Is Poly the true language of computation?

David I. Spivak

Grant FA9550-20-1-0348. Thanks to Fred Leve, Tristan Nguyen, & Doug Riecken

Outline

1. **Introduction**
   - Finding the right abstractions for complex systems
   - Mode dependent interaction
   - Plan for the talk
2. The well-spring
3. How to use it?
4. Conclusion

Algebraic structure of complex systems

How are complex systems composed, and how do they interact?

- Complex systems are all around us: me, you, a phone, the AFOSR.
- Each is composed of simpler systems that are interacting.
- Zooming in or out of these systems, you can say the same thing.
- If we can understand this fractal, perhaps we can steer it.

Algebraic structure of complex systems

How are complex systems composed, and how do they interact?

- Complex systems are all around us: me, you, a phone, the AFOSR.
- Each is composed of simpler systems that are interacting.
- Zooming in or out of these systems, you can say the same thing.
- If we can understand this fractal, perhaps we can steer it.

These systems are not haphazard: they are highly structured.

- Building a phone requires specialized but interoperable parts.
- Same for our bodies, our conceptual apparatus, the AFOSR, etc.
- They are put together in very particular ways: highly structured.
- The pieces are not generally static; they interact dynamically.
- A phone interacts dynam'ly with cell networks, the internet, and you.

Algebraic structure of complex systems

How are complex systems composed, and how do they interact?

- Complex systems are all around us: me, you, a phone, the AFOSR.
- Each is composed of simpler systems that are interacting.
- Zooming in or out of these systems, you can say the same thing.
- If we can understand this fractal, perhaps we can steer it.

These systems are not haphazard: they are highly structured.

- Building a phone requires specialized but interoperable parts.
- Same for our bodies, our conceptual apparatus, the AFOSR, etc.
- They are put together in very particular ways: highly structured.
- The pieces are not generally static; they interact dynamically.
- A phone interacts dynam'ly with cell networks, the internet, and you.

What algebra of purely structural rules can capture this richness?

Category theory: accounting for structure

I think of mathematical fields as accounting systems.

- Arithmetic accounts for the flow of quantities, as in finance.
- Hilbert spaces account for the states of elementary particles, as in QM.
- Probability distributions account for likelihoods, as in game theory.

Category theory: accounting for structure

I think of mathematical fields as accounting systems.

- Arithmetic accounts for the flow of quantities, as in finance.
- Hilbert spaces account for the states of elementary particles, as in QM.
- Probability distributions account for likelihoods, as in game theory.

An account of phenomena needs to be written in high-fidelity language.

- To understand the phenomena requires that certain aspects be tracked.
- The language must articulate the relevant type-differences ...
- ... and provide operations that correspond with their interactions.

Category theory: accounting for structure

I think of mathematical fields as accounting systems.

- Arithmetic accounts for the flow of quantities, as in finance.
- Hilbert spaces account for the states of elementary particles, as in QM.
- Probability distributions account for likelihoods, as in game theory.

An account of phenomena needs to be written in high-fidelity language.

- To understand the phenomena requires that certain aspects be tracked.
- The language must articulate the relevant type-differences ...
- ... and provide operations that correspond with their interactions.

Category theory is the accounting system for coherent structures.

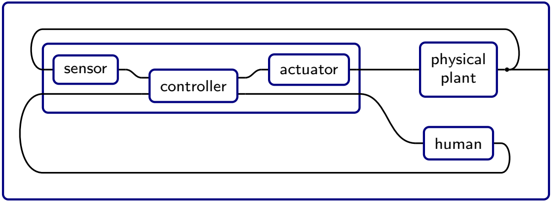
- It makes analogies—similarities of structure—into formal objects.
- It finds conceptual neighbors who can learn each others' techniques.
- It's been useful in math, CS, physics, materials science, linguistics, etc

To account for the structure of complex systems, we need the right cat'y.

How this project started

In 2016, I noticed a formal analogy between seemingly disparate fields.

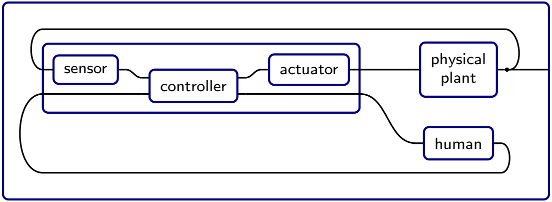
- We can wire open dynamical systems (ODS's) in patterns like this:



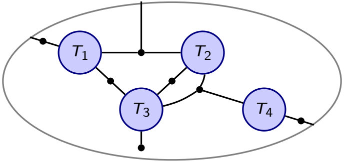
How this project started

In 2016, I noticed a formal analogy between seemingly disparate fields.

- We can wire open dynamical systems (ODS's) in patterns like this:



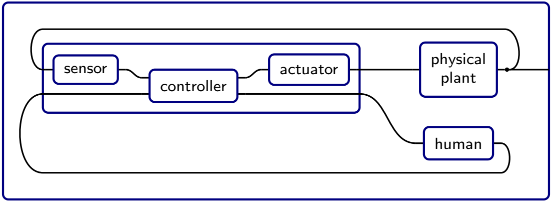
- We see similar sorts of diagrams for tensor networks:



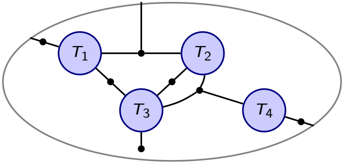
How this project started

In 2016, I noticed a formal analogy between seemingly disparate fields.

- We can wire open dynamical systems (ODS's) in patterns like this:



- We see similar sorts of diagrams for tensor networks:



If you associate to each (arbitrary, nonlinear) ODS its steady state tensor...
... then composing ODS's agrees with contracting tensor networks.

Mode dependent interaction

The algebraic gadget that lets you compose pieces is called an *operad*.

- Use one operad to specify how dynamical systems compose.
- Use another operad to specify how tensor networks compose.
- There's a functor that connects one to the other: steady states.

Mode dependent interaction

The algebraic gadget that lets you compose pieces is called an *operad*.

- Use one operad to specify how dynamical systems compose.
- Use another operad to specify how tensor networks compose.
- There's a functor that connects one to the other: steady states.

Most real-life systems have dynamic—not fixed—interaction patterns.

- The operads in the previous slide showed fixed interaction patterns.
- What we say to each other should influence how we're connected.
- I wanted an operad for *mode dependent interaction*.
- I found one, but the math wasn't elegant.

Mode dependent interaction

The algebraic gadget that lets you compose pieces is called an *operad*.

- Use one operad to specify how dynamical systems compose.
- Use another operad to specify how tensor networks compose.
- There's a functor that connects one to the other: steady states.

Most real-life systems have dynamic—not fixed—interaction patterns.

- The operads in the previous slide showed fixed interaction patterns.
- What we say to each other should influence how we're connected.
- I wanted an operad for *mode dependent interaction*.
- I found one, but the math wasn't elegant.

But in 2019, I found polynomial functors and felt I'd hit the jackpot.

Last time

In 2021, I talked about how polynomial functors model:

- open dynamical systems (both continuous and discrete),
- wiring diagrams (both static and dynamically changing), and
- deep learning: the changing interaction structure is gradient descent.

Last time

In 2021, I talked about how polynomial functors model:

- open dynamical systems (both continuous and discrete),
- wiring diagrams (both static and dynamically changing), and
- deep learning: the changing interaction structure is gradient descent.

But the more I see of **Poly**, the more potential it seems to have.

- Today I'm going to make the case that it's at least reasonable to ask:
- Is **Poly** the true language of computation?

I'll explain what that means, then give the evidence collected so far.

Plan for the talk

Here's the plan for the rest of the talk:

- Say what I mean by a "true language of computation".
- Give a bunch of evidence for why **Poly** is a contender.
- Say how I want to move forward.
- Conclude with a summary.

Outline

Some language is better than others

What does mathematics have over natural language?

- For certain topics, math better clarifies and advances the discussion.
- Let's be honest: certain languages just *work better* than others.
  - Isn't the Hindi number system just *better* than Roman?
  - Isn't the $(\top, \bot, \wedge, \vee, \Rightarrow, \forall, \exists)$-logic *better* than syllogisms?

Some language is better than others

What does mathematics have over natural language?

- For certain topics, math better clarifies and advances the discussion.
- Let's be honest: certain languages just *work better* than others.
  - Isn't the Hindi number system just *better* than Roman?
  - Isn't the $(\top, \bot, \wedge, \vee, \Rightarrow, \forall, \exists)$-logic *better* than syllogisms?

The Turing machine changed everything; but is it *right*?

- The Von Neumann architecture changed everything, but is it right?
- Rust, Python, Julia are great languages, but...
- In 100 years will we be using someone's invented language?

Some language is better than others

What does mathematics have over natural language?

- For certain topics, math better clarifies and advances the discussion.
- Let's be honest: certain languages just *work better* than others.
  - Isn't the Hindi number system just *better* than Roman?
  - Isn't the $(\top, \bot, \wedge, \vee, \Rightarrow, \forall, \exists)$-logic *better* than syllogisms?

The Turing machine changed everything; but is it *right*?

- The Von Neumann architecture changed everything, but is it right?
- Rust, Python, Julia are great languages, but...
- In 100 years will we be using someone's invented language?

Classic question: is math invented or discovered?

- Some of each! Sometimes people invent math, other times discover it.
- The complex numbers *just freaking work*. Right?
- You can't just invent the fundamental theorem of calculus or algebra.

What is a true language of computation?

It's hard to imagine improving on the Hindi-Arabic numerals.

- They have maximum entropy (the information cannot be compressed).
- Unlike Roman, they utilize the distributive law.
- It's hard to imagine improving on the complex numbers.
- This can't be said about Python: it's being improved all the time.

What is a true language of computation?

It's hard to imagine improving on the Hindi-Arabic numerals.

- They have maximum entropy (the information cannot be compressed).
- Unlike Roman, they utilize the distributive law.
- It's hard to imagine improving on the complex numbers.
- This can't be said about Python: it's being improved all the time.

Computation—the processing of information—is central to our world.

- Church-Turing thesis: all notions of computation are equally expressive
- Turing machines, λ-calculus, and recursive functions: all agree
- I propose that we can do better than Python, Rust, Julia.
- With something this central, there may be a "right" language.

What is a true language of computation?

It's hard to imagine improving on the Hindi-Arabic numerals.

- They have maximum entropy (the information cannot be compressed).
- Unlike Roman, they utilize the distributive law.
- It's hard to imagine improving on the complex numbers.
- This can't be said about Python: it's being improved all the time.

Computation—the processing of information—is central to our world.

- Church-Turing thesis: all notions of computation are equally expressive
- Turing machines, λ-calculus, and recursive functions: all agree
- I propose that we can do better than Python, Rust, Julia.
- With something this central, there may be a "right" language.

A true language of computation should be discovered, not invented.

- It should be constructed out of very basic ideas.
- It should have very diverse computational applications.
- It should have a small set of "orthogonal" constructors.
- It should be like legos or tinker-toys, but made out of math.

Poly: Built simply but has tons of structure

There is a construction $\mathscr{F}$ that makes new categories from old.

- It sends $\mathcal{C}$ to "the free coproduct completion of $\mathcal{C}^{\mathsf{op}}$."
- Let's see what happens when you perform this operation repeatedly.
  - Do it to the empty cat'y $\mathbf{0}$, get the one-object cat'y $\mathbf{1} = \mathscr{F}(\mathbf{0})$.
  - Do it to $\mathbf{1}$, get the category **Set** $= \mathscr{F}(\mathbf{1})$ of all sets.
  - Do it to **Set**, get **Poly** $= \mathscr{F}(\textbf{Set}) = \mathscr{F}(\mathscr{F}(\mathbf{1})) = \mathscr{F}(\mathscr{F}(\mathscr{F}(\mathbf{0})))$.
- You can keep going, but there's a sense in which **Poly** is sufficient.

Poly: Built simply but has tons of structure

There is a construction $\mathcal{F}$ that makes new categories from old.

- It sends $\mathcal{C}$ to "the free coproduct completion of $\mathcal{C}^{\mathsf{op}}$."
- Let's see what happens when you perform this operation repeatedly.
  - Do it to the empty cat'y $\mathbf{0}$, get the one-object cat'y $\mathbf{1} = \mathcal{F}(\mathbf{0})$.
  - Do it to $\mathbf{1}$, get the category $\mathbf{Set} = \mathcal{F}(\mathbf{1})$ of all sets.
  - Do it to **Set**, get $\mathbf{Poly} = \mathcal{F}(\mathbf{Set}) = \mathcal{F}(\mathcal{F}(\mathbf{1})) = \mathcal{F}(\mathcal{F}(\mathcal{F}(\mathbf{0})))$.
- You can keep going, but there's a sense in which **Poly** is sufficient.

The $\mathcal{F}$ operation preserves structures. If $\mathcal{C}$ has X then $\mathcal{F}(\mathcal{C})$ has Y.

- X=nothing; Y=coproducts
- X=colimits; Y=limits
- X=limits; Y=colimits
- X=monoidal structure; Y=monoidal structure
- X=closure; Y=coclosure
- X=coclosure; Y=closure

Since $\mathbf{1}$ has every structure above, **Poly** does too:
$+, \times, \otimes, \triangleleft, \vee, [-,-], \left[\frac{-}{-}\right], (- \frown -)$.

Natural applications of Poly

The point is that **Poly** is discovered and highly applicable.

- It's discovered in the sense that it's part of a fundamental sequence...
- ...namely the sequence of free op-completions: $\mathbf{0} \mapsto \mathbf{1} \mapsto \mathbf{Set} \mapsto \mathbf{Poly}$.

Natural applications of Poly

The point is that **Poly** is discovered and highly applicable.

- It's discovered in the sense that it's part of a fundamental sequence...
- ...namely the sequence of free op-completions: $\mathbf{0} \mapsto \mathbf{1} \mapsto \mathbf{Set} \mapsto \mathbf{Poly}$.

But in what sense is it highly applicable?

- Awodey showed that poly'l monads are universes for dependent types.
- Polynomial comodules migrate data between databases.
- Typed programming lang's rely heavily on poly datatypes and monads.
- Categories themselves are the comonoids in **Poly**.
- In fact higher categories (double cats, ∞-cats) live naturally in **Poly**.
- Cellular automata have a natural description in **Poly**.
- Dynamical systems—cts or discrete—have natural descrip'ns in **Poly**.
- Deep learning (as I explained last year) has a natural descrip'n in **Poly**
- Wiring diagrams, mode dependence have natural descriptions in **Poly**.
- Turing machines have a natural description in **Poly**.

Natural applications of Poly

The point is that **Poly** is discovered and highly applicable.

- It's discovered in the sense that it's part of a fundamental sequence...
- ...namely the sequence of free op-completions: $\mathbf{0} \mapsto \mathbf{1} \mapsto$ **Set** $\mapsto$ **Poly**.

But in what sense is it highly applicable?

- Awodey showed that poly'l monads are universes for dependent types.
- Polynomial comodules migrate data between databases.
- Typed programming lang's rely heavily on poly datatypes and monads.
- Categories themselves are the comonoids in **Poly**.
- In fact higher categories (double cats, ∞-cats) live naturally in **Poly**.
- Cellular automata have a natural description in **Poly**.
- Dynamical systems—cts or discrete—have natural descrip'ns in **Poly**.
- Deep learning (as I explained last year) has a natural descrip'n in **Poly**
- Wiring diagrams, mode dependence have natural descriptions in **Poly**.
- Turing machines have a natural description in **Poly**.

So it's got syntax, operations, dynamics, learning, nested control, etc.

Outline

The design environment idea

A few years ago, DARPA had a project called CASCADE

- Complex Adaptive System Composition and Design Environment.
- It was quite ambitious, but I don't think it was particularly successful.
- I think that **Poly** could serve as the foundational language for it.
- It's at once low level (Turing machines) and high level (dep. types).
- It naturally describes PL, DB, DS, TM, CA, IT, ML.

The design environment idea

A few years ago, DARPA had a project called CASCADE

- Complex Adaptive System Composition and Design Environment.
- It was quite ambitious, but I don't think it was particularly successful.
- I think that **Poly** could serve as the foundational language for it.
- It's at once low level (Turing machines) and high level (dep. types).
- It naturally describes PL, DB, DS, TM, CA, IT, ML.

So what might we mean by a "design environment"?

What do we mean by a design environment?

I am still fleshing out what I want to build, but here's the idea.

- Build new machines from old, using series, parallel, feedback.
- Control the speed of each component.
- Control when the components disconnect and reconnect.
- Construct "game-style" protocols for complex "handshakes".
- All of this is done using universal operations within **Poly**,

$$+, \times, \otimes, \triangleleft, \vee, [-,-], \begin{bmatrix} - \\ - \end{bmatrix}, (- \frown -)$$

What do we mean by a design environment?

I am still fleshing out what I want to build, but here's the idea.

- Build new machines from old, using series, parallel, feedback.
- Control the speed of each component.
- Control when the components disconnect and reconnect.
- Construct "game-style" protocols for complex "handshakes".
- All of this is done using universal operations within **Poly**,

$$+, \times, \otimes, \triangleleft, \vee, [-,-], \begin{bmatrix} - \\ - \end{bmatrix}, (- \frown -)$$

The workflow will likely be:

- Take some already-made components off the virtual shelf;
- Connect them together as above;
- Prove a guarantee from those of the parts (could be trivial `true`)
- Put the result back on the shelf for others (and later you) to use.

Examples

Examples of what you should be able to build in this design environment:

- Music software (e.g. Max): build songs by connecting components.
- Internet of things: connect your fridge camera to your car navigation!
- Build a universal Turing machine and have it run internet programs.
- Connect a deep learner to a data-set searcher.
- Make chat-bots that connect to twitter in specified ways.

Examples

Examples of what you should be able to build in this design environment:

- Music software (e.g. Max): build songs by connecting components.
- Internet of things: connect your fridge camera to your car navigation!
- Build a universal Turing machine and have it run internet programs.
- Connect a deep learner to a data-set searcher.
- Make chat-bots that connect to twitter in specified ways.

I think with this system, we could significantly bother Elon Musk.

Outline

Summary

The category **Poly** is a superlative mathematical object.

- It is unparalleled in combinatorial structure.
- It arises elegantly as $\mathscr{F}(\mathscr{F}(\mathscr{F}(\mathbf{0})))$, in the sequence $\mathbf{0}, \mathbf{1}, \mathbf{Set}, \mathbf{Poly}$.
- I found it in my search for open dyn'l systems that selectively rewire.
- It has applications to PL, DB, DS, TM, CA, IT, ML.

Summary

The category **Poly** is a superlative mathematical object.

- It is unparalleled in combinatorial structure.
- It arises elegantly as $\mathscr{F}(\mathscr{F}(\mathscr{F}(\mathbf{0})))$, in the sequence $\mathbf{0}, \mathbf{1}, \mathbf{Set}, \mathbf{Poly}$.
- I found it in my search for open dyn'l systems that selectively rewire.
- It has applications to PL, DB, DS, TM, CA, IT, ML.

Having everything from dependent types to Turing machines suggests more.

- There may be a "God-given" language of computation.
- **Poly** or not, it's worth considering the possibility.

Summary

The category **Poly** is a superlative mathematical object.

- It is unparalleled in combinatorial structure.
- It arises elegantly as $\mathscr{F}(\mathscr{F}(\mathscr{F}(\mathbf{0})))$, in the sequence $\mathbf{0}, \mathbf{1}, \mathbf{Set}, \mathbf{Poly}$.
- I found it in my search for open dyn'l systems that selectively rewire.
- It has applications to PL, DB, DS, TM, CA, IT, ML.

Having everything from dependent types to Turing machines suggests more.

- There may be a "God-given" language of computation.
- **Poly** or not, it's worth considering the possibility.

By constructing a design environment, I can make this idea accessible.

- As the idea becomes legible, more people will join the effort.
- Having this single, tight, elegant, highly articulate language...
- ...with so many diverse applications, could revolutionize computing.

Summary

The category **Poly** is a superlative mathematical object.

- It is unparalleled in combinatorial structure.
- It arises elegantly as $\mathscr{F}(\mathscr{F}(\mathscr{F}(\mathbf{0})))$, in the sequence $\mathbf{0}, \mathbf{1}, \mathbf{Set}, \mathbf{Poly}$.
- I found it in my search for open dyn'l systems that selectively rewire.
- It has applications to PL, DB, DS, TM, CA, IT, ML.

Having everything from dependent types to Turing machines suggests more.

- There may be a "God-given" language of computation.
- **Poly** or not, it's worth considering the possibility.

By constructing a design environment, I can make this idea accessible.

- As the idea becomes legible, more people will join the effort.
- Having this single, tight, elegant, highly articulate language...
- ...with so many diverse applications, could revolutionize computing.

Thanks; comments and questions welcome!