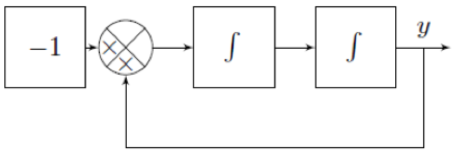
Synthetic CRN for Oscillators and Sigmoids

- CRN synthesis from GPAC circuits
- CRN synthesis for generating functions of time
- CRN synthesis for computing input/output functions
- Trigonometric functions for oscillators

F. Fages, 2019-2022

GPAC generating cosine(t)



In [1]:
```
compile_wgpac(y::integral integral-1*y).
```

Out[1]:

In [2]:
```
present(y,1).
```

Out[2]:

In [3]:
```
list_model.
```

Out[3]:
```
fast*[x2]*[y] for x2+y=>x1+x2+y.
fast*[x1] for x1=>_.
MA(1) for x1=>x0+x1.
MA(1) for x0=>x0+y.
initial_state(x2= -1).
initial_state(y=1).
parameter(
  fast = 1000
).
```

In [4]:
```
list_ode.
```

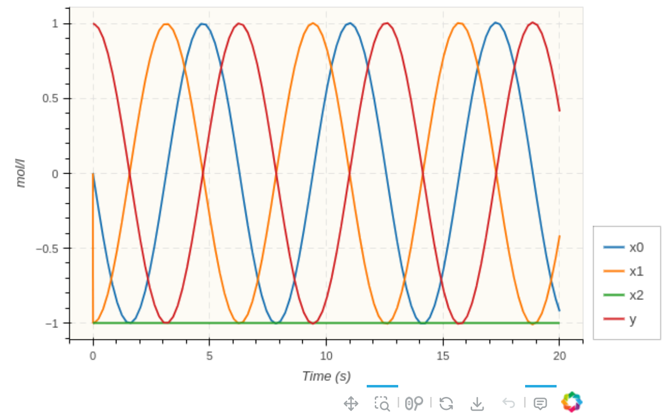
$$\begin{aligned} y_0 &= 1 \\ x0_0 &= 0 \\ x1_0 &= 0 \\ x2_0 &= -1 \\ fast &= 1000 \\ \frac{dy}{dt} &= x0 \\ \frac{dx0}{dt} &= x1 \\ \frac{dx1}{dt} &= fast * [x2] * [y] - fast * [x1] \\ \frac{dx2}{dt} &= 0 \end{aligned}$$

Out[4]:

In [5]:
```
numerical_simulation.
```

Out[5]:

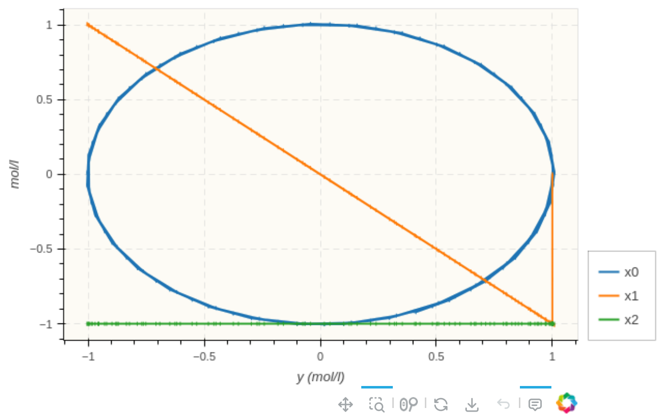
In [6]:
```
plot.
```



Out[6]:

In [7]:
```
plot(against:y).
```

```
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
```



Animate

Out[7]:

CRN synthesis for generating cosine(time)

In [8]:
```
clear_model.
```
Out[8]:

In [9]:
```
compile_from_expression(cos, time, f).
```
Out[9]:

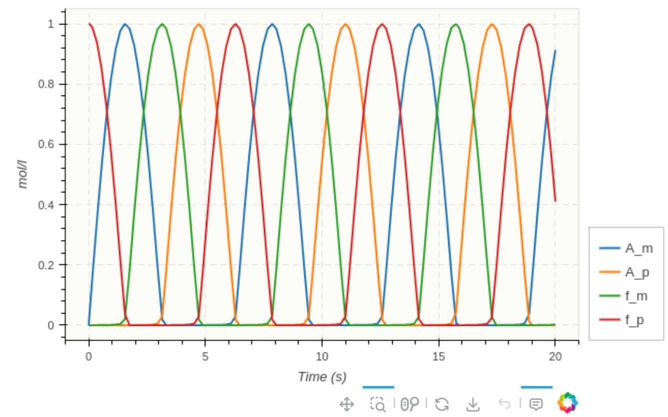
In [10]:
```
list_model.
```
Out[10]:
```
MA(fast) for f_m+f_p=>_.
MA(fast) for A_m+A_p=>_.
MA(1.0) for A_p=>A_p+f_p.
MA(1.0) for A_m=>A_m+f_m.
MA(1.0) for f_m=>A_p+f_m.
MA(1.0) for f_p=>A_m+f_p.
initial_state(f_p=1).
parameter(
  fast = 1000
).
```
In [11]:
```
search_conservations.
```
Out[11]:
```
No complex invariant found
```
In [12]:
```
list_ode.
```

$$A\_m_0 = 0$$
$$A\_p_0 = 0$$
$$f\_m_0 = 0$$
$$f\_p_0 = 1$$
$$fast = 1000$$
$$\frac{dA\_m}{dt} = f\_p - A\_m * A\_p * fast$$
$$\frac{dA\_p}{dt} = f\_m - A\_m * A\_p * fast$$
$$\frac{df\_m}{dt} = A\_m - f\_m * f\_p * fast$$
$$\frac{df\_p}{dt} = A\_p - f\_m * f\_p * fast$$

Out[12]:

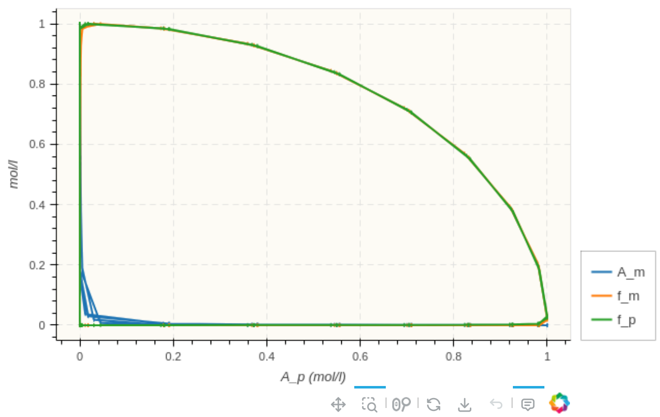
In [13]:
```
numerical_simulation.
```
Out[13]:

In [14]:
```
plot.
```



Out[14]:

In [15]:
```
plot(against:A_p).
```
```
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
```

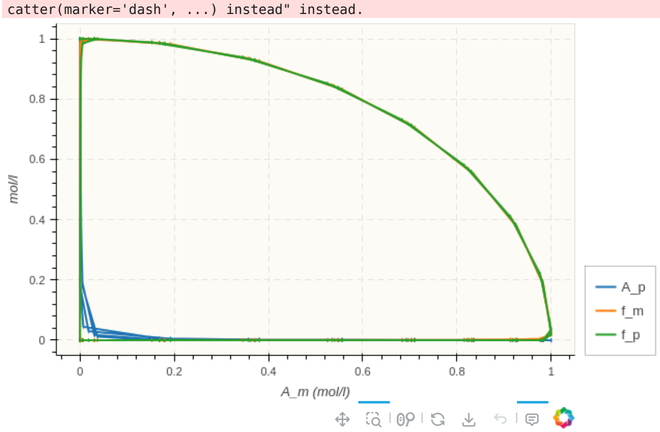


Animate

Out[15]:

In [16]:
```
plot(against:A_m).
```

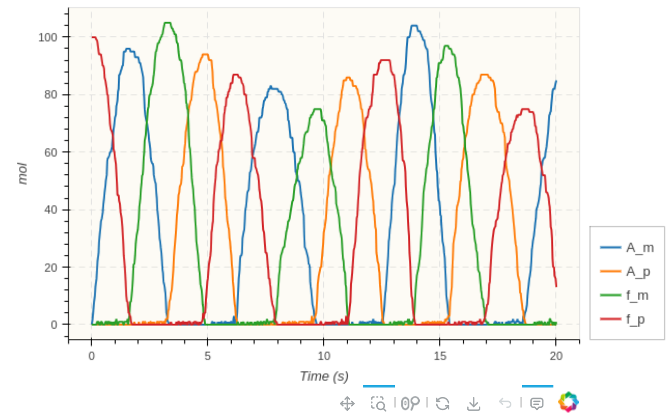
```
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
```



Animate

Out[16]:

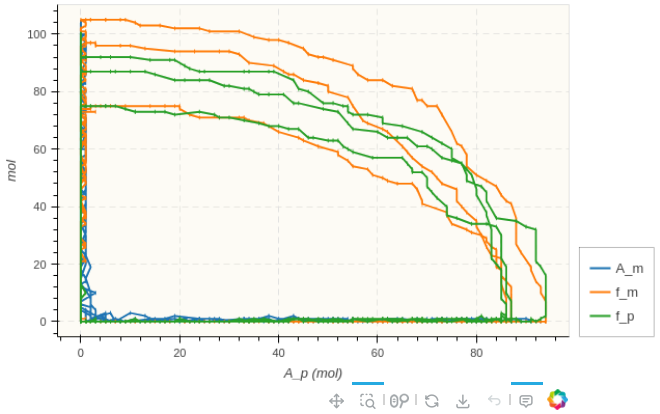
In [17]:
```
seed(2). numerical_simulation(method:ssa). plot.
```



Out[17]:

In [18]:
```
plot(against:A_p).
```

```
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
BokehDeprecationWarning: 'dash() method' was deprecated in Bokeh 3.4.0 and will be removed, use "s
catter(marker='dash', ...) instead" instead.
```



Animate

Out[18]:

CRN synthesis for generating cosine(time)+1

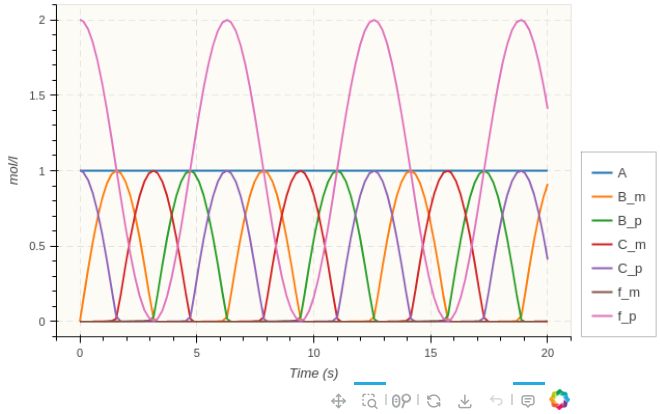
In [19]:
```
compile_from_expression(cos+1, f).
```

Out[19]:

In [20]:
```
list_model.
```

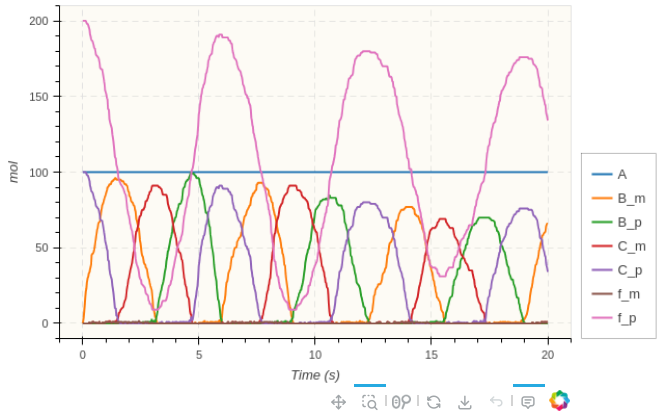
Out[20]:
```
MA(fast) for f_m+f_p=>_.
MA(fast) for C_m+C_p=>_.
MA(fast) for B_m+B_p=>_.
MA(1.0) for B_p=>B_p+C_p+f_p.
MA(1.0) for B_m=>B_m+C_m+f_m.
MA(1.0) for C_m=>B_p+C_m.
MA(1.0) for C_p=>B_m+C_p.
initial_state(f_p=2).
initial_state(C_p=1).
initial_state(A=1).
parameter(
  fast = 1000
).
```

In [21]:
```
numerical_simulation. plot.
```



Out[21]:

In [22]:
```
numerical_simulation(method:ssa). plot.
```



Out[22]:

CRN synthesis for computing cosine(x)

In [23]:
```
clear_model.
```

Out[23]:

In [24]:
```
compile_from_expression(cos, x, f).
```

Out[24]:

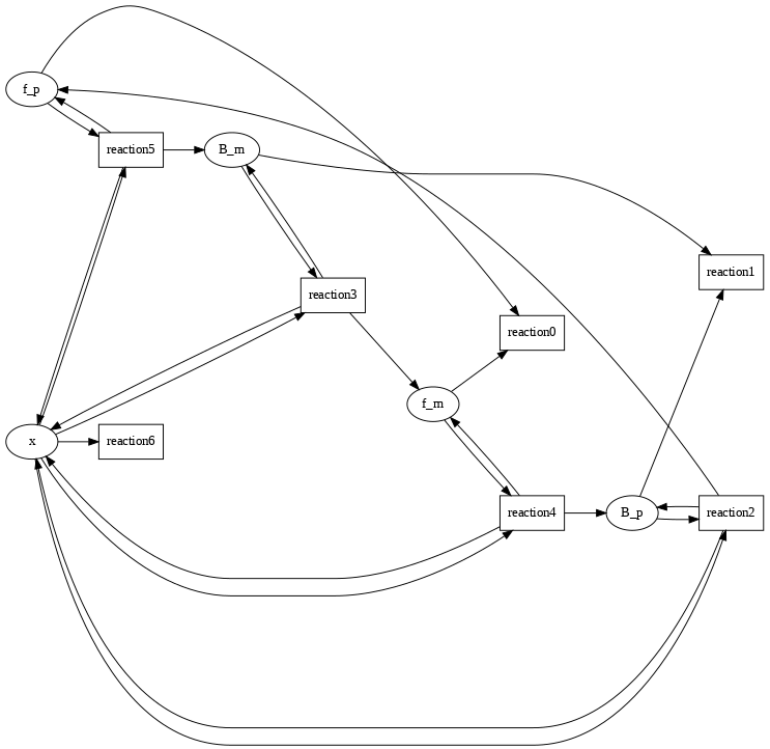
In [25]:
```
parameter(input=4).
```

Out[25]:

In [26]:
```
list_model.
```

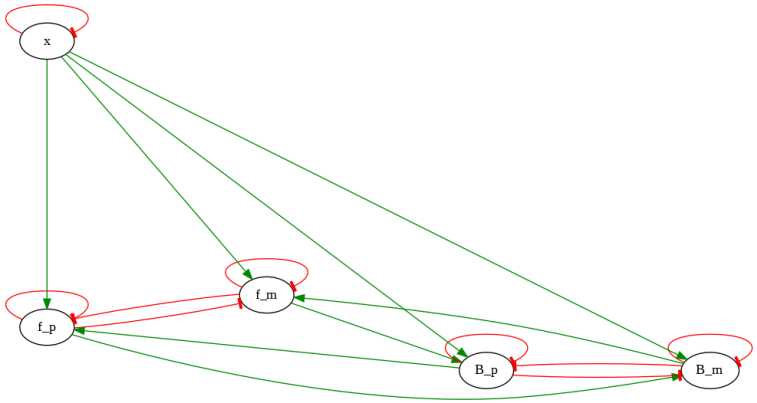
Out[26]:
```
MA(fast) for f_m+f_p=>_.
MA(fast) for B_m+B_p=>_.
MA(1.0) for B_p+x=>B_p+f_p+x.
MA(1.0) for B_m+x=>B_m+f_m+x.
MA(1.0) for f_m+x=>B_p+f_m+x.
MA(1.0) for f_p+x=>B_m+f_p+x.
MA(1.0) for x=>_.
initial_state(f_p=1).
initial_state(x=input).
parameter(
  fast = 1000,
  input = 4
).
```

In [27]:
```
draw_reactions.
```



Out[27]:

In [28]:
```
draw_influences.
```



Out[28]:

In [29]:
```
list_ode.
```

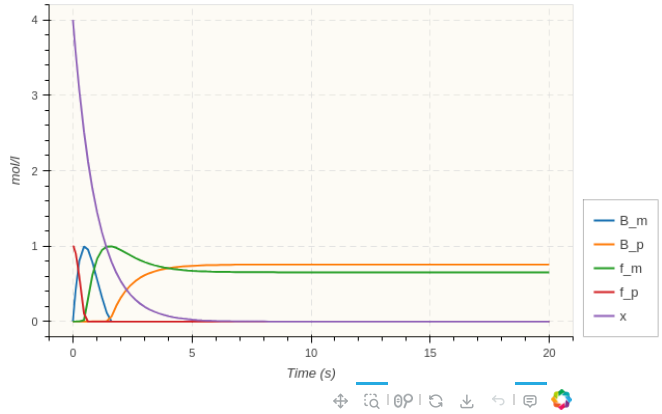
$$
\begin{aligned}
x_0 &= 4 \\
B\_m_0 &= 0 \\
B\_p_0 &= 0 \\
f\_m_0 &= 0 \\
f\_p_0 &= 1 \\
fast &= 1000 \\
input &= 4 \\
\frac{dx}{dt} &= -x \\
\frac{dB\_m}{dt} &= f\_p * x - B\_m * B\_p * fast \\
\frac{dB\_p}{dt} &= f\_m * x - B\_m * B\_p * fast \\
\frac{df\_m}{dt} &= B\_m * x - f\_m * f\_p * fast \\
\frac{df\_p}{dt} &= B\_p * x - f\_m * f\_p * fast
\end{aligned}
$$

Out[29]:

In [30]:
```
numerical_simulation.
```

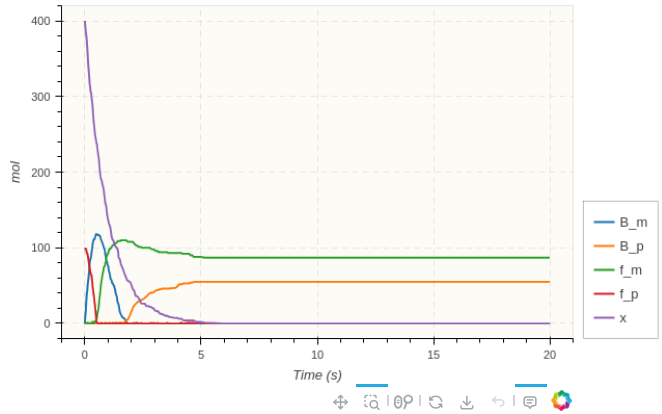
Out[30]:

In [31]:
```
plot.
```



Out[31]:

In [32]:
```
numerical_simulation(method:ssa). plot.
```



Out[32]:

General CRN Compilation Pipeline for Elementary Mathematical Functions

More generally, Biocham can compile any elementary mathematical function into a finite CRN using the following compilation pipeline:

1. symbolic differentiation of the function as a function of time
2. initial ODE that has the function as solution
3. polynomialization of the ODE

- by introduction of new variables for non monomial terms
- guaranteed to terminate on elementary functions

4. quadratization of the ODE

- by introduction of new variables for non quadratic monomials
- minimizing the number of introduced species is NP-hard
- SAT solver used

5. CRN synthesis from quadratic ODE with dual-rail encoding of negative values

In [33]:
```
option(quadratic_reduction:sat_species).
```

Out[33]:

In [34]:
```
clear_model.
compile_function(F = cos(X), X).
list_model.
```

Out[34]:
```
A = sin(t)
F = cos(t)

MA(fast) for A_m+A_p=>_.
MA(fast) for F_m+F_p=>_.
MA(1.0) for F_p+X=>A_p+F_p+X.
MA(1.0) for F_m+X=>A_m+F_m+X.
MA(1.0) for X=>t.
MA(1.0) for A_m+X=>A_m+F_p+X.
MA(1.0) for A_p+X=>A_p+F_m+X.
initial_state(A_p=0.0).
initial_state(F_p=1.0).
initial_state(X=input).
parameter(
  input = 1.0,
  fast = 1000
).
```

In [35]:
```
clear_model.
compile_function(H1 = X/(1+X), X).
list_model.
```

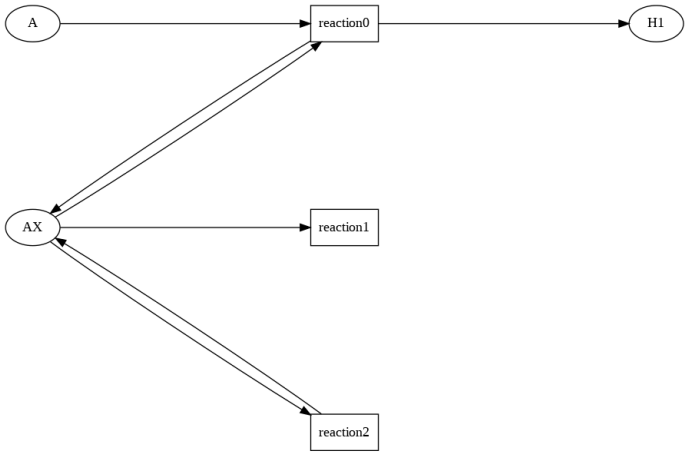
Out[35]:
```
A = 1/(1+t)
H1 = t/(1+t)

MA(1.0) for A+AX=>AX+H1.
MA(1.0) for AX=>_.
MA(1.0) for 2*AX=>AX.
initial_state(A=1).
initial_state(AX=input).
parameter(
  fast = 1000,
  input = 1.0
).
```

In [36]:
```
search_conservations.
```

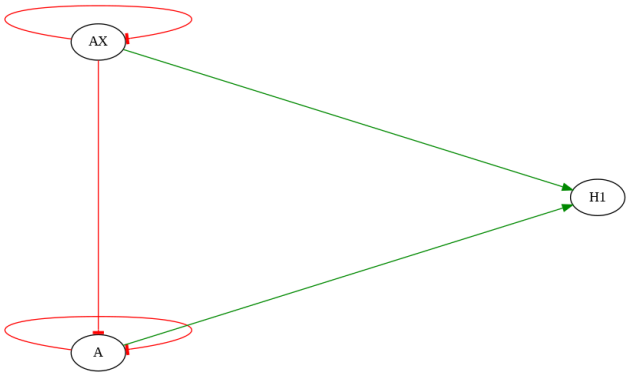
Out[36]:
```
A+H1
1 complex invariant(s)
```

In [37]:
```
draw_reactions.
```



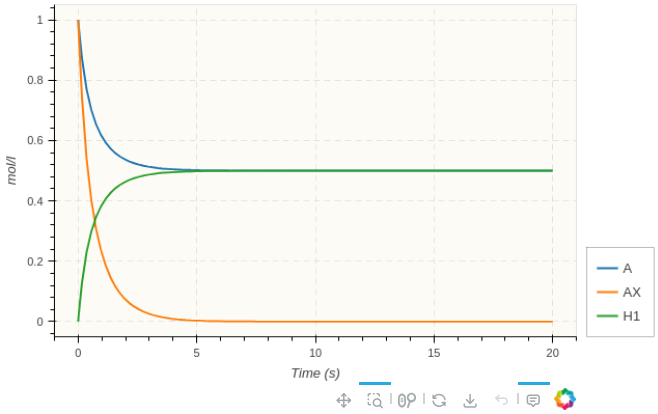
Out[37]:

In [38]:
```
draw_influences.
```



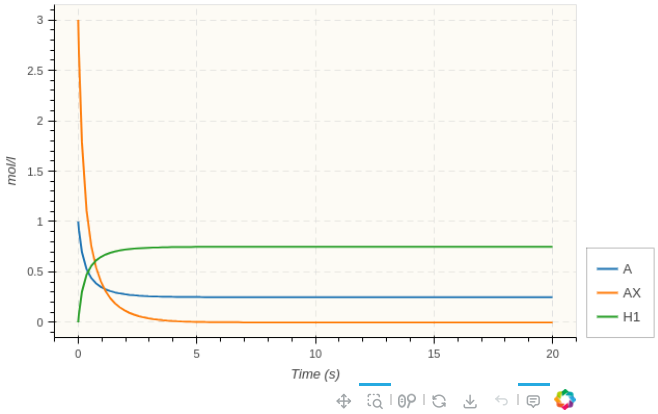
Out[38]:

In [39]:
```
numerical_simulation. plot.
```



Out[39]:

In [40]:
```
parameter(input=3).
numerical_simulation. plot.
```



Out[40]:

Hill 5: synthetic CRN analog of MAPK signaling input/ouput function

In [41]:
```
clear_model.
compile_function(H5 = X^5/(1+X^5), X).
list_model.
```

Out[41]:

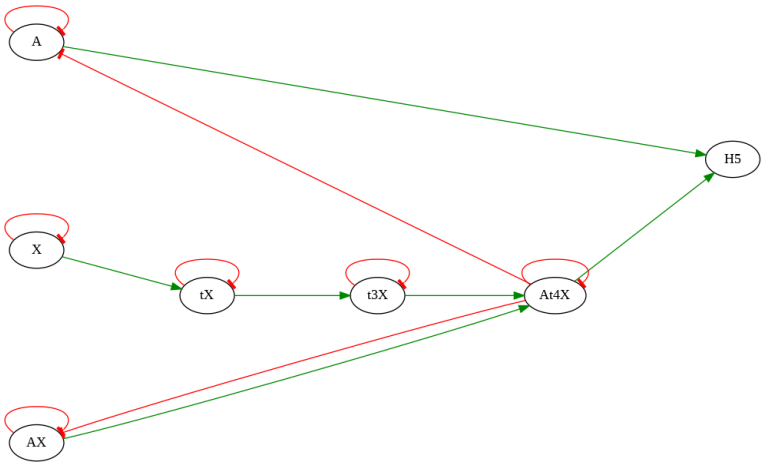
```
A = 1/(1+t^5)
H5 = t^5/(1+t^5)

MA(1.0) for X=>_.
MA(5.0) for A+At4X=>At4X+H5.
MA(1.0) for 2*X=>tX+2*X.
MA(1.0) for tX=>_.
MA(3.0) for 2*tX=>t3X+2*tX.
MA(1.0) for t3X=>_.
MA(1.0) for AX=>_.
MA(5.0) for AX+At4X=>At4X.
MA(4.0) for AX+t3X=>AX+At4X+t3X.
MA(1.0) for At4X=>_.
MA(5.0) for 2*At4X=>At4X.
initial_state(X=input).
initial_state(A=1).
initial_state(AX=input).
parameter(
  fast = 1000,
  input = 1.0
).
```

In [42]:
```
search_conservations.
```

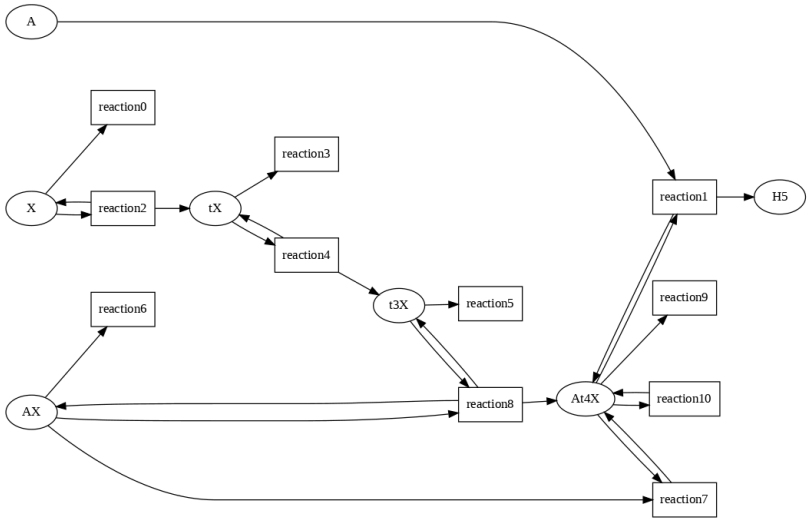
Out[42]:
```
A+H5
1 complex invariant(s)
```

In [43]:
```
draw_influences.
```



Out[43]:

In [44]:
```
draw_reactions.
```

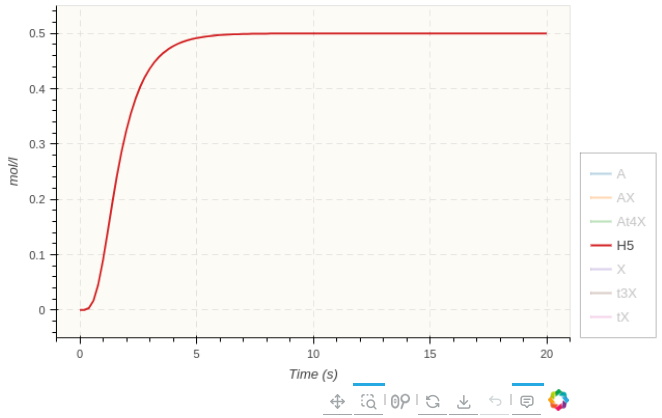


Out[44]:

In [45]:
```
search_conservations.
```

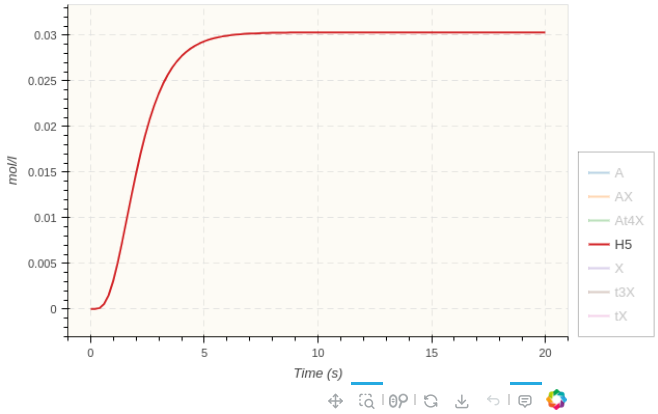
Out[45]:
```
A+H5
1 complex invariant(s)
```

In [46]:
```
numerical_simulation. plot(show:{H5}).
```



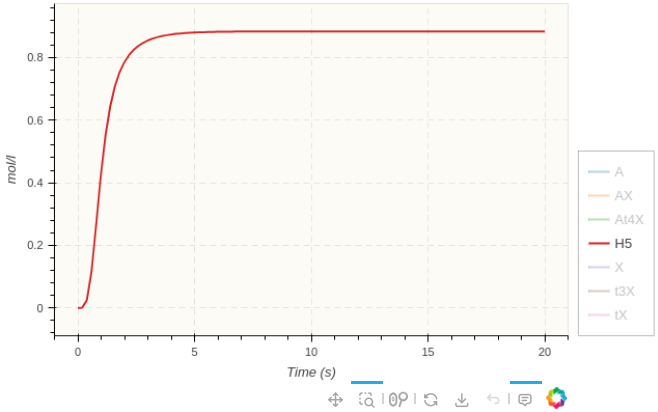
Out[46]:

In [47]:
```
parameter(input=0.5).
numerical_simulation. plot(show:{H5}).
```



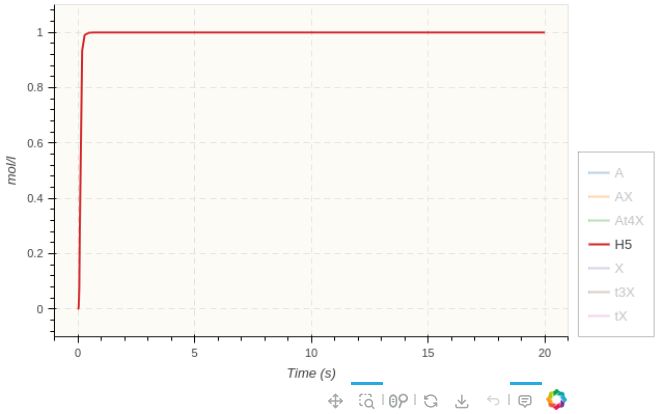
Out[47]:

In [48]:
```
parameter(input=1.5).
numerical_simulation. plot(show:{H5}).
```



Out[48]:

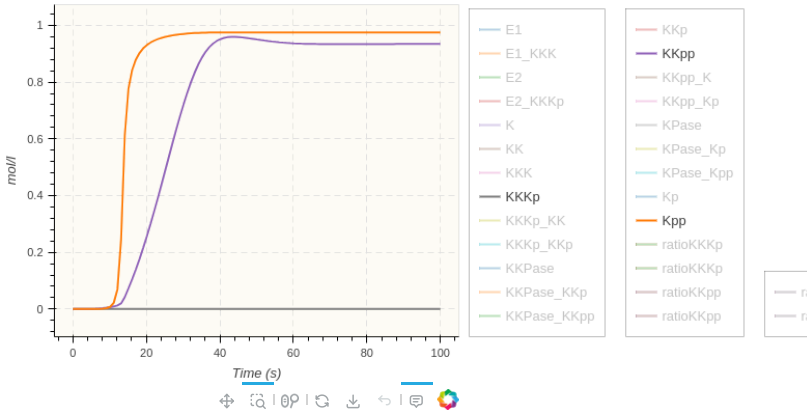
In [49]:
```
parameter(input=10).
numerical_simulation. plot(show:{H5}).
```



Out[49]:

Trajectories toward output stabilization

In [5]:
```
numerical_simulation.
plot.
```



Out[5]:

Behavioural Specifications in Quantitative Temporal Logic FO-LTL(Rlin)

- *First-Order Linear Time Logic* with free variables and *linear constraints* over the reals
- Real-valued states (concentrations of molecular species)
- Bounded model-checking: validity of LTL formulae on a finite trace completed with a loop on the last state

Validity domains of FOLTL(Rlin) constraint variables

- FO-LTL(Rlin) constraint solving: validity domain for variables

In [6]:
```
validity_domain(F(Kpp = v)). % all values in the trace
```

Out[6]:
```
v=0\/v=3.68e-10\/v=4.9367e-8\/v=8.82573e-7\/v=6.78262e-6\/v=3.31004e-5\/v=0.000122963\/v=0.000385086\/v=0.00108711\/v=0.0029129\/v=0.00777349\/v=0.0218366\/v=0.0690836\/v=0.245708\/v=0.613048\/v=0.7764\/v=0.842407\/v=0.879194\/v=0.902637\/v=0.918798\/v=0.930532\/v=0.939371\/v=0.946214\/v=0.951623\/v=0.95597\/v=0.959507\/v=0.962414\/v=0.964821\/v=0.966823\/v=0.968491\/v=0.969882\/v=0.971038\/v=0.971992\/v=0.972774\/v=0.973407\/v=0.973911\/v=0.974306\/v=0.974608\/v=0.974834\/v=0.974997\/v=0.97511\/v=0.975183\/v=0.975226\/v=0.975245\/v=0.975246\/v=0.975234\/v=0.975213\/v=0.975186\/v=0.975155\/v=0.975123\/v=0.975089\/v=0.975056\/v=0.975025\/v=0.974995\/v=0.974967\/v=0.974942\/v=0.97492\/v=0.9749\/v=0.974882\/v=0.974866\/v=0.974853\/v=0.974842\/v=0.974833\/v=0.974825\/v=0.974819\/v=0.974814\/v=0.974802\/v=0.974803\/v=0.974804\/v=0.974805\/v=0.974806\/v=0.974807\/v=0.974808\/v=0.974809\/v=0.97481\/v=0.974811
```

In [7]:
```
validity_domain(F(G(Kpp = v))). % last value of Kpp on the trace (stable value)
```

Out[7]:
```
v=0.974811
```

In [8]:
```
% times of last value
validity_domain(F(Time = t /\ G(Kpp = v))).
```

Out[8]:
```
t=93.1372/\v=0.974811\/t=94.1372/\v=0.974811\/t=95.1372/\v=0.974811\/t=96.1372/\v=0.974811\/t=97.1372/\v=0.974811\/t=98.1372/\v=0.974811\/t=99.1372/\v=0.974811\/t=100/\v=0.974811
```

In [9]:
```
% time bound for last value
validity_domain(F(Time <= t /\ G(Kpp = v))).
```

Out[9]:
```
t>=93.1372/\v=0.974811
```

In [10]:
```
% switching time
validity_domain(F(Time <= t /\ G(Kpp > 0.5))).
```

Out[10]:
```
t>=14.1372
```

Satisfaction degree of FOLTL(Rlin) formulae

- FOLTL(Rlin) constraint given objective values for the free variables
- satisfaction degree in [0,1] as the inverse of the distance between the objective point and the validity domain
- satisfaction degree in[1, +inf[as the penetration depth of the objective point in the validity domain (i.e. formula robustness)

In [11]:
```
satisfaction_degree(F(Time < t /\ G(Kpp > 0.5)), [t->14.1372]).
```

Out[11]:
```
1.000000
```

In [12]:
```
satisfaction_degree(F(Time < t /\ G(Kpp > 0.5)), [t->10]).
```

Out[12]:
```
0.194659
```

In [13]:
```
satisfaction_degree(F(Time < t /\ G(Kpp > 0.5)), [t->20]).
```

Out[13]:
```
6.862400
```

Robustness of FOLTL(Rlin) formulae with respect to parameter variations (i.e. extrinsic variability)

- Here robustness of the switching time property with respect to variation of the input concentration
- Default coefficient of variation is 0.1 (i.e. variation of parameter values by 10%)

In [14]:
```
present(E1, e1).
parameter(e1=3e-5).
```

Out[14]:

In [15]:
```
seed(0).
robustness(F(Time < t /\ G(Kpp > 0.5)), [e1], [t-> 14.2459]).
```

Out[15]:
```
 On [e1] robustness degree: 0.812053 sensitivity: 0.187947  (computation time 9.284000 s)
```

In [16]:
```
seed(0).
robustness(F(Time < t /\ G(Kpp > 0.5)), [e1], [t-> 14.2459], robustness_coeff_var: 0.2).
```

Out[16]:
```
 On [e1] robustness degree: 0.755177 sensitivity: 0.244823  (computation time 5.057000 s)
```

In [17]:
```
seed(0).
robustness(F(Time < t /\ G(Kpp > 0.5)), [e1], [t-> 14.2459], robustness_coeff_var: 0.3).
```

Out[17]:
```
 On [e1] robustness degree: 0.694281 sensitivity: 0.305719  (computation time 4.906000 s)
```

Robustness w.r.t. variation of kinetic parameters by 10%

In [18]:
```
seed(0).
robustness(F(Time < t /\ G(Kpp > 0.5)), [a1, c1, a2, c2], [t-> 14.2459]).
```

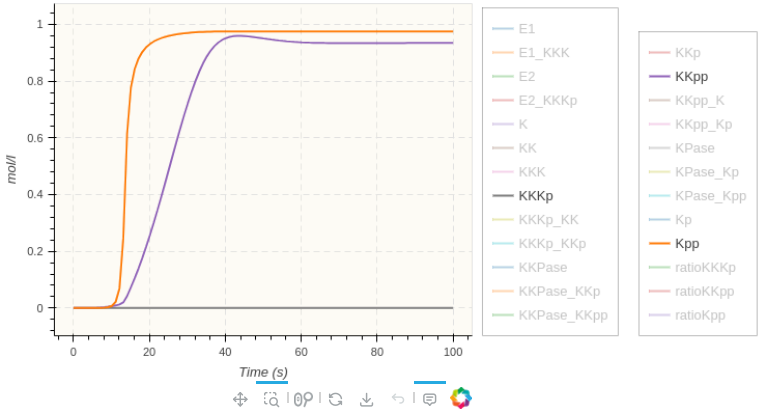
Out[18]:
```
On [a1,c1,a2,c2] robustness degree: 0.825163 sensitivity: 0.174837  (computation time 7.913000 s)
```

Parameter search for satisfying an FOLTL(Rlin) constraint with objective values

- Explores the search space of parameters in order to satisfy a FOLTL(Rlin) formula
- Uses a black-box continuous optimization algorithm to maximize the satisfaction degree
- BIOCHAM uses CMAES (covariance matrix adaptive evolution strategy)

E.g. find kinetic parameters a1, c1, a2, c2 for having a switching time before 10

In [19]:
```
load(BIOMD9renamedCTLreduced.bc).
option(method: bsimp).
numerical_simulation. plot.
```

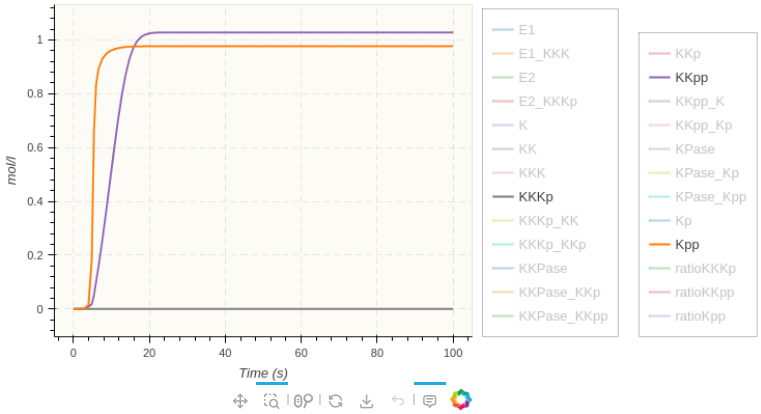


Out[19]:

In [20]:
```
seed(0).
search_parameters(F(Time < t /\ G(Kpp > 0.5)),
    [10<=a1<=1e4, 10<=c1<=1e4, 10<=a2<=1e4, 10<=c2<=1e4],
    [t-> 10]).
```

Out[20]:
```
Time: 3.395 s
Stopping reason: Fitness: function value -8.21e-01 <= stopFitness (1.00e-04)
Best satisfaction degree: 5.584296
[0] parameter(a1=7674.210850413168)
[1] parameter(c1=5374.014748948738)
[2] parameter(a2=2795.4498499859073)
[3] parameter(c2=667.868493333378)
```

In [21]:
```
numerical_simulation. plot.
```



Out[21]:

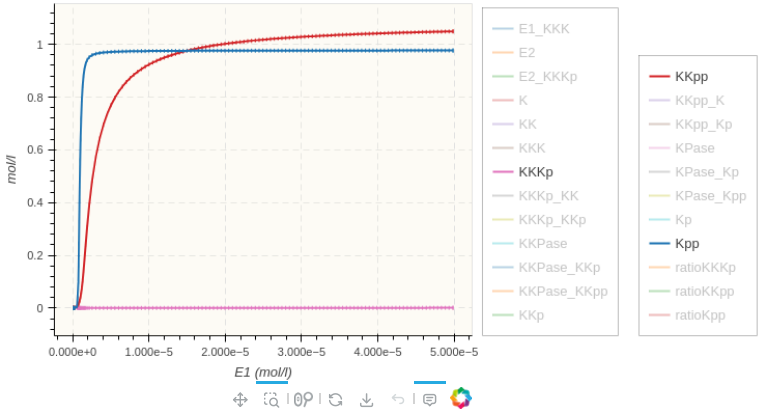
In [22]:
```
% switching time
validity_domain(F(Time <= t /\ G(Kpp > 0.5))).
```

Out[22]:
```
t>=5.4157
```

Dose-response diagram

- with a long time horizon for reaching the stable state with low values for E1

In [23]:
```
option(time:1000).
dose_response(E1, 1e-7, 5e-5).
```



Animate

Out[23]: