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.

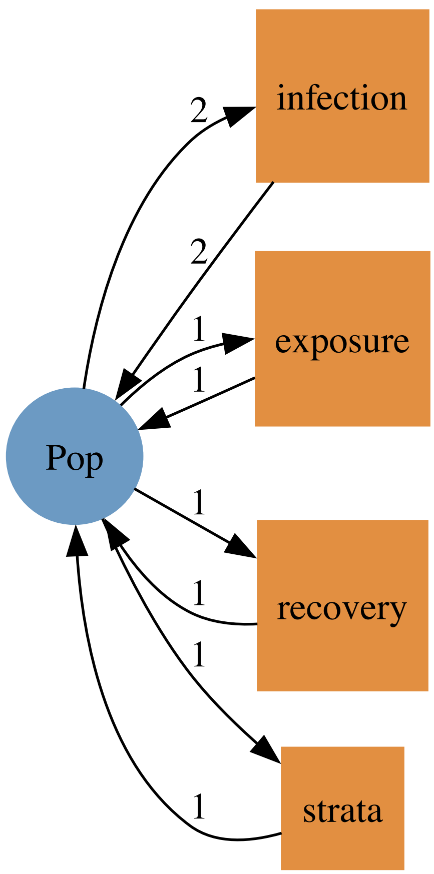
Libraries

```
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.

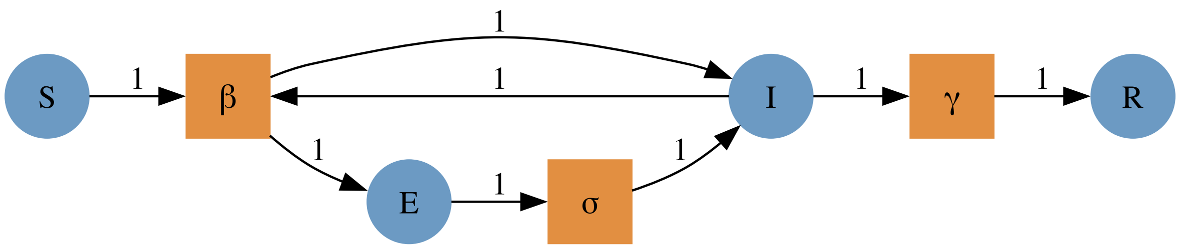
```
epi_transitions = LabelledPetriNet(
  [:Pop],
  :infection=>((:Pop, :Pop)=>(:Pop, :Pop)),
  :exposure=>(:Pop=>:Pop),
  :recovery=>(:Pop=>:Pop),
  :strata=>(:Pop=>:Pop)
)
to_graphviz(epi_transitions)
```



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, [:  , : , :])
seir_lpn = dom(seir_acst) # Extract labelled Petri net
to_graphviz(seir_lpn)
```



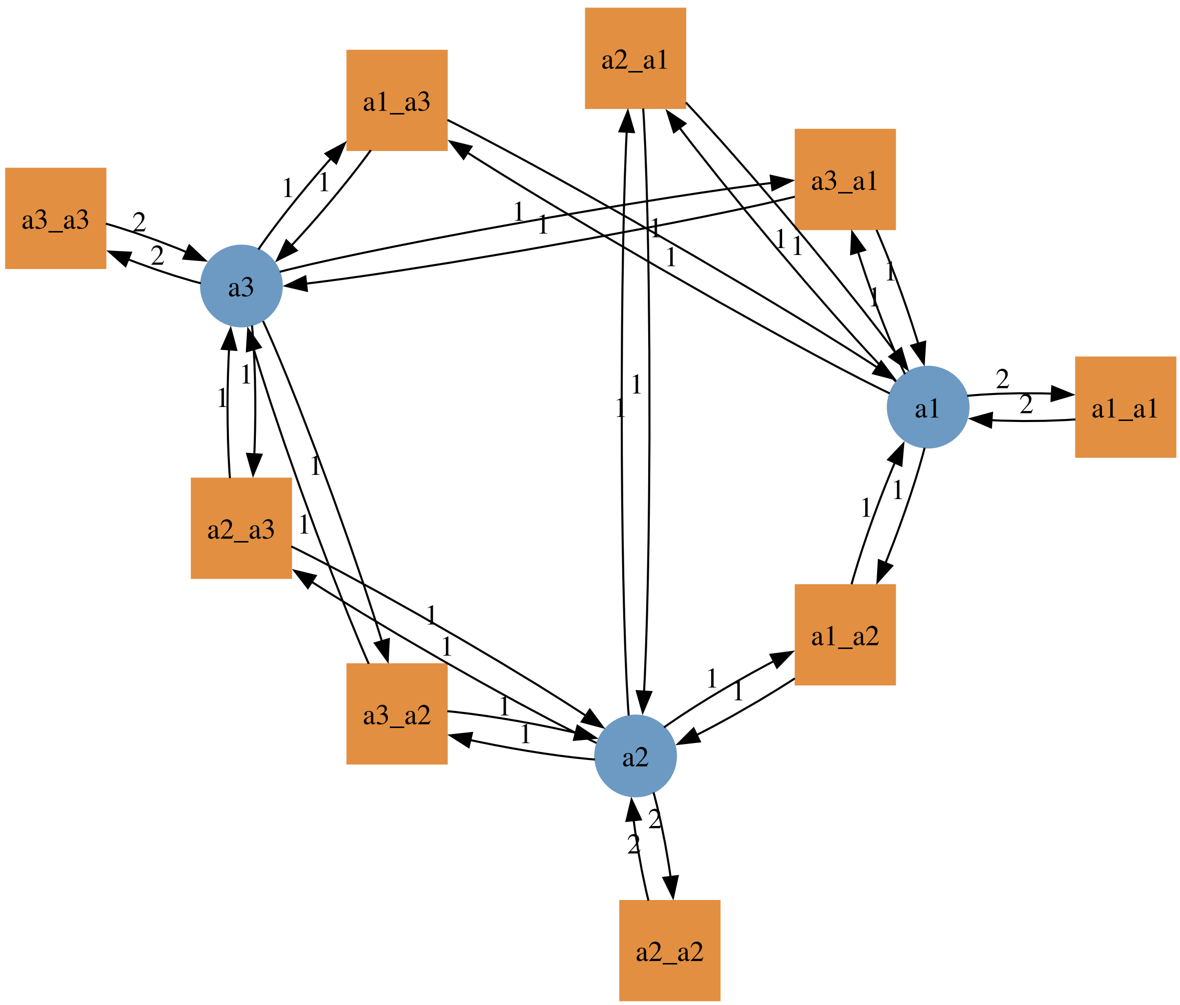
Age class model

```
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 vecto
        box = add_box!(uwd, [junction_type(uwd, junctions[p]) for p in ins_outs], name=:infe
        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")
```



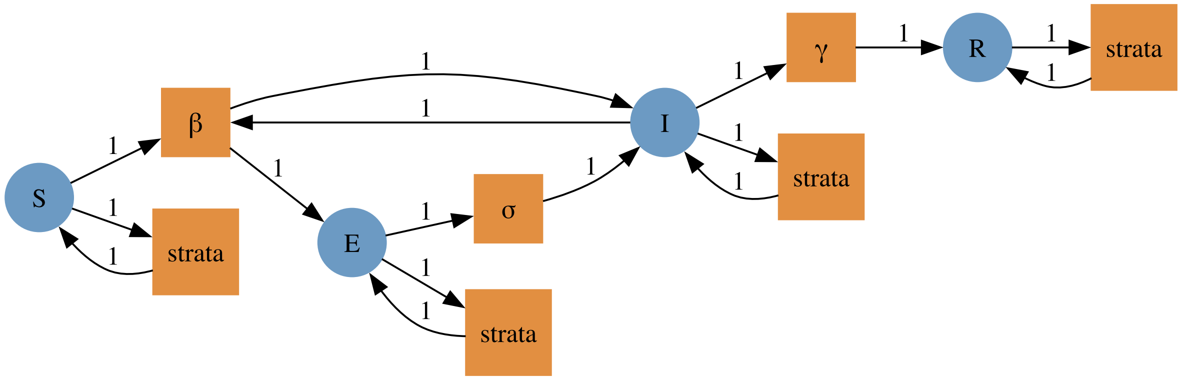
Composing the models

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_trans
```

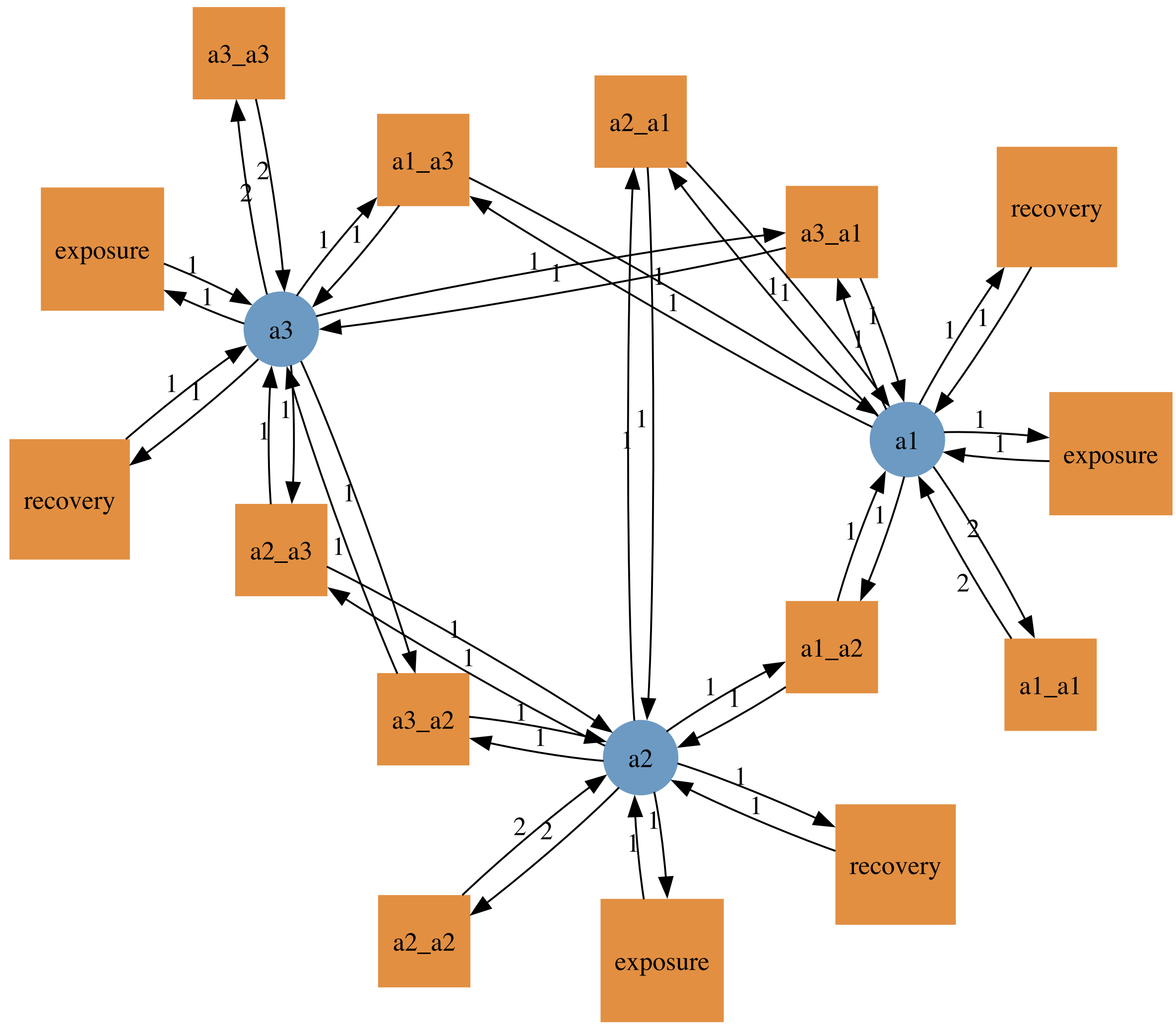
We can now visualize how the disease compartments will be stratified.

```
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.

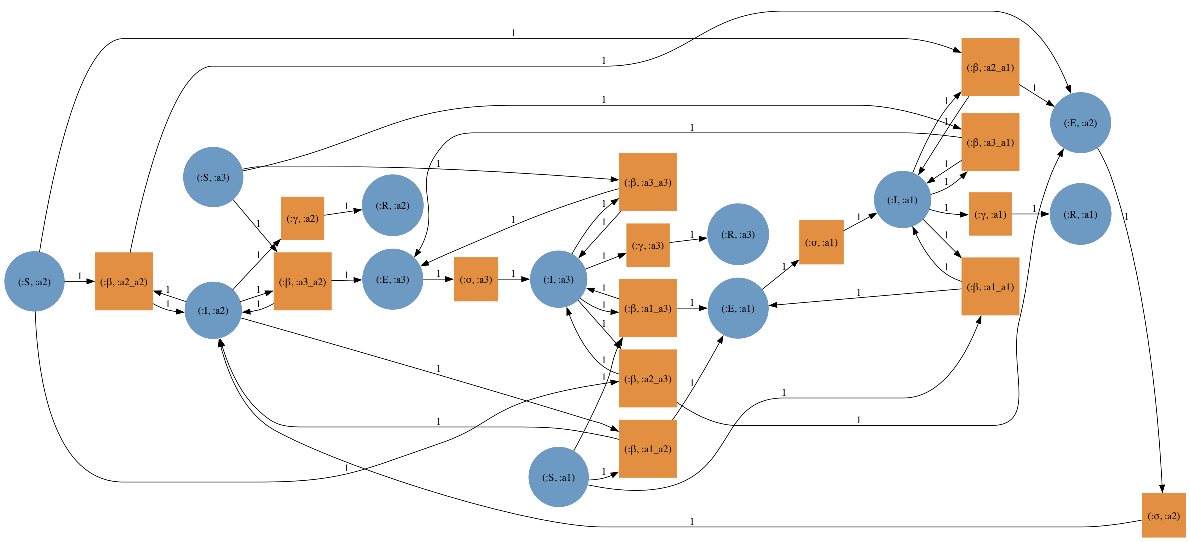
```
age_acst_tnames = dom(age_acst)[:tname]
age_acst_snames = dom(age_acst)[:sname]
for s in age_acst_snames
    for k in 1:2
        push!(age_acst_tnames,s)
    end
end
dom(age_acst_augmented)[:tname] = age_acst_tnames;
```

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);
```

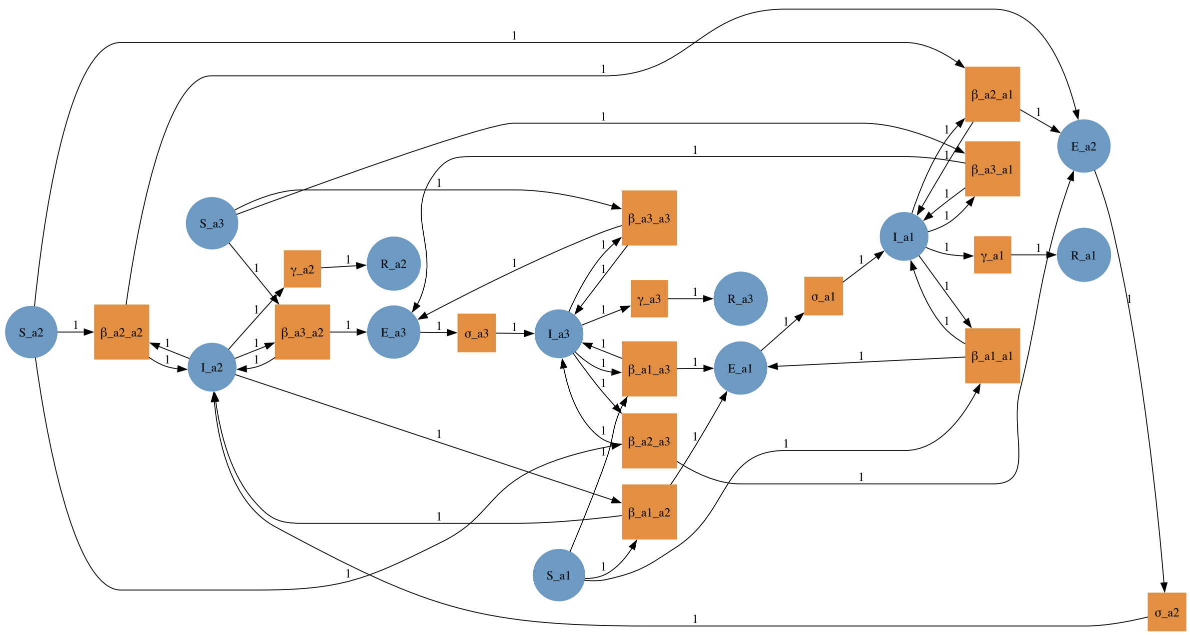
Note that for the transmission terms, the first index refers to the susceptible group, and the second to the infected group.

```
to_graphviz(seir_age_lpn)
```



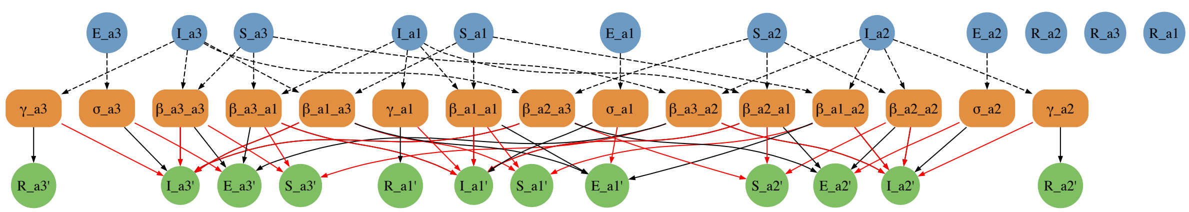
We flatten the labels to allow the model to be lowered into an ODE.

```
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)))
```

$$\frac{\mathrm{d}S_{a2}\left(t \right)}{\mathrm{d}t} = - \left(I_{a2}\left(t \right) S_{a2}\left(t \right) \beta_{a2\_a2} + S_{a2}\left(t \right) I_{a1}\left(t \right) \beta_{a2\_a1} + S_{a2}\left(t \right) I_{a3}\left(t \right) \beta_{a2\_a3} \right) \tag{1}$$

$$\frac{\mathrm{d}I_{a2}\left(t \right)}{\mathrm{d}t} = E_{a2}\left(t \right) \sigma_{a2} + S_{a3}\left(t \right) I_{a2}\left(t \right) \beta_{a3\_a2} + S_{a1}\left(t \right) I_{a2}\left(t \right) \beta_{a1\_a2} + I_{a2}\left(t \right) S_{a2}\left(t \right) \beta_{a2\_a2} - \left(I_{a2}\left(t \right) \gamma_{a2} + S_{a3} (\tag{2}$$

$$\frac{\mathrm{d}E_{a2}\left(t \right)}{\mathrm{d}t} = I_{a2}\left(t \right) S_{a2}\left(t \right) \beta_{a2\_a2} + S_{a2}\left(t \right) I_{a1}\left(t \right) \beta_{a2\_a1} + S_{a2}\left(t \right) I_{a3}\left(t \right) \beta_{a2\_a3} - E_{a2}\left(t \right) \sigma_{a2} \tag{3}$$

$$\frac{\mathrm{d}R_{a2}\left(t \right)}{\mathrm{d}t} = I_{a2}\left(t \right) \gamma_{a2} \tag{4}$$

$$\frac{\mathrm{d}S_{a3}\left(t \right)}{\mathrm{d}t} = - \left(S_{a3}\left(t \right) I_{a2}\left(t \right) \beta_{a3\_a2} + S_{a3}\left(t \right) I_{a1}\left(t \right) \beta_{a3\_a1} + S_{a3}\left(t \right) I_{a3}\left(t \right) \beta_{a3\_a3} \right) \tag{5}$$

$$\frac{\mathrm{d}I_{a3}\left(t \right)}{\mathrm{d}t} = E_{a3}\left(t \right) \sigma_{a3} + S_{a3}\left(t \right) I_{a3}\left(t \right) \beta_{a3\_a3} + S_{a1}\left(t \right) I_{a3}\left(t \right) \beta_{a1\_a3} + S_{a2}\left(t \right) I_{a3}\left(t \right) \beta_{a2\_a3} - \left(I_{a3}\left(t \right) \gamma_{a3} + S_{a3} (\tag{6}$$

$$\frac{\mathrm{d}E_{a3}\left(t \right)}{\mathrm{d}t} = S_{a3}\left(t \right) I_{a2}\left(t \right) \beta_{a3\_a2} + S_{a3}\left(t \right) I_{a1}\left(t \right) \beta_{a3\_a1} + S_{a3}\left(t \right) I_{a3}\left(t \right) \beta_{a3\_a3} - E_{a3}\left(t \right) \sigma_{a3} \tag{7}$$

$$\frac{\mathrm{d}R_{a3}\left(t \right)}{\mathrm{d}t} = I_{a3}\left(t \right) \gamma_{a3} \tag{8}$$

$$\frac{\mathrm{d}S_{a1}\left(t \right)}{\mathrm{d}t} = - \left(S_{a1}\left(t \right) I_{a2}\left(t \right) \beta_{a1\_a2} + S_{a1}\left(t \right) I_{a1}\left(t \right) \beta_{a1\_a1} + S_{a1}\left(t \right) I_{a3}\left(t \right) \beta_{a1\_a3} \right) \tag{9}$$

$$\frac{\mathrm{d}I_{a1}\left(t \right)}{\mathrm{d}t} = E_{a1}\left(t \right) \sigma_{a1} + S_{a3}\left(t \right) I_{a1}\left(t \right) \beta_{a3\_a1} + S_{a1}\left(t \right) I_{a1}\left(t \right) \beta_{a1\_a1} + S_{a2}\left(t \right) I_{a1}\left(t \right) \beta_{a2\_a1} - \left(I_{a1}\left(t \right) \gamma_{a1} + S_{a3} (\tag{10}$$

$$\frac{\mathrm{d}E_{a1}\left(t \right)}{\mathrm{d}t} = S_{a1}\left(t \right) I_{a2}\left(t \right) \beta_{a1\_a2} + S_{a1}\left(t \right) I_{a1}\left(t \right) \beta_{a1\_a1} + S_{a1}\left(t \right) I_{a3}\left(t \right) \beta_{a1\_a3} - E_{a1}\left(t \right) \sigma_{a1} \tag{11}$$

$$\frac{\mathrm{d}R_{a1}\left(t \right)}{\mathrm{d}t} = I_{a1}\left(t \right) \gamma_{a1} \tag{12}$$

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}:
 : _a2_a2
 : _a3_a2
 : _a1_a2
 : _a2_a3
 : _a3_a3
 : _a1_a3
 : _a2_a1
 : _a3_a1
 : _a1_a1
 : _a2
 : _a2
 : _a3
 : _a3
 : _a1
 : _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]
N = [14799290, 16526302, 28961159]
i  = 1e-6
inits = [1.0-i , 0.0, i , 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
 = 1.3/7
 = 1.0/2
 = 1.0/7
 _matrix = hcat([[ *cm[i,j] for i in 1:K] for j in 1:K]...)
 _names = hcat([[" _" * age_states[i] * "_" * age_states[j] for i in 1:K] for j in 1:K]...)
 _vector = repeat([], K)
 _names = [" _" * a for a in age_states]
 _vector = repeat([], K)
 _names = [" _" * a for a in age_states]
p_vector = [vec( _matrix); _vector; _vector]
p_names = [vec( _names); _names; _names]
```

```
15-element Vector{String}:
 " _a1_a1"
 " _a2_a1"
 " _a3_a1"
 " _a1_a2"
 " _a2_a2"
 " _a3_a2"
 " _a1_a3"
 " _a2_a3"
 " _a3_a3"
 " _a1"
 " _a2"
 " _a3"
 " _a1"
 " _a2"
 " _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.

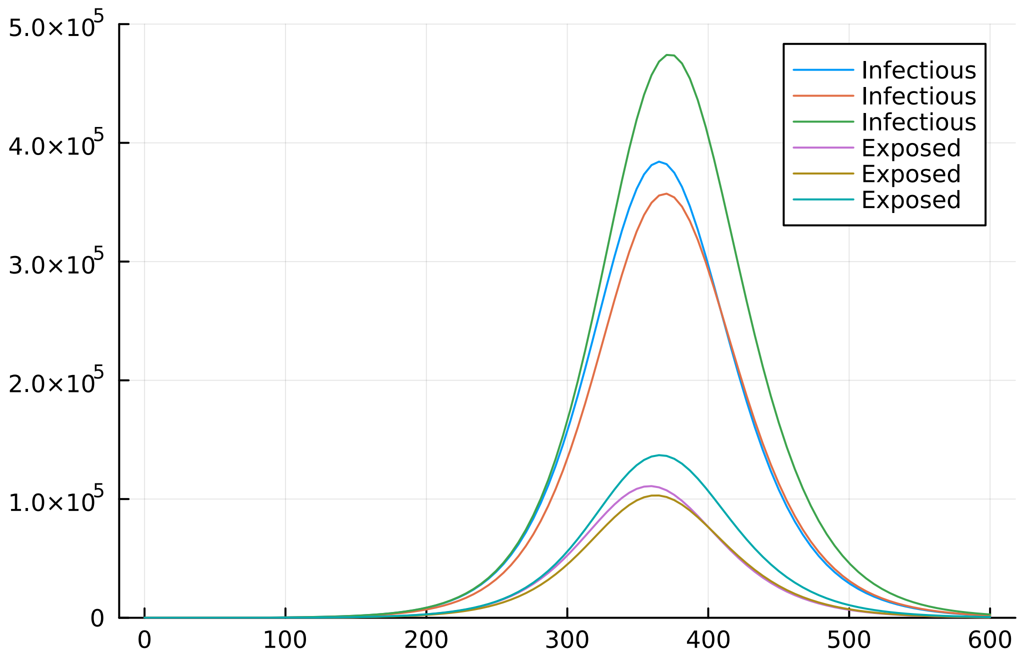
```
seir_age_vf = vectorfield(seir_age_lpn_flatlabels)
seir_age_prob = ODEProblem(seir_age_vf, u0, tspan, p)
```

```
ODEProblem with uType LArray{Float64, 1, Vector{Float64}, (:S_a1, :S_a2, :S_a3, :E_a1, :E_a2
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
 :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")
```



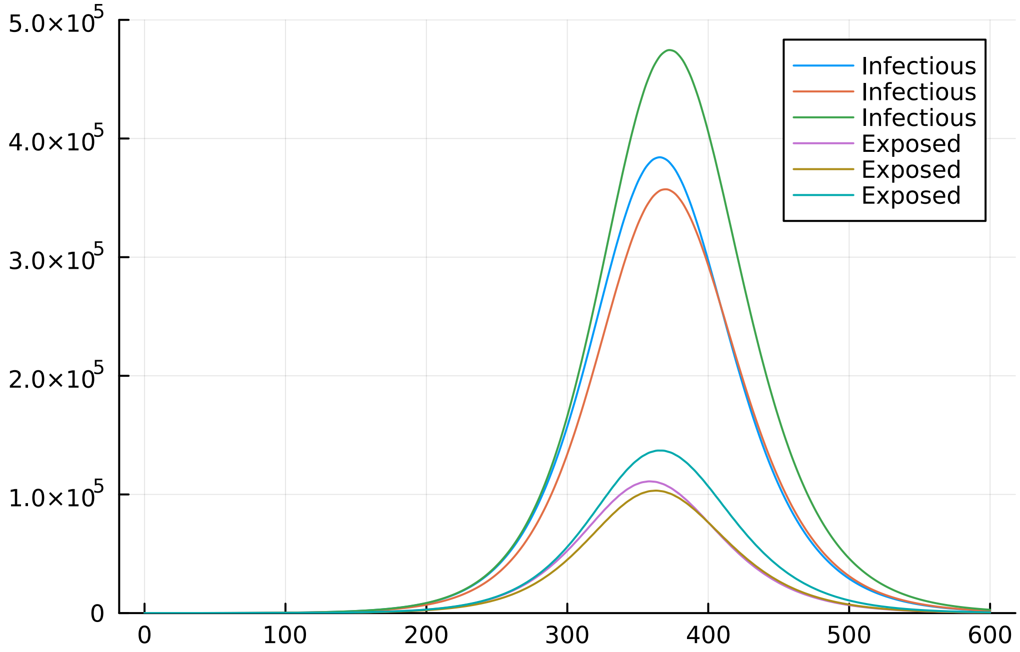
Comparison to Epidemics.jl

```
using Epidemics
```

```
population = Population("UK",
                         N,
                         permutedims(hcat(repeat([[1.0-i , 0.0, i , 0.0, 0.0]],K)...)),
                         cm_orig)
out = epidemic_default( =[],
                        =[],
                        =[],
                       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 th
Removing participants that have contacts without age information. To change this behaviour,
@ 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")
```