Structure and Dynamics of Working Language

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 first 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 first 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.

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: ACGT symbols code for chemistry.
- Computer programming languages do a lot of work in the world.

Working language for three funders

This is the first 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.

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: ACGT 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?

$\mathbb{C}$-like miracles in Poly

In 2019 I was seeking a fr'work for mode-dependent dynamics and interac'n.

- When your eyes are open vs. closed, the input datatype is different.
- Communication channels can change based on what is said over them.
- The category **Poly** fit very well, covering all the examples.

$\mathbb{C}$-like miracles in Poly

In 2019 I was seeking a fr'work for mode-dependent dynamics and interac'n.

- When your eyes are open vs. closed, the input datatype is different.
- Communication channels can change based on what is said over them.
- The category **Poly** fit very well, covering all the examples.

But then I became enamored with **Poly** based on certain "miracles".

- A "miracle" of $\mathbb{C}$: add roots of $x^2 + 1$, get differentiable $\implies$ analytic.
- A "miracle" of **Poly**: its comonoids are precisely categories.
- The work I'd done a decade earlier on data migration was subsumed.
- Automata, dependent types, dynamical systems, PL, all foregrounded.

$\mathbb{C}$-like miracles in Poly

In 2019 I was seeking a fr'work for mode-dependent dynamics and interac'n.

- When your eyes are open vs. closed, the input datatype is different.
- Communication channels can change based on what is said over them.
- The category **Poly** fit very well, covering all the examples.

But then I became enamored with **Poly** based on certain "miracles".

- A "miracle" of $\mathbb{C}$: add roots of $x^2 + 1$, get differentiable $\implies$ analytic.
- A "miracle" of **Poly**: its comonoids are precisely categories.
- The work I'd done a decade earlier on data migration was subsumed.
- Automata, dependent types, dynamical systems, PL, all foregrounded.

I became optimistic that **Poly** could be a unified framework for my research.

- It has potential to reveal deep insights about the nature of comput'n.
- Perhaps it would foreground something about how language works?
- Today: the way terminating scripts run on persistent machines...
- ...is foregrounded via free monads $\mathfrak{m}_p$ and cofree comonads $\mathfrak{c}_q$.

Plan

The plan for the rest of my talk is as follows:

- Being year 1, a review of basic category theory
- A review of polynomial functors
- Discuss new result: a module structure $\mathfrak{m}_p \otimes \mathfrak{c}_q \to \mathfrak{m}_{p \otimes q}$

This is joint work with Sophie Libkind.

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* $f\colon c \to c'$.
- Identities id_c and compositions $f \mathbin{⨟} g$ 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* $f\colon c \to c'$.
- Identities id_c and compositions $f \mathbin{;} g$ that are unital and assoc.

A functor $F\colon \mathcal{C} \to \mathcal{D}$ 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 $f\colon c \to c'$ to some $F(f)\colon F(c) \to F(c')$.
- It preserves identities and composition, i.e. $F(f_1 \mathbin{;} f_2) = F(f_1) \mathbin{;} F(f_2)$.

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* $f \colon c \to c'$.
- Identities id_c and compositions $f \mathbin{;} g$ that are unital and assoc.

A functor $F \colon \mathcal{C} \to \mathcal{D}$ 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 $f \colon c \to c'$ to some $F(f) \colon F(c) \to F(c')$.
- It preserves identities and composition, i.e. $F(f_1 \mathbin{;} f_2) = F(f_1) \mathbin{;} F(f_2)$.

A natural transformation $\begin{tikzcd} \mathcal{C} \arrow[r, bend left, "F"] \arrow[r, bend right, "G"'] \arrow[r, phantom, "\Downarrow\alpha"] & \mathcal{D} \end{tikzcd}$ 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 $f \colon c \to c'$ in $\mathcal{C}$, an equation $\alpha_c \mathbin{;} F(f) = F(f) \mathbin{;} \alpha_{c'}$.

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 $f\colon S \to T$ 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.

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 $f\colon S \to T$ 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 $F\colon \mathbf{Set} \to \mathbf{Set}$.

- 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 $X \mapsto 2^X$?
- Functors F, G can be added or multiplied, pointwise: $F + G$, $F \times G$.

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 $f\colon S \to T$ 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 $F\colon \mathbf{Set} \to \mathbf{Set}$.

- 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 $X \mapsto 2^X$?
- Functors F, G can be added or multiplied, pointwise: $F + G$, $F \times G$.

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

- For example **Set** has $(0, +)$, $(1, \times)$, and infinitely-many others.
- Given $f\colon A \to A'$ and $g\colon B \to B'$, get

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

- So **Set** is a *distributive category*.

Monoids and comonoids

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.

Monoids and comonoids

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.

What are these for **Set**?

- A $(1, \times)$-monoid is a usual monoid; every set is uniq'ly a $(0, +)$-monoid
- The only $(0, +)$-comonoid is 0.
- Every set is uniquely a $(1, \times)$-comonoid.

Monoids and comonoids

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.

What are these for **Set**?

- A $(1, \times)$-monoid is a usual monoid; every set is uniq'ly a $(0, +)$-monoid
- The only $(0, +)$-comonoid is 0.
- Every set is uniquely a $(1, \times)$-comonoid.

Comonoids were discovered late b/c they aren't interesting in caty's like **Set**

- In **Vect** any choice of basis gives comonoid: $V \to V \otimes V$ and $V \to k$.
- We'll see that they're very interesting in other categories too.

Monads and comonads

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

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

Monads and comonads

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

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

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

- You can see that small steps add up fast in CT.
- A monad is: a functor $F\colon \mathcal{C} \to \mathcal{C}$ and NTs $\mathrm{id}_\mathcal{C} \Rightarrow F$ and $F \circ F \Rightarrow F$.
- A comonad is: a functor $F\colon \mathcal{C} \to \mathcal{C}$ and NTs $F \Rightarrow \mathrm{id}_\mathcal{C}$ and $F \Rightarrow F \circ F$.
- Each must satisfy the respective (co)unitality and (co)assocativity.

Monads and comonads

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

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

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

- You can see that small steps add up fast in CT.
- A monad is: a functor $F\colon \mathcal{C} \to \mathcal{C}$ and NTs $\mathrm{id}_{\mathcal{C}} \Rightarrow F$ and $F \circ F \Rightarrow F$.
- A comonad is: a functor $F\colon \mathcal{C} \to \mathcal{C}$ and NTs $F \Rightarrow \mathrm{id}_{\mathcal{C}}$ and $F \Rightarrow F \circ F$.
- Each must satisfy the respective (co)unitality and (co)assocativity.

These come up throughout math and functional programming. Examples:

- For each alg. theory (e.g. groups), there's an associated monad on **Set**.
- Monads capture *effects* (IO, non-det'sm, exceptions) in functional PL.
- Our goal is to discuss free monads and cofree comonads.
- These'll model terminating programs and persistent machines resp.

Outline

Polynomial functors

Polynomial functors **Set** $\to$ **Set** are the closure of the identity under $\sum, \prod$.

- Let y denote $\mathrm{id}_{\mathbf{Set}}$. Then $y^A = \prod_{a \in A} y$ sends $X \mapsto X^A$, e.g. $y^0 = 1$.
- A polynomial functor is $\sum_{i \in I} \prod_{j \in J_i} y$. E.g. $y^{\mathbb{N}} + \mathbb{R}y^2 + 17$.
- We call each $i \in I$ a *position* and each $j \in J_i$ a *direction* at i.
- Maps between polynomial functors are natural transformations.
- The category **Poly** is nice because *calculation is easy*!
- It has infinitely many monoidal structures. We'll look at two.

Polynomial functors

Polynomial functors **Set** $\to$ **Set** are the closure of the identity under $\sum, \prod$.

- Let y denote $\mathrm{id}_{\mathbf{Set}}$. Then $y^A = \prod_{a \in A} y$ sends $X \mapsto X^A$, e.g. $y^0 = 1$.
- A polynomial functor is $\sum_{i \in I} \prod_{j \in J_i} y$. E.g. $y^{\mathbb{N}} + \mathbb{R}y^2 + 17$.
- We call each $i \in I$ a *position* and each $j \in J_i$ a *direction* at i.
- Maps between polynomial functors are natural transformations.
- The category **Poly** is nice because *calculation is easy*!
- It has infinitely many monoidal structures. We'll look at two.

Polynomials can be composed, and this is a monoidal product denoted $\circ$.

- Example: $y^2 \circ (y+1) = y^2 + 2y + 1$ and $(y+1) \circ y^2 = y^2 + 1$.
- Monoidal: given maps $p \to q$ and $p' \to q'$, get $p \circ p' \to q \circ q'$
- Example (applic'n as subst'n): $p(42) = p \circ 42y^0$.
- An (A, B)-Moore machine is a poly map $S \to By^A \circ S = B \times S^A$.

Polynomial functors

Polynomial functors **Set** $\to$ **Set** are the closure of the identity under $\sum, \prod$.

- Let y denote $\mathrm{id}_{\mathbf{Set}}$. Then $y^A = \prod_{a \in A} y$ sends $X \mapsto X^A$, e.g. $y^0 = 1$.
- A polynomial functor is $\sum_{i \in I} \prod_{j \in J_i} y$. E.g. $y^{\mathbb{N}} + \mathbb{R}y^2 + 17$.
- We call each $i \in I$ a *position* and each $j \in J_i$ a *direction* at i.
- Maps between polynomial functors are natural transformations.
- The category **Poly** is nice because *calculation is easy*!
- It has infinitely many monoidal structures. We'll look at two.

Polynomials can be composed, and this is a monoidal product denoted $\circ$.

- Example: $y^2 \circ (y + 1) = y^2 + 2y + 1$ and $(y + 1) \circ y^2 = y^2 + 1$.
- Monoidal: given maps $p \to q$ and $p' \to q'$, get $p \circ p' \to q \circ q'$
- Example (applic'n as subst'n): $p(42) = p \circ 42y^0$.
- An (A, B)-Moore machine is a poly map $S \to By^A \circ S = B \times S^A$.

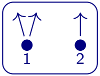
Polynomials can be tensored (Dirichlet product) $p \otimes q$, e.g. $y^3 \otimes y^3 = y^9$.

- We can use this to wire together Moore machines in block diagrams.

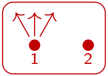
More on substitution product

We can draw polynomials as corolla forests. Substitution is stacking.

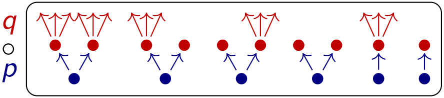
$p = y^2 + y$



$q = y^3 + 1$



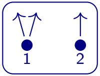
$p \circ q = y^6 + 3y^3 + 2$

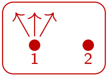


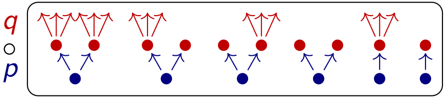
More on substitution product

We can draw polynomials as corolla forests. Substitution is stacking.

$p = y^2 + y$ $\quad$ $q = y^3 + 1$ $\qquad$ $p \circ q = y^6 + 3y^3 + 2$





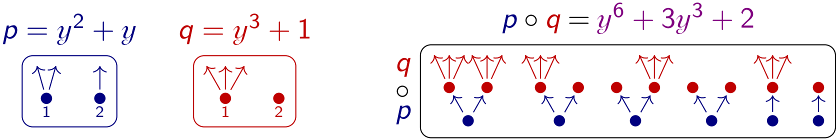


An element of $p \circ p \circ \cdots \circ p$ can be thought of as a flow-chart:

- The "questions" are positions of p; the "options" are the directions.

More on substitution product

We can draw polynomials as corolla forests. Substitution is stacking.



An element of $p \circ p \circ \cdots \circ p$ can be thought of as a flow-chart:

- The "questions" are positions of p; the "options" are the directions.

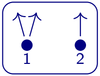
One of the miracles of **Poly** is that $\circ$-comonoids are exactly categories!

- A comonoid includes a polynomial c and maps $c \xrightarrow{\epsilon} y$ and $c \xrightarrow{\delta} c \circ c$...
- ...satisfying counit. and coassoc. Isn't it shocking that these = caty's?
- Idea: Ob's = positions; Mor's=directions; Id's=ϵ; Cod's & Comps=δ.

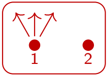
More on substitution product

We can draw polynomials as corolla forests. Substitution is stacking.

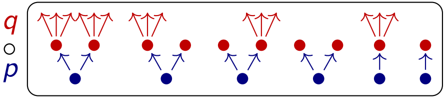
$p = y^2 + y$



$q = y^3 + 1$



$p \circ q = y^6 + 3y^3 + 2$



An element of $p \circ p \circ \cdots \circ p$ can be thought of as a flow-chart:

- The "questions" are positions of p; the "options" are the directions.

One of the miracles of **Poly** is that $\circ$-comonoids are exactly categories!

- A comonoid includes a polynomial c and maps $c \xrightarrow{\epsilon} y$ and $c \xrightarrow{\delta} c \circ c$...
- ...satisfying counit. and coassoc. Isn't it shocking that these = caty's?
- Idea: Ob's = positions; Mor's=directions; Id's=ϵ; Cod's & Comps=δ.

Another is that $\circ$-monoids are (very close to) "operads".

- An *operad* $\mathcal{O}$ is a "system of operations".
- For every arity N, a set $\mathcal{O}_N$, and how they compose.

Free monad monad and cofree comonad comonad

CT tries to foreground the most general abstractions from across mathematics.

- Monad and comonad are some of the most important concepts in CT.
- Free (like free group) is ubiquitous across math; cofree is the dual notion.
- And module (e.g. vector space, group action) is also ubiquitous.
- "The free monad monad is a module over the cofree comonad comonad"
- ...would suggest itself as a statement with theoretical significance.
- What we need to show is that it means something about working language.

Free monad monad and cofree comonad comonad

CT tries to foreground the most general abstractions from across mathematics.

- Monad and comonad are some of the most important concepts in CT.
- Free (like free group) is ubiquitous across math; cofree is the dual notion.
- And module (e.g. vector space, group action) is also ubiquitous.
- "The free monad monad is a module over the cofree comonad comonad"
- ...would suggest itself as a statement with theoretical significance.
- What we need to show is that it means something about working language.

First, let's say what the free monad and cofree comonad constructions are.

- Given a polynomial functor p, we have a monad $\mathfrak{m}_p$ and a comonad $\mathfrak{c}_p$.

$$\mathfrak{m}_p := \operatorname{colim}\left(\cdots \longleftarrow y+p\circ(y+p\circ(y+p)) \longleftarrow y+p\circ(y+p) \longleftarrow y+p \longleftarrow y\right)$$
$$\mathfrak{c}_p := \lim\left(\cdots \longrightarrow y\times p\circ(y\times p\circ(y\times p)) \longrightarrow y\times p\circ(y\times p) \longrightarrow y\times p \longrightarrow y\right)$$

- In fact $\mathfrak{m}_- : \mathbf{Poly} \to \mathbf{Poly}$ is itself a monad and $\mathfrak{c}_-$ is a comonad.
- Hence we refer to $\mathfrak{m}_-$ (resp. $\mathfrak{c}_-$) as the *(co)free (co)monad (co)monad*.

Free monad monad and cofree comonad comonad

CT tries to foreground the most general abstractions from across mathematics.

- Monad and comonad are some of the most important concepts in CT.
- Free (like free group) is ubiquitous across math; cofree is the dual notion.
- And module (e.g. vector space, group action) is also ubiquitous.
- "The free monad monad is a module over the cofree comonad comonad"
- ...would suggest itself as a statement with theoretical significance.
- What we need to show is that it means something about working language.

First, let's say what the free monad and cofree comonad constructions are.

- Given a polynomial functor p, we have a monad $\mathfrak{m}_p$ and a comonad $\mathfrak{c}_p$.

$$\mathfrak{m}_p := \operatorname{colim}\left(\cdots \longleftarrow y+p \circ (y+p \circ (y+p)) \longleftarrow y+p \circ (y+p) \longleftarrow y+p \longleftarrow y \right)$$

$$\mathfrak{c}_p := \lim\left(\cdots \longrightarrow y \times p \circ (y \times p \circ (y \times p)) \longrightarrow y \times p \circ (y \times p) \longrightarrow y \times p \longrightarrow y \right)$$

- In fact $\mathfrak{m}_- : \mathbf{Poly} \to \mathbf{Poly}$ is itself a monad and $\mathfrak{c}_-$ is a comonad.
- Hence we refer to $\mathfrak{m}_-$ (resp. $\mathfrak{c}_-$) as the *(co)free (co)monad (co)monad*.

Theorem: There is a natural map exhibiting $\mathfrak{m}_-$ as a module over $\mathfrak{c}_-$:

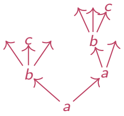
$$\mathfrak{m}_p \otimes \mathfrak{c}_q \to \mathfrak{m}_{p \otimes q}$$

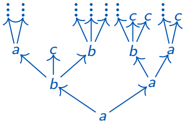
How it works

Both $\mathfrak{m}_p$ and $\mathfrak{c}_p$ are carried by poly'ls; what are their pos'ns and direc'ns?

- First let's define a *p-tree* to be a rooted tree, where each node is...
- ...labeled by a position $P \in p(1)$, and has $p[P]$-many branches.
- Each position in $\mathfrak{m}_p$ and $\mathfrak{c}_p$ can be represented by a p-tree.
  - In $\mathfrak{m}_p$, each tree is *well-founded*: always a finite path down to root
  - In $\mathfrak{c}_p$, they are generally infinite: only stops if it has no branches.

$p \coloneqq \{a\}y^2 + \{b\}y^3 + \{c\}$



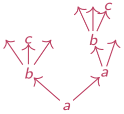


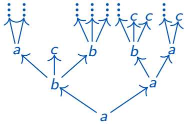
How it works

Both $\mathfrak{m}_p$ and $\mathfrak{c}_p$ are carried by poly'ls; what are their pos'ns and direc'ns?

- First let's define a *p-tree* to be a rooted tree, where each node is...
- ...labeled by a position $P \in p(1)$, and has $p[P]$-many branches.
- Each position in $\mathfrak{m}_p$ and $\mathfrak{c}_p$ can be represented by a p-tree.
  - In $\mathfrak{m}_p$, each tree is *well-founded*: always a finite path down to root
  - In $\mathfrak{c}_p$, they are generally infinite: only stops if it has no branches.

$p := \{a\}y^2 + \{b\}y^3 + \{c\}$





How do we think about the module structure $\mathfrak{m}_p \otimes \mathfrak{c}_q \to \mathfrak{m}_{p\otimes q}$?

- Think of $T \in \mathfrak{m}_p(1)$ as a terminating program, or a finite flowchart.
- Think of $U \in \mathfrak{c}_q(1)$ as a machine or operating system, running forever.
- We can lay T next to U and move forward through both in tandem.

We can use this to run programs that interact with a server/operating system.

- E.g. compose $\mathfrak{m}_p \otimes \mathfrak{c}_q \to \mathfrak{m}_{p\otimes q} \xrightarrow{\mathfrak{m}_\varphi} \mathfrak{m}_y \to y$ for some $p \otimes q \xrightarrow{\varphi} y$.
- This way, the program interacts with (controls) the machine.

Outline

Summary

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

- What minimally-assumptive math'l foundation supports language?
- Understand the structure and dynamics of everything involved.

Summary

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

- What minimally-assumptive math'l foundation supports language?
- Understand the structure and dynamics of everything involved.

Category theory is the language of structures and their relationships.

- In particular, **Poly** has tons of structure and many surprises.
- We're optimistic it's good ground for considering this question.

Summary

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

- What minimally-assumptive math'l foundation supports language?
- Understand the structure and dynamics of everything involved.

Category theory is the language of structures and their relationships.

- In particular, **Poly** has tons of structure and many surprises.
- We're optimistic it's good ground for considering this question.

Today: language as a relationship between program and machine.

- Programs terminate, machines persist; programs "run on" machines,...
- ...modeled via the (co)free (co)monad (co)monad module structure:

$$\mathfrak{m}_p \otimes \mathfrak{c}_q \to \mathfrak{m}_{p\otimes q}.$$

- CT's conciseness here suggests that this is a fundamental relationship.

Thanks! Comments and questions welcome...