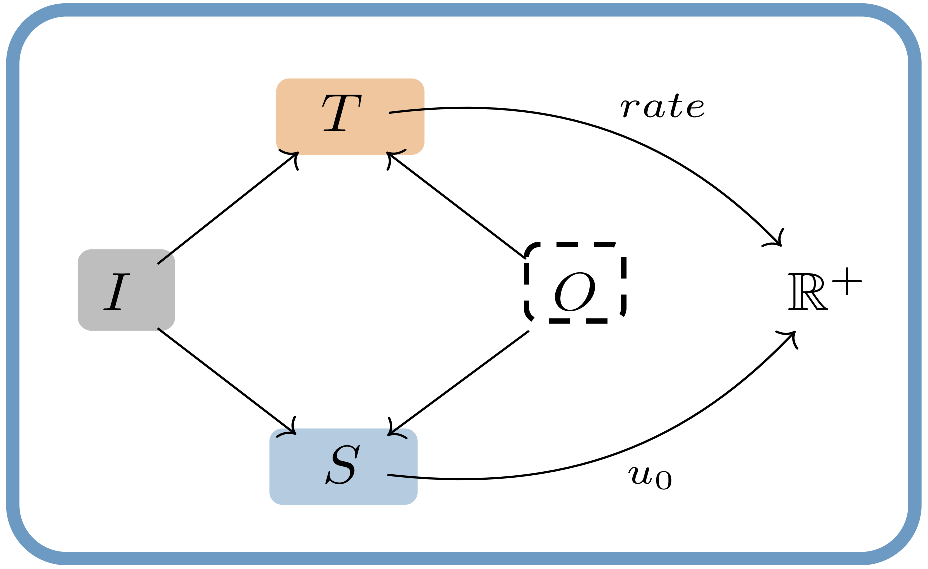
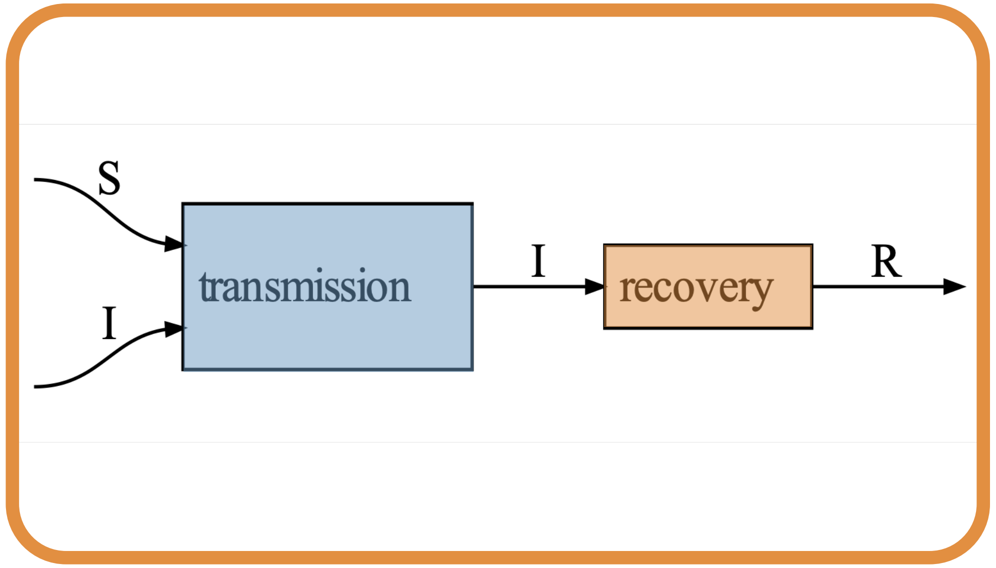
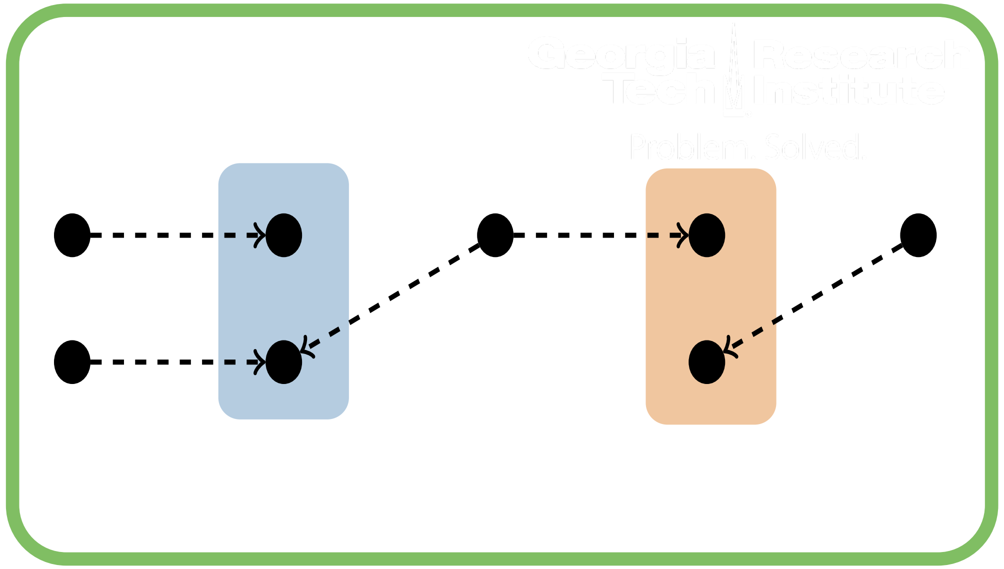






AlgebraicJulia: Applied Category Theory in Julia

Micah Halter, ***James Fairbanks***
Georgia Tech Research Institute (GTRI)

Evan Patterson
MIT

Spectrum of Scientific Computing Technology

Programs that have *explicit* representations of the model

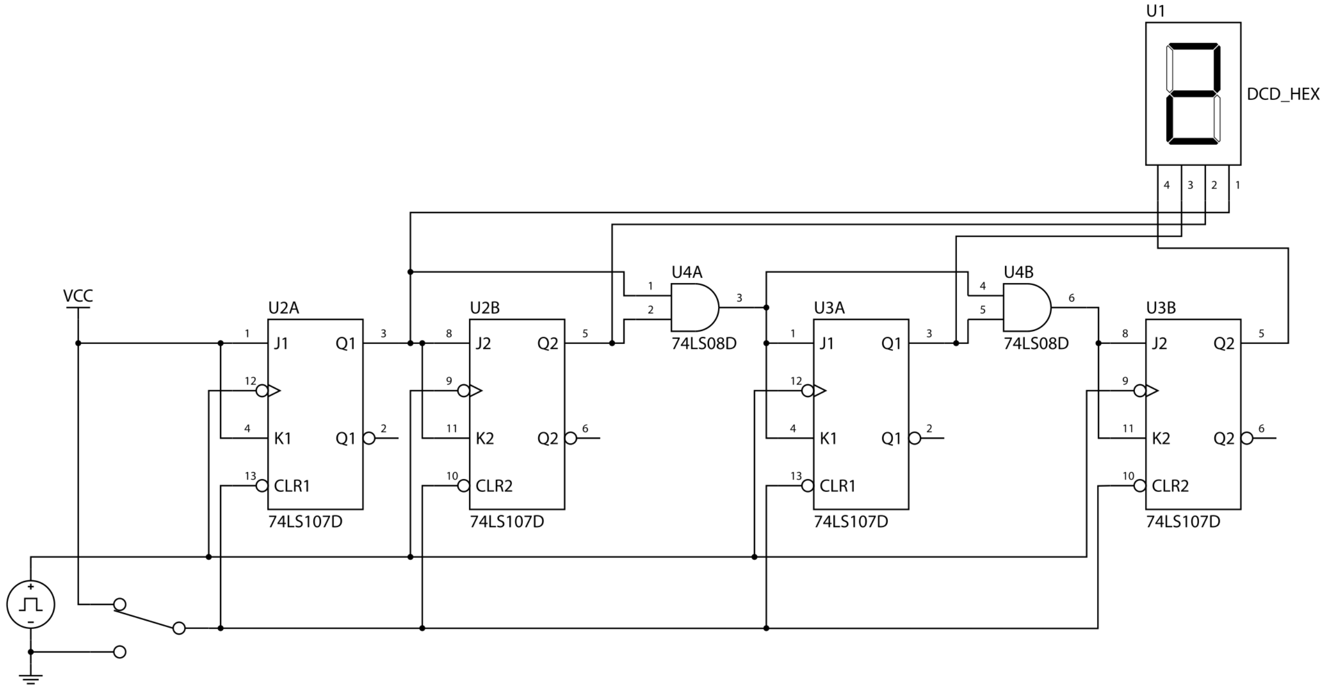
Arbitrary Code

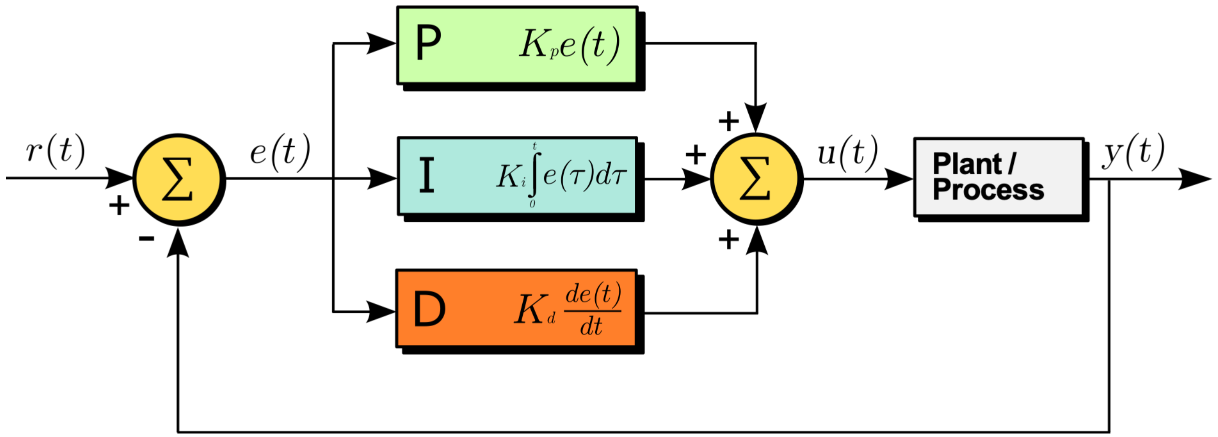
Domain Specific Languages

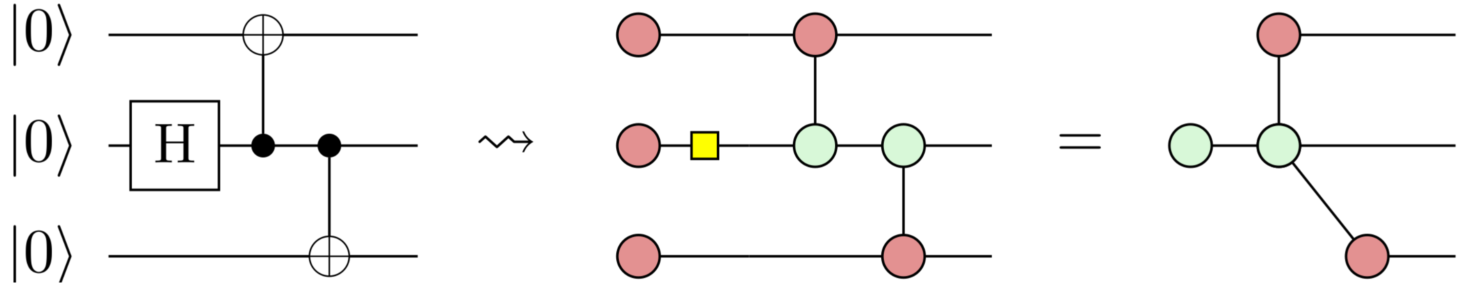
Modeling Frameworks

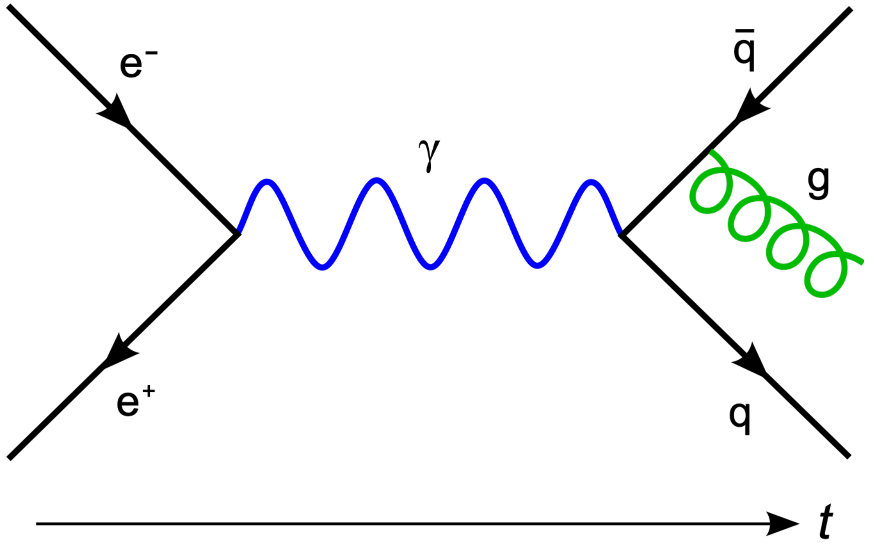
Computer Algebra Systems

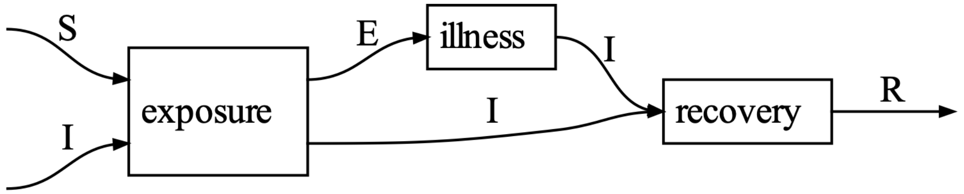
Formal Scientific Diagrams











Three Layers of GAT Based Modeling

AlgebraicJulia

Theory

$$A, B, C$$

$$a : A, b : B, c : C$$

$$a \otimes b \cdot c$$

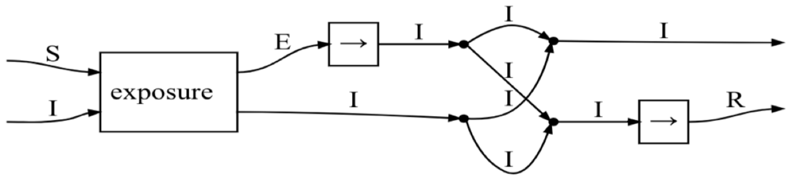
$$(a \otimes b) \cdot (c \otimes d) \\ = (a \cdot c) \otimes (b \cdot d)$$

Syntax

Formula Notation

$$seir = expo \cdot (f_{E,I} \otimes id_I) \cdot \nabla_I \cdot \Delta_I \cdot (id_I \otimes f_{I,R})$$

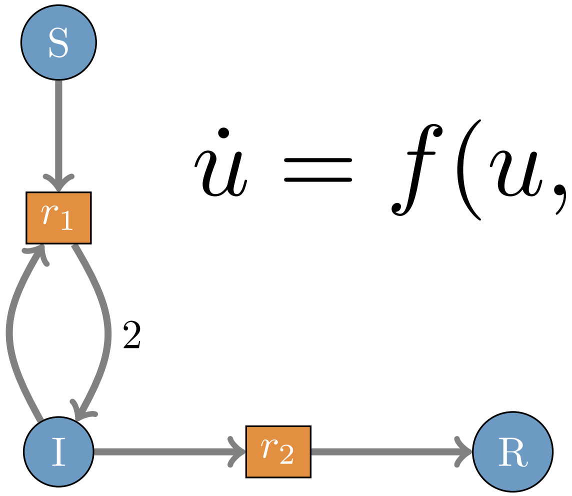
Wiring Diagram



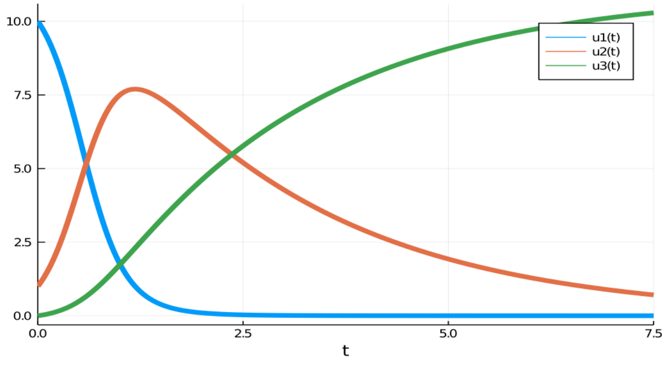
Embedded Domain Specific Language

```
d = @program Disease (s::S, e::E, i::I) begin
    e1, i1 = exposure{S,I,E}(s,i)
    i2 = spontaneous{E,I}(e1)
    e = [e, e1]
    e_out = spontaneous{E,E}(e)
    i1 = [i1, i2]
    r = spontaneous{I,R}(i1)
    s_out = spontaneous{S,S}(s)
    return s_out, e_out, spontaneous{I,I}(i1)
end
```

Instance



$$\dot{u} = f(u, t)$$



DSLs implemented in Catlab

AlgebraicJulia

@theory	Algebraic Structure that defines possible expressions (the theory of Groups)
@syntax	`GATExpr: Expr` tied to a specific Theory
@presentation	A specific example of the theory: (the Group of Integers mod 7)
@instance	A Julia implementation (types and functions) implementing the theory
@program	Lets you express formulas using program notation

FinOrd: the Category of Natural Numbers

AlgebraicJulia

```julia
@theory Category(Ob,Hom) begin
  Ob::TYPE
  Hom(dom::Ob,codom::Ob)::TYPE

  id(A::Ob)::(A → A)
  compose(f::(A → B), g::(B → C))::(A → C)
          ⊣ (A::Ob, B::Ob, C::Ob)
end
```

```julia
@instance Category(FinOrd, FinOrdMap) begin
  dom(f::FinOrdMap) = FinOrd(f.dom)
  codom(f::FinOrdMap) = FinOrd(f.codom)

  id(A::FinOrd) = FinOrdFunction(identity, A, A)

  function compose(f::FinOrdMap, g::FinOrdMap)
    @assert codom(f) == dom(g)
    FinOrdFunction(compose(f.func,g.func), dom(f), codom(g))
  end
end
```

```julia
struct FinOrd n:: Int end

struct FinOrdFunc <: FinOrdMap
  func::Function
  dom::Int
  codom::Int
end

(f::FinOrdFunc)(x) = f.func(x)

struct FinOrdVec <: FinOrdMap
  func::Vector{Int}
  codom::Int
end

(f::FinOrdVec)(x) = f.func[x]
```

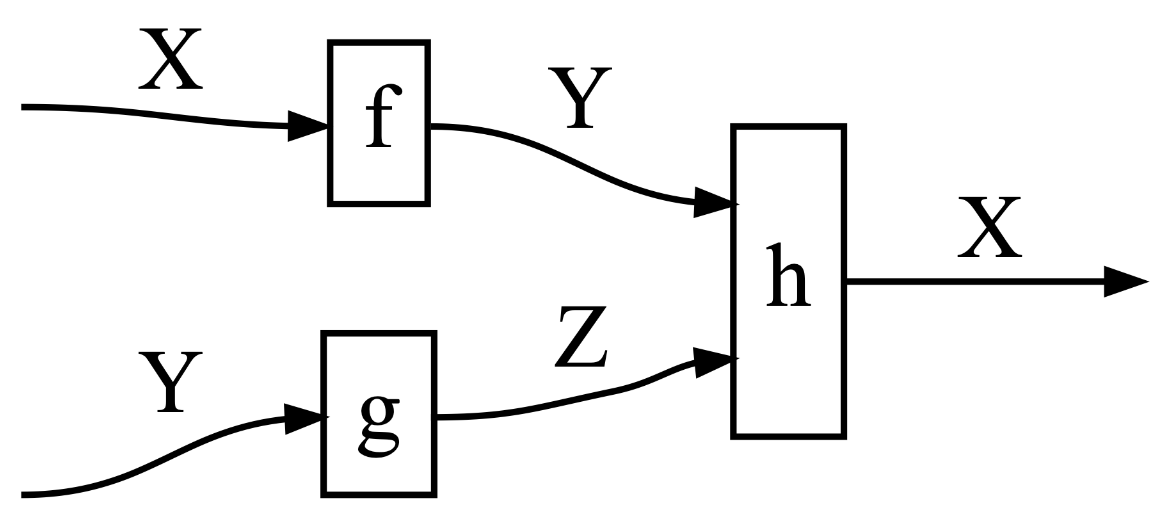
Monoidal Categories Support Programming

AlgebraicJulia

```
@present P(FreeSymmetricMonoidalCategory) begin
    X::Ob
    Y::Ob
    Z::Ob

    f::Hom(X,Y)
    g::Hom(Y,Z)
    h::Hom(Y⊗Z, X)
end
```

$$\begin{tikzcd} X \otimes Y \arrow[r] \arrow[d] & Y \arrow[r, "g"] & Z \\ X \arrow[ur, "f"'] & Y \otimes Z \arrow[u] \arrow[ur] \arrow[r, "h"'] & X \end{tikzcd}$$

$$(f \otimes g) \cdot h \colon X \otimes Y \to X$$




```
@signature Category(Ob,Hom) => SymmetricMonoidalCategory(Ob,Hom
  otimes(A::Ob, B::Ob)::Ob
  otimes(f::(A → B), g::(C → D))::((A ⊗ C) → (B ⊗ D)) ⊣
    (A::Ob, B::Ob, C::Ob, D::Ob)
  @op (⊗) := otimes
  munit()::Ob
  braid(A::Ob, B::Ob)::((A ⊗ B) → (B ⊗ A))
  @op (σ) := braid
end
```

```
d = @program P (x::X,y::Y) begin
      a = f(x)
      b = g(y)
      z = h(a,b)
      return z
end
```

Programming with Pictures

AlgebraicJulia

Initial Object

$$0 \longrightarrow A$$

```
initial(::Type{FinOrd}) = FinOrd(0)
```

Coproducts

$$\begin{tikzcd}
 & B \arrow[d, "i_2"] \\
A \arrow[r, "i_1"] & A \sqcup B
\end{tikzcd}$$

```
function coproduct(A::FinOrd, B::FinOrd)
  m, n = A.n, B.n
  ι1 = FinOrdMap(1:m, m, m+n)
  ι2 = FinOrdMap(m+1:m+n, n, m+n)
  Cospan(ι1, ι2)
end
```

Coequalizers

$$E \xleftarrow{u} A \leftleftarrows B$$

Pushout of f, g,

$u = coequalizer(f \cdot i_1, g \cdot i_2)$

$$\begin{tikzcd}
A & & \\
 & X \sqcup Y \arrow[ul, "u"'] & Y \arrow[l, "i_2"'] \arrow[ull, bend right] \\
 & X \arrow[u, "i_1"] \arrow[uul, bend left] & Z \arrow[l, "f"] \arrow[u, "g"']
\end{tikzcd}$$

```
function pushout(span::Span{<:FinOrdMap,<:FinOrdMap})
  f, g = left(span), right(span)
  coprod = coproduct(codom(f), codom(g))
  ι1, ι2 = left(coprod), right(coprod)
  coeq = coequalizer(f⋅ι1, g⋅ι2)
  Cospan(ι1⋅coeq, ι2⋅coeq)
end
```

Application: Simulating a Petri Net as ODEs

AlgebraicJulia

Novel coronavirus 2019-nCoV: early estimation of epidemiological parameters and epidemic predictions
Version 2. Updated 27 Jan 2020

Jonathan M. Read[1], Jessica R.E. Bridgen[1], Derek A.T. Cummings[2], Antonia Ho[3], Chris P. Jewell[1]

Affiliations:
1. Centre for Health Informatics, Computing and Statistics, Lancaster Medical School, Lancaster University, Lancaster, United Kingdom.
2. Department of Biology and Emerging Pathogens Institute, University of Florida, Gainesville, United States of America.
3. Medical Research Council-University of Glasgow Centre for Virus Research, Glasgow, United Kingdom.

Correspondence: jonathan.read@lancaster.ac.uk

Impact of international travel and border control measures on the global spread of the novel 2019 coronavirus outbreak

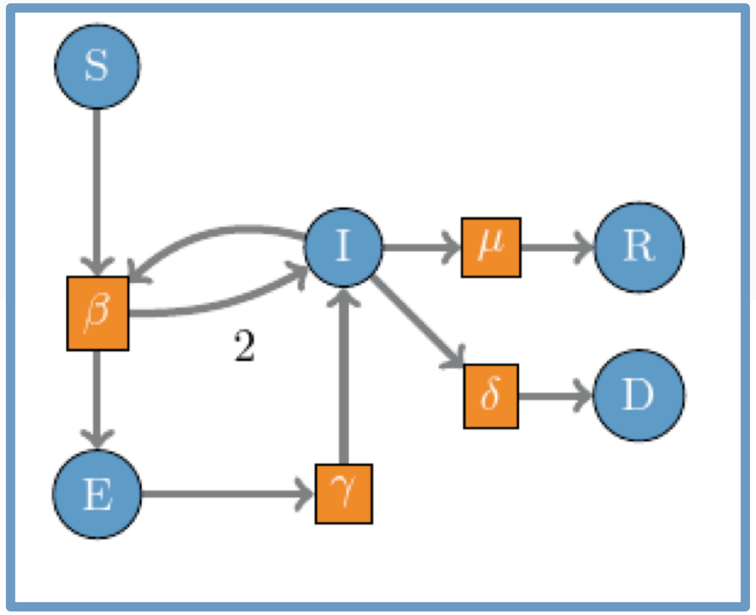
Chad R. Wells[a], Pratha Sah[a], Seyed M. Moghadas[b], Abhishek Pandey[a], Affan Shoukat[a], Yaning Wang[c], Zheng Wang[d], Lauren A. Meyers[e,f], Burton H. Singer[g,1], and Alison P. Galvani[a]

[a]Center for Infectious Disease Modeling and Analysis, Yale School of Public Health, New Haven, CT 06520; [b]Agent-Based Modelling Laboratory, York University, Toronto, ON M3J 1P3, Canada; [c]State Key Laboratory of Mycology, Institute of Microbiology, Chinese Academy of Sciences, 100101 Beijing, China; [d]Department of Biostatistics, Yale School of Public Health, New Haven, CT 06510; [e]Department of Integrative Biology, The University of Texas at Austin, Austin, TX 78712; [f]Santa Fe Institute, Santa Fe, NM 87501; and [g]Emerging Pathogens Institute, University of Florida, Gainesville, FL 32610

Contributed by Burton H. Singer, February 27, 2020 (sent for review February 12, 2020; reviewed by Yoav Keynan and Heman Shakeri)

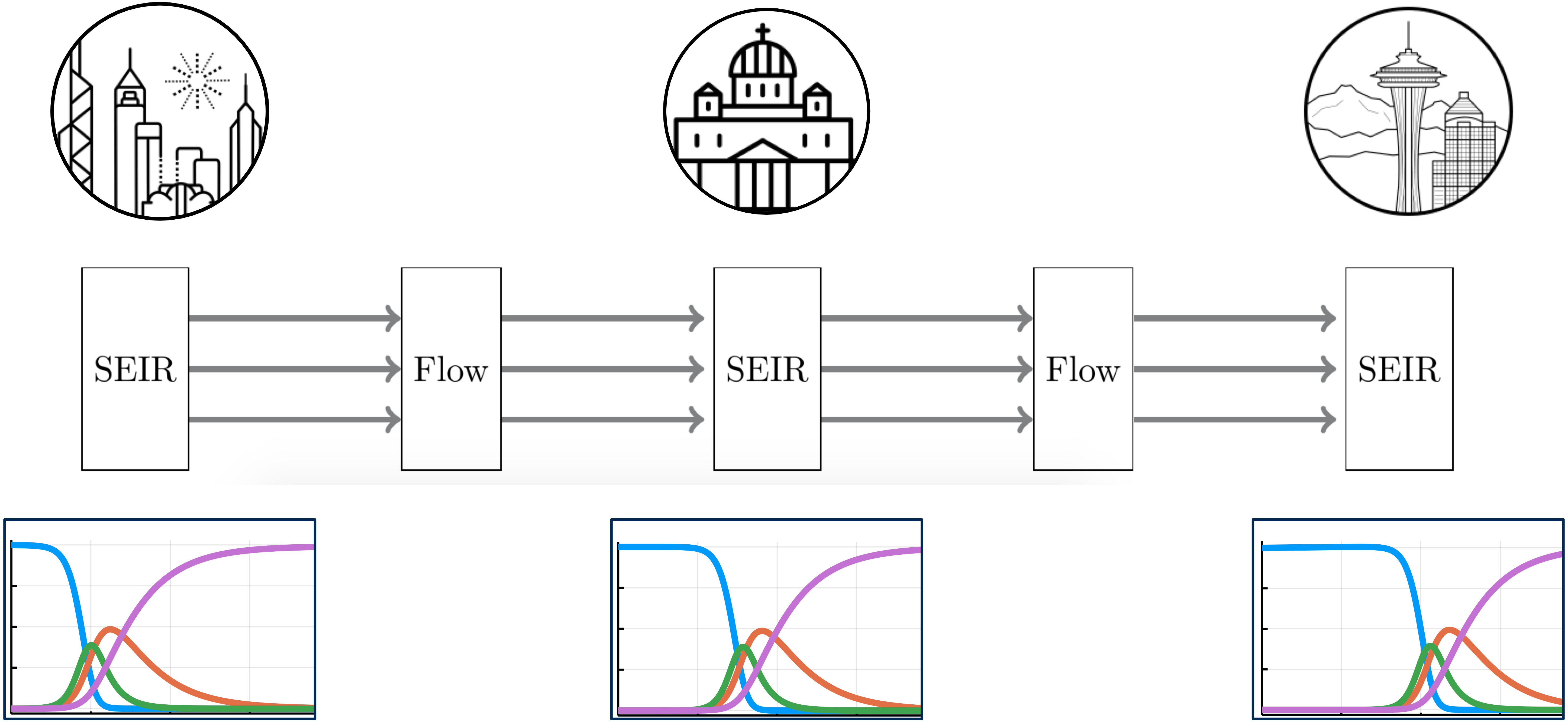
Solve the initial value problem for $\dot{u} = f(u,t)$

```
f(u, p) = begin
    for (i, t) in enumerate(reactions)
        reagents = t[1]
        products = t[2]
        ϕ = p[i]*prod(u[reagents])
        du[reagents] .-= ϕ
        du[products] .+= ϕ
    end
    return du
end
```



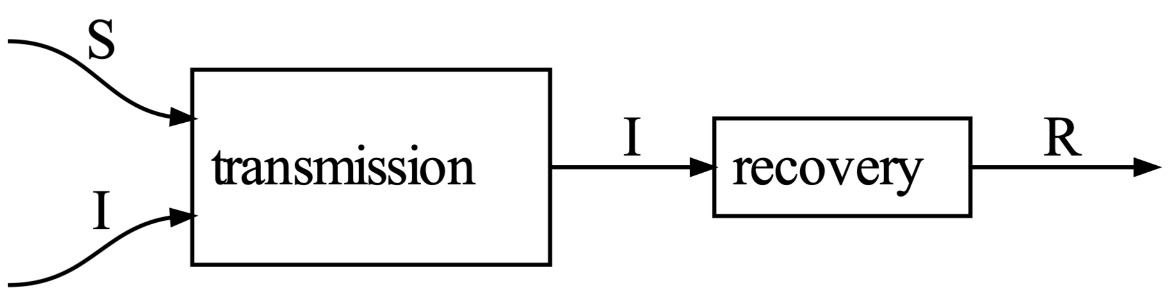
Epidemiology Modeling Framework

AlgebraicJulia



Basic SIR model

sir = transmission · recovery



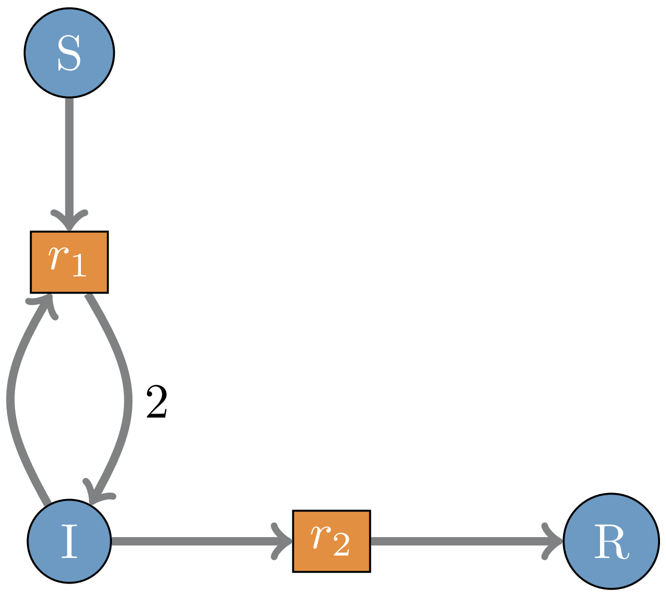
f = vectorfield(sirᵈ)

$$\dot{u}_1 = -r_1 u_1 u_2$$

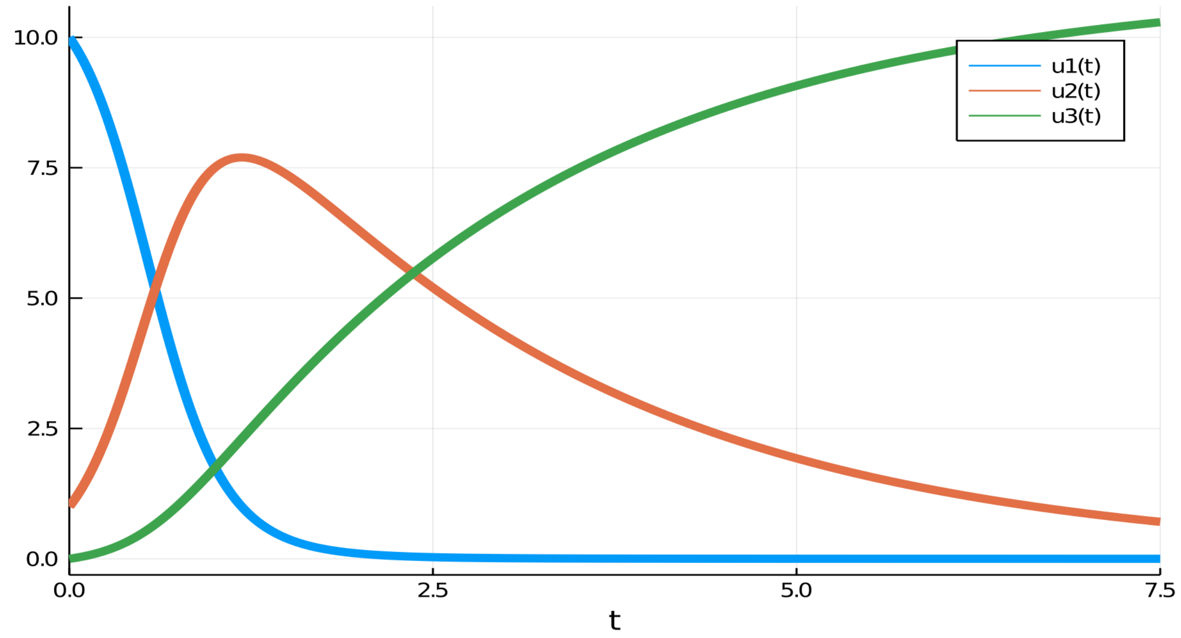
$$\dot{u}_2 = r_1 u_1 u_2 - r_3 u_2$$

$$\dot{u}_3 = r_3 u_2$$

sirᵈ = decoration(F(sir))



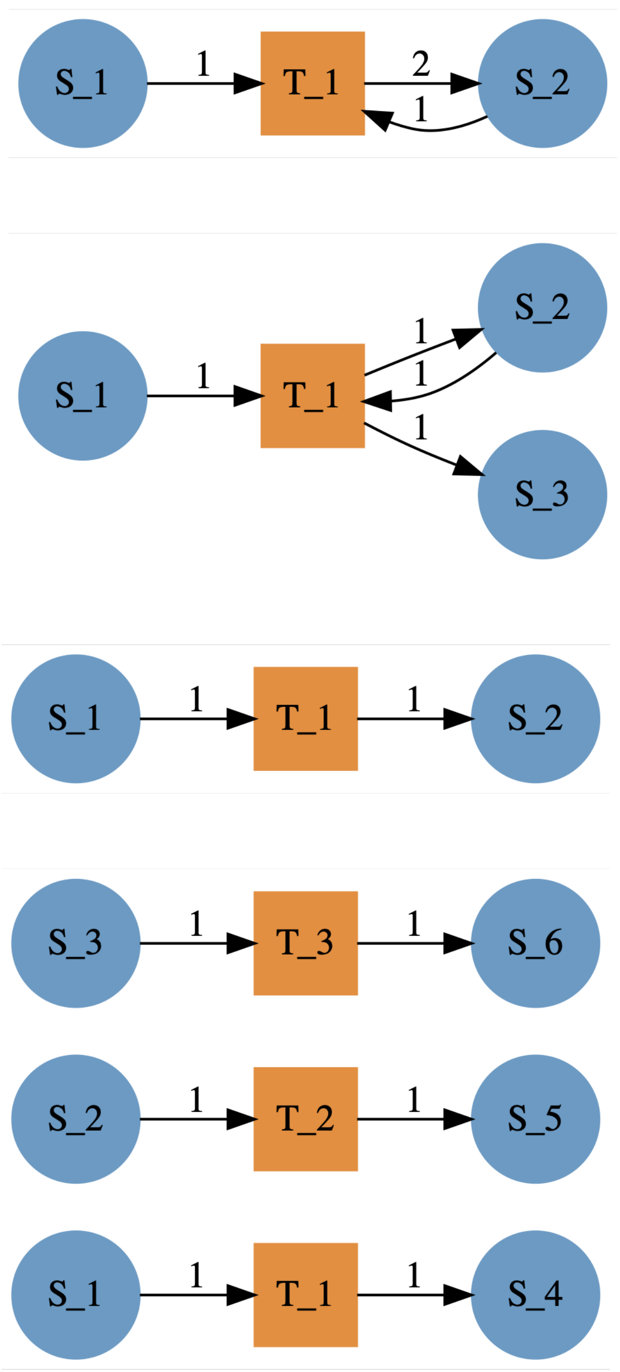
sol = solve(f, u_0, r, (t_0, t_1))



Epidemiology Modeling

```
@present Epidemiology(FreeBiproductCategory) begin
    S::Ob
    E::Ob
    I::Ob
    R::Ob
    D::Ob
    transmission::Hom(S⊗I, I)
    exposure::Hom(S⊗I, E⊗I)
    illness::Hom(E,I)
    recovery::Hom(I,R)
    death::Hom(I,D)
    travel::Hom(S⊗E⊗I,S⊗E⊗I)
end
```

$$Epi \xrightarrow{F} Petri$$



Computing Decorated Cospans

AlgebraicJulia

Cospan Composition

$$
\begin{tikzcd}
 & & C & & \\
 & A \arrow[ur, "f'"] \arrow[r, "\iota_A"] & A \sqcup B \arrow[u, "u"] & B \arrow[ul, "g'"'] \arrow[l, "\iota_B"'] & \\
X \arrow[ur] & & Y \arrow[ul, "f"] \arrow[ur, "g"'] & & Z \arrow[ul]
\end{tikzcd}
$$

Decorator Composition

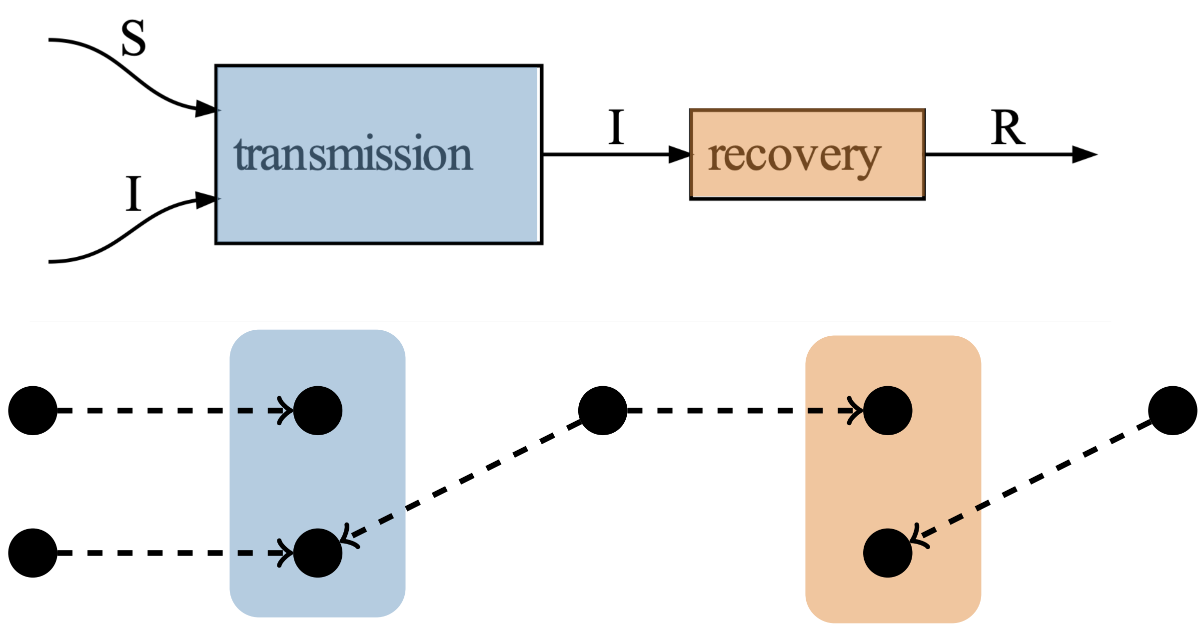
$$
\begin{tikzcd}
1 \arrow[rr, "d_A"] \arrow[dr] \arrow[dd, "d_B"'] & & FA \arrow[dl, "Fi_A"] \arrow[dd, "Ff'"] \\
 & FA \times FB \arrow[dr, "Fu"] & \\
FB \arrow[ur, "Fi_B"] \arrow[rr, "Fg'"'] & & FC
\end{tikzcd}
$$

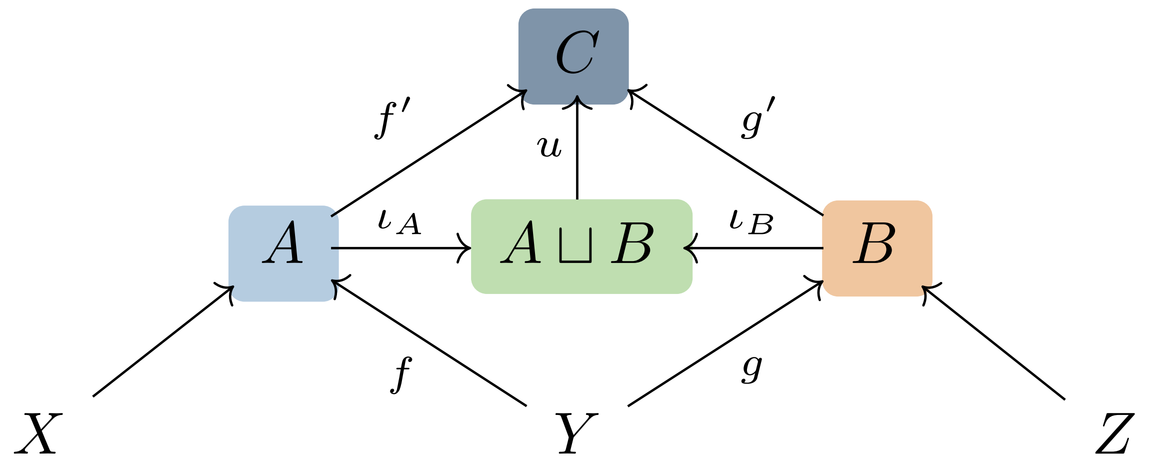
$$Epi \xrightarrow{F} Petri$$

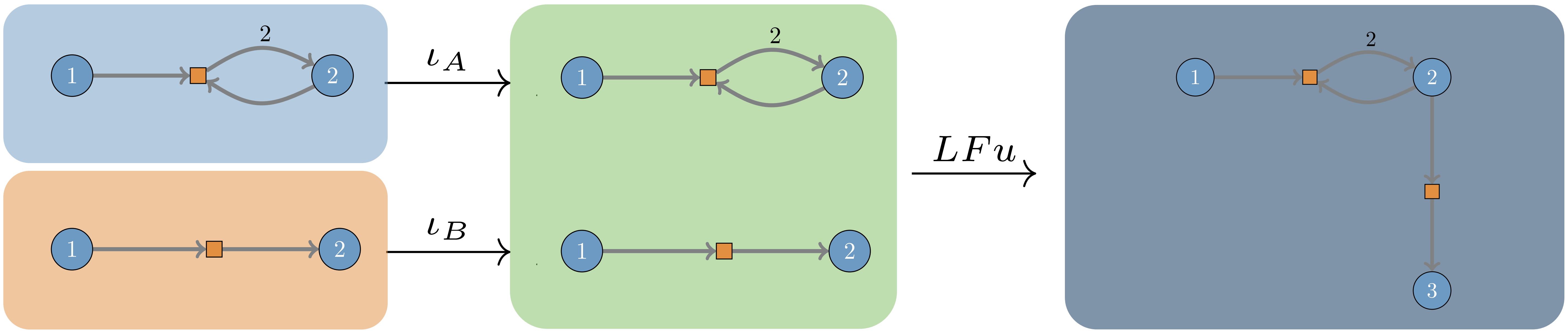
$$FA \sqcup FB \xrightarrow{L} F(A \sqcup B)$$

```
compose(p::PetriCospan, q::PetriCospan) = begin

  u, f', g' = pushout(f,g)

  F = functor(decorator(p))
  L = laxator(decorator(p))

  dc = F(u)(L(decoration(p), decoration(q)))

  return PetriCospan(Cospan(f', g'), decorator(p), dc)
end
```

Petri Net Decorated Cospan Composition

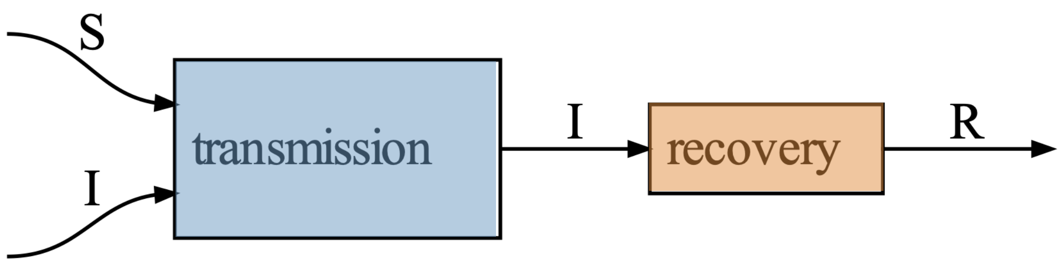


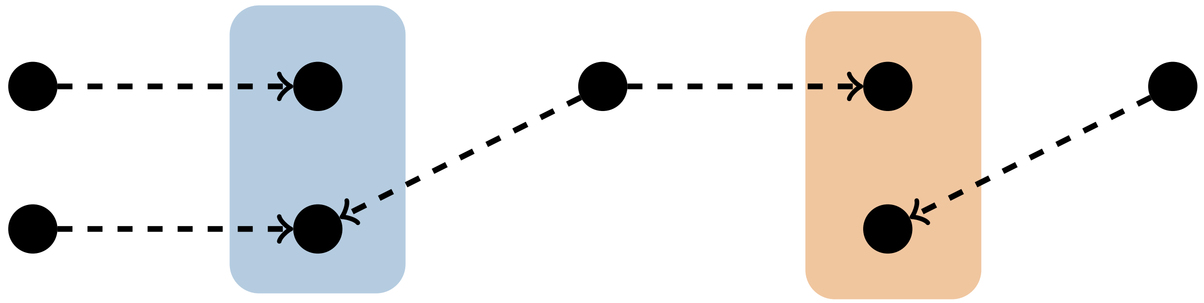


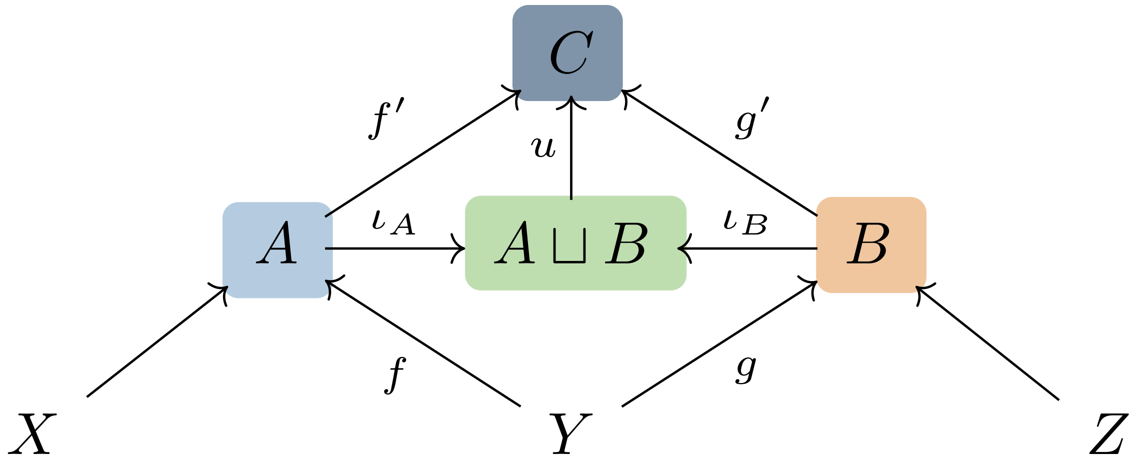


Resource Sharing to Assign Kinetics

AlgebraicJulia







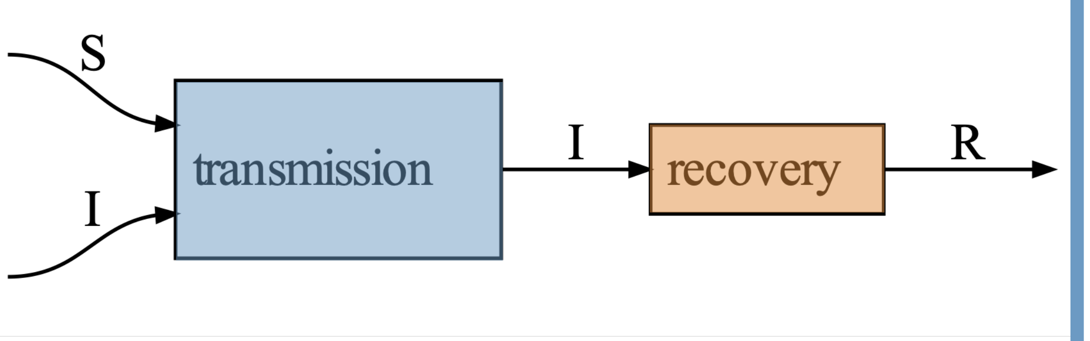
$$\begin{aligned} \dot u_1 &= -r_1 u_1 u_2 \\ \dot u_2 &= r_1 u_1 u_2 \end{aligned} \xrightarrow{\iota_A}$$

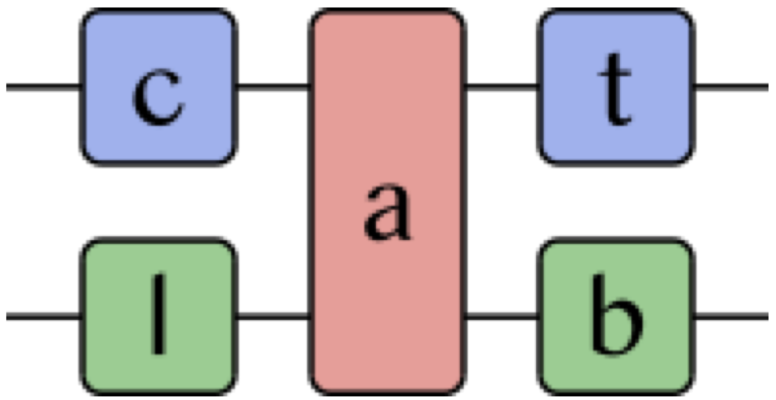
$$\begin{aligned} \dot u_1 &= -r_3 u_2 \\ \dot u_2 &= r_3 u_2 \end{aligned} \xrightarrow{\iota_B}$$

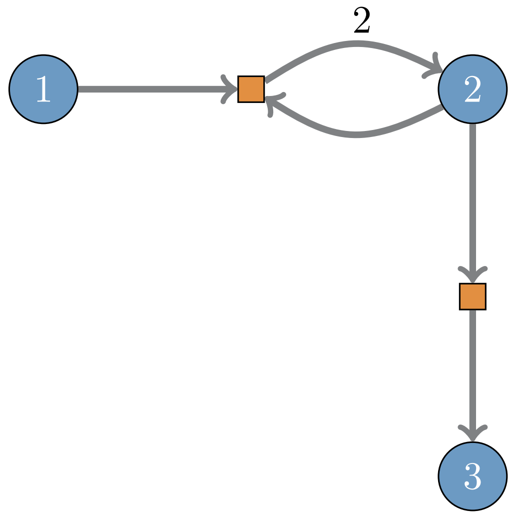
$$\begin{aligned} \dot u_1 &= -r_1 u_1 u_2 \\ \dot u_2 &= r_1 u_1 u_2 \\ \dot u_3 &= -r_3 u_2 \\ \dot u_4 &= r_3 u_2 \end{aligned} \xrightarrow{LFu} \begin{aligned} \dot u_1 &= -r_1 u_1 u_2 \\ \dot u_2 &= r_1 u_1 u_2 - r_3 u_2 \\ \dot u_3 &= r_3 u_2 \end{aligned}$$

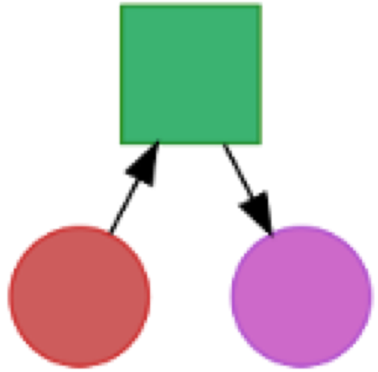
Functorial Modeling Pipeline

$$\mathit{Epi} \xrightarrow{F} \mathit{Petri} \xrightarrow{K} \mathit{Dynam}$$







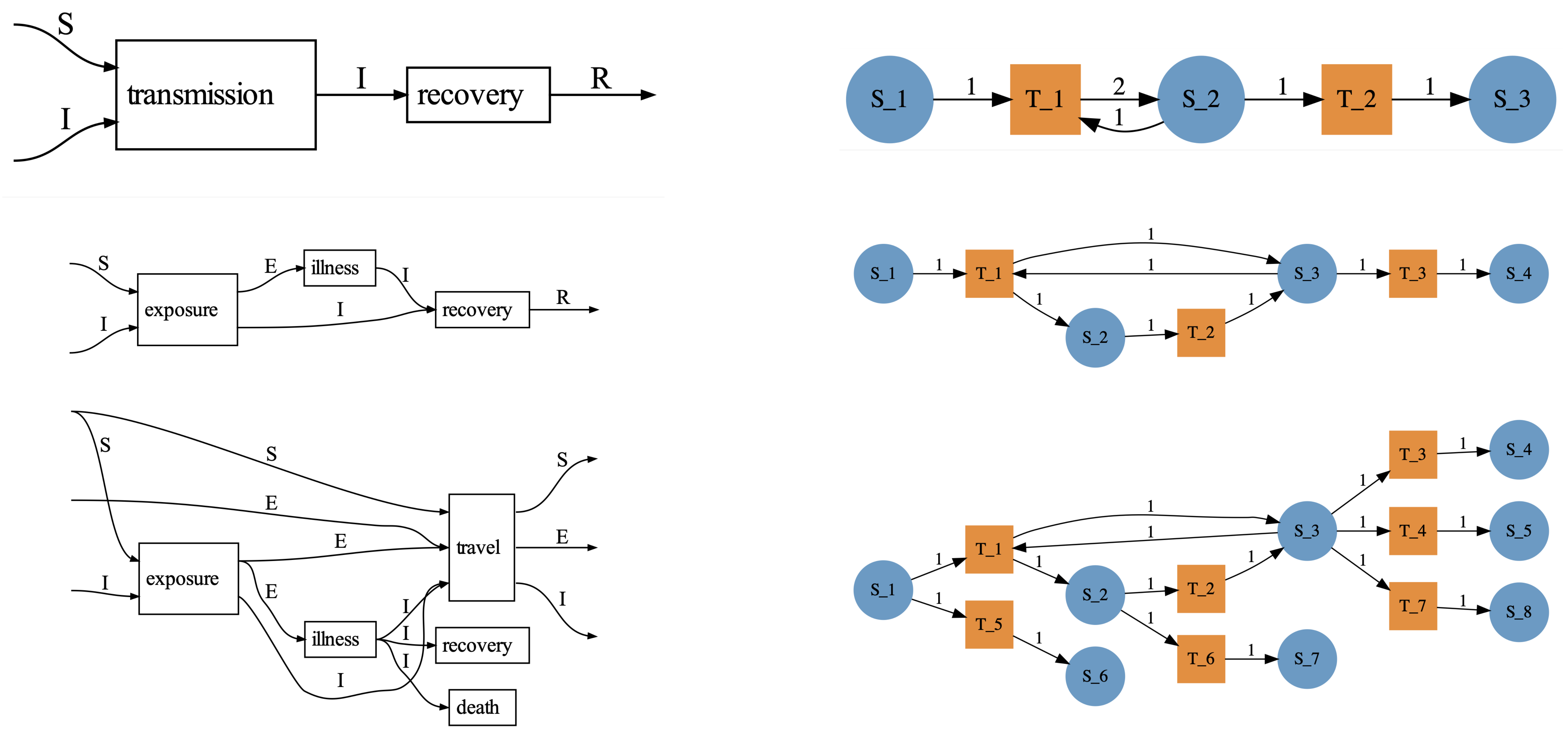


$$\begin{aligned}\dot{u}_1 &= -r_1 u_1 u_2 \\ \dot{u}_2 &= r_1 u_1 u_2 - r_3 u_2 \\ \dot{u}_3 &= r_3 u_2\end{aligned}$$

EpiModels: SIR, SEIR, SEIRD with Travel

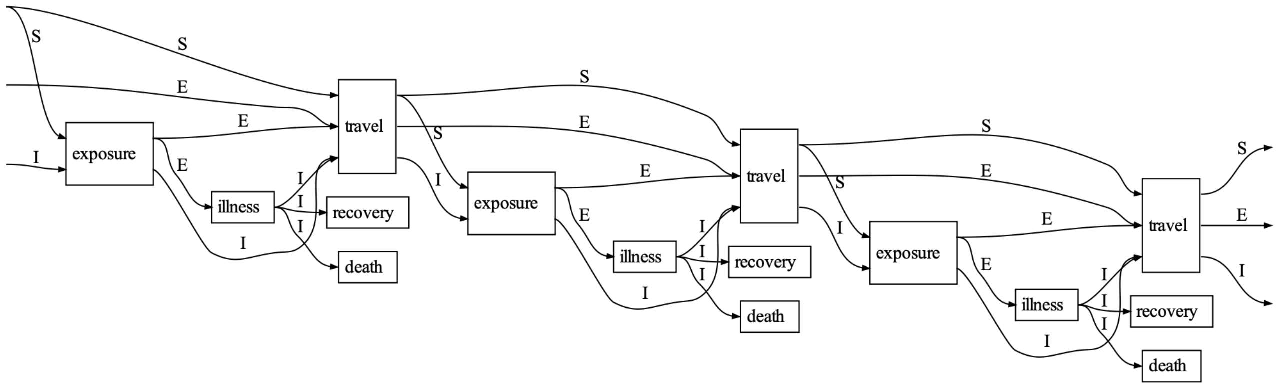
AlgebraicJulia

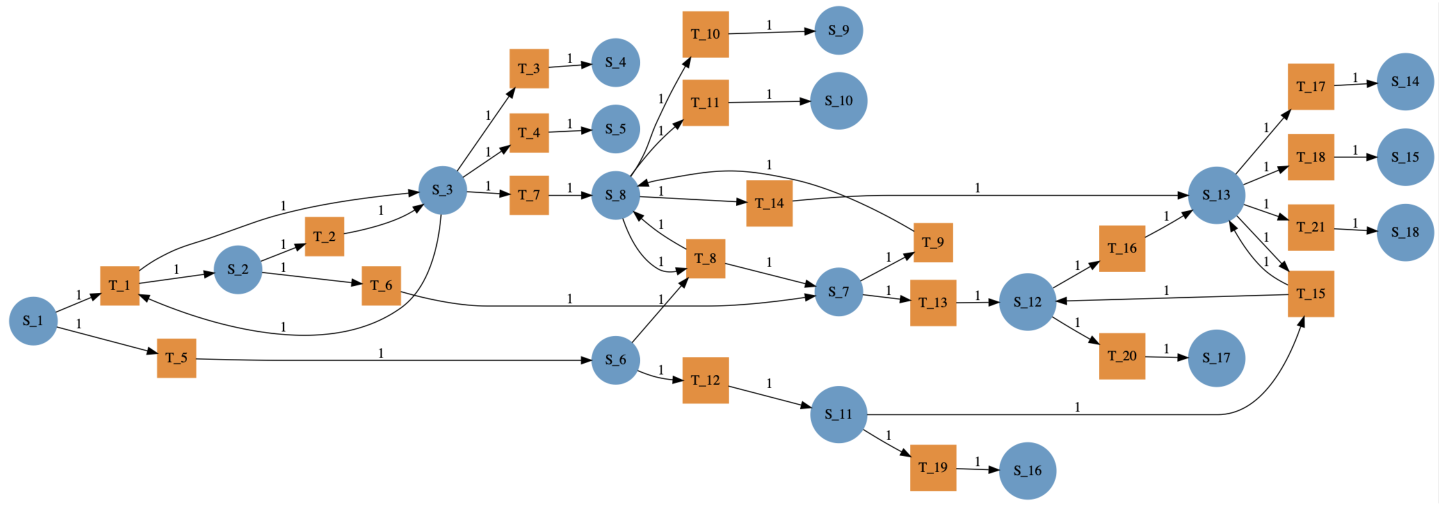
$$Epi \xrightarrow{F} Petri$$

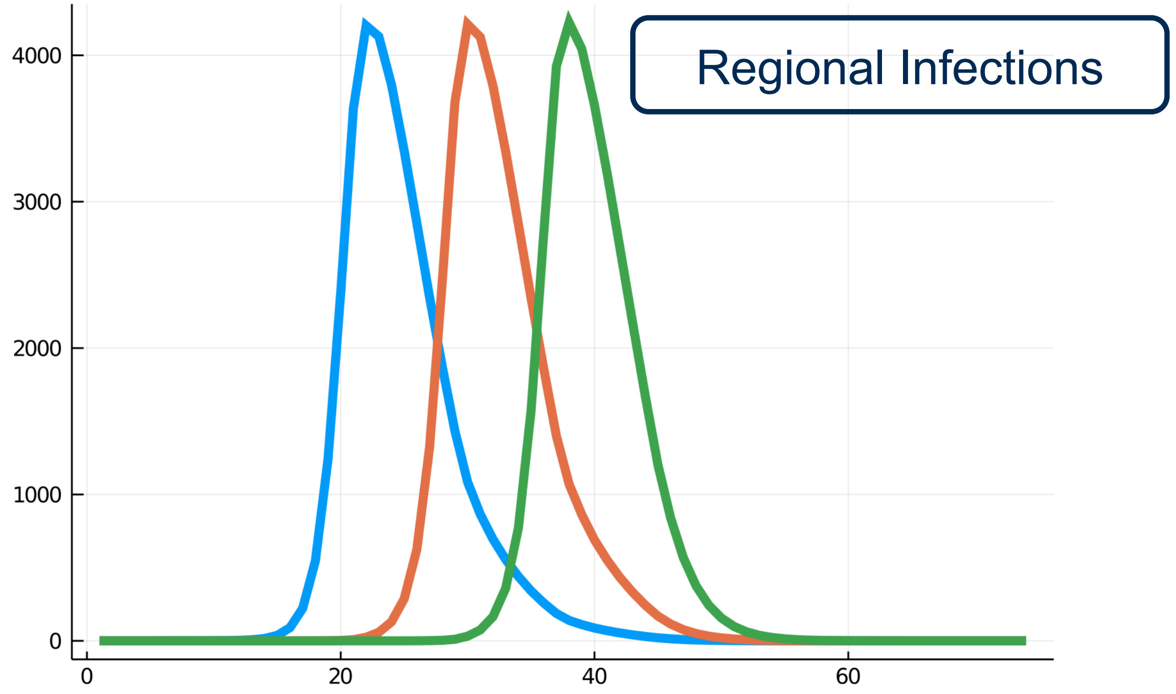


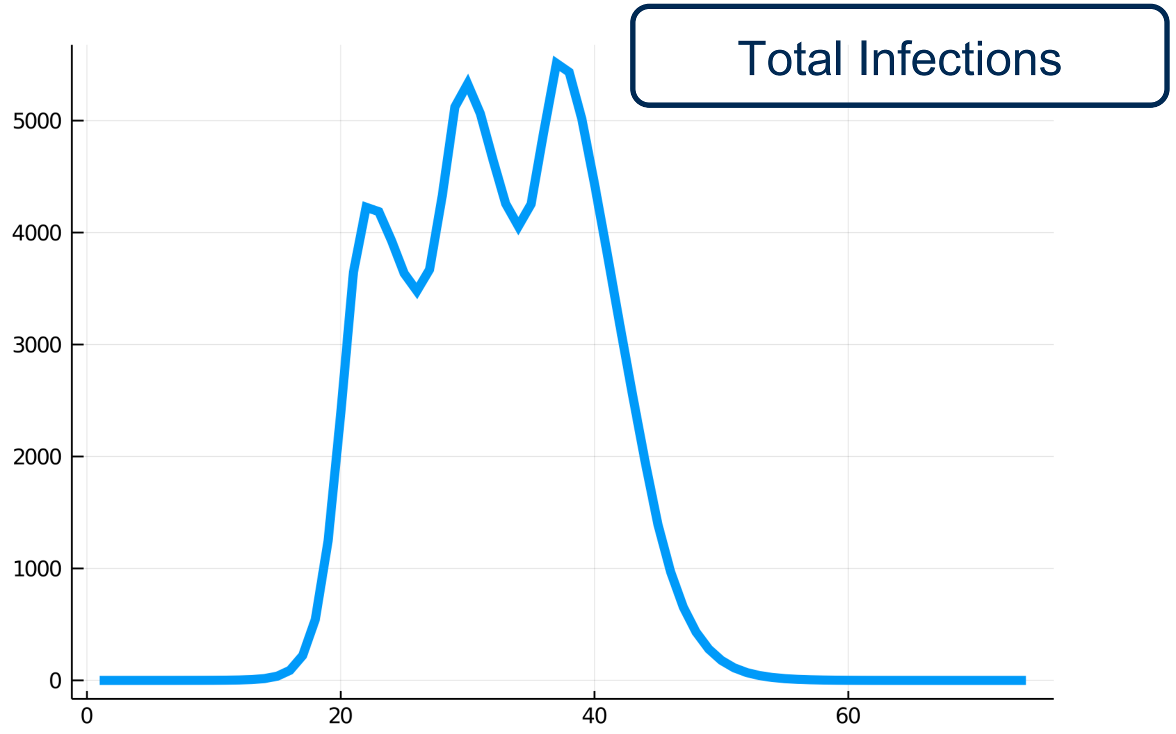
Superposition of Regions Yield Multi-peaks

```
seird_city = @program Epidemiology (s::S, e::E, i::I) begin
    e2, i2 = exposure(s, i)
    i3 = illness(e2)
    d = death(i3)
    r = recovery(i3)
    return travel(s, [e, e2], [i2, i3])
end

^(f,n::Int) = fold(compose, repeat(f,n))
seird_3 = seird_city^3
```









Behavioral Approach to Modeling Systems

AlgebraicJulia

$$Syntax \longrightarrow Semantics$$

Functional Programming

$$Programs \xrightarrow{F} Functions$$

$$\begin{tikzcd}
S \arrow[r, "F"] \arrow[d, "prog"'] & A \arrow[d, "F(prog)"] \\
T \arrow[r, "F"] & B
\end{tikzcd}$$

Behavioral Semantics

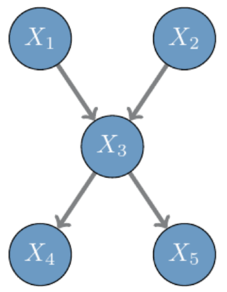
$$Designs \xrightarrow{F} Behaviors$$

$$\begin{tikzcd}
Input \arrow[r, "F"] \arrow[d, "sys"'] & V \arrow[d, "F(sys)"] \\
Output \arrow[r, "F"] & U
\end{tikzcd}$$

Modeling Frameworks use Graphs + Math

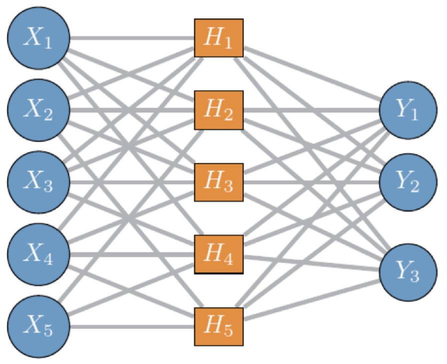
AlgebraicJulia

Bayesian Network



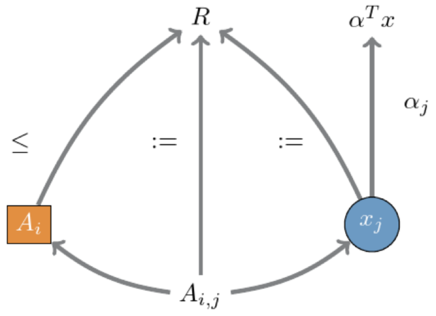
$P(X_4, X_5 \mid X_3)P(X_3 \mid X1, X_2)$

Neural Network



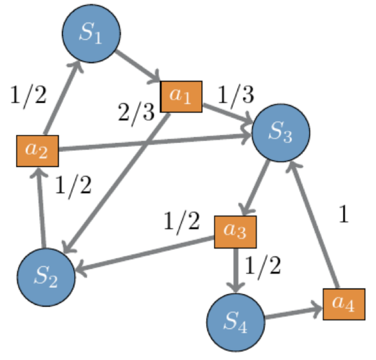
$\min \sum_i \ell_\theta(y, x)$

Optimization Problems



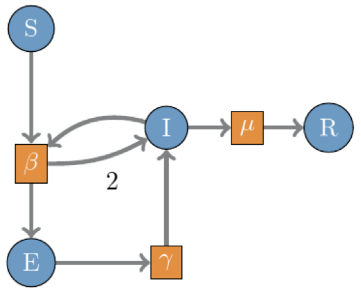
$\min_x \alpha^T x$ s.t. $Ax < b$

Markov Decision Process



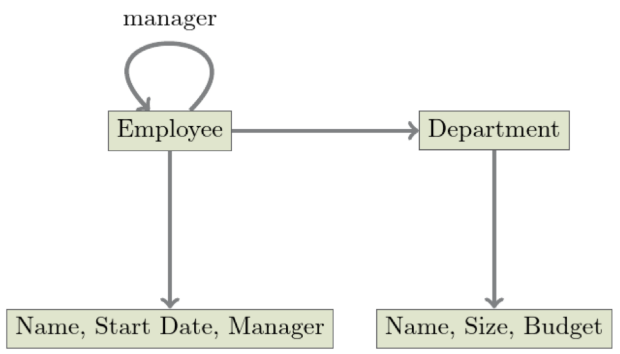
Construct π : *State* $\rightarrow$ *Action* maximizing $E[R]$

Petri Net



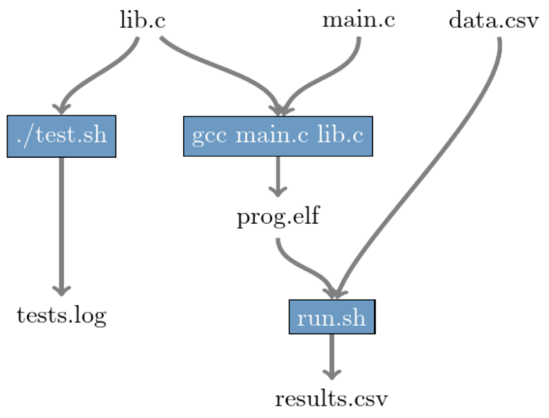
Law of Mass Action: $\dot{u} = f(u, t)$

Databases



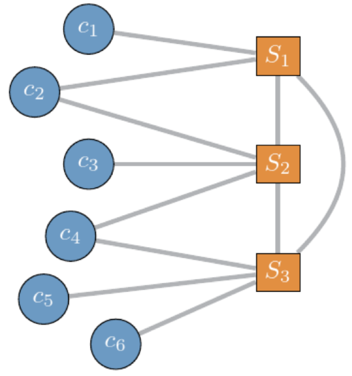
Relational Algebra

Dependency Graph



Scheduling and Compilation

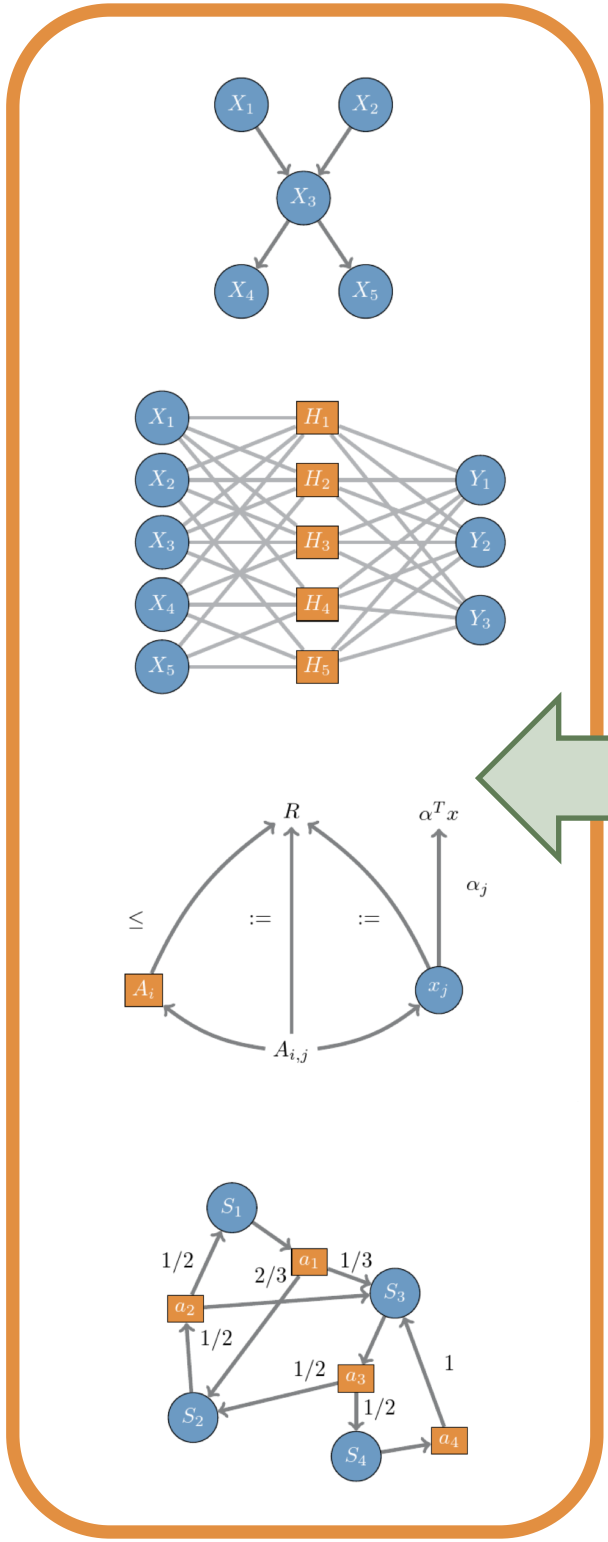
Computer Network

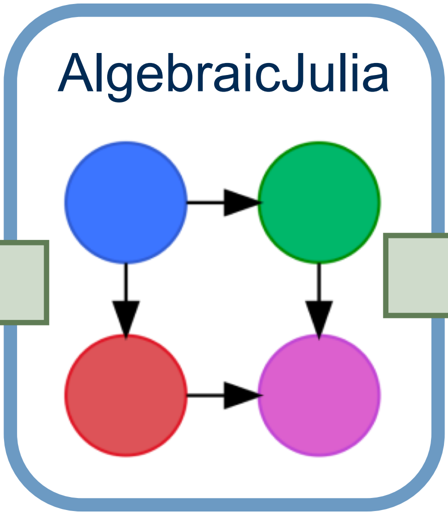


Bandwith, Latency, Routing

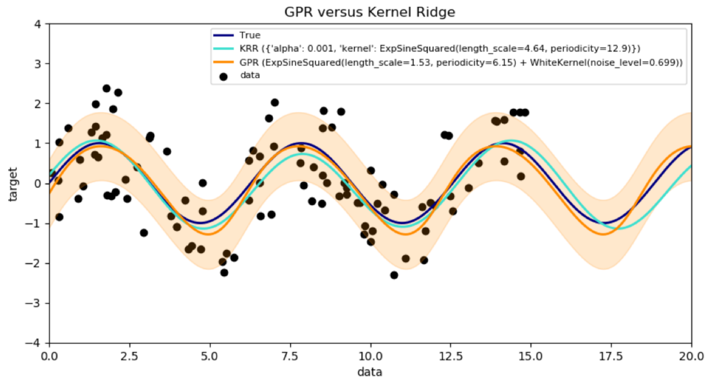
Model Aware Scientific Computing Vision

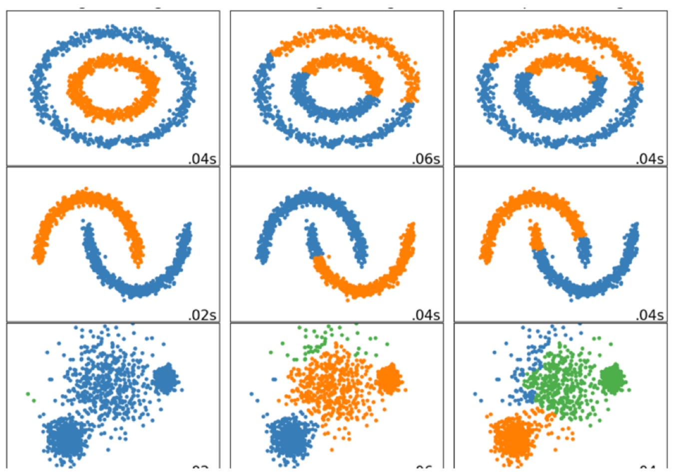
AlgebraicJulia





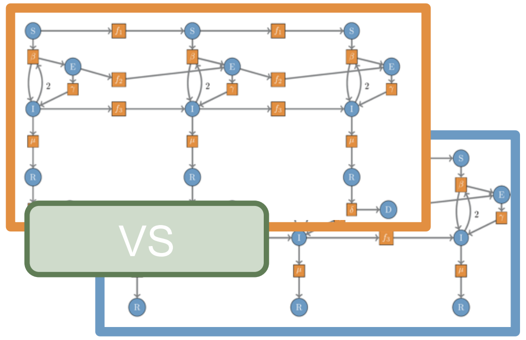
Model Specific Tools for Scientists

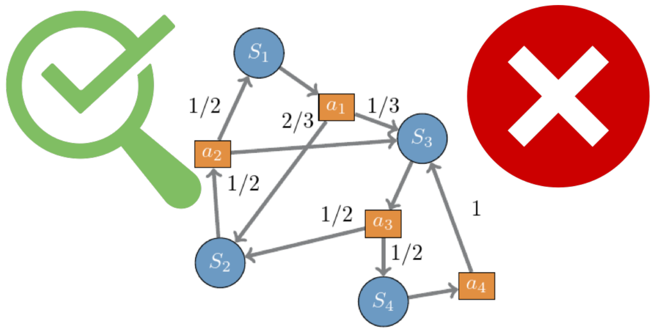




Calibration

Selection





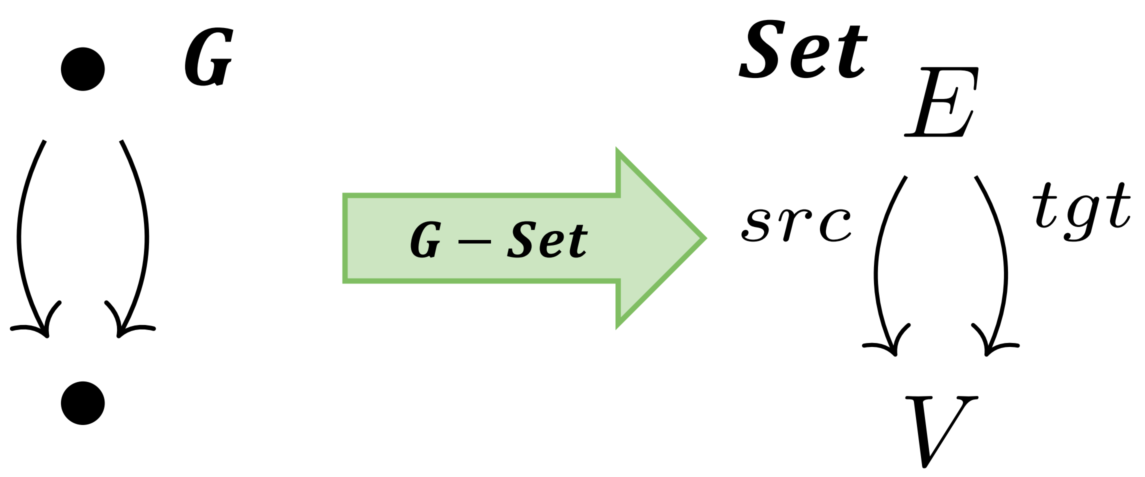
Comparison

Verification & Validation

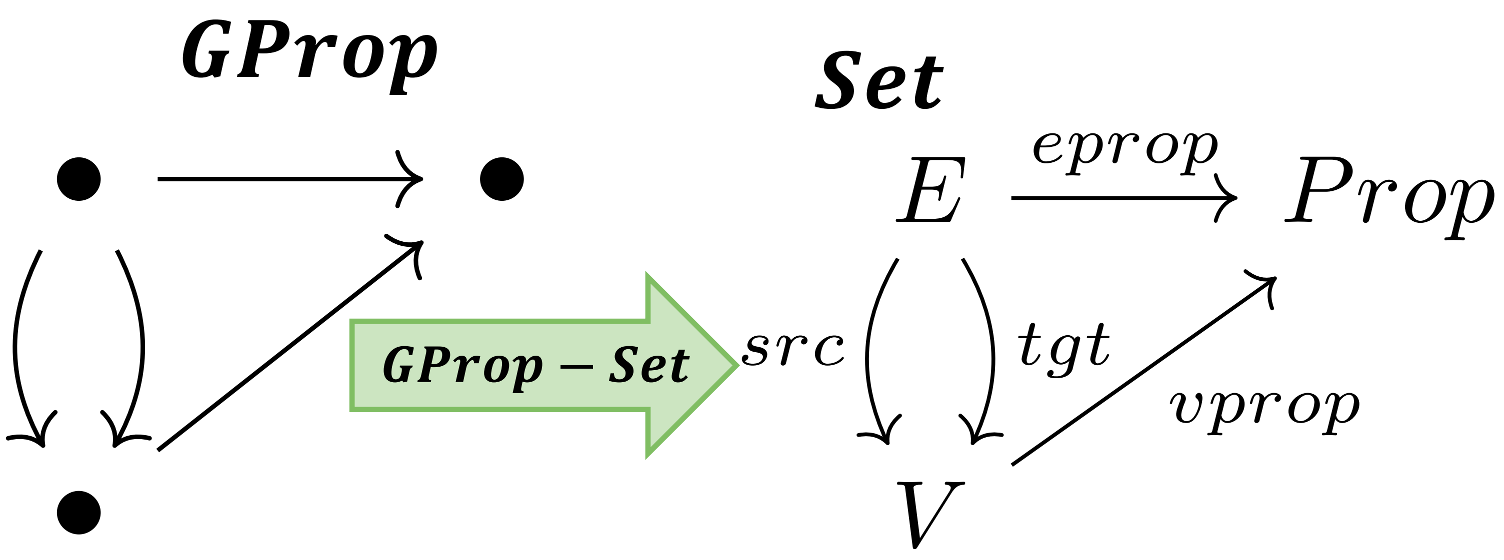
$\mathcal{C}$-Sets: Categorical Data Structures

AlgebraicJulia

Logically extending a data structure with ACT principles



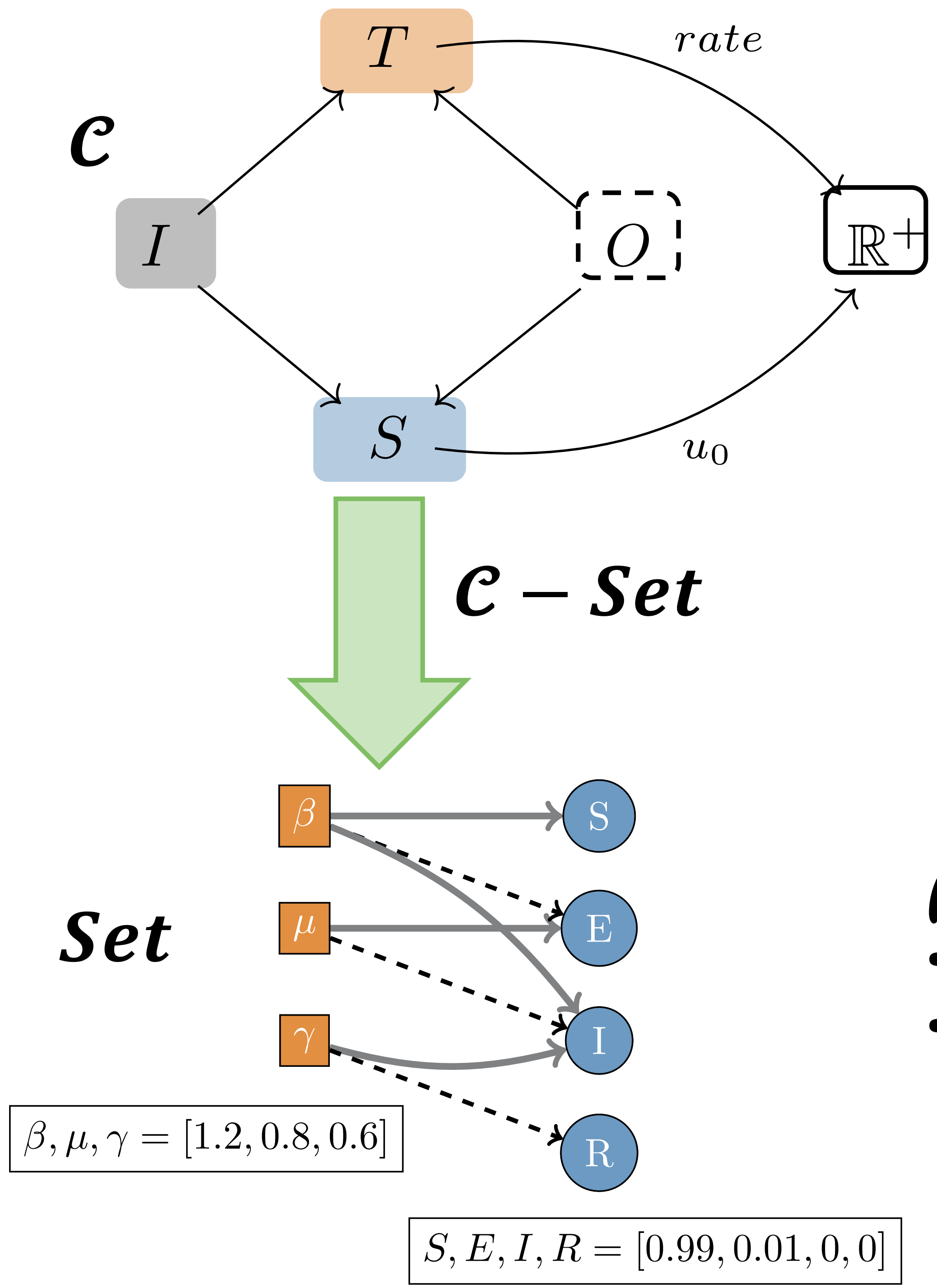

```julia
@present TheoryGraph(FreeCategory) begin
  V::Ob
  E::Ob
  src::Hom(E,V)
  tgt::Hom(E,V)
end
Graph = CSetType(TheoryGraph)
```



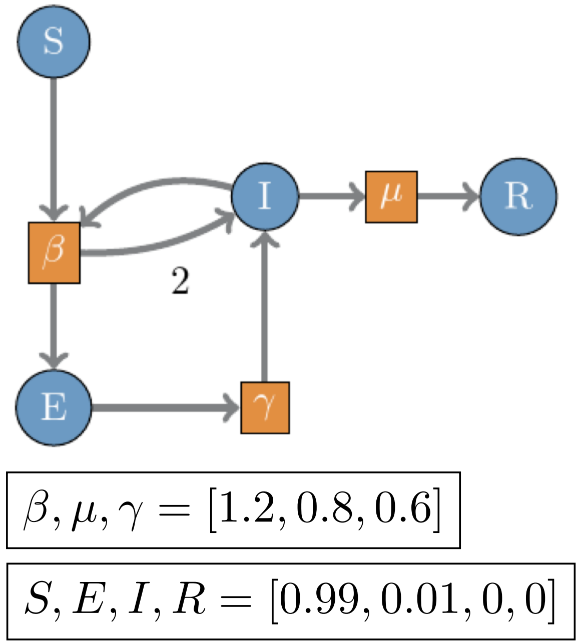

```julia
@present TheoryPropertyGraph <: TheoryGraph begin
  Prop::Ob

  vprops::Hom(V,Prop)
  eprops::Hom(E,Prop)
end

PropertyGraph = CSetType(TheoryPropertyGraph)
```

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

- The data structure is a directed graph
- Instances are Sets with Functions
- Provides compositional framework for structured data
- Mathematically Elegant

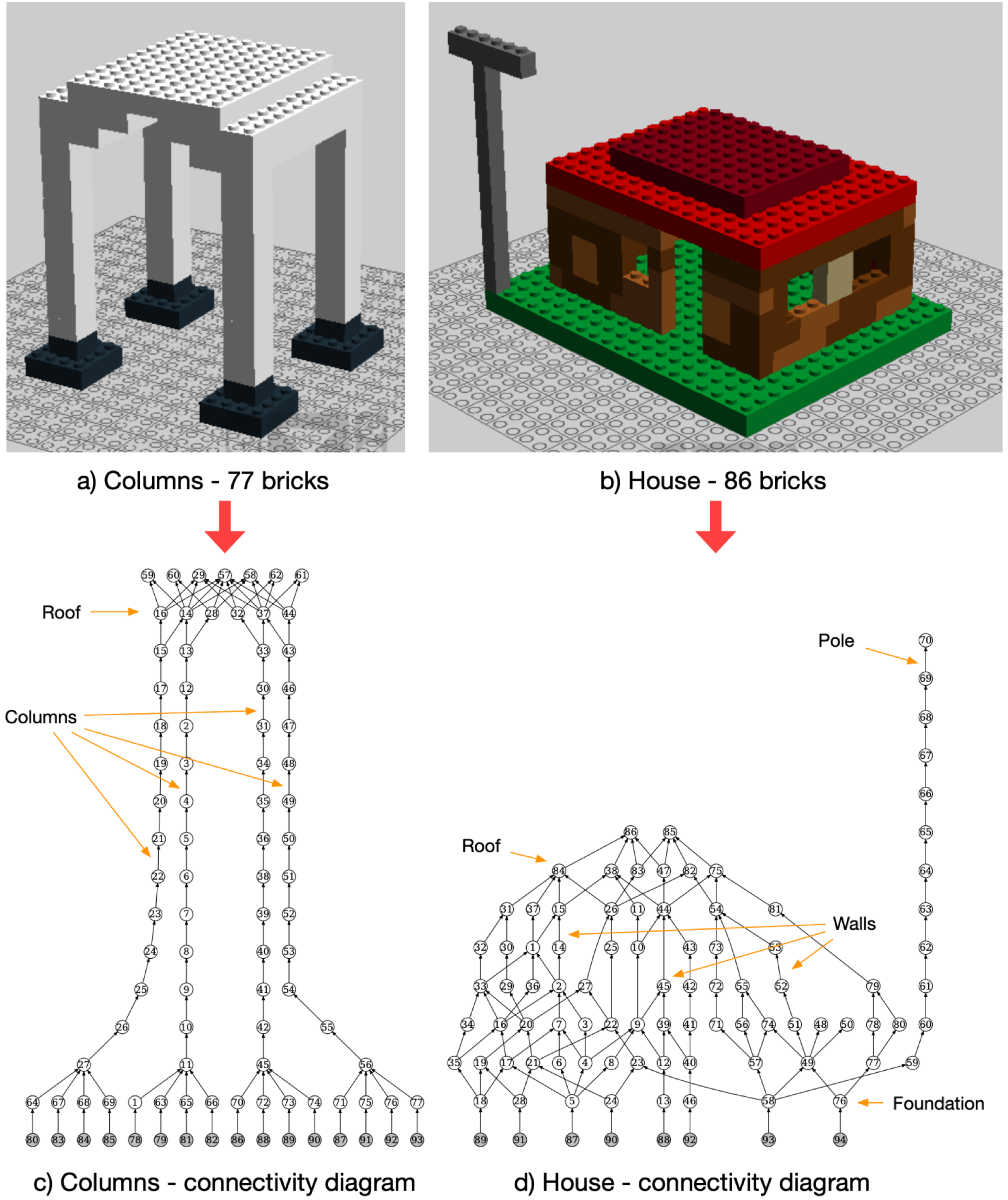


$\cong$



Automatic Parallelization for Construction

AlgebraicJulia



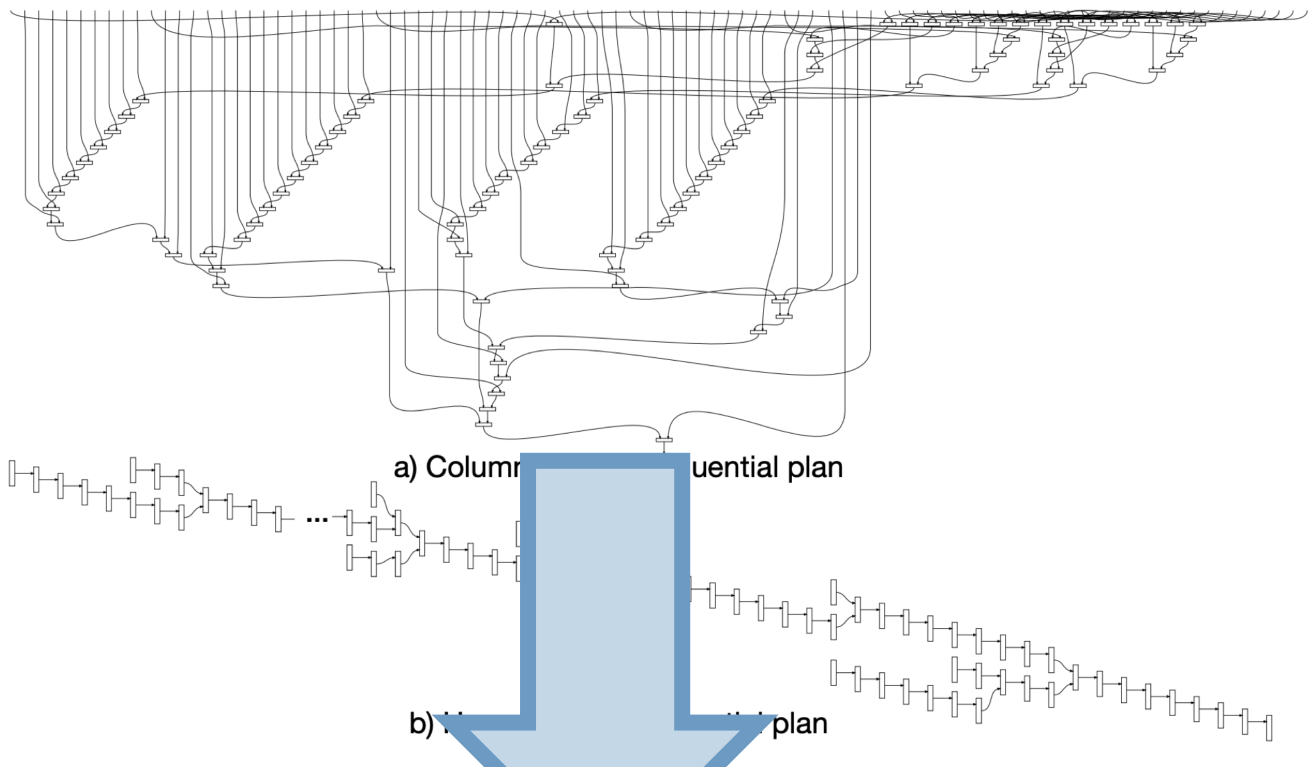


Fig. 4. Sequential plans gen... for the two LEGO CAD models.

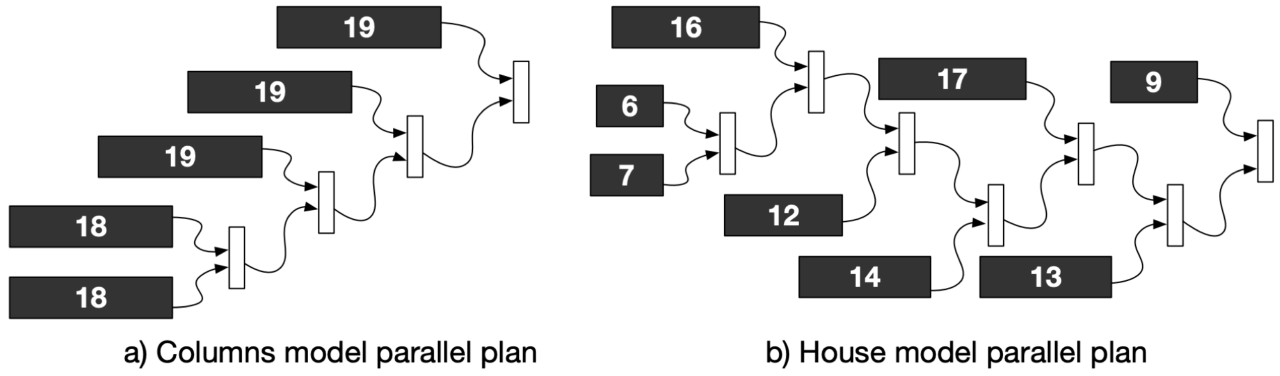
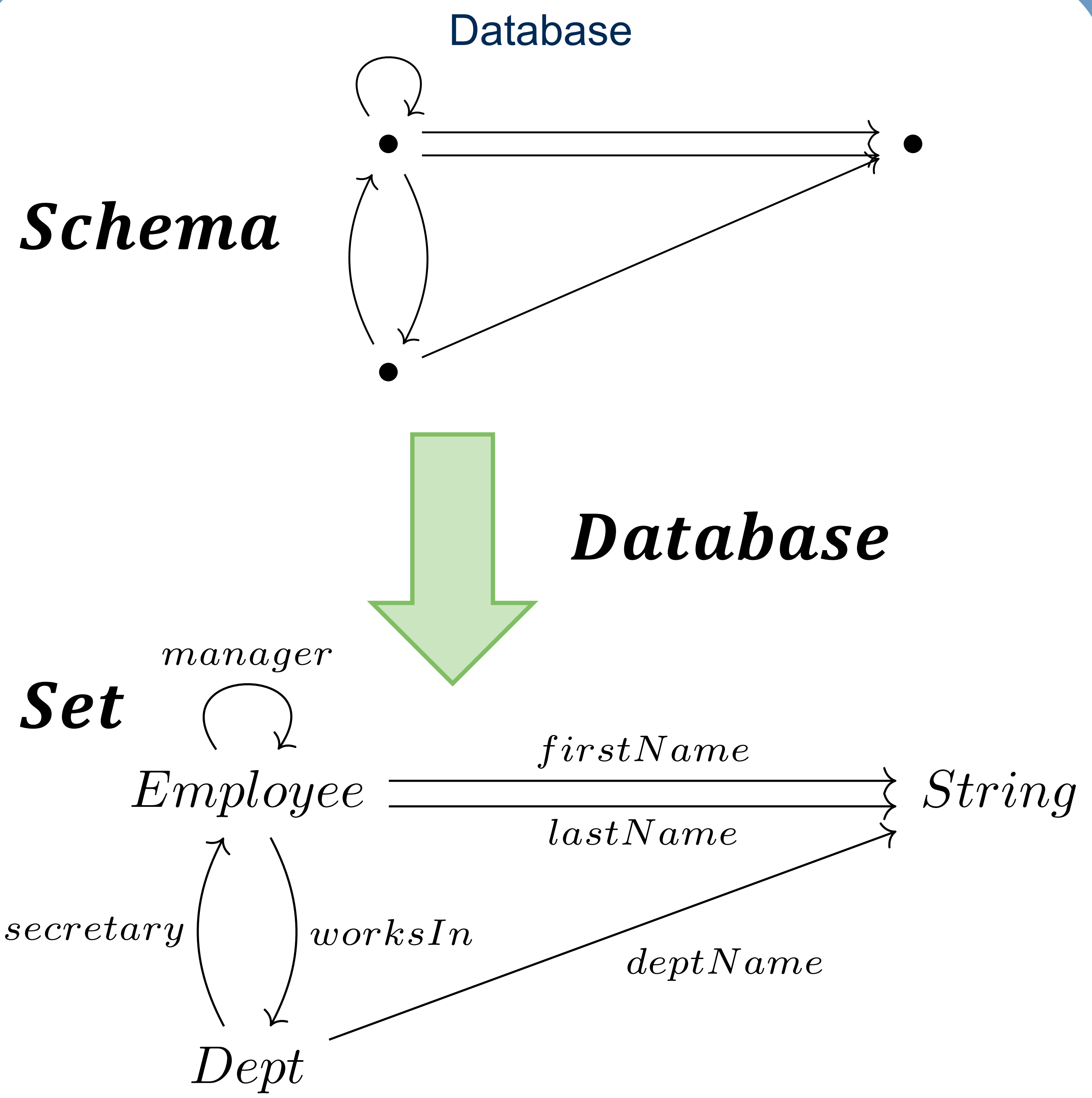


Fig. 5. Parallel plans generated for the two LEGO CAD models. The width of the black boxes represent a sub-plan (i.e., a stairs configuration).

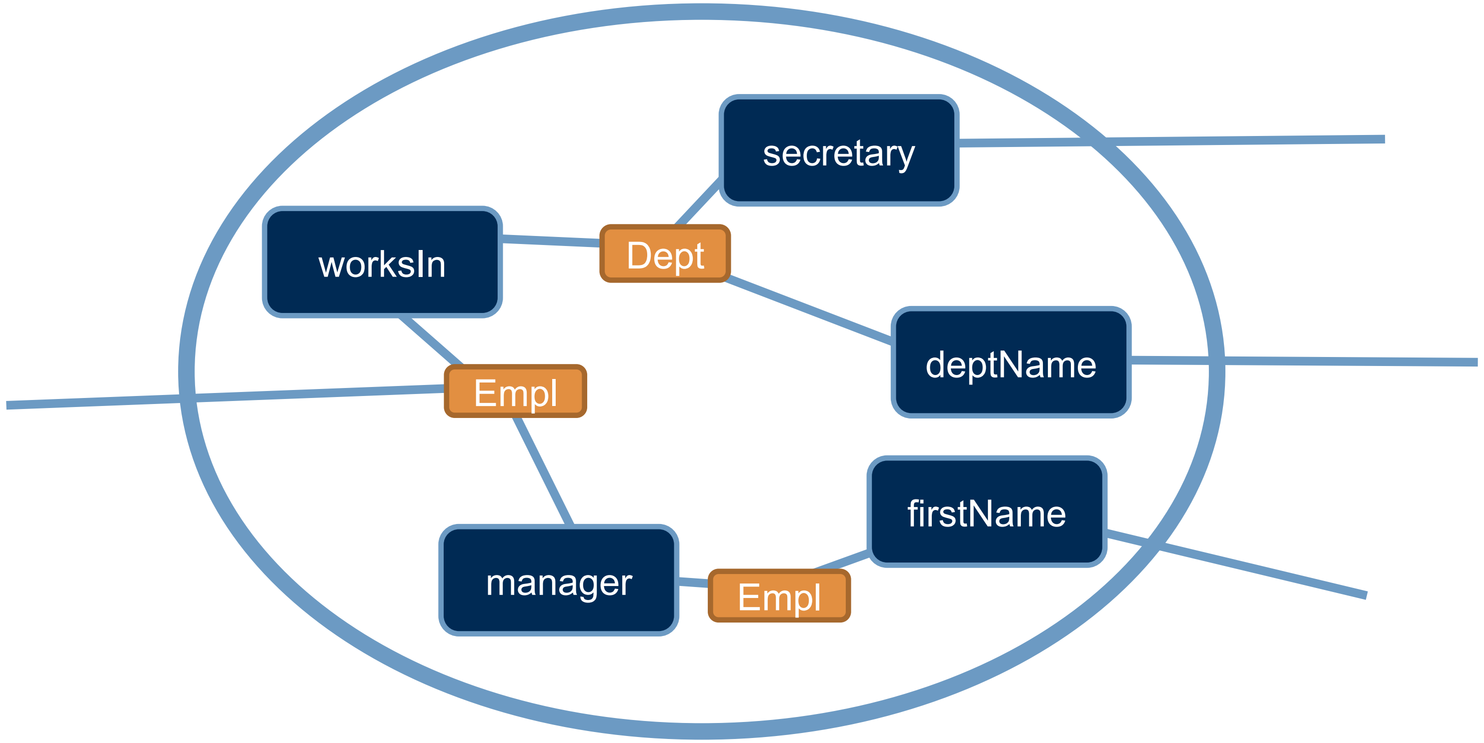
(Master, Patterson, Yousfi & Canedo, 2020)

Generating SQL Queries from *BiCatRel*

AlgebraicJulia



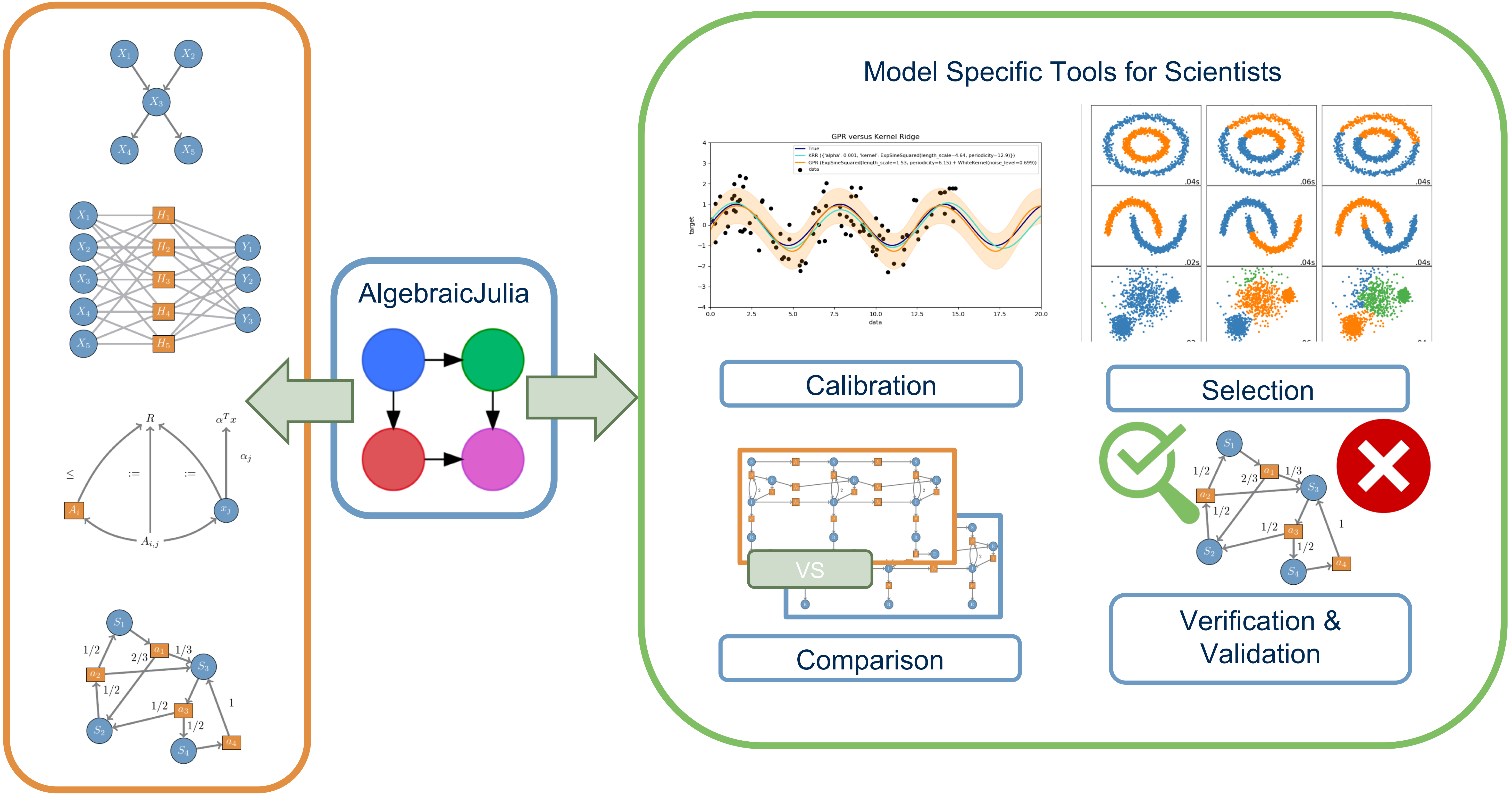
Query




```
SELECT t4.employee AS employee,
       t6.secretary AS secretary, t7.name AS name,
       t9.first_name AS first_name
FROM worksIn AS t4, secretary AS t6, deptName AS t7,
     manager AS t8, firstName AS t9
WHERE t4.employee=t8.employee AND
      t4.department=t6.department AND
      t4.department=t7.department AND
      t8.employee=t9.employee;
```

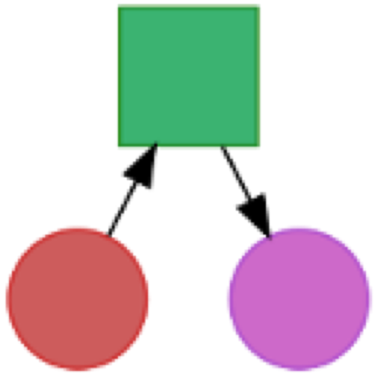
Model Aware Scientific Computing

AlgebraicJulia



AlgebraicJulia Ecosystem

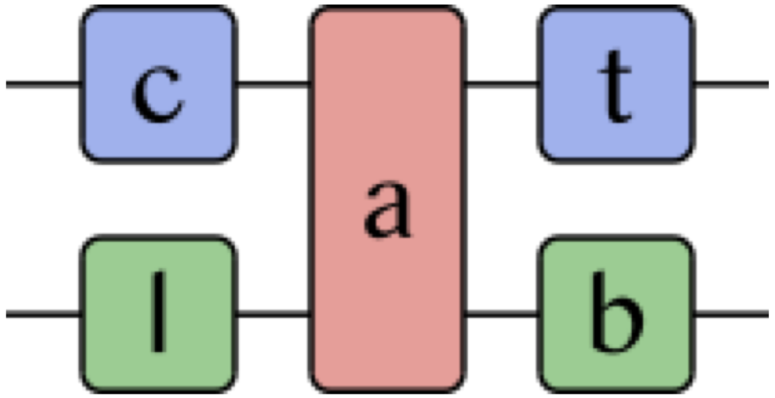
Petri.jl



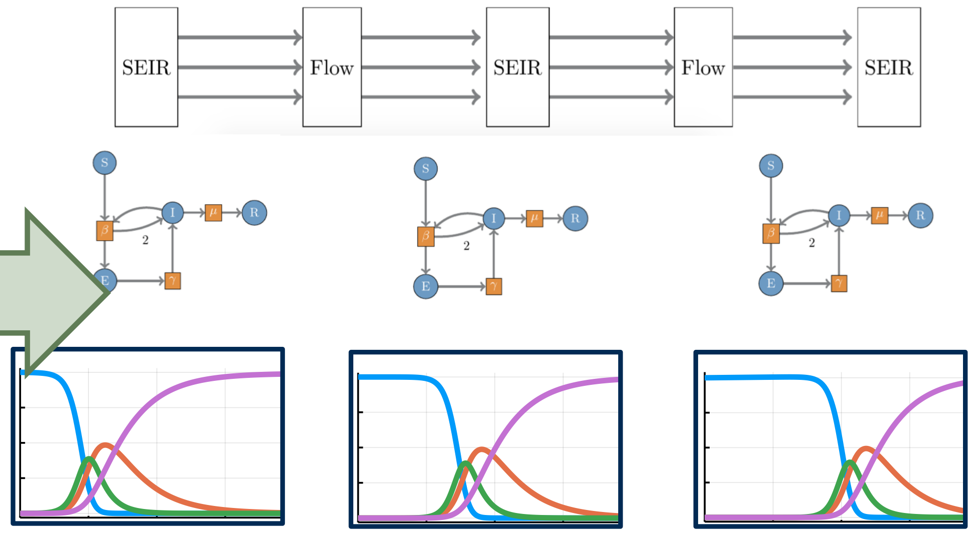
DiffEq.jl

AlgebraicJulia

A, B, C	Types
$a : A, b : B, c : C$	Terms
$a \otimes b \cdot c$	Term Constructors
$(a \otimes b) \cdot (c \otimes d) = (a \cdot c) \otimes (b \cdot d)$	Axioms



Solvers exploiting hierarchical structure



$$
\begin{tikzcd}
 & & C & & \\
 & A \arrow[ur, "f'"] \arrow[r, "\iota_A"] & A \sqcup B \arrow[u, "u"] & B \arrow[ul, "g'"'] \arrow[l, "\iota_B"'] & \\
X \arrow[ur] & & Y \arrow[ul, "f"] \arrow[ur, "g"'] & & Z \arrow[ul]
\end{tikzcd}
$$

Algebraic Julia Team

Evan Patterson

Micah Halter

Sophie Libkind

Andrew Baas