Operads for complex system design specification, analysis and synthesis[1]

John Foley
Metron, Inc.
foley@metsci.com
joint w/
Spencer Breiner, Eswaran Subrahmanian and John Dusel

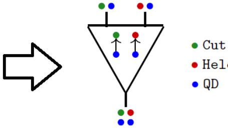
Applied Category Theory 2021

[1]This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. N66001-16-C-4048.

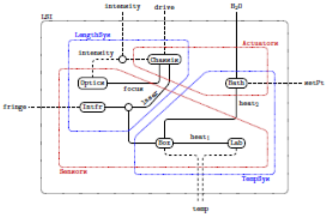
Three example applications motivate how typed operads address three issues for complex system design:

Specification

```
{'colors' : ['port', 'cut', ..., 'qd'],
 'directed' : {
    'carrying': {
        'cut': ['port'],
        'boat': ['port', 'cut'],
        ...,
        'qd': ['cut', ..., 'helo'] } } }
```



Analysis



Synthesis

$$m_{j+1} = m_j + M\Sigma_j$$

$M(\mathrm{UH60} \otimes \mathrm{HC130})$

$$\begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$m_{j+1} \geq M^s\Sigma_{j+1}$$

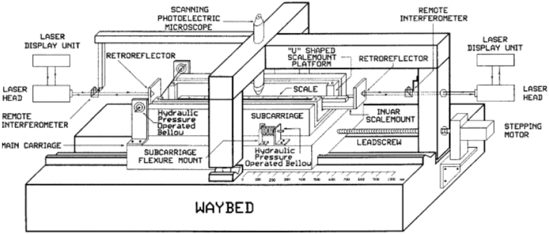
$M^s(\mathrm{UH60} \otimes \mathrm{HC130})$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The application domains:

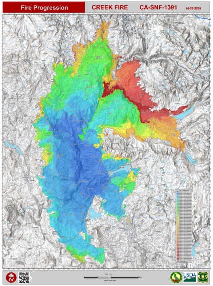
Maritime search and rescue (SAR) architectures

Precision measurement system



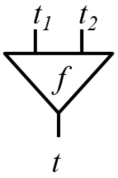
Beers & Penzes, J. Res. Natl. Inst. Stand. Technol. 104, 225 (1999)

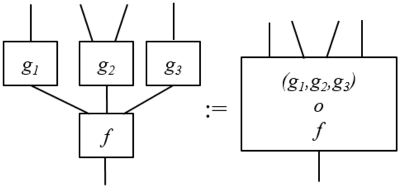
SAR tasking



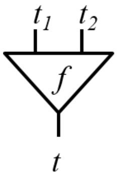
Typed operads naturally appear in many contexts where n objects are composed into a single object:

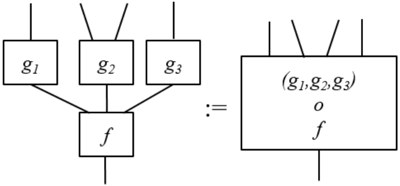
Operads	Tree	API	Equations	Systems
Types	Edges	Data types	Variables	Boundaries
Operations	Nodes	Methods	Operators	Architectures
Composites	Trees	Scripts	Evaluation	Nesting
Algebras	Labels	Implementations	Values	Models





A **typed operad** has





- a set T of **types**,
- sets of **operations** $O(t_1, ..., t_n; t)$ where $t_i, t \in T$,
- ways to **compose** operations

$$f \circ (g_1, \ldots, g_n) \in O(t_{1i}, \ldots, t_{1k_1}, \ldots, t_{n1}, \ldots t_{nk_n}; t),$$

- ways to permute the arguments of operations,

which obey some rules [9].

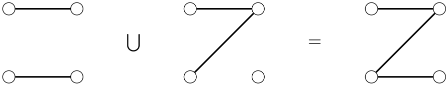
Specification becomes practical when simple, combinatorial ingredients define functorial semantics:

$$\text{Model}\colon \textbf{Syntax} \longrightarrow \textbf{Semantics}$$

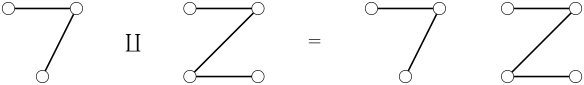
Here, we focus on specifying a typed operad for syntax.

Network models define how to overlay a **specific kind** of network and put such networks side-by-side. For example, simple graphs

- **overlay**



- and are **put side-by-side**



To construct a **network operad**:

- Define types of nodes C for your application
- Encode ways to combine these networks as a lax symmetric monoidal functor $F\colon \mathbf{S}(C) \to \mathbf{Cat}$ where $\mathbf{S}(C)$ is free on C
  - overlay $\leftrightarrow$ composition in target categories
  - put side-by-side $\leftrightarrow$ lax structure maps
- Apply symmetric **monoidal Grothendieck construction** [1, 8]
- Let $\mathrm{O}_F := \mathrm{op}(\int F)$ be the (typed) endomorphism operad: $\mathrm{op}(\mathbf{C})(c_1, \ldots, c_k; c) := \hom_{\mathbf{C}}(c_1 \otimes \cdots \otimes c_k, c)$

Theorem (Baez, F, Moeller, Pollard, [1])

The composite **functor**

$$\mathbf{NetMod} \xrightarrow{\int} \mathbf{SSMC} \xrightarrow{\mathrm{op}(-)} \mathbf{TypedOp}$$

constructs a network operad O_F *for each network model* F*.*

Once specification of a network model from simple, combinatorial ingredients is codified in a theorem–e.g.

Theorem (Baez, F, Moeller, Pollard, [1])

There is a functor

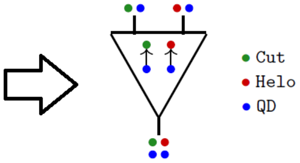
$$\Gamma\colon \mathbf{Mon} \to \mathbf{NetMod}$$

sending each monoid M to a network model $\Gamma(M)\colon \mathbf{S} \to \mathbf{Mon}$.

the construction can be reused in many contexts.

For example, to specify the atomic types (C) and relationships between types (family of monoids) for **search and rescue**:

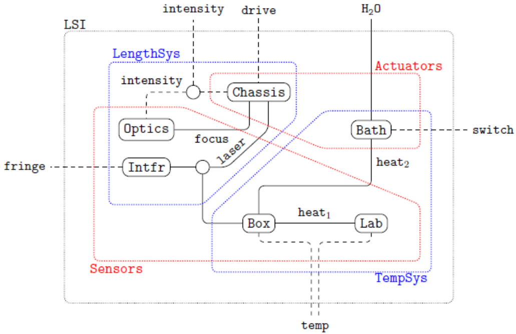
```
{'colors' : ['port', 'cut', ..., 'qd'],
 'directed' : {
    'carrying': {
        'cut': ['port'],
        'boat': ['port', 'cut'],
        ...,
        'qd': ['cut', ..., 'helo'] } } }
```



Compositionality guarantees *coherent* **analysis**.

That is, multiple, complementary analyses can be conducted.

$$\mathcal{W} = \text{diagram for analyzes} =$$



In particular, different semantic models can address different aspects of a design problem–e.g. function vs. control.

From the functional perspective, we can analyze the impact of component failure:

$P_{\mathtt{f}}$	`LengthSys`	$\mapsto$	40%	$P_{\mathtt{g}}$	`Sensors`	$\mapsto$	28%
	`TempSys`	$\mapsto$	60%		`Actuators`	$\mapsto$	72%
$P_{\mathtt{l}}$	`Intfr`	$\mapsto$	10%	$P_{\mathtt{s}}$	`Lab`	$\mapsto$	21.4%
	`Optics`	$\mapsto$	30%		`Bath`	$\mapsto$	21.4%
	`Chassis`	$\mapsto$	60%		`Optics`	$\mapsto$	42.9%
$P_{\mathtt{t}}$	`Bath`	$\mapsto$	80%		`Intfr`	$\mapsto$	14.3%
	`Box`	$\mapsto$	10%	$P_{\mathtt{a}}$	`Chassis`	$\mapsto$	33.3%
	`Lab`	$\mapsto$	10%		`Bath`	$\mapsto$	66.7%

Semantics in the operad of probabilities $\mathcal{W} \to \mathbf{Prob}$, in which relative probabilities compose by multiplication, describe how components contribute to failure probability.

From the control perspective, we can analyze dynamics.

For example, the `laser` interaction is parameterized by

- $T_{\texttt{laser}} :=$ temperature
- $P_{\texttt{laser}} :=$ pressure
- $RH_{\texttt{laser}} :=$ relative humidity
- $\lambda_0 :=$ laser wavelength (in vacuum)

which varies dynamically for $t \in \tau$

$$\mathrm{Traj}(\texttt{laser}) \cong ([-273.15, \infty)] \times [0, \infty)] \times [0, 1])^{\tau} \times [0, \infty) \subseteq (\mathbb{R}^3)^{\tau} \times \mathbb{R},$$

coupling `Chassis`, `Intfr`, and `Box`.

Semantics in the operad of relations $\mathcal{W} \to \mathbf{Rel}$ describe possible behaviors for joint interaction and component states.

Both semantic models leverage limited **focus**:

- $\mathcal{W}$ is only a small fragment of the operad of port graphs [3]
- this means only the specific semantics for the *problem at hand* need to be defined
- $\mathcal{W}$ could be extended for more detailed analyzes

which is controlled by limiting **syntax**.

Each model leverages a specific **filter**

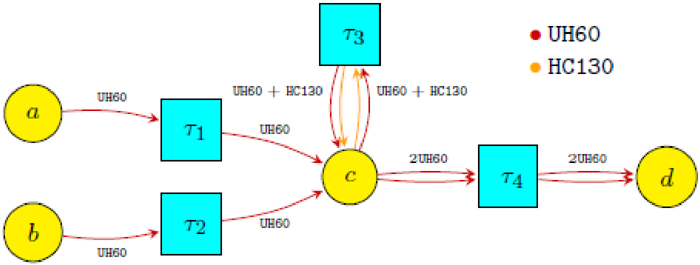
- failure probability semantics are simple, modeled in **Prob**
- semantics for dynamics are more sophisticated, modeled in **Rel**, in the tradition of Jan Willems's behavioral approach

which is controlled by the **semantic model**.

Check out our paper [6] for brief discussion of using natural transformations as a 'filter of filters'.

In theory, **synthesis** is straightforward when simple objects and morphisms generate syntax.

A Petri net declares primitive tasks and how they fit together:



- Transitions (squares) define primitive tasks $\tau_i \in T$
- Arcs indicate types involved in τ_i
- Species (circles) are coordination locations.

Petri nets are sufficient to coordinate multiple agent types [2, 5] and known to generate monoidal categories [4, 7].

That is, Petri nets provide simple, combinatorial ingredients to define a network model to task agents.

The construction of the network model $\Lambda\colon \mathbf{S}(C) \to \mathbf{Cat}$:

- $C :=$ set of token colors
- Transitions must preserve the number of tokens of each color
- $\Lambda(c_1 \otimes \cdots \otimes c_n) :=$ allowed behaviors for assembled agents

Theorem (F, [5])

There is network model $\Lambda\colon \mathbf{S}(C) \to \mathbf{Cat}$ with

$$\Lambda(c_1 \otimes \cdots \otimes c_n) \subset \mathbf{Free}(T)^n$$

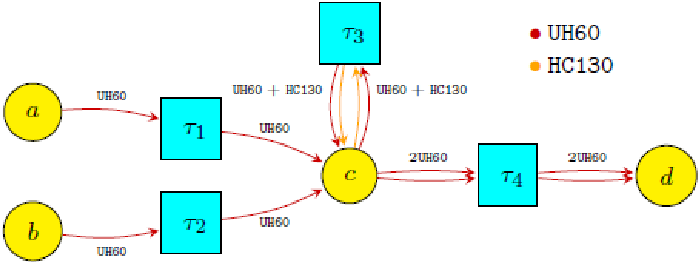
s.t. each projection is a sequence of tasks for a single agent and T is the set of transitions in a colored Petri net.

For example:

- $\Lambda(\mathtt{HC130}) := \langle a, b, c, d \rangle \subset \mathbf{Free}(T)$
- $\Lambda(\mathtt{UH60}) := \langle a, b, c, d, \tau_1 \colon a \to c, \tau_2 \colon b \to c \rangle \subset \mathbf{Free}(T)$

but $\Lambda(\mathtt{HC130} \otimes \mathtt{UH60}) \subset \mathbf{Free}(T) \times \mathbf{Free}(T)$ is generated by

- all pairs (f, g), (g, f) s.t. $f \in \Lambda(\mathtt{HC130})$, $g \in \Lambda(\mathtt{UH60})$
- $(\tau_3 \colon c \to c, \tau_3 \colon c \to c)$, a new joint behavior



We prototyped automated synthesis with a constraint program.

Idea: enforce type matching

- Types $\to$ boolean vectors m_j
- $||$ composition $\to$ boolean Σ_j

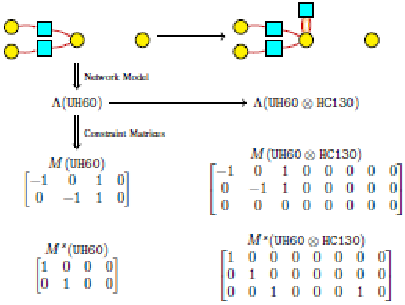
To compute target of morphism:

$$m_{j+1} = m_j + M\Sigma_j$$

To match target to source:

$$m_{j+1} \geq M^s\Sigma_{j+1}$$

NB: inequality allows for identities.



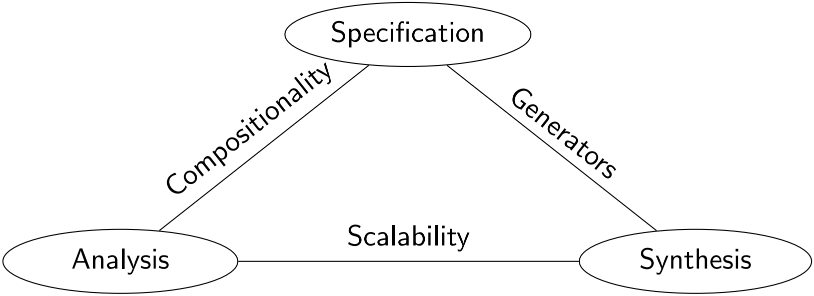
More tasks become possible with more agents and the dimensions of $M(-)$ and $M^s(-)$ increase.

In practice, the direct translation to a constraint program is not computationally efficient, so more research is needed.

We discussed how 3 examples address issues for automated design:

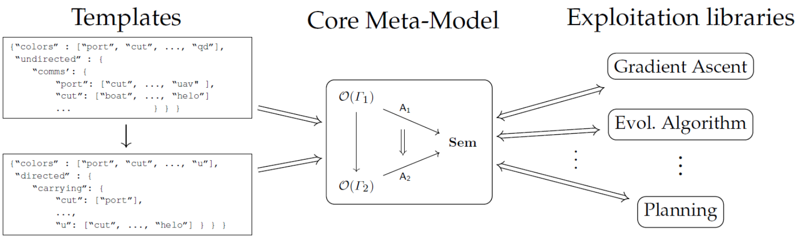
- Specification
- Analysis
- Synthesis

To make automated design synthesis practical, these three threads will need to be woven together.



There are many directions for further research:

- More systematic methods to specify semantics?
- More examples of focused analysis for complex systems?
- Unify analytic and synthetic perspectives?
- Exploit multiple representations for computational efficiency?



Check out our paper [6] for further discussion.

THANK YOU!

Further reading:

- Operads for complex system design specification, analysis and synthesis [6]
- Network models [1]
- Modeling hierarchical system with operads [3]
- Network models from Petri nets with catalysts [2]

This work was supported by the DARPA Complex Adaptive System Composition and Design Environment (CASCADE) project under Contract No. N66001-16-C-4048.

We thank John Baez, Tony Falcone, Ben Long, Tom Mifflin, John Paschkewitz, Ram Sriram and Blake Pollard for helpful discussions.

[1] J. C. Baez, J. D. Foley, J. Moeller and B. S. Pollard, Network models, *Theor. Appl. Categ.* **35** 20 (2020), 700–744.

[2] J. C. Baez, J. D. Foley and J. Moeller, Network models from Petri nets with catalysts, *Compositionality* **1** 4 (2019).

[3] S. Breiner, B. Pollard, E. Subrahmanian and O. Marie-Rose, Modeling Hierarchical System with Operads, *Proc. of ACT 2019*, (2020) 72–83.

[4] J. C. Baez and J. Master, Open Petri nets, *Math. Struct. Comp. Sci.* **30** 3 (2020), 314–341.

[5] J. D. Foley, An example of exploring coordinated SoS behavior with an operad and algebra integrated with a constraint program, 2018.

[6] J. D. Foley, S. Breiner, E. Subrahmanian and J. M. Dusel, Operads for complex system design specification, analysis and synthesis, *Proc. R. Soc. A* **477** (2021), 20210099.

[7] J. Meseguer and U. Montanari, Petri nets are monoids, *Inf. Comput.* **88** (1990), 105–155.

[8] J. Moeller and C. Vasilakopoulou, Monoidal Grothendieck Construction, *Theor. Appl. Categ.* **35** 31 (2020), 1159–1207.

[9] D. Yau, *Colored Operads*, American Mathematical Society, Providence, Rhode Island, 2016.