Some Applications of Category Theory

David I. Spivak

Mathematics Department
Massachusetts Institute of Technology

December 12, 2017

The need for organizational frameworks

Consider: "organizational frameworks for thinking"

- Math is itself a bunch of such frameworks
  - Arithmetic, algebra, calculus, probability
  - Frameworks for counting, operating, perturbing, and guessing

The need for organizational frameworks

Consider: "organizational frameworks for thinking"

- Math is itself a bunch of such frameworks
  - Arithmetic, algebra, calculus, probability
  - Frameworks for counting, operating, perturbing, and guessing
- Science and engineering disciplines need such frameworks
  - Math used throughout science and engineering
  - Required for *communication at scale*

The need for organizational frameworks

Consider: "organizational frameworks for thinking"

- Math is itself a bunch of such frameworks
  - Arithmetic, algebra, calculus, probability
  - Frameworks for counting, operating, perturbing, and guessing
- Science and engineering disciplines need such frameworks
  - Math used throughout science and engineering
  - Required for *communication at scale*
- New math can be invented for thinking about todays world:
  - Frameworks for thinking about multi-level systems
  - Ways of translating information from one field to another
  - I propose that category theory can work well for this.

A brief history of category theory (CT)

- CT was invented in the 1940s by Mac Lane and Eilenberg.
  - Goal: to connect algebra and topology
  - Rigorous bridges allow the two subjects to communicate
  - Theorems from algebra could be imported into topology

A brief history of category theory (CT)

- CT was invented in the 1940s by Mac Lane and Eilenberg.
  - Goal: to connect algebra and topology
  - Rigorous bridges allow the two subjects to communicate
  - Theorems from algebra could be imported into topology
- Highly successful:
  - Grothendieck used CT to prove conjectures in number theory
  - Has since transformed much of pure math research
  - Has created an "information superhighway" of math subjects

A brief history of category theory (CT)

- CT was invented in the 1940s by Mac Lane and Eilenberg.
  - Goal: to connect algebra and topology
  - Rigorous bridges allow the two subjects to communicate
  - Theorems from algebra could be imported into topology
- Highly successful:
  - Grothendieck used CT to prove conjectures in number theory
  - Has since transformed much of pure math research
  - Has created an "information superhighway" of math subjects
- What about outside of math?

CT Outside of math

- CT has been successful outside of math too:
  - Physics: Quantum processes
  - CS: Programming languages
  - Commerce: Information integration
  - Engineering: Modeling and design

CT Outside of math

- CT has been successful outside of math too:
  - Physics: Quantum processes
  - CS: Programming languages
  - Commerce: Information integration
  - Engineering: Modeling and design
- Identified by NIST as a potential standard for interoperable systems
- Used by Verizon, Facebook, Airbus, Dessault, BAE, ...

CT Outside of math

- CT has been successful outside of math too:
  - Physics: Quantum processes
  - CS: Programming languages
  - Commerce: Information integration
  - Engineering: Modeling and design
- Identified by NIST as a potential standard for interoperable systems
- Used by Verizon, Facebook, Airbus, Dessault, BAE, ...

Why? What does it really do?

Main focus of CT

CT focuses on relationships and composition.

- Relationships govern everything
  - In math, functions are relationships between sets
  - (Higher level): vector spaces are related to abelian groups

Main focus of CT

CT focuses on relationships and composition.

- Relationships govern everything
  - In math, functions are relationships between sets
  - (Higher level): vector spaces are related to abelian groups
  - In CS, programs are relationships between datatypes
  - (Higher level): Java can be compiled to machine code

Main focus of CT

CT focuses on relationships and composition.

- Relationships govern everything
  - In math, functions are relationships between sets
  - (Higher level): vector spaces are related to abelian groups
  - In CS, programs are relationships between datatypes
  - (Higher level): Java can be compiled to machine code
- Like a symphony, complex things are composed from simpler pieces.
  - Relationships between notes, relationships between instruments
  - CT gives a very general notion of composition.
  - "Composition" is the subject of the talk.

Plan of the talk

I'll discuss CT as mathematics for organizing thought.

- Main thrust: "composition", putting things together
- Formal notion: *operads*, a general framework for composition
  - I'll give examples and sketch a definition.
  - Operads: the mathematical framework for the talk

Plan of the talk

I'll discuss CT as mathematics for organizing thought.

- Main thrust: "composition", putting things together
- Formal notion: *operads*, a general framework for composition
  - I'll give examples and sketch a definition.
  - Operads: the mathematical framework for the talk
- Application of CT to studying systems:
  - Systems of equations: a new numerical method
  - Dynamical systems: compositional analyses

Plan of the talk

I'll discuss CT as mathematics for organizing thought.

- Main thrust: "composition", putting things together
- Formal notion: *operads*, a general framework for composition
  - I'll give examples and sketch a definition.
  - Operads: the mathematical framework for the talk
- Application of CT to studying systems:
  - Systems of equations: a new numerical method
  - Dynamical systems: compositional analyses

A lot of information coming: try to get the overarching storyline.

What I mean by composition

Composition is arranging many objects together to make one object.

What I mean by composition

Composition is arranging many objects together to make one object.

If a situation has any sort of composition, maybe you have an operad.

What I mean by composition

Composition is arranging many objects together to make one object.

If a situation has any sort of composition, maybe you have an operad.

An operad consists of:

- A collection of objects $X, Y, \ldots,$
- And ways to arrange them, $\varphi\colon X_1, \ldots, X_k \to Y,$
- Such that arrangements can be nested inside each other.

What I mean by composition

Composition is arranging many objects together to make one object.

If a situation has any sort of composition, maybe you have an operad.

An operad consists of:

- A collection of objects $X, Y, \ldots,$
- And ways to arrange them, $\varphi\colon X_1, \ldots, X_k \to Y,$
- Such that arrangements can be nested inside each other.

Slightly more formal definition to come.

Operads are everywhere

Operads are used unconsciously in many fields.

- Electrical engineering: "wiring diagrams".
- Design: "set-based design".
- Computer programming: "data flow".
- Natural language processing: "grammars".
- Materials science: "hierarchical materials".

Operads are everywhere

Operads are used unconsciously in many fields.

- Electrical engineering: "wiring diagrams".
- Design: "set-based design".
- Computer programming: "data flow".
- Natural language processing: "grammars".
- Materials science: "hierarchical materials".

Rallying cry: *Let's bring operads to the fore.*

- There's a common theme in the way we think.
- Operads structure and organize this sort of thinking.
- With mathematical structure, we can go much further.

Operads are everywhere

Operads are used unconsciously in many fields.

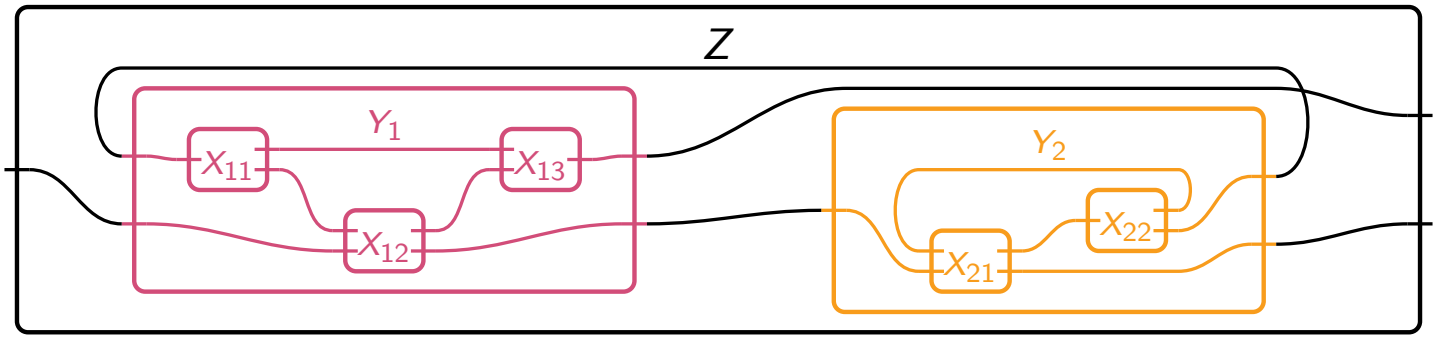
- Electrical engineering: "wiring diagrams".
- Design: "set-based design".
- Computer programming: "data flow".
- Natural language processing: "grammars".
- Materials science: "hierarchical materials".

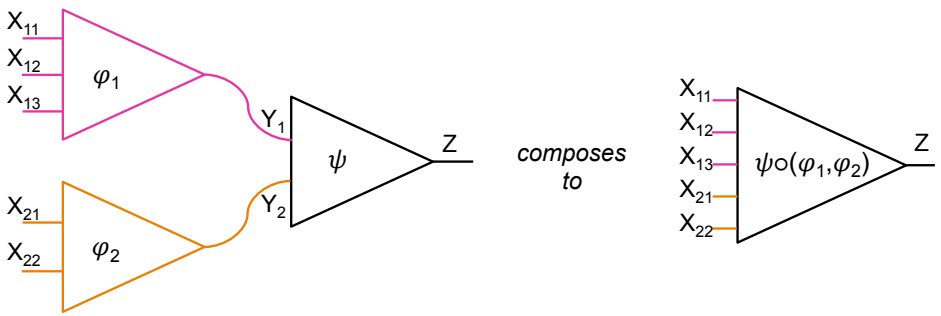
Rallying cry: *Let's bring operads to the fore.*

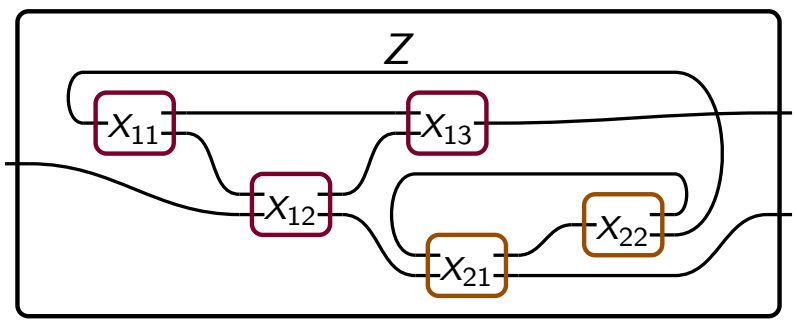
- There's a common theme in the way we think.
- Operads structure and organize this sort of thinking.
- With mathematical structure, we can go much further.

Let's look for objects, arrangements, and nesting in some examples.

Operad 1: wiring diagrams

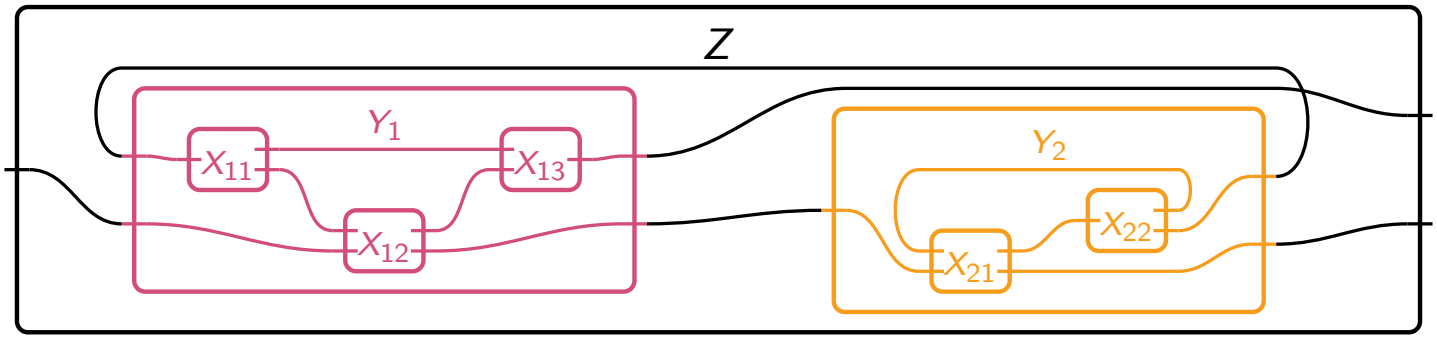


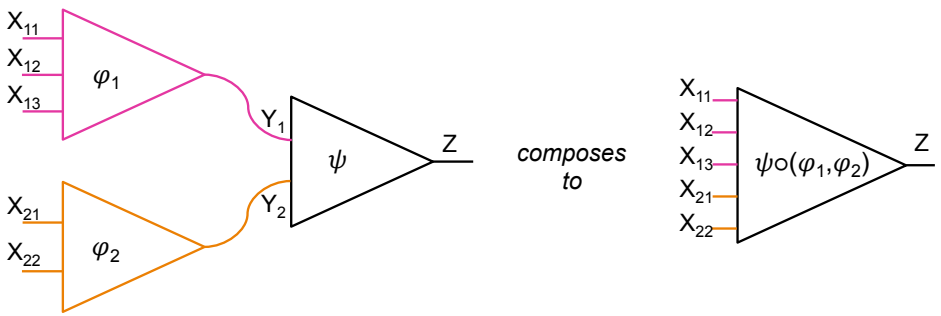


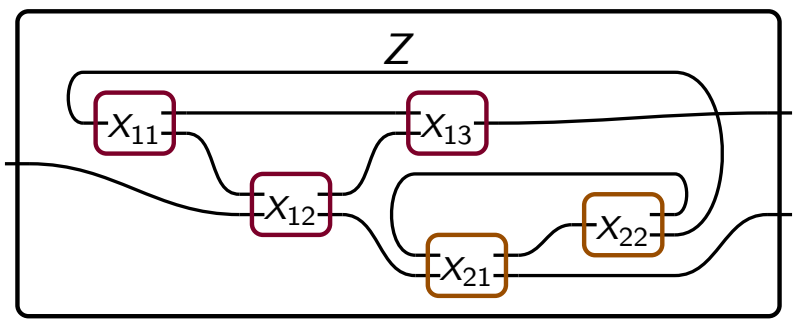


Objects: boxes with ports.

Operad 1: wiring diagrams

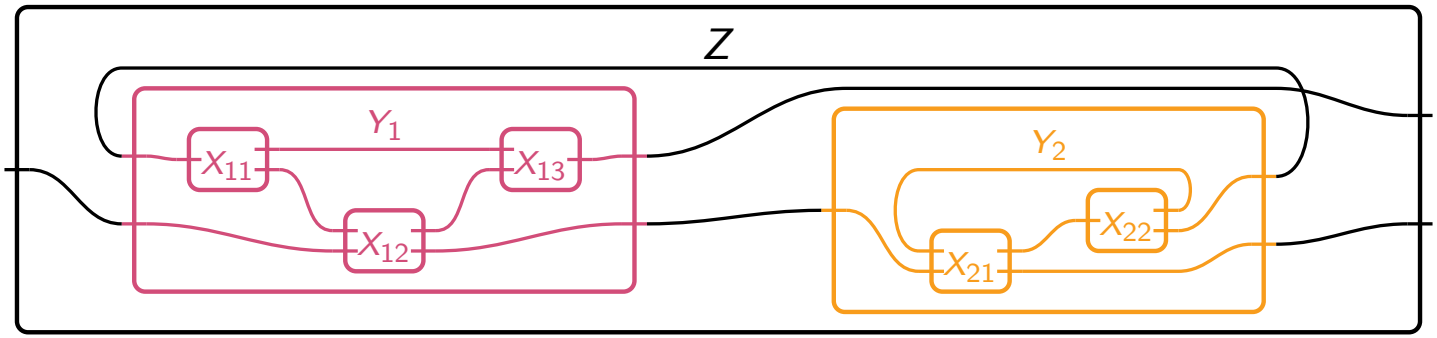


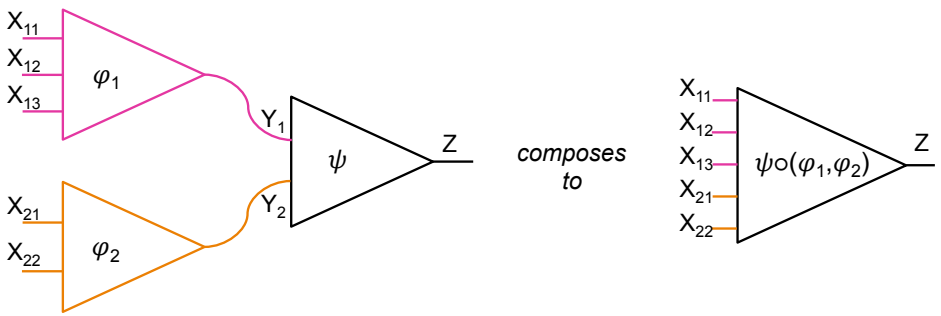


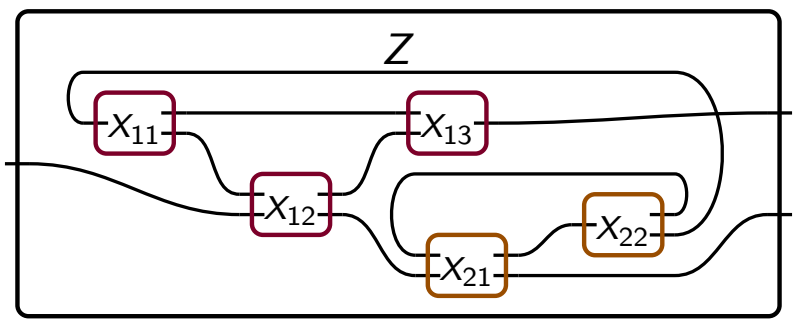


Objects: boxes with ports. Arrangements: wiring diagrams.

Operad 1: wiring diagrams







Objects: boxes with ports. Arrangements: wiring diagrams. Nesting: nesting.

Formal definition of operad

An operad $\mathcal{O}$ consists of

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

$$\mathrm{Mor}_{\mathcal{O}}(X_1, \ldots, X_k; Y)$$

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

$$\varphi \colon X_1, \ldots, X_k \to Y.$$

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

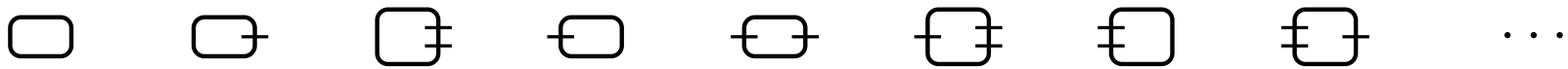
$$\psi \circ (\varphi_1, \ldots, \varphi_k) \colon (X_{i,j}) \xrightarrow{\varphi_i} (Y_i) \xrightarrow{\psi} Z.$$

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

Operad 1: WDs again

An operad $\mathcal{W}$ for composing wiring diagrams:

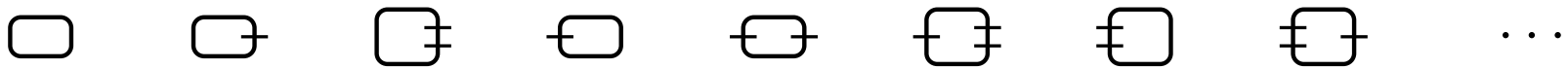
- Object $X \in \mathcal{W}$: any possible box-with-ports.



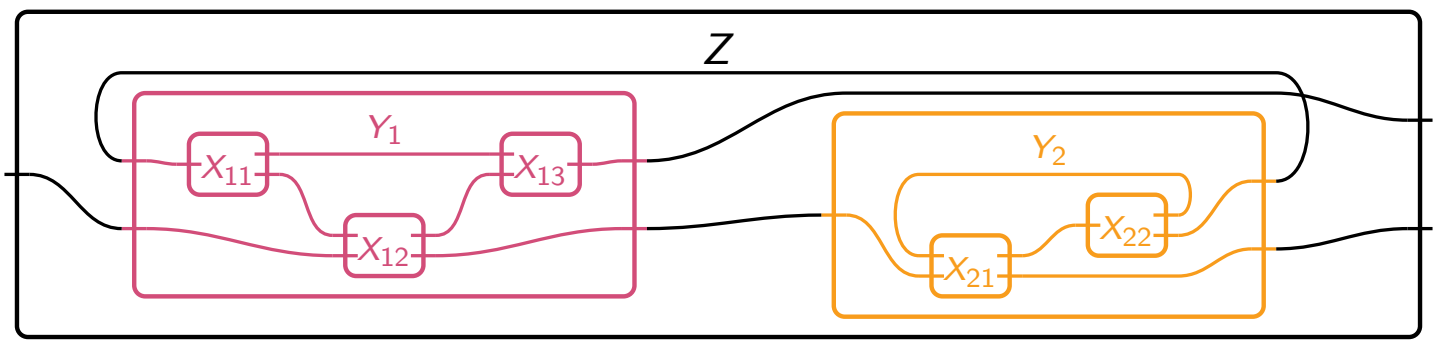
Operad 1: WDs again

An operad $\mathcal{W}$ for composing wiring diagrams:

- Object $X \in \mathcal{W}$: any possible box-with-ports.



- Arrangement $\phi\colon X_1, \ldots, X_k \to Y$ in $\mathcal{W}$: any wiring of X's in Y.
- Nesting: the facts about how wiring diagrams fit inside each other.



One operad $\mathcal{W}$ comprises all this.

Operad 2: hierarchical protein materials

There is an operad $\mathcal{M}$ for composing hierarchical protein materials.

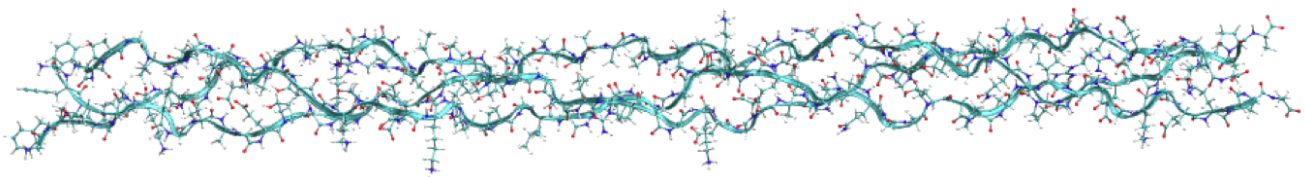
- A protein is an arrangement of simpler proteins.
  - There are "atomic" proteins: amino acids.
  - Protein materials include your skin: stretchable, breathable, waterproof.
  - Materials scientists would *love* to make materials like this.

[1] Giesa, T.; Jagadeesan, R.; Spivak, D.I.; Buehler, M.J. (2015) "Matriarch: a Python library for materials architecture." *ACS Biomaterials Science & Engineering*.

Operad 2: hierarchical protein materials

There is an operad $\mathcal{M}$ for composing hierarchical protein materials.

- A protein is an arrangement of simpler proteins.
  - There are "atomic" proteins: amino acids.
  - Protein materials include your skin: stretchable, breathable, waterproof.
  - Materials scientists would *love* to make materials like this.
- Assemble new proteins from old:
  - arrange in series or parallel (H-bonds), or
  - arrange in helices, double helices, any conceivable curve, etc.



- Collagen has a nested structure: it is an array, each fiber of which is a triple helix, each strand of which is a helix, each unit of which is an amino acid.[1]

[1] Giesa, T.; Jagadeesan, R.; Spivak, D.I.; Buehler, M.J. (2015) "Matriarch: a Python library for materials architecture." *ACS Biomaterials Science & Engineering*.

More operads: Grammars and recipes

Context-free grammars are "free" operads.

⟨sentence⟩	::=	⟨noun-phrase⟩ ⟨verb-phrase⟩
⟨noun-phrase⟩	::=	⟨pronoun⟩ | ⟨proper-noun⟩ | ⟨determiner⟩ ⟨nominal⟩
⟨nominal⟩	::=	⟨noun⟩ | ⟨noun⟩ ⟨nominal⟩
⟨verb-phrase⟩	::=	⟨verb⟩ | ⟨verb⟩ ⟨noun-phrase⟩ | ⟨verb⟩ ⟨prep-phrase⟩
⟨prep-phrase⟩	::=	⟨preposition⟩ ⟨noun-phrase⟩

More operads: Grammars and recipes

Context-free grammars are "free" operads.

⟨sentence⟩	::=	⟨noun-phrase⟩ ⟨verb-phrase⟩
⟨noun-phrase⟩	::=	⟨pronoun⟩ | ⟨proper-noun⟩ | ⟨determiner⟩ ⟨nominal⟩
⟨nominal⟩	::=	⟨noun⟩ | ⟨noun⟩ ⟨nominal⟩
⟨verb-phrase⟩	::=	⟨verb⟩ | ⟨verb⟩ ⟨noun-phrase⟩ | ⟨verb⟩ ⟨prep-phrase⟩
⟨prep-phrase⟩	::=	⟨preposition⟩ ⟨noun-phrase⟩

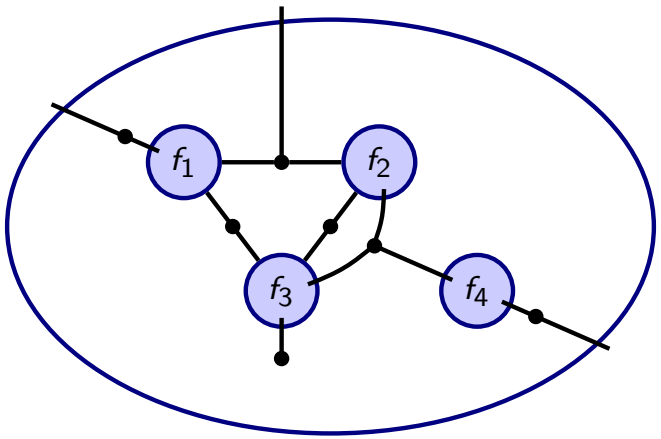
Recipes are also operads.

- Combine sub-recipes to make a recipe.
- The outline for this talk as an operad:
  - Objects: points I want to make.
  - Arrangements: putting points together to make a bigger point.
  - Nesting: the well-known outline structure.

Grammars and recipes organize communication and action.

Quickly: an operad we'll see later

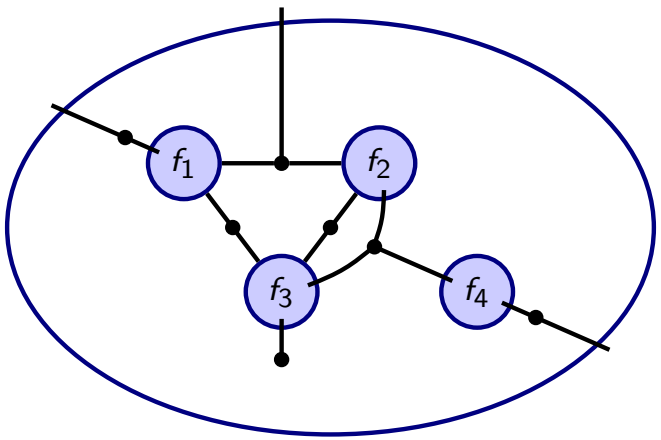
Another sort of wiring diagram (WD):



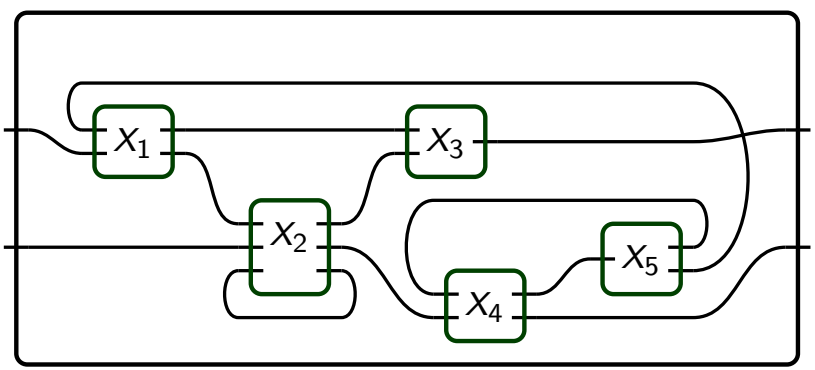
- This operad is an unoriented version of the first.

Quickly: an operad we'll see later

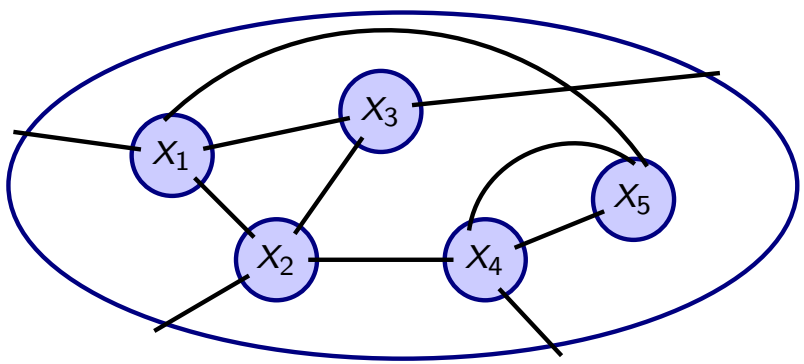
Another sort of wiring diagram (WD):



- This operad is an unoriented version of the first.
- Any oriented WD can be converted to an unoriented one.



$\rightsquigarrow$



Operads and their algebras

One more formal notion to discuss: algebras.

Operads and their algebras

One more formal notion to discuss: algebras.

Rules of composition vs. stuff being composed.

Operads and their algebras

One more formal notion to discuss: algebras.

Rules of composition vs. stuff being composed.

- Operad : Group theory :: Algebras : Groups.
- Operad : Ring theory :: Algebras : Rings.

Operads and their algebras

One more formal notion to discuss: algebras.

Rules of composition vs. stuff being composed.

- Operad : Group theory :: Algebras : Groups.
- Operad : Ring theory :: Algebras : Rings.

Operad = theory. Algebras = models.

Each operad has many algebras

Each operad $\mathcal{O}$ is a "theory of composition".

- Objects $X, Y, \ldots$: what *sorts* of elements in this theory?
- Arrangements ϕ, ψ: what are the operations?
- Nesting: what kind of laws?

Each operad has many algebras

Each operad $\mathcal{O}$ is a "theory of composition".

- Objects $X, Y, \ldots$: what *sorts* of elements in this theory?
- Arrangements ϕ, ψ: what are the operations?
- Nesting: what kind of laws?

The $\mathcal{O}$-algebras are the models of theory $\mathcal{O}$.[2]

- An $\mathcal{O}$-algebra A says what's actually being composed.
  - To each object X: a set $A(X)$ of elements.
  - To each arrangement ϕ: an k-ary operation $A(\phi)$.
    - If $\phi\colon X_1, \ldots, X_k \to Y$ is an arrangement,
    - Then $A(\phi)\colon A(X_1) \times \cdots \times A(X_k) \to A(Y)$ is a function.
  - To each nesting: a law in A.

[2] Technically, an algebra is a functor $A\colon \mathcal{O} \to \mathbf{Set}$.

Each operad has many algebras

Each operad $\mathcal{O}$ is a "theory of composition".

- Objects $X, Y, \ldots$: what *sorts* of elements in this theory?
- Arrangements ϕ, ψ: what are the operations?
- Nesting: what kind of laws?

The $\mathcal{O}$-algebras are the models of theory $\mathcal{O}$.[2]

- An $\mathcal{O}$-algebra A says what's actually being composed.
  - To each object X: a set $A(X)$ of elements.
  - To each arrangement ϕ: an k-ary operation $A(\phi)$.
    - If $\phi\colon X_1, \ldots, X_k \to Y$ is an arrangement,
    - Then $A(\phi)\colon A(X_1) \times \cdots \times A(X_k) \to A(Y)$ is a function.
  - To each nesting: a law in A.

Next, I'll explain what algebras on an operad look like.

[2] Technically, an algebra is a functor $A\colon \mathcal{O} \to \mathbf{Set}$.

Example: the operad for monoids

Monoids are groups without inverses.

- Any group G is a monoid.
- Natural numbers $\mathbb{N}$.
- $(n \times n)$-matrices form a monoid under multiplication.

Example: the operad for monoids

Monoids are groups without inverses.

- Any group G is a monoid.
- Natural numbers $\mathbb{N}$.
- $(n \times n)$-matrices form a monoid under multiplication.

There is one operad $\mathcal{O}$ whose algebras are monoids.

Example: the operad for monoids

Monoids are groups without inverses.

- Any group G is a monoid.
- Natural numbers $\mathbb{N}$.
- $(n \times n)$-matrices form a monoid under multiplication.

There is one operad $\mathcal{O}$ whose algebras are monoids.

$\mathcal{O}$ has one object $\bullet$ and one k-ary morphism for each k: "*".

Example: the operad for monoids

Monoids are groups without inverses.

- Any group G is a monoid.
- Natural numbers $\mathbb{N}$.
- $(n \times n)$-matrices form a monoid under multiplication.

There is one operad $\mathcal{O}$ whose algebras are monoids.

$\mathcal{O}$ has one object $\bullet$ and one k-ary morphism for each k: "$*$".

- To the object $\bullet$, a set $A(\bullet)$.
- To the k-ary morphism, a function $A(*)\colon A(\bullet) \times \cdots \times A(\bullet) \to A(\bullet)$.
  - Multiply k-many elements of a group G.
  - Add k-many numbers.
  - Multiply k-many $(n \times n)$-matrices.

Example: the operad for monoids

Monoids are groups without inverses.

- Any group G is a monoid.
- Natural numbers $\mathbb{N}$.
- $(n \times n)$-matrices form a monoid under multiplication.

There is one operad $\mathcal{O}$ whose algebras are monoids.

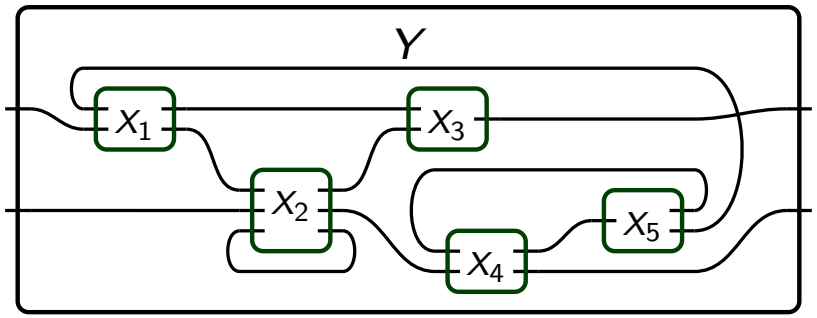
$\mathcal{O}$ has one object $\bullet$ and one k-ary morphism for each k: "$*$".

- To the object $\bullet$, a set $A(\bullet)$.
- To the k-ary morphism, a function $A(*)\colon A(\bullet) \times \cdots \times A(\bullet) \to A(\bullet)$.
  - Multiply k-many elements of a group G.
  - Add k-many numbers.
  - Multiply k-many $(n \times n)$-matrices.
- The operad laws (unitality, associativity) guarantee the monoid laws.

Takeaway: The simplest operad is the one for monoids.

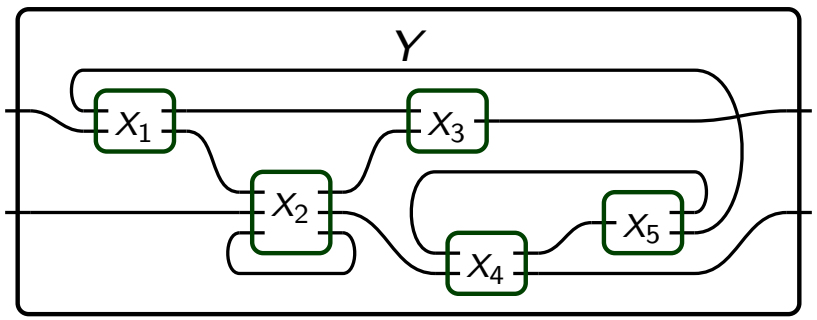
Algebras for wiring diagrams operad

Recall operad $\mathcal{W}$ of all boxes X and WDs, $\phi\colon X_1, \ldots, X_k \to Y$.



Algebras for wiring diagrams operad

Recall operad $\mathcal{W}$ of all boxes X and WDs, $\phi\colon X_1, \ldots, X_k \to Y$.

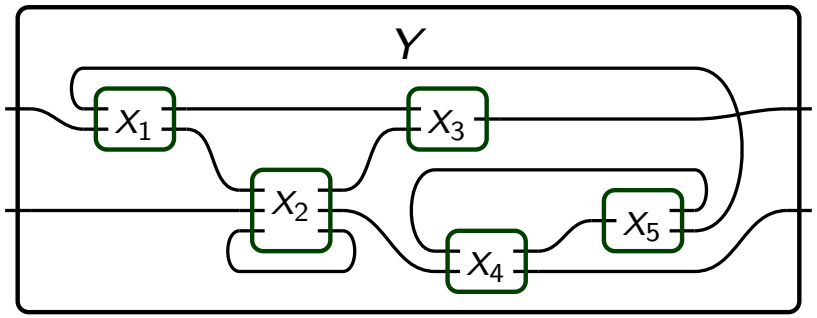


Many different $\mathcal{W}$-algebras $A\colon \mathcal{W} \to \mathbf{Set}$.

- What is an algebra A on $\mathcal{W}$?
- A says what elements can you put in each box.
- A says how a wiring diagram ϕ turns into an operation:

Algebras for wiring diagrams operad

Recall operad $\mathcal{W}$ of all boxes X and WDs, $\phi\colon X_1, \ldots, X_k \to Y$.

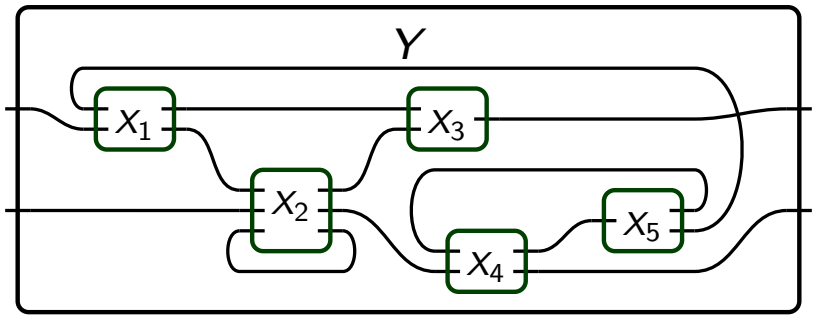


Many different $\mathcal{W}$-algebras $A\colon \mathcal{W} \to \mathbf{Set}$.

- What is an algebra A on $\mathcal{W}$?
- A says what elements can you put in each box.
- A says how a wiring diagram ϕ turns into an operation:
  - Given an element in each internal box X_i
  - The operation returns an element in the outer box Y.
  - This is all the algebra A's doing.

Algebras for wiring diagrams operad

Recall operad $\mathcal{W}$ of all boxes X and WDs, $\phi\colon X_1, \ldots, X_k \to Y$.

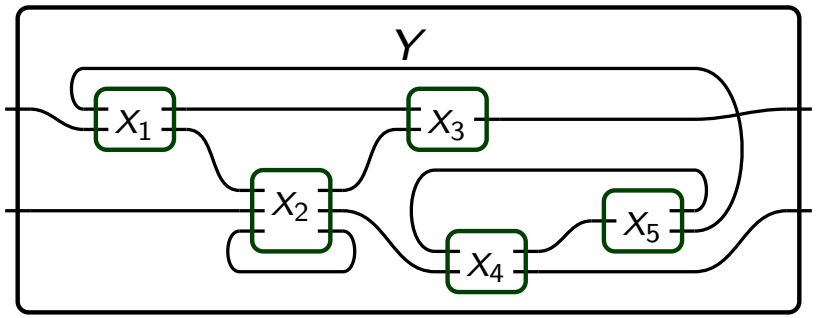


Many different $\mathcal{W}$-algebras $A\colon \mathcal{W} \to \mathbf{Set}$.

- What is an algebra A on $\mathcal{W}$?
- A says what elements can you put in each box.
- A says how a wiring diagram ϕ turns into an operation:
  - Given an element in each internal box X_i
  - The operation returns an element in the outer box Y.
  - This is all the algebra A's doing.

We'll see next that "open dynamical systems" form an algebra.

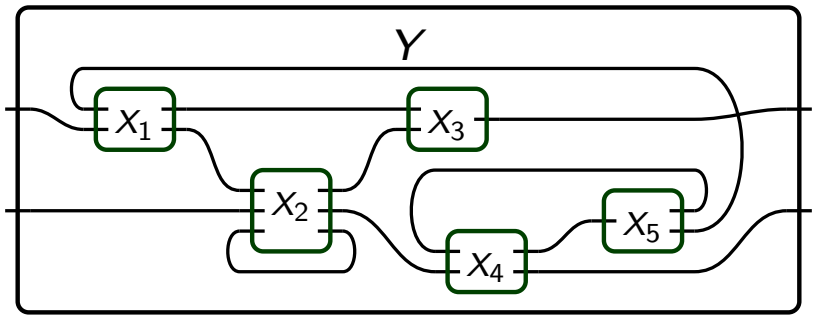
$\mathcal{W}$-algebra 1: open dynamical systems (ODS's)



To each box X: open dynamical systems $=$ machines with ports.

- Let k and ℓ be the number of input and output wires on X.
- X is assigned a state space, $\mathbb{R}^n$.

$\mathcal{W}$-algebra 1: open dynamical systems (ODS's)



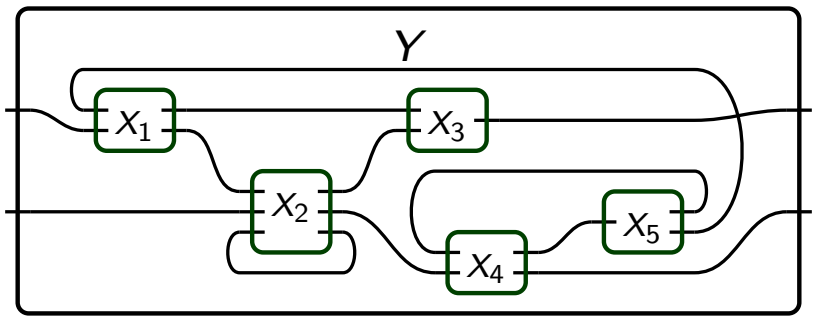
To each box X: open dynamical systems = machines with ports.

- Let k and ℓ be the number of input and output wires on X.
- X is assigned a state space, $\mathbb{R}^n$.
- X is assigned a system of ordinary differential equations:

$$\begin{aligned}
\dot{x}_1 &= f_1(x_1, \ldots, x_n, a_1, \ldots, a_k) & b_1 &= g_1(x_1, \ldots, x_n) \\
\dot{x}_2 &= f_2(x_1, \ldots, x_n, a_1, \ldots, a_k) & b_2 &= g_2(x_1, \ldots, x_n) \\
\vdots &= \vdots & \vdots &= \vdots \\
\dot{x}_n &= f_n(x_1, \ldots, x_n, a_1, \ldots, a_k) & b_\ell &= g_\ell(x_1, \ldots, x_n)
\end{aligned}$$

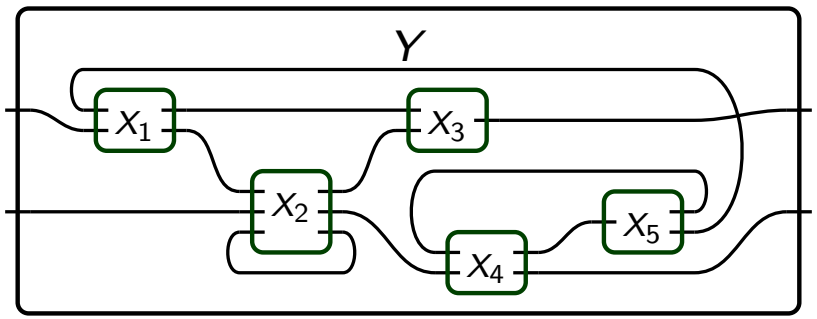
Continuously: change state based on current state and input; send output.

$\mathcal{W}$-algebra 1: open dynamical systems (ODS's)



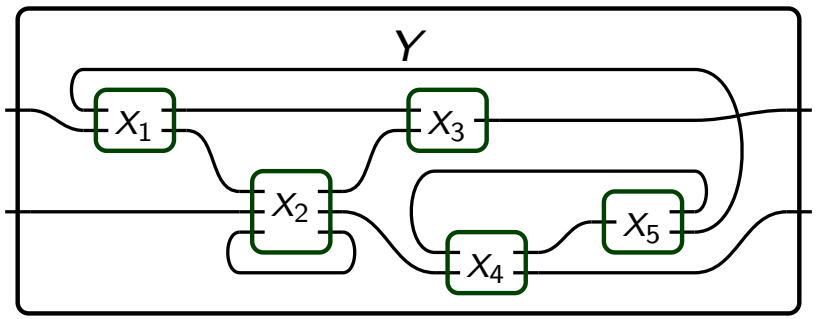
- So to each box X, the "elements" of the algebra are ODS's
- To each wiring diagram, get an "operation": combine ODS's
  - Easy: done by variable substitution
  - Basically *Boyce and Diprima*: tanks, pipes, brine

$\mathcal{W}$-algebra 1: open dynamical systems (ODS's)



- So to each box X, the "elements" of the algebra are ODS's
- To each wiring diagram, get an "operation": combine ODS's
  - Easy: done by variable substitution
  - Basically *Boyce and Diprima*: tanks, pipes, brine
  - Operad algebra solves all possible such problems at once.

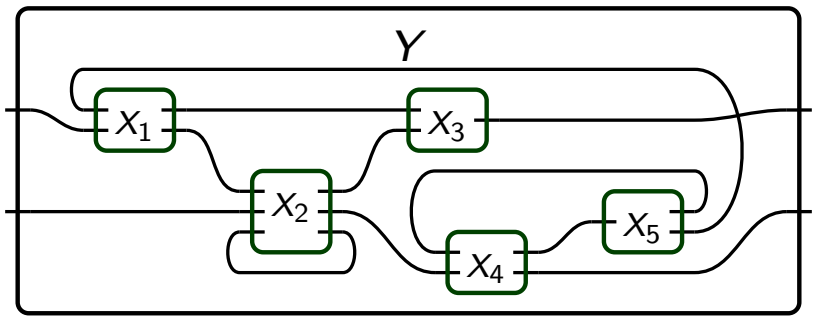
$\mathcal{W}$-algebra 1: open dynamical systems (ODS's)



- So to each box X, the "elements" of the algebra are ODS's
- To each wiring diagram, get an "operation": combine ODS's
  - Easy: done by variable substitution
  - Basically *Boyce and Diprima*: tanks, pipes, brine
  - Operad algebra solves all possible such problems at once.

Operad and algebra: organize thinking about composing dynamical systems.

Flavors of dynamical systems, as $\mathcal{W}$-algebras

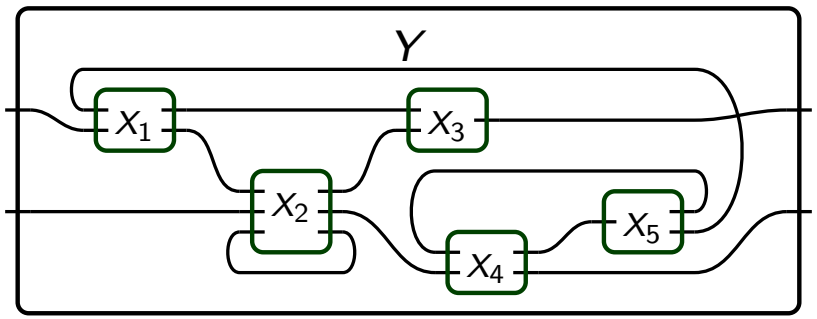


There are many other $\mathcal{W}$-algebras A. Just saw:

- A_1: continuous ODS's (ODE's with time-varying parameters).[3]

[3] ODE: ordinary differential equation. ODS: open dynamical system.

Flavors of dynamical systems, as $\mathcal{W}$-algebras



There are many other $\mathcal{W}$-algebras A. Just saw:

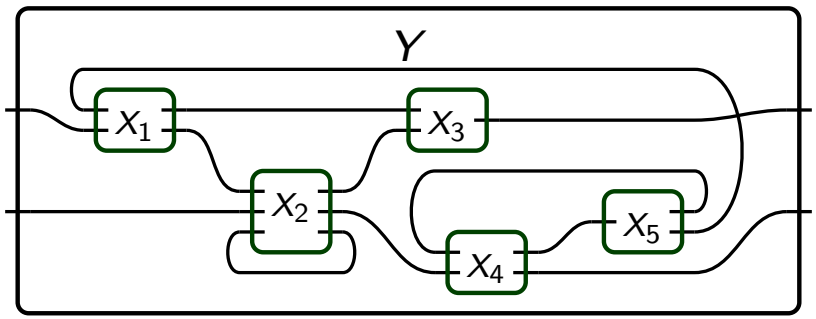
- A_1: continuous ODS's (ODE's with time-varying parameters).[3]

There are other sorts of machines:

- A_2: discrete ODS's (on a discrete clock).
- A_3: hybrid ODS's (cyber-physical systems).

[3] ODE: ordinary differential equation. ODS: open dynamical system.

Flavors of dynamical systems, as $\mathcal{W}$-algebras



There are many other $\mathcal{W}$-algebras A. Just saw:

- A_1: continuous ODS's (ODE's with time-varying parameters).[3]

There are other sorts of machines:

- A_2: discrete ODS's (on a discrete clock).
- A_3: hybrid ODS's (cyber-physical systems).

But there are somewhat different-seeming algebras too.

[3] ODE: ordinary differential equation. ODS: open dynamical system.

What's a relation?

Relations are everywhere.

- Definition: $R \subseteq A_1 \times A_2 \times \cdots \times A_n$.
  - Here R is "an n-ary relation on $A_1, \ldots, A_n$".
  - Out of all the tuples $(a_1, \ldots, a_n)$, R tells us which are "related".
  - We can write R as $\{(a_1, \ldots, a_n) \in A_1 \times \cdots \times A_n \mid \text{property } R\}$.

What's a relation?

Relations are everywhere.

- Definition: $R \subseteq A_1 \times A_2 \times \cdots \times A_n$.
  - Here R is "an n-ary relation on $A_1, \ldots, A_n$".
  - Out of all the tuples $(a_1, \ldots, a_n)$, R tells us which are "related".
  - We can write R as $\{(a_1, \ldots, a_n) \in A_1 \times \cdots \times A_n \mid \text{property } R\}$.
- Relations are the foundation of database theory. Examples:
  - $\{(p, c) \in \mathrm{Person} \times \mathrm{Person} \mid p \text{ is a parent of } c\}$
  - $\{(p, o, p') \in \mathrm{Person} \times \mathrm{Object} \times \mathrm{Person} \mid p \text{ gave } o \text{ to } p'\}$.
  - $\{(p, n) \in \mathbb{N} \times \mathbb{N} \mid p \text{ is a prime factor of } n\}$

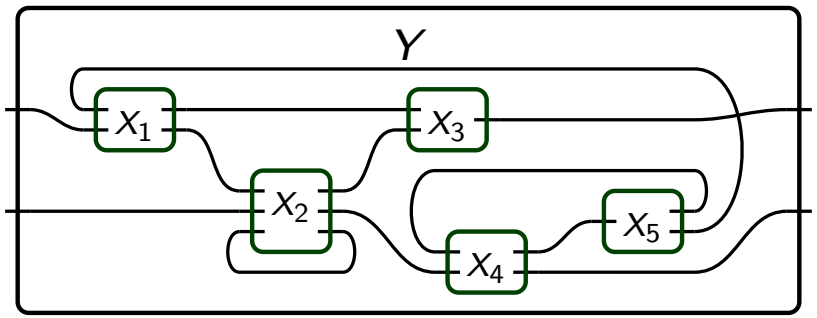
What's a relation?

Relations are everywhere.

- Definition: $R \subseteq A_1 \times A_2 \times \cdots \times A_n$.
  - Here R is "an n-ary relation on $A_1, \ldots, A_n$".
  - Out of all the tuples $(a_1, \ldots, a_n)$, R tells us which are "related".
  - We can write R as $\{(a_1, \ldots, a_n) \in A_1 \times \cdots \times A_n \mid \text{property } R\}$.
- Relations are the foundation of database theory. Examples:
  - $\{(p, c) \in \mathrm{Person} \times \mathrm{Person} \mid p \text{ is a parent of } c\}$
  - $\{(p, o, p') \in \mathrm{Person} \times \mathrm{Object} \times \mathrm{Person} \mid p \text{ gave } o \text{ to } p'\}$.
  - $\{(p, n) \in \mathbb{N} \times \mathbb{N} \mid p \text{ is a prime factor of } n\}$
- They're also subsets of Euclidean spaces:
  - $\{(x, y) \in \mathbb{R} \times \mathbb{R} \mid x^2 + y^2 = 1\}$.
  - $\{x \in \mathbb{R} \mid x \geq 5\}$.

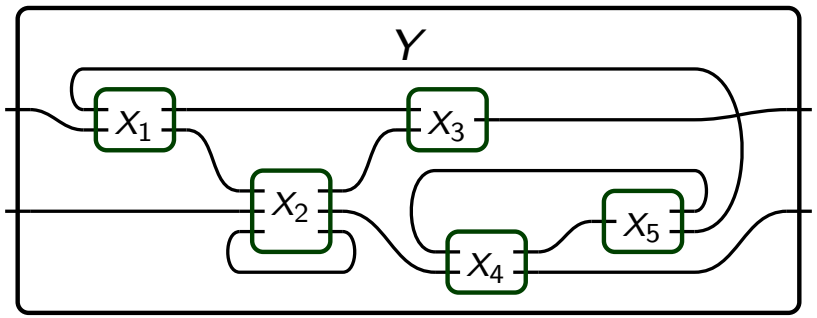
Relations are not dynamic at all, but they still form a $\mathcal{W}$-algebra.

Relations are a $\mathcal{W}$-algebra



Imagine the ports/wires are labeled with sets.

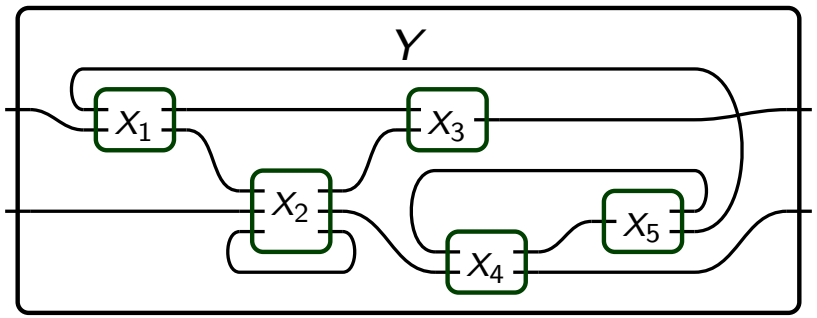
Relations are a $\mathcal{W}$-algebra



Imagine the ports/wires are labeled with sets.

- Suppose a box X has ports labeled by sets $A_1, A_2, \ldots, A_n$.
- Then an "element" for box X is a relation $R \subseteq A_1 \times \cdots \times A_n$.
- To each wiring diagram, get an "operation": simultaneous solution.

Relations are a $\mathcal{W}$-algebra



Imagine the ports/wires are labeled with sets.

- Suppose a box X has ports labeled by sets $A_1, A_2, \ldots, A_n$.
- Then an "element" for box X is a relation $R \subseteq A_1 \times \cdots \times A_n$.
- To each wiring diagram, get an "operation": simultaneous solution.

More on this later. First, let's discuss tensors.

What's a tensor?

Matrix: 2-dimensional array; Tensor: n-dimensional analogue.

What's a tensor?

Matrix: 2-dimensional array; Tensor: n-dimensional analogue.

- For example, consider a $(2 \times 5 \times 2 \times 3)$-tensor T.
  - It is called an order-4 tensor.
  - It has 60 entries, say in $\mathbb{R}$.
  - One of them is $T(2, 3, 1, 2) \in \mathbb{R}$.

What's a tensor?

Matrix: 2-dimensional array; Tensor: n-dimensional analogue.

- For example, consider a $(2 \times 5 \times 2 \times 3)$-tensor T.
  - It is called an order-4 tensor.
  - It has 60 entries, say in $\mathbb{R}$.
  - One of them is $T(2,3,1,2) \in \mathbb{R}$.
- Operations on tensors: Kronecker and contraction.
  - Given tensors T: $d_1 \times d_2$ and U: $d_3 \times d_4 \times d_5$.
  - Kronecker: $T \otimes U$: $d_1 \times d_2 \times d_3 \times d_4 \times d_5$
    - $(T \otimes U)(i_1, i_2, i_3, i_4, i_5) = T(i_1, i_2) * U(i_3, i_4, i_5)$.

What's a tensor?

Matrix: 2-dimensional array; Tensor: n-dimensional analogue.

- For example, consider a $(2 \times 5 \times 2 \times 3)$-tensor T.
  - It is called an order-4 tensor.
  - It has 60 entries, say in $\mathbb{R}$.
  - One of them is $T(2,3,1,2) \in \mathbb{R}$.
- Operations on tensors: Kronecker and contraction.
  - Given tensors T: $d_1 \times d_2$ and U: $d_3 \times d_4 \times d_5$.
  - Kronecker: $T \otimes U$: $d_1 \times d_2 \times d_3 \times d_4 \times d_5$
    - $(T \otimes U)(i_1, i_2, i_3, i_4, i_5) = T(i_1, i_2) * U(i_3, i_4, i_5)$.
  - Contract: can do it if any two dimensions match, say $d_1 = d_4$.
    - Contracting drops out shared dimension $d := d_1 = d_4$.
    - Like matrix multiplication, result is $d_2 \times d_3 \times d_5$:

What's a tensor?

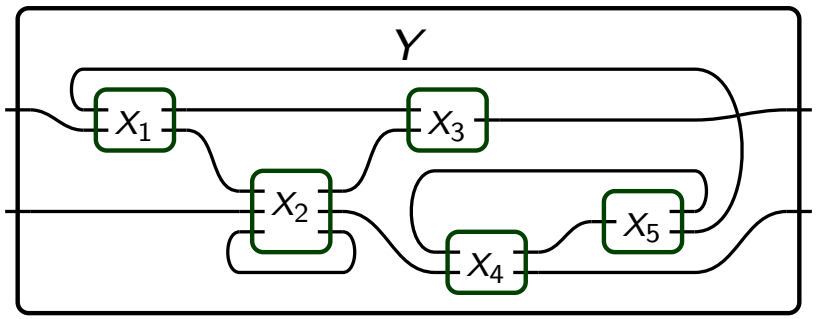
Matrix: 2-dimensional array; Tensor: n-dimensional analogue.

- For example, consider a $(2 \times 5 \times 2 \times 3)$-tensor T.
  - It is called an order-4 tensor.
  - It has 60 entries, say in $\mathbb{R}$.
  - One of them is $T(2,3,1,2) \in \mathbb{R}$.
- Operations on tensors: Kronecker and contraction.
  - Given tensors T: $d_1 \times d_2$ and U: $d_3 \times d_4 \times d_5$.
  - Kronecker: $T \otimes U$: $d_1 \times d_2 \times d_3 \times d_4 \times d_5$
    - $(T \otimes U)(i_1, i_2, i_3, i_4, i_5) = T(i_1, i_2) * U(i_3, i_4, i_5)$.
  - Contract: can do it if any two dimensions match, say $d_1 = d_4$.
    - Contracting drops out shared dimension $d := d_1 = d_4$.
    - Like matrix multiplication, result is $d_2 \times d_3 \times d_5$:

$$V(i_2, i_3, i_5) = \sum_{j=1}^{d} T(j, i_2) * U(i_3, j, i_5)$$

Tensors are a $\mathcal{W}$-algebra

Tensors are not dynamic at all, but they still form a $\mathcal{W}$-algebra.

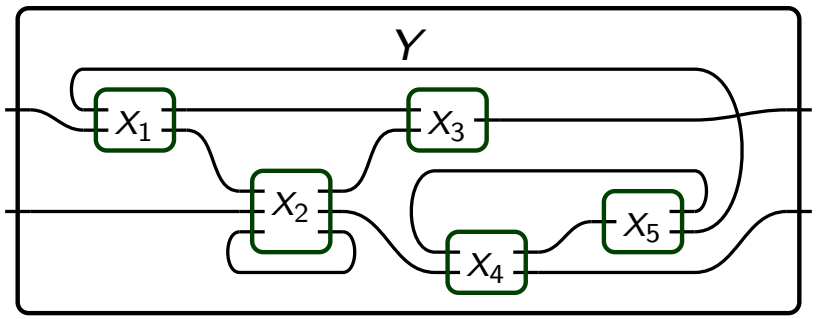


To each box X, the "elements" of the algebra are tensors.

- To each wiring diagram, get an "operation": contract tensors
  - Nothing is "passed" on the wires.
  - Instead, form the new tensor by contracting: sum of products

Tensors are a $\mathcal{W}$-algebra

Tensors are not dynamic at all, but they still form a $\mathcal{W}$-algebra.

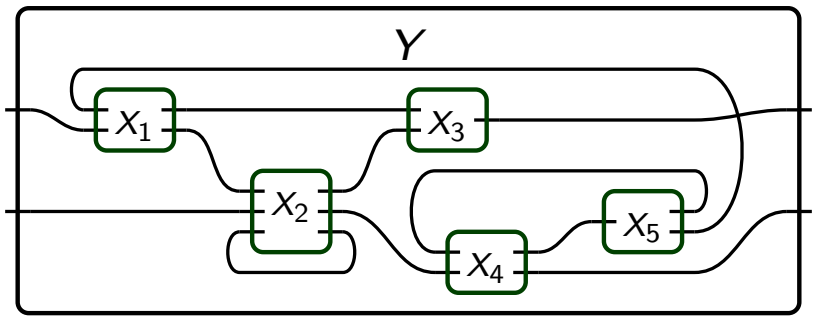


To each box X, the "elements" of the algebra are tensors.

- To each wiring diagram, get an "operation": contract tensors
  - Nothing is "passed" on the wires.
  - Instead, form the new tensor by contracting: sum of products
- This algebra solves all "contract tensor network" problems at once.

Tensors are a $\mathcal{W}$-algebra

Tensors are not dynamic at all, but they still form a $\mathcal{W}$-algebra.



To each box X, the "elements" of the algebra are tensors.

- To each wiring diagram, get an "operation": contract tensors
  - Nothing is "passed" on the wires.
  - Instead, form the new tensor by contracting: sum of products
- This algebra solves all "contract tensor network" problems at once.

Next up: using this algebra for solving general systems of equations.

Detailed look at an application

Separately plot the solutions to equations: $f(x, w) = 0$ and $g(w, y) = 0$.

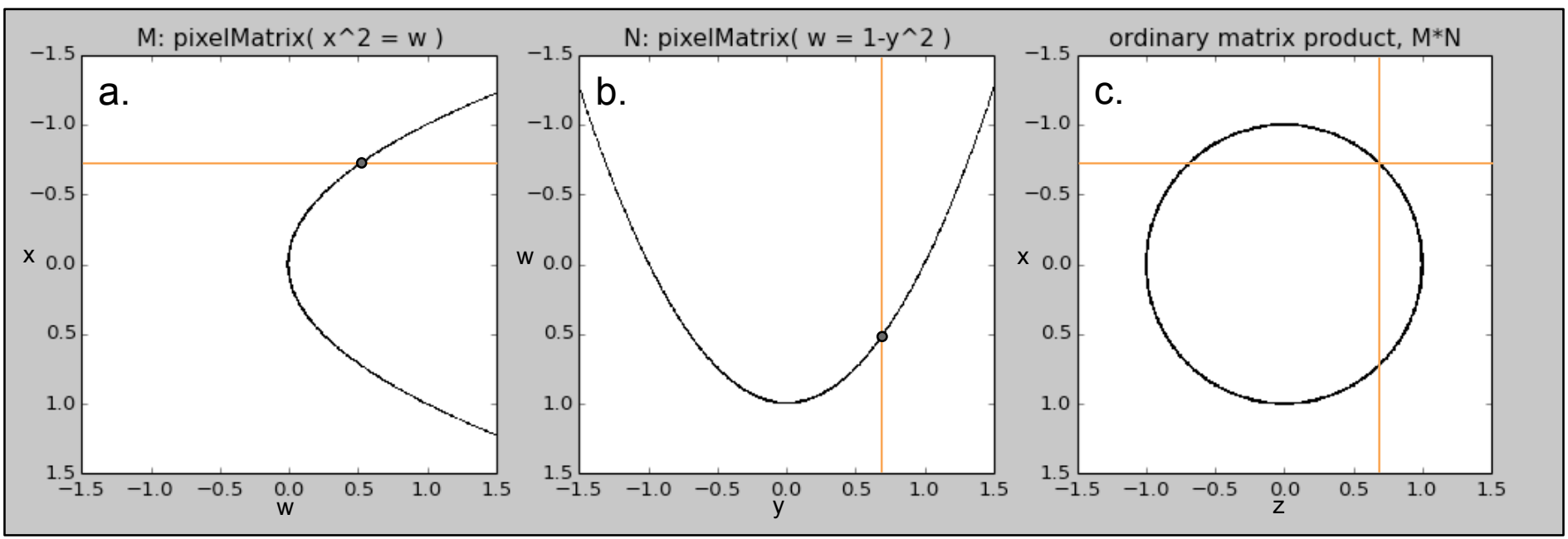
- Plot each in a bounding box, e.g. $[-1.5, 1.5]$.
- Consider plots as matrices of on/off pixels (booleans).
- Multiply matrices to solve system.

Detailed look at an application

Separately plot the solutions to equations: $f(x, w) = 0$ and $g(w, y) = 0$.

- Plot each in a bounding box, e.g. $[-1.5, 1.5]$.
- Consider plots as matrices of on/off pixels (booleans).
- Multiply matrices to solve system.

Example: $x^2 = w$ and $w = 1 - y^2$.



A more complex example

The following eq's are not differentiable, nor even defined everywhere.

$$\cos\left(\ln(z^2 + 10^{-3}x)\right) - x + 10^{-5}z^{-1} = 0 \qquad \text{(Equation 1)}$$

$$\cosh(w + 10^{-3}y) + y + 10^{-4}w = 2 \qquad \text{(Equation 2)}$$

$$\tan(x + y)(x - 2)^{-1}(x + 3)^{-1}y^{-2} = 1 \qquad \text{(Equation 3)}$$

Q: For what values of w and z does a simultaneous solution exist? [4]

[4] Spivak; Dobson; Kumari; Wu (2016) "Pixel Arrays: A fast and elementary method for solving nonlinear systems". https://arxiv.org/abs/1609.00061

A more complex example

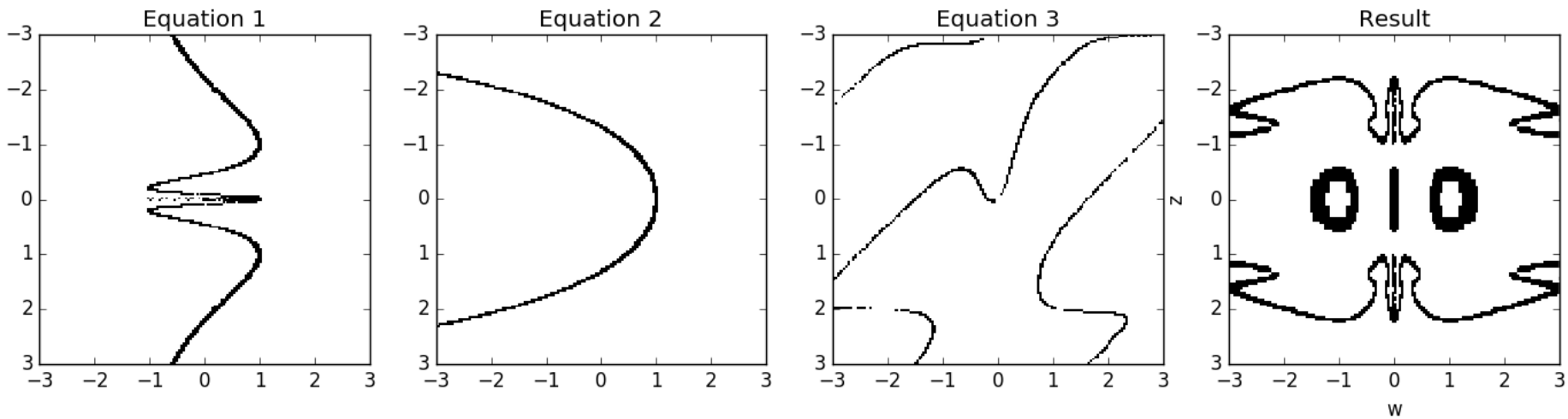
The following eq's are not differentiable, nor even defined everywhere.

$$\cos\left(\ln(z^2 + 10^{-3}x)\right) - x + 10^{-5}z^{-1} = 0 \qquad \text{(Equation 1)}$$

$$\cosh(w + 10^{-3}y) + y + 10^{-4}w = 2 \qquad \text{(Equation 2)}$$

$$\tan(x + y)(x - 2)^{-1}(x + 3)^{-1}y^{-2} = 1 \qquad \text{(Equation 3)}$$

Q: For what values of w and z does a simultaneous solution exist? [4]



[4] Spivak; Dobson; Kumari; Wu (2016) "Pixel Arrays: A fast and elementary method for solving nonlinear systems". https://arxiv.org/abs/1609.00061

Equations and wiring diagrams

Consider an arbitrary system of equations having the following form:

$$f_1(\mathbf{t}, u, \mathbf{v}) = 0$$
$$f_2(\mathbf{v}, w, x) = 0$$
$$f_3(u, w, x, y) = 0$$
$$f_4(x, \mathbf{z}) = 0$$

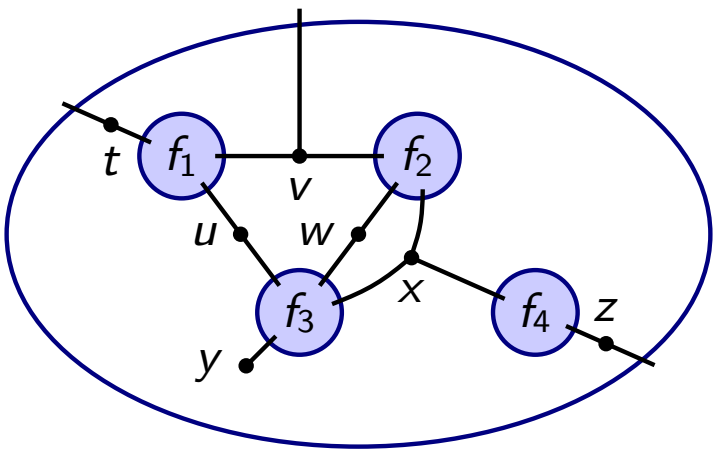
Bold variables are those we want to *expose*; others are *unexposed*.

Equations and wiring diagrams

Consider an arbitrary system of equations having the following form:

$$f_1(\mathbf{t}, u, \mathbf{v}) = 0$$
$$f_2(\mathbf{v}, w, x) = 0$$
$$f_3(u, w, x, y) = 0$$
$$f_4(x, \mathbf{z}) = 0$$

Bold variables are those we want to *expose*; others are *unexposed*.

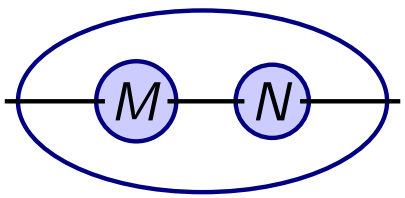


Said another way, we want $\{(t, v, z) \mid \exists u, w, x, y : f_1 = f_2 = f_3 = f_4 = 0\}$.

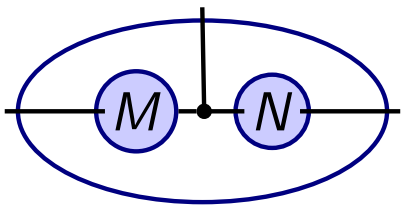
Example wiring diagrams for named operations

Some famous matrix products as wiring diagrams:

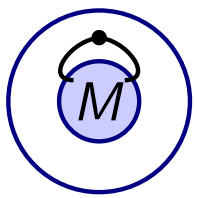
Multiplication: MN



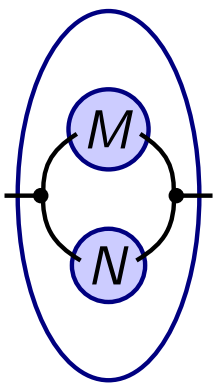
Khatri-Rao: $M \odot N$



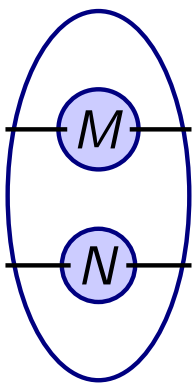
Trace: $\mathrm{Tr}(M)$



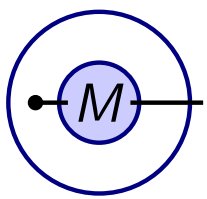
Hadamard: $M \circ N$



Kronecker: $M \otimes N$

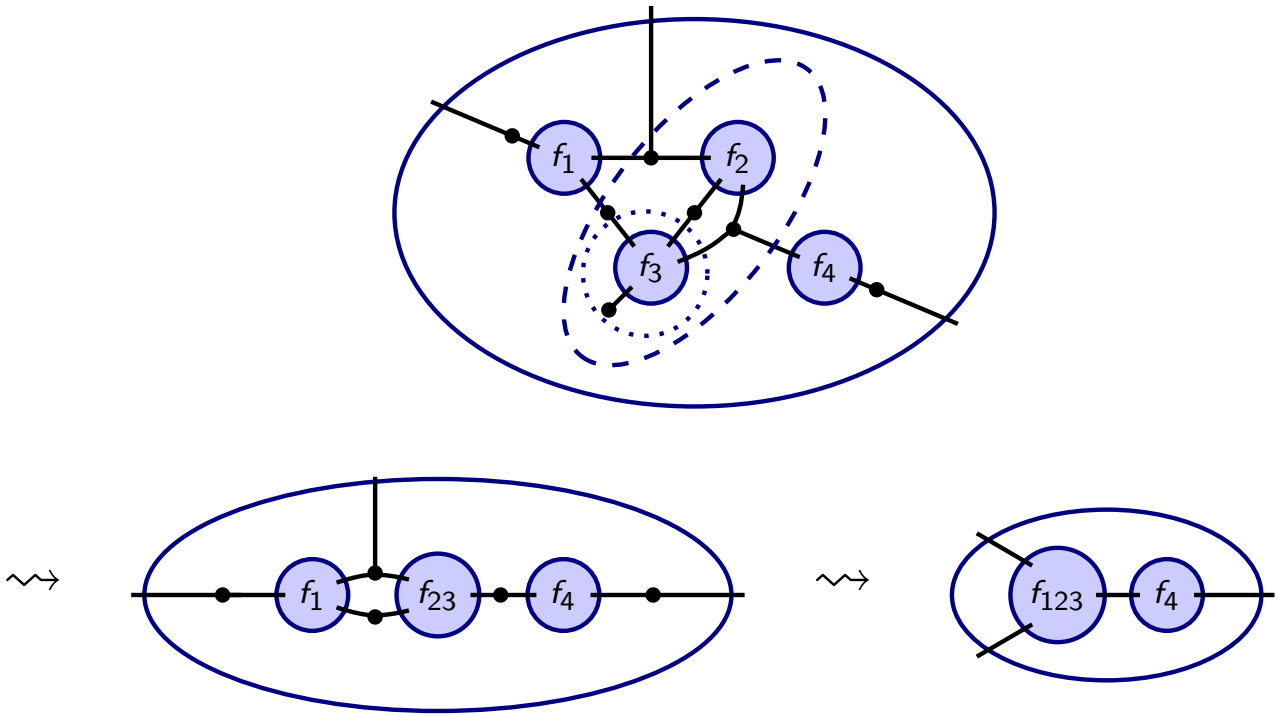


Marginalize: $\sum_i M_{i,j}$



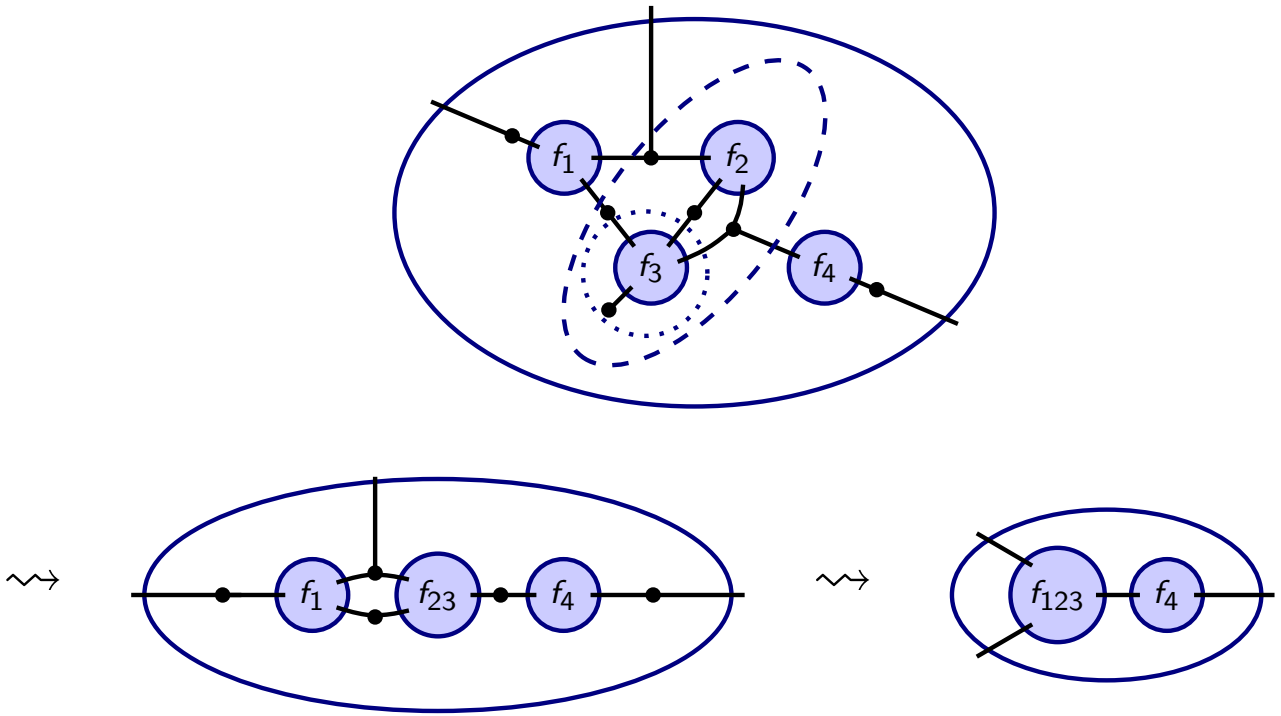
Clustering for speed

To solve systems of equations fast, cluster:



Clustering for speed

To solve systems of equations fast, cluster:



The operad associative law lets you choose the order.

- Order of contracting edges doesn't affect solution.
- But it does affect speed.

Speed test: apples and oranges

The inputs and outputs of the PA method are different than Newton's.

	Pixel Array method	**Newton's method**
Inputs:	a range for each variable	a good initial guess
Outputs:	all solutions for some variables	one solution in all variables.

- So it's hard to compare the speed of PA with Newton. (?)

Speed test: apples and oranges

The inputs and outputs of the PA method are different than Newton's.

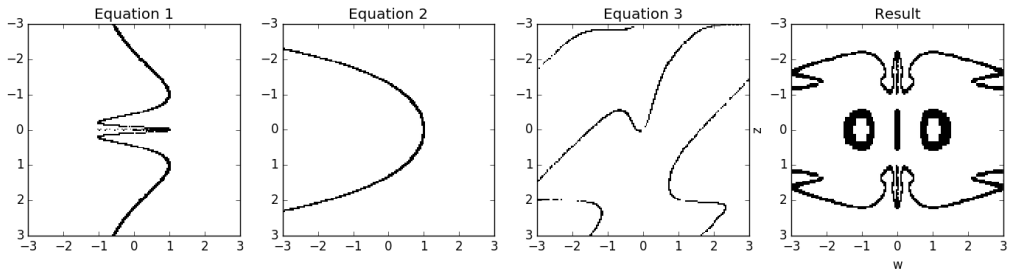
	Pixel Array method	**Newton's method**
Inputs:	a range for each variable	a good initial guess
Outputs:	all solutions for some variables	one solution in all variables.

- So it's hard to compare the speed of PA with Newton. (?)
- Our speed test assumes you want to produce "all solutions" in range.

$$\cos\left(\ln(z^2+10^{-3}x)\right)-x+10^{-5}z^{-1}=0$$

$$\cosh(w+10^{-3}y)+y+10^{-4}w=2$$

$$\tan(x+y)(x-2)^{-1}(x+3)^{-1}y^{-2}=1$$



- We iterated Newton over all boxes in the grid, each as an initial guess.

Speed test: apples and oranges

The inputs and outputs of the PA method are different than Newton's.

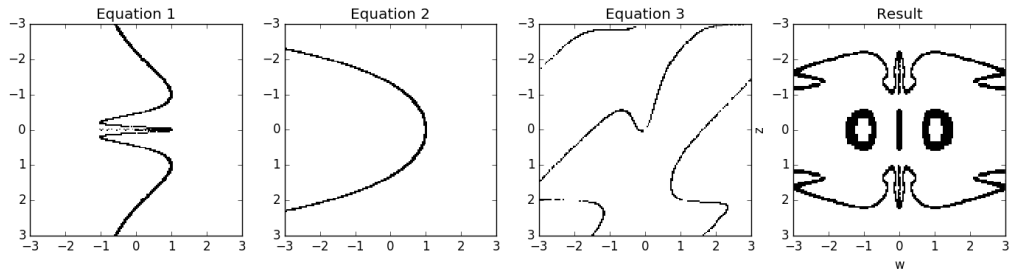
	Pixel Array method	**Newton's method**
Inputs:	a range for each variable	a good initial guess
Outputs:	all solutions for some variables	one solution in all variables.

- So it's hard to compare the speed of PA with Newton. (?)
- Our speed test assumes you want to produce "all solutions" in range.

$$\cos\left(\ln(z^2+10^{-3}x)\right)-x+10^{-5}z^{-1}=0$$

$$\cosh(w+10^{-3}y)+y+10^{-4}w=2$$

$$\tan(x+y)(x-2)^{-1}(x+3)^{-1}y^{-2}=1$$

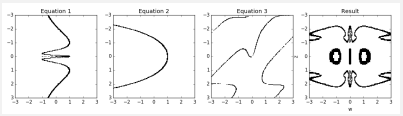


- We iterated Newton over all boxes in the grid, each as an initial guess.

Above case: PA was over 7200x faster.

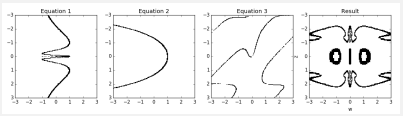
- PA: 1.5 seconds; Newton: we stopped Julia's NLsolve after 3 hours.
- PA was faster in every partially-decomposable system we tried.

Accuracy of the pixel array method



Output accuracy = input accuracy.

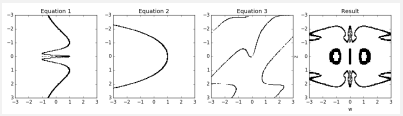
Accuracy of the pixel array method



Output accuracy = input accuracy.

- Start with a plot for each equation $f(x) = 0$ in system.
  - Fact: every cts function is uniformly cts in a bounding box.
  - This sets up a relationship between mesh-size δ and tolerance ϵ.

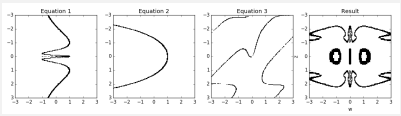
Accuracy of the pixel array method



Output accuracy = input accuracy.

- Start with a plot for each equation $f(x) = 0$ in system.
  - Fact: every cts function is uniformly cts in a bounding box.
  - This sets up a relationship between mesh-size δ and tolerance ϵ.
  - Plotting method: sample in center c of pixel, mark if $|f(c)| < \epsilon$.
  - This gives two accuracy guarantees:
    - If $f(x) = 0$ anywhere in the pixel then it is marked.
    - If pixel is marked then $|f(x)| < 2\epsilon$ everywhere in the pixel.

Accuracy of the pixel array method



Output accuracy = input accuracy.

- Start with a plot for each equation $f(x) = 0$ in system.
  - Fact: every cts function is uniformly cts in a bounding box.
  - This sets up a relationship between mesh-size δ and tolerance ϵ.
  - Plotting method: sample in center c of pixel, mark if $|f(c)| < \epsilon$.
  - This gives two accuracy guarantees:
    - If $f(x) = 0$ anywhere in the pixel then it is marked.
    - If pixel is marked then $|f(x)| < 2\epsilon$ everywhere in the pixel.
- PA method: multiply plots as tensors.
- Result (solution) has the same guarantees:
  - If there is a solution in a pixel, it is marked.
  - If a pixel is marked, then the pixel is 2ϵ-close to a solution.

Where we are

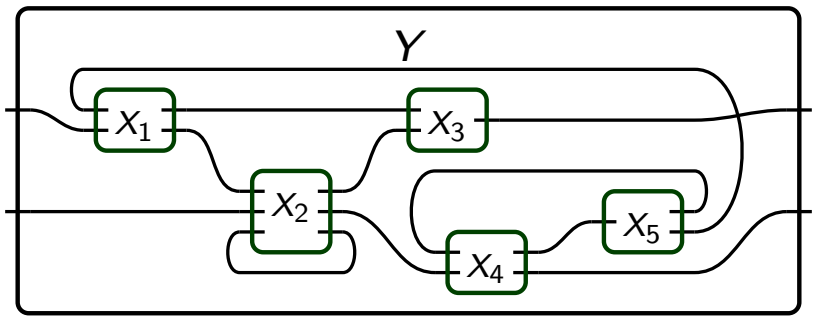
The talk is almost done!

- Discussed operads: general compositional structures
- Lots of algebras on the wiring diagrams operad $\mathcal{W}$
  - Open dynamical systems: behavior of interacting machines
  - Relations: find simultaneous solutions
  - Tensors: contract tensor networks
- Discussed the pixel array method
  - It turns relations into tensors by plotting.
  - Plotting is "compositional": solve-then-plot = plot-then-contract

Last thing: discuss compositional mappings for dynamical systems.

Compositional mappings

Suppose there's a continuous DS in each box:



Two choices; what's the difference?

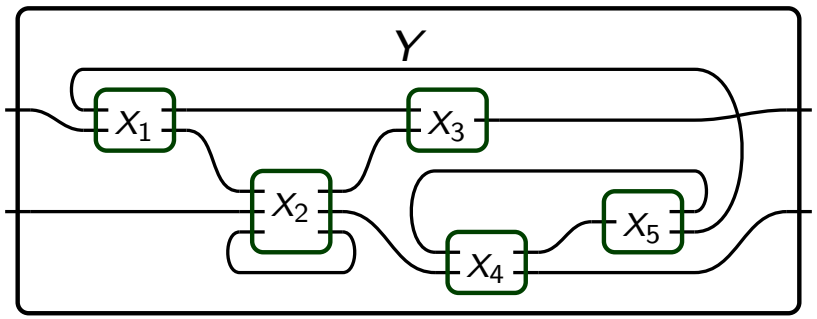
- Choice 1: compose the continuous systems and discretize the result;
- Choice 2: discretize each and then compose the discrete systems.

[5] Spivak "The steady states of dynamical systems". *arXiv* 1512.00802.

[6] Ngotiaoco "Compositionality of the Runge-Kutta Method". *arXiv* 1707.02804.

Compositional mappings

Suppose there's a continuous DS in each box:



Two choices; what's the difference?

- Choice 1: compose the continuous systems and discretize the result;
- Choice 2: discretize each and then compose the discrete systems.

"Discretization is compositional": you get the same result either way.

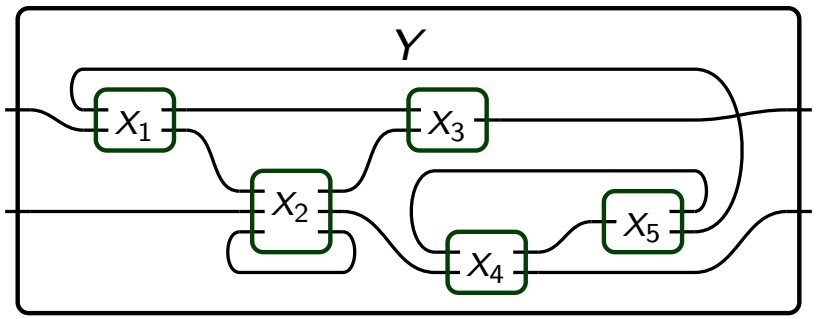
- This requires proof: arbitrary wiring diagrams, arbitrary DS's.[5] [6]

[5] Spivak "The steady states of dynamical systems". *arXiv* 1512.00802.

[6] Ngotiaoco "Compositionality of the Runge-Kutta Method". *arXiv* 1707.02804.

Steady state plots are compositional

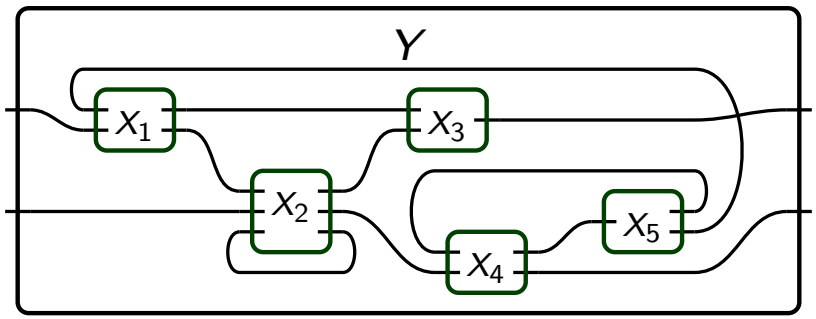
One last compositional mapping: steady state plots.



Start with an ODS in each box

Steady state plots are compositional

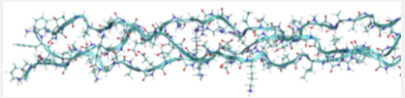
One last compositional mapping: steady state plots.

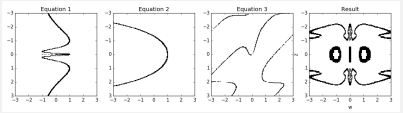


Start with an ODS in each box

- Compute steady states: solution to $\dot{x} = 0$
- Do this for each ODS in the wiring diagram
  - Get a system of equations
  - Plot to get "steady state plots" (a.k.a. bifurcation diagrams)
  - Compose these plots using PA method
- Steady state plots are compositional.
- Steady states of whole = simultaneous steady states of parts

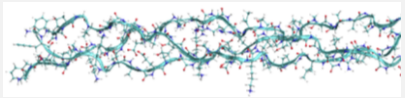
Summary

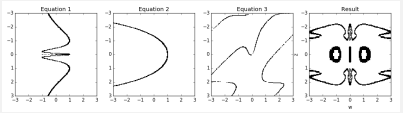




Applications of category theory. Focus: operads.

Summary

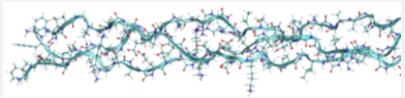


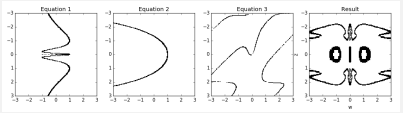


Applications of category theory. Focus: operads.

- Operads give a framework for composing things of any sort.
  - Object (interface), morphism (arrangement), composition (nesting).
  - Discussed wiring diagrams, protein materials, grammars, recipes
  - Operad = "theory of composition"; algebras = models

Summary

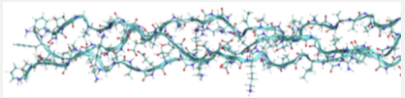


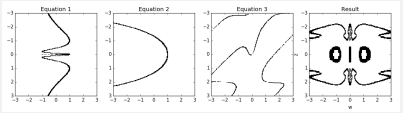


Applications of category theory. Focus: operads.

- Operads give a framework for composing things of any sort.
  - Object (interface), morphism (arrangement), composition (nesting).
  - Discussed wiring diagrams, protein materials, grammars, recipes
  - Operad = "theory of composition"; algebras = models
- Compositional mappings between algebras
  - Translate-then-operate = operate-then-translate
  - Solving systems of equations: pixel array method
  - Discretization and steady states of dynamical systems

Summary



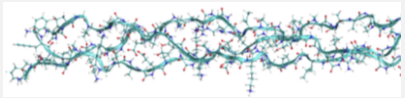


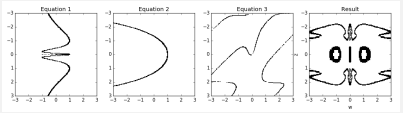
Applications of category theory. Focus: operads.

- Operads give a framework for composing things of any sort.
  - Object (interface), morphism (arrangement), composition (nesting).
  - Discussed wiring diagrams, protein materials, grammars, recipes
  - Operad = "theory of composition"; algebras = models
- Compositional mappings between algebras
  - Translate-then-operate = operate-then-translate
  - Solving systems of equations: pixel array method
  - Discretization and steady states of dynamical systems

Operads: just one aspect of CT, a beautiful framework for thinking.

Summary





Applications of category theory. Focus: operads.

- Operads give a framework for composing things of any sort.
  - Object (interface), morphism (arrangement), composition (nesting).
  - Discussed wiring diagrams, protein materials, grammars, recipes
  - Operad = "theory of composition"; algebras = models
- Compositional mappings between algebras
  - Translate-then-operate = operate-then-translate
  - Solving systems of equations: pixel array method
  - Discretization and steady states of dynamical systems

Operads: just one aspect of CT, a beautiful framework for thinking.

Questions and comments welcome!