A mathematical language for modular systems

Patrick Schultz

PI: David I. Spivak
dspivak@math.mit.edu
Mathematics Department
Massachusetts Institute of Technology

Presented on 2015/01/29
at the AFOSR program review

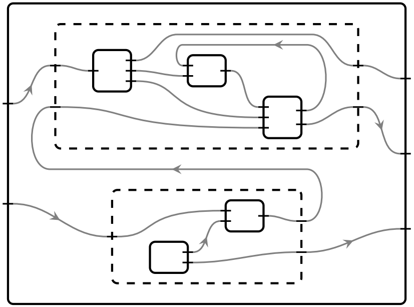
Outline

Outline

Modular systems are everywhere

Modular systems are everywhere; they need a mathematical foundation.

- In many different fields, people draw pictures like this.
- But what do they mean?



Operads as a foundation for modularity

Operads formalize the idea of building one thing out of many others.

We make the usual distinction between interface and implementation:

- An *interface* is an abstraction that we use for design.
- An *implementation* of an interface is a conforming manifestation.

This distinction will be explained mathematically using algebras.

Operads as a foundation for modularity

Operads formalize the idea of building one thing out of many others.
We make the usual distinction between interface and implementation:

- An *interface* is an abstraction that we use for design.
- An *implementation* of an interface is a conforming manifestation.

This distinction will be explained mathematically using algebras.

Why a mathematical foundation is needed

In many fields of design, the planning stage uses modular pictures.

- Box-and-line drawings in software architecture
- Data flow or work flow diagrams
- Compartmental models of the body

But there are complaints:

> *While these [box-and-line] descriptions may provide useful documentation, the current level of informality limits their usefulness. Since it is generally imprecise what is meant by such architectural descriptions, it may be impossible to analyze an architecture for consistency or determine non-trivial properties of it. Moreover, there is no way to check that a system implementation is faithful to its architectural design. –Allen and Garlan*

By making these descriptions formal, we also make them useful.

Why a mathematical foundation is needed

In many fields of design, the planning stage uses modular pictures.

- Box-and-line drawings in software architecture
- Data flow or work flow diagrams
- Compartmental models of the body

But there are complaints:

> *While these [box-and-line] descriptions may provide useful documentation, the current level of informality limits their usefulness. Since it is generally imprecise what is meant by such architectural descriptions, it may be impossible to analyze an architecture for consistency or determine non-trivial properties of it. Moreover, there is no way to check that a system implementation is faithful to its architectural design. –Allen and Garlan*

By making these descriptions formal, we also make them useful.

Why a mathematical foundation is needed

In many fields of design, the planning stage uses modular pictures.

- Box-and-line drawings in software architecture
- Data flow or work flow diagrams
- Compartmental models of the body

But there are complaints:

> *While these [box-and-line] descriptions may provide useful documentation, the current level of informality limits their usefulness. Since it is generally imprecise what is meant by such architectural descriptions, it may be impossible to analyze an architecture for consistency or determine non-trivial properties of it. Moreover, there is no way to check that a system implementation is faithful to its architectural design. –Allen and Garlan*

By making these descriptions formal, we also make them useful.

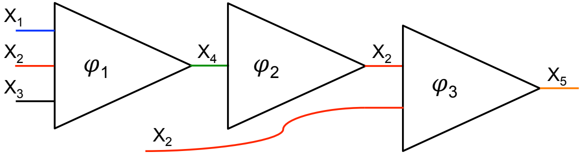
Outline

Operads

What I'm calling an operad is sometimes called

- a colored operad, or
- a symmetric multicategory.

Interfaces, colors, objects: three names for the same concept.
Operads formalize many-input, one-output relationships.
They are often drawn as "tree diagrams":



Definition of operad

Definition

An operad $\mathcal{O}$ consists of

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

  $$\mathrm{Hom}_{\mathcal{O}}(X_1, \ldots, X_n; Y),$$

  elements, called *morphisms from* $X_1, \ldots, X_n$ *to* Y.
  A morphism $\varphi \in \mathrm{Hom}_{\mathcal{O}}(X_1, \ldots, X_n; Y)$ may be denoted

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

- For each object $X \in \mathrm{Ob}(\mathcal{O})$, a morphism $\mathrm{id}_X \colon (X) \to X$
- A composition 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.

The first operad

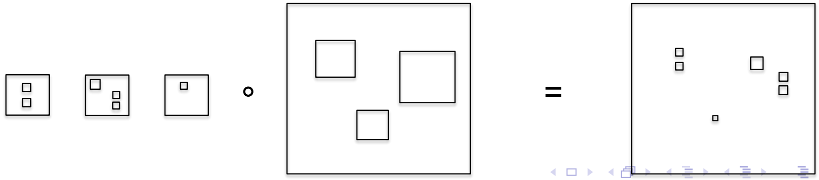
Operads were invented by Peter May, a mathematician at U. Chicago. The first operad was called *the little n-cubes operad* and denoted $\mathcal{E}_n$. When $n = 2$:

Definition

The operad $\mathcal{E}_2$ is defined by

- $\mathrm{Ob}(\mathcal{E}_2) := \{\square\}$. (One object.)
- $\mathrm{Hom}_{\mathcal{E}_2}(\square_1, \ldots, \square_n; \square) := \{n \text{ squares placed in a square}\}$
- $\mathrm{id}_\square$ is biggest possible square. (I may skip identities from now on.)

The composition formula in pictures:



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

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

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

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

- 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

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

- These two functions should respect identity and composition.

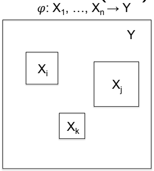
Definition

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

Example: two different algebras on $\mathcal{E}_2$

To give an algebra $F\colon \mathcal{E}_2 \to \mathbf{Sets}$, we must provide

- a set $F(X) \in \mathrm{Ob}(\mathbf{Sets})$ for each object $X \in \mathrm{Ob}(\mathcal{E}_2)$
- a function $F(\varphi)\colon F(X_1) \times \cdots \times F(X_n) \to F(Y)$ for each morphism



Children's drawings

Define $D(\square) = \{\text{ways to color the space in a square}\}$.
The function $D(\varphi)$ takes n-colorings and produce a coloring.

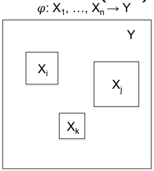
2-spheres in a based space X

Define $\Omega_X(\square) = \mathrm{Hom}_{\mathbf{Top}_*}(S^2, X)$.
A morphism φ takes n spheres in X and produces a new one.

Example: two different algebras on $\mathcal{E}_2$

To give an algebra $F\colon \mathcal{E}_2 \to \textbf{Sets}$, we must provide

- a set $F(X) \in \mathrm{Ob}(\textbf{Sets})$ for each object $X \in \mathrm{Ob}(\mathcal{E}_2)$
- a function $F(\varphi)\colon F(X_1) \times \cdots \times F(X_n) \to F(Y)$ for each morphism



Children's drawings

Define $D(\square) = \{$ways to color the space in a square$\}$.
The function $D(\varphi)$ takes n-colorings and produce a coloring.

2-spheres in a based space X

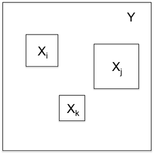
Define $\Omega_X(\square) = \mathrm{Hom}_{\textbf{Top}_*}(S^2, X)$.
A morphism φ takes n spheres in X and produces a new one.

Example: two different algebras on $\mathcal{E}_2$

To give an algebra $F\colon \mathcal{E}_2 \to \mathbf{Sets}$, we must provide

- a set $F(X) \in \mathrm{Ob}(\mathbf{Sets})$ for each object $X \in \mathrm{Ob}(\mathcal{E}_2)$
- a function $F(\varphi)\colon F(X_1) \times \cdots \times F(X_n) \to F(Y)$ for each morphism

φ: $X_1, \ldots, X_n \to Y$



Children's drawings

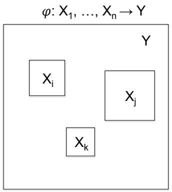
Define $D(\square) = \{$ways to color the space in a square$\}$.
The function $D(\varphi)$ takes n-colorings and produce a coloring.

2-spheres in a based space X

Define $\Omega_X(\square) = \mathrm{Hom}_{\mathbf{Top}_*}(S^2, X)$.
A morphism φ takes n spheres in X and produces a new one.

The nomenclature we will use

- Let $\mathcal{O}$ be an operad, and let $F\colon \mathcal{O} \to \mathbf{Sets}$ be an algebra.
  - $\mathcal{O}$ is the *abstract modular environment*.
  - F is an *interpretation* of $\mathcal{O}$.
- An object $X \in \mathrm{Ob}(\mathcal{O})$ in the operad will be called an *interface*.
  - An element $f \in F(X)$ is an *F-implementation of X*.
- A morphism $\varphi\colon (X_1, \ldots, X_n) \to Y$ will be called an *arrangement*.
  - φ forms interface Y as an arrangement of interfaces $X_1, \ldots, X_n$.
  - $F(\varphi)\colon F(X_1) \times \cdots \times F(X_n) \to F(Y)$ is a *production formula*.



- We'll refer to composition in $\mathcal{O}$ as *nesting*.

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⟩ | ""

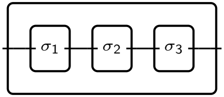
- Everything in ⟨brackets⟩ is an interface.
- Each line is a production formula, usually called a "production rule".
- Composition—nesting—of production rules is straightforward.
- The usual interpretation of this CFG: strings and concatenations.

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

The operad for monoids

Here's another operad, $\mathcal{A}$, with only one type of interface (one object).

- $\mathrm{Ob}(\mathcal{A}) = \{\boxminus\}$; just one interface.
- $\mathrm{Hom}_{\mathcal{A}}(\boxminus_1, \dots, \boxminus_n; \boxminus)$ is the set of permutations $\sigma \in \Sigma_n$.
- For example, if $n = 3$, we're calling σ an arrangement of 3 interfaces:



An algebra $M\colon \mathcal{A} \to \mathbf{Sets}$ provides:

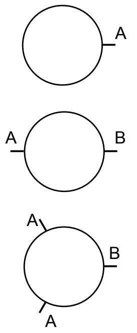
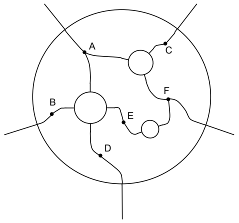
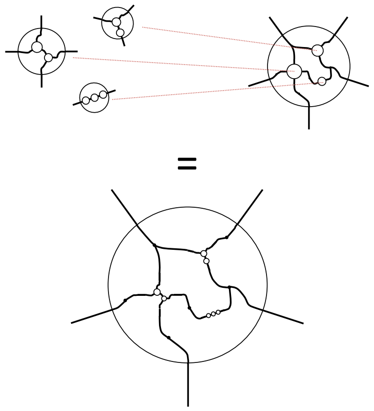
- a set $M(\boxminus)$ of "actions", i.e., the set of ways M implements $\boxminus$
- for every arrangement $\sigma \in \Sigma_n$, a production formula $M(\sigma)$:

  $M(\sigma)$ takes any actions $m_1, \dots, m_n$, does them in the order prescribed by σ, and produces an action $M(\sigma)(m_1, \dots, m_n)$.

The category of $\mathcal{A}$-algebras is equivalent to the category of monoids.

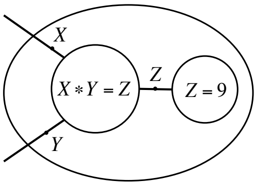
Outline

An operad $\mathcal{S}$ of static wiring diagrams

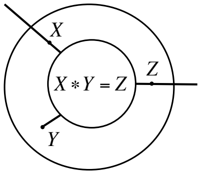
Interfaces	Arrangements	Nesting
		
"circles": finite labeled sets X	surjective, labeled cospans $\coprod X_i \to \bullet \leftarrow Y$	pushouts of labeled cospans

The $\mathcal{S}$-algebra of relations and queries

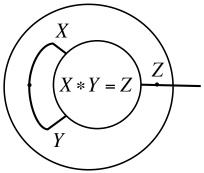
There is an $\mathcal{S}$-algebra of *relations*, $Rel: \mathcal{S} \to \textbf{Sets}$.



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



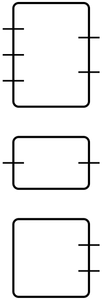
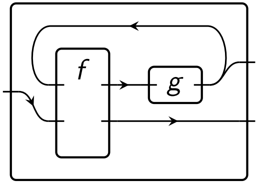
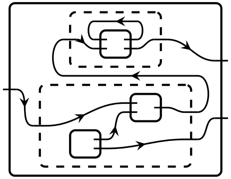
"all pairs of integers (*X*,*Z*) in which *Z* is divisible by *X*."



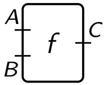
"all perfect squares *Z*"

- A *Rel*-implementation of a circle is a relation on that type.
- Each arrangement in $\mathcal{S}$ defines a conjunctive query.
- The *Rel*-production rule performs the query on the input relations.

An operad $\mathcal{T}$ of temporal wiring diagrams

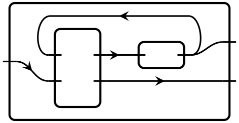
Interfaces	Arrangements	Nesting
		
pairs (X^-, X^+) of finite labeled sets	certain surjections $\varphi\colon X^- + Y^+ \twoheadrightarrow X^+ + Y^-$	Composition: see paper.

The $\mathcal{T}$-algebra of stream processors



There is a $\mathcal{T}$-algebra $P\colon \mathcal{T} \to \textbf{Sets}$ of stream processors.

- A P-implementation of a box X is a stream processor of type X.
- That is, $f \in P(X)$ inputs streams in $A \times B$ and outputs streams in C.
- Given arrangement φ, there's a straightforward production rule $P(\varphi)$.



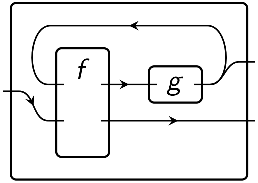
a morphism ("arrangement") in $\mathcal{T}$

Algebras that won't work

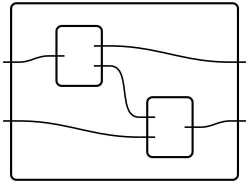
In case you have the idea that everything works....

- Is there an algebra of functions on $\mathcal{T}$?
  - A function is a "historically independent" stream processor.
  - But these are not closed under arrangements in $\mathcal{T}$.
  - Feedback loops create historical dependence.
  - For a $\mathcal{T}$-algebra you need something like stream processors.
- Functions do form an algebra on the directed sub-operad $\mathcal{T}' \subseteq \mathcal{T}$.

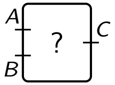
morphisms in $\mathcal{T}$ can have feedback



having no feedback, $\mathcal{T}'$ supports an algebra of functions



A $\mathcal{T}$-algebra of open dynamical systems



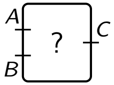
Let inp and outp be manifolds. (In the above, think: $\texttt{inp} = A \times B$ and $\texttt{outp} = C$.)

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.

A $\mathcal{T}$-algebra of open dynamical systems



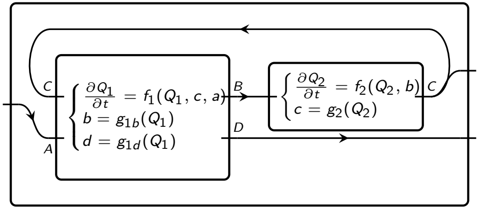
Let $\texttt{inp}$ and $\texttt{outp}$ be manifolds. (In the above, think: $\texttt{inp} = A \times B$ and $\texttt{outp} = C$.)

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.

A $\mathcal{T}$-algebra of open dynamical systems


$\begin{cases} \frac{\partial Q_1}{\partial t} = f_1(Q_1, c, a) \\ b = g_{1b}(Q_1) \\ d = g_{1d}(Q_1) \end{cases}$
$\begin{cases} \frac{\partial Q_2}{\partial t} = f_2(Q_2, b) \\ c = g_2(Q_2) \end{cases}$
A
B
C
C
D

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.

Summary: operads and algebras formalize modular systems

We discussed a wide variety of operads:

- Little squares $\mathcal{E}_2$, CFGs, and wiring diagrams (of various flavors).
- It is reasonable to call these abstract modular environments.
- Can you think of a modular environment that is not an operad?

We discussed some algebras, i.e., interpretations, of each:

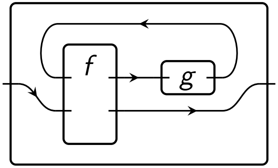
- Proposed interpretations $F\colon \mathcal{O} \to \textbf{Sets}$ can be rigorously checked.
- F's production formula is pseudo-code for actual implementation.

Open questions: how else might we apply these and other operads?

- Is there a software architecture interpretation of $\mathcal{T}$?
- What about logistics and planning, pharmaceutical drug recipes, etc.?

Outline

String diagrams in traced categories



Pictures like the above are well-known in category theory. They appear in the theory of *traced (symmetric monoidal) categories*.

- Traced categories are often used to model feedback or fixed points.
- The name comes from the notion of trace for linear transformations.

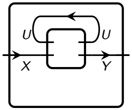
Traced categories and string diagrams are due to Joyal, Street, Verity.

Traced categories

Given a monoidal category $(\mathcal{C}, I, \otimes)$, a *trace* is an additional structure:

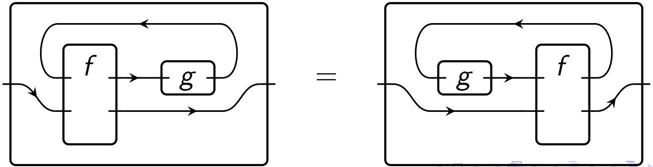
- For every $X, Y, U \in \mathrm{Ob}(\mathcal{C})$, there is a trace function

$$Tr^{U}_{X,Y} \colon \mathrm{Hom}(U \times X, U \times Y) \to \mathrm{Hom}(X, Y)$$

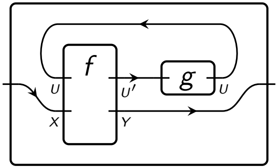


- These trace functions must satisfy six axioms, e.g.,

$$Tr^{U}_{X,Y}\big[(g \otimes \mathrm{id}_Y) \circ f\big] \;=\; Tr^{U'}_{X,Y}\big[f \circ (g \otimes \mathrm{id}_X)\big]$$



What’s the relationship?



Clearly the drawings are similar, but there’s a degree shift:

Wires: Objects in a traced category, labels in the operad.

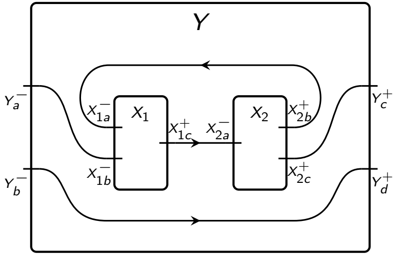
Boxes: Morphisms in a traced category, objects in the operad.

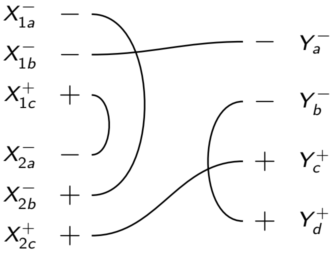
Diagrams: Compositions in a traced category, morphisms in the operad.

Nesting: Axioms of traced categories, composition in the operad.

So what’s the connection?

String diagrams are pictures of cobordisms





What we found was a bit surprising: String diagrams are cobordisms between oriented 0-manifolds.

- Let's denote this operad by **Cob**.
- If wires have labels in a set $\mathcal{L}$, then the operad is $\mathbf{Cob}_{/\mathcal{L}}$
- $\mathbf{Cob}_{/\mathcal{L}}$ is (the underlying multicategory of) the free compact closed symmetric monoidal category on the set $\mathcal{L}$ of labels.

Traced categories as $\mathbf{Cob}_{/\bullet}$-algebras

For each set $\mathcal{L}$, there is a category $\mathbf{Cob}_{/\mathcal{L}}$–$\mathbf{Alg}$.

- This assignment is contravariant in the set of labels:
- a function $\mathcal{L}' \to \mathcal{L}$ induces a functor $\mathbf{Cob}_{/\mathcal{L}}$–$\mathbf{Alg} \to \mathbf{Cob}_{/\mathcal{L}'}$–$\mathbf{Alg}$.
- There is a way of gluing these categories into a category $\mathbf{Cob}_{/\bullet}$–$\mathbf{Alg}$.

Let **TrCat** denote the category of traced categories.

Theorem

There is an equivalence of categories

$$\mathbf{TrCat} \simeq \mathbf{Cob}_{/\bullet}\text{–}\mathbf{Alg}.$$

What this equivalence tells us

- Traced categories are a popular formalism for modeling feedback.
  - Used in physics; e.g., Abramsky, Coecke.
  - Used in computer science; e.g., functional reactive programming (FRP).
- The axioms of traced categories are not obviously tweakable.
  - What if an FRP person wanted a wire to split or terminate.
  - This does not fit directly into the traced categories setup.
  - What axioms should it satisfy?
- The equivalence $\mathbf{TrCat} \simeq \mathbf{Cob}_{/\bullet}\text{–}\mathbf{Alg}$ tells us how to tweak.
  - As mentioned above, our operad for stream processors was not **Cob**.
  - So traced categories are put in a larger context of operad-algebras.

Outline

Where we are in the talk

- Above we showed how operads model abstract modular environments.
  - We showed $\mathcal{E}_2$, CFGs, and wiring diagrams.
  - Each had a notion of interfaces and arrangements.
  - And we discussed algebras, which interpreted these structures.
  - (Variously) joint with Dylan Rupel, Dmitry Vagner, Eugene Lerman.
- We will now give a concrete application of the operad-algebra idea.
  - It's more concrete in that we actually built software.
  - And this software solves a real need in Materials Science.
  - It's joint work with Tristan Giesa, Ravi Jagadeesan, Markus Buehler.

A problem in materials design

- In materials science, one tries to fabricate new better materials.
- An emerging paradigm in materials design: control at all levels.
    - Old idea: take known macro-materials and combine them in new ways.
    - New idea: design from the ground up, fine-tuning at all levels.
- Hierarchical protein materials offer the ability to do that.
    - Proteins are machines that do every task in your body.
    - They also form your waterproof breathable stretchable skin.
    - We can control the amino acid sequences using genetic engineering.
    - By controlling the structure at all levels, we get fine-tuned results.
- The problem is, we have no idea what we're doing.
    - Through experimental trial and error, we slowly learn how things work.
    - Recently, people are simulating protein materials to learn faster.

Our tool: Matriarch

- 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 and their algebras.

The operadic model of Matriarch

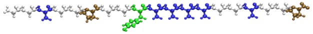
Let $\mathcal{M}$ and $P\colon \mathcal{M} \to \mathbf{Set}$ denote the operad and algebra for Matriarch.

- The objects (interfaces) in $\mathcal{M}$ are pairs (ℓ, r) of sequences in $\{+, -\}$
  - ℓ and r are imagined as the left and right terminals of a protein.
  - An interface $\big((+,+,-,+),(-,-,+)\big)$ is what's available for bonding.
- The morphisms (arrangements) 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.
- Algebra P interprets $\mathcal{M}$ as protein descriptions and transformations.
  - $P(\ell, r)$ is the set of proteins with that bondable interface.
  - $P(\varphi)$ is a formula to produce new proteins from old.

Sample architectures

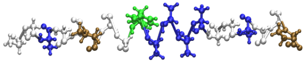
a

Strand1 = chain(seq1)



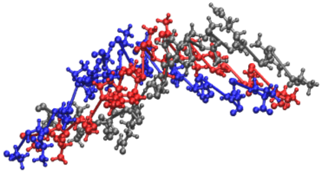
b

Hel1 = helix(Strand1, 1.0, 5.0)



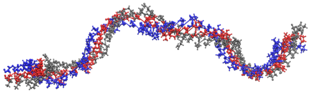
c

TH = collagen(Strand1, Strand2)



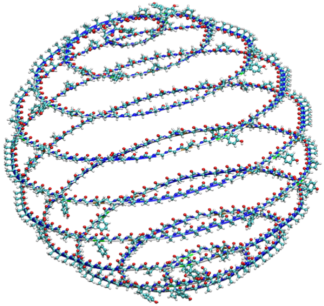
d

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



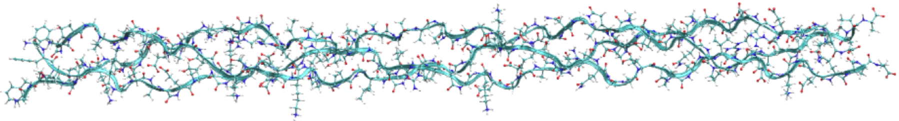
e

Apple = twist(Strand3, SSFunc)



Example of materials architecture: collagen

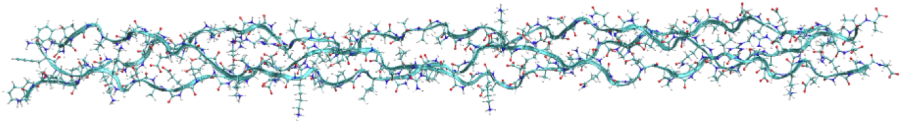
- Collagen is the most common protein in mammals.
- Its design is hierarchical.



```
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

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

- With Matriarch, it is easy to adjust protein material architecture.
  - Just play with the numbers (e.g., 85), or change the sequence (*seq*).
  - 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.
  - This, instead of a list of atomic coordinates, or a prose description.
  - It's more reproducible by other labs.
- Matriarch output can be fed to a molecular dynamics simulator.
  - It is being used in Buehler's lab at MIT.
  - It will soon be open-source and freely available.

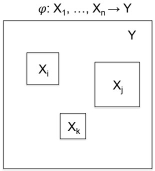
What did operads really do for us?

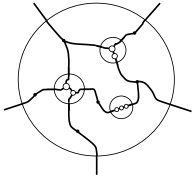
- One could object that the idea of Matriarch is straightforward.
  - It's simply some operations that you can do to materials.
  - Performing such operations in series or parallel is not new.
  - Why do you need category theory and operads?
- Operads were the software specification.
  - Goguen said that CT is a natural language for software specification.
  - Having an operad-algebra description simplified the code-writing.
  - The software has a mathematical description; it can be verified.
- The material architectures have mathematical descriptions.
- It's easy to change the operad or the algebra.
  - Adding new operations or objects can be done incrementally.
  - Functors between operads allow the software to evolve gracefully.

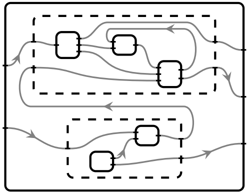
Restating the general idea

- There are no pictures in the Matriarch operad $\mathcal{M}$.
  - All those pictures were of the implementation, the algebra.
  - Sure, you could imagine the (ℓ, r) sequences.
  - But important operations like `twist` don't change those sequences.
- Is Matriarch an abstract modular environment?
  - It's a language for manipulating different proteins in parallel and series.
  - "Twist A, attach it to B, and then make an 8×8 array of that."
  - Building complex things out of simpler pieces.
- Modularity is a very general phenomenon.
  - We're not just talking wiring diagrams or squares in a square.
  - Matriarch expands our notion of what operads can model.

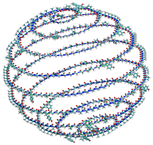
Outline







⟨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⟩ | ""



Summary

- In this talk, I discussed operads as abstract modular environments.
  - An algebra on an operad is an interpretation of the environment.
  - This is a mathematical language for a general phenomenon.
- I discussed several examples and applications.
  - For example, wiring diagrams can come in many flavors.
  - So can their algebras: stream processors, databases, dynamical systems.
- I talked about a result in pure math: $\mathbf{TrCat} \simeq (\mathbf{Cob}/\bullet)\text{–}\mathbf{Alg}$.
  - This equivalence puts wiring diagrams into context.
  - It also tells us how to tweak the axioms of traced categories.
- Finally, I discussed Matriarch, software for materials architecture.

Papers and transfers

Below are links for the papers, etc., most relevant to the AFOSR grant.

- Book: *Category Theory for the Sciences*.
- Papers:
  - "The operad of wiring diagrams: formalizing a graphical language for databases, recursion, and plug-and-play circuits".
  - (with Dylan Rupel) "The operad of temporal wiring diagrams: formalizing a graphical language for discrete-time processes".
  - (with Dmitry Vagner and Eugene Lerman), "Algebras of Open Dynamical Systems on the Operad of Wiring Diagrams".
  - (with Dylan Rupel, Patrick Schultz). "Traced Monoidal Categories as Lax Functors out of Free Compact Categories". In preparation.
  - (with Tristan Giesa, Ravi Jagadeesan, Markus Buehler) "A Python Library for Materials Architecture". In preparation.
- Transfer:
  - NASA (Langley research center). "Category-theoretic Approaches for the Analysis of Distributed Systems".