Adding vaccination to an SIR model using AlgebraicPetri.jl

Simon Frost (@sdwfrost)

2024-08-06

## Introduction

This notebook demonstrates how to add a new transition between states to an existing model; in this case, adding vaccination 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),  
 :vaccination=>(:Pop=>: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 transitions between S and R due to vaccination (at rate σ).

v\_uwd = @relation (S, R) where (S::Pop, R::Pop) begin  
 vaccination(S, R)  
end  
v\_acst = oapply\_typed(epi\_lpn, v\_uwd, [:σ])  
v\_lpn = dom(v\_acst)  
to\_graphviz(v\_lpn)



To glue the SI and vaccination models together to make an SIR model, we first define an undirected wiring diagram which contains all our states, and two transitions.

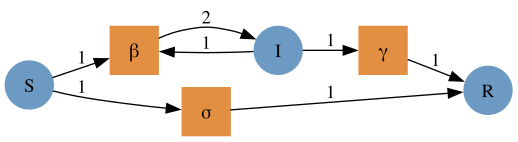
sirv\_uwd = @relation (S, I, R) where (S::Pop, I::Pop, R::Pop) begin  
 sir(S, I, R)  
 v(S, R)  
end;

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.

sirv\_smc = oapply(sirv\_uwd, Dict(  
 :sir => Open(sir\_lpn),  
 :v => Open(v\_lpn),  
));

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

sirv\_lpn = apex(sirv\_smc)  
to\_graphviz(sirv\_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.

sirv\_vf = vectorfield(sirv\_lpn);

The initial conditions and parameter values are written as labelled arrays.

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

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

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

3-element LArray{Float64, 1, Vector{Float64}, (:β, :γ, :σ)}:  
 :β => 0.0005  
 :γ => 0.25  
 :σ => 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.

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

