# Review of Kosta Dosen's categorial programming vs Richard Garner's 2-dimensional types: Cut-elimination in the double category of fibred profunctors with J-rule-eliminated adjunctions ("directed type theory").

**Short**: There is now sufficient evidence (ref [6], [7]) that Kosta Dosen's ideas and techniques (ref [1], [2], [3], [4], [5]) could be implemented for proof-assistants, sheaves and applications; in particular cut-elimination, rewriting and confluence for various enriched, internal, fibred or double categories with adjunctions, monads, negation, quantifiers or additive biproducts; quantitative/quantum linear algebra semantics; presheaf/profunctor semantics; inductive-sheafification and sheaf semantics; sheaf cohomology and duality... It was difficult to discover the correct grammatical-formulation out of a dozen semantically-meaningful ones, but this ongoing implementation (ref [6]) should work in the next months because it can already be expressed (computationally) that right adjoint preserves weighted limits and that covering sieves are merely fibred profunctors-types; now some of the remaining approximations in this text are speculations to be confirmed by actual labor. Outline:

**Introduction to Dosen's techniques.** The problem of "formulations of adjunction", the problem of "unit objects" and also the problem of "contextual composition/cut" can be understood as the same problem. The question arises when one attempts to write precisely the counit/eval transformation ϵ_X : catA(F G X, X) where F : catB → catA is the left-adjoint. One could instead write ϵ_1 : catA[F ,1](G , 1) where the profunctor object/datatype catA[F ,1] is used in lieu of the unit hom-profunctor catA; in other words: F is now some implicit context, and the contextual composition/cut (in contravariant action) of some g : catB[1, G] (Z, Y) in the unit profunctor against ϵ_1 should produce an element of the same datatype: ϵ_1∘g : catA[F,1](Z,Y)... Ultimately Dosen-Petric (ref [5]) would be extended in this setting where some dagger compact closed double category, of left-adjoint profunctors across Cauchy-complete categories, is both inner and outer dagger compact closed where the dagger operation on profunctors (as 1-cells) coincides with the negation operation on profunctors (as 0-cells), optionally with sheaf semantics and cohomology.

The initial key insight of Dosen-Petric, about the cut-elimination formulation in the domain-specific language of categorial adjunctions, can be understood as a problem of "unit profunctor" in the double category of profunctors and the (inner) cut elimination lemma becomes synonymous with elimination of the "J-rule"; and recall that such equality/path-induction J-rule would remain stuck in (non-domain-specific) directed (homotopy) type theory. Now the many "formulations of adjunctions" are for different purposes. Indeed the outer framework (closed monoidal category, with tensor bifunctor ⊗ with right adjoint implication connective ⇒) hosting such inner domain-specific language (unit-counit formulation of adjunctions) could itself be in another new formulation of adjunctions (lambda/eval bijection of hom-sets) where the implication bifunctor (with also contravariant argument _⇒_ ) is accumulated during computation via dinaturality (in contrast to the traditional Kelly-MacLane proof).

For reference (§ 4.1.5 in [1]), Kosta Dosen says that an adjunction with left adjoint functor $F: catB \to catA$, right adjoint functor $G: catA \to catB$, counit transformation $\phi_A: F\ G\ A \to A$, and unit transformation $\gamma_B: B \to G\ F\ B$ 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 naturality equations can be formulated generally for any transformation):

$$f_2 \circ (f_1)\text{“}1 \circ \phi \circ F\text{”} = (f_2 \circ f_1)\text{“}1 \circ \phi \circ F\text{”}$$

$$(f_1)\text{“}1 \circ \phi \circ F\text{”} \circ Gf_2 = (f_1 \circ f_2)\text{“}1 \circ \phi \circ F\text{”}$$

$$\text{“}1 \circ \phi \circ F\text{”}(g_1)\text{“} \circ F\text{”} \circ g_2 = \text{“}1 \circ \phi \circ F\text{”}(g_1 \circ g_2)$$

$$f_2 \circ \text{“}1 \circ \phi \circ F\text{”}(g_1) = \text{“}1 \circ \phi \circ F\text{”}(Gf_2 \circ g_1)$$

together with conversion (or rewrite) rules:

$$\text{“}1 \circ \phi \circ F\text{”}((f)\text{“}G \circ \gamma \circ 1\text{”}) = f$$

$$\text{“}1 \circ \phi \circ F\text{”}(\text{“}G \circ \gamma \circ 1\text{”}g) = Fg$$

where indeed the functions on arrows $F\ -$ and $G\ -$ of those functors are not primitive but are themselves the “consequential transformation” formulations “$1\ \circ\ F\ -$” of the identity arrow…

**New functorial lambda calculus.** Now such Dosen-style technique may be specialized to the instance of closed monoidal categories where the conjunction bifunctor _⊗_ has right adjoint implication bifunctor _⇒_ via lambda/eval bijection of hom-sets. Then dinaturality is used to accumulate the argument-component of the eval operation instead on its function-component:

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

$$= \text{“eval}_{A,O} \circ A \otimes \big((x \Rightarrow O) \circ (g \circ f)\big)\text{”}, \qquad x: A \to B$$

where this evaluation rule uses implication bifunctors skewed by reindexing (domain is left Kan extended):

```
| Eval_cov_transf : Π [A B C X A' X' : Cat] [P : mod A B] [Q : mod C X] [O : mod A' X'] [K : func
B C] [F : func A A'] [L : func X X'],
 P ⊢ F ∘> (Imply_cov_mod (O <∘ L) Q) <∘ K →
(Tensor_cov_mod P (K ∘> Q)) ⊢ F ∘> O <∘ L
```

In fact, such antecedental/consequential transformations may be formulated systematically, similarly as the “J-rule” for equality/paths in (homotopy) type theory. *And most importantly, because the general “J-rule” hide some cuts, therefore cut-elimination signify that the general “J-rule” must also be eliminated/admissible/computational.* The J-rule constructor Unit_con_transf has this form:

```
inductive cat : TYPE ≔
with func : Π (A B : cat), TYPE ≔
with mod : Π (A B : cat), TYPE ≔
with hom: Π [I A B : cat], func I A → mod A B → func I B → TYPE ≔
with transf: Π [A' B' A B: cat], mod A' B' → func A' A → mod A B → func B' B → TYPE ≔
| Unit_con_transf : Π [I A B J : cat] [F : func I A] [R : mod A B] [G : func I B], Π (M : func J
A),
hom F R G → transf (Unit_mod M F) Id_func (M ∘>> R) G ;
```

Then that right adjoint preserves weighted limits:

```
assert [B J J' A I : cat] (F : func J B) (W : mod J' J) (F_⇐_W : func J' B) (isl : isLimit F W F_⇐
_W) (R : func B A) (L : func A B) (isa : adj L R) (M : func I A),
((((M)_'∘> (Adj_cov_hom isa F)) ⇐2 (Id_transf W))
   ''∘ (limit_intro_transf isl (M ∘> L)))
        ''∘ ((Adj_con_hom isa M) ∘>'_(F_⇐_W)) ⊢
: transf ((Unit_mod M (R <∘ F)) ⇐ W) Id_func (Unit_mod M (R <∘ F_⇐_W)) Id_func ;
```

**New grammatical topology.** 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 that 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 possibly-incompatible functions to be glued, 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:

$$
\begin{aligned}
&F(U0) \coloneqq \text{sum over the slice U0 U01} = \mathbb{Z} \oplus \mathbb{Z};\\
&F(U1) \coloneqq \ \mathbb{Z} \oplus \mathbb{Z}; F(U01) \coloneqq \mathbb{Z}\\
&F(U) = \text{kan extension} = \ \mathbb{Z} \oplus \mathbb{Z} \oplus \mathbb{Z};
\end{aligned}
$$

$$
\begin{aligned}
&\text{gluing}: F(U0) \oplus F(U1) \oplus F(U01) \to \ F(U)\\
&\big((f0, f01), (g1, g01), (h01)\big) \mapsto (f0, g1, f01 + g01 - h01)
\end{aligned}
$$

where the signed sum generalizes to higher degrees because the Euler characteristic is 1 (or inclusion–exclusion principle).

Now dependent types are fibrations of (the double-category cograph of) (enriched) profunctors, and any covering (co)sieve is implemented simply as a fibration of profunctors, which covers the base (unit-hom) profunctor at some given-fixed covariant variable (the contravariant piece is understood as some codomain fibration of categories). 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) of the object $A$, and where $j_A(\mathcal{U})(f) \coloneqq$ "$f^*\mathcal{U} \in J(X)$" 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 $\forall f : some\ cover. j_A(\mathcal{U})(f)$), iff $j_A(j_A(\mathcal{U}))$ becomes

```
| Total_covering : Π [X Y I : cat] [A : catd X] [R : mod X Y] [B : catd Y] (RR : modd A R B) [J :
catd I] [G : func I Y] [H : funcd J G B], (Π [F : func I X] (r : hom F R G), coveringhd
(Fibre_hom_modhd RR r)  (Univ_funcd H)) → covering RR H
```

But this failure of sieves being monomorphisms is the motivation for this new "grammatical sheaf cohomology"…

**New fibred dependent types**. The implementation of covering sieves can be understood as a motivation for the implementation of dependent types. Another motivation for dependent types is the implementation of the comma/arrow category of a base profunctor as a fibration over this base. In the adjunctions $\Sigma_f \dashv f^* \dashv \Pi_f$ then the sum $\Sigma_f$ and product $\Pi_f$ are now injective symbols (constructors/formers for types) while the pullback $f^*$ is now a recursively defined symbol, therefore the grammar now allows morphisms $h: A \to B$ fibred over a span $\Gamma \leftarrow \Theta \to \Delta$ of delayed/pending explicit substitutions on the left and right, which denotes some morphism $f^*A \to g^*B$ , and which is also in correspondence with some morphism $\Sigma_g f^*A \to B$ or some morphism $A \to \Pi_f g^*B$ via introduction, elimination and computation rules.

**New executable applications.** Fibrations of (the double-category cograph) of (enriched) profunctors and their cut-elimination 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. The ability to express covering sieves as fibrations of profunctors would enable the description of algebraic geometry's schemes in their formulation as locally affine ringed sites (structured topos), instead of via their formulation as underlying topological space.

**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 J-rule-eliminated adjunctions: https://github.com/1337777/cartier/blob/master/cartierSolution12.lp
[7] Pierre Cartier

**Appendix: Review of Richard Garner "Two-dimensional models of type theory" vs Kosta Dosen.**

In summary, Garner describes a 2-truncated variant of the Identity types of Martin-Lof dependent type theory syntax, and equips it with a sound and complete semantics valued in 2-categories; which, as well as objects representing types, and morphisms $f : X \to Y$ representing terms, has 2-cells isomorphisms $\alpha: f \Rightarrow g$ representing witnesses for the propositional equality of terms f and g, so that each object/type in this 2-category can be understood as a groupoid.

*Note that the phrasing "complete semantics" (in the sense of possessing an "internal language" in the same style as its original syntax) is already a hint that the semantics, if formulated carefully, could be used directly as its own new syntax for computation.*

**Introduction to locally cartesian closed categories as models for types.**

It was usually proposed that the correct categorical models for extensional Martin-Lof type theory should be locally cartesian closed categories: these being categories C with finite limits in which each of the functors $f^* : C/X \to C/Y$ induced by pulling back along a morphism $f : Y \to X$ has a right adjoint. The idea is to think of each object X of a locally cartesian closed category C as a closed type, each morphism as a term, and each object of the slice category C/X as a type dependent upon X. Now substitution of terms in types may be interpreted by pullback between the slices of C; dependent sum and product types by left and right adjoints to pullback; and the equality type on X by the diagonal morphism $\Delta: X \to X \times X$ in $C/X \times X$. This picture is not wholly accurate, since in the syntax, the operation which to each morphism of types $f : Y \to X$ assigns the corresponding substitution operation $\text{Type}(X) \to \text{Type}(Y)$ is strictly functorial in f; whilst in the semantics, the corresponding assignation $(f : Y \to X) \to (f^* : C/X \to C/Y)$ is rarely so. Thus this notion of model is not *sound* for the syntax, and one is forced to refine it slightly: essentially by equipping the locally cartesian closed category with a split fibration $T \to C$ equivalent to its codomain fibration $C^2 \to C$. Types over X are now interpreted as objects of the fibre category T(X); and since $T \to C$ is a split fibration, the interpretation is sound for substitution.

*Note that any attempt to implement on the computer datatypes this locally cartesian closed category C would necessarily lead to implement such a split fibration $T \to C$ instead of the equivalent codomain fibration $C^2 \to C$.*

Garner makes use of the rules for identity types of dependent type theory in order to construct a 2-category C of contexts; a two-dimensional fibration T → C of types over contexts; and a comprehension 2-functor E: T → $C^2$, sending each type-in-context Γ ⊢ A type to the corresponding *dependent projection* map (Γ, x : A) → Γ. Moreover, each such dependent projection in the 2-categorical model carries the structure of a *normal isofibration*, which can be seen as the semantic correlate of the *Leibniz rule* in dependent type theory.

*Note that Garner's restraint on isofibrations, which allows transport of data in the total space along only isomorphisms/paths in the base space, is a consequence that elements of identity types are interpreted as isomomorphisms 2-cells (isomorphisms in the internal groupoid). Then the natural generalization to hom-types of the semantics would correspond to (co-)cartesian fibrations which allows transport of data along any morphism in the base space.*

**Grammar entries for the judgments forms.**

Garner now proceeds to summarize the notations for Martin-Lof type theory with type-formers for dependent sums, dependent products, identity types and the unit type. The calculus has four basic forms of judgement: A type ("A is a type"); a : A ("a is an element of the type A"); A = B type ("A and B are definitionally equal/convertible types"); and a = b : A ("a and b are definitionally equal/convertible elements of the type A"). These judgements may be made either absolutely, or relative to a context Γ of assumptions, in which case one writes them as

$$\Gamma \vdash A \text{ type}, \qquad \Gamma \vdash a : A, \qquad \Gamma \vdash A = B \text{ type} \qquad \text{and} \qquad \Gamma \vdash a = b : A$$

respectively. Here, a *context* is a list $\Gamma = (x_1 : A_1, x_2 : A_2, \dots, x_n : A_n)$, wherein each $A_i$ is a type relative to the context $(x_1 : A_1, \dots, x_{i-1} : A_{i-1})$.

Note that although the new functorial lambda calculus makes uses of grammatical/syntactic combinators instead of variables, it is still conceptually useful to think using this variables-notation. For example and approximately, a functor F : I → C (1-cell in the two/double category of profunctors), also written F : func I C, which picks an (undetermined) object in the category C may be thought of as the judgment

$$\Gamma \mid x : I \vdash F\,x : C \qquad := \qquad F : \text{func } I\ C \qquad \text{for } I, C : \text{cat},$$

in the outer context Γ and inner context (x : I). Also a hom α: F ⇒ G : I → C (vertical 2-cell in the double category), also written α: hom F C G, which picks an (undetermined) arrow in the category (or profunctor) C may be thought of as the judgment

$$\Gamma \mid x : I \vdash \alpha\,x : \text{hom } (F\,x)\ C\ (G\,x) \qquad := \qquad \alpha : \text{hom } F\ C\ G.$$

And a transformation α: P ⇒ Q [F, G] : I ⇸ J (horizontal 2-cell in the double category), also written α: transf P F Q G, which picks an (undetermined) arrow in the profunctor (or category) Q may be thought of as the judgment

$$\Gamma \mid x : I,\ p : P\,x\,y,\ y : J \vdash \alpha\,x\,y\,p : Q\ (F\,x)\ (G\,y) \quad := \qquad \alpha : \text{transf } P\ F\ Q\ G,$$

where it is understood that the implementation also has the context-dependent variants `catd, funcd, homd, transfd` for all such grammatical entries `cat, func, hom, transf`. And the ultimate goal is to be able to automatically decide the definitional-equality/convertibility of two data/elements Γ ⊢ a = b : A where a, b could be both objects or both arrows inside the category/type A. Garner then specifies the collection of inference rules over these forms of judgement. These rules include the *structural rules*, which deal with congruences of definitional-equalities, weakening, contraction, exchange and substitution. Then the *logical rules* are included, which deal with identity, unit, sum and product types:

**Rules of inference for the Identity types and the Unit type.**

For the Identity types

$$\frac{A \ \text{type} \qquad a, b : A}{\mathrm{Id}_A(a, b) \ \text{type}} \ \texttt{Id-form} \qquad \frac{A \ \text{type} \qquad a : A}{r(a) : \mathrm{Id}_A(a, a)} \ \text{Id-intro};$$

$$\frac{x, y : A,\ p : \mathrm{Id}_A(x, y) \vdash C(x, y, p) \ \text{type} \qquad x : A \vdash d(x) : C(x, x, r(x))}{x, y : A,\ p : \mathrm{Id}_A(x, y) \vdash J_d(x, y, p) : C(x, y, p)} \ \text{Id-elim};$$

$$\frac{x, y : A,\ p : \mathrm{Id}_A(x, y) \vdash C(x, y, p) \ \text{type} \qquad x : A \vdash d(x) : C(x, x, r(x))}{x : A \vdash J_d(x, x, r(x)) = d(x) : C(x, x, r(x))} \ \text{Id-comp}.$$

From which the *Leibniz rule* (transport of data along identifications/paths/arrows) is derivable, given x : A ⊢ B(x) type,

$$\frac{a1, a2 : A \qquad p : \mathrm{Id}(a1, a2) \qquad b2 : B(a2)}{p^*(b2) : B(a1)} \ \texttt{Id-subst} \qquad \frac{a : A \qquad b : B(a)}{r(a)^*(b) = b : B(a)} \ \texttt{Id-subst-comp}$$

And these transport rules will be reflected in the semantics by the fact that the dependent projection (A, x : B) → A for the judgment x : A ⊢ B(x) type carries the structure of a normal isofibration, or cartesian fibration (for the categorial non-groupoidal case).

The implementation code for the Leibniz fibrational-transport rule is approximately:

```
injective symbol CartFibd_homd :  Π [I X X' : cat] (xx : func X' X) (xi : func X I) [yi :
func X I]  [II : catd I] (r : hom (xx ∘> xi) ((Unit_mod0 I) <<∘ yi) xx) (G : funcd
(Triv_catd X) yi II),
homd r ((CartFibd_funcd xx xi r G)) ((Unit_modd0 II) d<<∘ G) (Triv_funcd xx);
```

Note that this type-theory syntax is very powerful and too general as it readily allows to express an infinite hierarchy of iterated identity types

$$a,b : A;\ p,q : \mathrm{Id}(a,b);\ \alpha : \mathrm{Id}_{\mathrm{Id}}(p, q);\ \ldots$$

so that the natural models would be infinity-categories or infinity-groupoids. Now Garner proceeds to put a limitation and truncate at *two-dimensional type theory* by specifying a further *rule* (without changing the basic syntax of term-expressions) that, in such situation, then the following definitional-equalities/conversion judgements hold:

$$\frac{a,b : A;\ p,q : \mathrm{Id}(a,b);\ \alpha : \mathrm{Id}_{\mathrm{Id}}(p, q)}{p = q : \mathrm{Id}(a,b) \ \text{and then} \ \alpha = r(p) : \mathrm{Id}(p,q)}$$

which signify that all identity types are *discrete*. But this trick of hacking a general theory for a new purpose is a symptom that *such repurposing is bad*, and instead it would be better to devise a new domain-specific language, syntax and implementation for the specific purpose (of a *categorial programming language for 1-category objects internally of a double-category of profunctors*).

Note that as a consequence, in the new Dosen's categorial programming, there is a grammatical entry (computer datatype) to store arrows/identifications/paths p : hom F C G, which is in addition to the grammatical entry to store objects/points F : func I C, of the category C. Here, discreteness/truncation means that whatever (constructive) properties of those arrows/identifications would be expressible/provable in a higher-categorial setting, is readily added as a (conversion) *rewrite-rule* in the metalogical framework (LambdaPi or Coq). Indeed, in the implementation of this new calculus, for example any wanted adjunction's triangular and naturality (higher-)equations (axioms) between arrows are added directly as rewrite-rule.

Moreover, the grammatical entry hom $F\ C\ G$ will be reflected into this calculus in the form of the *comma/arrow category* construction (comma $R$) : cat for a profunctor (category) $R : C \nrightarrow \mathrm{D}$ (which approximately means $\Sigma(c\colon C)(d\colon D), R(c,d)$), but this latter logical rule of arrow/identity *type formation* is less fundamental than the structural/intrinsic rule given by the arrow/identity grammatical-entry. Note that *Garner would not have structural rules for arrows but only for identities* (called definitional-equality in the situation of structural rules), therefore it would not be sufficient, without a substantial structural change, to simply declare that the Identity $Id$ type formation now denotes directed arrows inside 1-categories instead of groupoids.

Next the rules for the Unit type, which turns out to be interrelated with the Identity type in the new Dosen's calculus:

$$\frac{}{1 \text{ type}}\ \text{1-form;} \qquad \frac{}{\star : 1}\ \text{1-intro;}$$

$$\frac{z : 1 \vdash C(z)\ \text{type} \qquad d : C(\star)}{z : 1 \vdash U_d(z) : C(z)}\ \text{1-elim;}$$

$$\frac{z : 1 \vdash C(z)\ \text{type} \qquad d : C(\star)}{U_d(\star) = d : C(\star)}\ \text{1-comp.}$$

These rules allow to cast data from the empty context $\vdash d : C(*)$ to the dummy-unit context $z : 1 \vdash U_d(z) : C(z)$ and suggests that judgements with empty context should belong to a distinct grammatical-entry of the implementation, different than judgments with non-empty context. Indeed, in the new calculus the grammatical-entry α: hom F C G ( := x : I ⊢ α x : hom (F x) C (G x) ) is distinct from the grammatical-entry α: transf P F Q G ( := x : I, p : P x y , y : J ⊢ α x y p : Q (F x) (G y) ) which allows for a non-empty context p: P x y of arrow-variables; and this structural/intrinsic rule distinction is reflected into the calculus via the Unit type former. And the Unit type is not dummy anymore, but carries more data: it is the profunctor unit-hom Unit I : I ⇸ I at any category I (approximately, (Unit I)(x,y) := $\mathrm{hom}_I$(x,y)), which is the unit for the monoidal (coend) structure on profunctors; and the elimination rule becomes the covariant/contravariant composition/action/Yoneda (inner-cut) inside the 1-categories, and this inner-cut should also be eliminated/admissible.

In fact, now that the Unit type carries more data, *the general J-rule (Id-elim) is expressed/derivable by blending the special J-rule (Leibniz rule of fibrational transport) with the (non-dummy, linear variant of the)* *empty-inner-context-unit rule (which is the composition/action operation inside each categorial/profunctorial type, in Yoneda formulation).*

The implementation code for the (non-dependent) empty-context-unit rule is approximately:

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

And the dependent variant, empty-inner-context-unit rule, of this empty-context-unit rule is approximately:

```
injective 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] [FF : funcd (Triv_catd I) F A] [RR : modd A R B] [GG : funcd
(Triv_catd I) G B], Π [J: cat] [M : func J X] [JJ : catd J] (MM : funcd JJ M A),
homd r FF RR GG → transfd ((M)_'◦> r) (Unit_modd MM FF) Id_funcd (MM ◦>>d RR) GG;
```

**Rules of inference for the Sum types and Product types.**

Garner proceeds by reminding the syntactic category and the structure induced on it by the non-logical rules of type theory and then by the logical rules. The syntactic category defined is in fact a *2-category*, whose objects will be (vectors of) types (internal categories); whose morphisms will be (vectors) of terms (objects of those internal categories) between those types; and whose (vertical) 2-cells will be (vectors of) identity proofs (arrows of those internal categories) between these terms. The various forms of 2-cell composition (inner-cut) will be obtained using the identity elimination rules (J-rule), or more precisely using the (non-dummy) empty-inner-context-unit elimination rule in the new calculus, as it was noticed above that the general arrow/identity elimination has become a blend of the special Leibniz fibrational-transport rule with the empty-inner-context-unit elimination rule. In the new calculus this 2-category becomes a *double category* of (internal) categories, functors and profunctors, where the horizontal 2-cells are now transformations between profunctors.

Essentially with only the goal of fixing notations for conceptual thinking, here is an outline of the syntactic category. Garner says that it is a (full) *comprehension category*, where its *category of contexts* $C_S$ have as objects, contexts Γ, Δ, ..., and as morphisms Γ → Δ, judgements x : Γ ⊢ f(x) : Δ, and *later* whose 2-cells α: f ⇒ g: Γ → Δ are judgements x : Γ ⊢ α(x) : $Id_Δ$(fx,gx)… where this approximation of viewing contexts as types can be made precise but is not a problem in the semantics and its new calculus. For each context Γ, one now defines the category $T_S$ (Γ) of *types-in-context*-Γ, whose objects A are judgements x : Γ ⊢ A(x) type and whose morphisms A → B are judgements x : Γ, y : A(x) ⊢ f(x,y) : B(x), and *later* whose 2-cells α: f ⇒ g: A → B are judgements x : A ⊢ α(x) : $Id_B$(fx,gx) (or x : A ⊢ α(x) : $Hom_B$(fx,gx) for arrows, where this new notation abbreviates the implementation's grammatical-entry α : hom f B g, as explained above).

Each morphism f : Γ → Δ of $C_S$ induces a functor $T_S$(f): $T_S$ (Δ) → $T_S$ (Γ) which sends a type A in context Δ to the type f*A in context Γ given by x : Γ ⊢ A(f(x)) type. The assignation f → T(f) is itself functorial in f, and so one obtains an indexed category $T_S$ (–): $C_S^{op}$ → Cat; which via the total-fibration construction, one may equally well view as a split fibration p: $T_S$ → $C_S$. One then refers to this as the *fibration of types over contexts*.

Explicitly, objects of $T_S$ are pairs (Γ,A) of a context and a type in that context; whilst morphisms (Γ,A) → (Δ,B) are pairs (f,g) of a context morphism f : Γ → Δ together with a judgement x : Γ, y : A(x) ⊢ g(x,y) : B(f(x)), and *later* 2-cells (α,β): (f,g) ⇒ (f′,g′): (Γ,A) → (Δ,B) are given by pairs of judgements, as in the following diagram:

$$x : \Gamma \vdash \alpha(x) : \mathrm{Id}_\Delta(fx, f'x)$$

$$x : \Gamma, y : A(x) \vdash \gamma(x,y) : \mathrm{Id}_{B(fx)}\,(g(x,y), \alpha(x)^*(g'(x,y))$$

(1)
$$\begin{tikzcd}
\Gamma.A \arrow[r, bend left, "f.g"] \arrow[r, bend right, "f'.g'"'] \arrow[r, phantom, "\Downarrow\gamma"] \arrow[d, "\pi_A"'] & \Delta.B \arrow[d, "\pi_B"] \\
\Gamma \arrow[r, bend left, "f"] \arrow[r, bend right, "f'"'] \arrow[r, phantom, "\Downarrow\alpha"] & \Delta
\end{tikzcd}$$

Where for each object (Γ,A) of $T_S$ one has the extended context $x : \Gamma, y : A(x)$ , which one denotes by Γ.A; and one also has the judgement $x : \Gamma, y : A(x) \vdash x : \Gamma$, corresponding to a context morphism $\pi_A : \Gamma.A \to \Gamma$ which one calls the *dependent projection* associated to A. In fact, the assignation $(\Gamma, A) \mapsto \pi_A$ provides the action on objects of a functor $E: T_S \to C^2$ (where 2 denotes the arrow category $0 \to 1$), whose fully faithful action on maps sends the morphism $(f,g): (\Gamma,A) \to (\Delta,B)$ of $T_S$ to the morphism

(2)
$$\begin{tikzcd}
\Gamma.A \arrow[r, "f.g"] \arrow[d, "\pi_A"'] & \Delta.B \arrow[d, "\pi_B"] \\
\Gamma \arrow[r, "f"'] & \Delta
\end{tikzcd}$$

of $C^2$, where f.g denotes the judgement $x : \Gamma, y : A \vdash (f(x),g(x,y)) : \Delta.B$. The chosen cartesian lifting of a morphism $f : \Gamma \to \Delta$ at an object (Δ,B) is given by $(f,\iota): (\Gamma,f^*B) \to (\Delta,B)$, where ι denotes the judgement $x : \Gamma$, $y : B(fx) \vdash y : B(fx)$, and then the corresponding square (2) is a pullback square.

Every dependent projection $\pi_B : \Delta.B \to \Delta$ in $C_S$ may be equipped with the structure of a normal isofibration, or *later* one may require the structure of a (co-)cartesian fibration, in the sense that given an (invertible) 2-cell

(3)
$$\begin{tikzcd}
\Gamma \arrow[rr, "g"] \arrow[dr, "f"'] & \overset{\alpha}{\Rightarrow} & \Delta.B \arrow[dl, "\pi_B"] \\
& \Delta &
\end{tikzcd}$$

there is given a choice of 1-cell $s_\alpha : W \to X$ and 2-cell $\sigma_\alpha : s_\alpha \Rightarrow g$ satisfying $p \circ s_\alpha = f$ and $p \circ \sigma = \alpha$; and these choices are natural in W, in the sense that given further $k: W' \to W$, one has $s_{\alpha k} = s_\alpha \circ k$ and $\sigma_{\alpha k} = \sigma_\alpha \circ k$. Furthermore, *normality* means that for any $g: W \to X$, one has $s_{\mathrm{id}_{pg}} = g$ and $\sigma_{\mathrm{id}_{pg}} = \mathrm{id}_g$. Indeed $s_\alpha : \Gamma \to \Delta.B$ is given by the pair of judgements

$$x : \Gamma \vdash f(x) : \Delta \qquad \text{and} \qquad x : \Gamma \vdash (\alpha x)^*\,(g_2 x) : B(fx),$$

and $\sigma_\alpha : s_\alpha \Rightarrow g$ is given by the pair of judgements

$$x : \Gamma \vdash \alpha(x) : \mathrm{Id}(fx, g_1 x)$$

$$x : \Gamma, y : A(x) \vdash r((\alpha x)^*(g_2 x)) : \mathrm{Id}((\alpha x)^*(g_2 x), (\alpha x)^*(g_2 x)).$$

At the end one has obtained a so-called *full split comprehension category (or 2-category or double category)* given by a category C, together with a split fibration $p: T \to C$ and a full and faithful functor $E: T \to C_2$ rendering commutative the triangle

$$\begin{tikzcd} T \arrow[r, "E"] \arrow[dr, "p"'] & C^2 \arrow[d, "cod"] \\ & C, \end{tikzcd}$$

and sending cartesian morphisms in T to pullback squares in $C^2$.

*And the upshot is that, approximately the same notations developed above hold in arbitrary comprehension categories* $(p : T \to C, E : T \to C^2)$… *What about the various assumptions by Garner that identifications/paths p : Id(x,y) are isomorphisms in groupoids instead of general arrows in categories?* Garner critically uses this assumption only twice. In Proposition 4.1.1 to show some *3-dimensional* (not 2-dimensional as believed) aspect of the universal property of the *comma/arrow object* (which is the reflection into the calculus of the structural/intrinsic identity). As noted above, in the new calculus it is not necessary to reflect, the structural/intrinsic implementation of hom arrows, into the calculus in the form of some comma/arrow category/type former; but if this is done then the limitation is that *only its core groupoid would be expressible*, that is it would be a *comma/arrow groupoid* former, or maybe its input category can be limited such that the comma/arrow category becomes a double category with companions/conjoins without 2-cells (vertically-discrete)… In Proposition 3.4.5, the lemma (21) defining θ, which shows that the fibrational-lift of a path/arrow commutes with pullback, should instead be a conversion rewrite rule (axiom) of the model/semantics instead of being an invertible identification/path.

Finally, the rules of inference for the dependent product types are:

$$\frac{A \text{ type} \qquad x : A \vdash B(x) \text{ type}}{\Pi x : A.\, B(x) \text{ type}} \qquad\qquad \frac{x : A \vdash f(x) : B(x)}{\lambda x.\, f(x) : \Pi x : A.\, B(x)}\ \Pi\text{-abs};$$

$$\frac{m : \Pi x : A.\, B(x)}{y : A \vdash m \cdot y : B(y)} \qquad\qquad \frac{x : A \vdash f(x) : B(x)}{y : A \vdash \lambda x.\, f(x) \cdot y = f(y) : B(y)}\ \Pi\text{-}\beta.$$

And the rules of inference for the dependent sum types are:

$$\frac{A \text{ type} \qquad x : A \vdash B(x) \text{ type}}{\Sigma x : A.\, B(x) \text{ type}} \qquad\qquad \frac{a : A \qquad b : B(a)}{\langle a, b\rangle : \Sigma x : A.\, B(x)}\ \Sigma\text{-intro};$$

$$\frac{z : \Sigma x : A.\, B(x) \vdash C(z) \text{ type} \qquad x : A,\ y : B(x) \vdash d(x, y) : C(hx, yi)}{z : \Sigma x : A.\, B(x) \vdash E_d(z) : C(z)}\ \Sigma\text{-elim};$$

$$\frac{z : \Sigma x : A.\, B(x) \vdash C(z) \text{ type} \qquad x : A,\ y : B(x) \vdash d(x, y) : C(hx, yi)}{x : A,\ y : B(x) \vdash E_d(hx, yi) = d(x, y) : C(hx, yi)}\ \Sigma\text{-comp}.$$

Despite the symmetric adjunctions $\Sigma_f \dashv f^* \dashv \Pi_f$ , those above type-theory rules for product and sum are not symmetric/parallel. After careful analysis, it is found that *in the new calculus they can be made symmetric again*.

Garner proceeds to say that one mismatch is that the (abstraction/lambda) rules for the product types happen in the empty context instead of the unit context (or instead of the context of another domain type). Indeed, a morphism f : Γ.A → Γ.A.B over Γ.A in the empty context would become a morphism f : $Γ.A.1_{Γ.A}$ → Γ.A.B over Γ.A in the unit-type context $1_{Γ.A}$, which would then expose the left-adjoint (weakening) $1_{Γ.A} = Δ_A 1_Γ$, so that it becomes a morphism f : $Δ_A 1_Γ$ → B of T(Γ.A) on which one may apply the adjunction universality of $Π_A(B)$ to obtain a morphism λ(f) : $1_Γ \to Π_A(B)$ of T(Γ).

Another mismatch is that the elimination rule for sum is *very-dependent* in the sense that the codomain/output type C is dependent on the sum-type. This is not a problem for the adjunction $\Sigma_f \dashv f^*$ which can be enhanced to a very-dependent variant so that $f$ is itself the adjunction unit (injection equivalence) $f \coloneqq \eta : B \to \Sigma_g B$ over some other $g$, in which case $\Sigma_f$ would take the trivial (identity) fibration to the trivial fibration. But in fact, the introduction rule (abstraction) for product can also be made very-dependent…

The upshot is that the structural/intrinsic grammar should allow morphisms $h: A \to B$ fibred over a span $\Gamma \leftarrow \Theta \to \Delta$ of delayed/pending explicit substitutions on the left and right, which denotes some morphism $f^*A \to g^*B$ , and which is also in correspondence with some morphism $\Sigma_g f^*A \to B$ or some morphism $A \to \Pi_f g^*B$ via introduction, elimination and computation rules. *Then all the new Kosta Dosen's fibrational calculus become fully categorial, symmetric and computational.*

$$\begin{tikzcd}
\Gamma.A \arrow[rr, "h"] \arrow[d, "\pi_A"'] & & \Gamma.B \arrow[d, "\pi_B"] \\
\Gamma & \Theta \arrow[l, "f"] \arrow[r, "g"'] & \Delta
\end{tikzcd}$$

The full details of the essentially-completed implementation, which is readable coded mathematics (not less readable than latex-formulas), can be copied-pasted into this document here from the other file at:

https://github.com/1337777/cartier/blob/master/cartierSolution12.lp