Stratifying an SEIR model by age group using AlgebraicPetri.jl

Simon Frost (@sdwfrost)

2024-07-22

## Introduction

This example serves as an extension to the ‘Hello World’ to stratifying Petri net models; rather than specifying two risk groups manually, we use Catlab’s imperative interface to specify and wire together groups. The idea here is that we can specify a potentially large number of groups (e.g. age classes) programmatically. For simplicity, we will consider no movement between groups i.e. for age classes, no ageing over the course of the epidemic simulation.

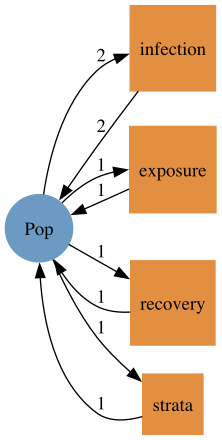
## Libraries

using AlgebraicPetri, AlgebraicPetri.TypedPetri, AlgebraicPetri.BilayerNetworks  
using Catlab, Catlab.CategoricalAlgebra, Catlab.Programs  
using Catlab.WiringDiagrams, Catlab.Graphics  
using AlgebraicDynamics.UWDDynam  
using OrdinaryDiffEq  
using ModelingToolkit  
using LinearAlgebra  
using LabelledArrays  
using Plots  
using Latexify

## Transitions

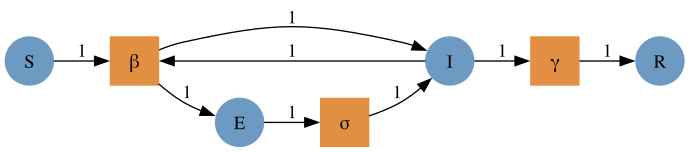
We first define a labelled Petri net that has the different types of transition in our models. The first argument is an array of state names as symbols (here, a generic :Pop), followed by the transitions in the model. Transitions are given as transition\_name=>((input\_states)=>(output\_states)). In this model, we consider the groups as fixed (i.e. no changes between strata), so we just need to have infection and recovery in the model.

epi\_transitions = LabelledPetriNet(  
 [:Pop],  
 :infection=>((:Pop, :Pop)=>(:Pop, :Pop)),  
 :exposure=>(:Pop=>:Pop),  
 :recovery=>(:Pop=>:Pop),  
 :strata=>(:Pop=>:Pop)  
)  
to\_graphviz(epi\_transitions)



We create a labelled Petri net of the SEIR model using the above transitions.

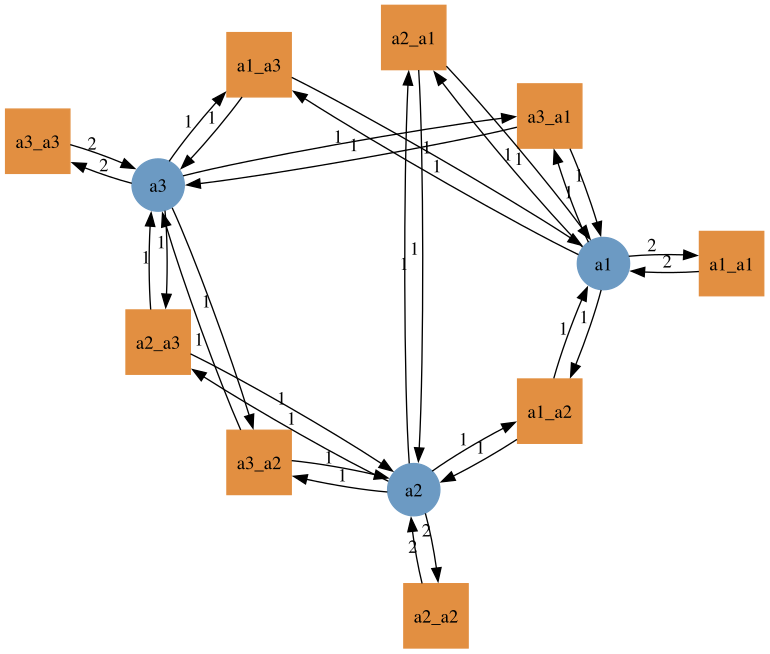
seir\_uwd = @relation () where (S::Pop, E::Pop, I::Pop, R::Pop) begin  
 infection(S, I, E, I)  
 exposure(E, I)  
 recovery(I, R)  
end  
seir\_acst = oapply\_typed(epi\_transitions, seir\_uwd, [:β , :σ, :γ])  
seir\_lpn = dom(seir\_acst) # Extract labelled Petri net  
to\_graphviz(seir\_lpn)



## Age class model

function make\_age\_classes(K, prefix="a")  
 # Start with a blank UWD with K populations  
 uwd = RelationDiagram(repeat([:Pop], K))  
 # Build junctions (just a `Dict`), with the side effect of updating the UWD  
 # `junctions` will be as follows similar to the following  
 # Dict{Symbol, Int64} with n entries:  
 # :a1 => 1  
 # :a2 => 2  
 junctions = Dict(begin  
 variable = Symbol(prefix \* "$(i)")  
 junction = add\_junction!(uwd, :Pop, variable=variable)  
 set\_junction!(uwd, port, junction, outer=true)  
 variable => junction  
 end for (i, port) in enumerate(ports(uwd, outer=true)))  
   
 # This generates all combinations of the keys  
 # Here, this will be a matrix of all pairs of keys  
 pairs = collect(Iterators.product(keys(junctions), keys(junctions)))  
 # This creates an empty vector to store the transition names in  
 tnames = Vector{Symbol}(undef,0)  
 ## Cycle through pairs and add boxes for infection  
 for pair in pairs  
 # We need 4 entries in the tuple as :infection is defined as infection(S, I, I, I)  
 ins\_outs = (pair[1], pair[2], pair[1], pair[2])  
 # In the below, [junction\_type(uwd, junctions[p]) for p in ins\_outs] is just a vector of 4 :Pop  
 box = add\_box!(uwd, [junction\_type(uwd, junctions[p]) for p in ins\_outs], name=:infection)  
 for (rgn, port) in zip(ins\_outs, ports(uwd, box))  
 set\_junction!(uwd, port, junctions[rgn])  
 end  
 # This adds the names (as `Symbol`s) to the vector of transition names  
 push!(tnames,Symbol("$(pair[1])\_$(pair[2])"))  
 end  
 ## Generate an ACSet transformation using the above `epi\_transitions`  
 act = oapply\_typed(epi\_transitions, uwd, tnames)  
 return act  
end;

K = 3  
age\_acst = make\_age\_classes(K)  
age\_lpn = dom(age\_acst)  
to\_graphviz(age\_lpn, prog="circo")



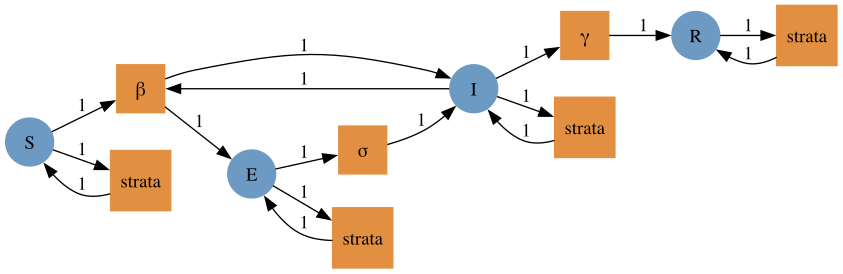
## Composing the models

The age model already has :infection, but not :exposure or :recovery, so we add these in.

seir\_acst\_augmented = add\_reflexives(seir\_acst, repeat([[:strata]], 4), epi\_transitions)  
age\_acst\_augmented = add\_reflexives(age\_acst, repeat([[:exposure, :recovery]], K), epi\_transitions);

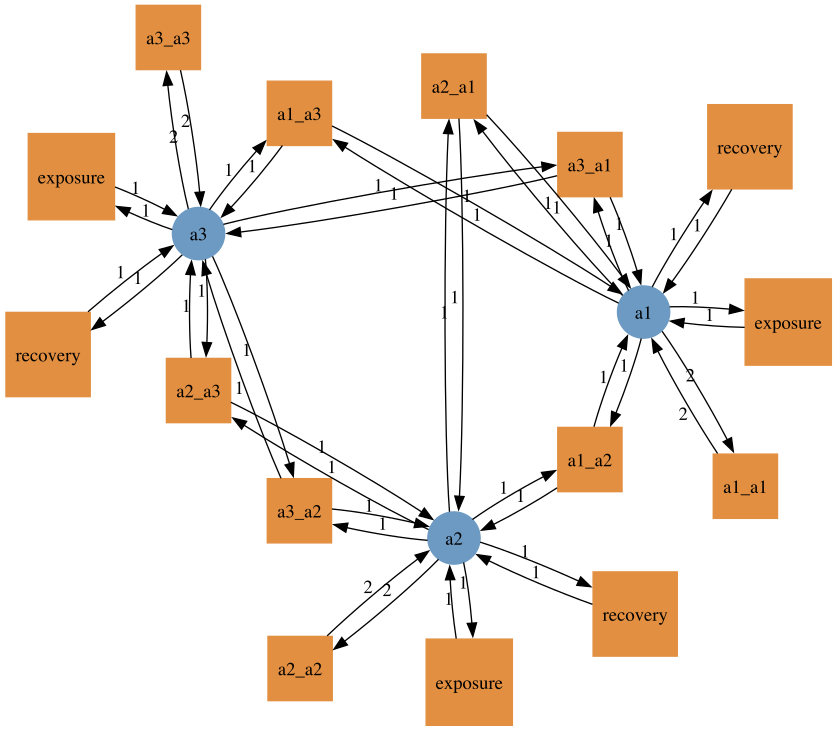
We can now visualize how the disease compartments will be stratified.

to\_graphviz(dom(seir\_acst\_augmented))



We can also visualize the transitions in the age model.

to\_graphviz(dom(age\_acst\_augmented), prog="circo")



We rename the transitions with their group, to avoid the use of the same name in the transitions for different age groups.

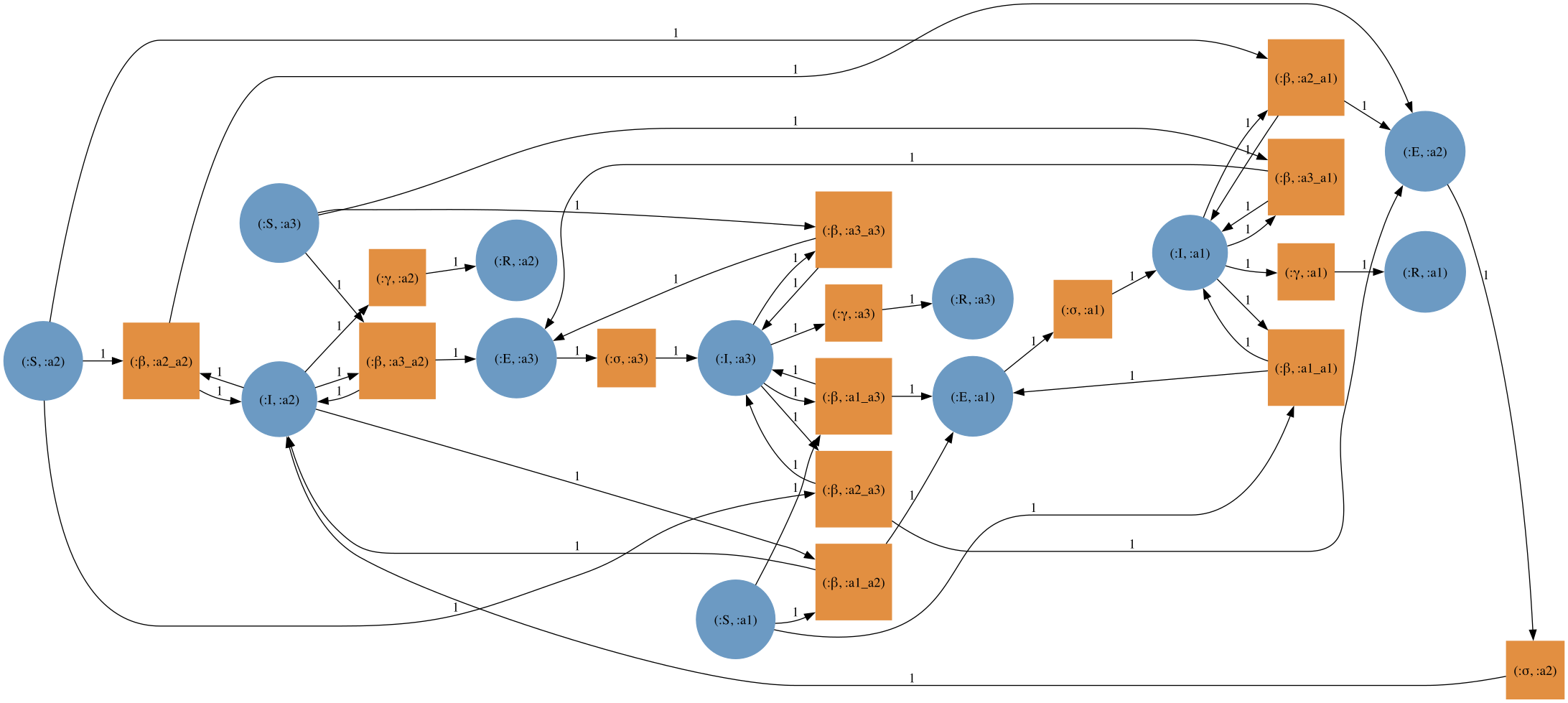
age\_acst\_tnames = dom(age\_acst)[:tname]  
age\_acst\_snames = dom(age\_acst)[:sname]  
for s in age\_acst\_snames  
 for k in 1:2  
 push!(age\_acst\_tnames,s)  
 end  
end  
dom(age\_acst\_augmented)[:tname] = age\_acst\_tnames;

We compose the models using typed\_product.

seir\_age\_acst = typed\_product(seir\_acst\_augmented, age\_acst\_augmented)  
seir\_age\_lpn = dom(seir\_age\_acst);

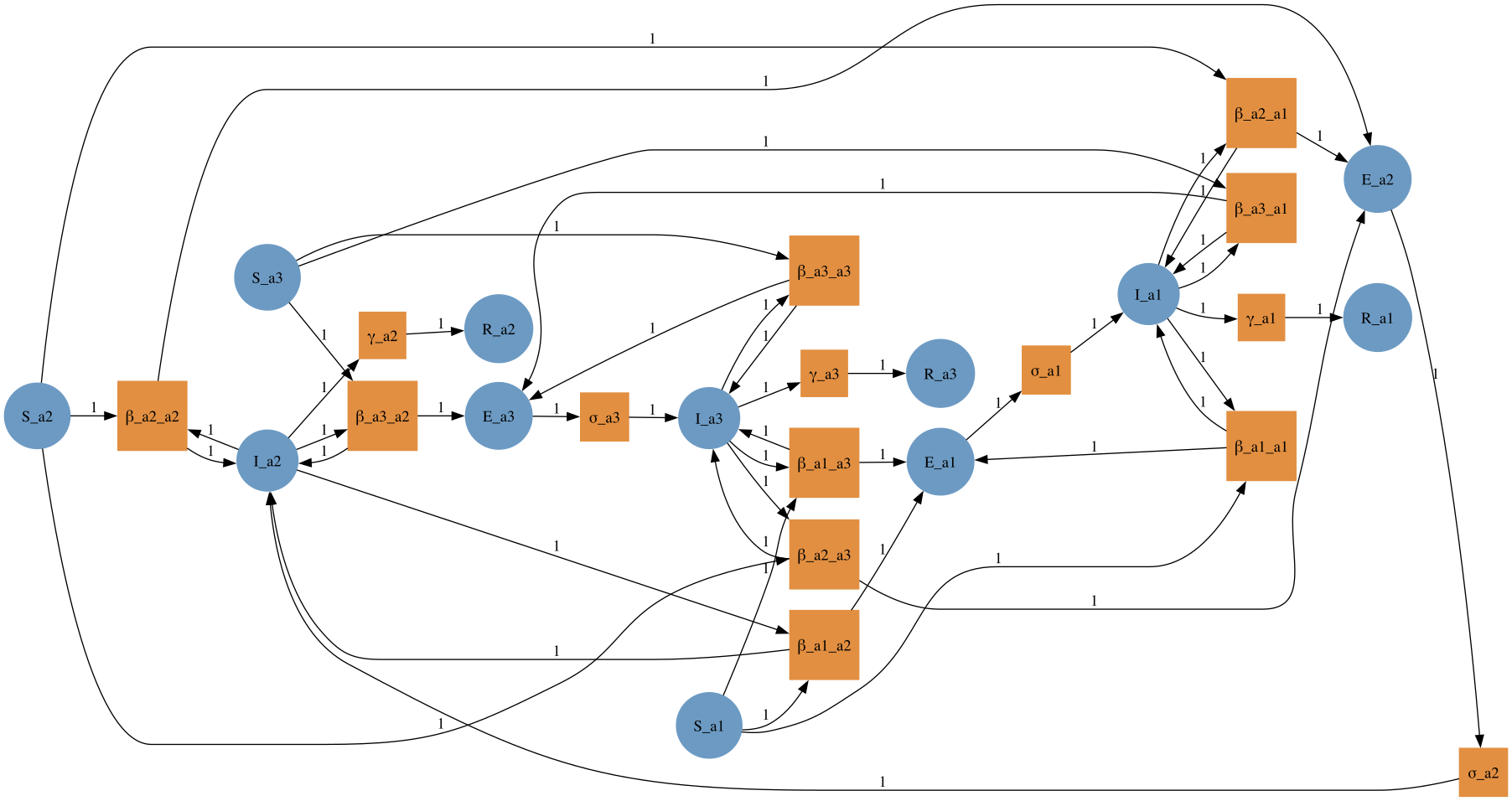
Note that for the transmission terms, the first index refers to the susceptible group, and the second to the infected group.

to\_graphviz(seir\_age\_lpn)



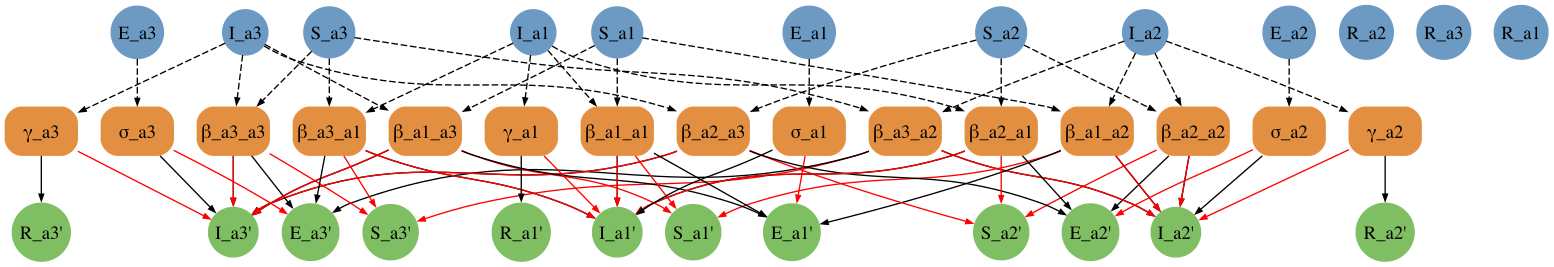
We flatten the labels to allow the model to be lowered into an ODE.

seir\_age\_lpn\_flatlabels = flatten\_labels(seir\_age\_lpn)  
to\_graphviz(seir\_age\_lpn\_flatlabels, prog="dot")



The transitions may be easier to inspect graphically if we represent the model as a bilayer network.

seir\_age\_bn = LabelledBilayerNetwork()  
migrate!(seir\_age\_bn, seir\_age\_lpn\_flatlabels)  
to\_graphviz(seir\_age\_bn)



We can also retrieve the equations from the bilayer network.

latexify(ModelingToolkit.equations(ODESystem(seir\_age\_bn)))

## Running the model

To run the model, we need to choose specific group sizes and parameter values. Note that the ordering of the states and parameters is not lexographic.

snames(seir\_age\_lpn\_flatlabels)

12-element Vector{Symbol}:  
 :S\_a2  
 :I\_a2  
 :E\_a2  
 :R\_a2  
 :S\_a3  
 :I\_a3  
 :E\_a3  
 :R\_a3  
 :S\_a1  
 :I\_a1  
 :E\_a1  
 :R\_a1

tnames(seir\_age\_lpn\_flatlabels)

15-element Vector{Symbol}:  
 :β\_a2\_a2  
 :β\_a3\_a2  
 :β\_a1\_a2  
 :β\_a2\_a3  
 :β\_a3\_a3  
 :β\_a1\_a3  
 :β\_a2\_a1  
 :β\_a3\_a1  
 :β\_a1\_a1  
 :σ\_a2  
 :γ\_a2  
 :σ\_a3  
 :γ\_a3  
 :σ\_a1  
 :γ\_a1

seir\_states = ["S", "E", "I", "R"]  
age\_states = ["a" \* string(i) for i in 1:K]  
pop\_names = permutedims(hcat(repeat([seir\_states], K)...)) .\* "\_" .\* hcat(repeat([age\_states], 4)...)  
N = [14799290, 16526302, 28961159]  
i₀ = 1e-6  
inits = [1.0-i₀, 0.0, i₀, 0.0]  
ukpop = hcat(repeat([N], 4)...)  
u0\_vector = vec(hcat(repeat([inits], K)...)' .\* ukpop)  
u0\_names = vec(pop\_names)  
cm\_orig = hcat([[7.883663, 2.794154, 1.565665],  
 [3.120220, 4.854839, 2.624868],  
 [3.063895, 4.599893, 5.005571]]...)'  
cm\_norm = cm\_orig/maximum(eigvals(cm\_orig))  
cm = cm\_norm ./ N  
β = 1.3/7  
σ = 1.0/2  
γ = 1.0/7  
β\_matrix = hcat([[β\*cm[i,j] for i in 1:K] for j in 1:K]...)  
β\_names = hcat([["β\_" \* age\_states[i] \* "\_" \* age\_states[j] for i in 1:K] for j in 1:K]...)  
σ\_vector = repeat([σ], K)  
σ\_names = ["σ\_" \* a for a in age\_states]   
γ\_vector = repeat([γ], K)  
γ\_names = ["γ\_" \* a for a in age\_states]  
p\_vector = [vec(β\_matrix); σ\_vector; γ\_vector]  
p\_names = [vec(β\_names); σ\_names; γ\_names]

15-element Vector{String}:  
 "β\_a1\_a1"  
 "β\_a2\_a1"  
 "β\_a3\_a1"  
 "β\_a1\_a2"  
 "β\_a2\_a2"  
 "β\_a3\_a2"  
 "β\_a1\_a3"  
 "β\_a2\_a3"  
 "β\_a3\_a3"  
 "σ\_a1"  
 "σ\_a2"  
 "σ\_a3"  
 "γ\_a1"  
 "γ\_a2"  
 "γ\_a3"

u0 = @LArray u0\_vector Tuple(Symbol.(u0\_names))  
p = @LArray p\_vector Tuple(Symbol.(p\_names))  
tspan = (0.0, 600.0);

We then compute the vector field from the labelled Petri net (with flatten labels), define the ODEProblem, and solve.

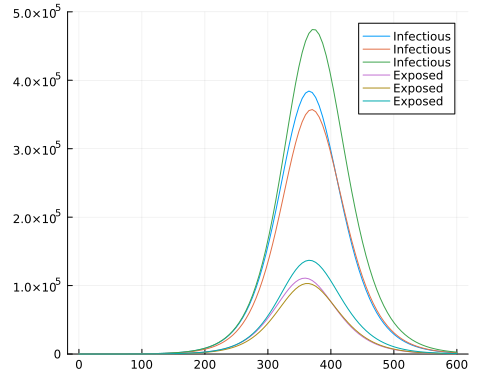
seir\_age\_vf = vectorfield(seir\_age\_lpn\_flatlabels)  
seir\_age\_prob = ODEProblem(seir\_age\_vf, u0, tspan, p)

ODEProblem with uType LArray{Float64, 1, Vector{Float64}, (:S\_a1, :S\_a2, :S\_a3, :E\_a1, :E\_a2, :E\_a3, :I\_a1, :I\_a2, :I\_a3, :R\_a1, :R\_a2, :R\_a3)} and tType Float64. In-place: true  
timespan: (0.0, 600.0)  
u0: 12-element LArray{Float64, 1, Vector{Float64}, (:S\_a1, :S\_a2, :S\_a3, :E\_a1, :E\_a2, :E\_a3, :I\_a1, :I\_a2, :I\_a3, :R\_a1, :R\_a2, :R\_a3)}:  
 :S\_a1 => 1.479927520071e7  
 :S\_a2 => 1.6526285473698e7  
 :S\_a3 => 2.8961130038840998e7  
 :E\_a1 => 0.0  
 :E\_a2 => 0.0  
 :E\_a3 => 0.0  
 :I\_a1 => 14.79929  
 :I\_a2 => 16.526301999999998  
 :I\_a3 => 28.961159  
 :R\_a1 => 0.0  
 :R\_a2 => 0.0  
 :R\_a3 => 0.0

seir\_age\_sol = solve(seir\_age\_prob, Tsit5());

## Processing the output

t = seir\_age\_sol.t  
E\_out = permutedims(hcat(seir\_age\_sol[[:E\_a1, :E\_a2, :E\_a3]]...))  
I\_out = permutedims(hcat(seir\_age\_sol[[:I\_a1, :I\_a2, :I\_a3]]...))  
plot(t, I\_out, label="Infectious", ylim=(0,500000))  
plot!(t, E\_out, label="Exposed")

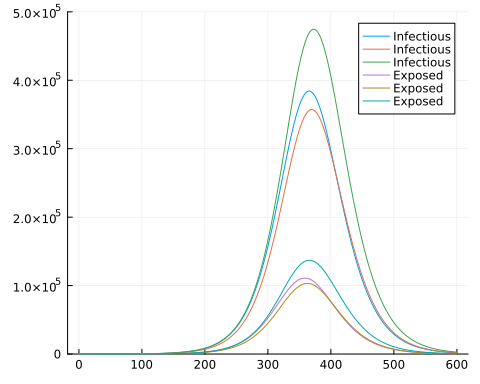


## Comparison to Epidemics.jl

using Epidemics

population = Population("UK",  
 N,  
 permutedims(hcat(repeat([[1.0-i₀, 0.0, i₀, 0.0, 0.0]],K)...)),  
 cm\_orig)  
out = epidemic\_default(β=[β],  
 σ=[σ],  
 γ=[γ],  
 population=population,  
 intervention=nothing,  
 vaccination=nothing,  
 time\_end = 600.0,  
 increment = 1.0);

epidemics\_sol = out[1]  
t = epidemics\_sol.t  
u = epidemics\_sol.u[1:length(epidemics\_sol)]  
E\_out = permutedims(hcat([x[4:6] for x in u]...))  
I\_out = permutedims(hcat([x[7:9] for x in u]...))  
plot(t, I\_out, label="Infectious", ylim=(0,500000))  
plot!(t, E\_out, label="Exposed")



## Comparison to Epiverse epidemics package

using RCall  
using DataFrames  
using Query  
using StatsPlots

R"""  
library(epidemics)  
# load contact and population data from socialmixr::polymod  
polymod <- socialmixr::polymod  
contact\_data <- socialmixr::contact\_matrix(  
 polymod,  
 countries = "United Kingdom",  
 age.limits = c(0, 20, 40),  
 symmetric = TRUE  
)  
  
# prepare contact matrix  
contact\_matrix <- t(contact\_data$matrix)  
  
# prepare the demography vector  
demography\_vector <- contact\_data$demography$population  
names(demography\_vector) <- rownames(contact\_matrix)  
  
# initial conditions: one in every 1 million is infected  
initial\_i <- 1e-6  
initial\_conditions <- c(  
 S = 1 - initial\_i, E = 0, I = initial\_i, R = 0, V = 0  
)  
  
# build for all age groups  
initial\_conditions <- rbind(  
 initial\_conditions,  
 initial\_conditions,  
 initial\_conditions  
)  
rownames(initial\_conditions) <- rownames(contact\_matrix)  
  
uk\_population <- population(  
 name = "UK",  
 contact\_matrix = contact\_matrix,  
 demography\_vector = demography\_vector,  
 initial\_conditions = initial\_conditions  
)  
  
epidemics\_sol <- model\_default(  
 population=uk\_population,  
 transmission\_rate = 1.3/7,  
 infectiousness\_rate = 1/2,  
 recovery\_rate = 1/7,  
 intervention = NULL,  
 vaccination = NULL,  
 time\_dependence = NULL,  
 time\_end = 600,  
 increment = 1.0  
)  
""";

┌ Warning: RCall.jl: Using POLYMOD social contact data. To cite this in a publication, use the 'cite' function  
│ Removing participants that have contacts without age information. To change this behaviour, set the 'missing.contact.age' option  
└ @ RCall ~/.julia/packages/RCall/FEbLj/src/io.jl:172

@rget epidemics\_sol;

epidemics\_sol |>  
 @filter(\_.compartment in ["exposed", "infectious"]) |>  
 @df plot(:time,  
 :value,  
 group=(:demography\_group, :compartment),  
 ylim=(0,500000),  
 xlabel="Time",  
 ylabel="Number")

