Protein materials architecture by design

David I. Spivak

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

Presented on 2015/06/16
at NIST

Outline

Outline

Design process

- In design, different components are fit together to make a whole.
- The designer creates subsystems at various levels, in any order.
    - Design is often neither top-to-bottom, nor bottom-to-top.
    - Design happens organically, as the human thinks.
- For example in preparing this talk:
    - Sometimes I added ideas to the broad outline.
    - Sometimes I filled in details in specific slides.
    - I did not impose a chronological order on how this was done.
- Modularity makes this possible.

Modularity as good design

Advantages of modular design

- Independence: modules can be designed independently.
  - A module is not just independent of others at the same level.
  - It's also independent of its components or super-structures.
  - This is achieved by committing to interfaces.
- Reusability: components can be reused and repurposed.
  - Reuse speeds up the design process.
  - It allows incremental improvements to be made.
  - Standards can then be established.
- Intelligibility: the design can be explained and comprehended.
  - Easier to debug—confine the problem to a specific component.
  - Reduce complexity: a 15-component system vs. five groups of 3:

$$15! = 1.3\text{E}+12, \quad \text{vs} \quad 3! * 3! * 3! * 3! * 3! = 7.8\text{E}+3.$$

Modularity in nature

- Biological modularity is ubiquitous.
- It's good design because selective pressures can act appropriately.
  - The species can adapt locally, rather than having to adapt globally.
  - That is, improvements are made incrementally.
- Distinct functions are performed by groups of genes or proteins.
  - They certainly interact with other functional modules.
  - But to some degree they remain autonomous.
- Whatever the reason, it is a very common design strategy.

Hierarchical protein materials in nature

- Bones, muscles, skin are common components used in many animals.
  - They are all (slight modifications of) collagen.
  - Collagen is hierarchical structure built from interacting components.
- It's useful to consider these materials in terms of construction.
  - What are the ingredients we need?
  - By what process do we put these together to produce the material?
- Today, we'll investigate this issue from a mathematical perspective.

A mathematical theory of modularity

By modularity, I mean

- The creation of new building blocks from a collection of existing ones.
  - Creating a home entertainment system from a TV, receiver, speakers.
  - Each is a whole, but also can be used as a part—a building block.
  - The interactions between components determine a higher-level system.
- The reliance on the interface / implementation distinction.
  - Building blocks come in types, which tell us how they are to be used.
  - We don't have to know the implementation details in order to design.

There is a mathematical theory of this sort of modularity.

Materials architecture

- A special case: the architecture of complex materials.
- This is modular: we have building blocks and building instructions.
    - Our material library begins with basic building blocks.
    - We use building instructions to create higher-level structures.
    - These become building blocks in our library.
- We will talk about a special case: protein materials.

Talk outline

For the remainder of this talk, I will discuss:

- A project we did in materials science: software called *Matriarch*.
  - Matriarch stands for Materials Architecture.
  - (Joint work with Tristan Giesa, Ravi Jagadeesan, and Markus Buehler.)
  - It's a Python library that one can use to design protein materials.
- Category theory, a very brief introduction.
  - Focus on *operads*, a framework within CT, useful for design.
  - How the protein materials work generalizes to other design problems.
- What's next.
  - Using and generalizing matriarch for materials science research.
  - Branching out to other modular design environments (e.g., circuits, dynamical systems).

Outline

Materials design

- What we want: High-quality, environmentally-friendly materials.
    - Old idea: high-quality macro requires high-quality micro.
    - New idea: high-quality macro is achievable with cheap, abundant micro.
- Examples: silk, collagen.
    - These materials are made by animals by eating widely-available food.
    - The micro is cheap and abundant, but the result has excellent qualities.
    - Silk is stronger than steel; collagen is used in bone, skin, cartilage, etc.
    - These materials are assemblies of simple (amino acid) building blocks.
- How to mimic these amazing materials and fit them to our needs?
    - In the wet lab, you investigate their hierarchical structures.
    - If you need to modify something, you'll want to use computers.

Computational modeling

- A new 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.
- This requires a massive amount of computation.
  - You can't do all this in a wet lab.
  - Simulation allows you to play around with micro-structures.
  - "This amino acid is preventing what you want; can we get rid of it?"
  - Molecular dynamics (MD) simulators are used to run experiments.
- The current state of computational materials design.
  - There does not currently exist a general tool to create new microstructures.
  - You have to do everything (place atoms and bonds) by hand.
  - This is extremely tedious, and leads to problematic work-arounds.

The challenges to overcome

- The dilemma: spend time programming or equilibrating??
  - If you want to save labor time, you place atoms into straight chains.
  - But these take “forever”, often weeks, to equilibrate (settle into place).
  - Moreover, they may equilibrate to the wrong shape (local minimum).
- Computational modelers develop tricks.
  - Tiny pieces of code (scripts) that fulfill a current need.
  - These one-off scripts are difficult to share, reuse, and explain.
- All these problems can be solved simultaneously.
  - Make a language to synthesize hierarchical structures.
  - Place atoms near their final positions to reduce equilibration time.
  - An organized language eases communication and reuse.

Materials architecture

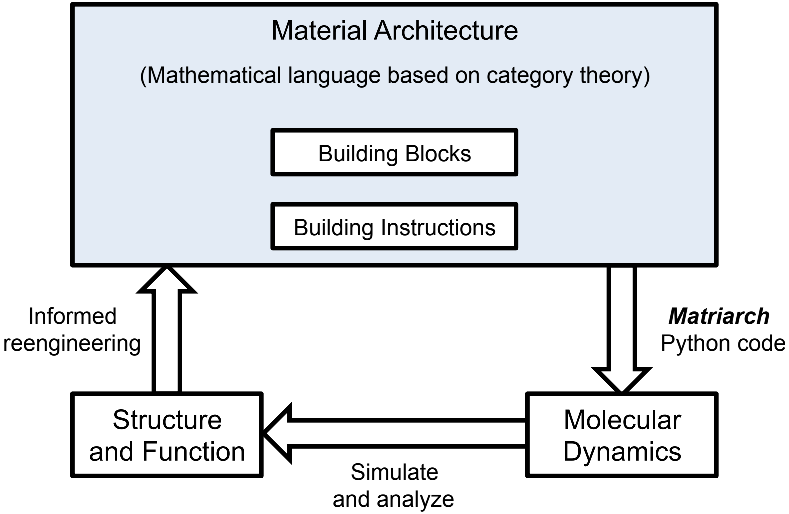
- In general, a modular design environment will be
  - Your set of building blocks,
  - Your set of building instructions.
  - The "tree" of architectures that can be assembled in this way.
- What is the modular design environment for materials?
  - Building blocks: proteins, from amino acids to collagen.
  - Building instructions: forming new blocks out of existing ones.
  - Hierarchical materials are built by combining these into programs.

Matriarch

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

- Matriarch is a language for materials architecture.
  - Starting with a library of amino acids
  - Attach, coil, overlay, etc. anything in your library to build new proteins.
  - Add these to your library, and keep building.
- Matriarch outputs .pdb files.
  - These can be visualized in VMD,
  - Experimental simulations can be run in LAMMPS, Gromacs, etc.
- Matriarch is a Python library.
  - It's open source and freely available online.
  - You can modify it, or just use it seamlessly with Python, etc.

Summary of the Matriarch design process



Matriarch's building blocks and building instructions

Building blocks:

- 20 standard amino acids, plus proline (for creating collagen).
- Users can import their own building blocks from PDB.

Building instructions:

- `attach`,
- `space`,
- `overlay`,
- `reverse`,
- `rigidMotion`,
- `twist`,
- `makeArray`.

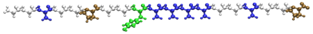
Matriarch programs:

- Any combination of building instructions applied to building blocks

Sample architectures

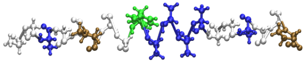
a

Strand1 = chain(seq1)



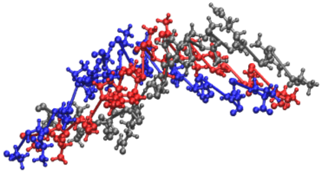
b

Hel1 = helix(Strand1, 1.0, 5.0)



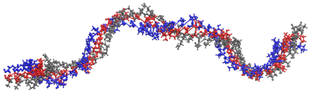
c

TH = collagen(Strand1, Strand2)



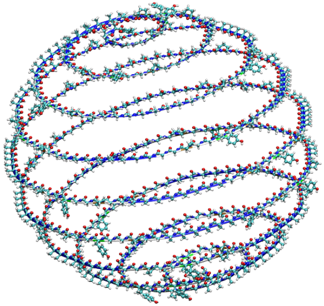
d

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



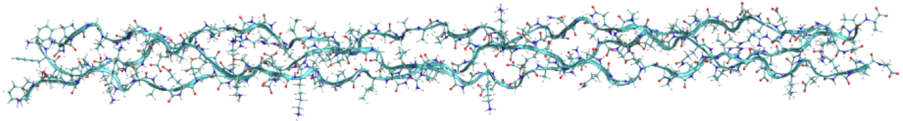
e

Apple = twist(Strand3, SSFunc)



Example of materials architecture: collagen

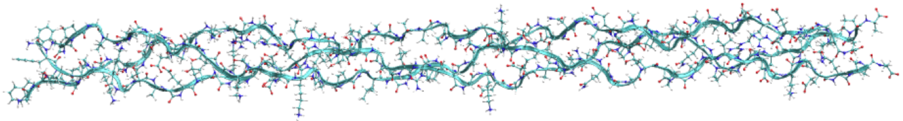
- Collagen is the most common protein in mammals.
- Its design is hierarchical.
  - A fibril of collagen is an array of tropocollagen molecules.
  - Each tropocollagen molecule is a right-handed triple helix.
  - Each of its three strands is a left-handed helix.
  - Each of these individual helices is 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)

Materials architecture

- A fibril of collagen is an array of tropocollagen molecules.
- Each tropocollagen module 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)

Demo

Let's run through a quick demonstration.

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

Matriarch as a design tool

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

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

Outline

What's this got to do with category theory?

- We said Matriarch was built "using" category theory (CT).
- But why is CT necessary?
- Isn't the Matriarch idea fairly simple and intuitive?
- Answer: category theory led to this intuitive design.
    - It's a feature that it disappears into the background.
    - There's a mathematical standard for designing this kind of software.
- Next we'll discuss operads.

Category theory was the software specification

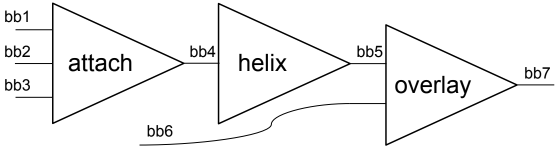
- To design Matriarch
  - We first understood the architecture problem using *operads*.
  - We then produced the operad for materials architecture.
  - This specified the code.
  - (Goguen proposed CT as a software specification language.)
- Here's what to think of when you hear the word "operad":
  - Building block types, or interfaces,
  - Building instructions, or arrangements,
  - A formula for composing these, i.e., nesting.
  - This formula guarantees equivalences between different programs.

$$\texttt{reverse}(\texttt{attach}(\texttt{x},\texttt{y})) = \texttt{attach}(\texttt{reverse}(y), \texttt{reverse}(x))$$

- Note: you don't need to know operads to work with Matriarch.

Explaining operads (a very rough sketch)

- What is the relation between operads and category theory?
  - Analogy: the relation between line integrals and calculus.
  - They are important for any expert to know, but it's just one piece.
  - They are useful for certain problems.
  - They are a generalization of the founding idea.
- An operad is a mathematical environment $\mathcal{O}$.
  - in $\mathcal{O}$ you have objects and you have many-to-one relationships.
  - A Matriarch command uses many, say X_1, X_2, X_3, to build one Y.



How operads are useful in design

- An operad can be thought of as a language of assembly.
  - I call it a modular design environment.
- As said above, an operad $\mathcal{O}$ consists of:
  - a set of interfaces,
  - a set of arrangements to form higher-level interfaces,
  - a formula for nesting arrangements.
  - This formula guarantees equivalences between different programs, e.g.,

$$\mathtt{reverse}(\mathtt{attach}(\mathrm{x},\mathrm{y})) = \mathtt{attach}(\mathtt{reverse}(y),\mathtt{reverse}(x))$$

- It is a language for thinking about building complex from simple.

Formal definition of operad

An operad $\mathcal{O}$ consists of

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

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

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

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

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

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

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 operadic model of Matriarch

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

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

Outline

Other modular environments

A quick look into what else this formalism can describe.

- Languages—context-free grammars—we've seen this already.
- Circuits and logic.
- Processes and recipes.

In each one, we will have interfaces, arrangements, and nesting.

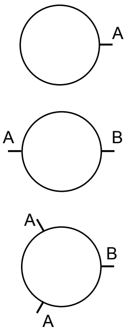
Context-free grammars

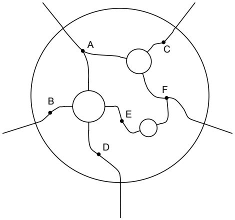
$$\begin{array}{rcl}
\langle\text{postal-address}\rangle & ::= & \langle\text{name-part}\rangle\ \langle\text{street-address}\rangle\ \langle\text{zip-part}\rangle \\
\langle\text{name-part}\rangle & ::= & \langle\text{personal-part}\rangle\ \langle\text{last-name}\rangle\ \langle\text{opt-suffix-part}\rangle\ \langle\text{EOL}\rangle \\
 & | & \langle\text{personal-part}\rangle\ \langle\text{name-part}\rangle \\
\langle\text{personal-part}\rangle & ::= & \langle\text{first-name}\rangle | \langle\text{initial}\rangle\ \text{"."} \\
\langle\text{street-address}\rangle & ::= & \langle\text{house-num}\rangle\ \langle\text{street-name}\rangle\ \langle\text{opt-apt-num}\rangle\ \langle\text{EOL}\rangle \\
\langle\text{zip-part}\rangle & ::= & \langle\text{town-name}\rangle\ \text{","}\ \langle\text{state-code}\rangle\ \langle\text{ZIP-code}\rangle\ \langle\text{EOL}\rangle \\
\langle\text{opt-suffix-part}\rangle & ::= & \text{"Sr."}\ |\ \text{"Jr."}\ |\ \langle\text{roman-numeral}\rangle\ |\ \text{""} \\
\langle\text{opt-apt-num}\rangle & ::= & \langle\text{apt-num}\rangle\ |\ \text{""}
\end{array}$$

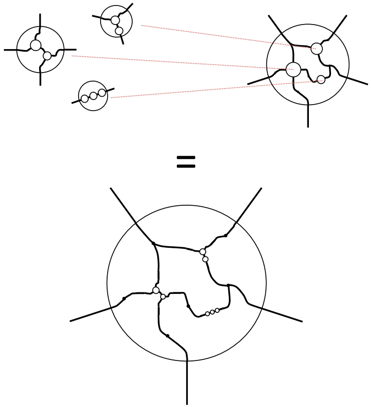
- Everything in ⟨brackets⟩ is an interface.
- Each line (production rule) is an arrangement.
- It's clear how to nest these production rules.

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

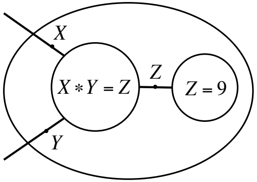
Interfaces	Arrangements	Nesting



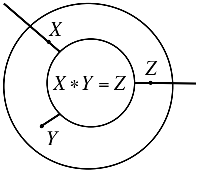




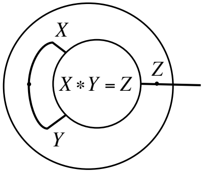
Databases



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



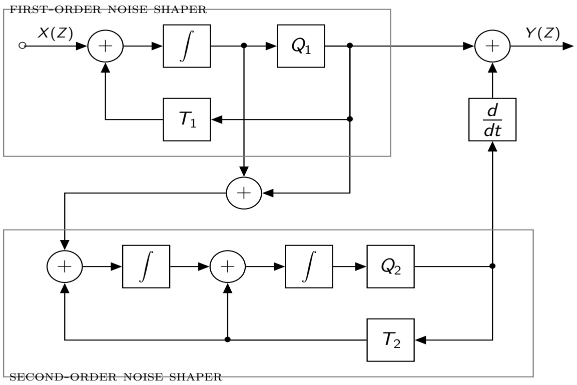
"all pairs of integers (X,Z) in which Z is divisible by X."



"all perfect squares Z"

Electrical circuits

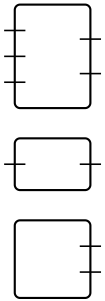
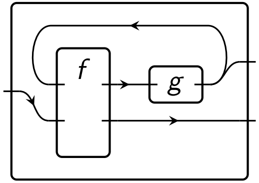
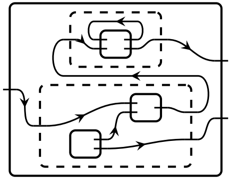
- Same kind of diagram;[2] very different semantics.



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

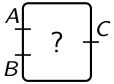
[2]Drawn by: Ramón Jaramillo. `http://www.texample.net/tikz/examples/noise-shaper/`

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

Interfaces	Arrangements	Nesting
		

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

Here is an interface with $\mathtt{input} = A \times B$ and $\mathtt{output} = C$.



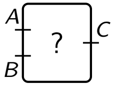
Definition

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

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

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

Here is an interface with $\mathtt{input} = A \times B$ and $\mathtt{output} = C$.

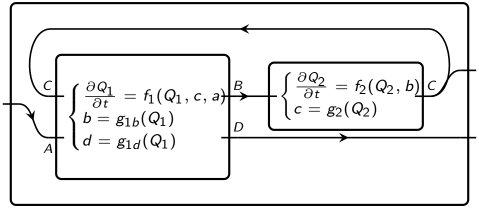


Definition

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

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

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


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

Definition

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

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

A matriarch-style program for dynamical systems

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

Back to materials

- I'm proposing that operads are the basic "arithmetic" of modularity.
  - The same idea, building one from many is used in:
  - Grammars, circuits, databases, control systems, protein materials.
- What else can we use operads for in materials science?
  - We can formulate experiments and recipes using an operad like $\mathcal{T}$.
  - It's an abstract language for putting recipes together.
- What else?

Scripting with Matriarch

- Matriarch is a Python library.
  - We can program using it.
  - Gromacs and LAMMPS have Python wrappers.
  - What can we do with this?

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

- Automate experiments.
  - Start with a known protein, say collagen or silk.
  - Write it parametrically as a building block.
  - Automatically mutate sequence and parameters, to improve properties.
- Or, use a genetic algorithm to search the space.

Other material architectures

- Matriarch is designed around hierarchical protein materials.
- What about other materials?
  - What's needed is an idea for how materials are formed hierarchically.
  - Think about building block types and building instructions.
  - From there, we construct an operad, which in turn specifies a program.

Summary

- There is a mathematical language for modular design.
  - The mathematics can serve as a proto-standard.
  - Different modular design environments can be compared with functors.
- Matriarch is a concrete example of this idea.
  - We can build hierarchical protein materials, such as collagen.
  - Once we've made something, it becomes a building block.
  - Building instructions are n-ary:
    They take n building blocks as input and produce a new one.
- This use of operads should work in general for hierarchical design.
  - Grammars, circuits, dynamical systems, databases.
  - Materials other than proteins.
  - Others?

Thank you

Thanks for the invitation to speak!