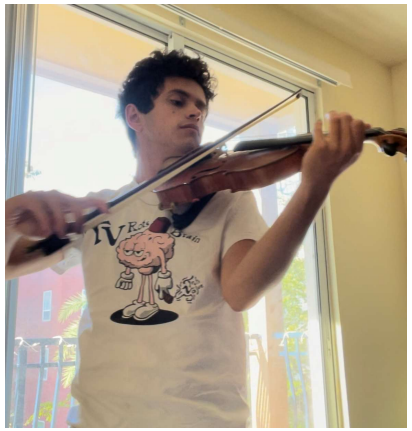
Sharing vector variables via typed UWDs in AlgebraicOptimization.jl

A summary of my work so far

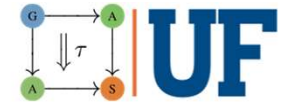
Sam Cohen

About me

Florida Community Innovation



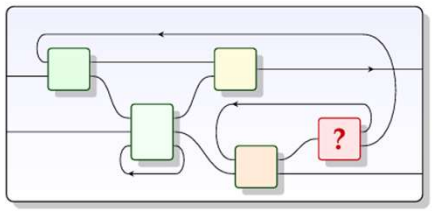
Prepping for GATAS

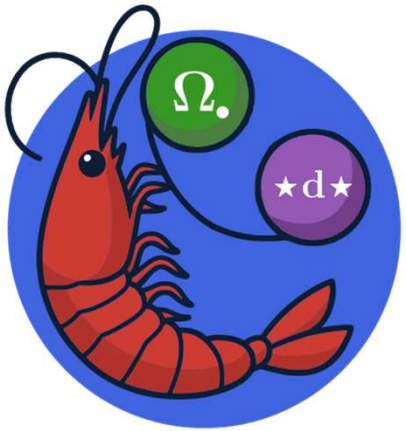


julia

Seven Sketches in Compositionality:

An Invitation to Applied Category Theory



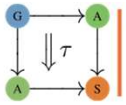


A Compositional Framework for First-Order Optimization

Tyler Hanks Matthew Klawonn Matthew Hale Evan Patterson
James Fairbanks

Abstract

Optimization decomposition methods are a fundamental tool to develop distributed solution algorithms for large scale optimization problems arising in fields such as machine learning, optimal control, and operations research. In this paper, we present an algebraic framework for hierarchically composing optimization problems defined on hypergraphs and automatically generating distributed solution algorithms that respect the given hierarchical structure. The central abstractions of our framework are operads, operad algebras, and algebra morphisms, which formalize notions of syntax, semantics, and structure preserving semantic transformations respectively. These abstractions allow us to formally relate composite optimization problems to the distributed algorithms that solve them. Specifically, we show that certain classes of optimization problems form operad algebras, and a collection of first-order solution methods, namely gradient descent, Uzawa's algorithm (also called gradient ascent-descent), and their subgradient variants, yield algebra morphisms from these problem algebras to algebras of dynamical systems. Primal and dual decomposition methods are then recovered by applying these morphisms to certain classes of composite problems. Using this framework, we also derive a novel sufficient condition for when a problem defined by compositional data is solvable by a decomposition method. We show that the minimum cost network flow problem satisfies this condition, thereby allowing us



UF

Understanding the problem

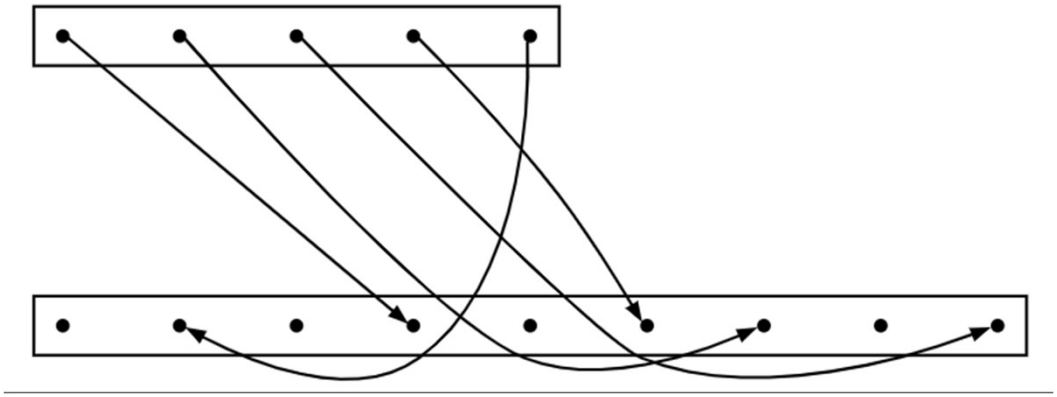
Why we need vector variable sharing in AlgebraicOptimizaiton.jl

Understanding the problem

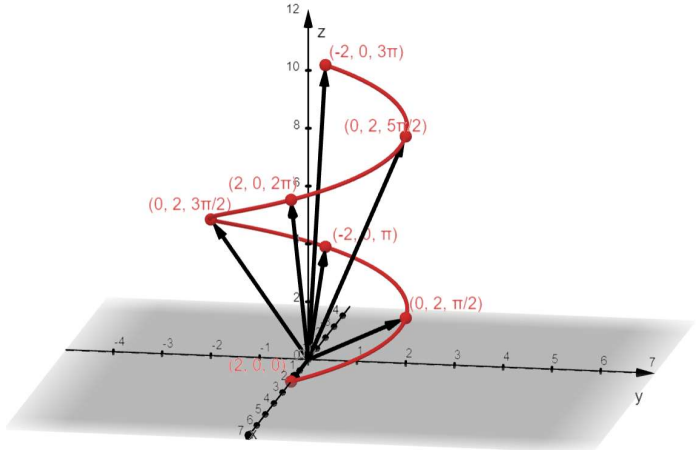
Too many variables

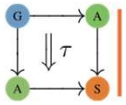
```
@relation (a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z ) begin
    f(a, c, d, e, m, n, o, s, t, v, u, w, x, y, z)
    g(u, w, y, a, c, d, e, m, n, o, s, t, v, f, g, l, n)
    h(h, i, j, k, l, m, n, o, p, q, r, s, t, x, y, a, c)
end
```

Variable sharing is confusing



Vector functions are useful

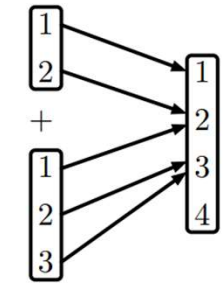




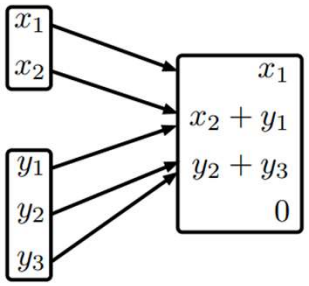
Solving the problem

My work so far

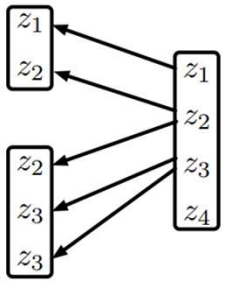
Vector pullback and pushforward functions



$\phi\colon [2] + [3] \to [4]$



$\phi_*(x, y)\colon \mathbb{R}^2 \times \mathbb{R}^3 \to \mathbb{R}^4$



$\phi^*(z)\colon \mathbb{R}^4 \to \mathbb{R}^2 \times \mathbb{R}^3$

Pushforward

```
"""     pushforward_matrix(f::FinFunction, v::Vector{Vector{Float64}})

The pushforward of f : n → m is the linear map f_* : Rⁿ → Rᵐ defined by
f_*(y)[j] =  ∑ y[i] for i ∈ f⁻¹(j).
"""
function pushforward_function(f::FinFunction, v::Vector{Vector{Float64}})::Vector
    output = [[] for _ in 1:length(codom(f))]
    for i in 1:length(dom(f))
        if isempty(output[f(i)])
            output[f(i)] = v[i]
        else
            output[f(i)] += v[i]
        end
    end
    return output
end
```

Pullback

```
"""     pullback_function(f::FinFunction, v::Vector)

The pullback of f : n → m is the linear map f^* : Rᵐ → Rⁿ defined by
f^*(y)[i] = y[f(i)].
"""
function pullback_function(f::FinFunction, v::Vector)::Vector
    return [v[f(i)] for i in 1:length(dom(f))]
end
```

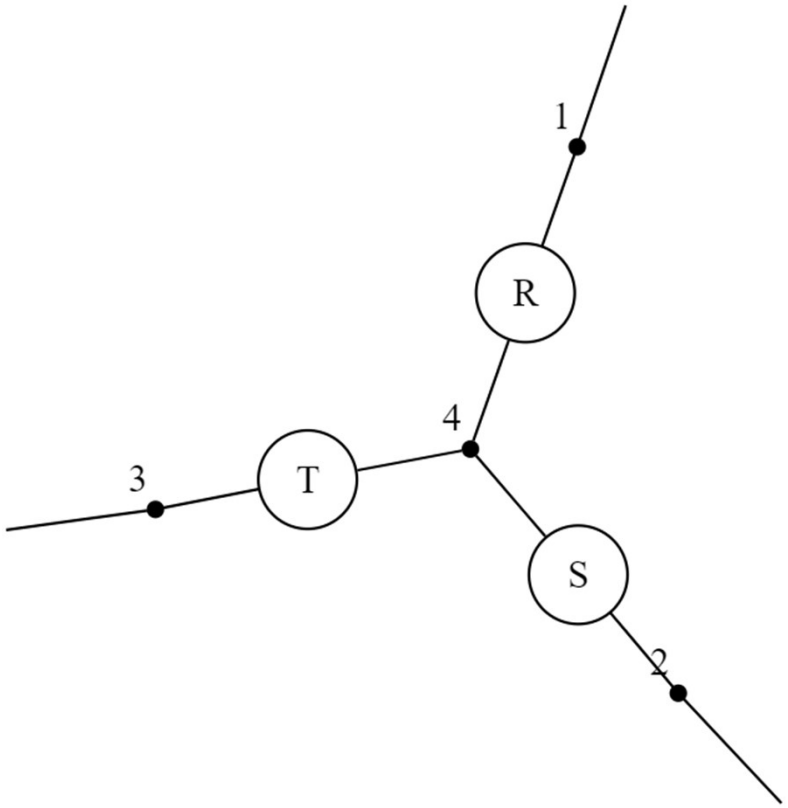
Int-typed UWDs

```
# Typed by Ints
#------

@relation (x,y,z) where (x::1, y::2, z::3, w::4) begin
  R(x,w)
  S(y,w)
  T(z,w)
end
```

```
function draw(uwd)
    to_graphviz(uwd, box_labels=:name, junction_label
end

function draw_types(uwd)    # Add better typing and er
    to_graphviz(uwd, box_labels=:name, junction_label
end
```



Vector variable sharing in Optimizers.jl


```
mA = v -> [[v[1][1]],  [v[2][1], v[1][1]], [v[3][2], v[3][1], v[4][4]],  [v[4][4], v[3][[1
mB = v -> [[6, 6, 6, 6, 6, 6], [2, 2], [7, 7, 7, 7, 7, 7, 7]]
mC = v -> v
```

```
x1::Vector{Vector{Float64}} = [[1], [1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2, 3, 4, 5], [1, 2, 3, 4, 5, 6], [1, 
tsteps = 100
r1 = simulate(discretization_of_composites, x1, tsteps) 9-element Vector{Vector{Float64}}:
r2 = simulate(composite_of_discretizations, x1, tsteps)
```

```
@test r1 ≈ r2
  Test Passed
```

Other minor contributions

Experimenting with ForwardDiff and ComponentArray

```
comp_rs = ComponentArray(r=[2, 3], s=

ForwardDiff.derivative(a -> basic_lin
ForwardDiff.derivative(b -> basic_lin
ForwardDiff.derivative(c -> basic_lin
ForwardDiff.derivative(d -> basic_lin
```

Corrected inconsistency between the paper and the code

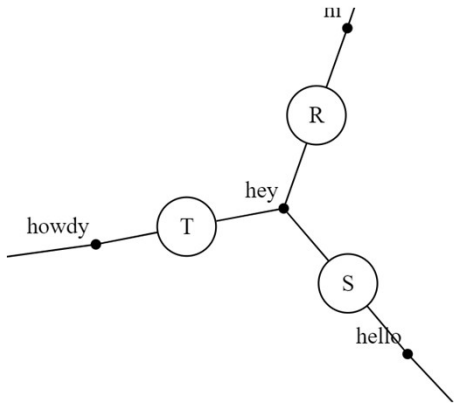
```
    continue
elseif src(g,e) == v
    A[v,e] = -1
elseif tgt(g,e) == v
    A[v,e] = 1
end
```

Bug fixes in AlgebraicOptimization.jl

```
# hello("World")
```

```
5
6  d = @relation (x,y,z) begin
7      f(w,x)
8      g(u,w,y)
9      h(u,w,z)
10 end
11
```

Expr typed UWDs



Eliminating the problem

What's up next

Goals

1. Implement vector variable sharing in Objectives.jl
2. UX for AlgebraicOptimization.jl
3. Implement ADT approach for Catlab @relation macro
4. Do fun math stuff!

Thanks for listening!

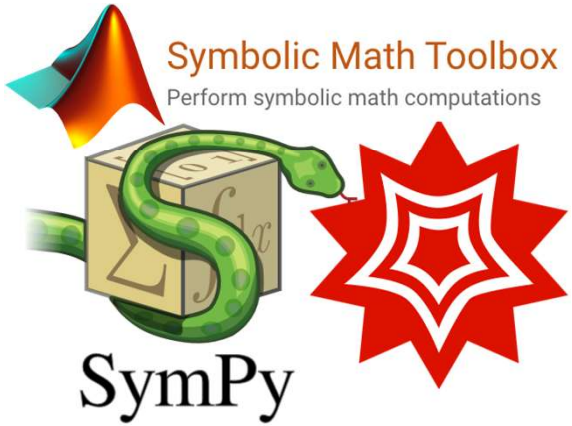
Vector functions in Optimizers.jl

Arbitrary Code

Domain Specific Languages

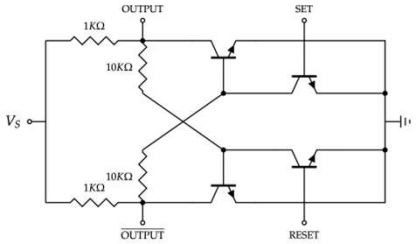
Modeling Frameworks

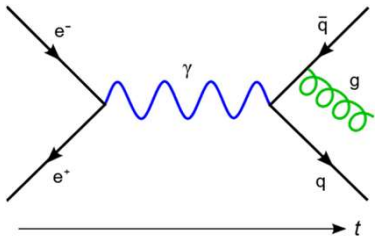
Computer Algebra Systems

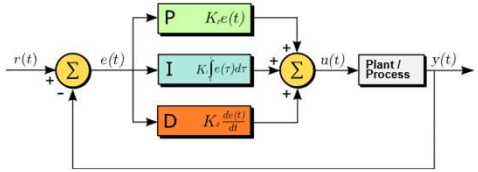


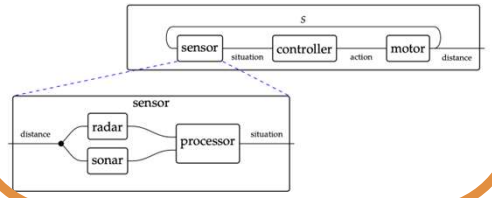
Vision: Model Aware Scientific Computing with Categories

All Possible Modeling Frameworks

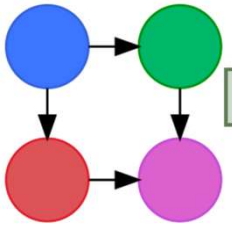




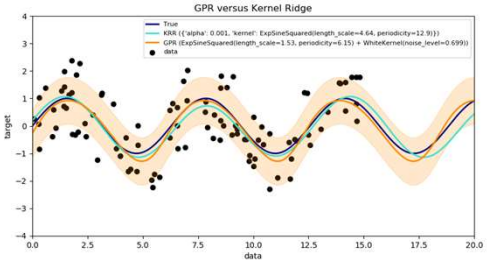


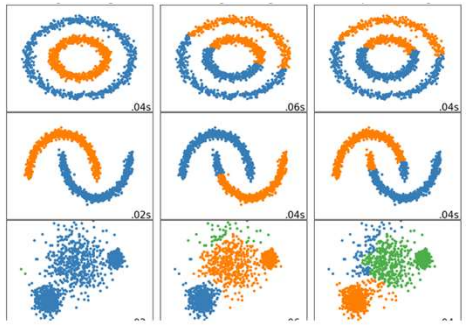


AlgebraicJulia



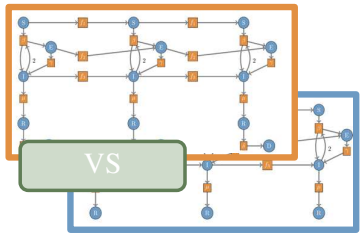
All Possible Modeling Tasks for Scientists

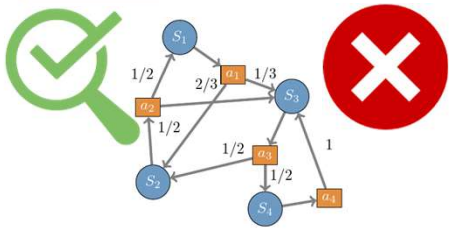




Calibration

Selection

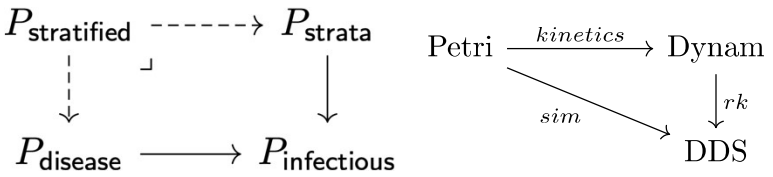




Comparison

Verification & Validation

Two Paths to Applying Category Theory to Scientific Computing



Math → CT (ACT)

```
function stratify(disease::Petri, strata::Petri)
  stratified = pullback(disease, strata)
  return object(stratified)
end
function sim(P::Petri)
  prob = ODEProblem(kinetics(P))
  return solve(prob, (t_o,t_e), alg=RungeKutta())
end
```

AlgebraicJulia

formalization

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash_D x : \sigma}$$

$$\frac{\Gamma \vdash_D e_0 : \tau \to \tau' \qquad \Gamma \vdash_D e_1 : \tau}{\Gamma \vdash_D e_0\ e_1 : \tau'}$$

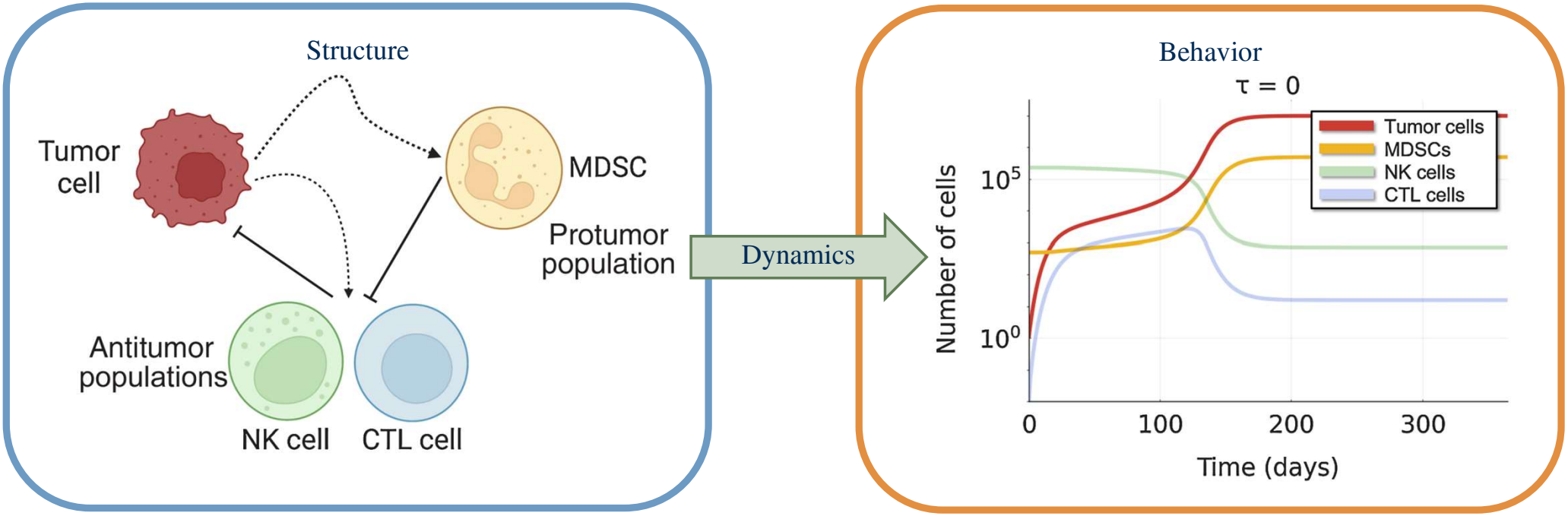
$$\frac{\Gamma,\ x : \tau \vdash_D e : \tau'}{\Gamma \vdash_D \lambda\, x\ .\ e : \tau \to \tau'}$$

Logic → Code (Func. Prog.)

```
rungekutta:: Float -> Float -> Float
f:: Float -> Float -> Float
scanl::(a -> a -> a) -> (a,a) -> List a -> List a
rungekutta (t, y) t' = (t', y + h*(k1 + 2.0*k2 + 2.0*k3 + k4)/6.0)
  where
  h = t' - t
  k1 = f t y
  k2 = f (t + 0.5*h) (y + 0.5*h*k1)
  k3 = f (t + 0.5*h) (y + 0.5*h*k2)
  k4 = f (t + 1.0*h) (y + 1.0*h*k3)

f t y = (t**2 - y**2)*sin(y)
scanl rungekutta (0, -1) [0.01, 0.03, 0.04, 0.06]
```

Biologists Study Regulation in Complex Systems



CANCER IMMUNOLOGY RESEARCH | RESEARCH ARTICLE

Myeloid-Derived Suppressor–Cell Dynamics Control Outcomes in the Metastatic Niche

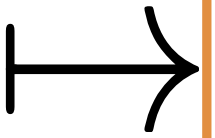
Jesse Kreger[1], Evanthia T. Roussos Torres[2], and Adam L. MacLean[1]

Mathematical Formalization of Biological Methods

Biological Concepts

- Data of a Regulatory Network
- Regulation Interactions
- Qualitative Interpretation

- Motifs
- Mechanistic Explanations
- Quantitative Analysis
- Complex Systems



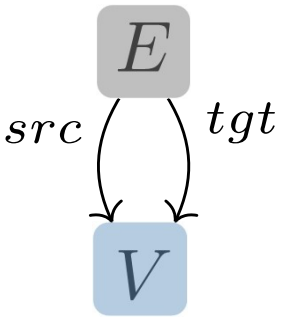
Mathematical Model

- Attributed C-Set for Signed Graphs
- Signed Categories
- Functorial Semantics
  SgnGraph -> SgnCat
- Representable Functors
- Signed Petri Nets Lifting Problem
- Functorial Semantics
  SgnGraph -> Para(Dynam)$_+$
- Colimits

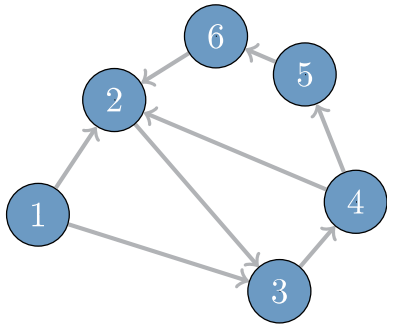
Attributed $\mathcal{C}$-Sets: Categorical Data Structures

Graphs are ubiquitous because they a simple & useful structure

Gr



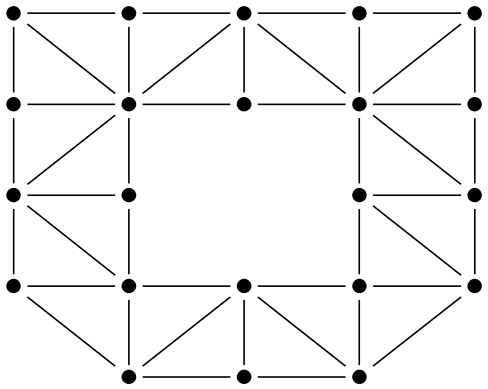
Gr − Set



Delta

$$T \;\overset{\partial_{2,0},\,\partial_{2,1},\,\partial_{2,2}}{\longrightarrow}\; E \;\overset{\partial_{1,0},\,\partial_{1,1}}{\longrightarrow}\; V$$

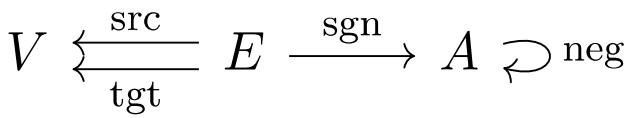
Delta−***Set***

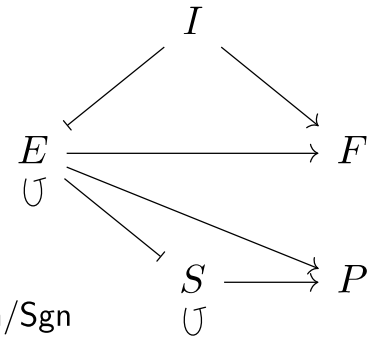


Signed Graphs and Signed Petri Nets

Represent Regulatory Networks (RegNet) as Signed Graphs

Sgn*Gr*

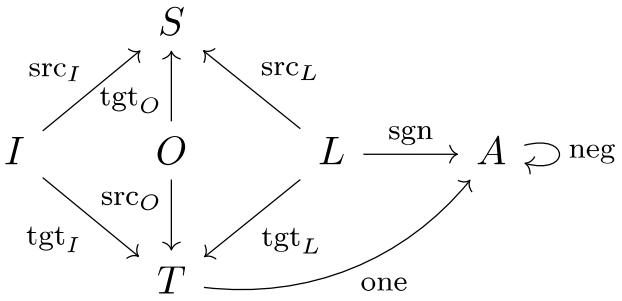




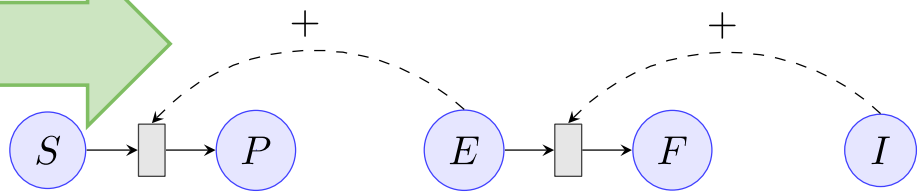
SgnGraph := Graph/Sgn

Represent Biochemical Processes as Signed Petri Nets

SgnPetri

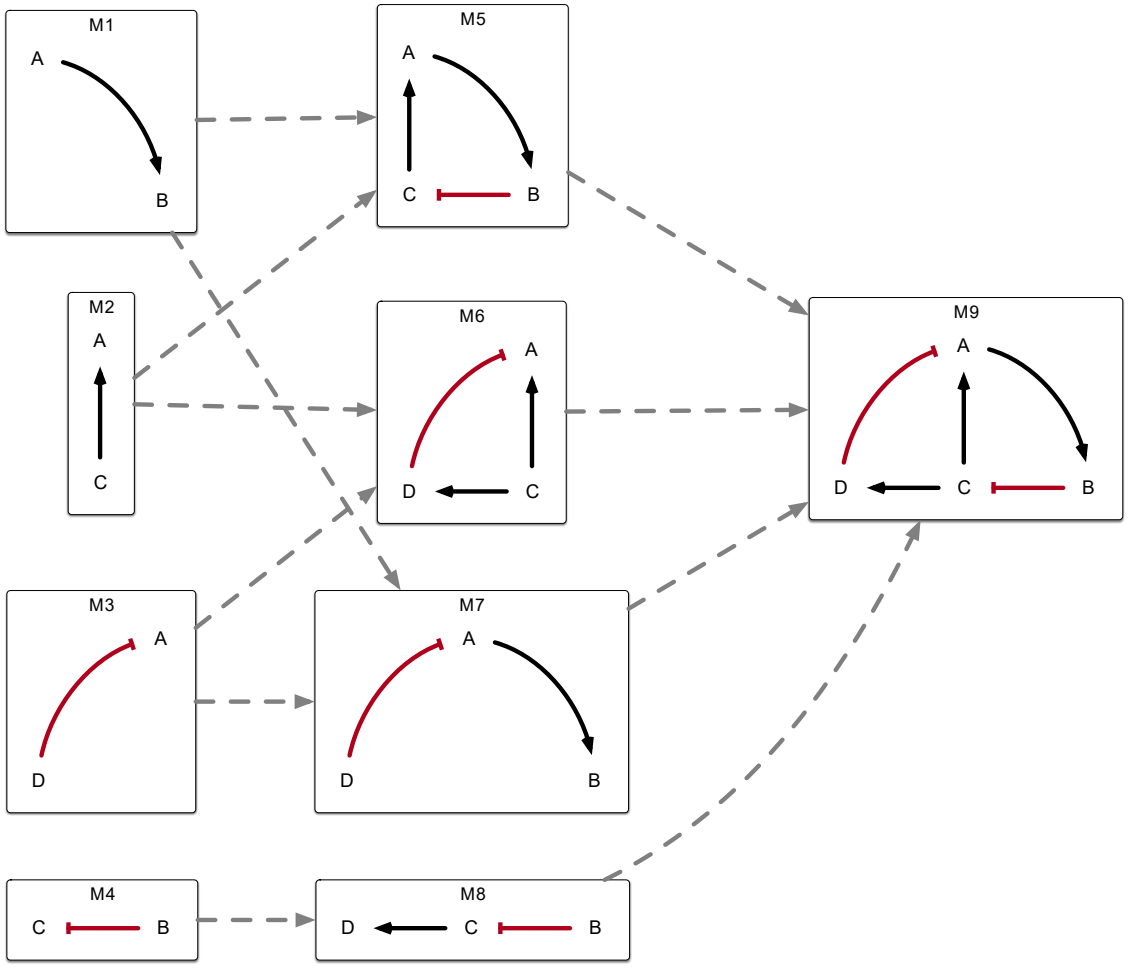


SgnPetri := LPetri/P_{Sgn}



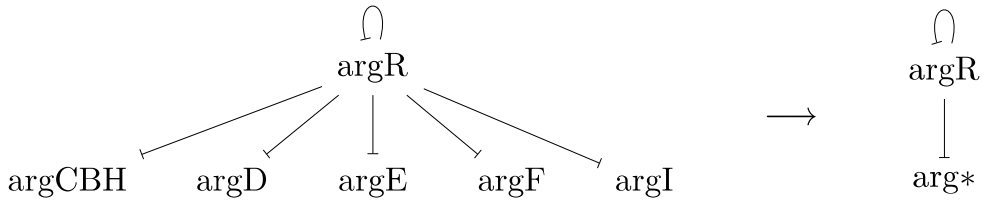
Topology of the Regulatory Network

- Represent Regulatory Networks (RegNet) as Signed Graphs
- Use subobject biheyting algebra to define a topology of subnetworks
- Future work: apply to topology tools to analyze the network structure



Morphisms of Signed Graphs Are Too Strict

- A graph homomorphism can pick out subgraphs or collapse edges



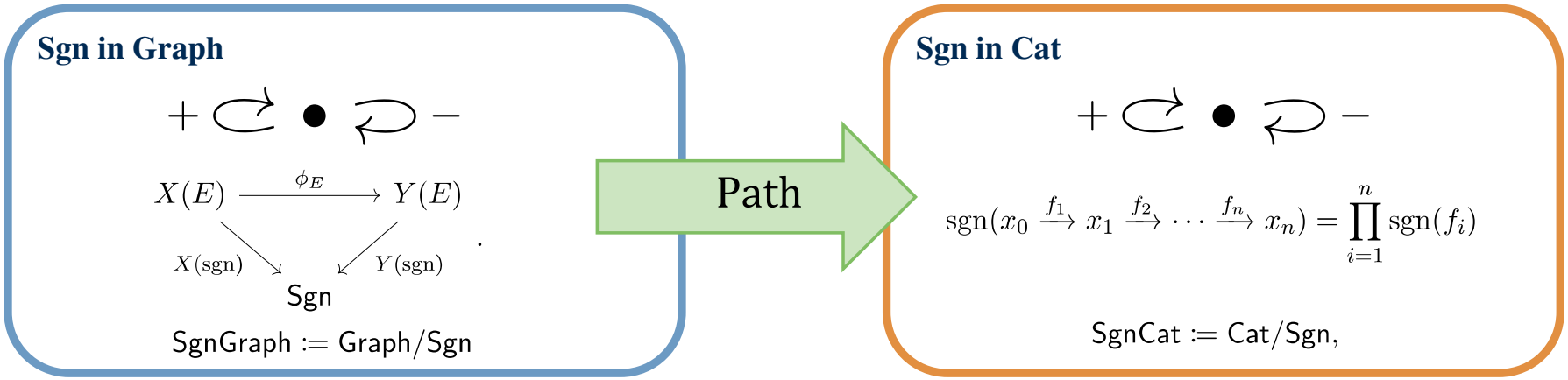
- But what do we do about paths?

$$A \longrightarrow B \longrightarrow C \quad \Longrightarrow \quad A \longrightarrow\!\!\!\!\!\longrightarrow C$$

$$A \longrightarrow B \longrightarrow\!\!\!\!\!\shortmid\ C \quad \Longrightarrow \quad A \longrightarrow\!\!\!\!\!\!\!\shortmid\ C$$

Qualitative Behavior with Kliesli Categories

- Signed Graphs represent the data of a regulatory network
- How do we interpret them?



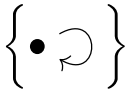
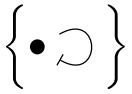
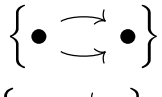
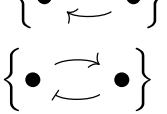
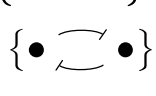
Regnets is the Kliesli Category of this Functor

Objects are Signed Graphs

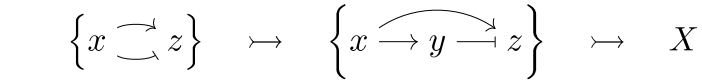
Morphisms are Signed Functors between their Path Categories

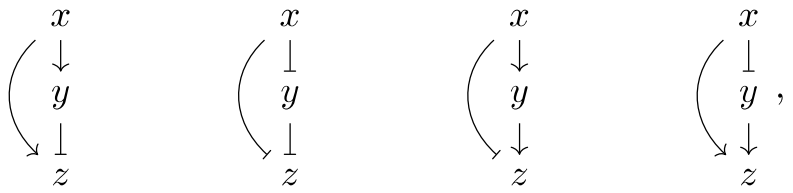
Motifs are figures of a fixed shape!

The standard definition of figures of shape s in space X
$Hom_{Monic}(s, X)$ recovers the notion of biological motif

Motif	Generic instance
Positive autoregulation	$L_+ :=$ 
Negative autoregulation	$L_- :=$ 
Coherent feedforward loop	$I_{++} :=$ 
Incoherent feedforward loop	$I_{\pm} :=$ 
Positive feedback loop	$L_{++} :=$ 
Negative feedback loop	$L_{\pm} :=$ 
Double-negative feedback loop	$L_{--} :=$ 

We get a category of motif refinement from functoriality of Hom

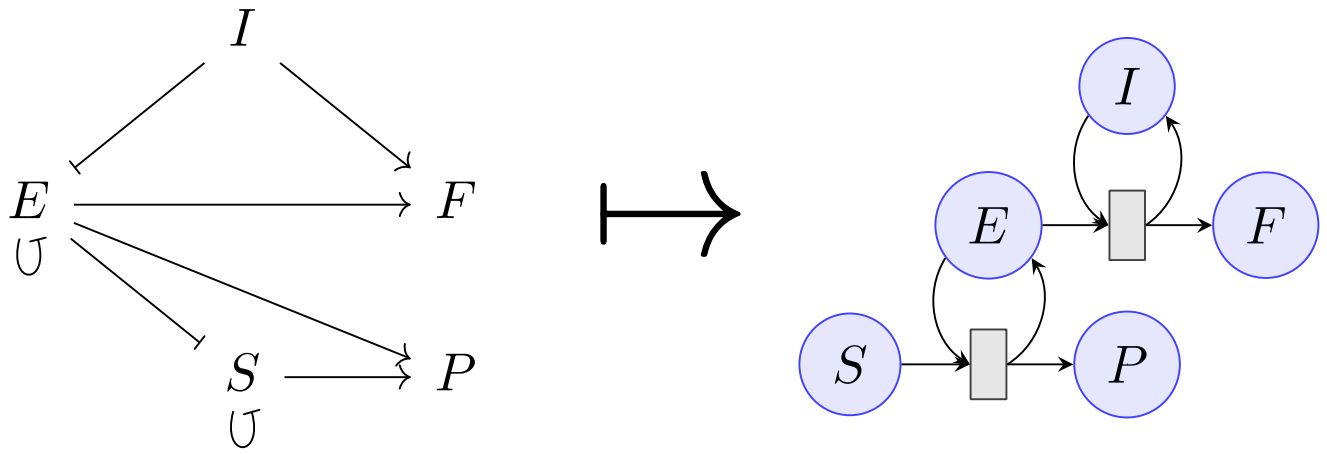




Common motifs in biochemical regulation networks [**alon2007**; **tyson2010**]

Biology is about Explaining Observed Phenomena

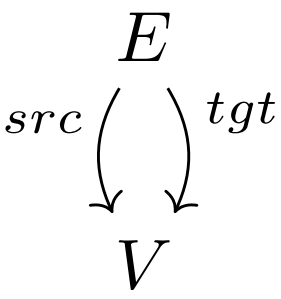
- Regulatory Networks are what you can observe by perturbing a system and measuring responses.



- Biology is about inferring the mechanisms that cause those responses.
- Regulatory Network can't directly make quantitative predictions

Important to easily change data model

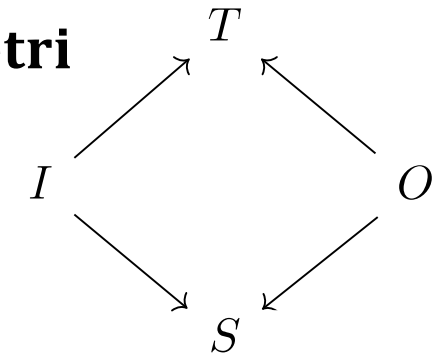
Graph



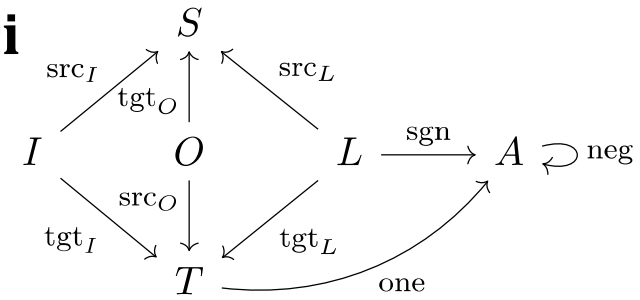
SgnGraph

$$V \underset{\text{tgt}}{\overset{\text{src}}{\leftleftarrows}} E \xrightarrow{\text{sgn}} A \circlearrowright \text{neg}$$

Petri



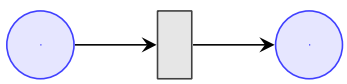
SgnPetri



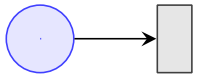
Because we operate in General Abstract Nonsense our software is more flexible

Constructing Interactions of a Mechanism

We can define a functor Int from $\boldsymbol{SgnPetri}$ to $\boldsymbol{SgnGraph}$

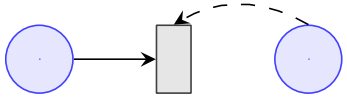


(a) Input-output pair to transition

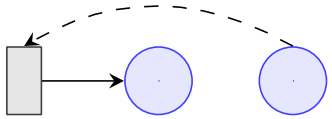


(b) Input to transition

$$\subset A \longrightarrow B \qquad A \supset$$



(c) Input to transition with incident link



(d) Output from transition with incident link

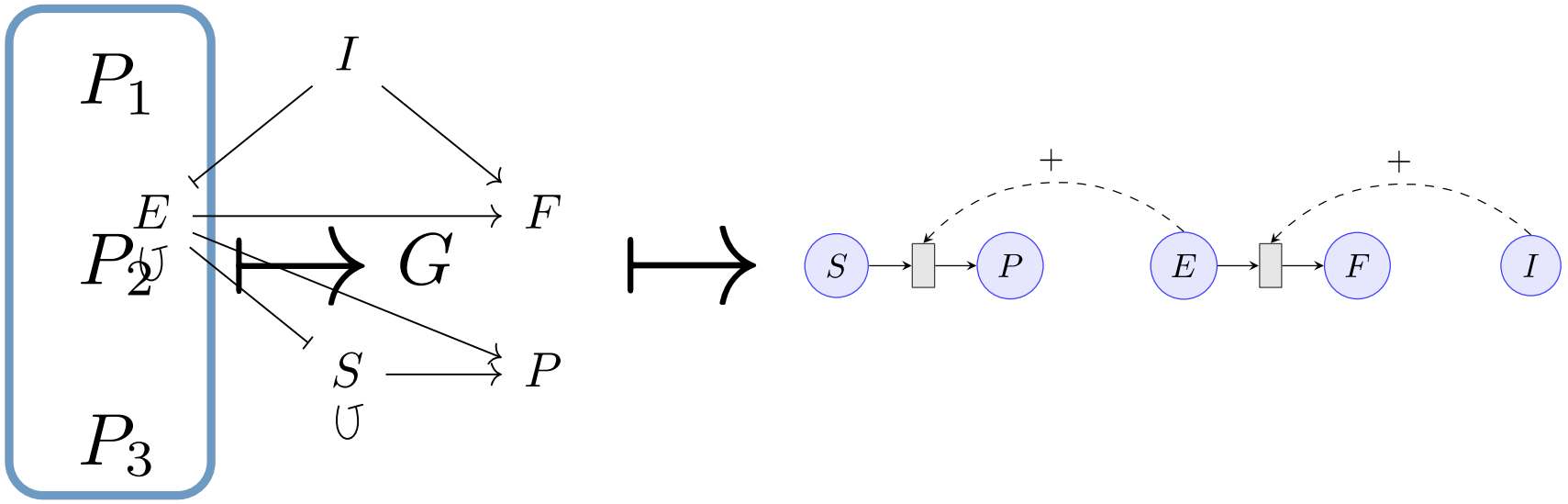
$$A \longmapsto B \qquad A \longleftarrow B$$

- These 4 cases generate the functor
- Int can be expressed with Functorial Data Migration
- Asking for an explanation of a Regnet G is asking for a P such that $Int(P) = G$

Biology is about Explaining Observed Phenomena

- Biological explanation is an inverse problem
- Which Petri Net best explains the regulatory interactions observed?
- Discovering the mechanism is finding a lift for this functor

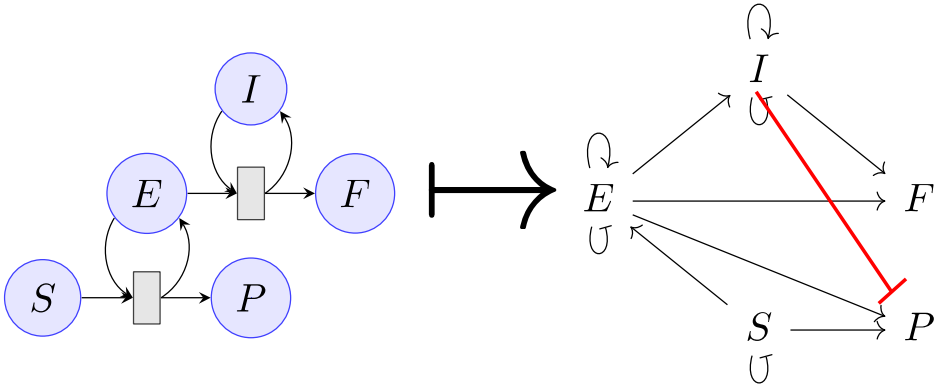
$$SgnPetri \xrightarrow{Int} SgnGraph \xrightarrow{Int^{-1}} SgnPetri$$



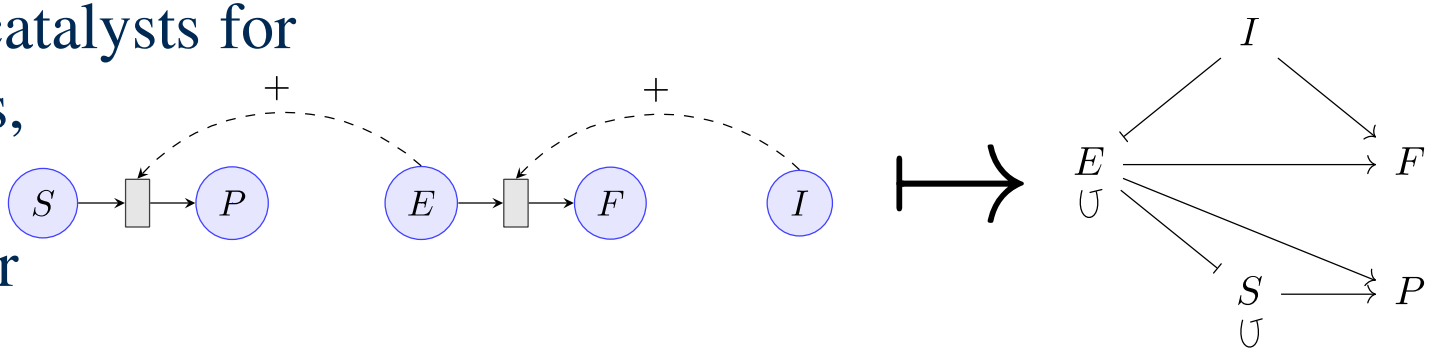
Why do we need links?

If we model enzyme catalyzed reactions with standard Petri Nets, our interactions functor gives the wrong answer

I is downregulating P

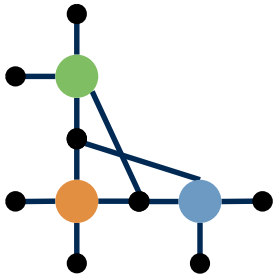


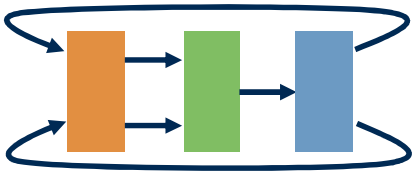
But by remembering which molecules are catalysts for which reactions, we get the right answer

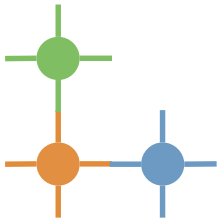


Mathematical Programming Languages Compile to ODES

Languages for Hierarchical System Descriptions







Mathematical Interpreters

Variable Sharing (Biology)

Signal Flow (Electronics / Controls)

FEM/Stencils (Physics)

Common Dynamical Systems Semantics
Support simulation with existing solvers

$$\begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \end{bmatrix} = f\left(\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \right)$$

Operadic Modeling of Dynamical Systems: Mathematics and Computation, S. Libkind, A. Baas, E. Patterson, J. Fairbanks, *Applied Category Theory* Proceedings 2021

New Categories for Semantics

- In traditional CS and Math, one embeds their construction into a small number of well understood categories ex. Dynamical Systems, Smooth Functions, Matrix Representations
- In ACT, we invent/discover new categories fitting our applications
- Parameterized dynamical systems can give semantics to these models

- We need to upgrade the Dynam functor from Baez & Pollard from Set to Vect
- Para(Dynam) is the coslice of Free Vector Space functor F over Dynam

Parameterized Dynamical Systems

Para(Dynam) := F/Dynam

- Objects: State Variables S, Parameter Variables P, Parameterized Vector Fields

$$v : \mathbb{R}^P \to \mathrm{Dynam}(S)$$

- Morphisms: Functions q, f satisfying the following equation:

$$q : P \to P'$$
$$f : S \to S'$$

$$\begin{array}{ccc} \mathbb{R}^P & \xrightarrow{v} & \mathrm{Dynam}(S) \\ {\scriptstyle q_*}\downarrow & & \downarrow{\scriptstyle f_* \circ (-) \circ f^*} \\ \mathbb{R}^{P'} & \xrightarrow[v']{} & \mathrm{Dynam}(S') \end{array}$$

$$f_*(v(f^*(x'); \theta)) = v'(x'; q_*(\theta))$$

Note: Para(Dynam) has finite colimits [Rydeheard1988]

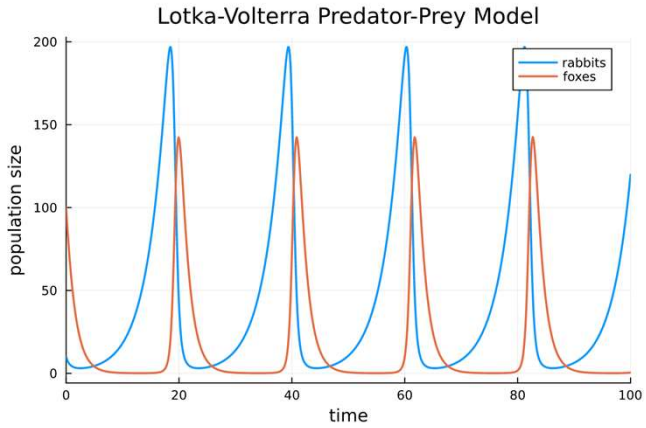
Lotka Volterra Dynamics

Classic Predator-Prey Model

$$\dot{x} = ax - bxy$$
$$\dot{y} = dxy - cy$$

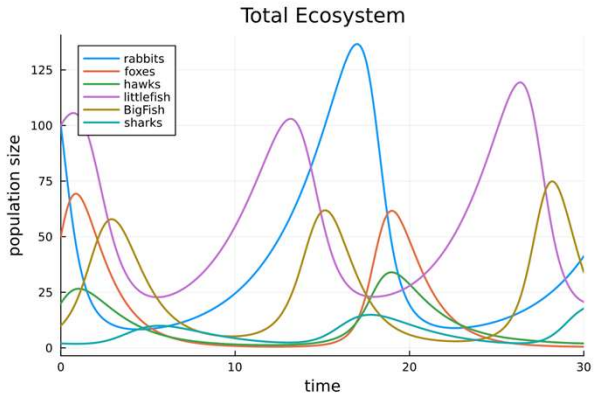
$$\rho = \begin{bmatrix} a \\ -c \end{bmatrix}$$

$$\beta = \begin{bmatrix} 0 & -b \\ d & 0 \end{bmatrix}$$



General Form

$$v(x;\rho,\beta) := x \odot (\rho + \beta x) = \mathrm{diag}(x)(\rho + \beta x)$$

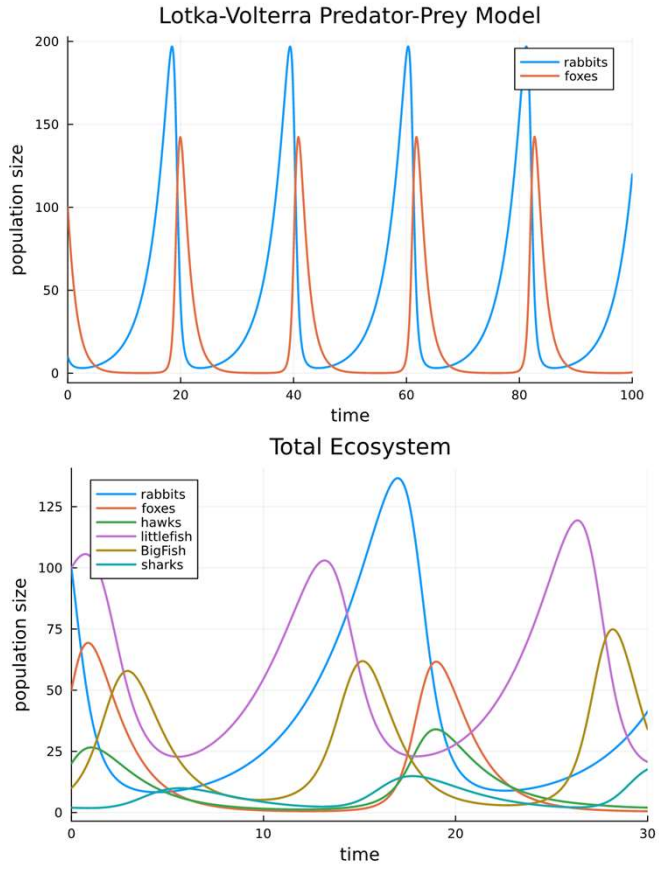


Biological Processes are Nonnegative

- You can't have less than zero molecules, plants, or animals
- Conic spaces are like Vector Spaces -- all our constructions go through
- $\mathrm{Para}(\mathrm{Dynam})_+ := \mathrm{F}_+/\mathrm{Dynam}_+$

  F_+ : Free Conic Space

  Dynam_+ : Essentially Nonnegative Algebraic Vector Fields



Note: $\mathrm{Para}(\mathrm{Dynam})_+$ has finite colimits [Rydeheard1988]

Lotka Volterra Dynamics

We can construct a LV dynamical system from a graph

$$\mathrm{LV} : \mathsf{FinGraph} \to \mathsf{Para}(\mathrm{Dynam})$$

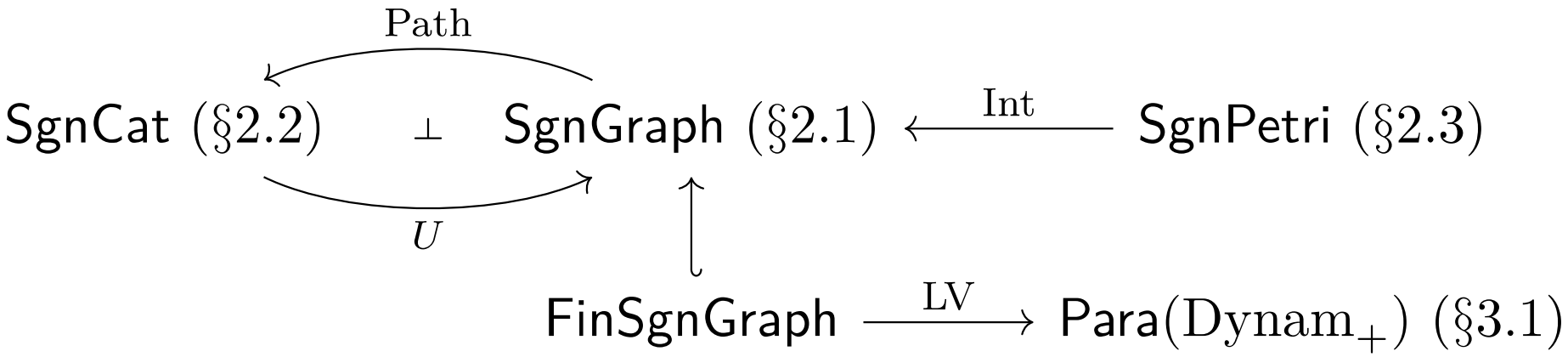
$$v(x;\rho,\beta)(i) \coloneqq \rho(i)\,x(i) + \sum_{(e:i'\to i)\in X} \beta(e)\,x(i')\,x(i), \qquad x \in \mathbb{R}^{X(V)},\ i \in X(V)$$

If the edges have signs, we can construct a LV dynamical system that exhibits the right behavior

$$\mathrm{LV} : \mathsf{FinSgnGraph} \to \mathsf{Para}(\mathrm{Dynam}_+)$$

$$v(x;\rho,\beta)(i) \coloneqq -\rho(i)\,x(i) + \sum_{(e:i'\to i)\in X} X(\mathrm{sgn})(e)\,\beta(e)\,x(i')\,x(i), \qquad x \in \mathbb{R}^{X(V)},\ i \in X(V)$$

Overview of Categories we Used

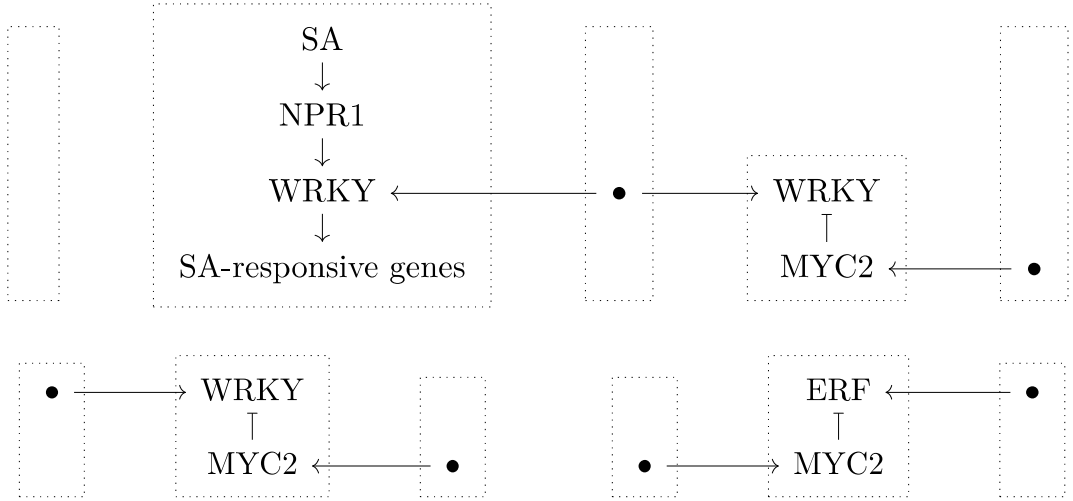


- Path, U: Relates the data of a Regulatory Network to its Behavior
- Int: Given a biochemical mechanism, what Regulatory Interactions does it imply? Preimages of this functor are "explanatory mechanisms"
- LV: Assigns a parameterized dynamical system to each Regnet consistent with the local information

Open Systems with Decorated Cospans

Using Structured/Decorated Cospans we lift everything to Open Systems

$$\begin{array}{ccccc} A_0 & \xrightarrow{\ell_0} & S & \xleftarrow{\ell_1} & A_1 \\ {\scriptstyle f_0}\downarrow & & {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f_1} \\ A_0' & \xrightarrow[\ell_0']{} & S' & \xleftarrow[\ell_1']{} & A_1' \end{array}$$



$$\begin{array}{ccc} \mathbb{O}\mathsf{pen}(\mathsf{SgnCat}) & \xleftarrow{\text{Path}} & \mathbb{O}\mathsf{pen}(\mathsf{SgnGraph}) \\ & & \uparrow \\ & & \mathbb{O}\mathsf{pen}(\mathsf{FinSgnGraph}) \xrightarrow{\text{LV}} \mathbb{O}\mathsf{pen}(\mathsf{Para}(\mathrm{Dynam}_+)) \ (\S 3.3) \end{array}$$

Conclusions

- Defining data models as presheaves over a schema is really useful for modeling and allows tools from many areas of math to converge
- Functors into Para(Dynam) as a *Dynamical Representation* of *Systems* can be applied to other kinds of biological data
- Our Abstract Nonsense pays off!
  - Rydeheard wasn't thinking about decorated cospans of open parameterized systems.
- We can push out the Pareto frontier of usability, generality, and performance in scientific modeling using ACT

https://github.com/AlgebraicJulia/RegNets.jl/blob/main/src/RegNets.jl

Thanks for the Support!

The Team

Dr. James Fairbanks

Dr. Kris Brown (emeritus)

Prof. Ben Bumpus

Dr. Wilmer Leal

TOPOS INSTITUTE

Dr. Evan Patterson

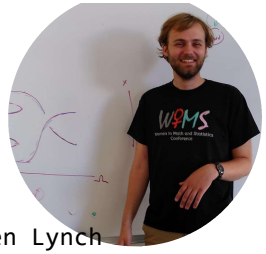
Tyler Hanks

Luke Morris

Matt Cuffaro

Richard Samuelson

Kevin Carlson



Owen Lynch

Publications and Software

1. Operadic Modeling of Dynamical Systems: Mathematics and Computation, S. Libkind, A. Baas, E. Patterson, J. Fairbanks, ACT 2021 Proceedings
2. Categorical Data Structures for Technical Computing, E. Patterson, O. Lynch, J. Fairbanks, *Compositionality*, Accepted Feb 2022
3. An Algebraic Framework for Structured Epidemic Modeling, S. Libkind, A. Baas, M. Halter, E. Patterson, and J. Fairbanks, accepted Mar 2022 with *The Royal Society Phil. Trans. A*
4. Computational category-theoretic rewriting, K. Brown, T. Hanks, E. Patterson, J. Fairbanks, accepted to *International Conference on Graph Transformation*, July 2022
5. A Diagrammatic View of Differential Equations in Physics, E. Patterson, A. Baas, T. Hosgood, J. Fairbanks, accepted May 2022 to *Mathematics in Engineering*
6. Compositional Exploration of Combinatorial Scientific Models. K. Brown, T. Hanks, and J. Fairbanks, Submitted May 2022 *Proceedings of the Conference on Applied Category Theory*
7. Diagrammatic Equations for Multiphysics Modeling and Simulation, A. Baas, K. Brown, J. Arias, M. Gaitlin, E. Patterson, J. Fairbanks in preparation

Package	Description
Catlab.jl	A framework for applied category theory in the Julia language
Semagrams.jl	A framework for developing GUIs for studying presheaf categories. Used to build HMIs for modeling tools.
AlgebraicPetri.jl	Build and analyze petri net based models compositionally. Supports both ODE and Stochastic execution
Individuals.jl	Agent Based Modeling with Rewriting Rules
AlgebraicDynamics.jl	Build and Simulate dynamical systems compositionally
StockFlow.jl (funded by AFOSR)	A Systems Dynamics approach to modeling based on Stock and Flow Diagrams
CombinatorialSpaces.jl	Simplicial sets and other combinatorial models of geometric spaces. A space to build geometric and PDE simulators
Decapodes.jl	A PDE solver based on the Method of Lines for Discrete Exterior Calculus. Supports compositional description of multiphysics.
AlgebraicRelations.jl	Integration with Relational Database Technology. Including building SQL categorically
AlgebraicWorkflows.jl	A system for representing analysis workflows based on monoidal categories. Tracks scientific data processing pipelines with provenenance
Hydrologics.jl (prerelease)	Hydrology modeling with operads of river systems
AlgebraicNeurons.jl (prerelease)	Deep Learning with complex architectures constructed hierarchically

The Team

Dr. James Fairbanks

Dr. Kris Brown

TOPOS INSTITUTE

Tyler Hanks

Luke Morris

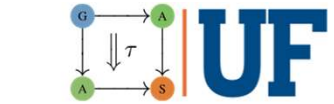
Dr. Tim Hosgood

Georgia Tech

Dr. Evan Patterson

Sophie Libkind

Owen Lynch



Andrew Baas

Maia Gatlin

Jesus Arias

Dr. Clayton Kerce