Stratifying an SIR model by two risk groups using AlgebraicPetri.jl

Simon Frost (@sdwfrost)

2023-06-13

## Introduction

This example serves as a ‘Hello World’ to stratifying Petri net models, where an SIR model is stratified by another model - in this case, two risk groups, H and L, with high and low contact rates respectively. The stratification is done using two manually specified models. At the end, we use Catlab’s imperative interface to construct the risk group model programmatically; while this is more useful for models with many groups, it serves as a good test case.

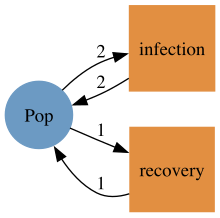
## Libraries

using AlgebraicPetri,AlgebraicPetri.TypedPetri  
using Catlab, Catlab.CategoricalAlgebra, Catlab.Programs  
using Catlab.WiringDiagrams, Catlab.Graphics  
using AlgebraicDynamics.UWDDynam  
using OrdinaryDiffEq  
using LabelledArrays  
using Plots

## 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)),  
 :recovery=>(:Pop=>:Pop)  
)  
to\_graphviz(epi\_transitions)



We create a labelled Petri net of the SIR model using the above transitions (or alternatively, we could compose from infection and recovery submodels).

sir\_uwd = @relation () where (S::Pop, I::Pop, R::Pop) begin  
 infection(S, I, I, I)  
 recovery(I, R)  
end  
sir\_acst = oapply\_typed(epi\_transitions, sir\_uwd, [:β , :γ])  
sir\_lpn = dom(sir\_acst)  
to\_graphviz(sir\_lpn)



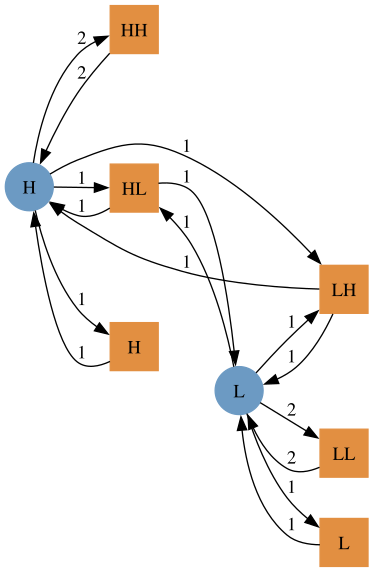
We then define a second model with two groups with different contact rates. This also has infection and recovery terms defined in terms of within- and between-group interactions.

risk\_uwd = @relation () where (H::Pop, L::Pop) begin  
 infection(H,H,H,H) # Within H infection  
 infection(H,L,H,L) # Infection of S\_H by I\_L  
 infection(L,H,L,H) # Infection of S\_L by I\_H  
 infection(L,L,L,L) # Within L infection  
end  
risk\_acst = oapply\_typed(epi\_transitions, risk\_uwd, [:HH, :HL, :LH, :LL])

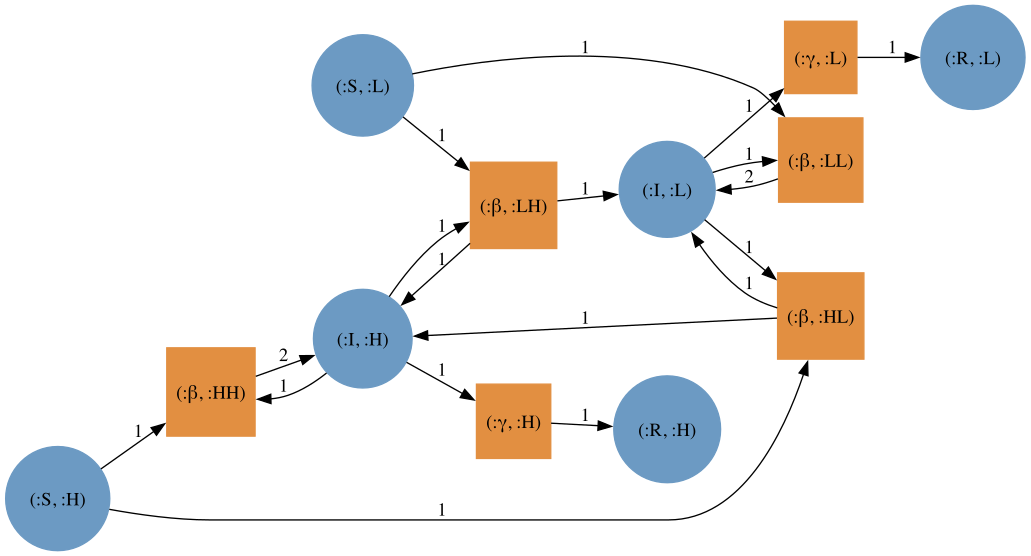
ACSetTransformation((T = FinFunction([1, 1, 1, 1], 4, 2), S = FinFunction([1, 1], 2, 1), I = FinFunction([1, 2, 1, 2, 1, 2, 1, 2], 8, 3), O = FinFunction([1, 2, 1, 2, 1, 2, 1, 2], 8, 3), Name = LooseVarFunction{Symbol, Symbol}(FinDomFunction(Union{AttrVar, Symbol}[], FinSet(0), TypeSet(Union{AttrVar, Symbol})), SetFunction(#10, TypeSet(Symbol), TypeSet(Symbol)), FinSet(0))), LabelledPetriNet {T:4, S:2, I:8, O:8, Name:0}, LabelledPetriNet {T:2, S:1, I:3, O:3, Name:0})

We create a stratified model by using a typed product between the SIR model and the risk model, to generate an ACSetTransformation, from which we subsequently extract a labelled Petri net.

## Add recovery within groups  
new\_tnames = [dom(risk\_acst)[:tname]; dom(risk\_acst)[:sname]]  
risk\_acst = add\_reflexives(risk\_acst, repeat([[:recovery]], 2), epi\_transitions)  
dom(risk\_acst)[:tname] = new\_tnames  
risk\_lpn = dom(risk\_acst)  
to\_graphviz(risk\_lpn)



sir\_risk\_acst = typed\_product(sir\_acst, risk\_acst)  
sir\_risk\_lpn = dom(sir\_risk\_acst)  
to\_graphviz(sir\_risk\_lpn)



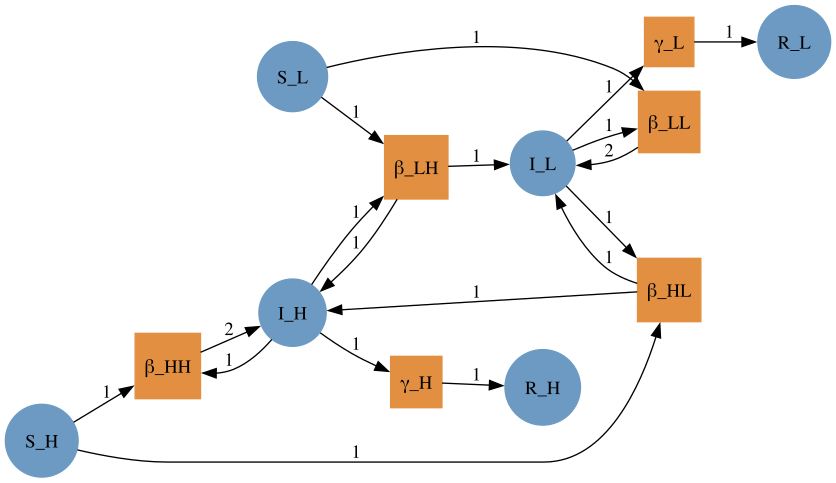
The state names of the resulting stratified model are tuples of symbols:

snames(sir\_risk\_lpn)

6-element Vector{Tuple{Symbol, Symbol}}:  
 (:S, :H)  
 (:I, :H)  
 (:R, :H)  
 (:S, :L)  
 (:I, :L)  
 (:R, :L)

The solvers in OrdinaryDiffEq.jl do not work with these state names, so we use flatten\_labels to obtain a vector of symbols (of the form :S\_H, :I\_H, etc.).

sir\_risk\_lpn\_flatlabels = flatten\_labels(sir\_risk\_lpn)  
to\_graphviz(sir\_risk\_lpn\_flatlabels)



## Running the model

To run the model, we need to choose specific group sizes and parameter values. We define two groups of equal size and initial composition.

K = 2  
S = [495.0, 495.0]  
I = [5.0, 5.0]  
R = [0.0, 0.0]  
N = [S[i]+I[i]+R[i] for i in 1:K]

2-element Vector{Float64}:  
 500.0  
 500.0

The model above is defined in terms of transmission rates between the two groups, which is a combination of (a) the contact rates between the two groups and (b) the probability of transmission per infectious contact. We assume that the two groups have different overall contact rates, c, and that the fraction of contacts between groups i and j, pij, follows a proportional mixing assumption.

β = 0.05  
c = [20.0, 5.0]  
pij = hcat([[c[j]\*N[j]/sum([c[k]\*N[k] for k in 1:K]) for j in 1:K] for i in 1:K]...)'  
betas = (β .\* (c .\* pij) ./ N)  
γ = 0.25;

We now define the initial conditions and parameter arrays as labelled arrays.

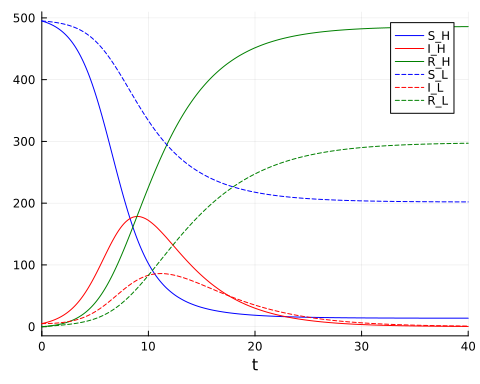
u0 = @LArray vec([S I R]')[:,1] Tuple(snames(sir\_risk\_lpn\_flatlabels))  
p = @LArray [vec(betas); γ; γ] Tuple(tnames(sir\_risk\_lpn\_flatlabels))  
tspan = (0.0, 40.0);

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

sir\_risk\_vf = vectorfield(sir\_risk\_lpn\_flatlabels)  
sir\_risk\_prob = ODEProblem(sir\_risk\_vf, u0, tspan, p)  
sir\_risk\_sol = solve(sir\_risk\_prob, Rosenbrock32());

## Plotting

plot(sir\_risk\_sol, linecolor=[:blue :red :green], linestyle=[:solid :solid :solid :dash :dash :dash])



## Automating the generation of risk groups

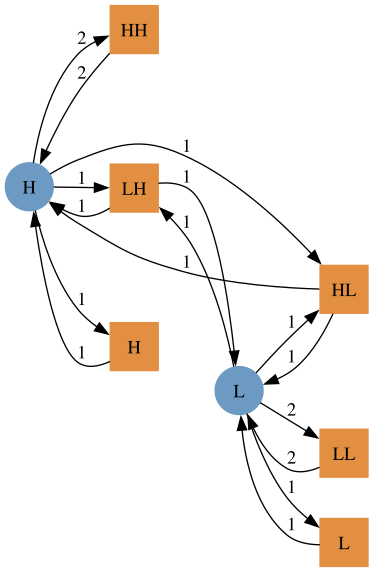
We can also build the risk model using Catlab’s imperative interface.

function make\_risk\_groups()  
 # Start with a blank UWD with 2 populations  
 uwd = RelationDiagram(repeat([:Pop], 2))  
 # Specify the names of the groups  
 names = ["H", "L"]  
 # Build junctions (just a `Dict`), with the side effect of updating the UWD  
 # `junctions` will be as follows  
 # Dict{Symbol, Int64} with 2 entries:  
 # :H => 1  
 # :L => 2  
 junctions = Dict(begin  
 variable = Symbol(names[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:  
 # 2×2 Matrix{Tuple{Symbol, Symbol}}:  
 # (:H, :H) (:H, :L)  
 # (:L, :H) (:L, :L)  
 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)  
 # So for the first pair, ins\_outs = (:H, :H, :H, :H)  
 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)  
 # The below loops through zip((:H, :H, :H, :H), [1, 2, 3, 4])  
 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;

We can now generate a LabelledPetriNet as follows.

risk\_automated\_acst = make\_risk\_groups()  
risk\_automated\_acst\_tnames = dom(risk\_automated\_acst)[:tname]  
risk\_automated\_acst\_snames = dom(risk\_automated\_acst)[:sname]  
## Add recovery within groups  
risk\_automated\_acst = add\_reflexives(risk\_automated\_acst, repeat([[:recovery]], 2), epi\_transitions)  
for s in risk\_automated\_acst\_snames  
 push!(risk\_automated\_acst\_tnames,s)  
end

dom(risk\_automated\_acst)[:tname] = risk\_automated\_acst\_tnames  
risk\_automated\_lpn = dom(risk\_automated\_acst)  
to\_graphviz(risk\_automated\_lpn)



We stratify the SIR model by the risk model using typed\_product, as before.

sir\_risk\_automated\_acst = typed\_product(sir\_acst, risk\_automated\_acst)  
sir\_risk\_automated\_lpn = dom(sir\_risk\_automated\_acst)  
to\_graphviz(sir\_risk\_automated\_lpn)

