Operads as a language for modular design

David I. Spivak

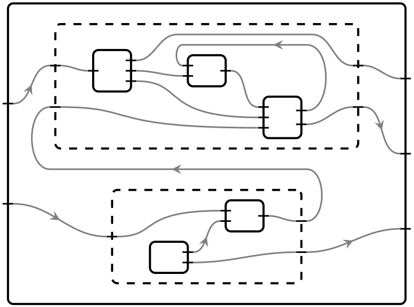
dspivak@math.mit.edu
Mathematics Department
Massachusetts Institute of Technology

Presented on 2015/06/05 at
Foundational Methods in Computer Science

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?



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.

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.

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*

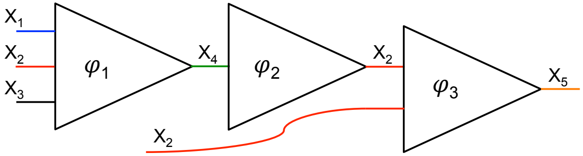
By making these descriptions formal, we also make them useful.

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.

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 $X_1, \dots, X_n$, there is a product set $X_1 \times \cdots \times X_n$.

Definition

The operad **Sets** is defined by

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

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

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

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

- These two functions should respect identity and composition.

Definition

An operad functor $F\colon \mathcal{O} \to \textbf{Sets}$ 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 $F\colon \mathcal{O} \to$ **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,
  - To each type, e.g., $\langle$house-num$\rangle$, assign the set of such strings.
  - Each production rules is assigned a function.
  - Namely, it produces a more complex object from a tuple of simpler ones.

To begin: a concrete application

I want to give a concrete application of the operad-algebra idea.

- It's 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.
- Based on operads and their algebras.

The operadic model of Matriarch

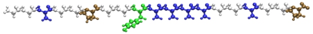
Let $\mathcal{M}$ and $P\colon \mathcal{M} \to \mathbf{Set}$ denote the operad and algebra for Matriarch.

- 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

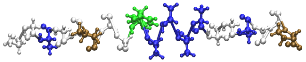
a

Strand1 = chain(seq1)



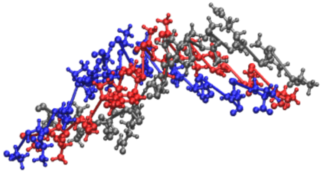
b

Hel1 = helix(Strand1, 1.0, 5.0)



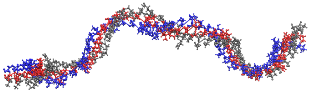
c

TH = collagen(Strand1, Strand2)



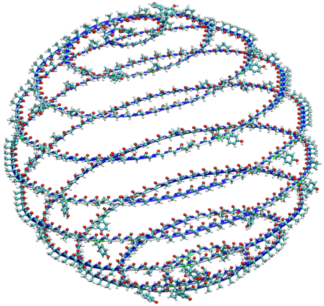
d

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



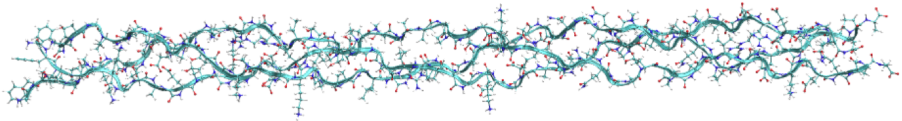
e

Apple = twist(Strand3, SSFunc)



Example of materials architecture: collagen

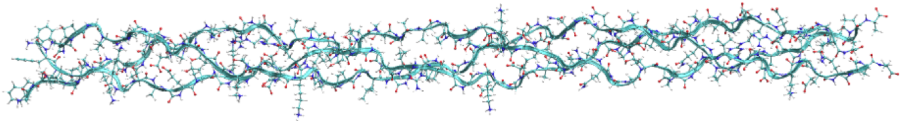
- Collagen is the most common protein in mammals.
- Its design is hierarchical.



```
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

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

- 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 is open-source and freely available:

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

On to the chalkboard

- This concludes the slide portion of my talk.
- Up next: using operads to design things like...
  - Relational algebra expressions,
  - Passive linear circuits,
  - Causal Bayesian theories,
  - Continuous and discrete dynamical systems.
- And I'll end by stating a theorem about traced monoidal categories.