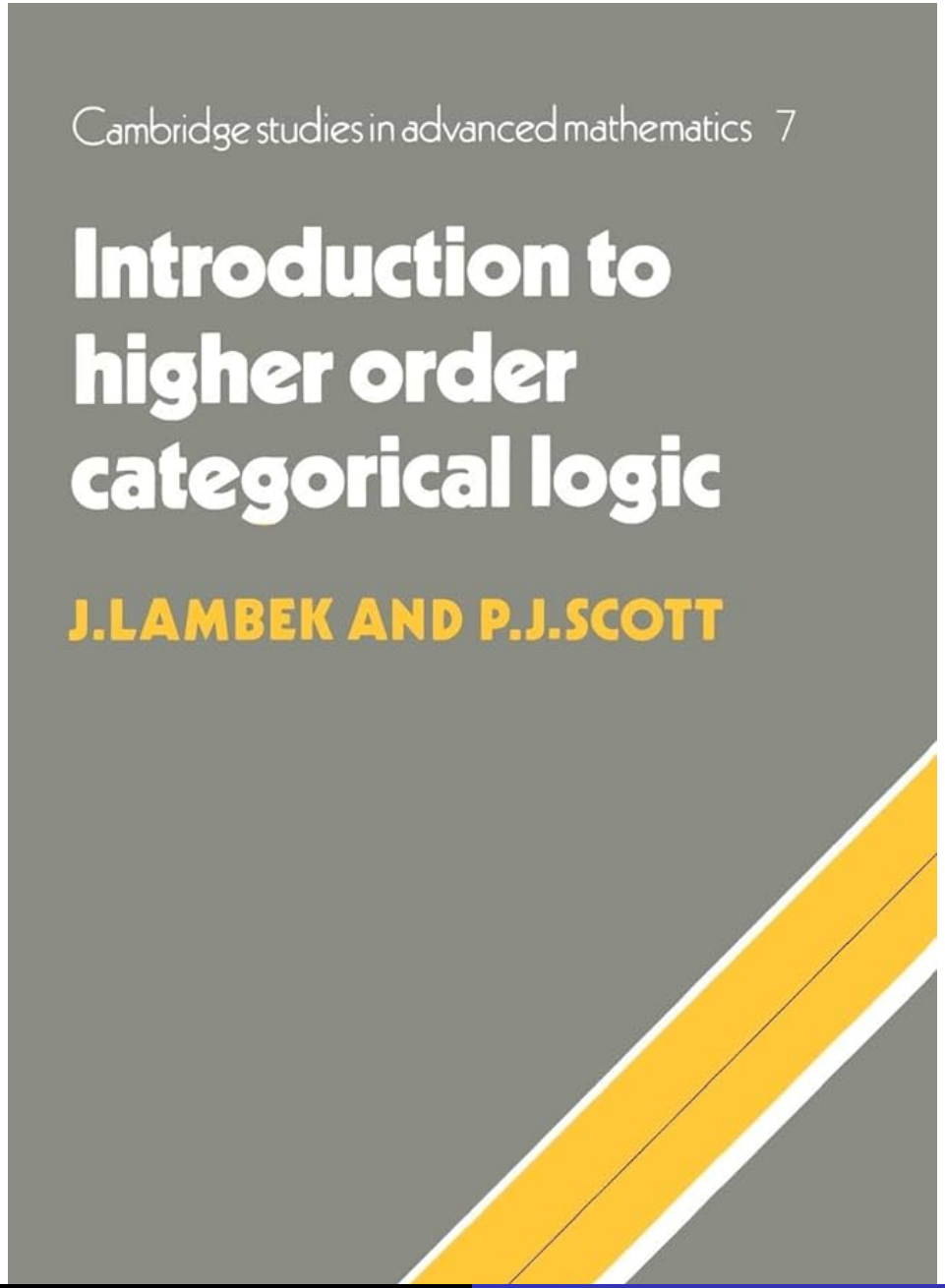
Proofs, Types and Hexagons
A Talk Dedicated To The Memory Of Phil Scott

Richard Blute
University of Ottawa

The book


Cambridge studies in advanced mathematics 7
Introduction to higher order categorical logic
J.LAMBEK AND P.J.SCOTT

Categorical Proof Theory I

- **Categorical proof theory** begins with the idea of forming a category whose objects are formulas (=types) in a given logic and whose arrows are (equivalence classes of) proofs (=programs).

Categorical Proof Theory I

- **Categorical proof theory** begins with the idea of forming a category whose objects are formulas (=types) in a given logic and whose arrows are (equivalence classes of) proofs (=programs).

- As a simple example, in *intuitionistic logic*, in the fragment corresponding to $\wedge$ and $\Rightarrow$, conjunction takes on the form of a categorical product. Logical implication gives the category closed structure.

Categorical Proof Theory I

- **Categorical proof theory** begins with the idea of forming a category whose objects are formulas (=types) in a given logic and whose arrows are (equivalence classes of) proofs (=programs).

- As a simple example, in *intuitionistic logic*, in the fragment corresponding to $\wedge$ and $\Rightarrow$, conjunction takes on the form of a categorical product. Logical implication gives the category closed structure.

- This construction gives the free cartesian closed category generated by the atomic formulas, and terms in simply-typed λ-calculus represent equivalence classes of proofs.

- The counit of the adjunction

$$Hom(A \wedge B, C) \cong Hom(B, A \Rightarrow C)$$

is a map $\eta\colon A \wedge (A \Rightarrow C) \to C$. This is the familiar *modus ponens* rule.

Categorical Proof Theory II

- The counit of the adjunction

$$Hom(A \wedge B, C) \cong Hom(B, A \Rightarrow C)$$

is a map $\eta\colon A \wedge (A \Rightarrow C) \to C$. This is the familiar *modus ponens* rule.

- Similar remarks hold in various fragments of Girard's *linear logic*. Proof nets play the role of λ-calculus terms. (Give or take problems with the units for the monoidal structures.)

Polymorphic Types

- Programs which run on *polymorphic types*, especially *parametric polymorphic types*, are programs which run on a family of types and have essentially the "same" behavior on each of those types.

Polymorphic Types

- Programs which run on *polymorphic types*, especially *parametric polymorphic types*, are programs which run on a family of types and have essentially the "same" behavior on each of those types.

- A very simple example which will illustrate the issues we are exploring is the *polymorphic identity*:

$$id : \forall \alpha . \alpha \Rightarrow \alpha$$

Polymorphic Types

- Programs which run on *polymorphic types*, especially *parametric polymorphic types*, are programs which run on a family of types and have essentially the "same" behavior on each of those types.

- A very simple example which will illustrate the issues we are exploring is the *polymorphic identity*:

$$id : \forall \alpha . \alpha \Rightarrow \alpha$$

- The idea behind *Functorial Polymorphism* (Bainbridge, Freyd, Scedrov, Scott) is that these variable types should be represented by functors and programs would be natural transformations.

Functorial Polymorphism

- **BUT**

Functorial Polymorphism

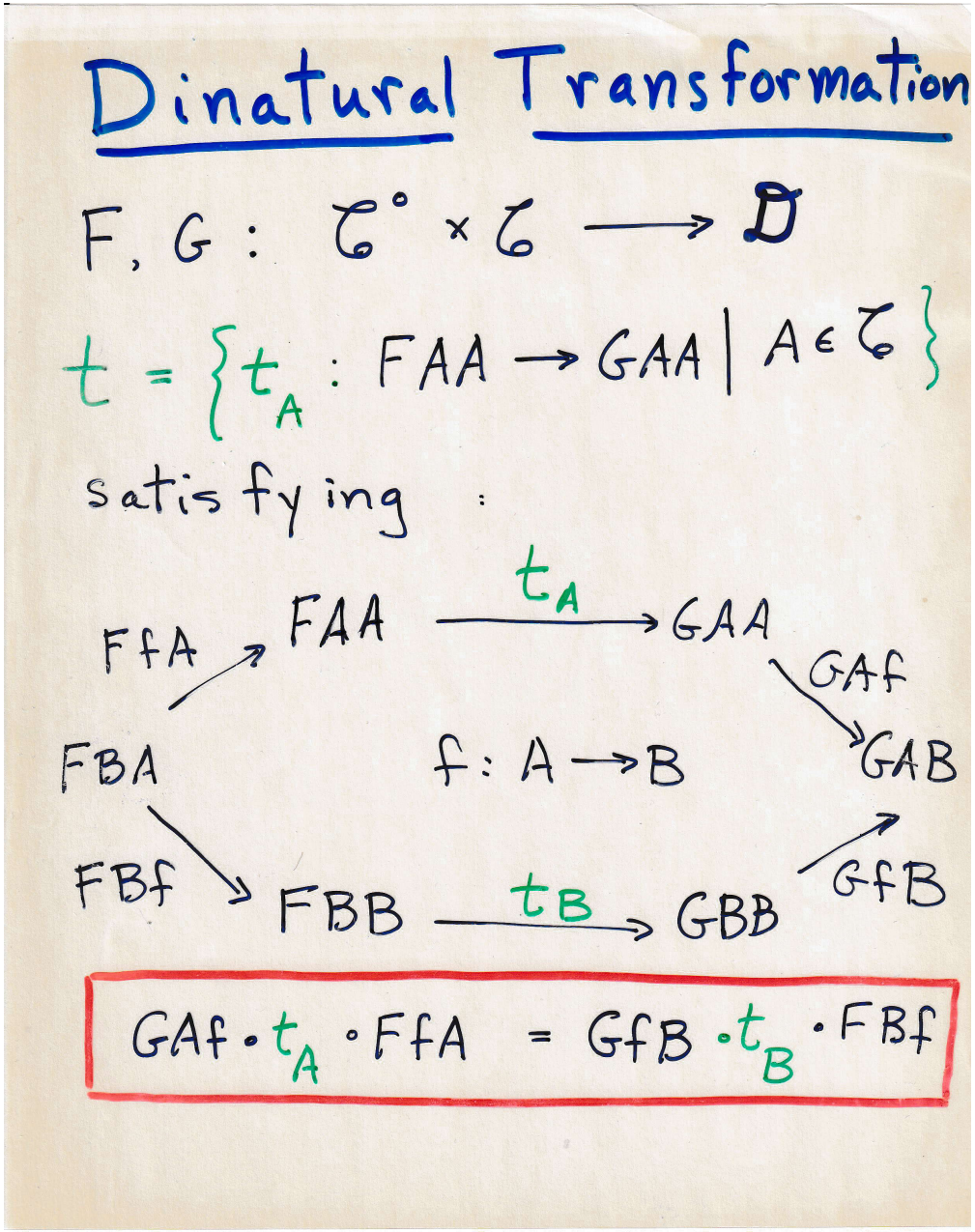
- **BUT**
- This doesn't work.

Functorial Polymorphism

- **BUT**
- This doesn't work.
- The most interesting polymorphic types, such as $\alpha \Rightarrow \alpha$ $\alpha \wedge (\alpha \Rightarrow \beta)$ can't be considered as functors, since at least one of the variables occurs simultaneously in a covariant and contravariant position.

- **BUT**
- This doesn't work.
- The most interesting polymorphic types, such as $\alpha \Rightarrow \alpha$ $\alpha \wedge (\alpha \Rightarrow \beta)$ can't be considered as functors, since at least one of the variables occurs simultaneously in a covariant and contravariant position.
- The solution is to view types such as $\alpha \Rightarrow \alpha$ as a specific instantiation of the more general multivariant functor $\alpha \Rightarrow \beta \colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, and then replace natural transformations with *dinatural transformations*, The "di" is short for diagonal.

Dinatural transformations

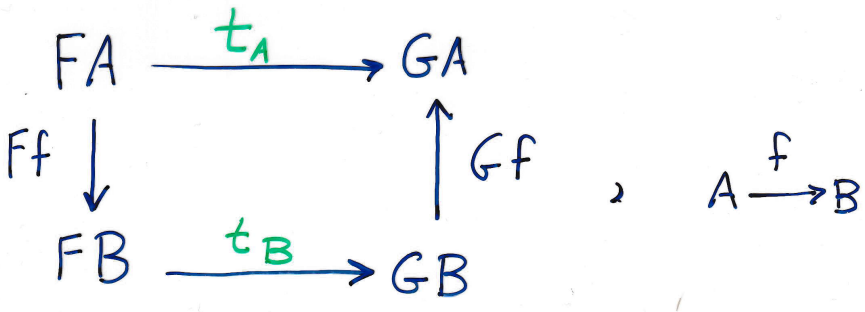


Dinatural transformations II: Examples

Examples

(1) F, G covariant (or contravariant) — gives ordinary n.t.

(2) F covariant, G contrav.



(3) $F = K_1$ (constant)

$G = (-)^{(-)}$, so $GAB = B^A$

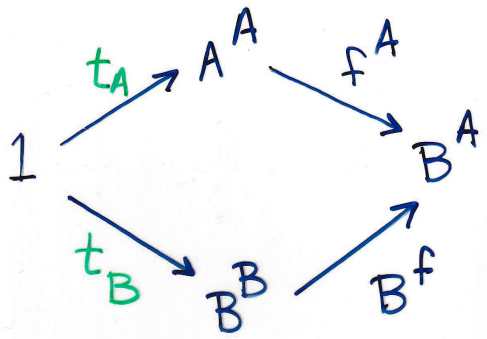
$\therefore t : F \to G = \{ t_A : 1 \to GAA \mid A \in \mathcal{C} \}$

$= \{ t_A : 1 \to A^A \mid A \in \mathcal{C} \}$

(Polymorphic identity)

Dinatural transformations III: Examples

This satisfies:



, $f : A \to B$

$$f \cdot t_A = t_B \cdot f$$

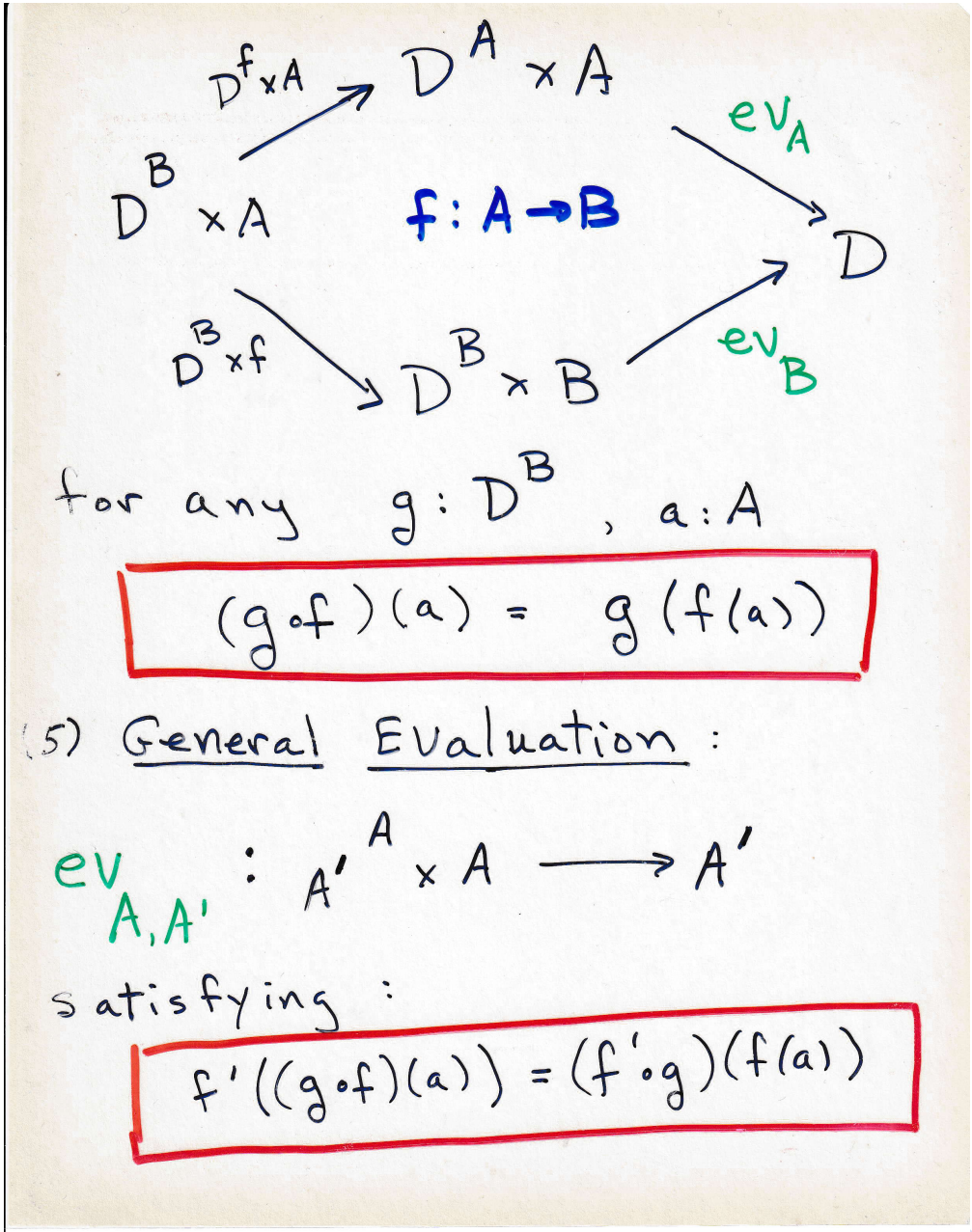
In Set, Vec, ... can prove $t_A = \ulcorner id_A \urcorner = \lambda x.\, x$ (ϵ_A)

(4) Simple Evaluation

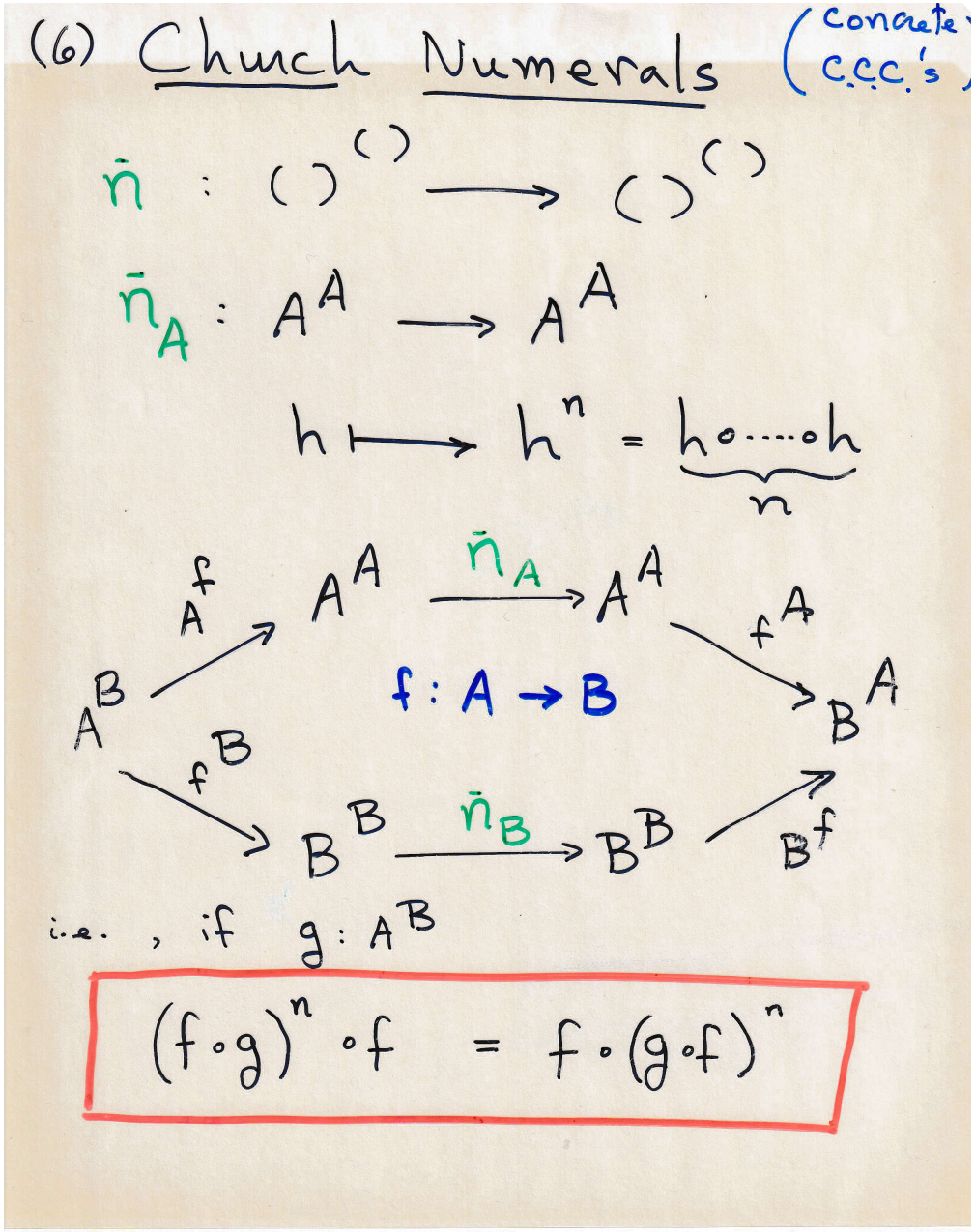
Fix D. $ev_- : D^{(-)} \times (-) \to K_D$

so $ev_A : D^A \times A \to D$ is evaluation in ccc $\mathcal{C}$.

Dinatural transformations III: Examples



Dinatural transformations III: Examples



This is a great idea?!?!

This seems to work really well. In fact, this idea seems to be...

This is a great idea?!?!

This seems to work really well. In fact, this idea seems to be...

This is a great idea?!?!, Part 2

- The intuition that variability=functoriality is quite pleasing.

This is a great idea?!?!, Part 2

- The intuition that variability=functoriality is quite pleasing.

- The equations that the dinatural transformations generate are exactly the sort of equations you would want polymorphic programs to satisfy. Thus for more complicated examples of types, dinaturality will suggest the equations that the programs of that type should satisfy.

This is a great idea?!?!, Part 2

- The intuition that variability=functoriality is quite pleasing.
- The equations that the dinatural transformations generate are exactly the sort of equations you would want polymorphic programs to satisfy. Thus for more complicated examples of types, dinaturality will suggest the equations that the programs of that type should satisfy.
- So let's take a cartesian closed category, say the category **Set** and form the category whose objects are these multivariant functors and arrows are dinatural transformations. This should be a cartesian closed category, right?

This is a great idea?!?!, Part 3

- **BUT**

This is a great idea?!?!, Part 3

- **BUT**
- This too doesn't work, for the most annoying reason imaginable.

This is a great idea?!?!, Part 3

- **BUT**
- This too doesn't work, for the most annoying reason imaginable.
- Dinatural transformations need not compose.

This is a great idea?!?!, Part 3

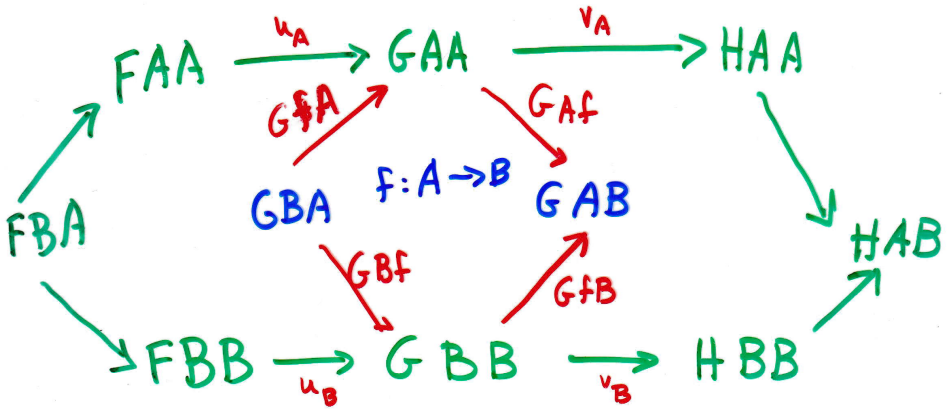
- **BUT**
- This too doesn't work, for the most annoying reason imaginable.
- Dinatural transformations need not compose.
- So we have a cartesian closed non-category.

This is a great idea?!?!, Part 3

- **BUT**
- This too doesn't work, for the most annoying reason imaginable.
- Dinatural transformations need not compose.
- So we have a cartesian closed non-category.
- In [Bainbridge, Freyd, Scedrov, Scott], they give an example of a composable class of dinaturals on a category of partial equivalence relations (PERs).

Composing dinatural transformations

Composing Dinats : Horizontal



In general, outer Hexagon does not commute (middle ◇ does).

FACT: • MIDDLE DIAMOND pullback
⇒ Hexagons compose.

∴ f iso ⇒ Dinat hexagons compose

(∵ families that are dinat w.r.t. isos compose to give dinat w.r.t

An example

Here's an example on the category of finite-dimensional vector spaces. Let K_I be the constant functor at the base field, and then consider the multivariant functor $F(V, W) = V^* \otimes W$.

- Define a dinat as follows:

$$\eta \colon K_I \to F \qquad \eta_V \colon I \to V^* \otimes V \qquad 1 \mapsto \sum e_i^* \otimes e_i$$

where the e_i's vary over an arbitrary basis.

An example

Here's an example on the category of finite-dimensional vector spaces. Let K_I be the constant functor at the base field, and then consider the multivariant functor $F(V, W) = V^* \otimes W$.

- Define a dinat as follows:

$$\eta \colon K_I \to F \qquad \eta_V \colon I \to V^* \otimes V \qquad 1 \mapsto \sum e_i^* \otimes e_i$$

  where the e_i's vary over an arbitrary basis.

- Next, consider the dinat defined by:

$$\epsilon \colon F \to K_I \qquad \epsilon_V \colon V^* \otimes V \to I \qquad f \otimes v \mapsto f(v)$$

An example

Here's an example on the category of finite-dimensional vector spaces. Let K_I be the constant functor at the base field, and then consider the multivariant functor $F(V, W) = V^* \otimes W$.

- Define a dinat as follows:

$$\eta \colon K_I \to F \qquad \eta_V \colon I \to V^* \otimes V \qquad 1 \mapsto \sum e_i^* \otimes e_i$$

  where the e_i's vary over an arbitrary basis.

- Next, consider the dinat defined by:

$$\epsilon \colon F \to K_I \qquad \epsilon_V \colon V^* \otimes V \to I \qquad f \otimes v \mapsto f(v)$$

- The composition of these two dinats should be a dinat of the form $K_I \to K_I$. A dinat between constant functors is a single arrow, but in fact, the composite of these two dinats depends on V, it's just $dim(V)$.

Back to categorical proof theory

The following is due to Jean-Yves Girard, Andre Scedrov and Phil Scott.

Theorem

For any "reasonable" logic with a categorical semantics, the denotations of cut-free proofs determine dinatural transformations.

Proof.

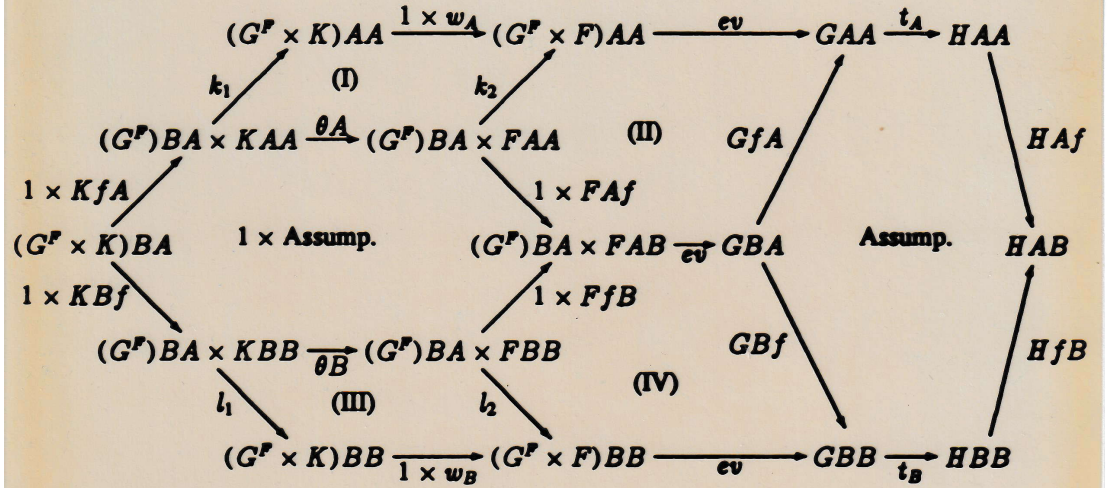
The proof is a straightforward induction on the complexity of the cut-free proof. □

Dinatural transformations III: Examples

$$(\Rightarrow L)\quad \frac{K \vdash F \qquad G \overset{t}{\vdash} H}{K \times (F \Rightarrow G) \vdash H}$$

$$\overset{?!?}{\Longleftrightarrow}$$

$$G^F \times K \vdash H$$



Cut-Elimination

The CUT rule in sequent calculus provides a method of stringing proofs together. In categorical proof theory, it provides our categories of proofs with composition.

$$\frac{\Gamma \vdash A \qquad \Delta, A \vdash C}{\Delta, \Gamma \vdash C}\ CUT$$

The traditional formulation of Gentzen's cut-elimination theorem says:

Theorem

Every proof that can be derived with the use of the cut rule can be derived without the cut rule.

The categorical formulation says:

Cut-Elimination

The CUT rule in sequent calculus provides a method of stringing proofs together. In categorical proof theory, it provides our categories of proofs with composition.

$$\frac{\Gamma \vdash A \qquad \Delta, A \vdash C}{\Delta, \Gamma \vdash C}\ CUT$$

The traditional formulation of Gentzen's cut-elimination theorem says:

Theorem

Every proof that can be derived with the use of the cut rule can be derived without the cut rule.

The categorical formulation says:

Theorem

Every proof that uses the cut rule is equivalent to a proof that does not use the cut rule.

Cut-Elimination II

As a consequence, we can conclude:

Theorem (Girard, Scedrov, Scott)

The dinatural transformations arising as the interpretations of cut-free proofs compose.

Cut-Elimination II

As a consequence, we can conclude:

Theorem (Girard, Scedrov, Scott)

The dinatural transformations arising as the interpretations of cut-free proofs compose.

To obtain less syntactic examples of composable classes of dinats, we will make use of *full completeness theorems*. The first results of this sort used game theory for fragments of linear logic:

- Abramsky & Jagadeesan obtained full completeness for MLL+MIX.
- Hyland & Ong obtained full completeness for MLL.

Full Completeness

While traditional completeness is with respect to provability, full completeness is completeness with respect to proofs. Here's a game-theoretic version of this idea:

Full Completeness

While traditional completeness is with respect to provability, full completeness is completeness with respect to proofs. Here's a game-theoretic version of this idea:

Theorem (Abramsky-Jagadeesan)

If σ is a a uniform history-free winning strategy for the game associated to the sequent $\vdash \Gamma$, then it is the denotation of a unique cut-free proof net proving Γ.

Full Completeness

While traditional completeness is with respect to provability, full completeness is completeness with respect to proofs. Here's a game-theoretic version of this idea:

Theorem (Abramsky-Jagadeesan)

If σ is a a uniform history-free winning strategy for the game associated to the sequent $\vdash \Gamma$, then it is the denotation of a unique cut-free proof net proving Γ.

RB and Phil Scott obtained full completeness in a category of topological vector spaces and dinatural transformations.

Lefschetz topology

Lefschetz introduced this notion of topology with the intent of having infinite-dimensional vector spaces which are isomorphic to their second dual.

Definition

A vector space is a Lefschetz space if equipped with a T_0-topology such that

- The vector operations are continuous, i.e. it is a topological vector space. (We'll assume that the base field is discrete.)
- $0 \in V$ has a neighborhood basis of open linear subspaces.

The category of Lefschetz spaces and continuous linear maps will be denoted Lef.

Lefschetz topology II

Lemma (Barr)

Lef *is symmetric, monoidal closed. The tensor is described by a topology on the algebraic tensor product.*

Lefschetz topology II

Lemma (Barr)

Lef *is symmetric, monoidal closed. The tensor is described by a topology on the algebraic tensor product.*

Lemma (Lefschetz)

The embedding $\rho\colon V \to V^{**}$ *is a bijection for all Lefschetz spaces.*

Lefschetz topology II

Lemma (Barr)

Lef *is symmetric, monoidal closed. The tensor is described by a topology on the algebraic tensor product.*

Lemma (Lefschetz)

The embedding $\rho\colon V \to V^{**}$ *is a bijection for all Lefschetz spaces.*

Definition

Let RLef *be the full subcategory of reflexive objects, i.e. those objects for which* ρ *is an isomorphism.*

Lefschetz topology III

Theorem (Barr)

RLef *is a* $*$*-autonomous category. In fact,* RLef *is a reflective subcategory of* Lef *with reflection given by* $(-)^{**}$.

In fact, this category was one of the primary motivations for the definition of $*$-autonomous category.

Linear Läuchli Semantics

Definition

Let F and F' be formulas in Multiplicative Linear Logic, interpreted as multivariant functors on RLef. We call the space of dinatural transformations $F \to F$ the *proof space* of $F \vdash F'$, denoted $\mathrm{PRF}(F, F')$. It is a vector space.

Linear Läuchli Semantics

Definition

Let F and F' be formulas in Multiplicative Linear Logic, interpreted as multivariant functors on RLef. We call the space of dinatural transformations $F \to F$ the *proof space* of $F \vdash F'$, denoted $\mathrm{PRF}(F, F')$. It is a vector space.

Theorem (RB, Scott)

The space $\mathrm{PRF}(F, F')$ *has as a basis the denotations of cut-free proofs of* $F \vdash F'$ *in* MLL+MIX. *Such dinatural transformations therefore necessarily compose.*

Linear Läuchli Semantics

Definition

Let F and F' be formulas in Multiplicative Linear Logic, interpreted as multivariant functors on RLef. We call the space of dinatural transformations $F \to F$ the *proof space* of $F \vdash F'$, denoted $\mathrm{PRF}(F, F')$. It is a vector space.

Theorem (RB, Scott)

The space $\mathrm{PRF}(F, F')$ *has as a basis the denotations of cut-free proofs of* $F \vdash F'$ *in* MLL+MIX. *Such dinatural transformations therefore necessarily compose.*

Theorem (RB, Scott)

If we consider only those dinatural transformations invariant under a noncocommutative Hopf algebra called the shuffle Hopf algebra, *we obtain the same theorem for the noncommutative version of linear logic called* Cyclic MLL+MIX.

Cyclic Linear Logic

Cyclic linear logic [Yetter] is a noncommutative linear logic, where one still assumes that the left negation is equal to the right negation,

$$^{\perp}A = A^{\perp}$$

As a consequence, one obtains a limited notion of exchange.

$$\frac{\vdash \Gamma}{\vdash \sigma(\Gamma)}$$

where σ is a *cyclic* permutation.

Cyclic Linear Logic II

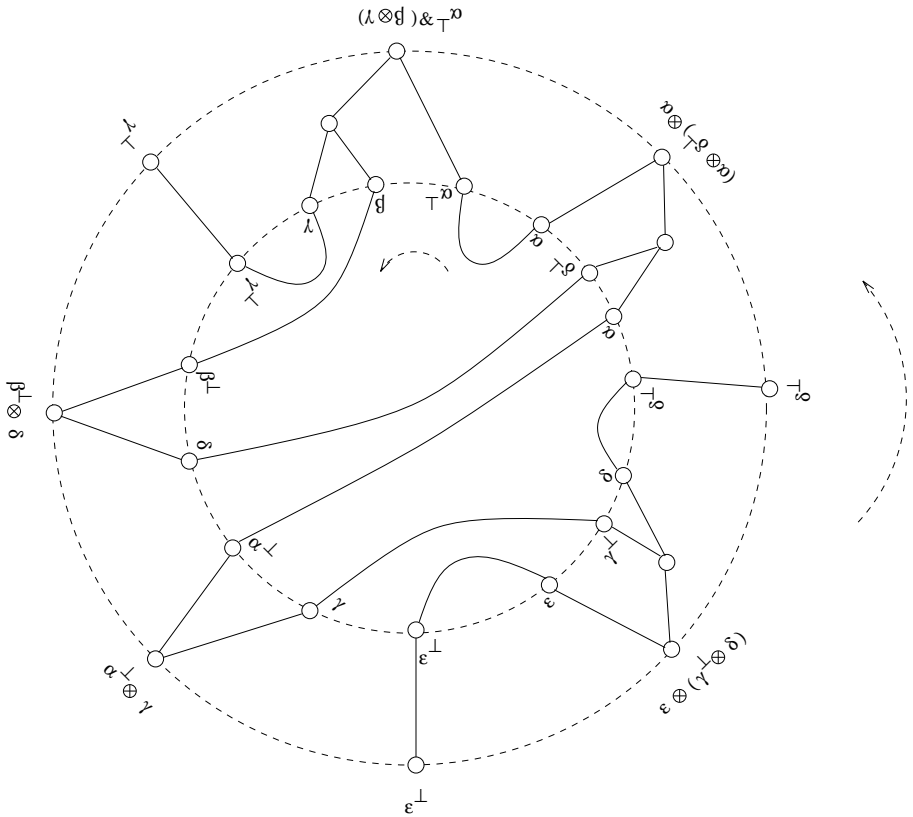


Figure 1: A cyclic proof net

Thanks!

Thank you for listening.