Categorical Epidemic Modelling

This notebook showcases the robustness of a categorical approach to modelling epidemics in their ability to accelerate the composition of various existing standard models, and stratifying the models to study their impact on different groups in the populace.

A preliminary understanding of categorical concepts such as Morphisms, Cospans, Structured Multicospans, Pullbacks and Natural Transformations, and a preliminary understanding of Petri-net modelling is assumed.

Setting up the Environment

We begin by importing the necessary modules for creating our system. Catlab provides support for Category Theoretical constructs and Graphviz, AlgebraicPetri is our friendly neighbourhood Petri-net maker, AlgebraicDynamics aids in converting Petri-nets to a dynamical system, the remaining modules have self-explanatory names.

If you face any issues in importing the modules in the following cell, make sure to resolve them using your package manager in Julia.

```julia
begin
    using Catlab, Catlab.CategoricalAlgebra, Catlab.Programs,
    Catlab.WiringDiagrams, Catlab.Graphics
    using AlgebraicPetri
    using AlgebraicDynamics.UWDDynam

    using DifferentialEquations

    using LabelledArrays
    using Plots
end
```

Our first Model (SEIRD)

The SEIRD model is a compartmental model involving the following compartments:

- S (Susceptible) - people who can potentially be infected by the disease through contact with an infected individual
- E (Exposed) - people who are infected with the disease, but are not infectious
- I (Infected) - people who are infected with the disease, and are infectious
- R (Recovered) - people who have gained permanent immunity to the disease
- D (Deceased) - people who died on account of the disease

We model this behaviour as a LabelledPetriNet (labelled because we wish to add 'names' to each transition and population compartment) and use the Open syntax to convert it to an 'open' petri-net in the form of a Structured Multicospan.

You are encouraged to use the Live Docs functionality to understand the syntax for creating a petri net. In short, here we have specified the list of comparments, the list of transitions along with their input and output arcs to create the petri net.

```
SEIRD_opetri_net =
 StructuredMulticospan(Multicospan(
                                          AlgebraicPetri.LabelledPetriNet {T:5, S:5, I:7, O:7, Nam
```

T	tname
1	expose_u
2	expose_u
3	infect_u
4	recover_u
5	death_u

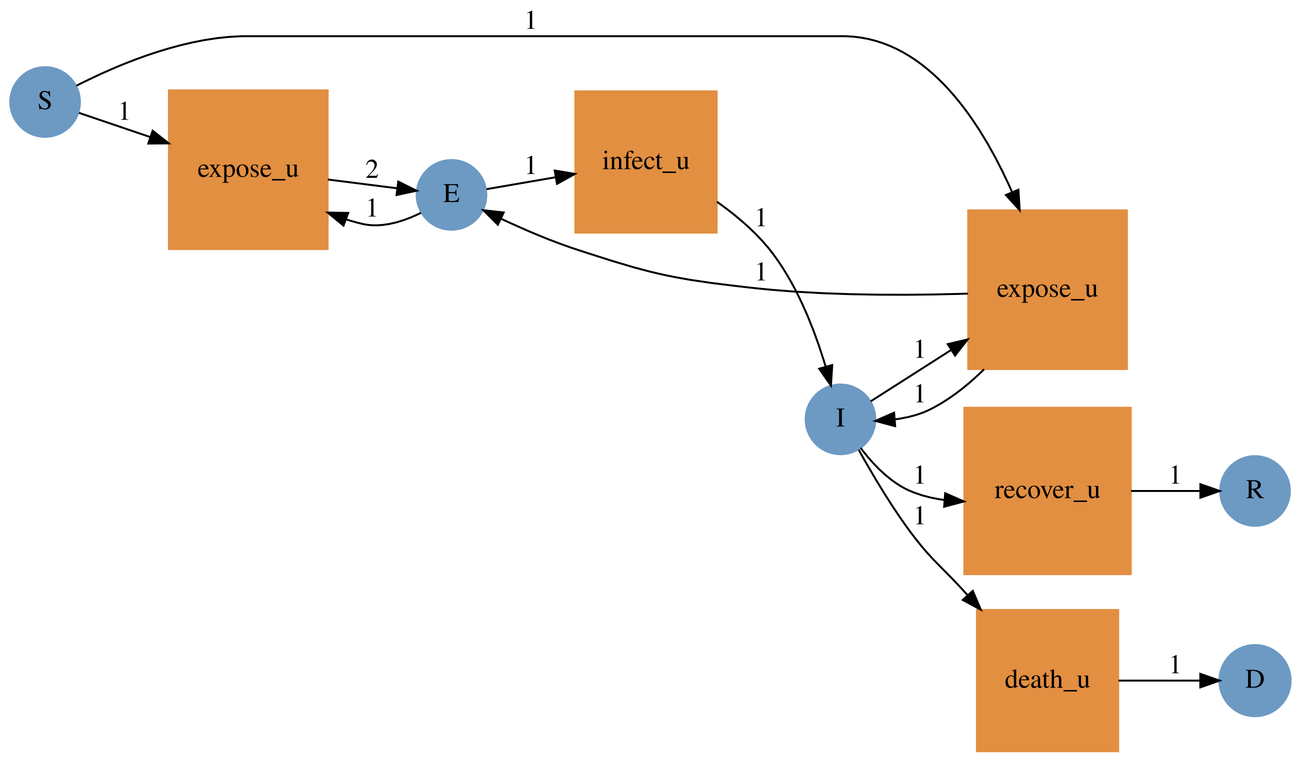
S	sname
1	S
2	E
3	I
4	R
5	D

I	it	is
1	1	1
2	1	2
3	2	1
4	2	3
5	3	2
6	4	3
7	5	3

O	ot	os
1	1	2
2	1	2
3	2	2
4	2	3
5	3	3
6	4	4
7	5	5

```julia
# An SIERD Model (Susceptible, Exposed, Infected, Recovered, Deceased)
SEIRD_opetri_net = Open(LabelledPetriNet(

    [:S, :E, :I, :R, :D],

    :expose_u => ((:S, :E) => (:E, :E)),
    :expose_u => ((:S, :I) => (:E, :I)),
    :infect_u => ((:E) => (:I)),
    :recover_u => ((:I) => (:R)),
    :death_u => ((:I) => (:D)),
    )
)
```

The tool graphviz can be used to visualize the petri-net created above.



```julia
to_graphviz(SEIRD_opetri_net)
```

Our second Model (VEvR)

The VEvR model is a non-standard compartmental model involving the following compartments:

- V (Vaccinated) - people who have been vaccinated
- Ev (Exposed-Vaccinated) - people who were once vaccinated, but came in contact with an infected individual or another exposed vaccinated individual. These people can never die from the disease and can only recover from it to gain immunity later.
- R (Recovered) - people who have gained permanent immunity to the disease

We use a similar syntax as the one in the previous cells to create and visualise the petri-net corresponding to this model.

```
VEvR_opetri_net =
  StructuredMulticospan{Catlab.CategoricalAlgebra.StructuredCospans.DiscreteACSet{AnonACSet{
     cospan =   SMulticospan{3, LabelledPetriNet, StructTightACSetTransformation{TypeLevelBa
                    apex =
                          AlgebraicPetri.LabelledPetriNet {T:2, S:3, I:3, O:3, Name:0}
```

T	tname
1	expose_v
2	recover_v

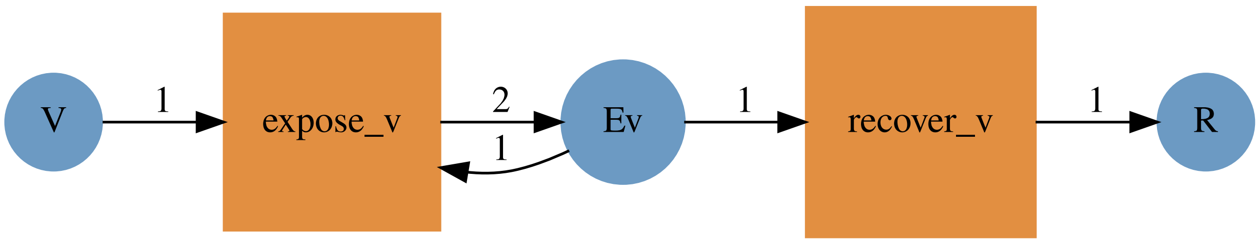
S	sname
1	V
2	Ev
3	R

I	it	is
1	1	1
2	1	2
3	2	2

O	ot	os
1	1	2
2	1	2
3	2	3

```
                    legs =   StaticArraysCore.SVector{3, Catlab.CategoricalAlgebra.CSets.Stru
                )
     feet =   StaticArraysCore.SVector{3, ACSets.DenseACSets.AnonACSet{ACSets.Schemas.TypeLe

)
```

```julia
# An VEvR Model (Vaccinated, Exposed_Vaccinated, Recovered)
VEvR_opetri_net = Open(LabelledPetriNet(

    [:V, :Ev, :R],

    :expose_v => ((:V, :Ev) => (:Ev, :Ev)),
    :recover_v => ((:Ev) => (:R)),
    )
)
```



```
to_graphviz(VEvR_opetri_net)
```

Dictating interaction between the two Models

```
md"
## Dictating interaction between the two Models

"
```

```
cross_exposure_opetri_net =
  StructuredMulticospan{Catlab.CategoricalAlgebra.StructuredCospans.DiscreteACSet{AnonACSet{
     cospan =   SMulticospan{4, LabelledPetriNet, StructTightACSetTransformation{TypeLevelBa
                    apex =
                           AlgebraicPetri.LabelledPetriNet {T:2, S:4, I:3, O:3, Name:0}
```

T	tname
1	iexpose_v
2	svax

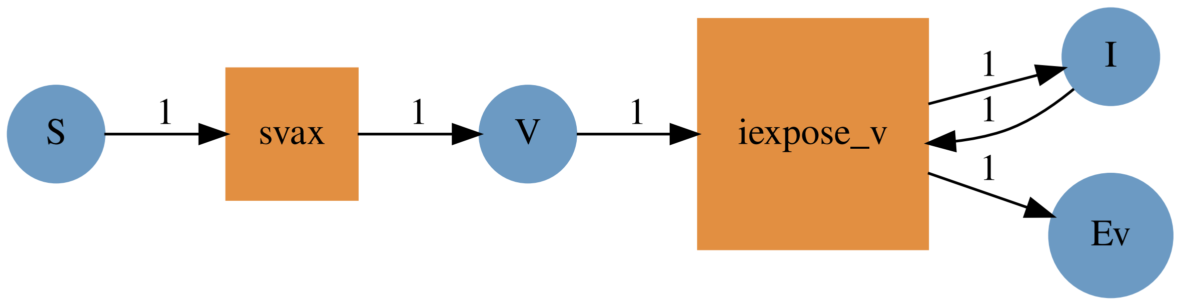
S	sname
1	S
2	I
3	V
4	Ev

I	it	is
1	1	3
2	1	2
3	2	1

O	ot	os
1	1	2
2	1	4
3	2	3

```
                    legs =   StaticArraysCore.SVector{4, Catlab.CategoricalAlgebra.CSets.Stru
                  )
     feet =   StaticArraysCore.SVector{4, ACSets.DenseACSets.AnonACSet{ACSets.Schemas.TypeLe
)
```

```
# Dicating cross exposure rules
cross_exposure_opetri_net = Open(LabelledPetriNet(

    [:S, :I, :V, :Ev],

    :iexpose_v => ((:V, :I) => (:I, :Ev)),
    :svax => ((:S) => (:V)),
    )
)
```



```
to_graphviz(cross_exposure_opetri_net)
```

```
SVIEEvRD_composition_uwd =
```

Catlab.Programs.RelationalPrograms.UntypedUnnamedRelationDiagram{Symbol, Symbol}
{Box:3, Port:12, OuterPort:7, Junction:7, Name:0, VarName:0}

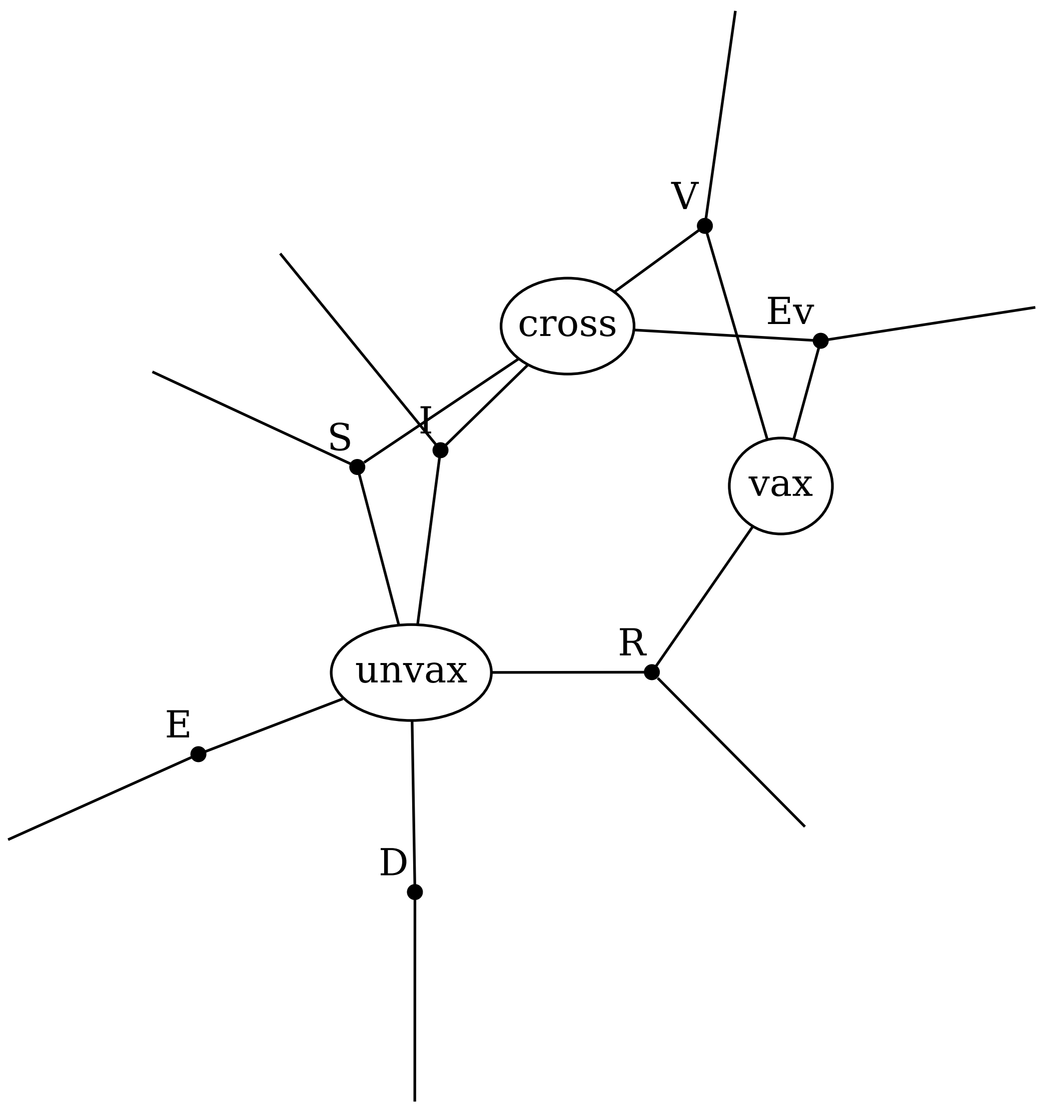
Box	name
1	unvax
2	vax
3	cross

Port	box	junction
1	1	1
2	1	4
3	1	3
4	1	6
5	1	7
6	2	2
7	2	5
8	2	6
9	3	1
10	3	3
11	3	2
12	3	5

OuterPort	outer_junction
1	1
2	2
3	3
4	4
5	5
6	6
7	7

Junction	variable
1	S
2	V
3	I
4	E
5	Ev
6	R
7	D

```
SVIEEvRD_composition_uwd = @relation (S, V, I, E, Ev, R, D) where (S, V, I, E,
Ev, R, D) begin
    unvax(S, E, I, R, D)
    vax(V, Ev, R)
    cross(S, I, V, Ev)
end
```



```
to_graphviz(SVIEEvRD_composition_uwd, box_labels=:name,
junction_labels=:variable, edge_attrs=Dict(:len => "1"))
```

```
SVIEEvRD_opetri_net =
 StructuredMulticospan(Multicospan(
                                   AlgebraicPetri.LabelledPetriNet {T:9, S:7, I:13, O:13, N
```

T	tname
1	expose_u
2	expose_u
3	infect_u
4	recover_u
5	death_u
6	expose_v
7	recover_v
8	iexpose_v
9	svax

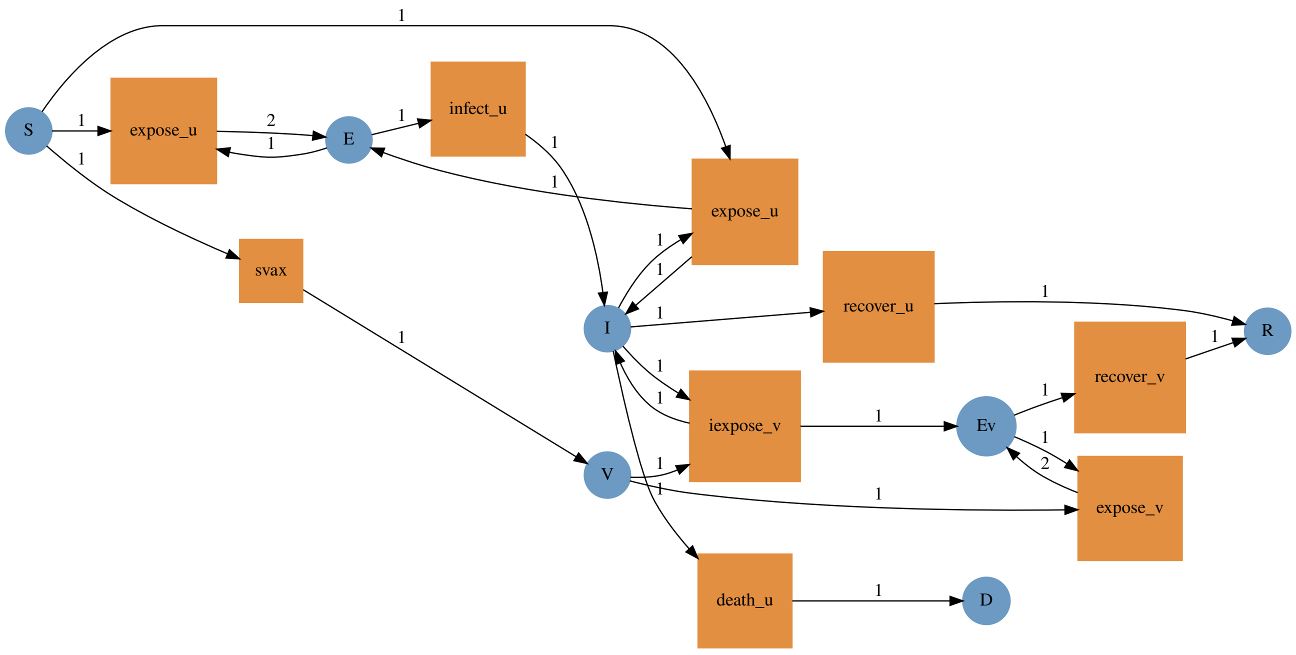
S	sname
1	S
2	E
3	I
4	R
5	D
6	V
7	Ev

I	it	is
1	1	1
2	1	2
3	2	1
4	2	3
5	3	2
6	4	3
7	5	3
8	6	6
9	6	7
10	7	7
11	8	6
12	8	3
13	9	1

O	ot	os
1	1	2
2	1	2
3	2	2
4	2	3
5	3	3
6	4	4

7	5	5
8	6	7
9	6	7
10	7	4
11	8	3
12	8	7
13	9	6

```
SVIEEvRD_opetri_net = oapply(
    SVIEEvRD_composition_uwd,
    Dict(:unvax => SEIRD_opetri_net, :vax => VEvR_opetri_net, :cross =>
cross_exposure_opetri_net)
)
```



```
to_graphviz(SVIEEvRD_opetri_net)
```

AlgebraicPetri.LabelledPetriNetUntyped{Nothing} {T:2, S:1, I:3, O:3, Name:0}

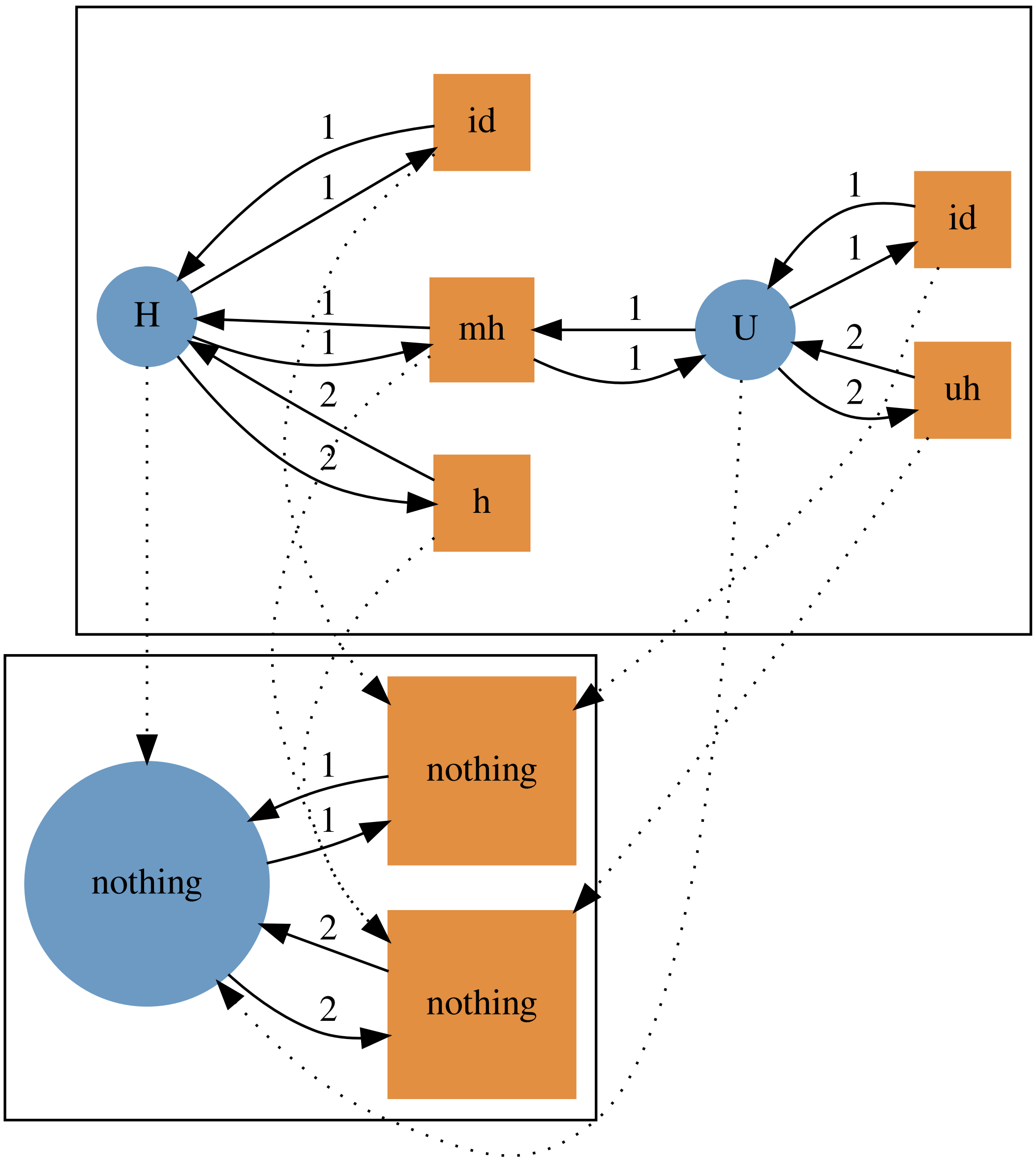
T	tname
1	nothing
2	nothing

S	sname
1	nothing

I	it	is
1	1	1
2	1	1
3	2	1

O	ot	os
1	1	1
2	1	1
3	2	1

```julia
begin
    # General population type
    infectious_type = LabelledPetriNet(
        [:Pop],

        :interact => ((:Pop, :Pop) => (:Pop, :Pop)),
        :t_status => ((:Pop) => (:Pop))
    )

    s, = parts(infectious_type, :S)
    t_interact, t_status = parts(infectious_type, :T)
    i_interact1, i_interact2, i_status = parts(infectious_type, :I)
    o_interact1, o_interact2, o_status = parts(infectious_type, :O)

    infectious_type = map(infectious_type, Name=name->nothing)
end
```


id
1
1
H
1
mh
1
1
1
U
1
1
id
2
uh
2
2
2
h
nothing
1
1
nothing
2
2
nothing

```julia
begin
    # Hygiene Stratification

    hygiene_petri_net = LabelledPetriNet(
        [:H, :U],

        :h => ((:H, :H) => (:H, :H)),
        :uh => ((:U, :U) => (:U, :U)),
        :mh => ((:U, :H) => (:U, :H)),

        :id => (:H => :H),
        :id => (:U => :U),
    )

    typed_hygiene_petri_net = ACSetTransformation(
        hygiene_petri_net, infectious_type,
        S = [s, s],
            # expose,    infect,    recover,  die,       id begins
        T = [t_interact, t_interact, t_interact, t_status, t_status],
        I = [i_interact1, i_interact2, i_interact1, i_interact2, i_interact1, i_interact2, i_status, i_status],
        O = [o_interact1, o_interact2, o_interact1, o_interact2, o_interact1, o_interact2, o_status, o_status],
        Name = name -> nothing
    )

    to_graphviz(typed_hygiene_petri_net)

end
```

```
add_id (generic function with 1 method)
```

```julia
# Adding identity transitions to the compartments

function add_id(petrinet::LabelledPetriNet)
    n_comp = ns(petrinet)
    ts = add_transitions!(petrinet, n_comp, tname = :id)
    add_inputs!(petrinet, n_comp, ts, 1:n_comp)
    add_outputs!(petrinet, n_comp, ts, 1:n_comp)
    return petrinet
end
```

```
SVIEEvRD_petri_net =
```

AlgebraicPetri.LabelledPetriNet {T:16, S:7, I:20, O:20, Name:0}

T	tname
1	expose_u
2	expose_u
3	infect_u
4	recover_u
5	death_u
6	expose_v
7	recover_v
8	iexpose_v
9	svax
10	id
11	id
12	id
13	id
14	id
15	id
16	id

S	sname
1	S
2	E
3	I
4	R
5	D
6	V
7	Ev

I	it	is
1	1	1
2	1	2
3	2	1
4	2	3
5	3	2
6	4	3
7	5	3
8	6	6
9	6	7
10	7	7
11	8	6
12	8	3
13	9	1
14	10	1
15	11	2
16	12	3
17	13	4
18	14	5

I	it	is
19	15	6
20	16	7

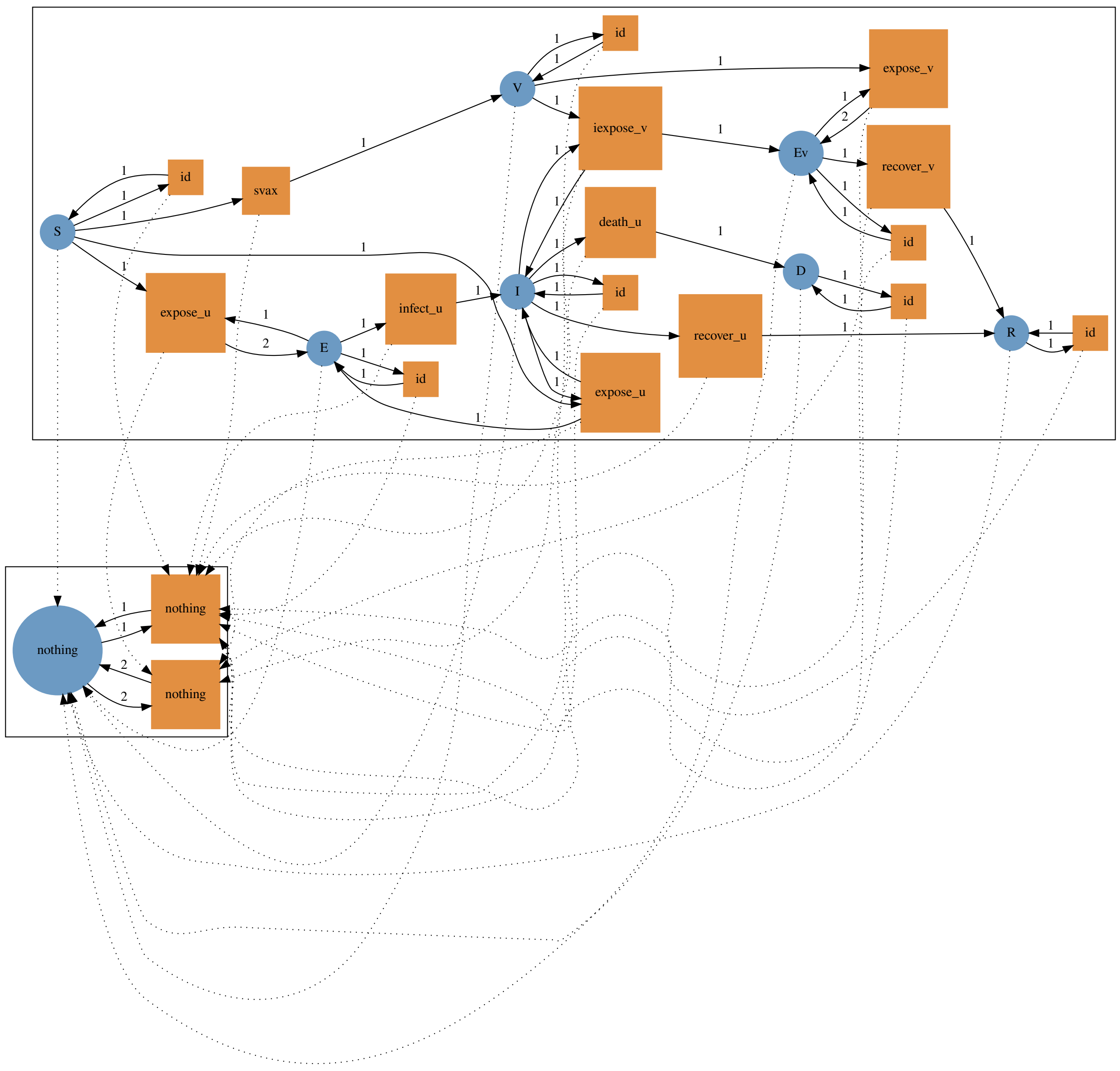
O	ot	os
1	1	2
2	1	2
3	2	2
4	2	3
5	3	3
6	4	4
7	5	5
8	6	7
9	6	7
10	7	4
11	8	3
12	8	7
13	9	6
14	10	1
15	11	2
16	12	3
17	13	4
18	14	5
19	15	6
20	16	7

```julia
SVIEEvRD_petri_net = add_id(apex(SVIEEvRD_opetri_net))
```

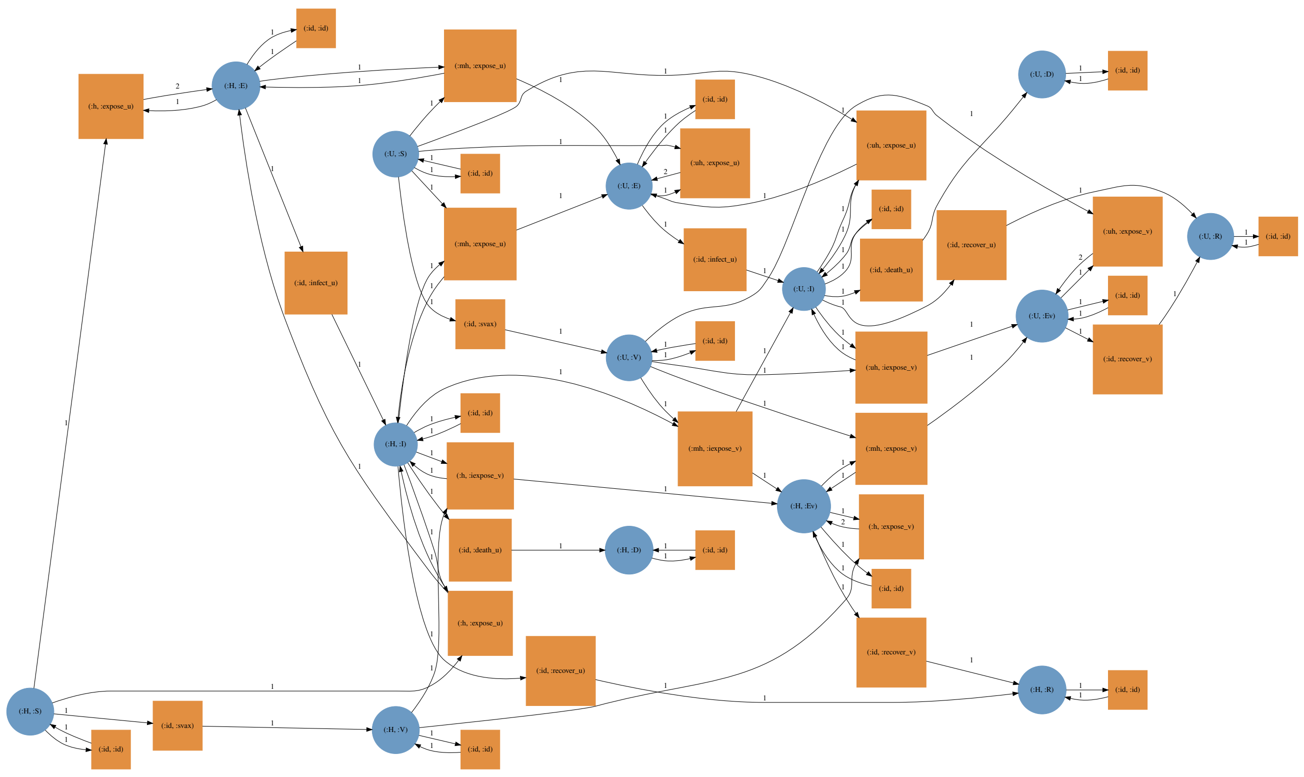
```
typed_SVIEEvRD_petri_net =
ACSetTransformation((T = FinFunction([1, 1, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2],
```

```julia
# Adding typing to the SVIEEvRD_petri_net

typed_SVIEEvRD_petri_net = ACSetTransformation(SVIEEvRD_petri_net,
infectious_type,
    S = repeat([s], ns(SVIEEvRD_petri_net)),
    T = vcat(t_interact, t_interact, t_status, t_status, t_status, t_interact,
t_status, t_interact, t_status, repeat([t_status], ns(SVIEEvRD_petri_net))),
    I = vcat(i_interact1, i_interact2, i_interact1, i_interact2, i_status,
i_status, i_status, i_interact1, i_interact2, i_status, i_interact1,
i_interact2, i_status, repeat([i_status], ns(SVIEEvRD_petri_net))),
    O = vcat(o_interact1, o_interact2, o_interact1, o_interact2, o_status,
o_status, o_status, o_interact1, o_interact2, o_status, o_interact1,
o_interact2, o_status, repeat([o_status], ns(SVIEEvRD_petri_net))),
    Name = name -> nothing
)
```

```julia
@assert is_natural(typed_SVIEEvRD_petri_net)
```



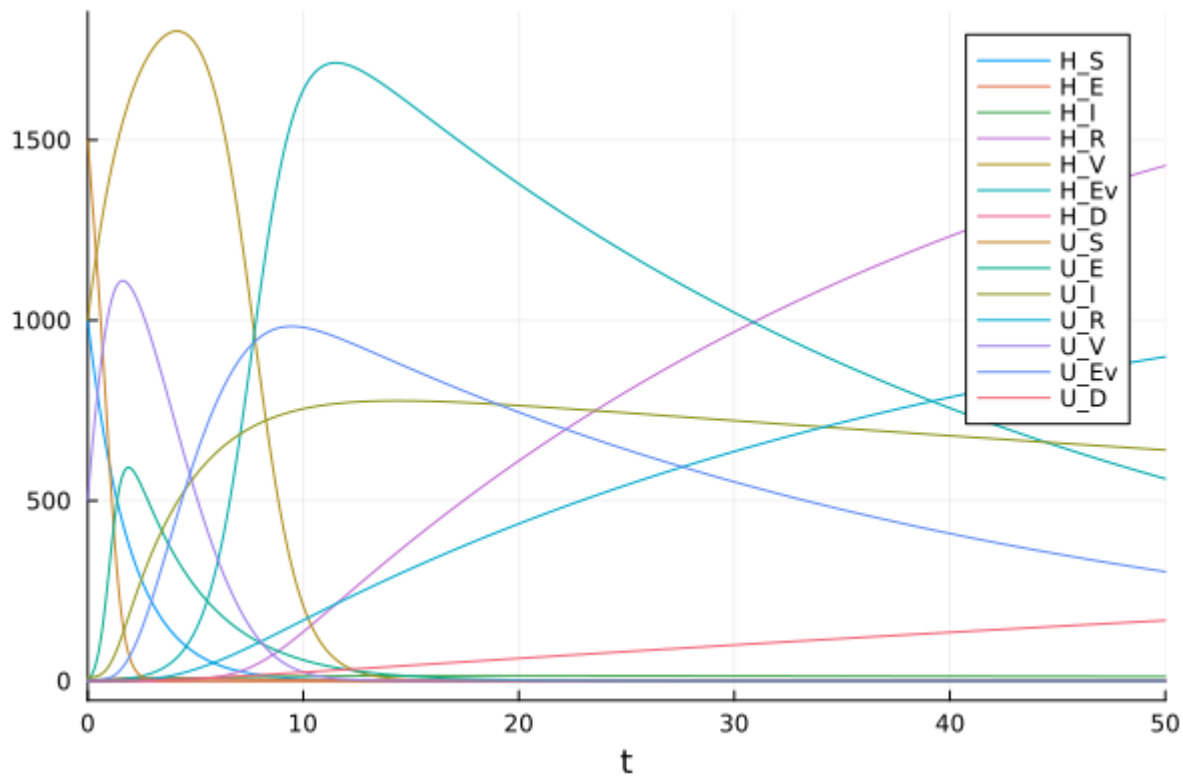
```
to_graphviz(typed_SVIEEvRD_petri_net, program="dot")
```



```julia
begin
    # Stratify the model
    UH_SVIEEvRD_petri_net =  ob(pullback(typed_hygiene_petri_net,
    typed_SVIEEvRD_petri_net))
    to_graphviz(UH_SVIEEvRD_petri_net)
end
```

```
LabelledArrays.LArray{Int64, 1, Vector{Int64}, (:H_S, :H_E, :H_I, :H_R, :H_V, :H_Ev, :H_D,
```

```julia
begin
    p = LVector(

        h_expose_u  = 0.001,
        uh_expose_u = 0.003,
        mh_expose_u = 0.002,
        id_infect_u = 0.3,
        id_recover_u = 0.001,
        id_death_u = 0.005,

        h_expose_v = 0.0005,
        uh_expose_v = 0.0001,
        mh_expose_v = 0.0002,
        id_recover_v = 0.03,

        h_iexpose_v = 0.0001,
        uh_iexpose_v = 0.0005,
        mh_iexpose_v = 0.0003,

        id_svax = 0.5,
        id_id = 1
    )

    u0 = LVector(
        H_S = 1000,
        H_E = 0,
        H_I = 5,
        H_R = 0,
        H_V = 1000,
        H_Ev = 0,
        H_D = 0,
        U_S = 1500,
        U_E = 0,
        U_I = 10,
        U_R = 0,
        U_V = 500,
        U_Ev = 0,
        U_D = 0,
    )
end
```



```julia
begin
    stratified_model = flatten_labels(UH_SVIEEvRD_petri_net)

    soln = solve(ODEProblem(vectorfield(stratified_model), u0, (0.0, 50.0), p))
    plot(soln)

end
```

```
Using arrays or dicts to store parameters of different types can hurt perform
ance.
Consider using tuples instead.
```