ModalTT: A Type Theory for Modal Virtual Double Theories

TOPOS INSTITUTE BERKELEY SEMINAR · AUGUST 18, 2026

Bryce Goldman

Introduction

Overview

I will be talking about a type theory and accompanying DSL for specifying **modal virtual double theories**.

Some familiarity with virtual double categories (VDCs) will be assumed. Very little type theory background will (hopefully) be required.

This project is ongoing and there is still much work to be done on both the implementation and type theory. Please be gentle with me. :)

Outline

The next 50 minutes of our lives should look roughly as follows:

1. Motivate the software problem this project is attempting to solve.
2. Outline some past work that motivated our approach.
3. Briefly demo our DSL.
4. Describe our type theory for modal virtual double theories.
5. (Maybe questions? Who can say what the world will look like in an hour.)

Motivation

DoubleTT and categorical logic

CatColab [1] is a collaborative modelling platform based on **categorical logic**.

DoubleTT [2] is a component of CatColab which facilitates model composition using a type theory for models of double theories.

The **theory** for a particular logic defines the "shape" of its models. Formally:

- a **theory** is a fibrational VDC (possibly unital, possibly with loose composites) equipped with zero or more double monads, called **modalities**.
- a **model** M of a theory $\mathbb{T}$ is a restriction-preserving functor

$$M : \mathbb{T} \to \mathbb{S}\mathrm{pan}$$

DoubleTT and categorical logic

Example: the theory of **categories** is a unital VDC with a single object and no (nonunital) tight/loose arrows.

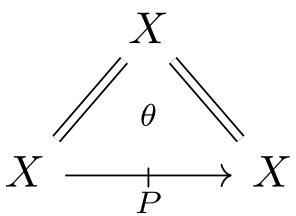
Example: the theory of **monoidal categories** is a unital VDC, equipped with "the" list monad List, generated by an object X and a tight morphism

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

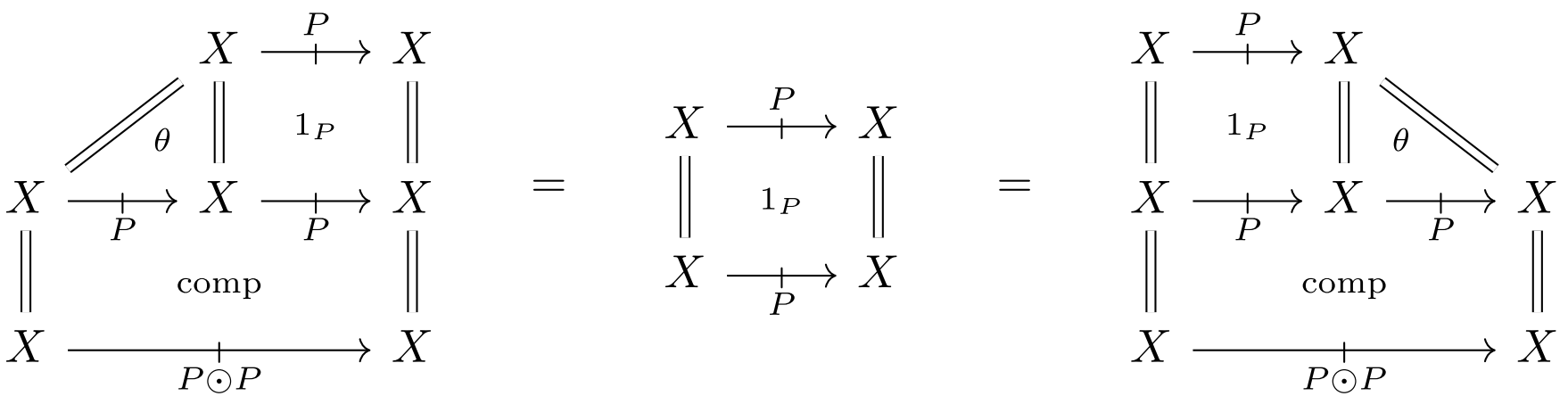
(These are equipped with the usual axioms making $\otimes$ a monoidal product.)

DoubleTT and categorical logic

Example: 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:



such that the cell equalities below are satisfied:



Specifying custom theories

Currently the theories available in DoubleTT are constructed directly in Rust, rather than a dedicated language. Users are limited to a small handful of pre-defined options unless they manually introduce their own using CatColab's internal data structures.

Goal: allow the user to easily define their favorite theory in a small DSL by

1. specifying a signature (generators)
2. specifying axioms (relations)
3. with a guarantee that the DSL only accepts theories which are well-defined (without requiring any internal consistency checks later).

Specifying custom theories

Idea: annotate the user-specified theory with types, such that a theory which successfully typechecks is always valid.

Observation: user-defined axioms can equate any pair of cells with the same boundary, so this amounts to a type theory for the internal language of VDCs.

Remark: as we saw for the theory of promonads, theories may also equate (pro)arrows, so even checking compatibility of cell boundaries is nontrivial.

Past Work

An internal logic of VDCs

Hayato Nasu's past work [3] on a type theory for (fibrational, cartesian) VDCs guided the early stages of this project.

Idea: describe a **pointful syntax** [2,4] for the constructs in a VDC, which reduces equivalent (via universal property) structures to a single normal form.

An internal logic of VDCs

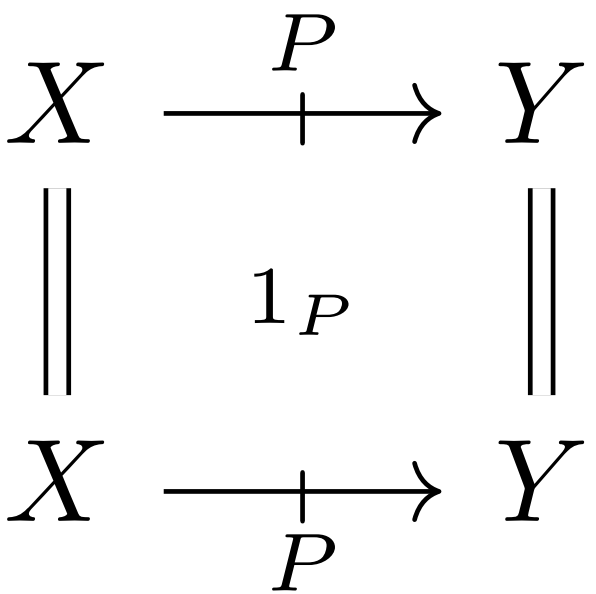
The correspondence between constructs in a VDC and the syntax of Nasu's type theory is outlined below:

VDC Construct	Syntactic Form	Example
Object	Type	$X \Leftrightarrow X \text{ type}$
Product	Context	$\prod_{i=1}^{n} X_i \Leftrightarrow x_1 : X_1, \dots, x_n : X_n \text{ ctx}$
Morphism	Term	$s : \Gamma \to X \Leftrightarrow \Gamma \vdash s : X$
Proarrow	Protype	$P : \Gamma \nrightarrow \Delta \Leftrightarrow \Gamma; \Delta \vdash P(s;t) \text{ protype}$
Proarrow path	Procontext	$\Gamma_0 \nrightarrow \Gamma_1 \nrightarrow \cdots \nrightarrow \Gamma_n \Leftrightarrow \rho_1 : P_1, \dots, \rho_n : P_n \text{ proctx}$
(Globular) Cell	Proterm	$\mu : P_1, \dots, P_n \nrightarrow Q \Leftrightarrow \Gamma_0; \dots; \Gamma_n \vert \rho_1 : P_1, \dots, \rho_n : P_n \vdash \mu : Q$

An internal logic of VDCs

Example

The (tight) identity cell



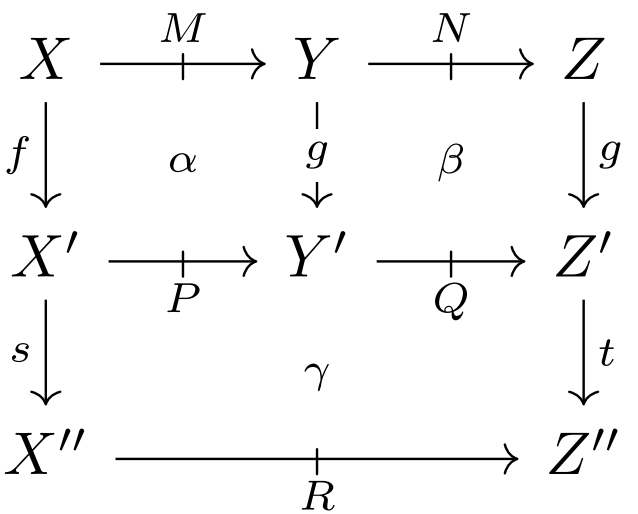
…is represented by the proterm

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

An internal logic of VDCs

Example

The virtual composite cell



…is represented by the proterm

$$x : X, y : Y, z : Z | m : M(x; y), n : N(y; z) \vdash$$
$$\gamma(\alpha(m), \beta(n)) : R(s(f(x)); t(g(z)))$$

An internal logic of VDCs

Restrictions

For this type theory to model the internal language of **fibrational** VDCs it needs to capture restriction cells and their universal property.

Given a proarrow $P : Y_0 \nrightarrow Y_1$ and tight morphisms $f_i : X_i \to Y_i$, the protype

$$x_0 : X_0; x_1 : X_1 \vdash P(f_0(x_0), f_1(x_1))$$

represents the restriction of P along f_0 and f_1.

An internal logic of VDCs

Restrictions

We recover the corresponding restriction cell

$$\begin{array}{ccc} X_0 & \xrightarrow{P(f_0,f_1)} & X_1 \\ {\scriptstyle f_0}\downarrow & \text{rest} & \downarrow{\scriptstyle f_1} \\ Y_0 & \xrightarrow[P]{} & Y_1 \end{array}$$

with the proterm

$$x_0 : X_0, x_1 : X_1 | p : P(f_0(x_0), f_1(x_1)) \vdash p : P(f_0(x_0), f_1(x_1)),$$

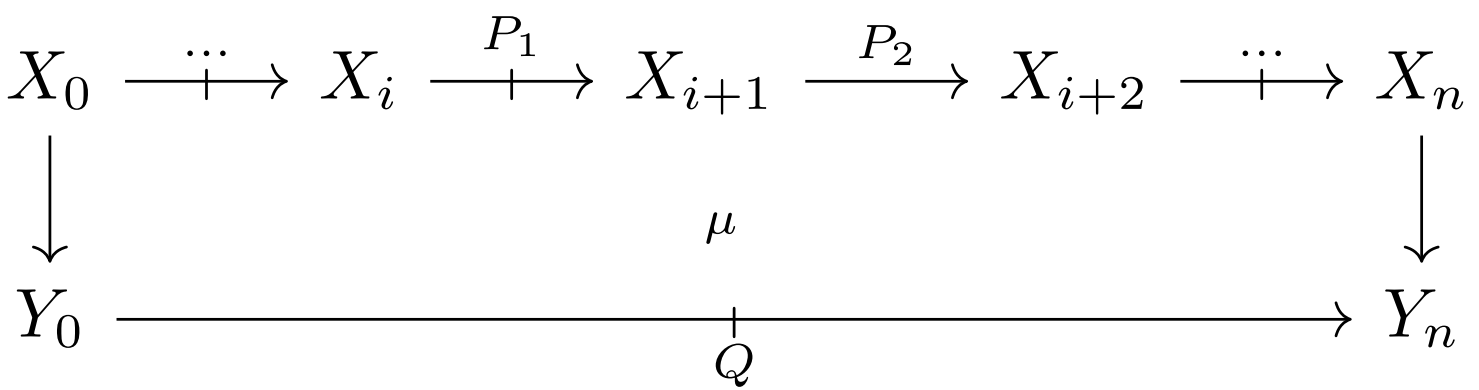
since the tight identity cell of a restriction proarrow corresponds uniquely with the restriction cell via the universal property.

An internal logic of VDCs

(Loose) Composition

The type theory can be extended with type formation, introduction, and elimination rules for loose composites.

The universal property for composites says that each cell μ induces a unique $\widetilde{\mu}$ such that

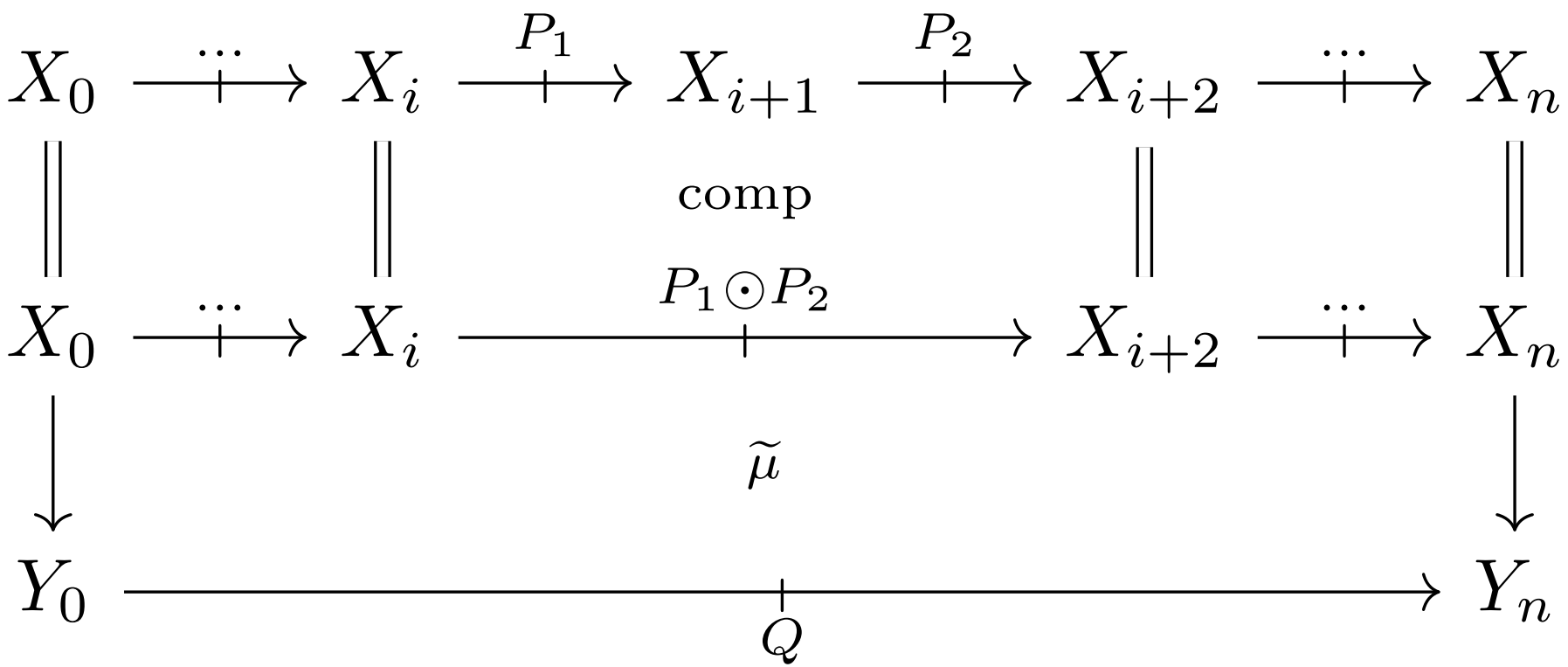


An internal logic of VDCs

(Loose) Composition

The type theory can be extended with type formation, introduction, and elimination rules for loose composites.

…is equal to the virtual composite



An internal logic of VDCs

(Loose) Composition

Like restriction, we encode the universal property in the internal language using a normal form.

Namely, the β/η-reduction rules (judgments relating a type's constructor(s) and eliminator(s)) for composites precisely capture this universal property.

An internal logic of VDCs

Extensions

Nasu's type theory offers a strong foundation, but it lacks some characteristics that are important to our use case:

1. **Rigid signature:** generator cells cannot map into a restriction or unit proarrow, for instance.
2. **Missing modalities:** virtual double monads are not supported by the type theory, which are essential for defining many important doctrines.
3. **Challenging to implement:** the type theory is not designed with algorithmic typechecking in mind.

ModalTT DSL

ModalTT

To address the problems outlined above, we have adapted Nasu's type theory in a number of ways:

1. Our signatures stratify VDC constructs into different syntactic categories to get a more expressive range of legal generators.
2. We introduce first class modality application into the syntax, without compromising the normal forms of (pro)terms.
3. Our type theory uses an ordered linear logic [5,6] to describe the resource-sensitivity and order-sensitivity of proterms. Loose composition reduces to the fuse operator.

ModalTT

DSL for theory specification

Users specify the generators (objects, tight/loose arrows, and cells) of their theory:

Objects

```
obj X
obj [Y, Z]
```

Arrows and proarrows

```
fun f : X -> Y
pro P : X => Y
```

Cells

```
cell α : [P, Q] => (List R) | f -> id Y
```

ModalTT

DSL for theory specification

Users can declare equalities of proarrows and of proterms (virtual cells), which define the axioms of the theory:

Proarrow Axioms

```
pro_axiom P := P ∗ P
```

Proterm Axioms (this is where the magic happens)

```
-- Promonad axioms
axiom [x0 : X, x1 : X] | [p : P[x0, x1]] |- θ (x0) ∗ p := p
axiom [x0 : X, x1 : X] | [p : P[x0, x1]] |- p := p ∗ θ (x1)

-- Can apply the list monad or use its (primitive) structure proterms
axiom [x : X, y : Y] | [p : (List P)[x, y]] |- μ List [(List (η List P)) [p]] := p

-- Loose composites are destructured via let binding (more on this soon)
axiom [x0 : X, x1 : X] | [...] |-
  let ([p : P[x0, x1], q : Q[x1, x2]] = α [...] ∗ β [...]) in γ[p, q] := ...
```

DEMO · MODALTT

(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™)

ModalTT Type Theory

ModalTT

Type theory

We adapt Nasu's type theory by introducing modalities and broadening the expressive power of the generator symbols.

To match the implementation, we also rely less heavily on substitution (which is badly behaved in the presence of non-syntactic equalities.)

ModalTT

Signature

The signature Σ determines the structures of a theory $\mathbb{T}_\Sigma$ by specifying finite sets of generators:

- Object symbols $\mathcal{T}_\Sigma$
- Arrow symbols $\mathcal{F}_\Sigma$
- Proarrow symbols $\mathcal{P}_\Sigma$
- Cell symbols $\mathcal{C}_\Sigma$
- Modality symbols $\mathcal{M}_\Sigma$

ModalTT

Signature

These generator symbols are closed under modal application (the sets of arrows and cells are also populated with the monad structure for each $L \in \mathcal{M}_\Sigma$).

The complete sets of arrows (resp. proarrows) are then generated from the modal closures by including identities (resp. loose units) and composites (resp. loose composites).

ModalTT

Typing Judgments

$$\begin{aligned}
\text{Type} &::= X \text{ type} \\
\text{Context} &::= \Gamma \text{ ctx} \\
\text{Term} &::= \Gamma \vdash s : X \\
\text{Protype} &::= \Gamma_0; \Gamma_1 \vdash P \text{ protype} \\
\text{Procontext} &::= \Gamma_0, \dots, \Gamma_n | \Omega \text{ proctx} \\
\text{Proterm} &::= \Gamma_0, \dots, \Gamma_n | \Omega \vdash \mu : P
\end{aligned}$$

ModalTT

Types

A type is (just) an object generated by the signature:

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

ModalTT

Term Contexts: Grammar

$$
\begin{aligned}
x &\in \mathrm{Var}_{\mathrm{Type}} \\
X &\in \widetilde{\mathcal{T}}_{\Sigma} \\
\Gamma &\in \mathrm{Context} ::= (x : X)
\end{aligned}
$$

Note: our theories are not necessarily cartesian, so our (term) contexts are unary [4], containing a single variable binding.

ModalTT

Term Contexts: Typing Judgment

$$\frac{X\ \text{type} \qquad x \in \text{Var}_{\text{Term}}}{(x : X)\ \text{ctx}}$$

ModalTT

Terms: Grammar

$$x \in \mathrm{Var}_{\mathrm{Term}}$$
$$f \in \widetilde{F}_{\Sigma}$$
$$s \in \mathrm{Term} ::= x \mid f(s)$$

ModalTT

Terms: Typing Judgments

$$\frac{}{(x : X) \vdash x : X}$$

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

ModalTT

Terms: Substitution

Terms are equipped with a meta-theoretic variable substitution, defined in the obvious way:

$$t[s/x] := \begin{cases} s & \text{if } t = x \\ f(t'[s/x]) & \text{if } t = f(t') \end{cases}$$

(As usual, we assume that x is not a variable in s to avoid variable capture.)

ModalTT

Protypes: Grammar

$$
\begin{aligned}
s_0, s_1 &\in \mathrm{Term} \\
P &\in \hat{\mathscr{P}}_\Sigma \\
\underline{P} &\in \mathrm{Protype} ::= P(s_0, s_1)
\end{aligned}
$$

ModalTT

Protypes: Typing Judgments

$$\frac{P \in \widetilde{\mathcal{P}}_\Sigma(X, Y) \qquad \Gamma_0 \vdash s_0 : X \qquad \Gamma_1 \vdash s_1 : Y}{\Gamma_0; \Gamma_1 \vdash P(s_0, s_1)\ \text{protype}}$$

$$\frac{\Gamma_0 \vdash s_0 : X \qquad \Gamma_1 \vdash s_1 : X}{\Gamma_0; \Gamma_1 \vdash \text{unit}_X(s_0, s_1)\ \text{protype}}$$

$$\frac{\Gamma_0; \Gamma_1 \vdash P(s_0, s_1)\ \text{protype} \qquad \Gamma_1; \Gamma_2 \vdash Q(s_1, s_2)\ \text{protype}}{\Gamma_0; \Gamma_2 \vdash (P \odot Q)(s_0, s_2)\ \text{protype}}$$

ModalTT

Procontexts: Grammar

For $n \geq 0$:

$$\begin{aligned}
\Gamma_i &\in \mathrm{Context} \\
\underline{P_i} &\in \mathrm{Protype} \\
p_i &\in \mathrm{Var}_{\mathrm{Proterm}} \\
\overline{\Gamma}|\Omega &\in \mathrm{Procontext} ::= \Gamma_0, \ldots, \Gamma_n | p_1 : \underline{P_1}, \ldots, p_n : \underline{P_n}
\end{aligned}$$

Note: both components of our procontexts are *ordered* (in the sense of ordered linear[1] logic [5,6]).

ModalTT

Procontexts: Typing Judgments

$$\frac{\Gamma \text{ ctx}}{\Gamma | \cdot \text{ proctx}}$$

$$\frac{\begin{array}{c}\Gamma_0, \ldots, \Gamma_n | p_1 : P_1(s_0, s_1), \ldots, p_n : P_n(s_{n-1}, s_n) \text{ proctx} \\ \Gamma_n ; \Gamma_{n+1} \vdash P_{n+1}(s_n, s_{n+1}) \text{ protype}\end{array}}{\Gamma_0, \ldots, \Gamma_{n+1} | p_1 : P_1(s_0, s_1), \ldots, p_{n+1} : P_{n+1}(s_n, s_{n+1}) \text{ proctx}}$$

ModalTT

Proterms: Grammar

$$
\begin{aligned}
p_i &\in \mathrm{Var}_{\mathrm{Proterm}} \\
f, g &\in \hat{\mathcal{F}}_\Sigma \\
s_i &\in \mathrm{Term} \\
\alpha &\in \hat{C}_\Sigma(P_1, \ldots, P_n \Rightarrow Q | f \to g) \\
\mu &\in \mathrm{Proterm} \\
&::= p \\
&\quad | \alpha\langle s_0, \ldots, s_n \rangle(\mu_1, \ldots, \mu_n) \\
&\quad | \mu_1 \odot \mu_2 \\
&\quad | \mathrm{let}\ [p_1 : \underline{P_1}, p_2 : \underline{P_2}] = \mu_1\ \mathrm{in}\ \mu_2
\end{aligned}
$$

ModalTT

Proterms: Typing Judgments

$$\frac{}{\overline{\Gamma}|p : \underline{P} \vdash p : \underline{P}}$$

$$\frac{\overline{\Gamma}_i|\Omega_i \vdash \mu_i : P_i(s_{i-1}, s_i) \qquad \alpha \in \widetilde{\mathcal{C}}_\Sigma(P_1, \ldots, P_n \Rightarrow Q|f \to g) \qquad i = 1, \ldots, n}{\overline{\Gamma}_1, \ldots, \overline{\Gamma}_n|\Omega_1, \ldots, \Omega_n \vdash \alpha(s_0, \ldots, s_n)\{\mu_1, \ldots, \mu_n\} : Q(f[\![s_0]\!], g[\![s_n]\!])}$$

ModalTT

Proterms: Typing Judgments

$$\frac{\overline{\Gamma}_1|\Omega_1 \vdash \mu_1 : P_1(s_0, s_1) \qquad \overline{\Gamma}_2|\Omega_2 \vdash \mu_2 : P_2(s_1, s_2)}{\overline{\Gamma}_1, \overline{\Gamma}_2|\Omega_1, \Omega_2 \vdash \mu_1 \odot \mu_2 : (P_1 \odot P_2)(s_0, s_2)}$$

$$\frac{\begin{array}{c}\overline{\Gamma}|\Omega \vdash \epsilon : (P_1 \odot P_2)(s_0, s_2) \\ \overline{\Gamma}_L, x_0 : X_0, x_1 : X_1, x_2 : X_2, \overline{\Gamma}_R|\Omega_L, p_1 : P_1(x_0, x_1), p_2 : P_2(x_1, x_2), \Omega_R \vdash \mu : Q(t_0, t_1)\end{array}}{\overline{\Gamma}_L, \overline{\Gamma}, \overline{\Gamma}_R|\Omega_L, \Omega, \Omega_R \vdash \text{let } (p_1 : P_1(x_0, x_1), p_2 : P_2(x_1, x_2) = \epsilon) \text{ in } \mu : Q(t_0[s_0/x_0], t_1[s_2/x_2])}$$

ModalTT

VDC Semantics

Note: the underlying VDC semantics of the type theory is still a work-in-progress.

We define a modal, fibrational VDC (with composites) $[\![\mathbb{T}_\Sigma]\!]$ as follows:

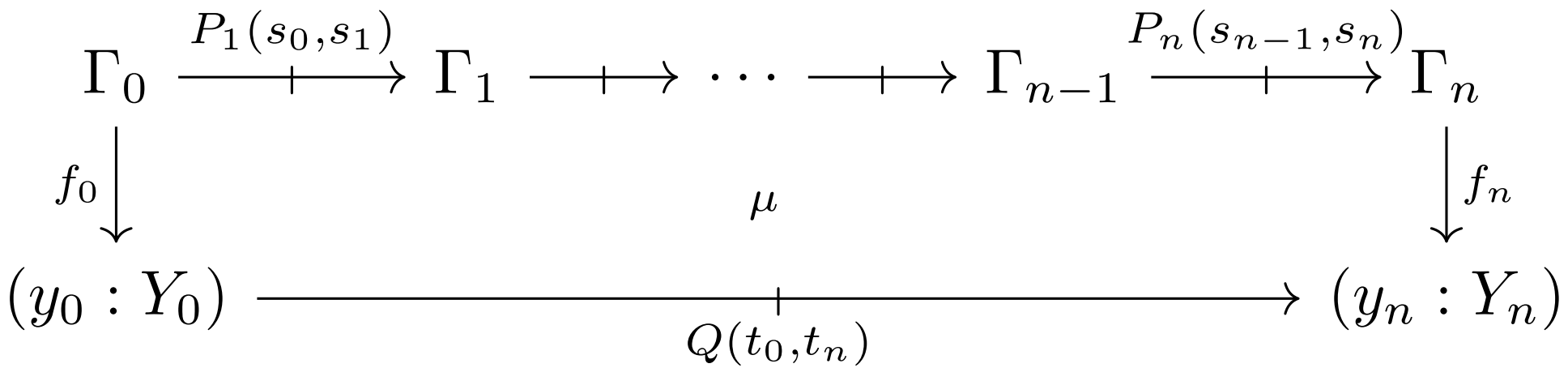
- Objects are term contexts $\Gamma = (x : X)$ (these are effectively just types annotated with a variable) modulo alpha equivalence of variables.
- Tight morphisms $(x : X) \to (y : Y)$ are terms $(x : X) \vdash s : Y$.
- Proarrows $(x : X) \nrightarrow (y : Y)$ are protypes

$$(x : X); (y : Y) \vdash P(s, t).$$

ModalTT

VDC Semantics

- Cells



are proterms

$$\Gamma_0, \dots, \Gamma_n | p_1 : P_1(s_0, s_1), \dots, p_n : P_n(s_{n-1}, s_n) \vdash$$
$$\mu : Q(t_0[f_0(s_0)/y_0], t_n[f_n(s_n)/y_n]).$$

ModalTT

VDC Semantics

- Restriction cells are obtained via term substitution:

$$x_0 : X_0, x_1 : X_1 | p : P(f_0(x_0), f_1(x_1)) \vdash p : P(f_0(x_0), f_1(x_1))$$

- Loose composition is precisely $\odot$-introduction on protypes and proterms, with the universal property supplied by $\odot$-elimination along with the β/η-computation rules (not shown).
- Each modality symbol $L \in \mathcal{M}_\Sigma$ corresponds to a monad on $[\![\mathbb{T}_\Sigma]\!]$ via the derived form $L[-]$ and the primitive structure morphisms/cells η and μ. Functoriality is immediate from the normal form.

Future work

- Typing judgments for loose units are subtle and not fully developed (hence their omission). As currently implemented, they lack a normal form.
- The expected metatheorems (cut elimination, soundness of the VDC semantics, etc.) need to be checked more carefully. The semantics themselves are also not yet fully complete.
- The surface syntax is cumbersome and not especially user-friendly.
- The internal representation still needs to be hooked up to CatColab so that the DSL can actually be used alongside DoubleTT models.

Thanks for listening!

References

1. Carlson K (2024) Introducing CatColab, 2024. Available from: https://topos.institute/blog/2024-10-02-introducing-catcolab/.
2. Patterson E CatColab: DoubleTT. Available from: https://next.catcolab.org/rfc/0002#mathematical-specification.
3. Nasu H (2025) An internal logic of virtual double categories.
4. Shulman M (2016) Categorical logic from a categorical point of view.
5. Petersen L, Harper R, Crary K, et al. (2003) A type theory for memory allocation and data layout, *Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on principles of programming languages*, New York, NY, USA, Association for Computing Machinery, 172–184.
6. Polakow J, Pfenning F (2001) Ordered linear logic and applications.
7. Lambert M, Patterson E (2024) Cartesian double theories: A double-categorical framework for categorical doctrines. *Advances in Mathematics* 444: 109630.