Language and substrates: Dependent type theory and topological spaces

David I. Spivak

TOPOS INSTITUTE

Grant FA9550-23-1-0376. Thanks to Fred Leve, Tristan Nguyen, & Doug Riecken

Outline

Working language for three funders

This is the second year of a grant that's funded by three AFOSR programs:

- Fred Leve: Dynamics and Control
- Doug Riecken: Computation and Learning, and
- Tristan Nguyen: Information Assurance.

Working language for three funders

This is the second year of a grant that's funded by three AFOSR programs:

- Fred Leve: Dynamics and Control
- Doug Riecken: Computation and Learning, and
- Tristan Nguyen: Information Assurance.

Originating question: What is *working language*? How does language *work*?

- Language works in the sense of basic physics: it directs energy.
- If I say "pass the salt," 10^{25} atoms move through space.
- This involves compositional planning and high-precision control.
- Getting it set up in the first place requires (evolution and) learning.
- DNA is working language: ATCG symbols code for chemistry.
- Computer programming languages do a lot of work in the world.

Working language for three funders

This is the second year of a grant that's funded by three AFOSR programs:

- Fred Leve: Dynamics and Control
- Doug Riecken: Computation and Learning, and
- Tristan Nguyen: Information Assurance.

Originating question: What is *working language*? How does language *work*?

- Language works in the sense of basic physics: it directs energy.
- If I say "pass the salt," 10^{25} atoms move through space.
- This involves compositional planning and high-precision control.
- Getting it set up in the first place requires (evolution and) learning.
- DNA is working language: ATCG symbols code for chemistry.
- Computer programming languages do a lot of work in the world.

Wanted: a minimally-assumptive mathematical framework to set up WL.

- Framing all this in strong/weak/gravity/EM forces, where's language?
- Want the structure and dynamics of language to be front and center.
- Can we relate matter and pattern using math rather than physics?

Working language for three funders

This is the second year of a grant that's funded by three AFOSR programs:

- Fred Leve: Dynamics and Control
- Doug Riecken: Computation and Learning, and
- Tristan Nguyen: Information Assurance.

Originating question: What is *working language*? How does language *work*?

- Language works in the sense of basic physics: it directs energy.
- If I say "pass the salt," 10^{25} atoms move through space.
- This involves compositional planning and high-precision control.
- Getting it set up in the first place requires (evolution and) learning.
- DNA is working language: ATCG symbols code for chemistry.
- Computer programming languages do a lot of work in the world.

Wanted: a minimally-assumptive mathematical framework to set up WL.

- Framing all this in strong/weak/gravity/EM forces, where's language?
- Want the structure and dynamics of language to be front and center.
- Can we relate matter and pattern using math rather than physics?

This isn't a CT audience, so I'll try to keep the talk self-contained.

Formal effects in material contexts

Aristotle had four notions of cause: material, formal, efficient, and final.

- My take: the two most fundamental are material and formal.
- Matter = the uncopiable stuff: Banana A$\neq$Banana B, share nothing.
- Form = the copiable "pattern": Banana A $\cong$ Banana B, same.
- If an aspect of this matter resonates in my brain-matter, it's copied.

Formal effects in material contexts

Aristotle had four notions of cause: material, formal, efficient, and final.

- My take: the two most fundamental are material and formal.
- Matter = the uncopiable stuff: Banana A≠Banana B, share nothing.
- Form = the copiable "pattern": Banana A ≅ Banana B, same.
- If an aspect of this matter resonates in my brain-matter, it's copied.

Working language: form somehow attaches to—resonates in—matter.

- The program runs the computer. The control loop runs the robot.
- The math "applies" to the world. Control theory works!
- Language works: a described effect is obtained in a material context.

Formal effects in material contexts

Aristotle had four notions of cause: material, formal, efficient, and final.

- My take: the two most fundamental are material and formal.
- Matter = the uncopiable stuff: Banana A≠Banana B, share nothing.
- Form = the copiable "pattern": Banana A ≅ Banana B, same.
- If an aspect of this matter resonates in my brain-matter, it's copied.

Working language: form somehow attaches to—resonates in—matter.

- The program runs the computer. The control loop runs the robot.
- The math "applies" to the world. Control theory works!
- Language works: a described effect is obtained in a material context.

In CT circles, effects and contexts are words for *monad* and *comonad*.

- Last year: the free monad $\mathfrak{m}$ is a module over the cofree comonad $\mathfrak{c}$...
- ...$\mathfrak{m}_p \otimes \mathfrak{c}_q \to \mathfrak{m}_{p \otimes q}$, and this grounds "pattern runs on matter".
- This year: further ground form as monadic and material as comonadic.

Plan

The plan for the remainder of this talk is:

- Recall basic category theory: cat'y, functor, transform, (co)monad.
- More on monads and progress toward dependent type theory
- More on comonads as generalized categories and spaces

Joint work with Corinthia Aberlé, Kevin Carlson, and Aaron D. Fairbanks.

Outline

Categories, functors, and natural transformations

The big 3 of category theory are: category, functor, natural transformation.

- Category = relational fabric. Functor = mapping. NT=trajectory.
- Example: $(\mathbb{N}, \leq)$, $(- \times 2) \colon (\mathbb{N}, \leq) \to (\mathbb{N}, \leq)$, $(- \times 2) \leq (- \times 3)$.

Categories, functors, and natural transformations

The big 3 of category theory are: category, functor, natural transformation.

- Category = relational fabric. Functor = mapping. NT=trajectory.
- Example: $(\mathbb{N}, \leq)$, $(- \times 2) \colon (\mathbb{N}, \leq) \to (\mathbb{N}, \leq)$, $(- \times 2) \leq (- \times 3)$.

A category $\mathcal{C}$ is a relational "fabric", like a space.

- It's got a set $\mathrm{Ob}(\mathcal{C})$, *objects* $\approx$ "points".
- For each $c, c' \in \mathrm{Ob}(\mathcal{C})$, a set $\mathrm{Mor}(c, c')$, *morphisms* $f : c \to c'$.
- Identities id_c and compositions $f \fatsemi g$ that are unital and assoc.

Categories, functors, and natural transformations

The big 3 of category theory are: category, functor, natural transformation.

- Category = relational fabric. Functor = mapping. NT=trajectory.
- Example: $(\mathbb{N}, \leq)$, $(- \times 2) \colon (\mathbb{N}, \leq) \to (\mathbb{N}, \leq)$, $(- \times 2) \leq (- \times 3)$.

A category $\mathcal{C}$ is a relational "fabric", like a space.

- It's got a set $\mathrm{Ob}(\mathcal{C})$, *objects* $\approx$ "points".
- For each $c, c' \in \mathrm{Ob}(\mathcal{C})$, a set $\mathrm{Mor}(c, c')$, *morphisms* $f : c \to c'$.
- Identities id_c and compositions $f \fatsemi g$ that are unital and assoc.

A functor $F : \mathcal{C} \to \mathcal{D}$ between categories is a "non-tearing" map.

- It sends each $c \in \mathrm{Ob}(\mathcal{C})$ to some $F(c) \in \mathrm{Ob}(\mathcal{D})$.
- It sends each morphism $f : c \to c'$ to some $F(f) \colon F(c) \to F(c')$.
- It preserves identities and composition, i.e. $F(f_1 \fatsemi f_2) = F(f_1) \fatsemi F(f_2)$.

Categories, functors, and natural transformations

The big 3 of category theory are: category, functor, natural transformation.

- Category = relational fabric. Functor = mapping. NT=trajectory.
- Example: $(\mathbb{N}, \leq)$, $(- \times 2) \colon (\mathbb{N}, \leq) \to (\mathbb{N}, \leq)$, $(- \times 2) \leq (- \times 3)$.

A category $\mathcal{C}$ is a relational "fabric", like a space.

- It's got a set $\mathrm{Ob}(\mathcal{C})$, *objects* $\approx$ "points".
- For each $c, c' \in \mathrm{Ob}(\mathcal{C})$, a set $\mathrm{Mor}(c, c')$, *morphisms* $f : c \to c'$.
- Identities id_c and compositions $f \fatsemi g$ that are unital and assoc.

A functor $F : \mathcal{C} \to \mathcal{D}$ between categories is a "non-tearing" map.

- It sends each $c \in \mathrm{Ob}(\mathcal{C})$ to some $F(c) \in \mathrm{Ob}(\mathcal{D})$.
- It sends each morphism $f : c \to c'$ to some $F(f) \colon F(c) \to F(c')$.
- It preserves identities and composition, i.e. $F(f_1 \fatsemi f_2) = F(f_1) \fatsemi F(f_2)$.

A natural transformation $\mathcal{C} \underset{G}{\overset{F}{\Downarrow \alpha}} \mathcal{D}$ is a $\mathcal{D}$-trajectory param'd by $\mathcal{C}$.

- For each $c \in \mathrm{Ob}(\mathcal{C})$, a morphism $\alpha_c \colon F(c) \to G(c)$ in $\mathcal{D}$.
- For each $f : c \to c'$ in $\mathcal{C}$, an equation $\alpha_c \fatsemi F(f) = F(f) \fatsemi \alpha_{c'}$.

Set, its endofunctors, and its monoidal structures

The big 4 would be the above, plus **Set**, the category of sets.

- The objects of **Set** are sets (in some mathematical universe of sets).
- A morphism $f : S \to T$ is a function, and composition is as usual.
- For any $N \in \mathbb{N}$, let $N := \{\text{`}1\text{`}, \ldots, \text{`}N\text{`}\}$ denote a set with N elements.
- Disjoint union $A + B$, Cartesian product $A \times B$, and exponential B^A.

Set, its endofunctors, and its monoidal structures

The big 4 would be the above, plus **Set**, the category of sets.

- The objects of **Set** are sets (in some mathematical universe of sets).
- A morphism $f : S \to T$ is a function, and composition is as usual.
- For any $N \in \mathbb{N}$, let $N := \{`1`, \ldots, `N`\}$ denote a set with N elements.
- Disjoint union $A + B$, Cartesian product $A \times B$, and exponential B^A.

Let's think about functors $F : \mathbf{Set} \to \mathbf{Set}$, "*endofunctors* on **Set**."

- F needs to send each set to a set and function to a function.
- How about $X \mapsto X^2$? Or $X \mapsto X + 1$? Or $X \mapsto 7$? Or $X \mapsto 2^X$?
- Endofunctors can be composed: $(F \circ G) : \mathbf{Set} \to \mathbf{Set}$.

Set, its endofunctors, and its monoidal structures

The big 4 would be the above, plus **Set**, the category of sets.

- The objects of **Set** are sets (in some mathematical universe of sets).
- A morphism $f : S \to T$ is a function, and composition is as usual.
- For any $N \in \mathbb{N}$, let $N := \{`1`, \ldots, `N`\}$ denote a set with N elements.
- Disjoint union $A + B$, Cartesian product $A \times B$, and exponential B^A.

Let's think about functors $F : \mathbf{Set} \to \mathbf{Set}$, "*endofunctors* on **Set**."

- F needs to send each set to a set and function to a function.
- How about $X \mapsto X^2$? Or $X \mapsto X + 1$? Or $X \mapsto 7$? Or $X \mapsto 2^X$?
- Endofunctors can be composed: $(F \circ G) : \mathbf{Set} \to \mathbf{Set}$.

Another category we'll be interested in today: topological spaces, i.e.: ...

- ...a set T and $\mathbf{Op}_T \subseteq 2^T$ closed under arbitrary union, finite inters'n.
- These $(T, \mathbf{Op}_T)$ are the objects. Morphisms are "continuous maps".
- We previously said categories were like spaces; in what sense? Soon....

Monoidal structures, monoids, and comonoids

A *monoidal structure* $(I, \odot)$ on $\mathcal{C}$ lets you combine things.

- Functors $I : 1 \to \mathcal{C}$ and $\odot : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ with $I \odot c \cong c \cong c \odot I$, etc.
- For example **Set** has $(1, \times)$ and infinitely-many others.
- Combine objects $A \times B$ or morphisms: given $A \xrightarrow{f} A'$ and $B \xrightarrow{g} B'$, get

$$(f \times g) : (A \times B) \to (A' \times B').$$

Monoidal structures, monoids, and comonoids

A *monoidal structure* $(I, \odot)$ on $\mathcal{C}$ lets you combine things.

- Functors $I : 1 \to \mathcal{C}$ and $\odot \colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ with $I \odot c \cong c \cong c \odot I$, etc.
- For example **Set** has $(1, \times)$ and infinitely-many others.
- Combine objects $A \times B$ or morphisms: given $A \xrightarrow{f} A'$ and $B \xrightarrow{g} B'$, get

$$(f \times g) \colon (A \times B) \to (A' \times B').$$

Given a cat'y $\mathcal{C}$ with a mon'l structure $(I, \odot)$, we can define (co)monoids.

- A *monoid* (m, η, μ)consists of an object $m \in \mathrm{Ob}(\mathcal{C})$,...
- ...and maps $I \xrightarrow{\eta} m$, $m \otimes m \xrightarrow{\mu} m$, satisfying unitality and associativity.

Monoidal structures, monoids, and comonoids

A *monoidal structure* $(I, \odot)$ on $\mathcal{C}$ lets you combine things.

- Functors $I : 1 \to \mathcal{C}$ and $\odot : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ with $I \odot c \cong c \cong c \odot I$, etc.
- For example **Set** has $(1, \times)$ and infinitely-many others.
- Combine objects $A \times B$ or morphisms: given $A \xrightarrow{f} A'$ and $B \xrightarrow{g} B'$, get

$$(f \times g) : (A \times B) \to (A' \times B').$$

Given a cat'y $\mathcal{C}$ with a mon'l structure $(I, \odot)$, we can define (co)monoids.

- A *monoid* (m, η, μ)consists of an object $m \in \mathrm{Ob}(\mathcal{C})$,...
- ...and maps $I \xrightarrow{\eta} m$, $m \otimes m \xrightarrow{\mu} m$, satisfying unitality and associativity.
- A *comonoid* (c, ϵ, δ) consists of an object $c \in \mathrm{Ob}(\mathcal{C})$,...
- ...and maps $c \xrightarrow{\epsilon} I$, $c \xrightarrow{\delta} c \otimes c$, satisfying counitality and coassoc'ity.

Monoidal structures, monoids, and comonoids

A *monoidal structure* $(I, \odot)$ on $\mathcal{C}$ lets you combine things.

- Functors $I : 1 \to \mathcal{C}$ and $\odot : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ with $I \odot c \cong c \cong c \odot I$, etc.
- For example **Set** has $(1, \times)$ and infinitely-many others.
- Combine objects $A \times B$ or morphisms: given $A \xrightarrow{f} A'$ and $B \xrightarrow{g} B'$, get

$$(f \times g) : (A \times B) \to (A' \times B').$$

Given a cat'y $\mathcal{C}$ with a mon'l structure $(I, \odot)$, we can define (co)monoids.

- A *monoid* (m, η, μ)consists of an object $m \in \mathrm{Ob}(\mathcal{C})$,...
- ...and maps $I \xrightarrow{\eta} m$, $m \otimes m \xrightarrow{\mu} m$, satisfying unitality and associativity.
- A *comonoid* (c, ϵ, δ) consists of an object $c \in \mathrm{Ob}(\mathcal{C})$,...
- ...and maps $c \xrightarrow{\epsilon} I$, $c \xrightarrow{\delta} c \otimes c$, satisfying counitality and coassoc'ity.

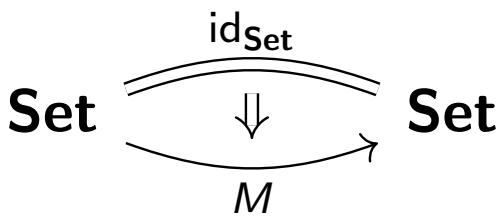
For any category $\mathcal{C}$, the cat'y $\mathrm{End}(\mathcal{C})$ of endofunctors on $\mathcal{C}$ is monoidal.

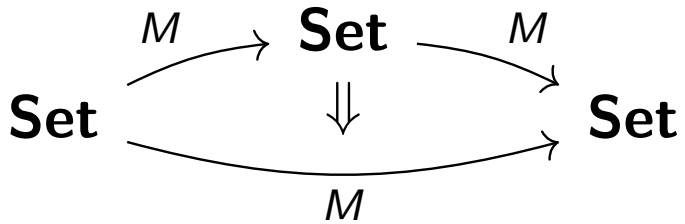
- Objects in $\mathrm{End}(\mathcal{C})$ are functors $F : \mathcal{C} \to \mathcal{C}$.
- Morphisms $\alpha : F \Rightarrow G$ are natural transformations.
- Monoidal unit is identity $\mathrm{id}_{\mathcal{C}}$; monoidal product is composition $F \circ G$.

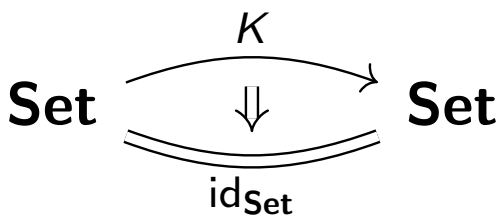
Monads and comonads

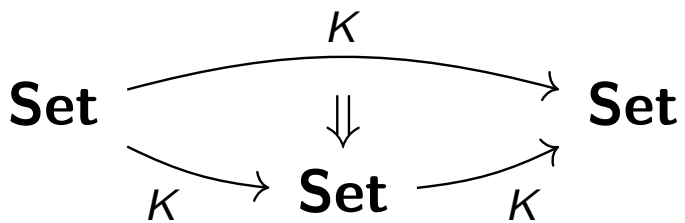
A *(co)monad* is: a (co)monoid in the monoidal category of endofunctors.

- You can see that small steps add up fast in CT.
- Monad is: a functor $M : \mathcal{C} \to \mathcal{C}$ and NTs $\mathrm{id}_{\mathcal{C}} \Rightarrow M$ and $M \circ M \Rightarrow M$.
- Comonad is: a functor $K : \mathcal{C} \to \mathcal{C}$ and NTs $K \Rightarrow \mathrm{id}_{\mathcal{C}}$ and $K \Rightarrow K \circ K$.
- We'll be using $\mathcal{C} = \mathbf{Set}$ all day.









- Each must satisfy the respective (co)unitality and (co)assocativity.

More on monads and comonads

In CT PL theory, monads are used for *effects*. Given $x \in X$ and $X \to M(Y)$,

- $M =$ "state monad" uses & updates some memory state to get a $y \in Y$.
- $M =$ "list monad" produces $N : \mathbb{N}$; needs $n < N$ to get a $y \in Y$.
- $M =$ "IO monad" prints output and reads input to get a $y \in Y$.

More on monads and comonads

In CT PL theory, monads are used for *effects*. Given $x \in X$ and $X \to M(Y)$,

- $M =$ "state monad" uses & updates some memory state to get a $y \in Y$.
- $M =$ "list monad" produces $N : \mathbb{N}$; needs $n < N$ to get a $y \in Y$.
- $M =$ "IO monad" prints output and reads input to get a $y \in Y$.

In CT PL th'y, comonads are used for *contexts*. A function $K(X) \to Y$...

- ...can't operate on just $x \in X$. It needs the "context".
- $K =$ "stream comonad": give me a whole stream $x : X^{\mathbb{N}}$ to get a $y \in Y$.
- $K = \sum_{T:\mathbb{N}} y^{\mathbb{N}_{\leq T}}$: give me a time step T and history $x_{\leq T}$ to get $y \in Y$.

More on monads and comonads

In CT PL theory, monads are used for *effects*. Given $x \in X$ and $X \to M(Y)$,

- M = "state monad" uses & updates some memory state to get a $y \in Y$.
- M = "list monad" produces $N : \mathbb{N}$; needs $n < N$ to get a $y \in Y$.
- M = "IO monad" prints output and reads input to get a $y \in Y$.

In CT PL th'y, comonads are used for *contexts*. A function $K(X) \to Y$...

- ...can't operate on just $x \in X$. It needs the "context".
- K = "stream comonad": give me a whole stream $x : X^{\mathbb{N}}$ to get a $y \in Y$.
- $K = \sum_{T:\mathbb{N}} y^{\mathbb{N}_{\leq T}}$: give me a time step T and history $x_{\leq T}$ to get $y \in Y$.

Last year: free monad $\mathfrak{m}$ as "program" and cofree comonad $\mathfrak{c}$ as "machine".

- The program affects the machine context: $\mathfrak{m}_p \otimes \mathfrak{c}_{[p,y]} \to y$

More on monads and comonads

In CT PL theory, monads are used for *effects*. Given $x \in X$ and $X \to M(Y)$,

- M = "state monad" uses & updates some memory state to get a $y \in Y$.
- M = "list monad" produces $N : \mathbb{N}$; needs $n < N$ to get a $y \in Y$.
- M = "IO monad" prints output and reads input to get a $y \in Y$.

In CT PL th'y, comonads are used for *contexts*. A function $K(X) \to Y$...

- ...can't operate on just $x \in X$. It needs the "context".
- K = "stream comonad": give me a whole stream $x : X^{\mathbb{N}}$ to get a $y \in Y$.
- $K = \sum_{T:\mathbb{N}} y^{\mathbb{N}_{\leq T}}$: give me a time step T and history $x_{\leq T}$ to get $y \in Y$.

Last year: free monad $\mathfrak{m}$ as "program" and cofree comonad $\mathfrak{c}$ as "machine".

- The program affects the machine context: $\mathfrak{m}_p \otimes \mathfrak{c}_{[p,y]} \to y$

This year: more on monad as formal language, comonad as mat'l substrate.

Outline

Algebraic theories

Monads in PL are effects, but they're older in math: algebraic theories.

- Semigroups, monoids, grps, rings, $\mathbb{R}$-vect. spcs, cpt Hausdorff spcs...
- Each of these caty's is M-**alg** for some monad $M\colon$ **Set** $\to$ **Set**
- An M-algebra is a set X and an action $M(X) \to X$, usual axioms.
- For example, $\mathbb{Z}[-]\colon$ **Set** $\to$ **Set** is the monad for commutative rings.
  - $\mathbb{Z}[x, y]$="polynomials in variables x, y", e.g. $x^4 + 3xy - 7$.
  - For any comm. ring R there's an "evaluation" function $\mathbb{Z}[R] \to R$
  - E.g. for any $r, s \in R$, evaluate formal $r^4 + 3rs - 7$ to an elt. of R.
  - And any X with action $\mathbb{Z}[X] \to X$ defines a commutative ring.
- Any map between monads lets you forget or freely add structure.

Algebraic theories

Monads in PL are effects, but they're older in math: algebraic theories.

- Semigroups, monoids, grps, rings, $\mathbb{R}$-vect. spcs, cpt Hausdorff spcs...
- Each of these caty's is M-**alg** for some monad $M\colon$ **Set** $\to$ **Set**
- An M-algebra is a set X and an action $M(X) \to X$, usual axioms.
- For example, $\mathbb{Z}[-]\colon$ **Set** $\to$ **Set** is the monad for commutative rings.
  - $\mathbb{Z}[x, y]$="polynomials in variables x, y", e.g. $x^4 + 3xy - 7$.
  - For any comm. ring R there's an "evaluation" function $\mathbb{Z}[R] \to R$
  - E.g. for any $r, s \in R$, evaluate formal $r^4 + 3rs - 7$ to an elt. of R.
  - And any X with action $\mathbb{Z}[X] \to X$ defines a commutative ring.
- Any map between monads lets you forget or freely add structure.

One way to *define* an algebraic theory is as a monad on **Set**.

- And replacing **Set** with other things just boosts the generality.
- For example 2-monads on **Cat** include "monoidal categories", etc.

A monad for dependent type theory

Dependent type theory is the formal language of mathematics.

- You can encode any math statement and any proof:
  - Prime number th'm, Gödel incompleteness, Kepler, Poincare, ...
  - Idea: "some $N : \mathbb{N}$ and for each $n < N$, some n-dim mfd X."
  - Later types can depend on values of previous types.
- You can write any computer program in this DTT. Very expressive.

A monad for dependent type theory

Dependent type theory is the formal language of mathematics.

- You can encode any math statement and any proof:
  - Prime number th'm, Gödel incompleteness, Kepler, Poincare, ...
  - Idea: "some $N : \mathbb{N}$ and for each $n < N$, some n-dim mfd X."
  - Later types can depend on values of previous types.
- You can write any computer program in this DTT. Very expressive.

The main symbols of this language are $1, \Sigma, \Pi$: one, sum, and product.

- Think 1="done", Σ_A="some $a : A$ and", and Π_A="per $a : A$,".
- $\sum_{N:\mathbb{N}} \prod_{n<N} \sum_{X:\mathbf{Mfd}_n} 1$: some N and per $n < N$, some X and done.
- Laws: $\sum_{i:1} A \cong A \cong \sum_{a:A} 1$, $\prod_{i:1} A \cong A$, $\prod_{a:A} 1 \cong 1$,...
- $\ldots \sum_{a:A} \sum_{b:B_a} C_{a,b} \cong \sum_{(a,b):\sum_{a:A} B_a} C_{a,b}$, $\prod_{a:A} \sum_{b:B_a} C_{a,b} \cong \sum_{\bar{b}:\prod_{a:A} B_a} \prod_{a:A} C_{a,\bar{b}(a)}$.

A monad for dependent type theory

Dependent type theory is the formal language of mathematics.

- You can encode any math statement and any proof:
  - Prime number th'm, Gödel incompleteness, Kepler, Poincare, ...
  - Idea: "some $N : \mathbb{N}$ and for each $n < N$, some n-dim mfd X."
  - Later types can depend on values of previous types.
- You can write any computer program in this DTT. Very expressive.

The main symbols of this language are $1, \Sigma, \Pi$: one, sum, and product.

- Think 1="done", Σ_A="some $a : A$ and", and Π_A="per $a : A$,".
- $\sum_{N:\mathbb{N}} \prod_{n<N} \sum_{X:\mathbf{Mfd}_n} 1$: some N and per $n < N$, some X and done.
- Laws: $\sum_{i:1} A \cong A \cong \sum_{a:A} 1$, $\prod_{i:1} A \cong A$, $\prod_{a:A} 1 \cong 1$,...
- $\ldots \sum_{a:A} \sum_{b:B_a} C_{a,b} \cong \sum_{(a,b):\sum_{a:A} B_a} C_{a,b}$, $\prod_{a:A} \sum_{b:B_a} C_{a,b} \cong \sum_{\bar{b}:\prod_{a:A} B_a} \prod_{a:A} C_{a,\bar{b}(a)}$.

We can encode all the laws using a monad and a self-distributive law.

- Joint work with CB Aberlé. Pick any set-theoretic *universe* $\mathbb{U}$.
- Take $u := \sum_{A:\mathbb{U}} y^{[A]}$. This is a monad 1: $\mathrm{id} \Rightarrow u$ and Σ: $u \circ u \Rightarrow u$
- A self-distributive law $u \circ u \Rightarrow u \circ u$ defines Π and all the laws.

A monad for dependent type theory

Dependent type theory is the formal language of mathematics.

- You can encode any math statement and any proof:
  - Prime number th'm, Gödel incompleteness, Kepler, Poincare, ...
  - Idea: "some $N : \mathbb{N}$ and for each $n < N$, some n-dim mfd X."
  - Later types can depend on values of previous types.
- You can write any computer program in this DTT. Very expressive.

The main symbols of this language are $1, \Sigma, \Pi$: one, sum, and product.

- Think 1="done", Σ_A="some $a : A$ and", and Π_A="per $a : A$,".
- $\sum_{N:\mathbb{N}} \prod_{n<N} \sum_{X:\mathbf{Mfd}_n} 1$: some N and per $n < N$, some X and done.
- Laws: $\sum_{i:1} A \cong A \cong \sum_{a:A} 1$, $\prod_{i:1} A \cong A$, $\prod_{a:A} 1 \cong 1$,...
- $\ldots \sum_{a:A} \sum_{b:B_a} C_{a,b} \cong \sum_{(a,b):\sum_{a:A} B_a} C_{a,b}$, $\prod_{a:A} \sum_{b:B_a} C_{a,b} \cong \sum_{\bar{b}:\prod_{a:A} B_a} \prod_{a:A} C_{a,\bar{b}(a)}$.

We can encode all the laws using a monad and a self-distributive law.

- Joint work with CB Aberlé. Pick any set-theoretic *universe* $\mathbb{U}$.
- Take $u := \sum_{A:\mathbb{U}} y^{[A]}$. This is a monad $1 : \mathrm{id} \Rightarrow u$ and $\Sigma : u \circ u \Rightarrow u$
- A self-distributive law $u \circ u \Rightarrow u \circ u$ defines Π and all the laws.

This "self-distributivity" structure is a very concise description of DTT.

Outline

Categories as comonads on Set

A comonad c on set is a pretty simple-to-state thing:

- It's a comonoid (c, ϵ, δ) in the category **Fun**(**Set**, **Set**), i.e....
  - For each set X, you give me a set $c(X)$.
  - For each fn. $f : X \to Y$, you give me a fn. $c(f) : c(X) \to c(Y)$.
  - It preserves identity and composition $c(f \circ g) = c(f) \circ c(g)$.
  - For each X, you give me fns $c(X) \xrightarrow{\epsilon_X} X$ and $c(X) \xrightarrow{\delta_X} c(c(X))$
- But what does all this structure do? Why should anyone care?

Categories as comonads on Set

A comonad c on set is a pretty simple-to-state thing:

- It's a comonoid (c, ϵ, δ) in the category **Fun**(**Set**, **Set**), i.e....
  - For each set X, you give me a set $c(X)$.
  - For each fn. $f : X \to Y$, you give me a fn. $c(f) : c(X) \to c(Y)$.
  - It preserves identity and composition $c(f \circ g) = c(f) \circ c(g)$.
  - For each X, you give me fns $c(X) \xrightarrow{\epsilon_X} X$ and $c(X) \xrightarrow{\delta_X} c(c(X))$
- But what does all this structure do? Why should anyone care?

It turns out that every category $\mathcal{C}$ can be described as one.

- For each $C \in \mathrm{Ob}\,\mathcal{C}$, let $\mathcal{C}[C] := \sum_{C' : \mathrm{Ob}\,\mathcal{C}} \mathrm{Hom}(C, C')$, "maps out".
- Let $c(X) := \sum_{C \in \mathrm{Ob}\,\mathcal{C}} X^{\mathcal{C}[C]}$. "Polynomial comonad."
- There's an equivalence of categories between **Cat** and...
- ...polynomial comonads on **Set** with "continuous" maps between them
- Functors $\mathcal{C} \to$ **Set** are exactly c-coalgebras $X \to c(X)$.

Categories as comonads on Set

A comonad c on set is a pretty simple-to-state thing:

- It's a comonoid (c, ϵ, δ) in the category **Fun**(**Set**, **Set**), i.e....
  - For each set X, you give me a set $c(X)$.
  - For each fn. $f : X \to Y$, you give me a fn. $c(f) : c(X) \to c(Y)$.
  - It preserves identity and composition $c(f \circ g) = c(f) \circ c(g)$.
  - For each X, you give me fns $c(X) \xrightarrow{\epsilon_X} X$ and $c(X) \xrightarrow{\delta_X} c(c(X))$
- But what does all this structure do? Why should anyone care?

It turns out that every category $\mathcal{C}$ can be described as one.

- For each $C \in \mathrm{Ob}\,\mathcal{C}$, let $\mathcal{C}[C] := \sum_{C' : \mathrm{Ob}\,\mathcal{C}} \mathrm{Hom}(C, C')$, "maps out".
- Let $c(X) := \sum_{C \in \mathrm{Ob}\,\mathcal{C}} X^{\mathcal{C}[C]}$. "Polynomial comonad."
- There's an equivalence of categories between **Cat** and...
- ...polynomial comonads on **Set** with "continuous" maps between them
- Functors $\mathcal{C} \to$ **Set** are exactly c-coalgebras $X \to c(X)$.

How general are comonads on **Set**, if we drop the "polynomial" restriction?

How general are comonads on Set?

It turns out that comonads on **Set** are very general, including all

- "Grothendieck toposes equipped with a separating set of points".
  - This is a corollary of a result of Garner.
  - E.g. the cat'y **Top** of topological spaces T embeds fully faithfully.
  - And we've seen that the category **Cat** of categories $\mathcal{C}$ does too.
  - Just like $\mathcal{C}$-coalg is $\mathcal{C}$-set, T-coalg will be sheaves on T.
- There are far more, e.g. the category of partitioned sets is $\mathbb{P}_\star$-coalg.

How general are comonads on Set?

It turns out that comonads on **Set** are very general, including all

- "Grothendieck toposes equipped with a separating set of points".
  - This is a corollary of a result of Garner.
  - E.g. the cat'y **Top** of topological spaces T embeds fully faithfully.
  - And we've seen that the category **Cat** of categories $\mathcal{C}$ does too.
  - Just like $\mathcal{C}$-coalg is $\mathcal{C}$-set, T-coalg will be sheaves on T.
- There are far more, e.g. the category of partitioned sets is $\mathbb{P}_\star$-coalg.

Next up, intuition, but here let's just explain the case of topological spaces.

- Let $(T, \mathbf{Op}_T)$ be a top'l space. Define a functor $c_T : \mathbf{Set} \to \mathbf{Set}$ by

$$c_T(X) := \sum_{t:T} \operatorname*{colim}_{t \in U \in \mathbf{Op}_T} X^U.$$

  - E.g. $T = \mathbb{R}$. At each $t \in \mathbb{R}$, sections $U \xrightarrow{f} X$ over shrinking ϵ-nbhds.
  - Counit $c_T(X) \to X$, evaluate any such function at the point t.

How general are comonads on Set?

It turns out that comonads on **Set** are very general, including all

- "Grothendieck toposes equipped with a separating set of points".
  - This is a corollary of a result of Garner.
  - E.g. the cat'y **Top** of topological spaces T embeds fully faithfully.
  - And we've seen that the category **Cat** of categories $\mathcal{C}$ does too.
  - Just like $\mathcal{C}$-coalg is $\mathcal{C}$-set, T-coalg will be sheaves on T.
- There are far more, e.g. the category of partitioned sets is $\mathbb{P}_\star$-coalg.

Next up, intuition, but here let's just explain the case of topological spaces.

- Let $(T, \mathbf{Op}_T)$ be a top'l space. Define a functor $c_T : \mathbf{Set} \to \mathbf{Set}$ by

$$c_T(X) := \sum_{t:T} \operatorname*{colim}_{t \in U \in \mathbf{Op}_T} X^U.$$

  - E.g. $T = \mathbb{R}$. At each $t \in \mathbb{R}$, sections $U \xrightarrow{f} X$ over shrinking ϵ-nbhds.
  - Counit $c_T(X) \to X$, evaluate any such function at the point t.
- A coalg. $X \xrightarrow{h} c_T(X)$ is equiv'y a T-sheaf $\mathcal{O}$ with X the set of $\mathcal{O}$-stalks
  - To each $x : X$, assigns $h(x) = (t, f)$ with $t \in T$ and $f \in \mathcal{O}_t$ a germ.

Topological intuition for comonads on Set

Intuition: comonads generalize "points and neighborhood systems".

- In a space $(T, \mathbf{Op}_T)$, each point t has nbhd sys: open sets $t \in U \subseteq U'$.
- In a cat'y $\mathcal{C}$, each object has one nbhd: all maps out .
- $\mathbb{P}_\star$ has one point and nbhd system of pointed sets and surj'ns $U \twoheadrightarrow U'$.

Topological intuition for comonads on Set

Intuition: comonads generalize "points and neighborhood systems".

- In a space $(T, \mathbf{Op}_T)$, each point t has nbhd sys: open sets $t \in U \subseteq U'$.
- In a cat'y $\mathcal{C}$, each object has one nbhd: all maps out .
- $\mathbb{P}_\star$ has one point and nbhd system of pointed sets and surj'ns $U \twoheadrightarrow U'$.

How does it work formally? Let $c\colon \mathbf{Set} \to \mathbf{Set}$ be an arbitrary comonad.

- Points: $c(1) : \mathbf{Set}$. Each $C \in c(1)$ assigned a connected cat'y $\mathcal{N}_C$.
- Now the comonad can be written $\sum_{C:c(1)} \mathrm{colim}_{U:\mathcal{N}_c(C)} y^U$.
- A c-coalgebra $h\colon X \to c(X)$ puts every element $x \in X$ at a point C,...
- ...and tells you an elt x_u, for all u in some small enough nbhd U of C.
- Thus we can think of h as a "sheaf" and X as its set of all "stalks".

Topological intuition for comonads on Set

Intuition: comonads generalize "points and neighborhood systems".

- In a space $(T, \mathbf{Op}_T)$, each point t has nbhd sys: open sets $t \in U \subseteq U'$.
- In a cat'y $\mathcal{C}$, each object has one nbhd: all maps out .
- $\mathbb{P}_\star$ has one point and nbhd system of pointed sets and surj'ns $U \twoheadrightarrow U'$.

How does it work formally? Let $c\colon \mathbf{Set} \to \mathbf{Set}$ be an arbitrary comonad.

- Points: $c(1) : \mathbf{Set}$. Each $C \in c(1)$ assigned a connected cat'y $\mathcal{N}_C$.
- Now the comonad can be written $\sum_{C:c(1)} \mathrm{colim}_{U:\mathcal{N}_c(C)} y^U$.
- A c-coalgebra $h\colon X \to c(X)$ puts every element $x \in X$ at a point C,...
- ...and tells you an elt x_u, for all u in some small enough nbhd U of C.
- Thus we can think of h as a "sheaf" and X as its set of all "stalks".

Example: how a partitioned set $(X, \sim)$ gives a coalgebra $h\colon X \to \mathbb{P}_\star(X)$?

- Each x is assigned a pointed subset of X; namely $\{x' \mid x \sim x'\}$.
- The counit $\epsilon_X : \mathbb{P}_\star(X) \to X$ sends $(U, x) \mapsto x$.
- The comult $\delta_X : \mathbb{P}_\star(X) \to \mathbb{P}_\star(\mathbb{P}_\star(X))$ sends it to $(\{(U,x') | x' \in U\}, (U,x))$.
- So a "sheaf" h on the one-point "space" $\mathbb{P}_\star$ is a partitioned set...
- ... and its "sections" are parts.

Outline

Summary

This grant is for studying the structure and dynamics of working language.

- What minimally-assumptive math'l foundation supports language?
- Understand the dynamics of how language affects substrate.

Summary

This grant is for studying the structure and dynamics of working language.

- What minimally-assumptive math'l foundation supports language?
- Understand the dynamics of how language affects substrate.

Monads and comonads on **Set**: low description length & highly expressive.

- Monads are about effects, patterns, control, language.
- Comonads are about contexts, matter, space, substrate.
- Last time: pattern runs on matter $= \mathfrak{m}_p \otimes \mathfrak{c}_q \to \mathfrak{m}_{p \otimes q}$ "free monad is cofree comonad module"

Summary

This grant is for studying the structure and dynamics of working language.

- What minimally-assumptive math'l foundation supports language?
- Understand the dynamics of how language affects substrate.

Monads and comonads on **Set**: low description length & highly expressive.

- Monads are about effects, patterns, control, language.
- Comonads are about contexts, matter, space, substrate.
- Last time: pattern runs on matter $= \mathfrak{m}_p \otimes \mathfrak{c}_q \to \mathfrak{m}_{p \otimes q}$ "free monad is cofree comonad module"

Today: dependent type theory as a monad and top'l spaces as comonads.

- For any universe of sets $\mathbb{U}$, a monad u giving dep. sum Σ and ...
- ...a distributive law $u \circ u \to u \circ u$ giving dependent product Π.
- For any generalized space, a comonad whose coalgebras are its sheaves.

Summary

This grant is for studying the structure and dynamics of working language.

- What minimally-assumptive math'l foundation supports language?
- Understand the dynamics of how language affects substrate.

Monads and comonads on **Set**: low description length & highly expressive.

- Monads are about effects, patterns, control, language.
- Comonads are about contexts, matter, space, substrate.
- Last time: pattern runs on matter $= \mathfrak{m}_p \otimes \mathfrak{c}_q \to \mathfrak{m}_{p \otimes q}$ "free monad is cofree comonad module"

Today: dependent type theory as a monad and top'l spaces as comonads.

- For any universe of sets $\mathbb{U}$, a monad u giving dep. sum Σ and ...
- ...a distributive law $u \circ u \to u \circ u$ giving dependent product Π.
- For any generalized space, a comonad whose coalgebras are its sheaves.

Language and substrate are mathematically dual, long "before" physics.

Thanks! Comments and questions welcome...