Pattern runs on matter: The free monad monad as a module over the cofree comonad comonad

Sophie Libkind and **David I. Spivak**

TOPOS INSTITUTE

ACT 2024
2024 June 17

Outline

What's going on here?

What are we? Let's just talk naively about this.

- In some sense we're material: chemical processes formed as bodies.
- But on this material run little scripts: beliefs, habits, know-how, etc.
- Same goes for our computers: they're material running scripts.
- Same goes for our cells: genes are protein-production scripts.

What's going on here?

What are we? Let's just talk naively about this.

- In some sense we're material: chemical processes formed as bodies.
- But on this material run little scripts: beliefs, habits, know-how, etc.
- Same goes for our computers: they're material running scripts.
- Same goes for our cells: genes are protein-production scripts.

Whether "matter" and "pattern" work as names here is up for debate.

- This talk is about a pretty and straightforward math idea:
- Free monads differ from—but interact with—cofree comonads.
- It's about how to both intuit this formally and see its usefulness.

Summary of the talk

Here are four examples of what Sophie and I call "pattern runs on matter":

- Interviews run on people;
- Programs run on operating systems;
- Voting schemes run on voters;
- Games run on players.

Summary of the talk

Here are four examples of what Sophie and I call “pattern runs on matter”:

- Interviews run on people;
- Programs run on operating systems;
- Voting schemes run on voters;
- Games run on players.

We explain them in our paper; the main point is the module structure

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

where $\mathfrak{c}_p$ is the cofree comonad on p and $\mathfrak{m}_q$ is the free monad on q.

Outline

Polynomial functors

I love **Poly** because it hits a sweet spot of elementary, expressive, elegant.

- It's *elementary* in that it's just well-organized sets and functions.
- It's *expressive* in that it spans databases, dynamics, and programming.
- It's *elegant* in that it has tons of structure and delightful surprises.

Polynomial functors

I love **Poly** because it hits a sweet spot of elementary, expressive, elegant.

- It's *elementary* in that it's just well-organized sets and functions.
- It's *expressive* in that it spans databases, dynamics, and programming.
- It's *elegant* in that it has tons of structure and delightful surprises.

A functor $p\colon \mathbf{Set} \to \mathbf{Set}$ is polynomial if (TFAE):

- It's a coproduct of representables $p \cong \sum_{i \in I} \mathbf{Set}(A_i, -) = \sum_{i \in I} \mathcal{y}^{A_i}$.
- It preserves connected limits (e.g. pullbacks, equalizers, filtered limits).

A map $\varphi\colon p \to q$ between polynomials is (TFAE):

- A natural transformation $p \to q$. Yoneda and coproduct UP give the equivalence.
- An element of the set $\prod_{i \in I} \sum_{j \in J} \prod_{b \in B_j} \sum_{a \in A_i} 1$.

Polynomial functors

I love **Poly** because it hits a sweet spot of elementary, expressive, elegant.

- It's *elementary* in that it's just well-organized sets and functions.
- It's *expressive* in that it spans databases, dynamics, and programming.
- It's *elegant* in that it has tons of structure and delightful surprises.

A functor $p\colon \mathbf{Set} \to \mathbf{Set}$ is polynomial if (TFAE):

- It's a coproduct of representables $p \cong \sum_{i\in I} \mathbf{Set}(A_i, -) = \sum_{i\in I} y^{A_i}$.
- It preserves connected limits (e.g. pullbacks, equalizers, filtered limits).

A map $\varphi\colon p \to q$ between polynomials is (TFAE):

- A natural transformation $p \to q$. Yoneda and coproduct UP give the equivalence.
- An element of the set $\prod_{i\in I} \sum_{j\in J} \prod_{b\in B_j} \sum_{a\in A_i} 1$.

The category **Poly** of polynomial functors has tons of structure. Today:

- It has coproducts, and products that distribute over them.
- There's another distributive monoidal closed structure $(y, \otimes, [-,-])$.
- The latter is duoidal with a fourth monoidal structure $(y, \triangleleft)$:

$$(p_1 \triangleleft p_2) \otimes (q_1 \triangleleft q_2) \longrightarrow (p_1 \otimes q_1) \triangleleft (p_2 \otimes q_2)$$

Moore & Mealy machines, and wiring diagrams

Machines of type (A, B) input lists of A's and produce lists of B's

- We start with a set S, elements of which are called *states*.
- A *Moore machine* is a function $S \to B \times S^A$.
- A *Mealy machine* is a function $S \to (B \times S)^A$.

More gen'ly, for any polynomial p, a p-machine is a p-coalgebra $S \to p(S)$.

- As **Poly** has left Kan extensions, this can be identified with $Sy^S \to p$.
- When $p = By^A$ these give Moore; when $p = B^A y^A$ these give Mealy.

Moore & Mealy machines, and wiring diagrams

Machines of type (A, B) input lists of A's and produce lists of B's

- We start with a set S, elements of which are called *states*.
- A *Moore machine* is a function $S \to B \times S^A$.
- A *Mealy machine* is a function $S \to (B \times S)^A$.

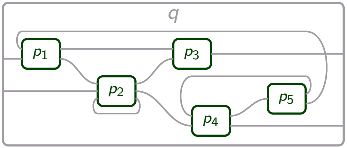
More gen'ly, for any polynomial p, a p-machine is a p-coalgebra $S \to p(S)$.

- As **Poly** has left Kan extensions, this can be identified with $Sy^S \to p$.
- When $p = By^A$ these give Moore; when $p = B^A y^A$ these give Mealy.

Wiring diagrams depict maps in **Poly**.

- Right, we see $\varphi \colon p_1 \otimes \cdots \otimes p_5 \to q$
- The $\otimes$ is a monoidal structure.
  - It's "Day convolution of $\times$".
  - It's got an easy formula in **Poly**.

$\varphi =$



Moore & Mealy machines, and wiring diagrams

Machines of type (A, B) input lists of A's and produce lists of B's

- We start with a set S, elements of which are called *states*.
- A *Moore machine* is a function $S \to B \times S^A$.
- A *Mealy machine* is a function $S \to (B \times S)^A$.

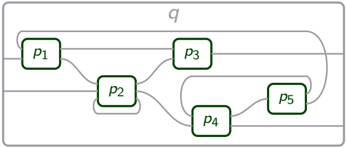
More gen'ly, for any polynomial p, a p-machine is a p-coalgebra $S \to p(S)$.

- As **Poly** has left Kan extensions, this can be identified with $Sy^S \to p$.
- When $p = By^A$ these give Moore; when $p = B^A y^A$ these give Mealy.

Wiring diagrams depict maps in **Poly**.

- Right, we see $\varphi\colon p_1 \otimes \cdots \otimes p_5 \to q$
- The $\otimes$ is a monoidal structure.
  - It's "Day convolution of $\times$".
  - It's got an easy formula in **Poly**.

$\varphi =$



It turns out that $\otimes$ has a closure $[-,-]$.

- Mealy machines are the "universal other" (dual) of Moore machines.
- Ask me about this afterwards, but basically $[Ay^B, y] \cong B^A y^A$.

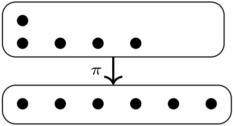
Polynomial functors and trees

There are three nice ways to denote a polynomial.

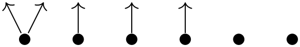
Algebraic

$y^2 + 3y + 2$

Bundle



Corolla forest



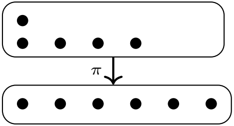
Polynomial functors and trees

There are three nice ways to denote a polynomial.

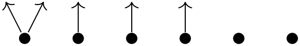
Algebraic

$y^2 + 3y + 2$

Bundle



Corolla forest



Some terminology:

- There are $p(1) = 6$ dots on the bottom; we call these *positions*.
- Each pos'n $P : p(1)$ has a fiber $p[P]$; call its elements *directions*.

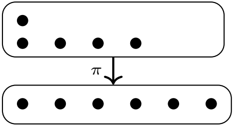
Polynomial functors and trees

There are three nice ways to denote a polynomial.

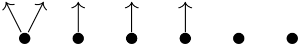
Algebraic

$y^2 + 3y + 2$

Bundle



Corolla forest



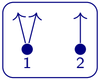
Some terminology:

- There are $p(1) = 6$ dots on the bottom; we call these *positions*.
- Each pos'n $P : p(1)$ has a fiber $p[P]$; call its elements *directions*.

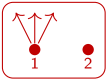
The composite of polynomial functors is again polynomial.

- We denote $p \circ q$ by $p \triangleleft q$, for various reasons.
- We can draw $p \triangleleft q$ by grafting q-corollas on top of p-corollas.

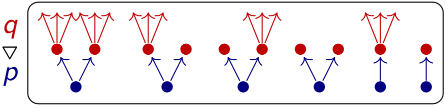
$p = y^2 + y$



$q = y^3 + 1$



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



Outline

Monads and comonads

A $(y, \triangleleft)$-monoid structure on $m\colon \mathbf{Set} \to \mathbf{Set}$ consists of coherent maps

$$\eta\colon y \to m \qquad \text{and} \qquad \mu\colon m \triangleleft m \to m$$

And a $(y, \triangleleft)$-comonoid structure on $c\colon \mathbf{Set} \to \mathbf{Set}$ consists of coh'nt maps

$$\epsilon\colon c \to y \qquad \text{and} \qquad \delta\colon c \to c \triangleleft c$$

Monads and comonads

A $(y, \triangleleft)$-monoid structure on $m \colon \mathbf{Set} \to \mathbf{Set}$ consists of coherent maps

$$\eta \colon y \to m \qquad \text{and} \qquad \mu \colon m \triangleleft m \to m$$

And a $(y, \triangleleft)$-comonoid structure on $c \colon \mathbf{Set} \to \mathbf{Set}$ consists of coh'nt maps

$$\epsilon \colon c \to y \qquad \text{and} \qquad \delta \colon c \to c \triangleleft c$$

Since $\triangleleft$ is functor composition, these are in fact *polynomial (co)monads*.

- One can think of a polynomial monad m as a variant of an *operad*.
  - We'll be interested in free monads, "flowchart languages".
- And a polynomial comonad c is precisely the same as a *category*.
  - We will be interested in cofree comonads, "machines".

The free monad $\mathfrak{m}_p$

We can build the free monad $\mathfrak{m}_p$ on a polynomial p by induction. Define:

$$p_{(0)} \coloneqq y \qquad \text{and} \qquad p_{(i+1)} \coloneqq y + p \triangleleft p_{(i)}$$

Let's define $\varphi_{(i)} \colon p_{(i)} \to p_{(i+1)}$ inductively.

$$y \xrightarrow{\varphi_{(0)} \coloneqq \mathsf{inc}} y + p \qquad \text{and} \qquad y + p \triangleleft p_{(i)} \xrightarrow{\varphi_{(i+1)} \coloneqq y + p \triangleleft \varphi_{(i)}} y + p \triangleleft p_{(i+1)}$$

The free monad $\mathfrak{m}_p$

We can build the free monad $\mathfrak{m}_p$ on a polynomial p by induction. Define:

$$p_{(0)} \coloneqq y \qquad \text{and} \qquad p_{(i+1)} \coloneqq y + p \triangleleft p_{(i)}$$

Let's define $\varphi_{(i)} \colon p_{(i)} \to p_{(i+1)}$ inductively.

$$y \xrightarrow{\varphi_{(0)} \coloneqq \text{inc}} y + p \qquad \text{and} \qquad y + p \triangleleft p_{(i)} \xrightarrow{\varphi_{(i+1)} \coloneqq y + p \triangleleft \varphi_{(i)}} y + p \triangleleft p_{(i+1)}$$

Let $p_{(\omega)} \coloneqq \operatorname{colim}_{i<\omega} p_{(i)} = \operatorname{colim}(y \xrightarrow{\varphi_{(0)}} y+p \xrightarrow{\varphi_{(1)}} y+p \triangleleft (y+p) \to \cdots)$.

- When p is finitary (all exponents are finite), we have $\mathfrak{m}_p = p_{(\omega)}$.
- When p is κ-small, you need more directed colimits along the way...
- ...but there's nothing at all complicated here: $\mathfrak{m}_p = \operatorname{colim}_{i<\kappa} p_{(i)}$.
- The map $\eta \colon y \to \mathfrak{m}_p$ is obvious, and the map $\mu \colon \mathfrak{m}_p \triangleleft \mathfrak{m}_p \to \mathfrak{m}_p$ ...
- involves induction and the interplay between directed colimits and $\triangleleft$.

The cofree comonad $\mathfrak{c}_p$

We can also build the cofree comonad $\mathfrak{c}_p$ on p by induction. Define:

$$p^{(0)} := y \qquad \text{and} \qquad p^{(i+1)} := y \times p \triangleleft p^{(i)}$$

Let's define $\varphi^{(i)} \colon p^{(i+1)} \to p^{(i)}$ inductively.

$$y \times p \xrightarrow{\varphi^{(0)} := \mathsf{prj}} y \qquad \text{and} \qquad y \times p \triangleleft p^{(i+1)} \xrightarrow{\varphi^{(i+1)} := y \times p \triangleleft \varphi^{(i)}} y \times p \triangleleft p^{(i)}$$

The cofree comonad $\mathfrak{c}_p$

We can also build the cofree comonad $\mathfrak{c}_p$ on p by induction. Define:

$$p^{(0)} := y \qquad \text{and} \qquad p^{(i+1)} := y \times p \triangleleft p^{(i)}$$

Let's define $\varphi^{(i)} \colon p^{(i+1)} \to p^{(i)}$ inductively.

$$y \times p \xrightarrow{\varphi^{(0)} := \mathsf{prj}} y \qquad \text{and} \qquad y \times p \triangleleft p^{(i+1)} \xrightarrow{\varphi^{(i+1)} := y \times p \triangleleft \varphi^{(i)}} y \times p \triangleleft p^{(i)}$$

Let $\mathfrak{c}_p := \lim(\cdots \to y \times p \triangleleft (y \times p) \xrightarrow{\varphi^{(1)}} y \times p \xrightarrow{\varphi^{(0)}} y)$.

- Unlike $\mathfrak{m}$, one can stop here, building $\mathfrak{c}$ doesn't need higher ordinals.
- The map $\epsilon \colon \mathfrak{c}_p \to y$ is obvious and the map $\delta \colon \mathfrak{c}_p \to \mathfrak{c}_p \triangleleft \mathfrak{c}_p$...
- ...involves induction and the interplay between directed limits and $\triangleleft$.

The cofree comonad $\mathfrak{c}_p$

We can also build the cofree comonad $\mathfrak{c}_p$ on p by induction. Define:

$$p^{(0)} := y \qquad \text{and} \qquad p^{(i+1)} := y \times p \triangleleft p^{(i)}$$

Let's define $\varphi^{(i)} \colon p^{(i+1)} \to p^{(i)}$ inductively.

$$y \times p \xrightarrow{\varphi^{(0)}:=\mathrm{prj}} y \qquad \text{and} \qquad y \times p \triangleleft p^{(i+1)} \xrightarrow{\varphi^{(i+1)}:=y \times p \triangleleft \varphi^{(i)}} y \times p \triangleleft p^{(i)}$$

Let $\mathfrak{c}_p := \lim(\cdots \to y \times p \triangleleft (y \times p) \xrightarrow{\varphi^{(1)}} y \times p \xrightarrow{\varphi^{(0)}} y)$.

- Unlike $\mathfrak{m}$, one can stop here, building $\mathfrak{c}$ doesn't need higher ordinals.
- The map $\epsilon \colon \mathfrak{c}_p \to y$ is obvious and the map $\delta \colon \mathfrak{c}_p \to \mathfrak{c}_p \triangleleft \mathfrak{c}_p$...
- ...involves induction and the interplay between directed limits and $\triangleleft$.

Remember p-machines, e.g. Mealy $p = (Ay)^B$, and Moore $p = Ay^B$?

- A position of $\mathfrak{c}_p$ is an initialized p-machine, up to behav'l equivalence.

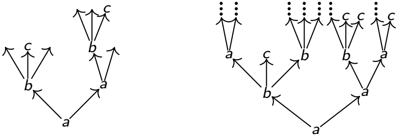
So how similar are the free monad $\mathfrak{m}_p$ and the cofree comonad $\mathfrak{c}_p$?

Tree representation of $\mathfrak{m}_p$ and $\mathfrak{c}_p$

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 $P : p(1)$, 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.

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

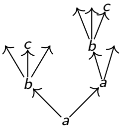


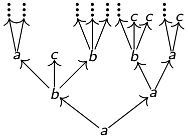
Tree representation of $\mathfrak{m}_p$ and $\mathfrak{c}_p$

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 $P : p(1)$, 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.

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





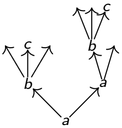
- The directions at a p-tree are very different in $\mathfrak{m}_p$ vs. $\mathfrak{c}_p$.
  - In $\mathfrak{m}_p$, the set of directions at a p-tree is its set of leaves.
  - In $\mathfrak{c}_p$, the set of directions at a p-tree is its set of nodes.

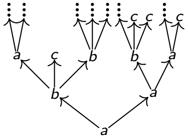
Tree representation of $\mathfrak{m}_p$ and $\mathfrak{c}_p$

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 $P : p(1)$, 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.

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





- The directions at a p-tree are very different in $\mathfrak{m}_p$ vs. $\mathfrak{c}_p$.
  - In $\mathfrak{m}_p$, the set of directions at a p-tree is its set of leaves.
  - In $\mathfrak{c}_p$, the set of directions at a p-tree is its set of nodes.

They're so similar, yet uncannily diff't! What else do we know about them?

Monad monad & Comonad comonad

The “free monad” functor $p \mapsto \mathfrak{m}_p$ is a monad **Poly** $\xrightarrow{\mathfrak{m}_-}$ **Poly**.

- There are maps $p \xrightarrow{\eta} \mathfrak{m}_p$ and $\mathfrak{m}_{\mathfrak{m}_p} \xrightarrow{\mu} \mathfrak{m}_p$ that obey the usual eqns.
- So we could call $\mathfrak{m}_-$ the *free monad monad*.
- Any polynomial monad m is an algebra of this monad, $\mathfrak{m}_m \to m$.

Monad monad & Comonad comonad

The “free monad” functor $p \mapsto \mathfrak{m}_p$ is a monad $\textbf{Poly} \xrightarrow{\mathfrak{m}_-} \textbf{Poly}$.

- There are maps $p \xrightarrow{\eta} \mathfrak{m}_p$ and $\mathfrak{m}_{\mathfrak{m}_p} \xrightarrow{\mu} \mathfrak{m}_p$ that obey the usual eqns.
- So we could call $\mathfrak{m}_-$ the *free monad monad*.
- Any polynomial monad m is an algebra of this monad, $\mathfrak{m}_m \to m$.

And the “cofree comonad” functor $p \mapsto \mathfrak{c}_p$ is a comonad $\textbf{Poly} \xrightarrow{\mathfrak{c}_-} \textbf{Poly}$.

- There are maps $\mathfrak{c}_p \xrightarrow{\epsilon} p$ and $\mathfrak{c}_p \xrightarrow{\delta} \mathfrak{c}_{\mathfrak{c}_p}$ that obey the usual eqns.
- So we could call $\mathfrak{c}_-$ the *cofree comonad comonad*.
- Any polynomial comonad c is a coalgebra of this monad, $c \to \mathfrak{c}_c$.

Interactions between $\mathfrak{m}_{-}$ and $\mathfrak{c}_{-}$

There are various interactions amongst free monads and cofree comonads.

- (Turi-Plotkin) "Oper'l semantics" is a distrib. law $\mathfrak{m}_p \triangleleft \mathfrak{c}_p \to \mathfrak{c}_p \triangleleft \mathfrak{m}_p$.
- The cofree comonad $\mathfrak{c}_{-}$ is lax monoidal, $y \to \mathfrak{c}_y$ and $\mathfrak{c}_p \otimes \mathfrak{c}_{p'} \to \mathfrak{c}_{p \otimes p'}$.
- The free monad $\mathfrak{m}_{-}$ is *not* lax mon'l for $\otimes$ (though it is for $+$ and $\vee$).

---

[1] The notion of module here comes from nlab, "module over a monoidal functor". The module structure Φ is similar to a result of Katsumata-Rivas-Uustalu.

Interactions between $\mathfrak{m}_-$ and $\mathfrak{c}_-$

There are various interactions amongst free monads and cofree comonads.

- (Turi-Plotkin) "Oper'l semantics" is a distrib. law $\mathfrak{m}_p \triangleleft \mathfrak{c}_p \to \mathfrak{c}_p \triangleleft \mathfrak{m}_p$.
- The cofree comonad $\mathfrak{c}_-$ is lax monoidal, $y \to \mathfrak{c}_y$ and $\mathfrak{c}_p \otimes \mathfrak{c}_{p'} \to \mathfrak{c}_{p \otimes p'}$.
- The free monad $\mathfrak{m}_-$ is *not* lax mon'l for $\otimes$ (though it is for $+$ and $\vee$).

For any p, q : **Poly**, there's a natural map $\Phi_{p,q} \colon \mathfrak{c}_p \otimes \mathfrak{m}_q \to \mathfrak{m}_{p \otimes q}$.[1]

---

[1] The notion of module here comes from nlab, "module over a monoidal functor". The module structure Φ is similar to a result of Katsumata-Rivas-Uustalu.

Interactions between $\mathfrak{m}_\_$ and $\mathfrak{c}_\_$

There are various interactions amongst free monads and cofree comonads.

- (Turi-Plotkin) “Oper'l semantics” is a distrib. law $\mathfrak{m}_p \triangleleft \mathfrak{c}_p \to \mathfrak{c}_p \triangleleft \mathfrak{m}_p$.
- The cofree comonad $\mathfrak{c}_\_$ is lax monoidal, $y \to \mathfrak{c}_y$ and $\mathfrak{c}_p \otimes \mathfrak{c}_{p'} \to \mathfrak{c}_{p \otimes p'}$.
- The free monad $\mathfrak{m}_\_$ is *not* lax mon'l for $\otimes$ (though it is for $+$ and $\vee$).

For any p, q : **Poly**, there's a natural map $\Phi_{p,q} \colon \mathfrak{c}_p \otimes \mathfrak{m}_q \to \mathfrak{m}_{p \otimes q}$.[1]

- We see that $\mathfrak{m}_\_$ is a left module over $\mathfrak{c}_\_$ by checking two diagrams:

$$\begin{tikzcd}
y \otimes \mathfrak{m}_q \arrow[r, "\cong"] \arrow[d] & \mathfrak{m}_q \arrow[d, "\cong"] \\
\mathfrak{c}_y \otimes \mathfrak{m}_q \arrow[r, "\Phi_{y,q}"'] & \mathfrak{m}_{y \otimes q}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathfrak{c}_p \otimes \mathfrak{c}_{p'} \otimes \mathfrak{m}_q \arrow[r, "\mathfrak{c}_p \otimes \Phi_{p',q}"] \arrow[d] & \mathfrak{c}_p \otimes \mathfrak{m}_{p' \otimes q} \arrow[d, "\Phi_{p,p' \otimes q}"] \\
\mathfrak{c}_{p \otimes p'} \otimes \mathfrak{m}_q \arrow[r, "\Phi_{p \otimes p',q}"'] & \mathfrak{m}_{p \otimes p' \otimes q}
\end{tikzcd}$$

---

[1] The notion of module here comes from nlab, "module over a monoidal functor". The module structure Φ is similar to a result of Katsumata-Rivas-Uustalu.

Interactions between $\mathfrak{m}_\_$ and $\mathfrak{c}_\_$

There are various interactions amongst free monads and cofree comonads.

- (Turi-Plotkin) “Oper’l semantics” is a distrib. law $\mathfrak{m}_p \triangleleft \mathfrak{c}_p \to \mathfrak{c}_p \triangleleft \mathfrak{m}_p$.
- The cofree comonad $\mathfrak{c}_\_$ is lax monoidal, $y \to \mathfrak{c}_y$ and $\mathfrak{c}_p \otimes \mathfrak{c}_{p'} \to \mathfrak{c}_{p \otimes p'}$.
- The free monad $\mathfrak{m}_\_$ is *not* lax mon’l for $\otimes$ (though it is for $+$ and $\vee$).

For any p, q : **Poly**, there’s a natural map $\Phi_{p,q} \colon \mathfrak{c}_p \otimes \mathfrak{m}_q \to \mathfrak{m}_{p \otimes q}$.[1]

- We see that $\mathfrak{m}_\_$ is a left module over $\mathfrak{c}_\_$ by checking two diagrams:

$$\begin{tikzcd}
y \otimes \mathfrak{m}_q \arrow[r, "\cong"] \arrow[d] & \mathfrak{m}_q \arrow[d, "\cong"] \\
\mathfrak{c}_y \otimes \mathfrak{m}_q \arrow[r, "\Phi_{y,q}"'] & \mathfrak{m}_{y \otimes q}
\end{tikzcd}$$

$$\begin{tikzcd}
\mathfrak{c}_p \otimes \mathfrak{c}_{p'} \otimes \mathfrak{m}_q \arrow[r, "\mathfrak{c}_p \otimes \Phi_{p',q}"] \arrow[d] & \mathfrak{c}_p \otimes \mathfrak{m}_{p' \otimes q} \arrow[d, "\Phi_{p,p' \otimes q}"] \\
\mathfrak{c}_{p \otimes p'} \otimes \mathfrak{m}_q \arrow[r, "\Phi_{p \otimes p',q}"'] & \mathfrak{m}_{p \otimes p' \otimes q}
\end{tikzcd}$$

But what does it mean, and how do you use it?

---

[1] The notion of module here comes from nlab, "module over a monoidal functor". The module structure Φ is similar to a result of Katsumata-Rivas-Uustalu.

How it works

How do we think about the map $\mathfrak{c}_p \otimes \mathfrak{m}_q \xrightarrow{\Phi} \mathfrak{m}_{p\otimes q}$?

- Think of $T : \mathfrak{c}_p(1)$ as a machine / operating system running forever.
- Think of $U : \mathfrak{m}_q(1)$ as a terminating program, or a finite flowchart.
- We can lay T next to U and move forward through both in tandem.
  - The root of the tandem thing is the pair of roots.
  - A branch of the tandem thing is a pair of branches.
  - Put a leaf whenever U hits a leaf; return to the remainder of T.

How it works

How do we think about the map $\mathfrak{c}_p \otimes \mathfrak{m}_q \xrightarrow{\Phi} \mathfrak{m}_{p\otimes q}$?

- Think of $T : \mathfrak{c}_p(1)$ as a machine / operating system running forever.
- Think of $U : \mathfrak{m}_q(1)$ as a terminating program, or a finite flowchart.
- We can lay T next to U and move forward through both in tandem.
    - The root of the tandem thing is the pair of roots.
    - A branch of the tandem thing is a pair of branches.
    - Put a leaf whenever U hits a leaf; return to the remainder of T.

Example: running Moore machines

- We said that an (A, B)-Moore machine sends A-lists to B-lists.
- An initialized (A, B)-Moore machine is a position $M : \mathfrak{c}_{By^A}(1)$.
- An A-list is a position $L : \mathfrak{m}_{Ay} = \mathsf{List}(A)y$.
- There is a map $By^A \otimes Ay \cong BAy^A \xrightarrow{B\epsilon} By$.
- Get: $y \cong y \otimes y \xrightarrow{M\otimes L} \mathfrak{c}_{By^A} \otimes \mathfrak{m}_{Ay} \xrightarrow{\Phi} \mathfrak{m}_{By^A\otimes Ay} \xrightarrow{B\epsilon} \mathfrak{m}_{By} = \mathsf{List}(B)y$.

Programs run on operators

```
def guessing_game(max_guesses, goal):
    if max_guesses==0:
        return False
    guess=read()
    if guess==goal:
        return True
    return guessing_game(max_guesses-1, goal)
```

Programs run on operators

```
def guessing_game(max_guesses, goal):
    if max_guesses==0:
        return False
    guess=read()
    if guess==goal:
        return True
    return guessing_game(max_guesses-1, goal)
```

Let's consider the following polynomial:

$$r := \sum_{\mathit{max\_guesses}:\mathbb{N}} \sum_{\mathit{goal}:\mathbb{N}} y^{\mathsf{Bool}}$$

Programs run on operators

```
def guessing_game(max_guesses, goal):
    if max_guesses==0:
        return False
    guess=read()
    if guess==goal:
        return True
    return guessing_game(max_guesses-1, goal)
```

Let's consider the following polynomial:

$$r := \sum_{\mathit{max\_guesses}:\mathbb{N}} \sum_{\mathit{goal}:\mathbb{N}} y^{\mathsf{Bool}}$$

We define a map $r \to \mathbb{N}y$ that plays the game. Ingredients:

- A pos'n in $\mathfrak{m}_{y^{\mathbb{N}}}$ is a flowchart of guesses. The program is $\pi\colon r \to \mathfrak{m}_{y^{\mathbb{N}}}$.
- A position $\sigma\colon y \to \mathfrak{c}_{\mathbb{N}y}$ is an operator (you? OS?) emitting guesses in $\mathbb{N}$.
- Note that $y^{\mathbb{N}}$ and $\mathbb{N}y$ are dual, i.e. $[y^{\mathbb{N}}, y] \cong \mathbb{N}y$.
- Use composite: $r \cong y \otimes r \xrightarrow{\sigma\otimes\pi} \mathfrak{c}_{\mathbb{N}y} \otimes \mathfrak{m}_{y^{\mathbb{N}}} \xrightarrow{\Phi} \mathfrak{m}_{[y^{\mathbb{N}},y]\otimes y^{\mathbb{N}}} \to \mathfrak{m}_y \cong \mathbb{N}y$

Programs run on operators

```
def guessing_game(max_guesses, goal):
    if max_guesses==0:
        return False
    guess=read()
    if guess==goal:
        return True
    return guessing_game(max_guesses-1, goal)
```

Let's consider the following polynomial:

$$r := \sum_{\mathit{max\_guesses}:\mathbb{N}} \sum_{\mathit{goal}:\mathbb{N}} y^{\mathsf{Bool}}$$

We define a map $r \to \mathbb{N}y$ that plays the game. Ingredients:

- A pos'n in $\mathfrak{m}_{y^{\mathbb{N}}}$ is a flowchart of guesses. The program is $\pi\colon r \to \mathfrak{m}_{y^{\mathbb{N}}}$.
- A position $\sigma\colon y \to \mathfrak{c}_{\mathbb{N}y}$ is an operator (you? OS?) emitting guesses in $\mathbb{N}$.
- Note that $y^{\mathbb{N}}$ and $\mathbb{N}y$ are dual, i.e. $[y^{\mathbb{N}}, y] \cong \mathbb{N}y$.
- Use composite: $r \cong y \otimes r \xrightarrow{\sigma\otimes\pi} \mathfrak{c}_{\mathbb{N}y} \otimes \mathfrak{m}_{y^{\mathbb{N}}} \xrightarrow{\Phi} \mathfrak{m}_{[y^{\mathbb{N}},y]\otimes y^{\mathbb{N}}} \to \mathfrak{m}_y \cong \mathbb{N}y$

Here the stream σ didn't take inputs because $\mathbb{N}y = [y^{\mathbb{N}}, y]$ was particularly simple.

Outline

Summary

We are interested in the relationship between pattern and matter.

- Here, we're thinking of patterns as terminating programs, like scripts.
- And we're thinking of matter as dynamics that continues forever.
- What does it mean to run the pattern on the matter?

Summary

We are interested in the relationship between pattern and matter.

- Here, we're thinking of patterns as terminating programs, like scripts.
- And we're thinking of matter as dynamics that continues forever.
- What does it mean to run the pattern on the matter?

One answer: think of pattern as monad and matter as comonad.

- We constructed the (co)free (co)monad on any polynomial functor p.
- We showed how $\mathfrak{c}_p$ and $\mathfrak{m}_p$ look like two different types of p-tree.

Summary

We are interested in the relationship between pattern and matter.

- Here, we're thinking of patterns as terminating programs, like scripts.
- And we're thinking of matter as dynamics that continues forever.
- What does it mean to run the pattern on the matter?

One answer: think of pattern as monad and matter as comonad.

- We constructed the (co)free (co)monad on any polynomial functor p.
- We showed how $\mathfrak{c}_p$ and $\mathfrak{m}_p$ look like two different types of p-tree.

There are many interesting interactions between $\mathfrak{c}_p$ and $\mathfrak{m}_p$.

- Matter runs on matter: $\mathfrak{c}_p \otimes \mathfrak{c}_{p'} \to \mathfrak{c}_{p\otimes p'}$. We noted that pattern doesn't run on pattern.
- So it's meaningful to say that $\mathfrak{m}_-$ is a $\mathfrak{c}_-$-module: $\mathfrak{c}_p \otimes \mathfrak{m}_q \to \mathfrak{m}_{p\otimes q}$.
- This statement gives math'ical meaning to "pattern runs on matter."

Thanks; comments and questions welcome!