# Operadic Modeling of Dynamical Systems: Mathematics and Computation

Sophie Libkind, Andrew Baas,
Evan Patterson, and James Fairbanks

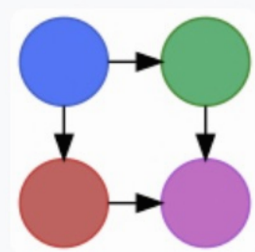
Applied Category Theory Conference, 2021

# Introduction

FINDING
the right abstractions

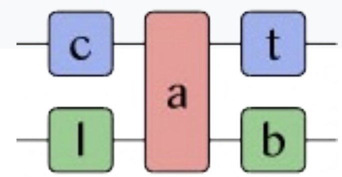
⟷

IMPLEMENTING
the right abstractions



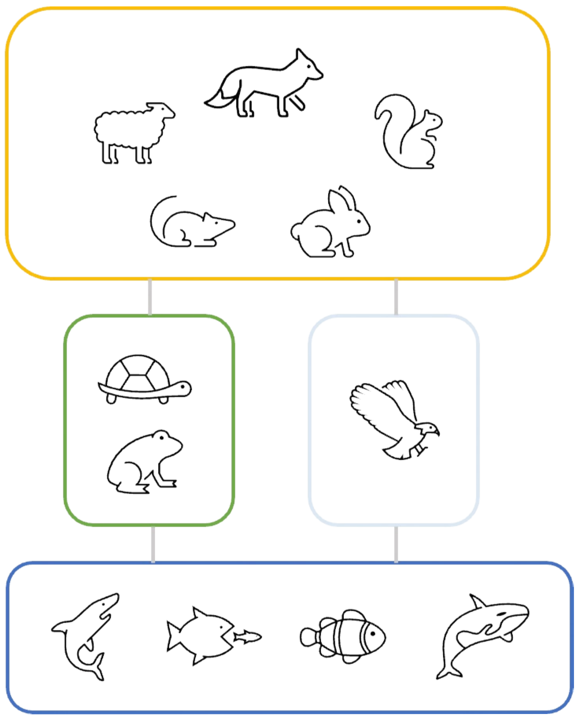
**AlgebraicJulia**

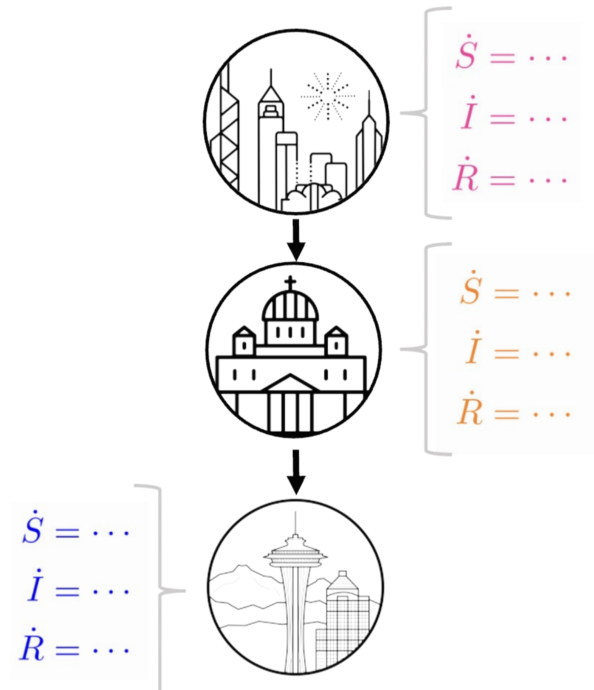
https://www.algebraicjulia.org



# Introduction

## systems have modular structure





# Main idea

| Modeling terminology | mathematical abstractions $F : \mathcal{O} \to \mathsf{Set}$ | Julia implementation AlgebraicDynamics.jl |
|---|---|---|
| system interface | $t \in \mathrm{ob}\, \mathcal{O}$ | |
| diagram of systems | $\phi \in \mathcal{O}(s_1, \ldots, s_n; t)$ <br> $\phi_{\text{inner}} \in \mathcal{O}(r_1, \ldots, r_m; s_i)$ | `diagram::ACSet{TheoryO}` <br> `inner_diagram::ACSet{TheoryO}` |
| hierarchical diagram | $\phi \circ_i \phi_{\text{inner}}$ | `ocompose(diagram, i, inner_diagram)` |
| elementary models | $(m_1, \ldots, m_n) \in F s_1 \times \ldots \times F s_n$ | `models::Vector{T}` |
| composite model | $F(\phi)(m_1, \ldots, m_n) \in F t$ | `oapply(diagram, models)` |

# Outline

I. undirected composition

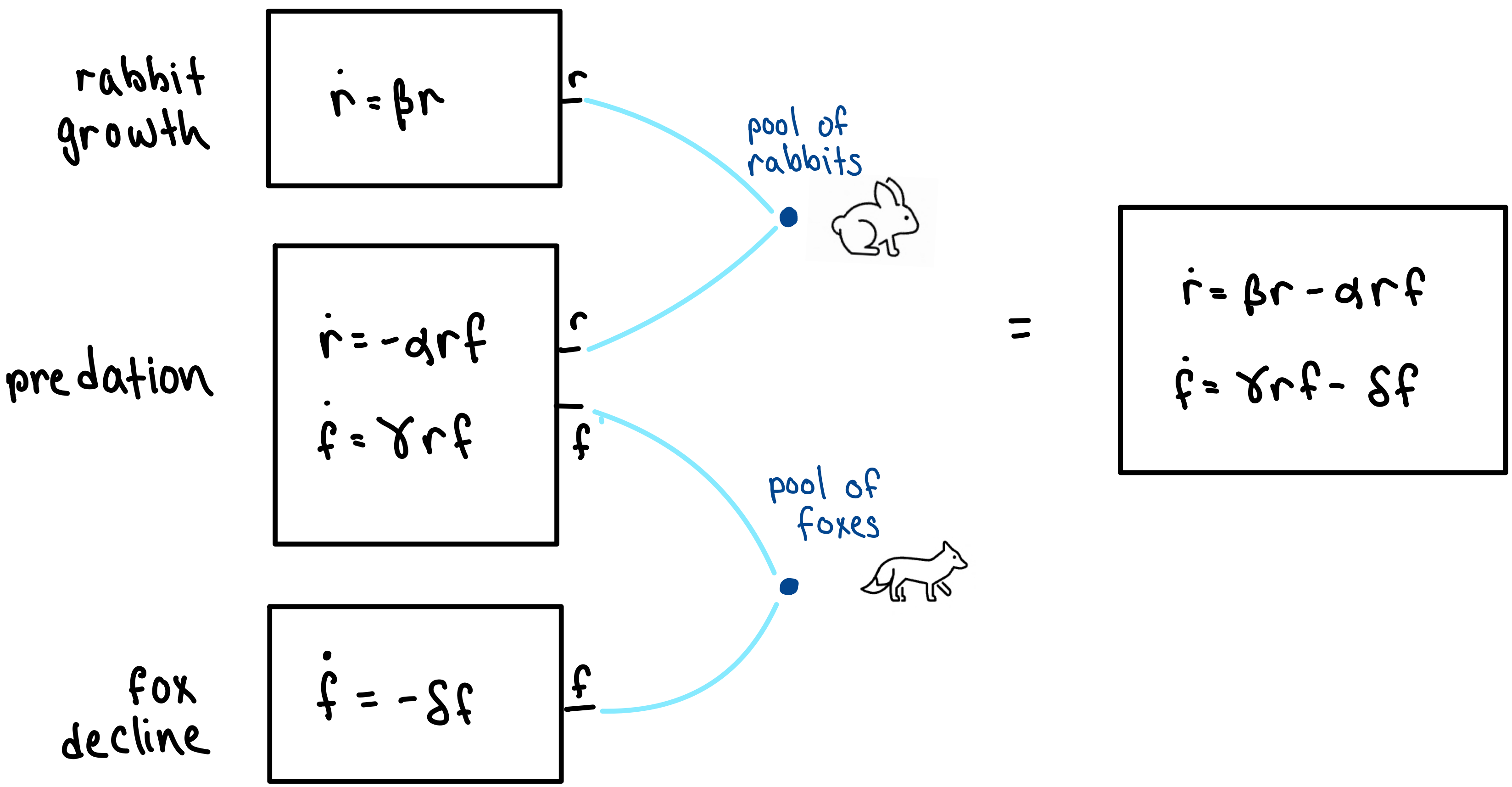
II. Directed composition

III. Conclusion

| Modeling terminology | mathematical abstractions $F : \mathcal{O} \to \mathsf{Set}$ | Julia implementation AlgebraicDynamics.jl |
|---|---|---|
| system interface | $t \in \mathrm{ob}\, \mathcal{O}$ | |
| diagram of systems | $\phi \in \mathcal{O}(s_1, \ldots, s_n; t)$<br>$\phi_{\text{inner}} \in \mathcal{O}(r_1, \ldots, r_m; s_i)$ | `diagram::ACSet{TheoryO}`<br>`inner_diagram::ACSet{TheoryO}` |
| hierarchical diagram | $\phi \circ_i \phi_{\text{inner}}$ | `ocompose(diagram, i, inner_diagram)` |
| elementary models | $(m_1, \ldots, m_n) \in Fs_1 \times \ldots \times Fs_n$ | `models::Vector{T}` |
| composite model | $F(\phi)(m_1, \ldots, m_n) \in Ft$ | `oapply(diagram, models)` |

# I. Undirected Composition - Motivation

Baez and Pollard (2017)

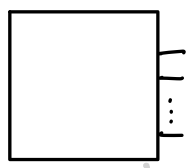


# I. Undirected composition - abstractions for syntax

composition syntax (operad) $UWD := \mathcal{O}(\mathrm{Cospan}_{\mathrm{FinSet}})$

composition semantics

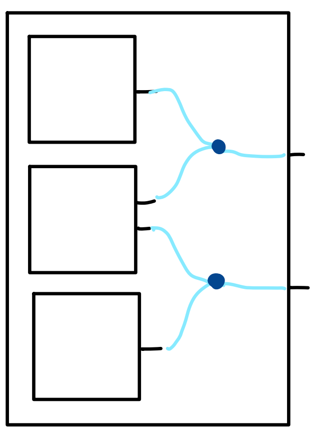
## system interface (types)



$M \in \mathrm{ob}(UWD)$

## elementary models

## diagram of systems (terms)



$1+2+1 \to 2 \leftarrow 2 \in UWD(1,2,1;2)$

## composite model

# I. Undirected composition - abstractions for syntax

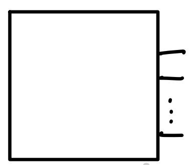
composition syntax (operad)

$\mathsf{UWD} := \mathcal{O}(\mathsf{Cospan}_{\mathsf{Finset}})$

composition semantics (operad algebra)

$\mathrm{Dynam}_c^{-\circ} : \mathsf{UWD} \longrightarrow \mathsf{Set}$

## system interface (types)

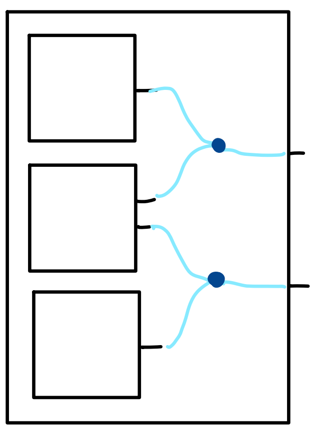


$M \in \mathrm{ob}(\mathsf{UWD})$

## elementary models

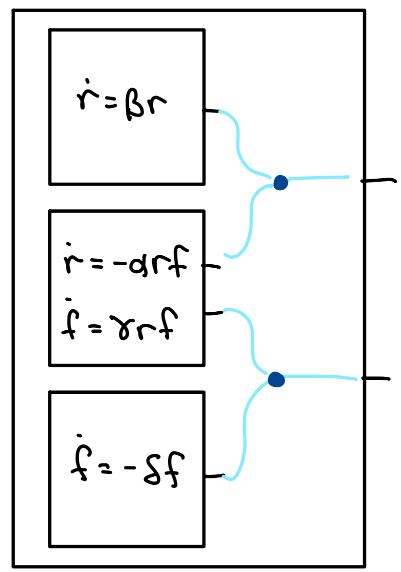
$$\mathrm{Dynam}_c^{-\circ}(M) = \left\{ \boxed{\begin{array}{c} S \in \mathsf{Finset} \\ v : \mathbb{R}^S \to T\mathbb{R}^S \\ p : M \to S \end{array}} \right\}$$

## diagram of systems (terms)



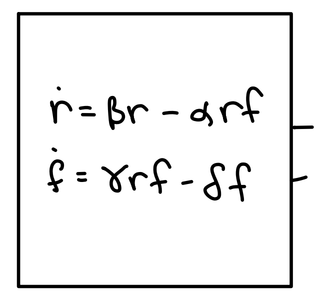
$1+2+1 \to 2 \leftarrow 2 \in \mathsf{UWD}(1,2,1;2)$

## composite model




"add along shared coordinates"

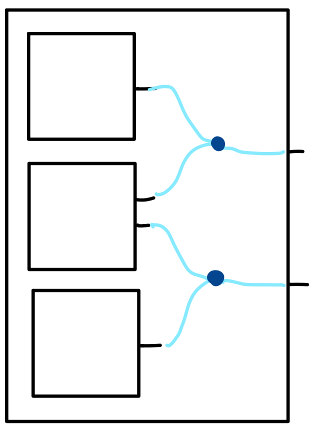
$=$



## I. Undirected Composition - Implementation of Syntax

**def** A $\mathcal{C}$-Set is a copresheaf $X : \mathcal{C} \to \mathbf{Set}$

mathematical abstraction



data structure

**def** define a schema for undirected wiring diagrams by

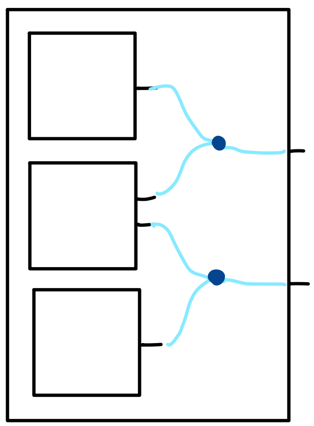
$$\mathrm{Th}(UWD) := \quad B \leftarrow P \rightarrow J \leftarrow Q$$

**prop** undirected wiring diagrams $\xleftrightarrow{1\text{-}1}$ finite instances of Th(UWD)

ie. terms of
$UWD := \mathcal{O}(\mathrm{Cospan}_{\mathrm{FinSet}})$

# I. Undirected composition - implementation of syntax

Mathematical Abstraction



$1+2+1 \to 2 \leftarrow 2$

$\in UWD(1,2,1;2)$

Julia Code

```julia
rabbitfox_diagram = @relation (rabbits, foxes) begin
    growth(rabbits)
    predation(rabbits, foxes)
    decline(foxes)
end
```

Julia Output

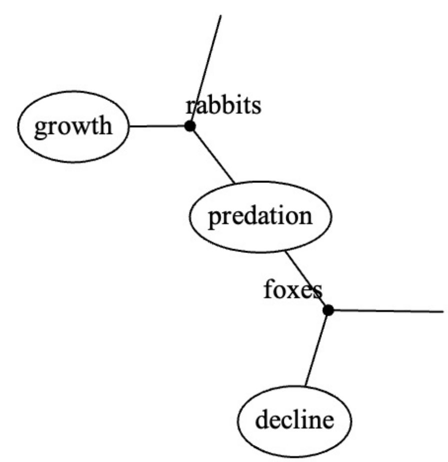
ACSet with elements Box = 1:3, Port = 1:4, OuterPort = 1:2, Junction = 1:2

| Box | name |
|---|---|
| 1 | growth |
| 2 | predation |
| 3 | decline |

| Port | box | junction |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 2 | 1 |
| 3 | 2 | 2 |
| 4 | 3 | 2 |

| OuterPort | outer_junction |
|---|---|
| 1 | 1 |
| 2 | 2 |

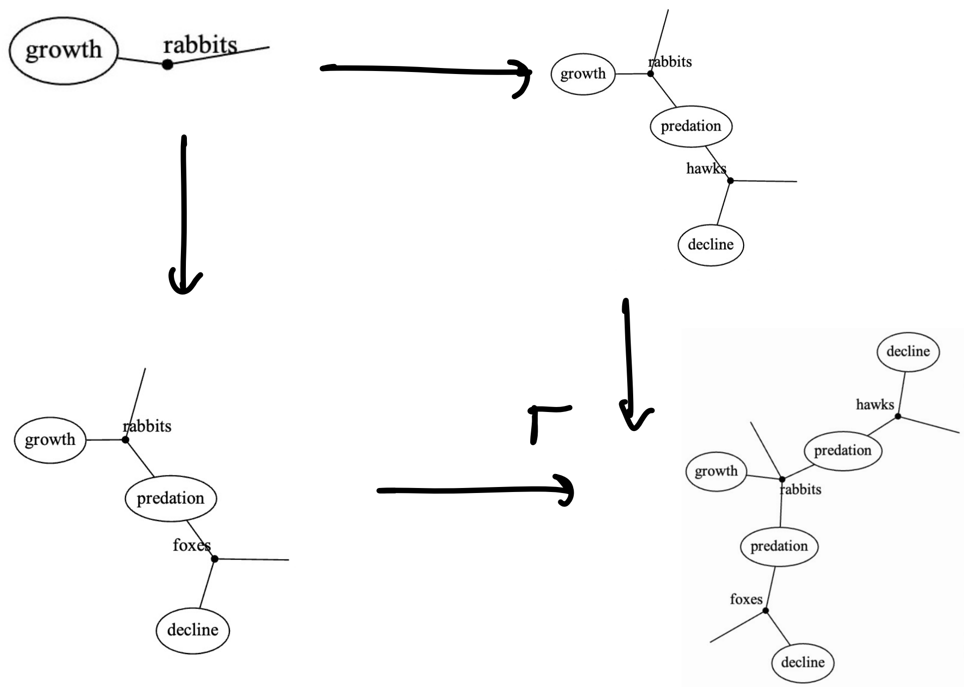
| Junction | variable |
|---|---|
| 1 | rabbits |
| 2 | foxes |



# I. Undirected composition - implementation of syntax

Advantages of $\mathcal{C}$-Set implementation:

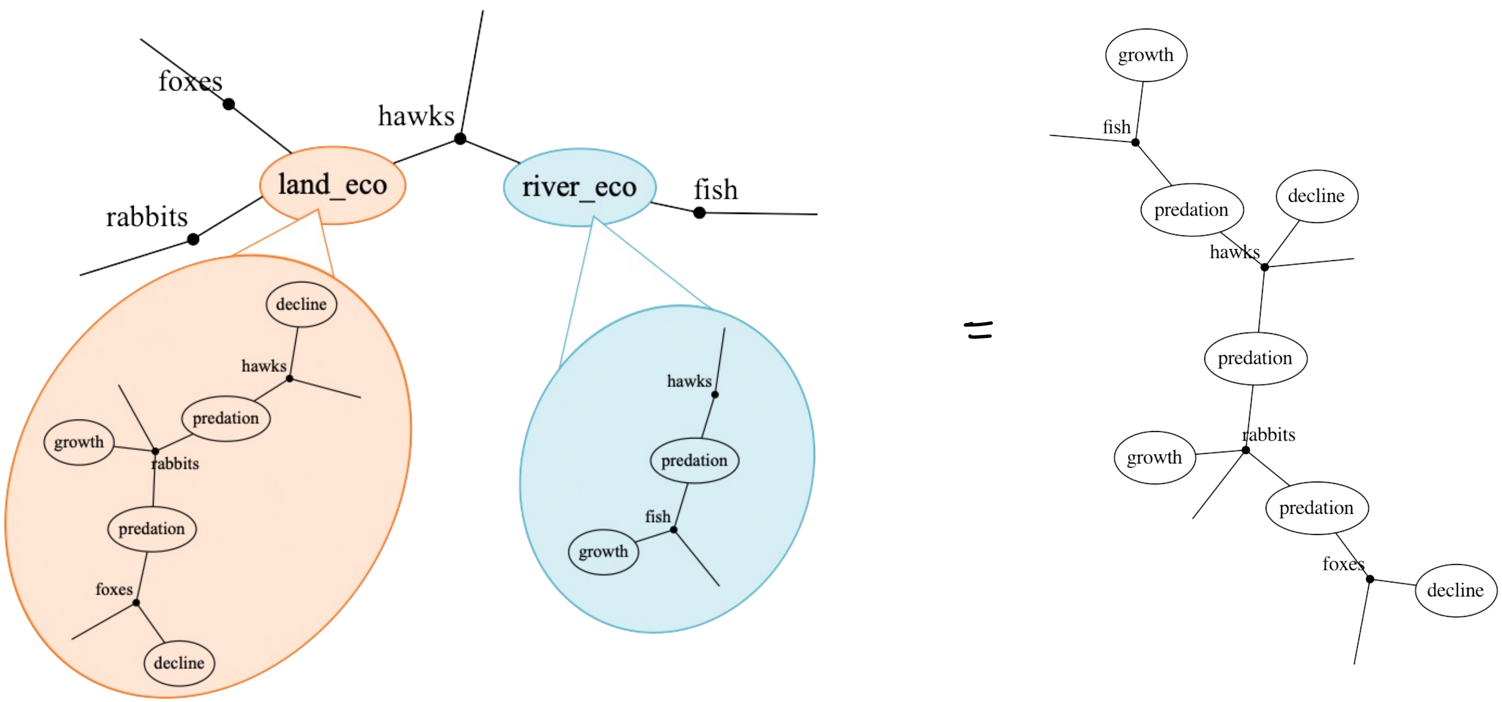
1. Domain specific data structure
2. apply features of $[\mathcal{C}, \mathrm{Set}]$
   - limits / colimits
   - functorial data migration




```julia
# Define elementary diagrams of systems
rabbitfox_diagram = @relation (rabbits, foxes) begin
    growth(rabbits)
    predation(rabbits, foxes)
    decline(foxes)
end

rabbithawk_diagram = @relation (rabbits, hawks) begin
    growth(rabbits)
    predation(rabbits, hawks)
    decline(hawks)
end

rabbit_diagram = @relation (rabbits,) -> growth(rabbits)

# Define transformations between the diagrams
rabbitfox_transform  = ACSetTransformation(
    (Box=[1], Junction=[1], Port=[1], OuterPort=[1]),
    rabbit_diagram, rabbitfox_diagram
)
rabbithawk_transform = ACSetTransformation(
    (Box=[1], Junction=[1], Port=[1], OuterPort=[1]),
  rabbit_diagram, rabbithawk_diagram
)

# Take the pushout
land_diagram = ob(pushout(rabbitfox_transform, rabbithawk_transform))
```

## I. Undirected composition - implementation of syntax

### Hierarchical Diagrams (substitution in UWD)




Julia Code

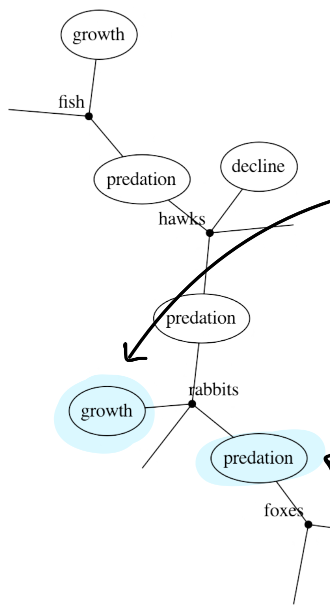
```
ocompose(total_diagram, [land_diagram, river_diagram])
```

# I. Undirected composition - Implementation of semantics

Recall,

**mathematical abstraction**

**Julia code**




$$\dot{r} = \beta r$$

$r$

```julia
rabbit_growth = ContinuousResourceSharer{Float64}(
    1,  # exposed ports
    1,  # states
    (u,p,t) -> p.β * u, # dynamics
    [1]                 # port map
)
```

$$\dot{r} = -\alpha r f$$
$$\dot{f} = \gamma r f$$

$r$ $f$

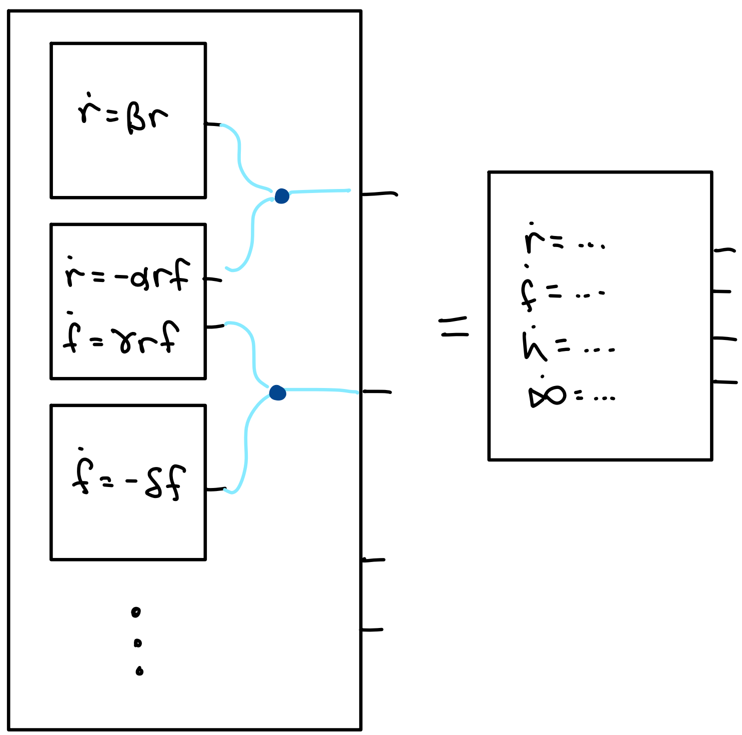
```julia
rabbit_fox_predation = ContinuousResourceSharer{Float64}(
    2,  # exposed ports
    2,  # states
    (u,p,t) -> [p.α*u[1]*u[2], p.γ*u[1]*u[2]], # dynamics
    [1,2]                                     # port map
)
```

diagram of systems

elementary models

# I. Undirected composition - Implementation of semantics

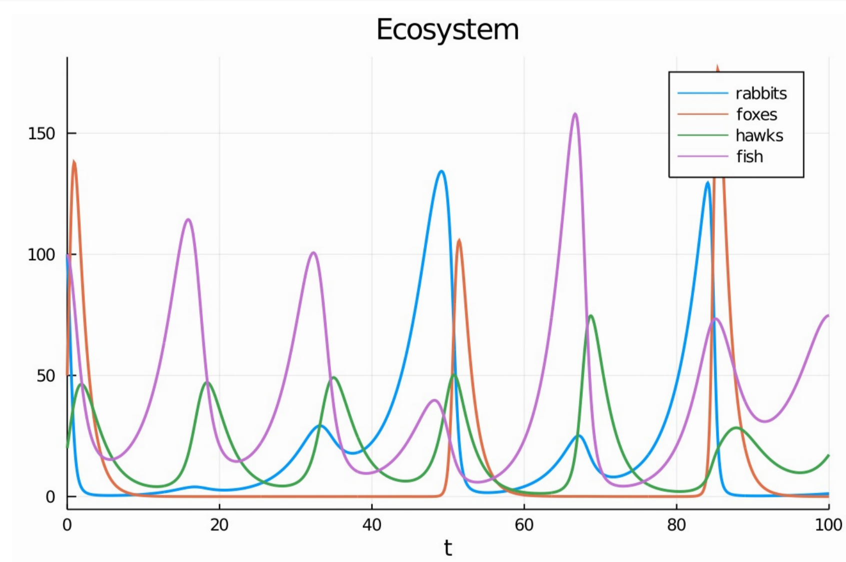
mathematical abstraction




composite model

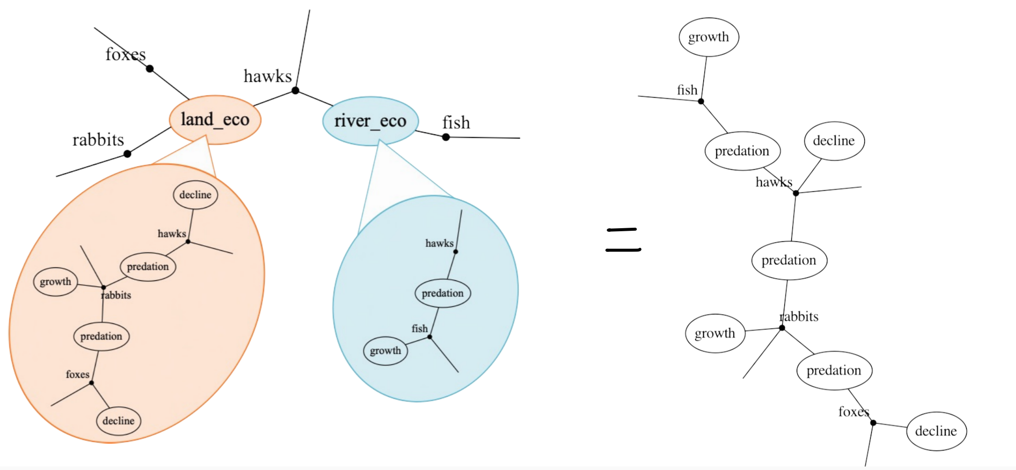
Julia code + output

```
# compose models
eco_sys = oapply(eco_diagram, vcat(land_models, river_models))

# solve and plot
tspan = (0.0, 100.0)
u0 = [100.0, 50.0, 20.0, 100.0]

prob = ODEProblem(eco_sys, u0, tspan, params)
sol = solve(prob, Tsit5())
```



# I. Undirected Composition - Highlights of AlgebraicDynamics

## #1: Compositional and Hierarchical Model specification and analysis



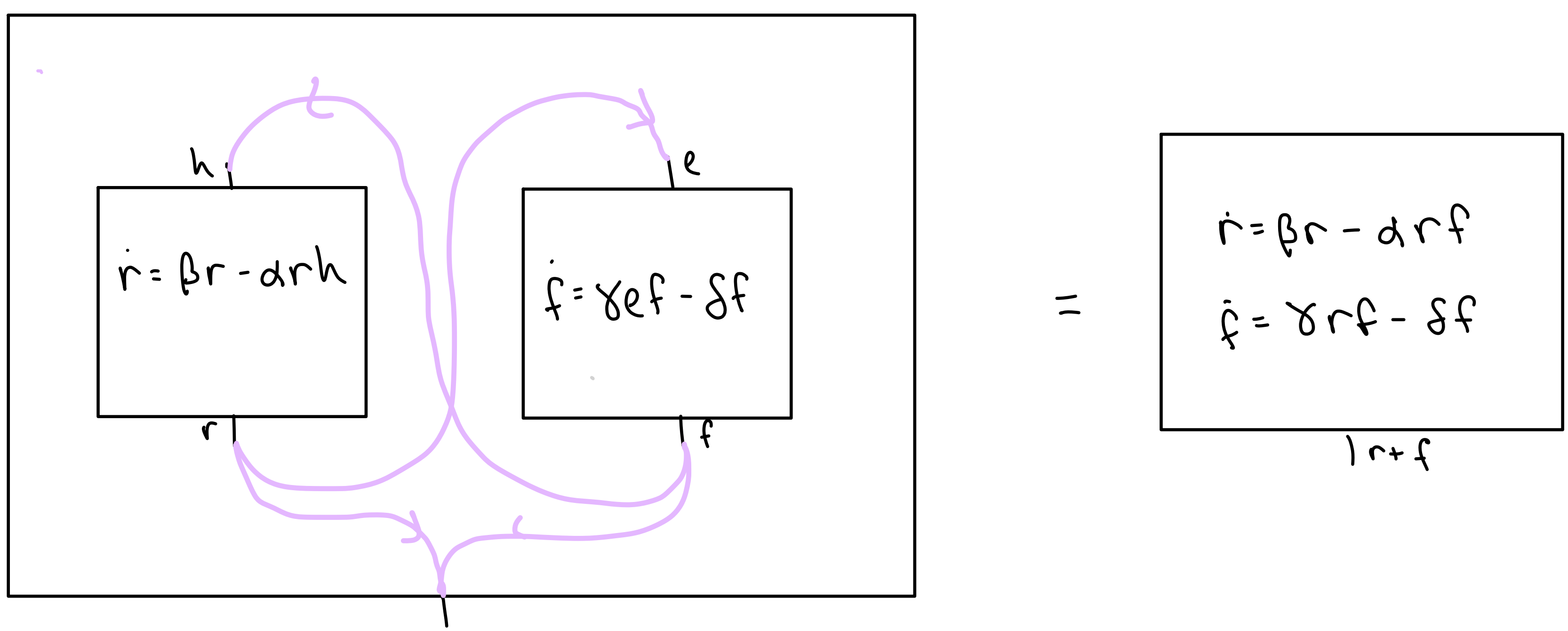

```
# flattened model specification
eco_diagram = ocompose(total_diagram, [land_diagram, river_diagram])
eco_sys     = oapply(eco_diagram, vcat(land_models, river_models))

# hierarchical model specification
land_sys  = oapply(land_diagram, land_models)
river_sys = oapply(river_diagram, river_models)

eco_sys   = oapply(total_diagram, [land_sys, river_sys])
```

# Main idea

| Modeling terminology | mathematical abstractions $F : \mathcal{O} \to \mathsf{Set}$ | Julia implementation AlgebraicDynamics.jl |
|---|---|---|
| system interface | $t \in \mathrm{ob}\, \mathcal{O}$ | |
| diagram of systems | $\phi \in \mathcal{O}(s_1, \ldots, s_n; t)$ <br> $\phi_{\text{inner}} \in \mathcal{O}(r_1, \ldots, r_m; s_i)$ | `diagram::ACSet{TheoryO}` <br> `inner_diagram::ACSet{TheoryO}` |
| hierarchical diagram | $\phi \circ_i \phi_{\text{inner}}$ | `ocompose(diagram, i, inner_diagram)` |
| elementary models | $(m_1, \ldots, m_n) \in Fs_1 \times \ldots \times Fs_n$ | `models::Vector{T}` |
| composite model | $F(\phi)(m_1, \ldots, m_n) \in Ft$ | `oapply(diagram, models)` |

# II. Directed composition - Motivation

Vagner, Spivak, Lerman (2015)

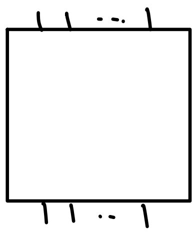


# II. Directed composition – abstractions for syntax and semantics

composition syntax (operad) $\mathrm{DWD} := \mathcal{O}(\mathrm{Lens}_{\mathrm{cospan}(\mathrm{FinSet}^{op})})$

composition semantics (operad algebra) $\mathrm{Dynam}_c^{\rightarrow} : \mathrm{DWD} \longrightarrow \mathrm{Set}$

## system interface (types)

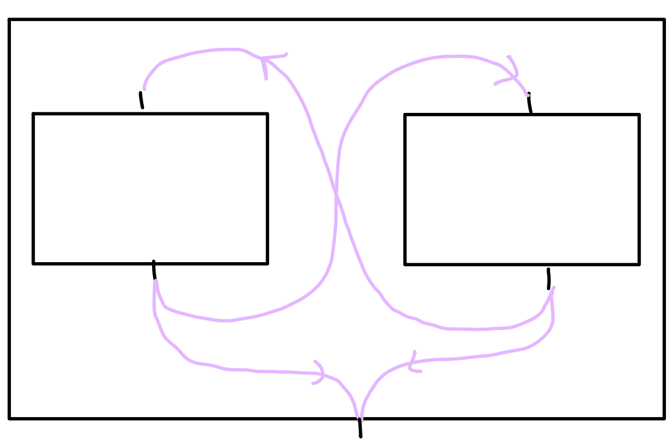


$\begin{pmatrix} X_{in} \\ X_{out} \end{pmatrix} \in \mathrm{ob}\,\mathrm{DWD}$

## elementary models

$$\mathrm{Dynam}_c^{\rightarrow}\begin{pmatrix} X_{in} \\ X_{out} \end{pmatrix} = \left\{ \boxed{\begin{array}{l} S \in \mathrm{FinSet} \\ v : \mathbb{R}^S \times \mathbb{R}^{X_{in}} \longrightarrow T\mathbb{R}^S \\ r : \mathbb{R}^S \longrightarrow \mathbb{R}^{X_{out}} \end{array}} \right\}$$

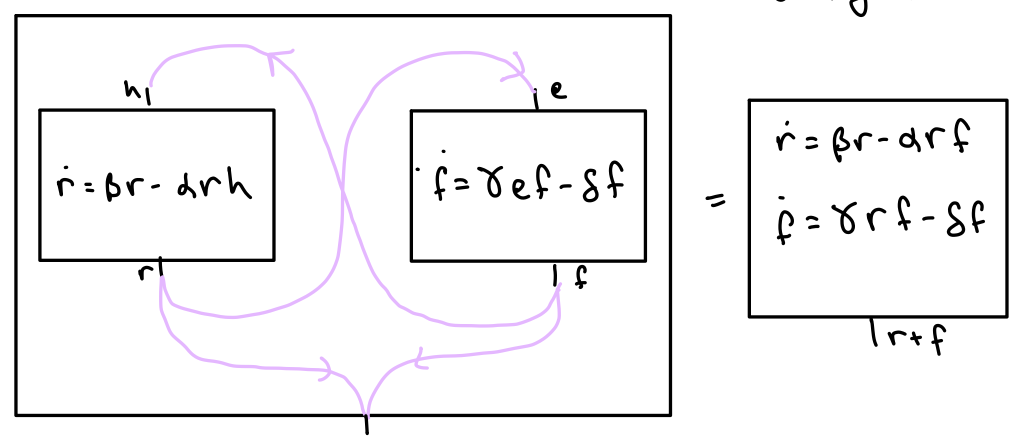
## diagram of systems (terms)



$\in \mathrm{DWD}\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}; \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right)$

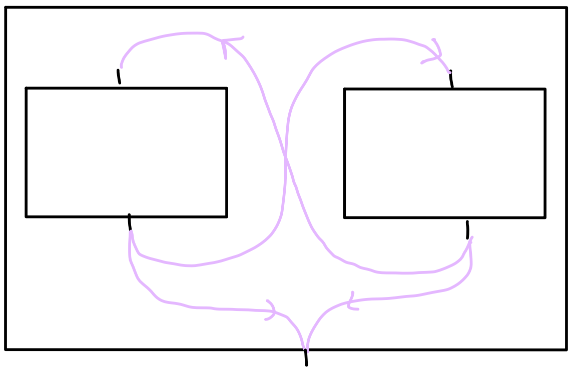
## composite model

"send information along wires"



## II. Directed composition - Implementation of syntax

**mathematical abstraction**



**data structure**

**def** define a schema for directed wiring diagrams by

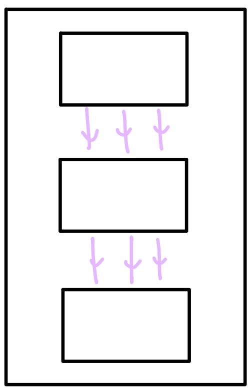
$$
Th(DWD) := \begin{tikzcd}
 & Q_{in} & \\
W_{in} \arrow[ur] \arrow[dr] & & \\
 & P_{in} \arrow[dr] & \\
W \arrow[ur] \arrow[dr] & & B \\
 & P_{out} \arrow[ur] & \\
W_{out} \arrow[ur] \arrow[dr] & & \\
 & Q_{out} &
\end{tikzcd}
$$

**prop** directed wiring diagrams $\xleftrightarrow{1\text{-}1}$ finite instances of Th(DWD)

ie. terms of
$DWD := Lens(Cospan_{FinSet^{op}})$

# II. Directed composition – Implementation of syntax

## Mathematical Abstraction

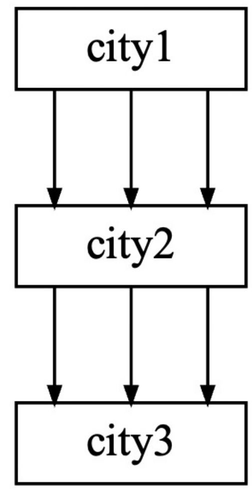


$\in \mathrm{DWD}\left(\binom{3}{0}, \binom{3}{3}, \binom{0}{3} ; \binom{0}{0}\right)$

## Julia Code

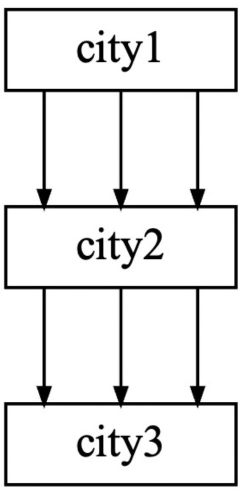
```julia
function get_city_diagram(ncities, roads)
    nout_roads = map(i -> count(r -> r.first == i, roads), 1:ncities)

    city_diagram = WiringDiagram([], [])
    cities = map(1:ncities) do i
        add_box!(city_diagram,
            Box(Symbol("city", i), [:S, :I, :R], [:S, :I, :R]))
    end

    wires = map(Base.Iterators.product(roads, 1:3)) do ((src, tgt), j)
        add_wire!(city_diagram, (cities[src], j) => (cities[tgt], j))
    end
    return city_diagram
end


ncities = 3
roads = [1 => 2, 2 => 3]
city_diagram = get_city_diagram(ncities, roads)
```

## Julia Output

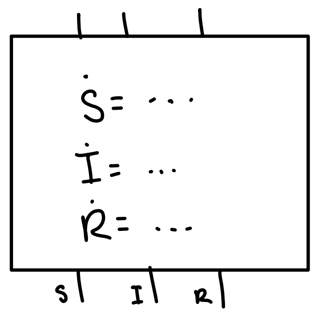


# II. Directed composition - Implementation of Semantics

Recall,




mathematical abstraction




Julia code

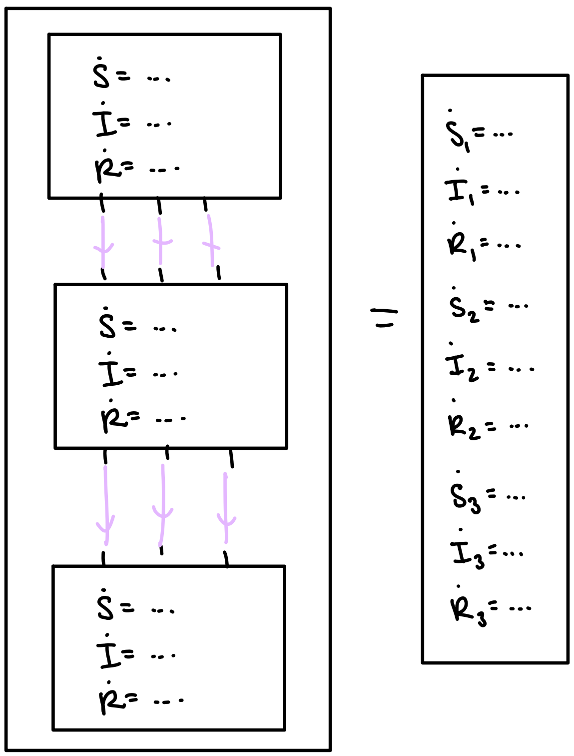
```
city_models = map(1:ncities) do i

  ContinuousMachine{Float64}(
    3,  # inputs
    3,  # states
    3,  # outputs
    (u,x,p,t) -> p.μ*(x - nout_roads[i]*u) + [
            -p[β(i)]*u[1]*u[2],                     # Ṡ
            p[β(i)]*u[1]*u[2] - p[γ(i)]*u[2],       # İ
            p[γ(i)]*u[2]],                          # Ṙ
    u -> u)

end
```

diagram of systems

elementary models

# II. Directed composition - Implementation of semantics

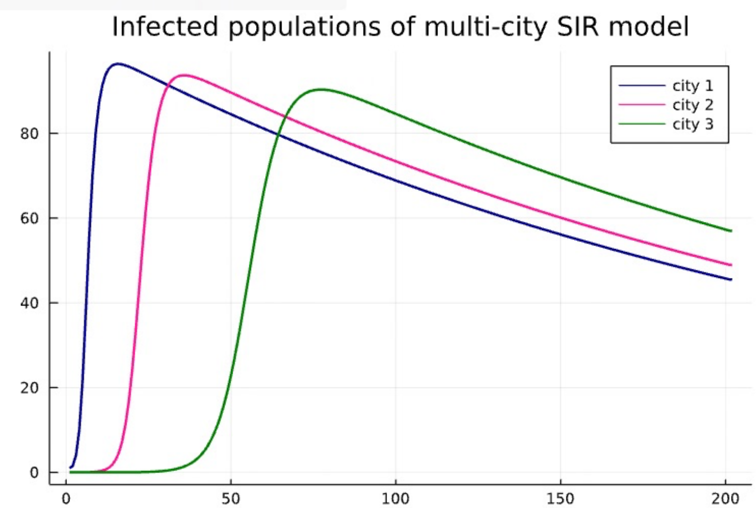
mathematical abstraction




composite model

Julia code + output

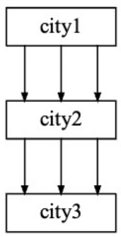
```julia
# compose models
sir_model = oapply(city_pattern, city_models)

# solve and plot
params = LVector(μ = 0.01,
    β1 = 0.7, γ1 = 0.4,
    β2 = 0.4, γ2 = 0.4,
    β3 = 0.2, γ3 = 0.4
)

u0 = [100.0, 1.0, 0.0,0,0,
      100.0, 0.0, 0.0, 0,0,
      100.0, 0.0, 0.0, 0,0]  # initial populations

tspan = (0.0, 2.0)

prob = ODEProblem(sir_model, u0, tspan, params)
sol = solve(prob, Tsit5(); dtmax = 0.01)
```

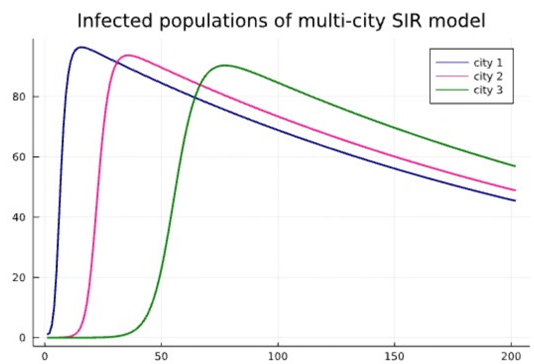


# II. Directed composition - Highlights of AlgebraicDynamics

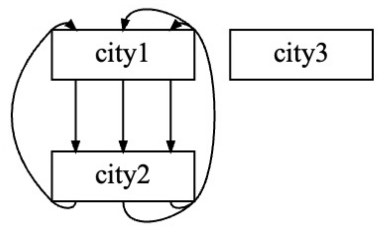
## #2: independence of model syntax and model semantics

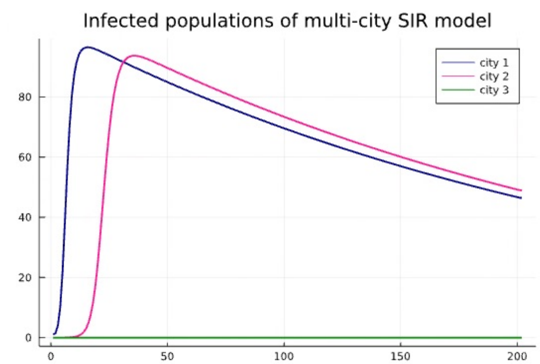
```
ncities = 3
roads = [1 => 2, 2 => 3]
city_pattern1 = get_city_pattern(ncities, roads)

sir_model = oapply(city_pattern1, city_models)
```



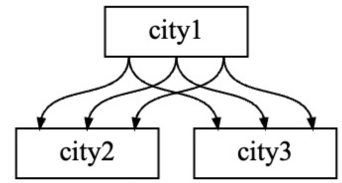



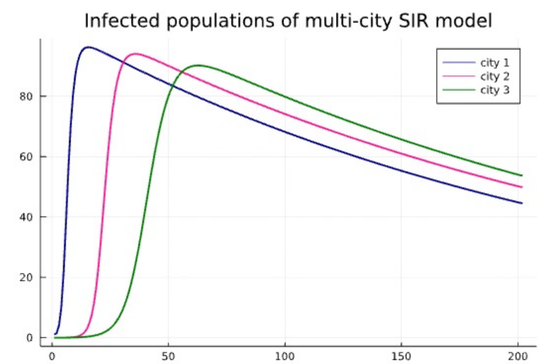

```
ncities = 3
roads = [1 => 2, 2 => 1]
city_pattern2 = get_city_pattern(ncities, roads)

sir_model = oapply(city_pattern2, city_models)
```



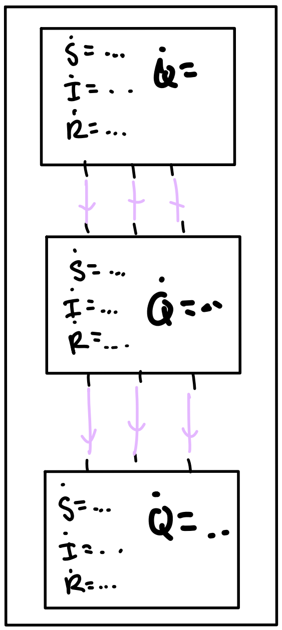

```
ncities = 3
roads = [1 => 2, 1 => 3]
city_pattern3 = get_city_pattern(ncities, roads)

sir_model = oapply(city_pattern3, city_models)
```



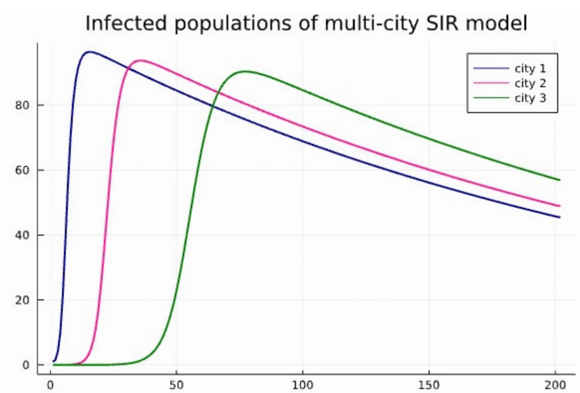
## II. Directed composition – Highlights of AlgebraicDynamics

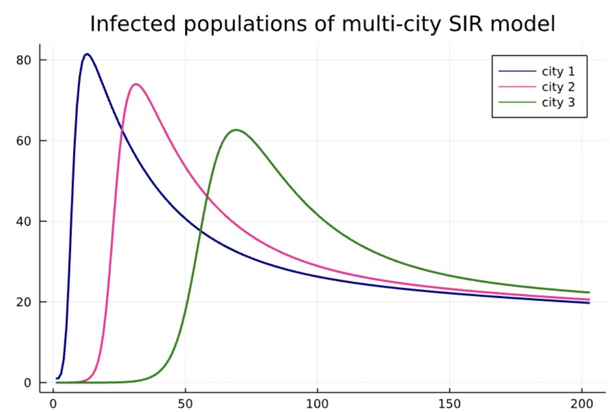
### #2: independence of model syntax and model semantics




```
sir_city_models = map(1:ncities) do i

    ContinuousMachine{Float64}(
        3,  # inputs
        3,  # states
        3,  # outputs
        (u,x,p,t) -> p.μ*(x - nout_roads[i]*u) + [
            -p[β(i)]*u[1]*u[2],                      # Ṡ
            p[β(i)]*u[1]*u[2] - p[γ(i)]*u[2],       # İ
            p[γ(i)]*u[2]],                           # Ṙ
        u -> u
    )

end

sir_model = oapply(city_pattern, sir_city_models)
```

```
sirq_city_models = map(1:ncities) do i

    ContinuousMachine{Float64}(
        3,  # inputs
        5,  # states
        3,  # outputs
        (u,x,p,t) -> begin
            q_rate = 0.025 * u[2]
            return p.μ*([x..., 0,0] - nout_roads[i]*u) + [
                -p[β(i)]*u[1]*u[2] - q_rate*u[1] + (1 - q_rate)*u[4],                  # Ṡ
                p[β(i)]*u[1]*u[2] - p[γ(i)]*u[2] - q_rate*u[2] + (1 - q_rate)*u[5],   # İ
                p[γ(i)]*(u[2] + u[5]),                                                  # Ṙ
                q_rate * u[1] - (1 - q_rate) * u[4],                                    # Q̇s
                q_rate * u[2] - (1 - q_rate) * u[5] - p[γ(i)]*u[5]]                     # Q̇i
        end,
        u -> u[1:3]
    )

end

sirq_model = oapply(city_pattern, sirq_city_models)
```



## III. Conclusion - contributions

undirected composition

$$\begin{tikzcd} \mathsf{UWD} \arrow[r, bend left=50, "\mathsf{Dynam}^{\multimap}_{C}"] \arrow[r, bend right=50, "\mathsf{Dynam}^{\multimap}_{D}"'] \arrow[r, phantom, "\Downarrow \mathsf{Euler}^{\multimap}_{h}"] & \mathsf{Set} \end{tikzcd}$$

directed composition

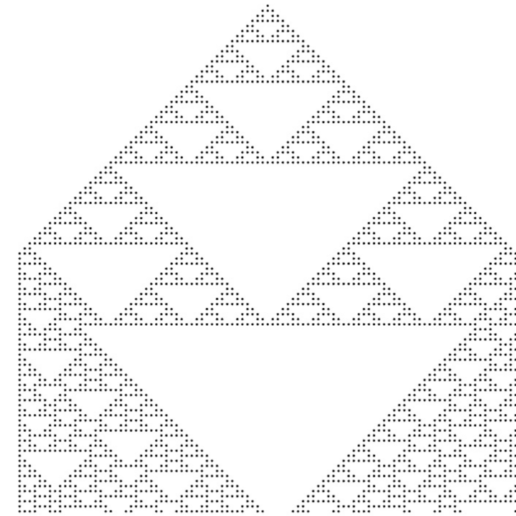
$$\begin{tikzcd} \mathsf{DWD} \arrow[r, bend left=50, "\mathsf{Dynam}^{\rightarrow}_{C}"] \arrow[r, bend right=50, "\mathsf{Dynam}^{\rightarrow}_{D}"'] \arrow[r, phantom, "\Downarrow \mathsf{Euler}^{\rightarrow}_{h}"] & \mathsf{Set} \end{tikzcd}$$

- implementation of operads + operad algebras in Julia
- new algebras for composing dynamical systems
- Functorial euler's method: proofs and implementation

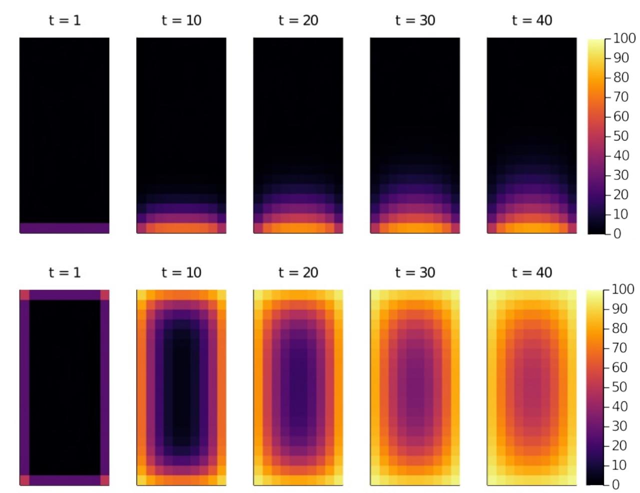
## III. Conclusion - contributions

directed
composition

$$
\begin{tikzcd}
\mathsf{CPG} \arrow[r] & \mathsf{DWD} \arrow[r, bend left, "\overrightarrow{\mathsf{Dynam}}_C"] \arrow[r, bend right, "\overrightarrow{\mathsf{Dynam}}_D"'] \arrow[r, phantom, "\Downarrow \overrightarrow{\mathsf{Euler}}_h"] & \mathsf{Set}
\end{tikzcd}
$$



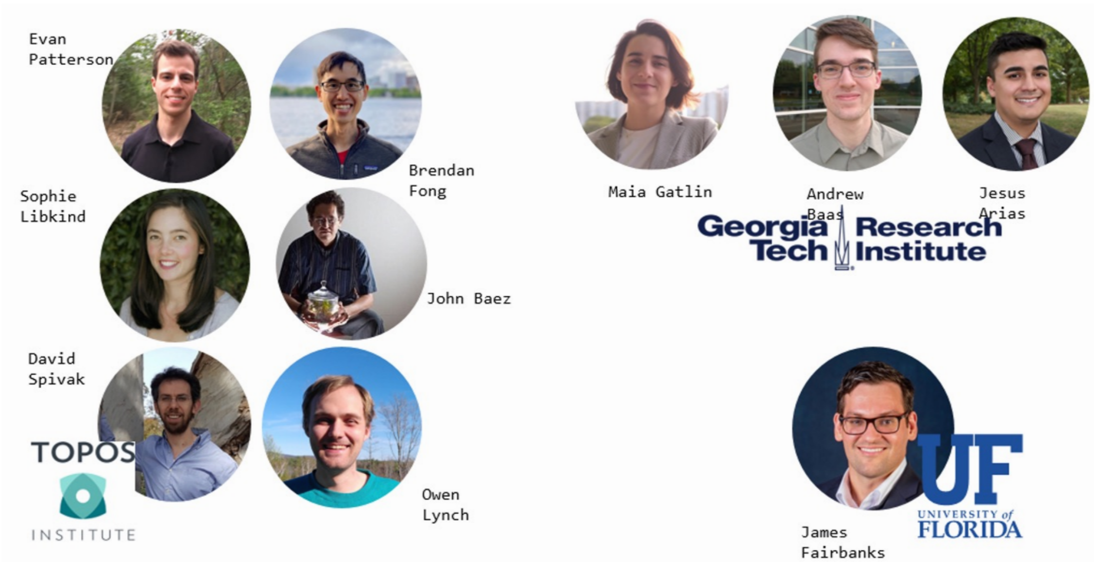
- a new operad of circular port graphs



## III. Conclusion - Future Work

- "abstractions informed by implementation"
- implementing abstractions coming from Higher Category Theory
- General theory of operads defined by e-sets

# THANK YOU!




**AlgebraicJulia**

https://www.algebraicjulia.org


- Baez, John C., and Blake S. Pollard. 2017. "A Compositional Framework for Reaction Networks." *Reviews in Mathematical Physics* 29 (09): 1750028. https://doi.org/10.1142/S0129055X17500283.
- Libkind, Sophie, Andrew Baas, Evan Patterson, and James Fairbanks. 2021. "Operadic Modeling of Dynamical Systems: Mathematics and Computation." *ArXiv:2105.12282 [Math]*, May. http://arxiv.org/abs/2105.12282.
- Patterson, Evan, Owen Lynch, and James Fairbanks. 2021. "Categorical Data Structures for Technical Computing." *ArXiv:2106.04703 [Cs, Math]*, June. http://arxiv.org/abs/2106.04703.
- Vagner, Dmitry, David I. Spivak, and Eugene Lerman. 2015. "Algebras of Open Dynamical Systems on the Operad of Wiring Diagrams." *Theory and Applications of Categories* 30 (51): 1793–1822.