone's universality operation:

```
compute arr_category_Arr (((((sigma_Snd (construct_inductively_limit_instance_liset
_ example_graph_isf example_diagram))₂ ) _ _ _ example_cone) ∘>'_ ( _ ) ) ∘a'
(Id_cov_arr (category_Obj_obj One))) liset_terminal_natUniv;

// Pair_natUniv (Pair_natUniv (Pair_natUniv (Base_natUniv 0) (Base_natUniv 12))
(Base_natUniv 22)) (Base_natUniv 31)
```

• Besides, cut-elimination in the double category of fibred profunctors have immediate executable/computational applications to graphs transformations understood as categorial rewriting, where the objects are graphs (or sheaves in a topos), the vertical monomorphisms are pattern-matching subterms inside contexts and the horizontal morphisms are congruent/contextual rewriting steps…

References.

[1] Dosen-Petric: Cut Elimination in Categories 1999;

[2] Proof-Theoretical Coherence 2004;

[3] Proof-Net Categories 2005;

[4] Coherence in Linear Predicate Logic 2007;

[5] Coherence for closed categories with biproducts 2022;

[6] Cut-elimination in the double category of fibred profunctors with inner cut-eliminated adjunctions:

https://github.com/1337777/cartier/blob/master/cartierSolution13.lp

[7] Pierre Cartier