AlgebraicPetri Application Demo

AlgebraicPetri.jl (https://algebraicjulia.github.io/AlgebraicPetri.jl/dev/) is a library which simultaneously provides modelers access to the model construction and analysis tools contained in AlgebraicJulia (https://www.algebraicjulia.org/) and the simulation and analysis tools contained in SciML (https://sciml.ai/).

In [1]:

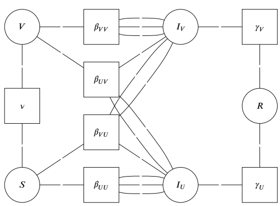
```julia
# Necessary imports
using AlgebraicPetri
using AlgebraicPetri.ModelComparison
using Semagrams, Semagrams.Examples
using Catlab
using Catlab.CategoricalAlgebra
using DifferentialEquations
using Plots
using JSON
using LabelledArrays
using PrettyTables

include("../src/GrometInterop.jl")
include("../src/ModelStratify.jl")
include("../src/Sensitivity.jl")
include("../src/EpiModel.jl")
using .GrometInterop
using .ModelStratify
using .Sensitivity
using .EpiModel;

solution(model::LabelledReactionNet, tspan) = begin
    solve(ODEProblem(vectorfield(model), concentrations(model), tspan, rates(model)), Tsit5())
end
default(linewidth=5, xaxis="Time", yaxis="Population")
function sens_table(sens; kw...)
    sens_matrix = hcat(collect(keys(sens)), collect(values(sens)))
    pretty_table(sens_matrix[sortperm(sens_matrix[:, 2]; rev=true), :]; header=(["Transition", "Sensitivity"],), kw...)
end;
function signed_log(vals)
    scale_fac = 1.0/(minimum(abs.(vals)) + 1e-10)
    log_sens = log10.(abs.((vals .+ 1e-10) .* scale_fac)) .* sign.(vals)
end;
get_ptnet(a) = begin
    epn = get_acset(a)
    pn = LabelledReactionNet(epn)
    set_subparts!(pn, 1:nt(pn),
        rate=map(r->try parse(Float64, r) catch e;  (t...)->Base.invokelatest(eval(Meta.parse(r)), t...) end,
            subpart(pn, :rate)))
    pn
end

flt_rates(pn) = begin
    r = rates(pn)
    l = findall(t-> t isa Float64, r)
    Dict(zip(l, r[l]))
    @LArray r[l] Tuple(l)
end

function show_graph(g)
    display("text/html",html"<style>div.graphs svg{max-width:100% !important;max-height:100% !important;</style>")
    display("text/html", "<div class=graphs>" * read(Catlab.Graphics.Graphviz.run_graphviz(g, format = "svg"), String) * "</div>")
end;
```

Unable to load WebIO. Please make sure WebIO works for your Jupyter client. For troubleshooting, please see the WebIO/IJulia documentation (https://juliagizmos.github.io/WebIO.jl/latest/providers/ijulia/).

In [10]:

```julia
export_html(p)
```

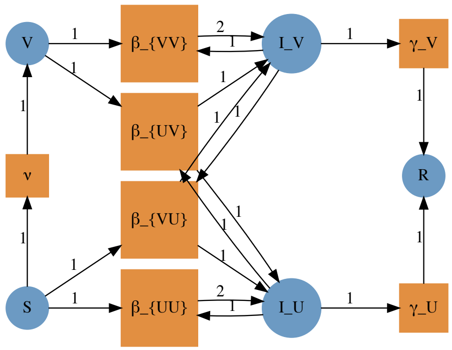
Out[10]:



In [3]:

```julia
load(p, "SVIIR.sema");
```

In [4]:
```
model = get_ptnet(p)
show_graph(Graph(model, p; scale=144 * 0.8))
```



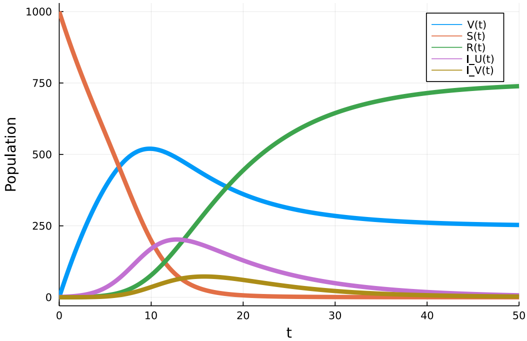
Code from Petrinet

In [5]:
```
using AlgebraicPetri.BilayerNetworks
bln = LabelledBilayerNetwork()
migrate!(bln, LabelledPetriNet(model))
BilayerNetworks.compile(bln, :du, :phi, :psi, rates(model))
```

Out[5]:
```
:(f!(du, phi, psi, t) = begin
          #= /Users/abaas3/Documents/work/projects/act/aske/algebraic_petri/upstream/graphics_update/AlgebraicPetri.jl/src/BilayerNet
works.jl:269 =#
          begin
              #= /Users/abaas3/Documents/work/projects/act/aske/algebraic_petri/upstream/graphics_update/AlgebraicPetri.jl/src/Bilaye
rNetworks.jl:274 =#
              du .= 0.0
              #= /Users/abaas3/Documents/work/projects/act/aske/algebraic_petri/upstream/graphics_update/AlgebraicPetri.jl/src/Bilaye
rNetworks.jl:275 =#
              phi .= 1.0
              phi[3] *= psi[2]
              phi[1] *= psi[2]
              phi[2] *= psi[4]
              phi[5] *= psi[4]
              phi[7] *= psi[4]
              phi[7] *= psi[2]
              phi[6] *= psi[5]
              phi[4] *= psi[1]
              phi[2] *= psi[1]
              phi[4] *= psi[5]
              phi[3] *= psi[5]
              phi[1] *= 0.1
              phi[2] *= 0.0002
              phi[3] *= 0.001
              phi[4] *= 0.0002
              phi[5] *= 0.1
              phi[6] *= 0.3
              phi[7] *= 0.001
              du[2] -= phi[3]
              du[2] -= phi[1]
              du[4] -= phi[2]
              du[4] -= phi[5]
              du[4] -= phi[7]
              du[2] -= phi[7]
              du[5] -= phi[6]
              du[1] -= phi[4]
              du[1] -= phi[2]
              du[5] -= phi[4]
              du[5] -= phi[3]
              du[5] += phi[4]
              du[4] += phi[7]
              du[4] += phi[3]
              du[3] += phi[6]
              du[4] += phi[2]
              du[3] += phi[5]
              du[5] += phi[2]
              du[5] += phi[4]
              du[1] += phi[1]
              du[4] += phi[7]
              du[5] += phi[3]
              return du
          end
      end)
```

In [7]:
```
sol = solution(model, (0.0,50.0))
plot(sol)
```

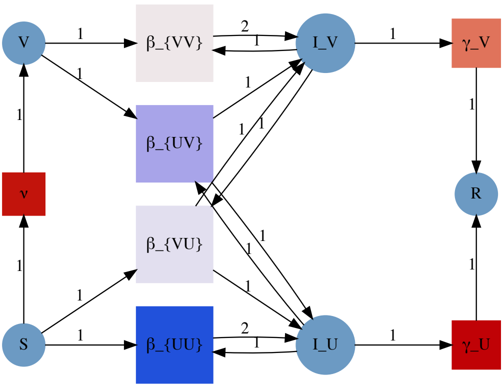
Out[7]:



In [8]:
```
int_met = Sensitivity.int_metric(model, [:R, :S, :V], t_range=(0.0,50.0))
sens = Sensitivity.sensitivity(int_met, flt_rates(model)) .* flt_rates(model) ./ int_met(flt_rates(model))
sens_table(sens)
```

Transition	Sensitivity
γ_U	0.101564
ν	0.0961692
γ_V	0.0363567
β_{VV}	-0.00636655
β_{VU}	-0.00815847
β_{UV}	-0.0221078
β_{UU}	-0.102475

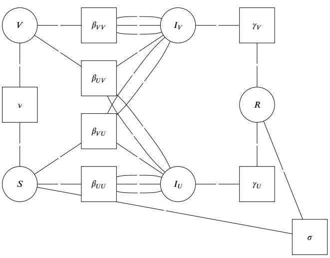
In [9]:
```
log_sens = signed_log(sens)
max_rng = maximum(abs.(log_sens))
show_graph(GraphHeatmap(model, log_sens, clims=(-max_rng, max_rng); positions=locations(model, save(p), scale=144 * 0.7)))
```



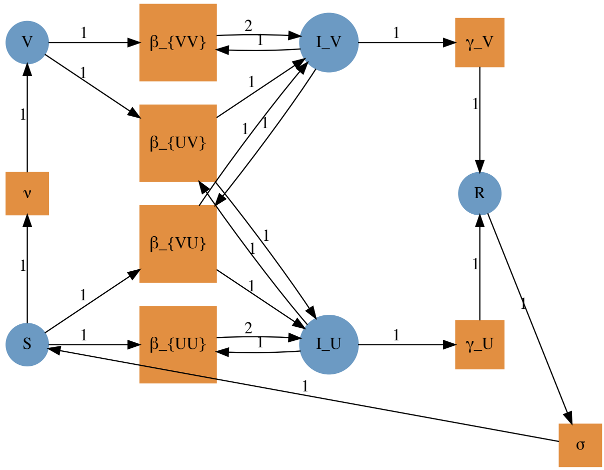
Add a hypothesized transition to the original model

In [12]:
```
export_html(p)
```

Out[12]:

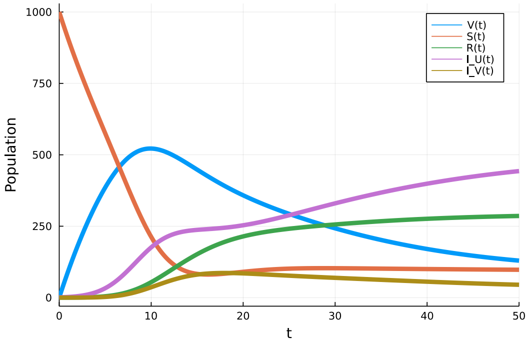


In [11]:
```
model_susc = get_ptnet(p)
show_graph(Graph(model_susc, p, scale = 144 * 0.7))
```



In [13]:
```
sol = solution(model_susc, (0.0, 50.0))
plot(sol)
```

Out[13]:

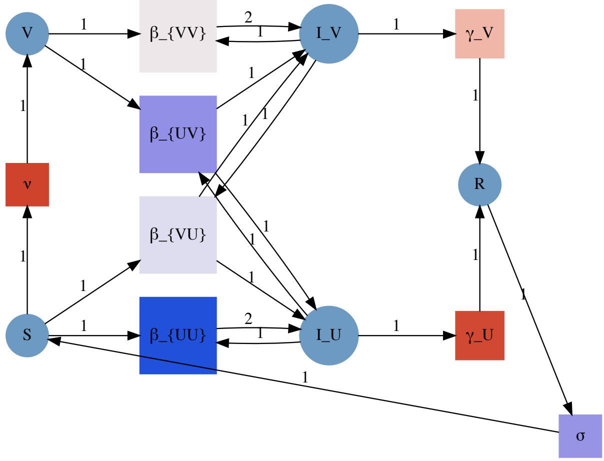


In [14]:
```
int_met = int_metric(model_susc, [:R, :S, :V], t_range=(0.0,50.0))
sens = sensitivity(int_met, rates(model_susc)) .* rates(model_susc) ./ int_met(rates(model_susc))
sens_table(sens)
```

Transition	Sensitivity
ν	0.365404
γ_U	0.34431
γ_V	0.0910854
β_{VV}	-0.0435335
β_{VU}	-0.0587142
σ	-0.1797
β_{UV}	-0.190486
β_{UU}	-0.560356

In [15]:
```
log_sens = signed_log(sens)
max_rng = maximum(abs.(log_sens))
GraphHeatmap(model_susc, log_sens; clims=(-max_rng, max_rng), positions=locations(model_susc, save(p), scale=144 * 0.7))
```

Out[15]:

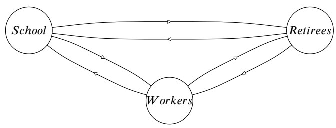


Hypothesized stratification of model

Specification of a Mixing Matrix

In [30]:
```
export_html(gen_graph)
```

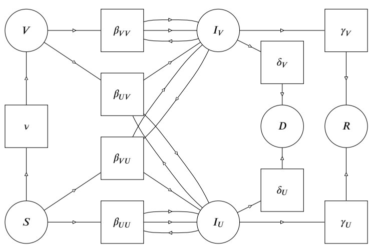
Out[30]:



In [17]:
```
load(gen_graph, "gen_graph.sema");
```

In [35]:
```
export_html(p_death)
```

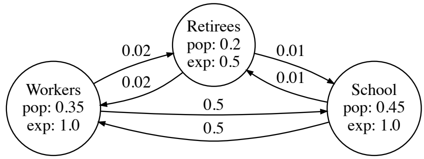
Out[35]:



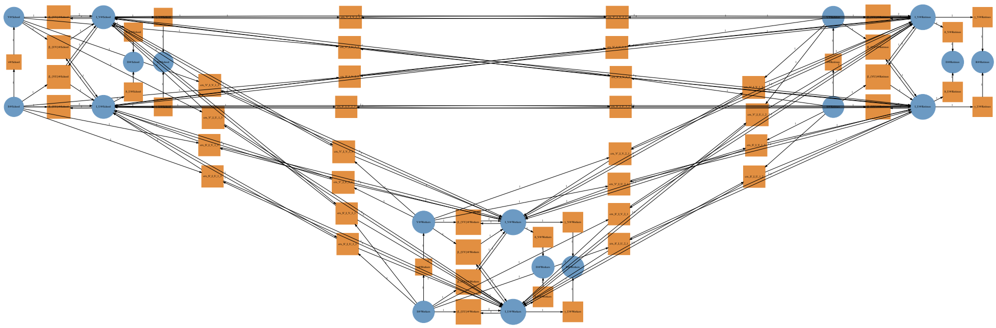
In [19]:
```
# Add a death transition
load(p_death, "SVIIRD.sema");
```

In [20]:
```
model_death = get_ptnet(p_death)
(generations = get_acset(gen_graph)) |> ModelStratify.show_graph
```

Out[20]:

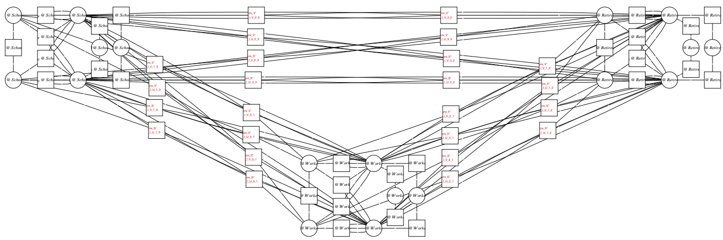


In [21]:
```
strat_model = ModelStratify.dem_strat(model_death, generations)
str_loc = ModelStratify.strat_location(strat_model, model_death, generations, locations(model_death, save(p_death), scale=144 * 0.4),
locations(generations, save(gen_graph)), scale=1.4)
show_graph(Graph(strat_model, positions=str_loc))
```



In [36]:
```
export_html(strat_sg)
```

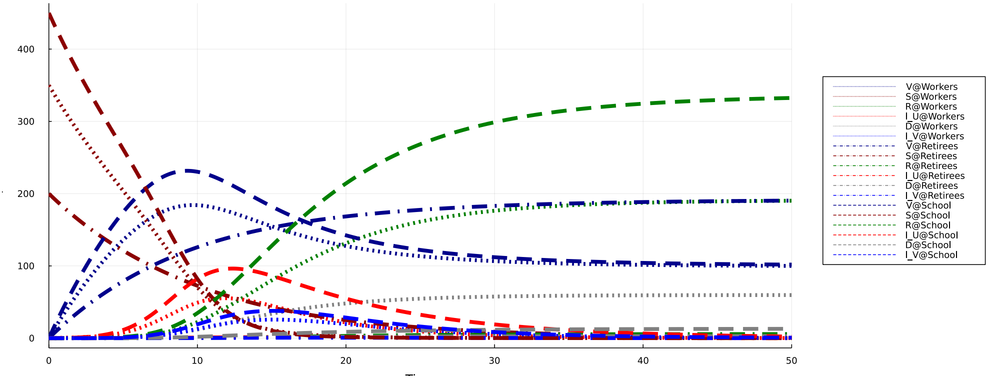
Out[36]:



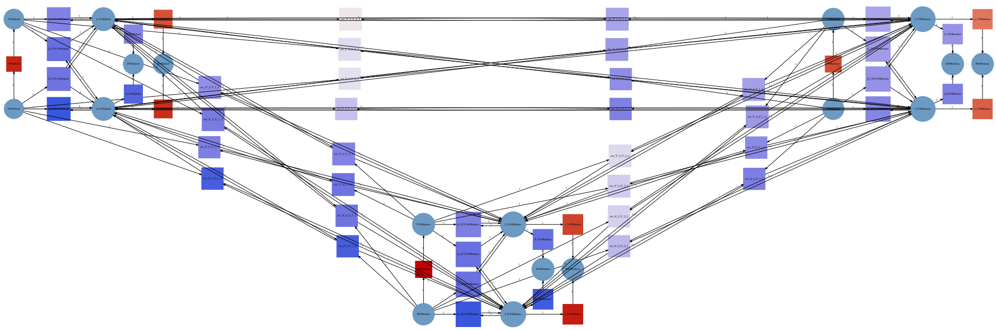
In [23]:
```
# Change some of the initial conditions and rates for different populations

strat_model[:δ_V, :School] *= 0.1
strat_model[:δ_U, :School] *= 0.1
strat_model[:δ_V, :Retirees] *= 1.5
strat_model[:δ_U, :Retirees] *= 1.5

strat_model[:I_U, :Retirees] = 0.0
strat_model[:I_U, :School] = 0.0;
```

In [24]:
```julia
tspan = (0.0, 50.0)
sol = solution(strat_model, tspan)
linestyles = Dict("School"=>:dash, "Retirees"=>:dashdot, "Workers"=>:dot)
linecolors = Dict("S"=>:darkred, "V"=>:darkblue, "I_U"=>:red, "I_V"=>:blue, "R"=>:green, "D"=>:gray)
times = range(tspan..., length=1000)

states = ["V", "I_U", "I_V", "D", "R", "S"]
pops = ["School", "Retirees", "Workers"]
names = filter(s ->first(split("$s", "@")) ∈ states && last(split("$s", "@")) ∈ pops,  snames(strat_model))
plot(times, hcat(collect([sol(t)[n] for t in times] for n in names)...); linestyle=reshape([linestyles[last(split("$s", "@"))] for s in names], (1,length(names))),
         color=reshape([linecolors[first(split("$s", "@"))] for s in names], (1,length(names))), labels=reshape(string.(names), (1,length(names))), legend=:outerright, size=(1.3e3,5e2), xrange=tspan)
```

Out[24]:



In [25]:
```julia
fin_met = Sensitivity.final_metric(strat_model, vcat([[Symbol(:D, "@$i")] for i in [:School, :Workers, :Retirees]]...), t_range=(0.0, 50.0))
sens = Sensitivity.sensitivity(fin_met, rates(strat_model)) .* rates(strat_model) ./ fin_met(rates(strat_model))
sens *= -1
log_sens = signed_log(sens)
max_rng = maximum(abs.(log_sens))
show_graph(Sensitivity.GraphHeatmap(strat_model, log_sens, clims=(-max_rng, max_rng), positions=str_loc))
```



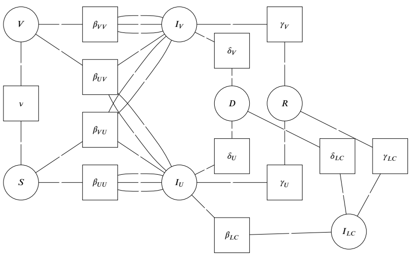
In [26]:
```
vax_vals = Highlighter((data, i, j)->(startswith("$(data[i,1])", "ν")), crayon"fg:black bold bg:yellow")
sens_table(sens; highlighters=(vax_vals,))
```

Transition	Sensitivity
ν@Workers	**0.848943**
γ_U@Workers	0.505455
ν@School	**0.396903**
γ_U@School	0.263965
γ_V@Workers	0.0876328
ν@Retirees	**0.0534387**
γ_V@School	0.0461593
γ_U@Retirees	0.0182882
γ_V@Retirees	0.00436017
crx_V′_I_V_2_3	-1.17187e-6
crx_S′_I_V_2_3	-3.47378e-6
crx_V′_I_U_2_3	-5.37703e-6
crx_V′_I_V_2_1	-6.16468e-6
crx_S′_I_V_2_1	-1.38395e-5
crx_V′_I_U_2_1	-1.72182e-5
crx_S′_I_U_2_3	-4.44581e-5
crx_S′_I_U_2_1	-9.81893e-5
β_{VV}@Retirees	-0.00045537
crx_V′_I_V_3_2	-0.00050463
crx_V′_I_V_1_2	-0.000683066
β_{UV}@Retirees	-0.000960095
crx_V′_I_U_1_2	-0.00132267
crx_V′_I_U_3_2	-0.00136724
δ_V@Retirees	-0.00207656
β_{VU}@Retirees	-0.00256549
crx_V′_I_V_1_3	-0.0030104
crx_S′_I_V_3_2	-0.00342454
crx_S′_I_V_1_2	-0.00463712
β_{UU}@Retirees	-0.00720639
δ_V@School	-0.00774339
β_{VV}@School	-0.00884027
crx_V′_I_V_3_1	-0.00934874
crx_S′_I_U_3_2	-0.0112359
crx_S′_I_V_1_3	-0.0119143
crx_S′_I_U_1_2	-0.0125454
β_{VV}@Workers	-0.0126722
δ_U@Retirees	-0.0129051
crx_V′_I_U_1_3	-0.0175438
crx_S′_I_V_3_1	-0.0304167
β_{VU}@School	-0.0335797
crx_V′_I_U_3_1	-0.0359078
β_{VU}@Workers	-0.0418467
β_{UV}@Workers	-0.042212
δ_V@Workers	-0.0504631
β_{UV}@School	-0.0510987
δ_U@School	-0.142307
crx_S′_I_U_3_1	-0.235213
crx_S′_I_U_1_3	-0.247976
δ_U@Workers	-0.333775
β_{UU}@School	-0.428701
β_{UU}@Workers	-0.439939

Long Covid Example

In [43]:
```
export_html(p_lcv)
```

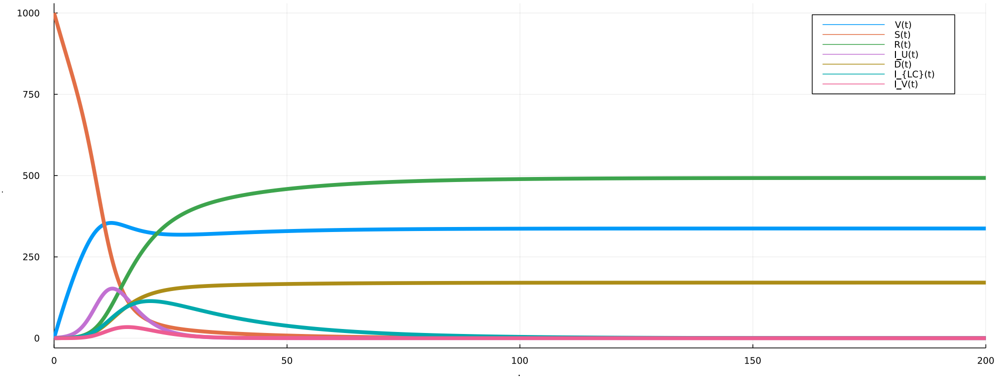
Out[43]:



In [28]:
```
load(p_lcv, "SVIIRD.sema");
```

```
In [41]: lcv_model = get_ptnet(p_lcv)
         tspan = (0.0, 200.0)
         sol = solution(lcv_model, tspan)
         plot(sol, xrange=tspan, size=(1.3e3,5e2))
```

Out[41]:



Acknowledgements

AlgebraicJulia

This library provides the tooling necessary for both the interactive editing of models and the stratification of these models.

SciML

This library provides the basic sensitivity analysis and simulation tooling.

Julia Community

This work would not be possible without the high level of work put in to developing the thriving community of Julia developers, especially the developers who are involved in the AlgebraicJulia and SciML projects.

Relevant Resources

John Baez and Jade Master, 2020. Open Petri nets. DOI:10.1017/S0960129520000043. arXiv:1808.05415

Micah Halter et al. Compositional Scientific Computing with Catlab and SemanticModels. arXiv:2005.04831