Adding hospitalization to an SIR model using AlgebraicPetri.jl

Simon Frost (@sdwfrost)

2024-08-06

## Introduction

This notebook demonstrates how to add a new states to an existing model; in this case, adding hospitalization to an SIR model.

## Libraries

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

epi\_lpn = LabelledPetriNet(  
 [:Pop],  
 :infection=>((:Pop, :Pop)=>(:Pop, :Pop)),  
 :recovery=>(:Pop=>:Pop),  
 :hospitalization=>(:Pop=>:Pop),  
 :death=>(:Pop=>())  
);

Labelled Petri nets contain four types of fields; S, states or species; T, transitions; I, inputs; and O, outputs.

Next, we define the transmission model as an undirected wiring diagram using the @relation macro, referring to the transitions in our labelled Petri net above (infection and recovery). We include a reference to Pop in the definition of the state variables to allow us to do this.

sir\_uwd = @relation (S, I, R) where (S::Pop, I::Pop, R::Pop) begin  
 infection(S, I, I, I)  
 recovery(I, R)  
end;

We then use oapply\_typed, which takes in a labelled Petri net (here, epi\_lpn) and an undirected wiring diagram (si\_uwd), where each of the boxes is labeled by a symbol that matches the label of a transition in the Petri net, in addition to an array of symbols for each of the rates in the wiring diagram. This produces a Petri net given by colimiting the transitions together, and returns the ACSetTransformation from that Petri net to the type system.

sir\_acst = oapply\_typed(epi\_lpn, sir\_uwd, [:β, :γ]);

To obtain the labelled Petri net, we extract the domain of the ACSetTransformation using dom.

sir\_lpn = dom(sir\_acst);

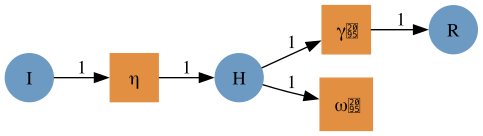
We can obtain a GraphViz representation of the labelled Petri net using to\_graphviz.

to\_graphviz(sir\_lpn)



We now define another model that considers another population representing individuals that are hospitalized following infection, and either recover or die.

h\_uwd = @relation (I, H, R) where (I::Pop, H::Pop, R::Pop) begin  
 hospitalization(I, H)  
 recovery(H, R)  
 death(H)  
end  
h\_acst = oapply\_typed(epi\_lpn, h\_uwd, [:η, :γₕ, :ωₕ])  
h\_lpn = dom(h\_acst)  
to\_graphviz(h\_lpn)



We also add death due to infection to the model.

i\_uwd = @relation (I,) where (I::Pop,) begin  
 death(I)  
end  
i\_acst = oapply\_typed(epi\_lpn, i\_uwd, [:ω])  
i\_lpn = dom(i\_acst)  
to\_graphviz(i\_lpn)



To glue the models together, we first define an undirected wiring diagram which contains all our states, and two transitions.

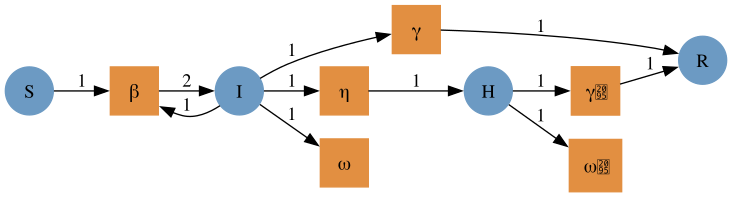
sirh\_uwd = @relation (S, I, R, H) where (S::Pop, I::Pop, R::Pop, H::Pop) begin  
 sir(S, I, R)  
 h(I, H, R)  
 i(I)  
end;

We then create a StructuredMulticospan using this wiring diagram, telling oapply that si in the wiring diagram corresponds to the si\_lpn labelled Petri net, etc.. Open converts a PetriNet to an OpenPetriNet where each state is exposed as a leg of the cospan, allowing it to be composed over an undirected wiring diagram.

sirh\_smc = oapply(sirh\_uwd, Dict(  
 :sir => Open(sir\_lpn),  
 :h => Open(h\_lpn),  
 :i => Open(i\_lpn)  
));

We extract the labelled Petri net by extracting the object that is the codomain of all the legs, using the apex function.

sirh\_lpn = apex(sirh\_smc)  
to\_graphviz(sirh\_lpn)



## Running the model

To run an ODE model from the labelled Petri net, we generate a function that can be passed to SciML’s ODEProblem using vectorfield.

sirh\_vf = vectorfield(sirh\_lpn);

The initial conditions and parameter values are written as labelled arrays. We can (and should) check the ordering of these variables.

snames(sirh\_lpn)

4-element Vector{Symbol}:  
 :S  
 :I  
 :R  
 :H

u0 = @LArray [990.0, 10.0, 0.0, 0.0] Tuple(snames(sirh\_lpn))

4-element LArray{Float64, 1, Vector{Float64}, (:S, :I, :R, :H)}:  
 :S => 990.0  
 :I => 10.0  
 :R => 0.0  
 :H => 0.0

tnames(sirh\_lpn)

6-element Vector{Symbol}:  
 :β  
 :γ  
 :η  
 :γₕ  
 :ωₕ  
 :ω

p = @LArray [0.5/1000, 0.25, 0.05, 0.2, 0.05, 0.05] Tuple(tnames(sirh\_lpn))

6-element LArray{Float64, 1, Vector{Float64}, (:β, :γ, :η, :γₕ, :ωₕ, :ω)}:  
 :β => 0.0005  
 :γ => 0.25  
 :η => 0.05  
 :γₕ => 0.2  
 :ωₕ => 0.05  
 :ω => 0.05

tspan = (0.0, 40.0);

We can now use the initial conditions, the time span, and the parameter values to simulate the system.

sirh\_prob = ODEProblem(sirh\_vf, u0, tspan, p)  
sirh\_sol = solve(sirh\_prob, Rosenbrock32())  
plot(sirh\_sol)

