Operadics: the mathematics of modular design

David I. Spivak

dspivak@math.mit.edu
Mathematics Department
Massachusetts Institute of Technology

Presented on 2015/06/17

at NIST

Outline

Outline

The promise of fractals

- I recall my father telling me about a kind of "fractal fever".
- In the 1980s scientists were very interested in fractals, e.g., in:
    - Plants (a single leaf or broccoli).
    - Rivers, faults, and vasculature.
    - Stock market fluctuations.
- Scientists wanted to use fractals as a conceptual tool for explaining phenomena.

It didn't quite work for everyone; why?

- Fractals are a little too special: the machinery is too limited.
- Scientists for whom the analogy was compelling couldn't always produce:
  - shapes with fractional dimension,
  - patterns that repeat no matter how far you zoom in,
  - iterated functions or recurrence relations to generate their phenomena.
- Fractals are always about space and geometry.
- The inspirational and compelling idea wasn't completely realized.
  - Unlike fractals, the cases of interest weren't always geometric objects.
  - Example: heredity and evolution occur hierarchically, but not spacially.

Operads describe similar phenomena

- I believe the promise of fractals may still be realized by operads.
- By "the promise of fractals" I roughly mean:
  - a mathematical formalism for understanding self-similarity across scales.
- An operad $\mathcal{O}$ is a collection of operations, which can be combined.
  - Operads can reproduce fractals as fixed points of operations on $\mathbb{C}$.
  - But operads are much more flexible than fractals.
  - They're not just about geometry and contraction mappings.
- Operads are the mathematics of modularity.
  - Modules can be combined according to the operations in $\mathcal{O}$.
  - The result is a new module, ready to be further put in combination.

Plan of the talk

- I'm leaving fractals aside; they were just motivation.
- I want to explain operads: how they might be interesting to scientists.
- Here's the plan:
    - Discuss a better running example: recipes.
    - Give the formal definition of operads.
    - Provide a couple different examples: materials and networks.
    - Conclude the talk.
- The main theme will be modularity:
    - Building up complex systems by combining subsystems.

Outline

Recipes

Here's a recipe for impressing ones new friend:

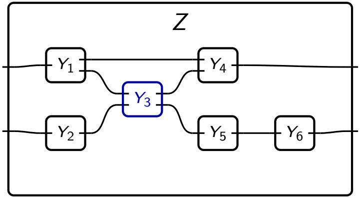
- Invite them over.
- Prepare before they arrive.
    - Make sure the house is clean.
    - Cook a fancy dinner.
        - Find a recipe that people say is good.
        - Go to the store to get ingredients.
        - Follow the recipe. [Itself a recipe....]
    - Think of a few things to talk about with the guests.
- When they arrive:
    - Offer them a drink.
        - Ask them what kind of drinks they like.
        - Determine which of these can be made with ingredients.
        - Follow the recipe. [Itself a recipe....]
    - (etc.)
- (etc.)

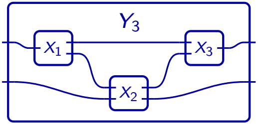
What's operadic about recipes

- A recipe is built out of steps which are themselves sub-recipes.
- These sub-recipes can be done in series, or in parallel.
- It has to do with zooming and chunking.
  - Can we zoom in forever and see recipes all the way down?
  - Maybe, but that's not a necessary part of being an operad.
  - What's necessary is that you can zoom out.
  - You can put recipes together (series and parallel); the result is a recipe.
- Put together a recipe for batter and one for frosting, and make a cake.

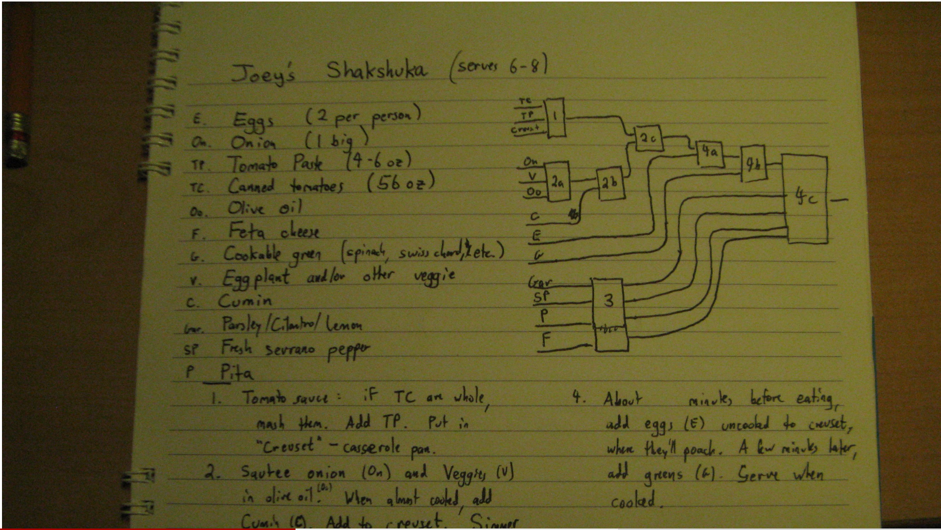
A picture of a recipe

- On the left you see a recipe for Z.
- The steps are $Y_1, \dots Y_6$.
  - Some have a specific order: step Y_1 must be done before Y_3.
  - Others don't: step Y_4 can be done in any order with Y_5 and Y_6.
- We can elaborate on the details of Y_3, to see how it's implemented.
  - Shown on the right: note it has the correct number of in/out ports.
  - To substitute it on the left, replace module Y_3 with X_1, X_2, X_3.





Example: a recipe for shakshuka



Category Theory

- Operads are a sub-discipline of category theory (CT).
- Since its invention in the 1940s, CT has revolutionized math.
    - It is able to connect disparate disciplines into a unified framework.
    - It abstracts common themes from algebra, topology, and logic.
    - It's the key to accessing the world of pure math.
- Category theory has been applied outside of math as well.
    - Computer science (functional programming, databases),
    - Physics (Feynman diagrams, quantum information theory).

Applied category theory

- Operads, like all of CT, was invented for its use in pure math.
- The notion of "modular systems" fits naturally into this framework.
- I'm speaking to you in the very early stages of this application.
  - I don't yet know all the ways in which operads will be useful.
  - But operads have demonstrated their power in pure math.
  - And pure math has demonstrated its utility in science.
- Future progress will be driven by collaborations.

Outline

An operad is an "abstract modular environment"

- I will define operads formally in a few slides.
- An operad $\mathcal{O}$ is a framework for any sort of modularity.
- To specify $\mathcal{O}$ is to specify:
  - The set of module types (or interfaces) you'll consider.
  - The ways that modules can be put together to form larger ones.
  - How nesting works. (Usually feels obvious, but it must be specified.)
- Recipes, as we discussed, fits this description:
  - A module type is a box with input and output channels (ingredients).
  - Boxes are put together by connecting ingredient supply to demand.
  - Nesting is accomplished by expanding a step as a recipe of its own.

What is an operad? An overview

- An operad consists of a few interlocking components, including:
  1. A set of *objects*, a.k.a. module types, interfaces, or building blocks.
  2. A set of *morphisms*, a.k.a., arrangements or building instructions.
  3. A formula for *composition*, a.k.a, nesting or instruction composition.
- Objects, morphisms, and compositions are the heart and soul of CT.

Formal definition of operad

An operad $\mathcal{O}$ consists of

- A set $\text{Ob}(\mathcal{O})$, elements of which are called *objects*, or interfaces.
- For interfaces $X_1, \ldots, X_n, Y \in \text{Ob}(\mathcal{O})$, a set

$$\text{Mor}_{\mathcal{O}}(X_1, \ldots, X_n; Y)$$

  Its elements are called *morphisms* or arrangements of $X_1, \ldots, X_n$ in Y. An arrangement $\varphi \in \text{Mor}_{\mathcal{O}}(X_1, \ldots, X_n; Y)$ may be denoted

$$\varphi\colon (X_1, \ldots, X_n) \to Y.$$

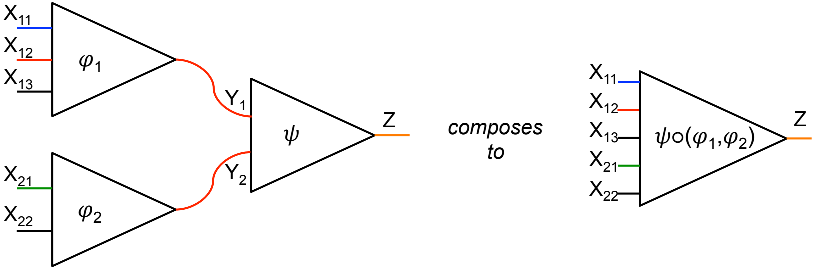
- For each object $X \in \text{Ob}(\mathcal{O})$, an identity arrangement $\text{id}_X \colon (X) \to X$
- A composition, or nesting formula, e.g.,

$$\psi \circ (\varphi_1, \ldots, \varphi_n)\colon (X_{i,j}) \xrightarrow{\varphi_i} (Y_i) \xrightarrow{\psi} Z.$$

These are required to satisfy well-known "unital" and "associative" laws.

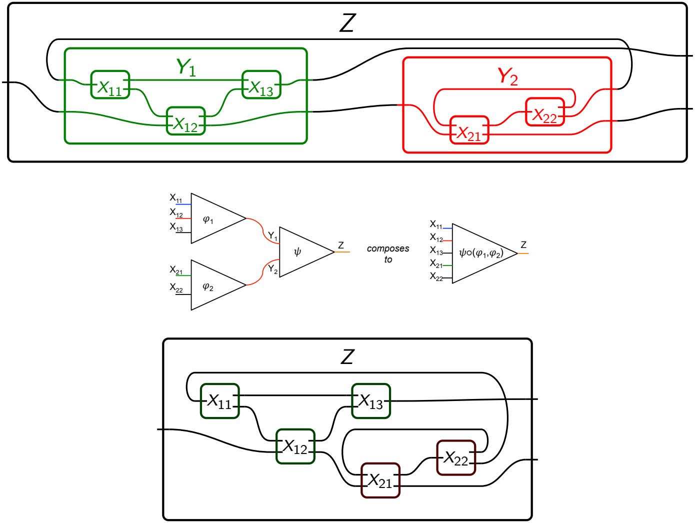
Another way to see it

- Often the objects in operad are shown as colors.
- The morphisms are many-input, one-output relationships.
- They can be composed:



- Here, ψ represents an arrangement of a Y_1 and a Y_2 to make a Z.

Example: composition of networks



Every context-free grammar (CFG) is an operad

The abstract modular environment of postal addresses: [1]

⟨postal-address⟩	::=	⟨name-part⟩ ⟨street-address⟩ ⟨zip-part⟩
⟨name-part⟩	::=	⟨personal-part⟩ ⟨last-name⟩ ⟨opt-suffix-part⟩ ⟨EOL⟩
	|	⟨personal-part⟩ ⟨name-part⟩
⟨personal-part⟩	::=	⟨first-name⟩ | ⟨initial⟩ "."
⟨street-address⟩	::=	⟨house-num⟩ ⟨street-name⟩ ⟨opt-apt-num⟩ ⟨EOL⟩
⟨zip-part⟩	::=	⟨town-name⟩ "," ⟨state-code⟩ ⟨ZIP-code⟩ ⟨EOL⟩
⟨opt-suffix-part⟩	::=	"Sr." | "Jr." | ⟨roman-numeral⟩ | ""
⟨opt-apt-num⟩	::=	⟨apt-num⟩ | ""

- Everything in ⟨brackets⟩ is an object.
- Each line is a morphism, usually called a "production rule".
- Composition—nesting—of production rules is straightforward.
- The usual interpretation of this CFG: strings and concatenations.

[1] Copied verbatim from Wikipedia page on Backus-Naur Form.

The operad of sets

Recall the category **Set**: objects are sets, morphisms are functions. Also, for any n sets $X_1, \ldots, X_n$, there is a product set $X_1 \times \cdots \times X_n$.

Definition

The operad **Sets** is defined by

- $\mathrm{Ob}(\mathbf{Sets}) = \mathrm{Ob}(\mathbf{Set})$
- $\mathrm{Mor}_{\mathbf{Sets}}(X_1, \ldots, X_n; Y) = \mathrm{Mor}_{\mathbf{Set}}(X_1 \times \cdots \times X_n, Y)$
- Identity and composition are straightforward and well-known.

This construction works for any monoidal category, not just $(\mathbf{Set}, 1, \times)$.

Operad functors and operad algebras

Let $\mathcal{O}$ and $\mathcal{O}'$ be operads.

Definition

An operad functor $F\colon \mathcal{O} \to \mathcal{O}'$ consists of:

- a function $F\colon \mathrm{Ob}(\mathcal{O}) \to \mathrm{Ob}(\mathcal{O}')$,
- for objects $X_1, \ldots, X_n, Y$, a function

$$F\colon \mathrm{Mor}_{\mathcal{O}}(X_1, \ldots, X_n; Y) \to \mathrm{Mor}_{\mathcal{O}'}(FX_1, \ldots, FX_n; FY).$$

- These two functions should respect identity and composition.

Definition

An operad functor $F\colon \mathcal{O} \to \mathbf{Sets}$ is called an *$\mathcal{O}$-algebra*.

Operad functors and operad algebras

Let $\mathcal{O}$ and $\mathcal{O}'$ be operads.

Definition

An operad functor $F\colon \mathcal{O} \to \mathcal{O}'$ consists of:

- a function $F\colon \mathrm{Ob}(\mathcal{O}) \to \mathrm{Ob}(\mathcal{O}')$,
- for objects $X_1, \ldots, X_n, Y$, a function

$$F\colon \mathrm{Mor}_{\mathcal{O}}(X_1, \ldots, X_n; Y) \to \mathrm{Mor}_{\mathcal{O}'}(FX_1, \ldots, FX_n; FY).$$

- These two functions should respect identity and composition.

Definition

An operad functor $F\colon \mathcal{O} \to \textbf{Sets}$ is called an $\mathcal{O}$-*algebra*.

Operads and algebras = syntax and semantics

Throughout this talk we'll have:

- An operad $\mathcal{O}$ governing the types and constructions,
  - The objects and morphisms of $\mathcal{O}$.
  - I might call them building block types and building instructions.
- And an algebra $X\colon \mathcal{O} \to \mathbf{Set}$.
  - It'll tell us the set of building blocks of each type.
  - And how to build new ones by applying instructions.
- For example, if $\mathcal{C}$ is a context-free grammar
  - What people call an "attribute grammar" is a $\mathcal{C}$-algebra.
  - Attribute grammars have been used in design, e.g., shape grammars.
- An operad $\mathcal{O}$ is just a (possibly infinite) CFG with equations.
- An $\mathcal{O}$-algebra can be thought of as an attribute grammar on $\mathcal{O}$.

Outline

Potential domains of application

Operads might organize how we think about a variety of applied problems:

- Potential applications to:
  - Manufacturing processes,
  - Signaling networks in systems biology,
  - Neural circuits.
- A successful collaboration: applying operads in materials science.
- Plan for remainder of talk:
  - We'll switch gears and discuss the materials case in some detail.
  - Then we'll wind down with networks.

A tool for producing hierarchical protein materials

- Bio-inspired design of hierarchical protein materials.
  - Materials such as silk and collagen have excellent properties.
  - We want to modify their structure, e.g, to make them heat resistant.
  - Scientists do so by simulating the structures using molecular dynamics.
- The process for simulating hierarchical protein materials is tedious.
  - Because it's such a new field, there is a lack of organization.
  - People program amino-acid placement by hand.
  - Compromise equilibration-time efficiency for programming efficiency.
- We developed a tool for creating hierarchical protein materials.
- It is called *Matriarch*, standing for materials architecture.
- And (of course) it is based on operads.

http://web.mit.edu/matriarch/

The operadic model of Matriarch

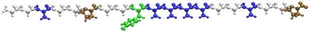
Let's describe the operad $\mathcal{M}$ for Matriarch.

- The objects (building blocks) in $\mathcal{M}$ are proteins.
  - These start with amino acids, but include everything you can build.
  - They are differentiated according to their bondable interface.
- The morphisms (building instructions) in $\mathcal{M}$ are commands such as:
  - 1-ary: `reverse`, `rigidMotion`, `twist`,
  - 2-ary: `attach`, `space`, `overlay`,
  - n-ary: `makeArray`, `attachSeries`, `spaceSeries`.
  - Compositions: `helix`, `collagen` — these are nested operations.
- The composition (nesting) is straightforward.
  - You keep building materials of higher and higher complexity.
  - And then putting the results together (using the above commands).
  - The result is a new building block of higher complexity.

Sample architectures

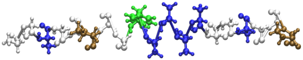
a

Strand1 = chain(seq1)



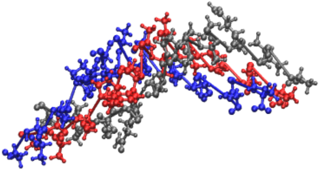
b

Hel1 = helix(Strand1, 1.0, 5.0)



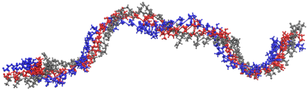
c

TH = collagen(Strand1, Strand2)



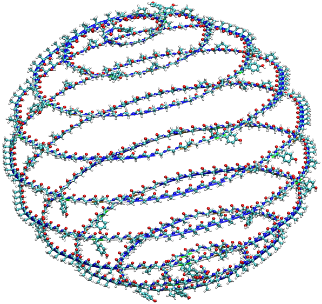
d

Worm = twist(attachSeries(TH,5), W)



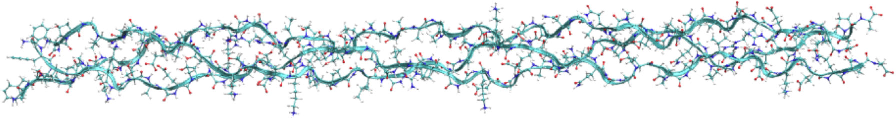
e

Apple = twist(Strand3, SSFunc)



Example of materials architecture: collagen

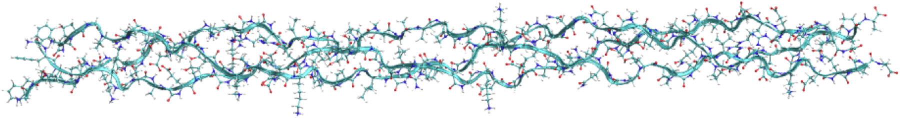
- Collagen is the most common protein in mammals.
- Its design is hierarchical.



a1	=	chain(seq1)
a2	=	chain(seq2)
hel1	=	helix(a1, rad=1.5, pitch=9.5, handed=L)
hel2	=	helix(a2, rad=1.5, pitch=9.5, handed=L)
helhel1	=	helix(hel1, rad=4, pitch=85, handed=R)
helhel2	=	helix(hel2, rad=4, pitch=85, handed=R)
helhel1rot	=	rigidMotion(helhel1, rotate=120, shift=2.8)
helhel2rot	=	rigidMotion(helhel2, rotate=240, shift=-5.6)
tropocollagen	=	overlay(helhel1, helhel1rot, helhel2rot)
collagen	=	makeArray(tropocollagen,1000,1000,distance=8.1)

Example of materials architecture: collagen

- A fibril of collagen is an array of tropocollagen molecules.
- Each molecule of tropocollagen is a right-handed triple helix.
- Each of its three strands is a left-handed helix.
- Each of these individual helices is a chain of many amino acids.



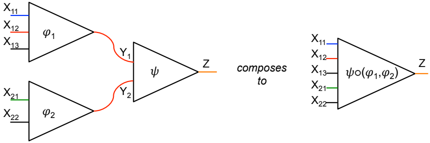
a1	=	chain(seq1)
a2	=	chain(seq2)
hel1	=	helix(a1, rad=1.5, pitch=9.5, handed=L)
hel2	=	helix(a2, rad=1.5, pitch=9.5, handed=L)
helhel1	=	helix(hel1, rad=4, pitch=85, handed=R)
helhel2	=	helix(hel2, rad=4, pitch=85, handed=R)
helhel1rot	=	rigidMotion(helhel1, rotate=120, shift=2.8)
helhel2rot	=	rigidMotion(helhel2, rotate=240, shift=-5.6)
tropocollagen	=	overlay(helhel1, helhel1rot, helhel2rot)
collagen	=	makeArray(tropocollagen,1000,1000,distance=8.1)

Matriarch as a design tool

$$\texttt{attachSeries}\Big(\texttt{helix}(\mathit{seq},\ \mathrm{rad}{=}4,\ \mathrm{pitch}{=}85), \texttt{copies} = 10\Big)$$

- We already said:
  - With Matriarch, it is easy to adjust protein material architecture.
  - Equilibration times are drastically reduced.
  - The equilibration is controlled: no wrong foldings.
- Just as important: The result is a human-understandable structure.
  - A set of descriptive commands to synthesize the material.
  - “Carve nature at its joints.”
  - This, instead of a list of atomic coordinates, or a prose description.
  - Provides a good position from which to consider material design.
- Note: this includes parametric design, but not limited to it.
  - One optimizes a given product (“what’s the best seq, rad, pitch?”)
  - But hierarchical continuation is key: use it as a part in a bigger whole.

What did operads really do for us?



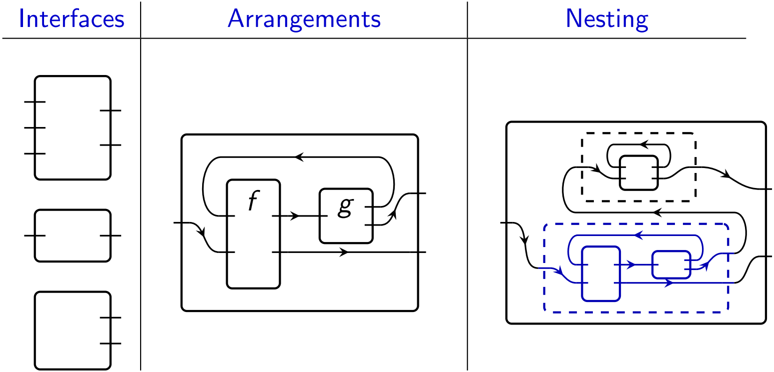
- Operads provided a design framework.
  - The Matriarch operad served as software specification for the program.
  - It efficiently translated user requirements into functional requirements.
  - Later change requests were easy to implement: the formalism is flexible.
- Category theory as a mathematical software specification.
  - The Matriarch program itself is neither exceptional nor unusual.
  - The operad / algebra formalism can serve as a mathematical standard.
  - It fits a wide range of applications.
- What might be new: operad functors
  - Functors as formal translators between different design environments.
  - Operadically-designed tools can be linked using such functors.

Outline

A zoo of operads

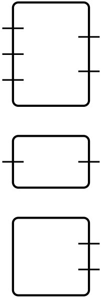
- There's a whole zoo of operads—very different animals.
  - The operad of networks looks pretty different from that of materials.
  - One involved wiring diagrams, the other involved `attach` and `twist`.
- The reason is that operads are just the rules of modularity.
  - If you can tell me your interfaces, arrangements, and nesting,
  - you probably have an operad.
  - Modularity is a very general phenomenon; it takes on many forms.
- Luckily, unlike in zoology, we have an excellent formalism for comparing these animals.
  - Comparing things is what category theory is all about.
- Even just for wiring diagrams, there's an interrelated sub-zoo.

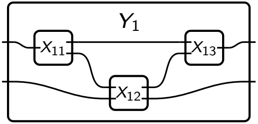
Directed wiring diagrams are modular

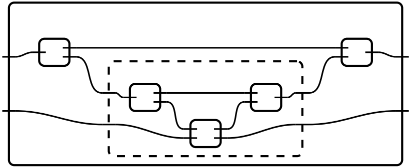


And another: wiring diagrams without feedback

Interfaces	Arrangements	Nesting



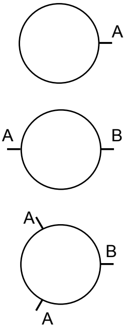


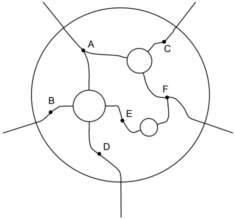


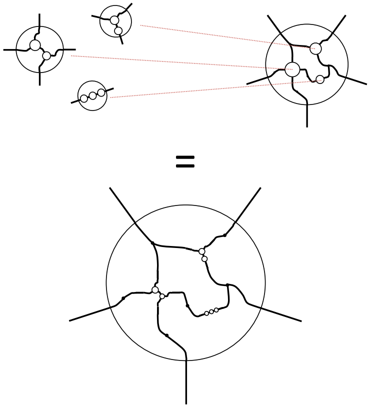
(Getting a sense of how fractals are a special case?)

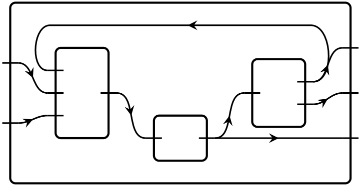
Another modular notion of wiring diagram

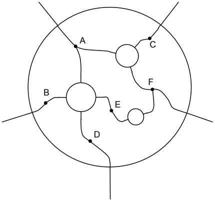
Interfaces	Arrangements	Nesting







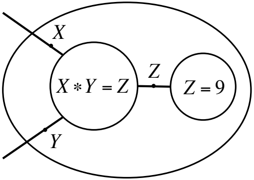




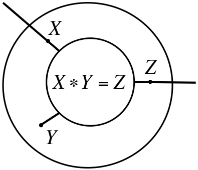
- Two operads, $\mathcal{S}$ and $\mathcal{T}$, whose morphisms look like wiring diagrams.
  - I'm hiding the actual mathematical definitions of these operads.
  - But these pictures correspond to formal mathematical objects.
- There is an operad functor $\mathcal{T} \to \mathcal{S}$.
  - Basically, this is done by turning rectangles to circles.
  - For example, the diagram on the left becomes that on the right.
  - Every object and morphism in $\mathcal{T}$ turns into one in $\mathcal{S}$.
  - This means the semantics of $\mathcal{S}$ can be imported to $\mathcal{T}$.

Databases

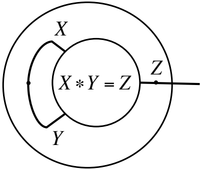
Here's how to use the "circle" operad to design database queries.



"all pairs of integers (*X*, *Y*) whose product is 9"



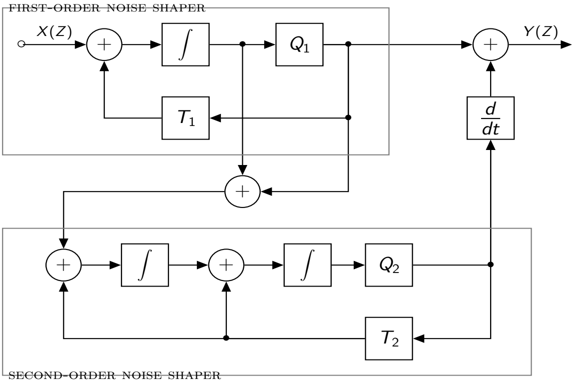
"all pairs of integers (*X*,*Z*) in which *Z* is divisible by *X*."



"all perfect squares *Z*"

Electrical circuits

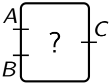
- Same kind of diagram;[2] very different semantics.



- See Baez and Fong: `http://arxiv.org/pdf/1504.05625v1.pdf`.

[2] Drawn by: Ramón Jaramillo. `http://www.texample.net/tikz/examples/noise-shaper/`

Open dynamical systems



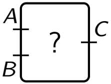
Let inp and outp be manifolds. (In the above, think: $\texttt{inp} = A \times B$ and $\texttt{outp} = C$.)

Definition

An $(\texttt{inp}, \texttt{outp})$-*dynamical system* $X = (Q, f, g)$ consists of

- a manifold Q, called the *state manifold* of X,
- an equation $\frac{\partial Q}{\partial t} := f(Q, \texttt{inp})$, where f is smooth, the *control* function,
- an equation $\texttt{outp} := g(Q)$, where g is smooth, the *readout* function.

Open dynamical systems



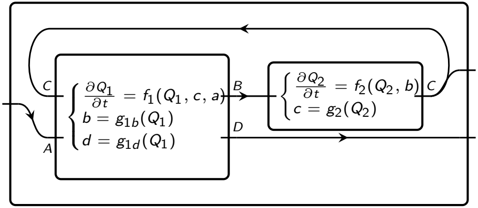
Let inp and outp be manifolds. (In the above, think: $\mathtt{inp} = A \times B$ and $\mathtt{outp} = C$.)

Definition

An $(\mathtt{inp}, \mathtt{outp})$-*dynamical system* $X = (Q, f, g)$ consists of

- a manifold Q, called the *state manifold* of X,
- an equation $\frac{\partial Q}{\partial t} := f(Q, \mathtt{inp})$, where f is smooth, the *control* function,
- an equation $\mathtt{outp} := g(Q)$, where g is smooth, the *readout* function.

Open dynamical systems



Definition

An $(\mathtt{inp}, \mathtt{outp})$-*dynamical system* $X = (Q, f, g)$ consists of

- a manifold Q, called the *state manifold* of X,
- an equation $\frac{\partial Q}{\partial t} := f(Q, \mathtt{inp})$, where f is smooth, the *control* function,
- an equation $\mathtt{outp} := g(Q)$, where g is smooth, the *readout* function.

A matriarch-style program for dynamical systems

- Example: your computer is a dynamical system.
  - Instead of amino acids, it's built from transistors.
  - A computer's complexity is found in the arrangement of transistors.
  - To get there, you make logic gates, adder circuits, registers, etc.
- What can you do with the operad for arranging dynamical systems?
  - Put together dynamical systems as components of larger system.
  - For example, Simulink, Modelica, etc.
  - The operad would be a mathematical ("open source") language.

Outline

Conclusion

- Somehow, the human brain handles a huge range of problems.
  - Planning a wedding or a space mission.
  - Assembling Ikea furniture or architecting a house.
  - Understanding societies, or individual biology or psychology.
- In each case, the understanding comes from putting pieces together.
- There is a certain principle at work across many scales and domains.
  - Each system emerges out of interactions among its parts.
  - Parts can be chunked into sub-systems, which are again parts.
- Operads provide a language in which to consider such issues.
  - As a mathematical language, it can serve as a standard.
  - Many incarnations: many modular environments = many operads.
  - Functors provide translations between modular environments.

Thank you!