A Programming Language for Virtual Double Theories

VIRTUAL WORKSHOP ON DOUBLE CATEGORIES · NOVEMBER 5, 2026

Bryce Goldman





Introduction



Outline

1. Introduction (you are here)
2. A Type Theory for Virtual Double Categories
3. A Domain-Specific Language for Virtual Double Theories
4. Future Work
5. Maybe questions? Who knows what the world will look like in 35 minutes



Presentations of Theories

We are often interested in presenting (first-order, algebraic, etc.) theories by some small collection of operation symbols and relations between them.

Definition:

- The **signature** of a theory is the collection of operation symbols.
- The **axioms** of a theory are the equalities or formulas relating those symbols.

Idea:

- The signature is an *interface*.
- The axioms are a description of the interface's *behavior*.

A **model** of a theory is a concrete interpretation of the symbols in the signature which satisfy the axioms.



Double Theories

In *categorical logic* we are interested in viewing theories themselves as categorical structures.

Idea: many theories can be expressed as double categories; their models are naturally valued in categories (Lambert and Patterson 2024).

These can likewise be presented via a signature and axioms:

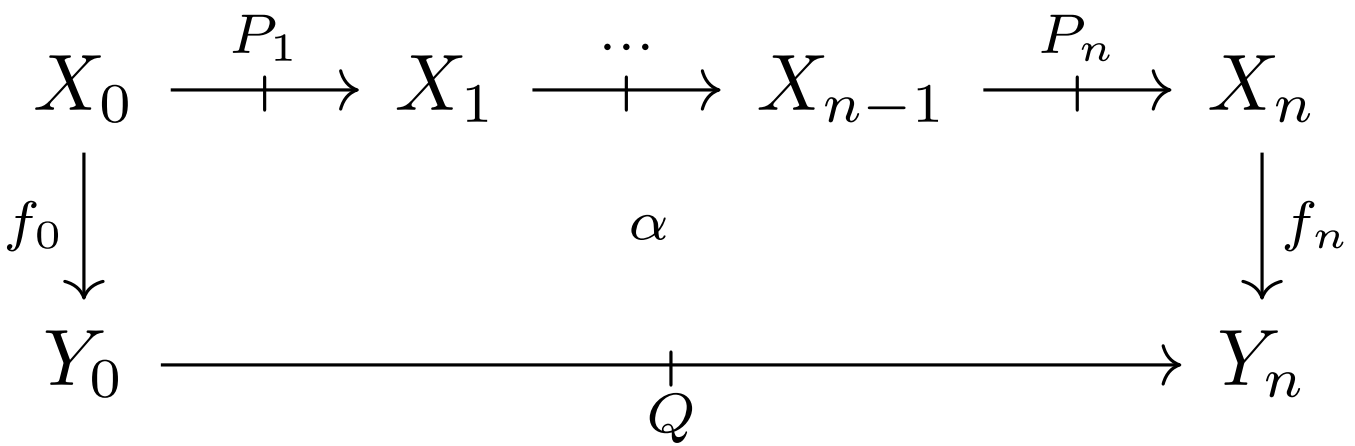
- The signature declares the generating objects, tight morphisms, proarrows, and cells of the double category.
- The axioms impose equalities on the components of the double category.



Virtual Double Categories

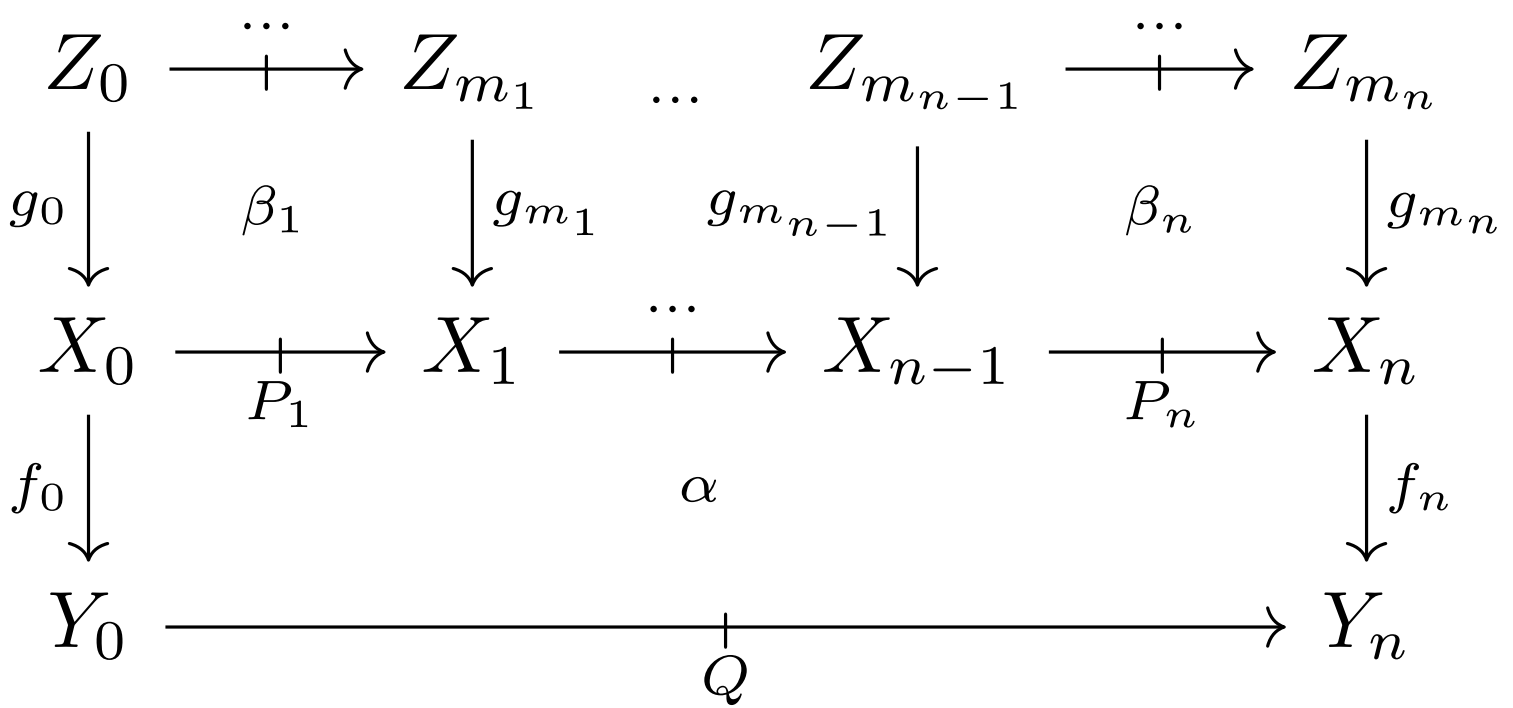
Our double theories will be *virtual*, meaning they are represented as **virtual double categories**.

Like ordinary double categories the tight direction is still a (strict) category. Cells map from a path of (zero or more) proarrows, into a single proarrow:



Virtual Double Categories (Cont.)

Tight composition is defined for compatible cells.



Loose composition and loose units are additional universal properties that a VDC may or may not have. For now we will ignore these, but be emotionally prepared for them to show up in some examples later.

Fibrational VDCs

Our VDCs will be **fibrational**, meaning they admit *restrictions*.

Given a proarrow $P : X' \nrightarrow Y'$ and tight morphisms $f : X \to X'$, $g : Y \to Y'$, there is a cartesian cell

$$\begin{array}{ccc} X & \xrightarrow{P(f;g)} & Y \\ {\scriptstyle f}\downarrow & \text{res} & \downarrow{\scriptstyle g} \\ X' & \xrightarrow[P]{} & Y' \end{array}$$

We say that $P(f;g)$ is the *restriction of* P *along* f *and* g.

Virtual Double Theories in Software

There is a mechanical procedure for constructing a type theory over the internal languages of models of a virtual double theory (Lynch and Patterson 2026).

Example: there is a simple presentation for the double theory of database schemas consisting of a single proarrow. This induces a type theory for the internal language of databases over a given schema.

CatColab, the Topos Institute's a scientific modeling ecosystem, uses this idea to implement a uniform type system across many application domains.

Goal: develop a domain-specific programming language (DSL) for specifying virtual double theories, so that users can work with any the categorical logic in CatColab.

We want to prevent the user from specifying a malformed theory, so we will use a type theory for VDCs to eliminate bugs.



A Type Theory for VDCs



FVDblTT

Our type theory is an extension of FVDblTT (Nasu 2025), which proposes a type theory for reasoning in the internal language of cartesian fibrational virtual double categories. Our setup differs in a few ways, in order to more readily facilitate our software implementation for (finitely presented) virtual double theories:

1. We permit cells in the signature to map out of arbitrary proarrows (including restrictions). This generalization is necessary to specify certain theories in categorical logic, such as the theory of adjunctions we will see as an example later.
2. Our type theory for terms is unary, since we do not require our VDCs to be cartesian.
3. The syntax and judgments of our type theory are adjusted slightly to translate more directly into a mechanical type-checking procedure.



Judgments

Judgment	Notation
Type	X ty
Context	Γ ctx
Term	$\Gamma \vdash s : X$
Protype	$\Gamma_0, \Gamma_1 \vdash \underline{P}$ proty
Boundary	$\Gamma_0, \dots, \Gamma_n \mid \underline{P}_1, \dots, \underline{P}_n \to \underline{Q}$
Procontext	$\Gamma_0, \dots, \Gamma_n \mid p_1 : \underline{P}_1, \dots, p_n : \underline{P}_n$ proctx
Substitution	$\overline{\Gamma} \mid \Omega \vdash \{s_0, \dots, s_n\}(\tau_1, \dots, \tau_n) : (\underline{P}_1, \dots, \underline{P}_n)$
Proterm	$\overline{\Gamma} \mid \Omega \vdash \tau : \underline{P}$

(Here $\overline{\Gamma}$ is shorthand for $\Gamma_0, \dots, \Gamma_n$, and Ω for $p_1 : \underline{P}_1, \dots, p_n : \underline{P}_n$.)



Signature

The signature Σ of a virtual double category consists of:

1. a set of *object symbols* $\mathcal{T}_\Sigma$.
2. a set of *function symbols* $\mathcal{F}_\Sigma(X, Y)$ for each pair $X, Y \in \mathcal{T}_\Sigma$.
3. a set of *proarrow symbols* $\mathcal{P}_\Sigma(X, Y)$ for each pair $X, Y \in \mathcal{T}_\Sigma$.
4. a set of *cell symbols* $\mathcal{C}_\Sigma$, each of which is associated with a *boundary* specifying its arity, domain, and codomain.



Types and Term Contexts

Types are merely object symbols:

$$\frac{X \in \mathcal{T}_\Sigma}{X\ \mathrm{ty}}$$

Term contexts are unary, consisting of exactly one variable binding:

$$\frac{X\ \mathrm{ty}}{(x : X)\ \mathrm{ctx}}$$



Terms

Terms are typed by term contexts. The type theory for terms forms a *(cut-free) unary type theory for categories* (Shulman 2016):

$$\frac{X\ \mathrm{ty}}{x : X \vdash x : X}$$

$$\frac{\Gamma \vdash s : X \quad f \in \mathcal{F}_\Sigma(X, Y)}{\Gamma \vdash f(s) : Y}$$



Pointful Syntax

Given morphisms

$$X \xrightarrow{f} Y \xrightarrow{g} Z \xrightarrow{h} W$$

the composite $h \circ (g \circ f) = (h \circ g) \circ f$ has the *unique* representation (up to α-equivalence)

$$x : X \vdash h(g(f(x))) : W.$$

The identity at X is represented by

$$x : X \vdash x : X.$$



Protypes

Each proarrow symbol in $\mathcal{P}_\Sigma$ gives rise to a dependent family of "protypes":

$$\frac{\Gamma_0 \vdash s : X \quad \Gamma_1 \vdash t : Y \quad P \in \mathcal{P}_\Sigma(X, Y)}{\Gamma_0, \Gamma_1 \vdash P(s,t)\ \mathrm{proty}}$$

(We abbreviate $P(s,t)$ by $\underline{P}$ when s and t are not relevant.)

Recall that s and t represent tight morphisms. The protype $P(s,t)$ represents the restriction of the proarrow P along s and t.



Cell Boundaries

Recall: each cell symbol $\alpha \in \mathcal{C}_\Sigma$ is associated with a boundary. These are judgments of the form

$$\frac{x_{i-1} : X_{i-1}, x_i : X_i \vdash \underline{P}_i \text{ proty} \quad (i = 1, \dots, n) \qquad x_0 : X_0, x_n : X_n \vdash \underline{Q} \text{ proty}}{\alpha : (x_0 : X_0, \dots, x_n : X_n \mid \underline{P}_1, \dots, \underline{P}_n) \to \underline{Q}}$$

We say that α has *arity* n, with *domain* $\underline{P}_1, \dots, \underline{P}_n$ and *codomain* $\underline{Q}$.

Procontexts

The “procontext” that a cell (“proterm”) is typed in represents a path of zero or more proarrows:

$$X_0 \overset{P_1}{\longrightarrow\!\!\!\!\!\!\mid} X_1 \overset{\cdots}{\longrightarrow\!\!\!\!\!\!\mid} X_{n-1} \overset{P_n}{\longrightarrow\!\!\!\!\!\!\mid} X_n$$

These are formed inductively, as a sequence of protypes with contexts matching at common junctures:

$$\frac{\Gamma \text{ ctx}}{\Gamma \mid \cdot \text{ proctx}} \qquad \frac{\overline{\Gamma}_L, \Gamma \mid \Omega \text{ proctx} \quad \Gamma, \Gamma_R \vdash \underline{P} \text{ proty}}{\overline{\Gamma}_L, \Gamma, \Gamma_R \mid \Omega, p : \underline{P} \text{ proctx}}$$

Note: both components of procontexts are *ordered*. The proterm component is also *linear*, meaning that each proterm variable must be used exactly once.



Proterms

Types and terms represented objects and (tight) morphisms of the VDC.

Proarrows behave like objects with a source and target—these should correspond to types (“protypes”) with a certain kind of dependency, as we’ve seen.

Cells are like n-ary maps between proarrows, so these should be terms (“proterms”) constructed as trees of cell symbols. Like the syntax for terms, our syntax for proterms will be pointful.



Proterms: Identity Cells

The (tight) identity cell on a proarrow P is represented by the typing judgment for a proterm variable:

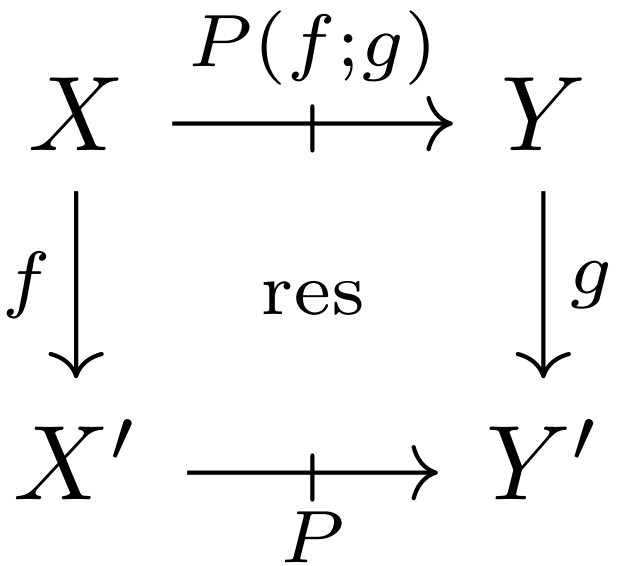
$$\frac{\Gamma_0, \Gamma_1 \vdash \underline{P}\ \mathrm{proty}}{\Gamma_0, \Gamma_1 \mid p : \underline{P} \vdash p : \underline{P}}$$

Like the variables in the unary type theory for terms, these are the leaves of proterm typing derivations.



Proterms: Restriction Cells

Note: a proterm variable $p : P(f(x), g(y))$ simultaneously represents the identity on $P(f; g)$ and the cartesian cell



In this way the syntax of the language normalizes the fibrational structure of a virtual equipment.

Proterms: Substitutions

We represent the input to an n-ary cell as a sequence of proterms of length n which have compatible boundary terms (tight morphisms):

$$\frac{\Gamma_0 \vdash s_0 : X_0}{\Gamma_0 \mid \cdot \vdash \{s_0\}() : ()}$$

$$\frac{\begin{array}{c}\overline{\Gamma}_L, x_n : X_n \mid \Omega_L \vdash \{s_0, \ldots, s_n\}(\tau_1, \ldots, \tau_n) : (\underline{P}_1, \ldots, \underline{P}_n) \\ x_n : X_n, \overline{\Gamma}_R, x_{n+1} : X_{n+1} \mid \Omega_R \vdash \tau_{n+1} : P_{n+1}(t_L[s_n/x_n], t_R[s_{n+1}/x_{n+1}])\end{array}}{\overline{\Gamma}_L, x_n : X_n, \overline{\Gamma}_R, x_{n+1} : X_{n+1} \vdash \{s_0, \ldots, s_{n+1}\}(\tau_1, \ldots, \tau_{n+1}) : (\underline{P}_1, \ldots, \underline{P}_{n+1})}$$



Proterms: Cell Composition

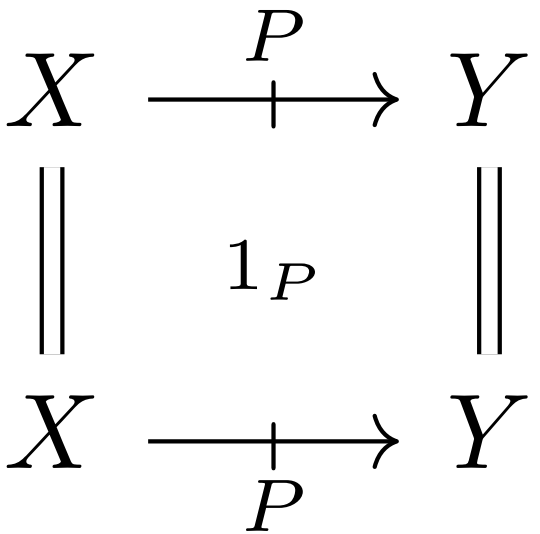
Cell generators are applied to substitutions of the correct form:

$$\frac{\begin{array}{c}\alpha : (x_0 : X_0, \ldots, x_n : X_n \mid \underline{P}_1, \ldots, \underline{P}_n) \to \underline{Q} \\ \overline{\Gamma} \mid \Omega \vdash \{s_0, \ldots, s_n\}(\tau_1, \ldots, \tau_n) : (\underline{P}_1[s_0/x_0, s_1/x_1], \ldots, \underline{P}_n[s_{n-1}/x_{n-1}, s_n/x_n])\end{array}}{\overline{\Gamma} \mid \Omega \vdash \alpha\{s_0, \ldots, s_n\}(\tau_1, \ldots, \tau_n) : \underline{Q}[s_0/x_0, s_n/x_n]}$$

Note that the term arguments $s_0, \ldots, s_n$ allow the domain protypes to be restricted. The idea is that a protype $P(f(x), g(y))$ should be thought of as a "subtype" of $P(x', y')$, for example.

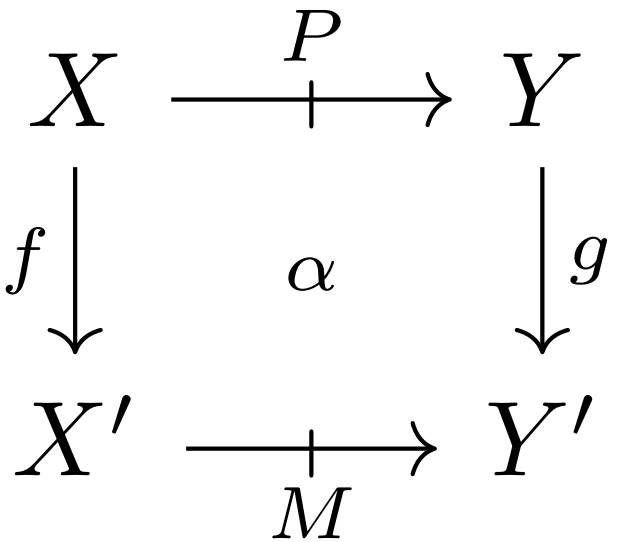


Examples: Identity Cell



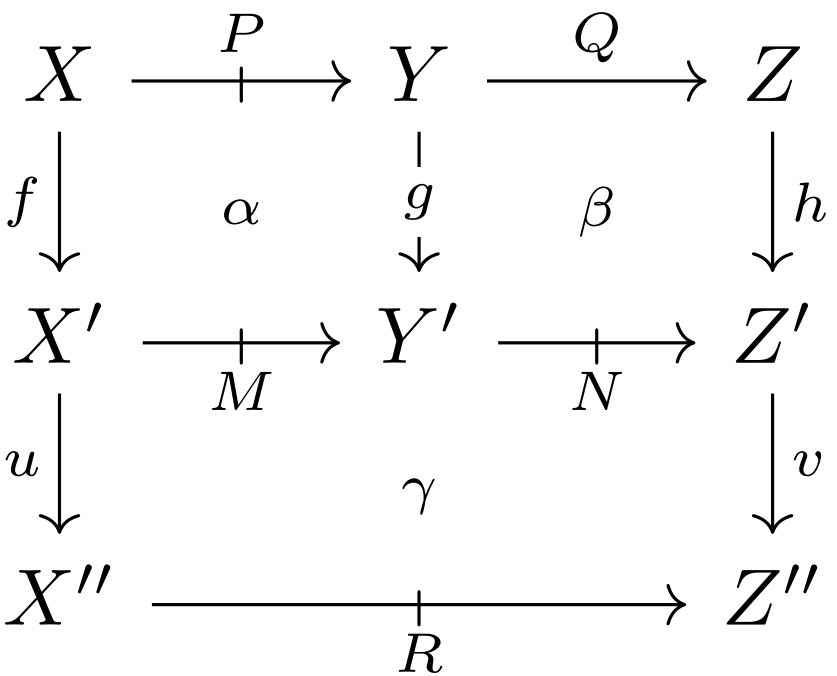
$$x : X, y : Y \mid p : P(x, y) \vdash p : P(x, y)$$

Examples: Cell Generator



$$\frac{\alpha : (x : X, y : Y \mid P(x, y)) \to M(f(x), g(y)) \qquad x : X, y : Y \mid p : P(x, y) \vdash \{x, y\}(p) : (P(x, y))}{x : X, y : Y \mid p : P(x, y) \vdash \alpha\{x, y\}(p) : M(f(x), g(y))}$$

Examples: Cell Composite

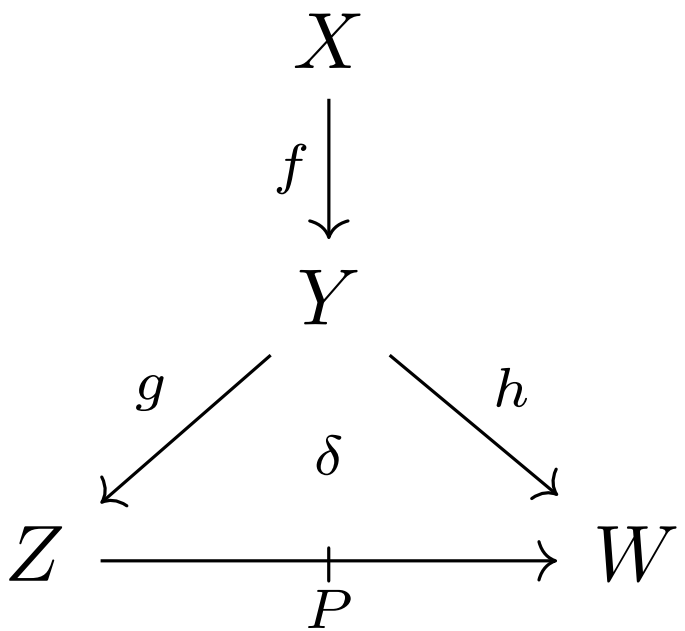


$$\frac{\begin{array}{c} x : X, y : Y \mid p : P(x,y) \vdash \alpha\{x,y\}(p) : M(f(x), g(y)) \\ y : Y, z : Z \mid q : Q(y,z) \vdash \beta\{y,z\}(q) : N(g(y), h(z)) \end{array}}{\begin{array}{c} x, y, z \mid p, q \vdash \{x,y,z\}(\alpha\{x,y\}(p), \beta\{y,z\}(q)) : (M(f(x), g(y)), N(g(y), h(z))) \\ \gamma : (x' : X', y' : Y', z' : Z' \mid M(x', y'), N(y', z')) \to R(u(x'), v(z')) \end{array}}$$

$$x, y, z \mid p, q \vdash \gamma\{f(x), g(y), h(z)\}(\alpha\{x,y\}(p), \beta\{y,z\}(q)) : R(u(f(x)), v(h(z)))$$

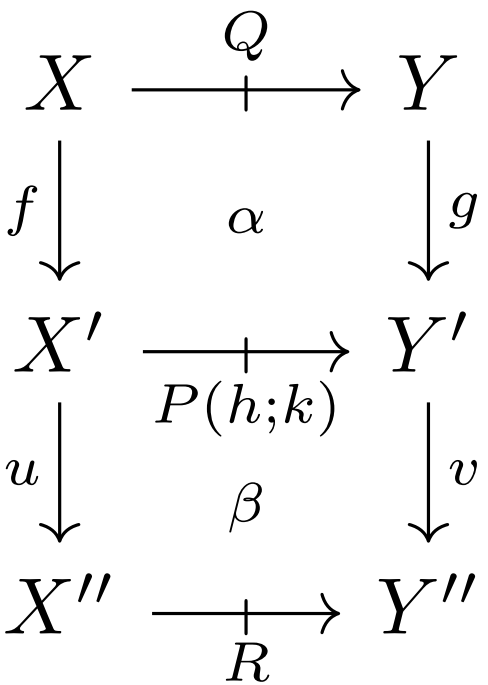


Examples: Nullary Cell



$$\frac{\delta : (y : Y \mid \cdot) \to P(g(y), h(y)) \quad x : X \mid \cdot \vdash \{f(x)\}() : ()}{x : X \mid \cdot \vdash \delta\{f(x)\}() : P(g(f(x)), h(f(x)))}$$

Examples: Restricted Domain



$$\frac{\begin{array}{c} x : X, y : Y \mid q : Q(x,y) \vdash \{f(x), g(y)\}(\alpha\{x,y\}(q)) : (P(h(f(x)), k(g(y)))) \\ \beta : (x' : X', y' : Y' \mid P(h(x'), k(y'))) \to R(u(x'), v(y')) \end{array}}{x : X, y : Y \mid q : Q(x,y) \vdash \beta\{f(x), g(y)\}(\alpha\{x,y\}(q)) : R(u(f(x)), v(g(x)))}$$

A DSL for Virtual Double Theories



Surface Syntax: Objects and Arrows

Objects

```
obj X       -- declares an object symbol X
obj [Y, Z]  -- declares object symbols Y, Z
```

Tight Morphisms

```
-- function symbol declarations
fun f : X -> Y
fun g : Y -> Z
```

Proarrows

```
-- proarrow symbol declarations
pro P : X => Y
pro Q : Y => Z
pro R : X => Z
```



Surface Syntax: Cells

Cells

```
-- declares a unary cell α from P to Q with source
-- arrow f and target arrow g
cell α : [x : X, y : Y] | [P[x, y]] -> Q[f(x), g(y)]

-- declares a globular binary cell β from [P, R] to Q
cell β : [x : X, y : Y, z : Z] | [P[x, y], Q[y, z]] -> R[x, z]
```

Proarrow Isomorphisms

```
-- provides an isomorphism between the proarrows P(g; id) and Q(id; g)
iso [y0 : Y, y1 : Y] ⊢ [p2q, q2p] : P[g(y0), y1] <=> Q(y0, g(y1))
```



Surface Syntax: Axioms

Axioms

```
-- declares a proterm axiom making a globular cell γ : P => P idempotent
axiom [x : X, y : Y] | [p : P[x, y]] ⊢ γ [γ [p]] ≡ γ [p]
```

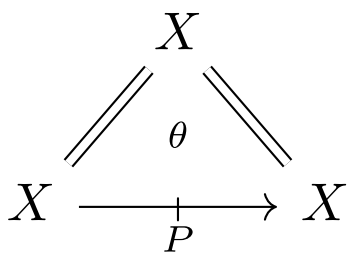
Proarrow isomorphisms are syntactic sugar for a pair of cell declarations and a pair of axioms making them inverses.

```
--  the previous isomorphism induces the following axioms:
axiom [y0 : Y, y1 : Y] | [_p0 : P[g(y0), y1]] ⊢ q2p[p2q[_p0]] ≡ _p0
axiom [y0 : Y, y1 : Y] | [_p1 : Q[y0, g(y1)]] ⊢ p2q[q2p[_p1]] ≡ _p1
```

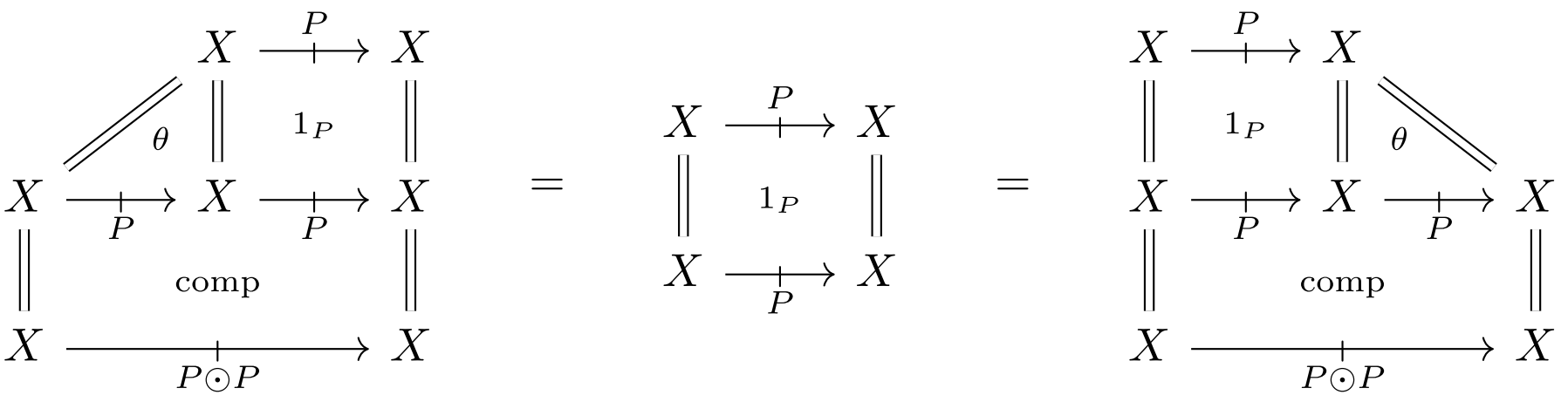


Examples: Theory of Promonads

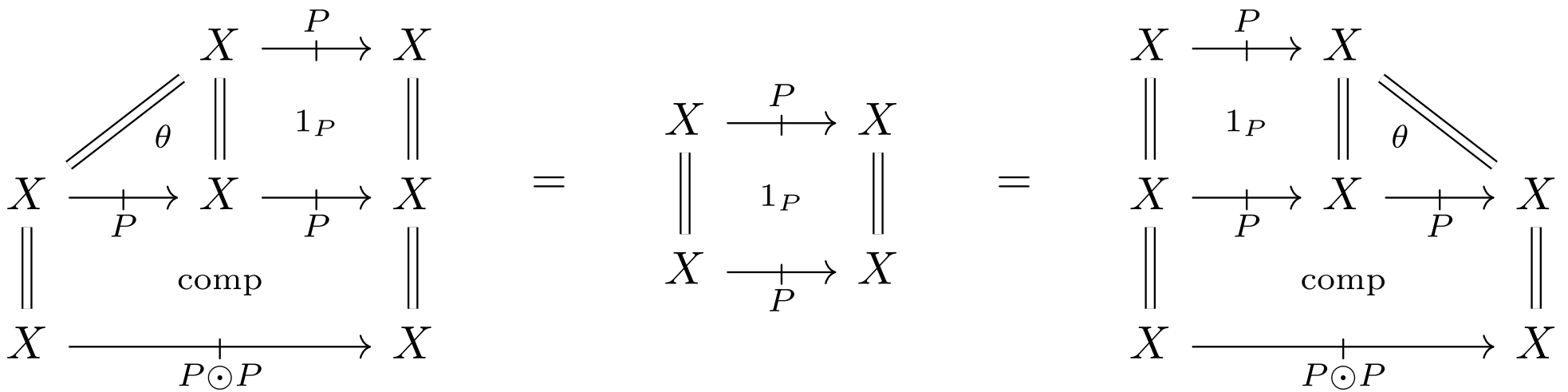
The theory of **promonads** is a unital VDC with an object X, a proarrow $P : X \nrightarrow X$ such that the loose composite $P \odot P$ is defined and is equal to P, and a virtual cell θ:



subject to the following axioms:



Examples: Theory of Promonads (Cont.)



We can specify this theory in our DSL via

```
obj X
pro P : X => X
cell θ : [x : X] -> P[x, x]
cell comp : [x0 : X, x1 : X, x2 : X] | [P[x0, x1], P[x1, x2]] -> P[x0, x2]

axiom [x0 : X, x1 : X] | [p : P[x0, x1]] ⊢ comp[θ(x0), p] ≡ p
axiom [x0 : X, x1 : X] | [p : P[x0, x1]] ⊢ comp[p, θ(x1)] ≡ p
```



Examples: Theory of Adjunctions

One presentation for the theory of adjunctions is specified by a pair of objects X, Y and tight morphisms $f : X \to Y$ (the left adjoint) and $g : Y \to X$ (the right adjoint), together with inverse cells

$$\begin{array}{ccc} X & \xrightarrow{\mathrm{id}_Y(f;1_Y)} & Y \\ \| & \varphi & \| \\ X & \xrightarrow[\mathrm{id}_X(1_X;g)]{} & Y \end{array} \qquad \begin{array}{ccc} X & \xrightarrow{\mathrm{id}_X(1_X;g)} & Y \\ \| & \varphi^{-1} & \| \\ X & \xrightarrow[\mathrm{id}_Y(f;1_Y)]{} & Y \end{array}$$

where id denotes the loose unit and 1 denotes the tight unit.



Examples: Theory of Adjunctions (Cont.)

We can describe this theory in our programming language with

```
obj [X, Y]
fun f : X -> Y
fun g : Y -> X
pro idX : X => X
pro idY : Y => Y
```

and cells

```
-- inverse cells
cell l2r : [x : X, y : Y] | [l : idY[f(x), y]] -> idX[x, g(y)]
cell r2l : [x : X, y : Y] | [r : idX[x, g(y)]] -> idY[f(x), y]

-- axioms making the cells inverses
axiom [x : X, y : Y] | [l : idY[f(x), y]] ⊢ r2l[l2r[l]] ≡ l
axiom [x : X, y : Y] | [r : idX[x, g(y)]] ⊢ l2r[r2l[r]] ≡ r
```

or we can use the syntactic sugar for isocells and simply write

```
-- simultaneously declares the cells l2r and r2l along with the axioms making them inverses
iso  [x : X, y : Y] ⊢ [l2r, r2l] : idY[f(x), y] <=> idX[x, g(y)]
```



DEMO

(This is where Bryce will do The Demo™)

(if he is not already doing The Demo™ you should politely remind him to do The Demo™)



Future Work



Loose Units and Composites

In the two example theories we needed to use both **loose units** and **loose composites**. For illustrative purposes we defined these manually, but ideally they should be primitives in the language.

Problem: loose units and composites satisfy universal properties, which impose a huge number of axioms.

Goal: capture these axioms with the language's *operational semantics*, so that applying the universal properties becomes a normalization procedure.

This is hard. :(



Modalities

Many theories use **modalities** (read: well-behaved double monads). For instance, one can give a presentation of the theory of monoidal categories using the list modality, replacing the monoidal identity and binary tensor product with a single tight morphism

$$\otimes : \mathrm{List}(X) \to X.$$

Idea: take the closure of the generators under modal application, and introduce primitive cells for the monad unit and multiplication.

Problem: the monad laws demand some computational mechanism to handle equalities of tight morphisms, which induce equalities of (pro)types. Judgmental equalities between types famously make type-checking hard.



Thanks for listening!





References

CatColab Development Team. 2026. *CatColab*. V. v0.6.1. GitHub, released. https://github.com/ToposInstitute/CatColab.

Cruttwell, G. S. H., and Michael A. Shulman. 2010. *A Unified Framework for Generalized Multicategories*. https://arxiv.org/abs/0907.2460.

Lambert, Michael, and Evan Patterson. 2024. “Cartesian Double Theories: A Double-Categorical Framework for Categorical Doctrines.” *Advances in Mathematics* 444 (May): 109630. https://doi.org/10.1016/j.aim.2024.109630.

Li, Runming, Harrison Grodin, and Robert Harper. 2026. *Directed Proof-Relevant Logical Relations in Simplicial HoTT*. https://arxiv.org/abs/2607.08154.

Lynch, Owen. 2026. *Fnotation*. GitHub, released. https://github.com/ToposInstitute/fnotation.

Lynch, Owen, and Evan Patterson. 2026. “DoubleTT.” Topos Institute, January. https://next.catcolab.org/rfc/0002.

Nasu, Hayato. 2025. *An Internal Logic of Virtual Double Categories*. https://arxiv.org/abs/2410.06792.

Nelson, Greg. 1980. “Techniques for Program Verification.” PhD thesis, Stanford University.

Nelson, Greg, and Derek C. Oppen. 1980. “Fast Decision Procedures Based on Congruence Closure.” *J. ACM* (New York, NY, USA) 27 (2): 356–64. https://doi.org/10.1145/322186.322198.

Patterson, Evan. 2026. “Internal Languages for Models.” Topos Institute, April. https://next.catcolab.org/rfc/0004.

Petersen, Leaf, Robert Harper, Karl Crary, and Frank Pfenning. 2003. “A Type Theory for Memory Allocation and Data Layout.” *SIGPLAN Not.* (New York, NY, USA) 38 (1): 172–84. https://doi.org/10.1145/640128.604147.

Polakow, Jeff, and Frank Pfenning. 2001. “Ordered Linear Logic and Applications.” https://dl.acm.org/

