The Cell Cycle Switch Computes Approximate Majority - Supplement

Luca Cardelli[1], Attila Csikász-Nagy[2]

[1] Microsoft Research Cambridge, 7 J J Thomson Ave Cambridge CB3 0FB, UK
[2] The Microsoft Research-University of Trento Centre for Computational and Systems Biology, Piazza Manufattura , Rovereto-38068, Italy

Correspondence to:
- Attila Csikász-Nagy, The Microsoft Research-University of Trento Centre for Computational and Systems Biology, Piazza Manufattura , Rovereto-38068, Italy, email: csikasz@cosbi.eu.
- Luca Cardelli, Microsoft Research Cambridge, 7 J J Thomson Ave Cambridge CB3 0FB, UK, email: luca@microsoft.com.

Table of Contents:

Supplementary 1: Comparison of the CC model with earlier cell cycle switch and cell cycle oscillator models

The diagram on the left of Figure S1-1 is a reproduction of Fig 1B, M-phase control system, from [1] The blue letters in the left and right diagrams illustrate the one-to-one correspondence with the CC network studied in this paper (Fig. 1d.1). The feedback loops are indicated in color. In these diagrams we use a hollow-ball, informally, to indicate catalytic activity that may be achieved by various mechanisms.

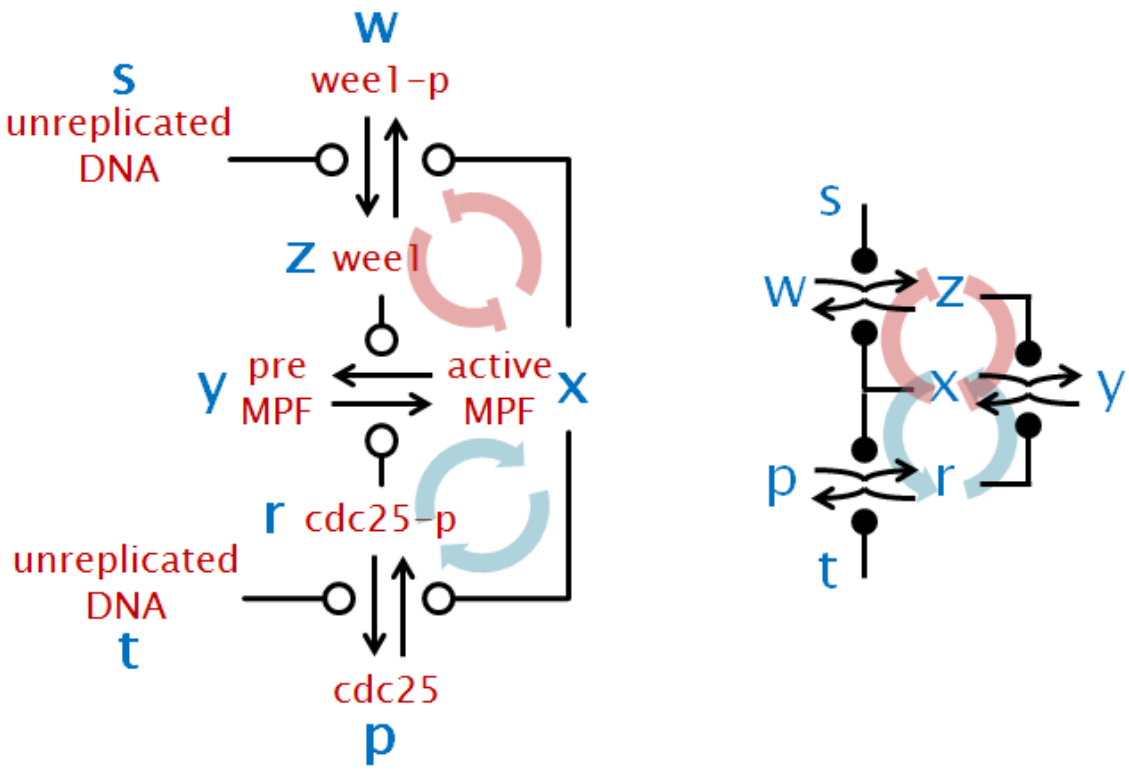


Figure S1-1: The cell cycle switch of the Novak - Tyson model and the CC model

The diagram in Figure S1-2 is a summary of the model of the cell cycle oscillator from [1], where cyclin (c) influx and degradation are integrated in a more complex way into the main switch (*Cf*. Fig. 3c), and where the second switch has a different structure while still providing a similarly connected negative feedback to the main switch.

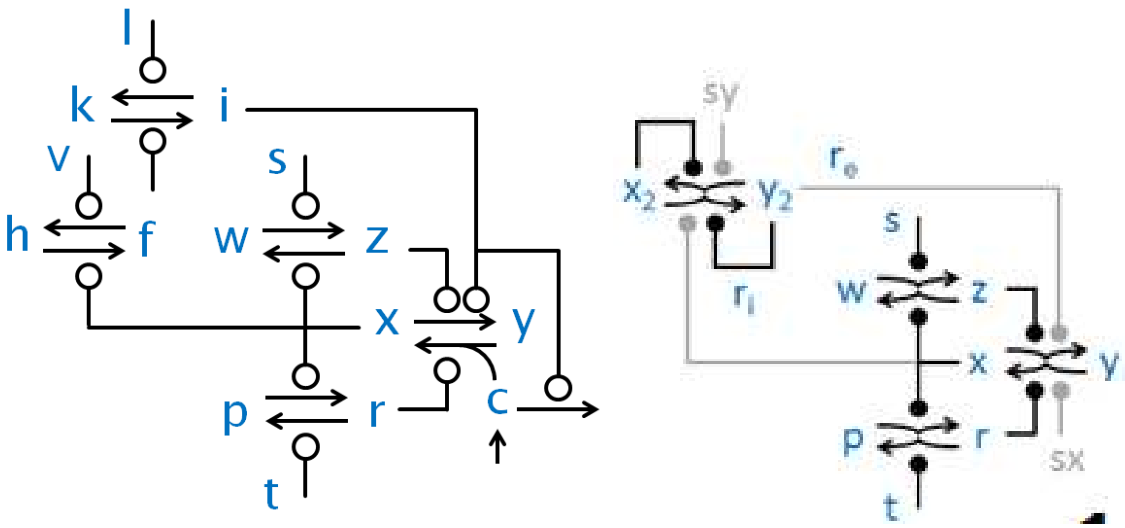


Figure S1-2: The cell cycle oscillator of the Novak-Tyson model and the CC-AM oscillator. The left panel is a representation of the original model of Novak and Tyson [1], the right panel is the CC-AM inner model from Fig 3c. Biological interpretation of *s*, *w*, *z*, *x*, *y*, *p*, *r*, *t* are given on figure S1-1, *c* represents free cyclin, *h*, *f*, *k*, *i*, *v*, *l* are representing the different forms and regulators of *IE* and *UbE* variables of the original model.

Supplementary 2: Comparing convergence times at different system sizes.

In the original AM algorithm [2] it is shown that in a system AM_N with $N = x + b + y$ molecules, convergence is obtained with high probability within an asymptotic bound of $O(N \log N)$ *reactions*, when counting also unproductive *collision* reactions of the form $x + x \rightarrow x + x$ (which are included for technical reasons). In the execution model, on the order of N such reactions can happen in parallel within a time interval, so that the convergence bound is $O(\log N)$ in *parallel time*, which is simply the number of reactions divided by the number of molecules. Seen as a chemical system, increasing the size N and still assuming that N reactions happen in parallel is equivalent to assuming a constant concentration, and hence assuming that the volume inhabitated by the molecules increases with N. The volume increase is in turn equivalent to reducing the stochastic rates (which have dimension $seconds^{-1}$) so that the overall propensity of the reactions (the stochastic rate times the number of possible reactions) remains the same. In biological terms, this means that, e.g., if a cell grows but manages to keep the concentration of the components of an AM-like algorithm constant, it experiences a slowdown in decision time of only $O(\log N)$.

The unproductive collisions mentioned above have no kinetic effect in the chemical interpretation (both in the deterministic interpretation because of ODE mass-action kinetics where they cancel out, and in the stochastic interpretation because of the Markov memory-less assumption) and can be ignored. The algorithm thus reduces to the four chemical reactions we have discussed. When considering also the collisions, it is still the case that in the chemical interpretation on the order of N reactions happen in parallel, and the authors remark that the same bound of $O(\log N)$ convergence time holds (at constant concentration). .

Simulations of AM show that the expected (mean) number of reactions is lower than the asymptotic bound, but follows the same law. For example, in the worst case of $x = y$ initial conditions, almost all AM runs complete in ~4 $N \log N$ reactions, indicating where the true bound lies. But the expectation is seen to be ~3 $N \log N$ reactions [2]. We can therefore talk equivalently about the bound or the expectation, up to a constant, both having the form $k\, N \log N$.

In this paper we have taken an equivalent but different point of view about system scaling. We keep the stochastic rates fixed (at 1.0) when varying N: the result is that increasing N increases the reaction propensities and hence speeds up the convergence of the algorithm. Effectively, by increasing N and keeping the rates fixed, we keep our interaction volume fixed and hence increase the concentration. In particular, since all reactions are bimolecular, increasing N without changing the rates causes the propensity of all reactions to increase as N^2. Because of the speedup in propensities, our convergence time is not $O(\log N)$ as in the previous case, but $O(N^{-1} \log N)$ in the fixed volume. Nothing deep is changing by adopting this point of view: just the choice of the quantities to keep fixed.

Therefore, according to our $O(N^{-1} \log N)$ time bounds, for a factor k depending on the initial conditions, the expected or high-probability convergence time in a *fixed volume* for a system of size 15 is $E(AM_{15}) = k\, 15^{-1} \log 15$, and similarly for $E(AM_{15000})$. The ratio between those numbers is $E(AM_{15000}) / E(AM_{15}) = 0.00355$, independently of k which cancels out. We use this ratio to calibrate the time scales on the horizontal axis of Fig. 1b.3, obtaining good agreement between stochastic simulations at size 15000 and probability distributions at size 15. For Fig. 1c and Fig. 1d it is instead our *hypothesis* that SC and CC behave similarly to AM, and that therefore should have a bound of $O(N^{-1} \log N)$. We use the same ratio there too, which is independent of the k for those systems (note that convergence is not so clearly-cut for these systems, so k would be more difficult to measure). The fact that plots at different system sizes still match up supports our hypothesis. For Fig. 1(a) we instead

have an algorithmic bound of $O(N^2)$ reactions to convergence, since the system performs an unbiased random walk in state space. Again, under our assumptions, the reaction propensities grow as N^2 so that the ratio of convergence times at different system sizes is 1.0: no rescaling in performed here for different system sizes. Note therefore that the rescaling of the different systems is based on the *algorithmic* properties of the systems.

Supplementary 3: Circuits with a single feedback loop

All the systems under consideration in the main body of the paper contain two positive feedback loops. In nature there are several systems where a single positive feedback can lead to proper bistability. The necessity of the presence of both feedbacks in the cell cycle switch has been investigated [3,4] and it has been shown that the presence of two feedbacks makes the switch more robust. It is then interesting to see how AM and CC compare in absence of one of the feedback loops.

We see that bistability remains if we remove either positive feedback loops from the AM or the CC models (Fig. S3-1). Still, we have to fine-tune the parameters to find these bistable solutions, and even so the width of the bistable region is greatly reduced. We further observe that the behaviors of AM and CC are similar even when one of their feedback loops are removed. When the double negative feedback loop is removed from the CC model we see an even larger decrease in the maximal value *x* can reach as a result of a constantly active *z* in this system (Fig. S3-1d). The main problem with the system with only double negative feedback (Fig. S3-1c) is that *x* and *z* act on each other, thus when they interact there could be two different outcomes of the reaction. Either *x* converts *z* to *w* or *z* converts *x* to *y*. Such type of direct antagonism leads to reduced or even to total absence of bistability [5].

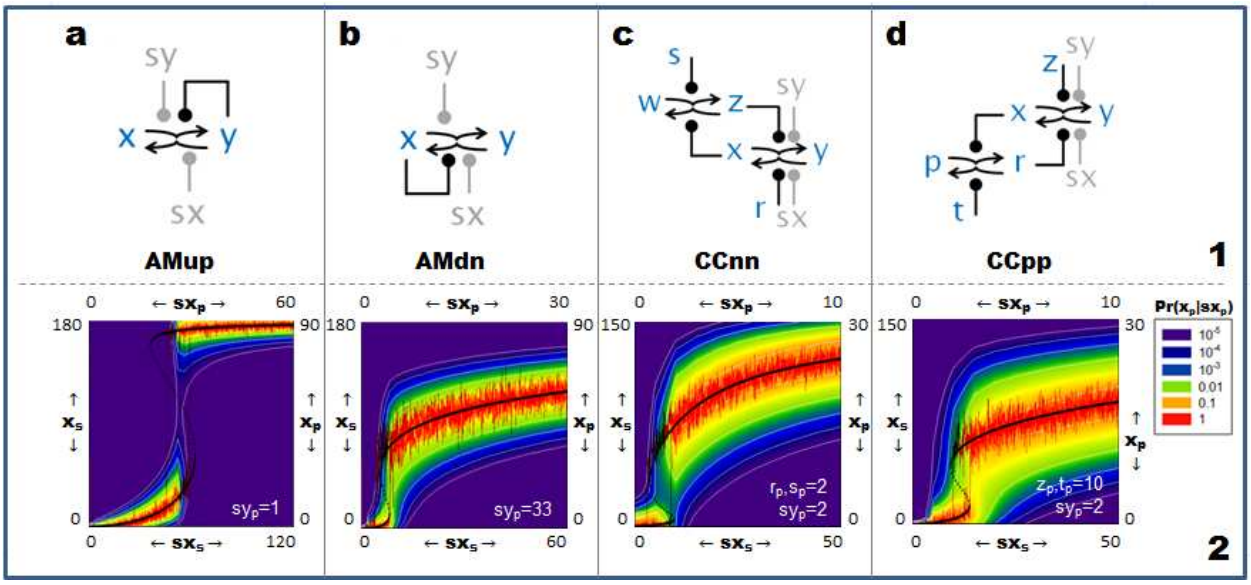


Figure S3-1. Circuits with a single feedback loop. The plot axes are similar to those of Fig. 2, but we have obtained a higher resolution in the heat maps because of the smaller system sizes.

Regions of bistability for AM, AMup, AMdn models

The reduced bistability in the models with single feedback loop can be nicely visualized in a two parameter bifurcation diagram. On Fig. S3-2 we show the parameter regimes where each of the three models (AM, AMup and AMdn) shows bistability. AM shows bistability inside the blue curve (what labels the positions of saddle-node bifurcations), meaning that any combination of *sx* and *sy* inside this area would lead to a bistable system. The two other models work as a bistable switch only in a far smaller parameter regime as any parameter combination out of the colored regions lead to monostable systems. Plots created by Oscill8 (http://sourceforge.net/projects/oscill8/).

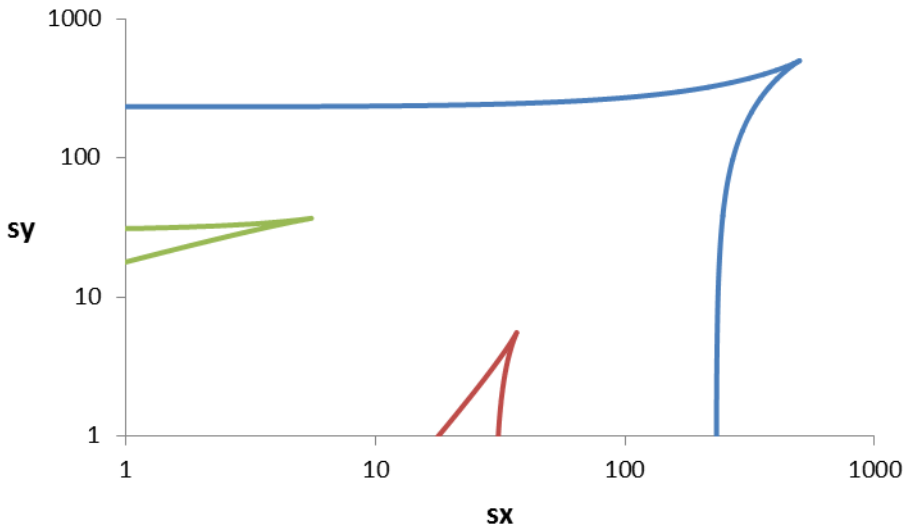


Figure S3-2. Regions of bistability for AM, AMup, AMdn This bifurcation diagram describes the regions of bistability, on the inside of the cusps, of AM (blue), AMup (red) and AMdn (green).

Supplementary 4: Mechanical Oscillators

We describe two mechanical oscillators that can be seen as analogous to the chemical oscillators discussed in the main text.

The Trammel of Archimedes (Fig S4-1) is a device used since antiquity to draw ellipses [6]. As the handle is moved in circular motion, the two 'switches' placed inside the grooves obey the following state transitions, where each switch extremal state (e.g.. x_1 in (1)) remains relatively fixed while inducing a trasition in the other switch (e.g., $x_2 \rightarrow y_2$ in (1)). This is the same pattern of connections and state transitions as in Fig. 3a, where the switches are AM switches.

$x_2 + x_1 \rightarrow x_1 + y_2$ (1)
$x_1 + y_2 \rightarrow y_2 + y_1$
$y_2 + y_1 \rightarrow y_1 + x_2$
$y_1 + x_2 \rightarrow x_2 + x_1$

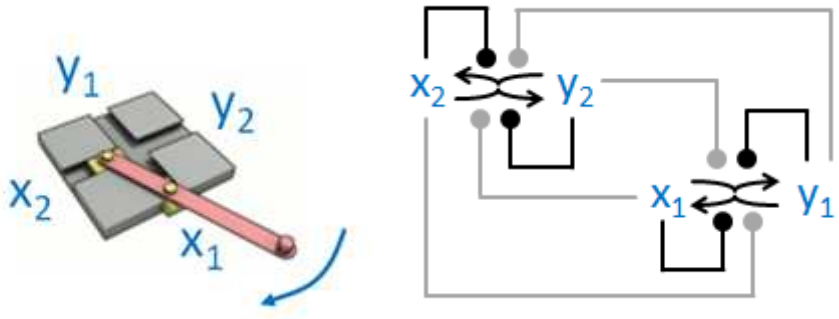


Fig S4-1: The Trammel of Archimedes
http://commons.wikimedia.org/wiki/File:Trammel_of_Archimedes_Small_White.gif
This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

The Shishi-Odoshi (Fig S4-2) is a Japanese device used to scare animals by sudden noise and motion. Its recurring pattern is similar to the Trammel, but two connections are replaced by unregulated inputs: a continuous flow of water to fill the pipe when it is empty and pointing up, and the force of gravity on an off-center pivot to cause the pipe to point up when it is empty and pointing down. The other two transitiosn are as above. This is the pattern of connections and state transitions form Fig. 3b. The Shishi-Odoshi has been used as a paradigm for the cell-cycle osciallator [7,8].

x_1 = pipe-full, y_1 = pipe-empty, x_2 = mouth-up, y_2 = mouth-down,
sx = water-flow, sy = counterbalance.

$x_2 + x_1 \rightarrow x_1 + y_2$ (x_1=pipe-full causes mouth-up $\rightarrow$ mouth-down)
$x_1 + y_2 \rightarrow y_2 + y_1$ (y_2=mouth-down causes pipe-full $\rightarrow$ pipe-empty)
$y_2 + sy \rightarrow sy + x_2$ (sy=counterbalance causes mouth-down $\rightarrow$ mouth-up)
$y_1 + sx \rightarrow sx + x_1$ (sx=water-flow causes pipe-empty $\rightarrow$ pipe-full)

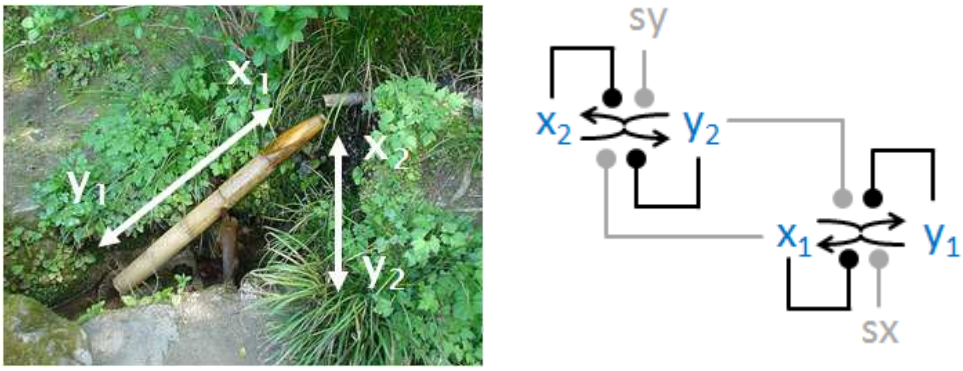


Figure S4-2: The Shishi-Odoshi
http://upload.wikimedia.org/wikipedia/commons/1/1c/Shisendo_Souzu.jpg
This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

Supplementary 5: Greatwall variations

Fig. S5-1b shows a switching speed comparison between GW (Fig. 5a) and GW-Indirect (Fig. S5-1a). A sample of three stochastic upper trajectories and one lower trajectory is given for each system. In both systems, all species start at the same level (5000) but with v=0, and all rates set at 1.0. GW-Ind shows the same speedup as GW with respect to CC.

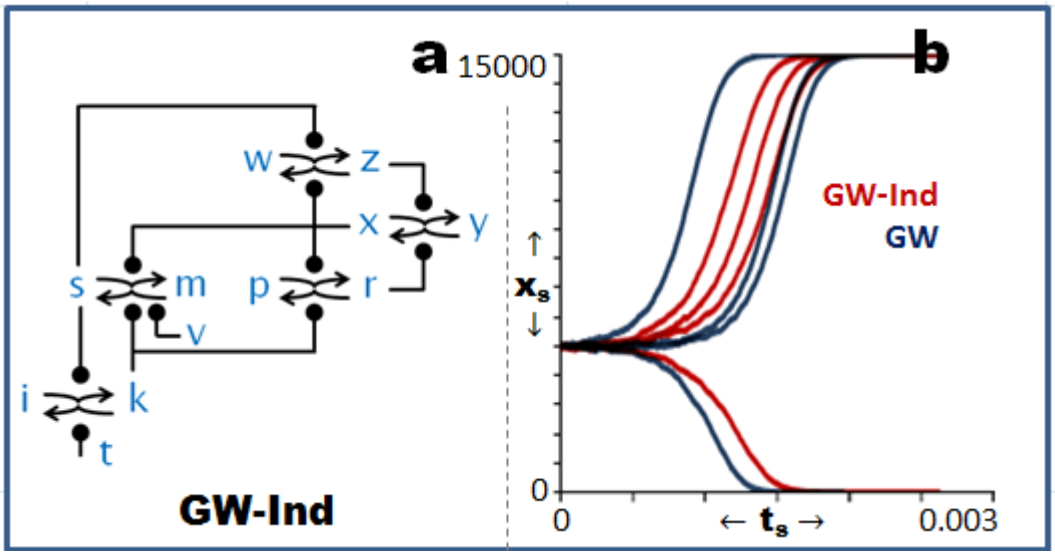
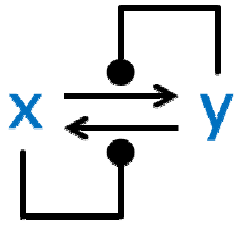


Figure S5-1. Oscillator with indirect Greatwall feedback

Supplementary 6: Models

DC (Fig 1a)



Chemical Reaction Model

x + y → y + y
y + x → x + x

Deterministic Model

```
dx/dt = (x+sx)*y - (y+sy)*x
y = xby-x

xby=1500, sy=0, sx=0
```

XPPAUT / Oscill8 Source (.ode)

```
p xby=1500
p sy=0, sx=0
y = xby-x
dx/dt = (x+sx)*y - (y+sy)*x
aux y=y
```

SPiM Source (.spi)

```
directive sample 1.0 1000
directive plot x()
val r = 1.0
new xcat@r:chan
new ycat@r:chan
let x() =
  do !xcat; x()
  or ?ycat; y()
and y() =
  do !ycat; y()
  or ?xcat; x()
run 7000 of x()
run 8000 of y()
```

PRISM Source

Model file (.sm)

```
ctmc

const int xMax = xInit+yInit;
const int yMax = xInit+yInit;
const int sxMax =  xInit+yInit;
const int syMax = xInit+yInit;

const int xInit;
const int yInit;
const int sxInit;
const int syInit;

const double rt;

module DC

x : [0..xMax] init xInit;
y : [0..yMax] init yInit;
sx : [0..sxMax] init sxInit;
sy: [0..syMax] init syInit;

// X + Y -> Y + Y
[rt] x>0 & y>0 -> (rt*x*y) : (x'=x-1) &
(y'=min(y+1, yMax));
// Y + X -> X + X
[rt] y>0 & x>0 -> (rt*y*x) : (y'=y-1) &
(x'=min(x+1, xMax));

// X + SY -> SY + Y
[rt] x>0 & sy>0 -> (rt*x*sy) : (x'=x-1) &
(y'=min(y+1, yMax));
// Y + SX -> SX + X
[rt] y>0 & sx>0 -> (rt*y*sx) : (y'=y-1) &
(x'=min(x+1, xMax));

endmodule

rewards "x" true : x; endrewards
rewards "y" true : y; endrewards
rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards

rewards "time" true : 1; endrewards
```

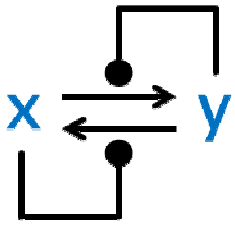
Query file (.csl)

```
const double T;
const int i;
P=? [F[T,T]x=i]
```

Initial Parameters

```
xInit = 7
yInit = 8
rt = 1.0
i = 0 .. 15 by 1
T = 0 .. 1 by 0.05
(sxInit = syInit = 0)
```

AM (Fig 1b)



Chemical Reaction Model

x + y → y + b
y + x → x + b
b + x → x + x
b + y → y + y

Deterministic Model

```
dx/dt = (x+sx)*b - (y+sy)*x
db/dt = (x+sx)*y + (y+sy)*x - (y+sy)*b -
(x+sx)*b
y = xby-b-x

xby=1500, sy=0, sx=0
```

XPPAUT / Oscill8 Source (.ode)

```
p xby=1500
p sy=0, sx=0
```

```
y = xby-b-x
dx/dt = (x+sx)*b - (y+sy)*x
db/dt = (x+sx)*y + (y+sy)*x - (y+sy)*b -
(x+sx)*b
aux y=y
```

SPiM Source (.spi)

```
directive sample 0.00355 1000
directive plot x()
val r = 1.0
new xcat@r:chan
new ycat@r:chan
let x() =
   do !xcat; x()
   or ?ycat; b()
and y() =
   do !ycat; y()
   or ?xcat; b()
and b() =
   do ?xcat; x()
   or ?ycat; y()
run 5000 of x()
run 5000 of b()
run 5000 of y()
```

PRISM Source

Model file (.sm)

```
ctmc

const int xMax = xInit+yInit+bInit;
const int yMax = xInit+yInit+bInit;
const int bMax = xInit+yInit+bInit;
const int sxMax =  xInit+yInit+bInit;
const int syMax = xInit+yInit+bInit;

const int xInit;
const int yInit;
const int bInit;
const int sxInit;
const int syInit;

const double rt;

module AM

x : [0..xMax] init xInit;
y : [0..yMax] init yInit;
b : [0..bMax] init bInit;
sx : [0..sxMax] init sxInit;
sy : [0..syMax] init syInit;

// X + Y -> Y + B
[rt] x>0 & y>0 -> (rt*x*y) : (x'=x-1) &
(b'=min(b+1, bMax));
// Y + X -> X + B
[rt] y>0 & x>0 -> (rt*y*x) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + X -> X + X
[rt] b>0 & x>0 -> (rt*b*x) : (b'=b-1) &
(x'=min(x+1, xMax));
// B + Y -> Y + Y
[rt] b>0 & y>0 -> (rt*b*y) : (b'=b-1) &
(y'=min(y+1, yMax));

// X + SY -> SY + B
[rt] x>0 & sy>0 -> (rt*x*sy) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + SY -> SY + Y
[rt] b>0 & sy>0 -> (rt*b*sy) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + SX -> SX + B
[rt] y>0 & sx>0 -> (rt*y*sx) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + SX -> SX + X
[rt] b>0 & sx>0 -> (rt*b*sx) : (b'=b-1) &
(x'=min(x+1, xMax));

endmodule

rewards "x" true : x; endrewards
rewards "y" true : y; endrewards
rewards "b" true : b; endrewards
rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards

rewards "time" true : 1; endrewards
```

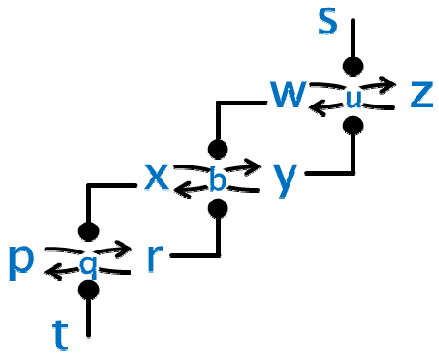
Query file (.csl)

```
const double T;
const int i;
P=? [ F[T,T] x=i ]
```

Initial Parameters

```
xInit = yInit = bInit = 5
rt = 1.0
i = 0 .. 15 by 1
T = 0 .. 1 by 0.05
(sxInit = syInit = 0)
```

SC (Fig 1c)



(w and z are swapped w.r.t. figure 1c)

Chemical Reaction Model

```
x + w → w + b
b + w → w + y
y + r → r + b
b + r → r + x

w + s → s + u
u + s → s + z
z + y → y + u
u + y → y + w

p + x → x + q
q + x → x + r
r + t → t + q
q + t → t + p
```

Deterministic Model

(t is time: we use tt for species t)

```
(p is parameter: we use pp for species p)
dz/dt = s*u - z*y
du/dt = s*w + z*y - s*u - y*u
dr/dt = x*q - tt*r
dq/dt = tt*r + x*pp - x*q - tt*q
dx/dt = (r+sx)*b - (w+sy)*x
db/dt = (r+sx)*y + (w+sy)*x - (w+sy)*b -
(r+sx)*b
w = wuz-z-u
pp = pqr-q-r
y = xby-b-x

wuz=1500, pqr=1500, xby=1500, s=500,
tt=500, sy=0, sx=0
```

XPPAUT / Oscill8 Source (.ode)

```
# t is time: we use tt for species t
# p is parameter: we use pp for species p
p wuz=1500, pqr=1500, xby=1500
p s=500, tt=500, sy=0, sx=0
w = wuz-z-u
pp = pqr-q-r
y = xby-b-x
dz/dt = s*u - z*y
du/dt = s*w + z*y - s*u - y*u
dr/dt = x*q - tt*r
dq/dt = tt*r + x*pp - x*q - tt*q
dx/dt = (r+sx)*b - (w+sy)*x
db/dt = (r+sx)*y + (w+sy)*x - (w+sy)*b -
(r+sx)*b
aux w=w
aux pp=pp
aux y=y
```

SPiM Source (.spi)

```
directive sample 0.00710 1000
directive plot x()
val rt = 1.0
new xcat@rt:chan
new ycat@rt:chan
new wcat@rt:chan
new rcat@rt:chan
new scat@rt:chan
new tcat@rt:chan
let x() =
   do !xcat; x()
   or ?wcat; b()
and y() =
   do !ycat; y()
   or ?rcat; b()
and b() =
   do ?rcat; x()
   or ?wcat; y()
and z() =
   ?ycat; u()
and r() =
   do !rcat; r()
   or ?tcat; q()
and w() =
  do !wcat; w()
  or ?scat; u()
and u() =
   do ?scat; z()
   or ?ycat; w()
and s() =
   !scat(); s()
and t() =
   !tcat; t()
and p() =
   ?xcat; q()
and q() =
   do ?xcat; r()
   or ?tcat; p()
run 5000 of x()
run 5000 of b()
run 5000 of y()
run 5000 of z()
run 5000 of u()
run 5000 of w()
run 5000 of p()
run 5000 of q()
run 5000 of r()
run 5000 of s()
run 5000 of t()
```

PRISM Source

Model file (.sm)

```
ctmc

const int xMax = xInit+yInit+bInit;
const int yMax = xInit+yInit+bInit;
const int bMax = xInit+yInit+bInit;

const int wMax = wInit+zInit+uInit;
const int zMax = wInit+zInit+uInit;
const int uMax = wInit+zInit+uInit;

const int pMax = pInit+rInit+qInit;
const int rMax = pInit+rInit+qInit;
const int qMax = pInit+rInit+qInit;

const int sMax = wInit+zInit+uInit;
const int tMax = pInit+rInit+qInit;
const int sxMax = xInit+yInit+bInit;
const int syMax = xInit+yInit+bInit;

const int xInit;
const int yInit;
const int bInit;

const int wInit;
const int zInit;
const int uInit;

const int pInit;
const int rInit;
const int qInit;

const int sInit;
const int tInit;
const int sxInit;
const int syInit;

const double rt;

module SC

x : [0..xMax] init xInit;
y : [0..yMax] init yInit;
b : [0..bMax] init bInit;

w : [0..wMax] init wInit;
z : [0..zMax] init zInit;
u : [0..uMax] init uInit;

p : [0..pMax] init pInit;
r : [0..rMax] init rInit;
q : [0..qMax] init qInit;

s : [0..sMax] init sInit;
t : [0..tMax] init tInit;
sx: [0..sxMax] init sxInit;
sy: [0..syMax] init syInit;

// X + W -> W + B
[rt] x>0 & w>0 -> (rt*x*w) : (x'=x-1) &
(b'=min(b+1, bMax));
```

```
// B + W -> W + Y
[rt] b>0 & w>0 -> (rt*b*w) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + R -> R + B
[rt] y>0 & r>0 -> (rt*y*r) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + R -> R + X
[rt] b>0 & r>0 -> (rt*b*r) : (b'=b-1) &
(x'=min(x+1, xMax));

// W + S -> S + U
[rt] w>0 & s>0 -> (rt*w*s) : (w'=w-1) &
(u'=min(u+1, uMax));
// U + S -> S + Z
[rt] u>0 & s>0 -> (rt*u*s) : (u'=u-1) &
(z'=min(z+1, zMax));
// Z + Y -> Y + U
[rt] z>0 & y>0 -> (rt*z*y) : (z'=z-1) &
(u'=min(u+1, uMax));
// U + Y -> Y + W
[rt] u>0 & y>0 -> (rt*u*y) : (u'=u-1) &
(w'=min(w+1, wMax));

// P + X -> X + Q
[rt] p>0 & x>0 -> (rt*p*x) : (p'=p-1) &
(q'=min(q+1, qMax));
// Q + X -> X + R
[rt] q>0 & x>0 -> (rt*q*x) : (q'=q-1) &
(r'=min(r+1, rMax));
// R + T -> T + Q
[rt] r>0 & t>0 -> (rt*r*t) : (r'=r-1) &
(q'=min(q+1, qMax));
// Q + T -> T + P
[rt] q>0 & t>0 -> (rt*q*t) : (q'=q-1) &
(p'=min(p+1, pMax));

// X + SY -> SY + B
[rt] x>0 & sy>0 -> (rt*x*sy) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + SY -> SY + Y
[rt] b>0 & sy>0 -> (rt*b*sy) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + SX -> SX + B
[rt] y>0 & sx>0 -> (rt*y*sx) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + SX -> SX + X
[rt] b>0 & sx>0 -> (rt*b*sx) : (b'=b-1) &
(x'=min(x+1, xMax));

endmodule

rewards "x" true : x; endrewards
rewards "y" true : y; endrewards
rewards "b" true : b; endrewards

rewards "w" true : w; endrewards
rewards "z" true : z; endrewards
rewards "u" true : u; endrewards

rewards "p" true : p; endrewards
rewards "r" true : r; endrewards
rewards "q" true : q; endrewards

rewards "s" true : s; endrewards
rewards "t" true : t; endrewards
rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards

rewards "time" true : 1; endrewards
```

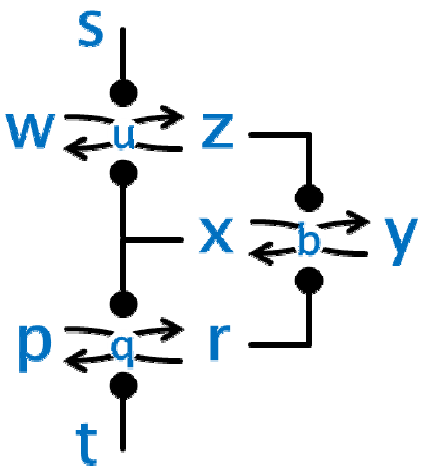
Query file (.csl)

```
const double T;
const int i;
P=? [F[T,T]x=i]
```

Initial Parameters

```
xInit = yInit = bInit = wInit = zInit =
uInit = pInit = rInit = qInit = sInit =
tInit = 5
rt = 1.0
i = 0 .. 15 by 1
T = 0 .. 2 by 0.1
(sxInit = syInit = 0)
```

CC (Fig 1d)



Chemical Reaction Model

```
x + z → z + b
b + z → z + y
y + r → r + b
b + r → r + x

w + s → s + u
u + s → s + z
z + x → x + u
u + x → x + w

p + x → x + q
q + x → x + r
r + t → t + q
q + t → t + p
```

Deterministic Model

```
(t is time: we use tt for species t)
(p is parameter: we use pp for species p)
dw/dt = -s*w + x*u
du/dt = s*w + z*x - s*u - x*u
dr/dt = x*q - tt*r
dq/dt = tt*r + x*pp - x*q - tt*q
dx/dt = (r+sx)*b - (z+sy)*x
db/dt = (r+sx)*y + (z+sy)*x - (z+sy)*b -
(r+sx)*b
z = wuz-w-u
pp = pqr-q-r
y = xby-b-x

wuz=1500, pqr=1500, xby=1500, s=500,
tt=500, sy=0, sx=0
```

XPPAUT / Oscill8 Source (.ode)

```
# t is time: we use tt for species t
# p is parameter: we use pp for species p
p wuz=1500, pqr=1500, xby=1500
p s=500, tt=500, sy=0, sx=0
z = wuz-w-u
pp = pqr-q-r
y = xby-b-x
dw/dt = -s*w + x*u
du/dt = s*w + z*x - s*u - x*u
dr/dt = x*q - tt*r
dq/dt = tt*r + x*pp - x*q - tt*q
dx/dt = (r+sx)*b - (z+sy)*x
db/dt = (r+sx)*y + (z+sy)*x - (z+sy)*b -
(r+sx)*b
aux z=z
aux pp=pp
aux y=y
```

SPiM Source (.spi)

```
directive sample 0.00710 1000
directive plot x()
val rt = 1.0
new xcat@rt:chan
new zcat@rt:chan
new scat@rt:chan
new tcat@rt:chan
new rcat@rt:chan
let x() =
   do !xcat; x()
   or ?zcat; b()
and y() =
   ?rcat; b()
and b() =
   do ?rcat; x()
   or ?zcat; y()
and z() =
   do !zcat; z()
   or ?xcat; u()
and r() =
   do !rcat; r()
   or ?tcat; q()
and w() =
   ?scat; u()
and u() =
   do ?xcat; w()
   or ?scat; z()
and s() =
   !scat; s()
and t() =
   !tcat; t()
and p() =
   ?xcat; q()
and q() =
   do ?xcat; r()
   or ?tcat; p()
run 5000 of x()
run 5000 of b()
run 5000 of y()
run 5000 of z()
run 5000 of u()
run 5000 of w()
run 5000 of p()
run 5000 of q()
run 5000 of r()
run 5000 of s()
run 5000 of t()
```

PRISM Source

Model file (.sm)

```
ctmc

const int xMax = xInit+yInit+bInit;
const int yMax = xInit+yInit+bInit;
const int bMax = xInit+yInit+bInit;

const int wMax = wInit+zInit+uInit;
const int zMax = wInit+zInit+uInit;
const int uMax = wInit+zInit+uInit;

const int pMax = pInit+rInit+qInit;
const int rMax = pInit+rInit+qInit;
const int qMax = pInit+rInit+qInit;

const int sMax = wInit+zInit+uInit;
const int tMax = pInit+rInit+qInit;
const int sxMax = xInit+yInit+bInit;
const int syMax = xInit+yInit+bInit;

const int xInit;
const int yInit;
const int bInit;

const int wInit;
const int zInit;
const int uInit;

const int pInit;
const int rInit;
const int qInit;

const int sInit;
const int tInit;
const int sxInit;
const int syInit;

const double rt;

module CC

x : [0..xMax] init xInit;
y : [0..yMax] init yInit;
b : [0..bMax] init bInit;

w : [0..wMax] init wInit;
z : [0..zMax] init zInit;
u : [0..uMax] init uInit;

p : [0..pMax] init pInit;
r : [0..rMax] init rInit;
q : [0..qMax] init qInit;

s : [0..sMax] init sInit;
t : [0..tMax] init tInit;
sx: [0..sxMax] init sxInit;
sy: [0..syMax] init syInit;

// X + Z -> Z + B
[rt] x>0 & z>0 -> (rt*x*z) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + Z -> Z + Y
[rt] b>0 & z>0 -> (rt*b*z) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + R -> R + B
[rt] y>0 & r>0 -> (rt*y*r) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + R -> R + X
[rt] b>0 & r>0 -> (rt*b*r) : (b'=b-1) &
(x'=min(x+1, xMax));

// W + S -> S + U
[rt] w>0 & s>0 -> (rt*w*s) : (w'=w-1) &
(u'=min(u+1, uMax));
// U + S -> S + Z
[rt] u>0 & s>0 -> (rt*u*s) : (u'=u-1) &
(z'=min(z+1, zMax));
// Z + X -> X + U
```

```
[rt] z>0 & x>0 -> (rt*z*x) : (z'=z-1) &
(u'=min(u+1, uMax));
// U + X -> X + W
[rt] u>0 & x>0 -> (rt*u*x) : (u'=u-1) &
(w'=min(w+1, wMax));

// P + X -> X + Q
[rt] p>0 & x>0 -> (rt*p*x) : (p'=p-1) &
(q'=min(q+1, qMax));
// Q + X -> X + R
[rt] q>0 & x>0 -> (rt*q*x) : (q'=q-1) &
(r'=min(r+1, rMax));
// R + T -> T + Q
[rt] r>0 & t>0 -> (rt*r*t) : (r'=r-1) &
(q'=min(q+1, qMax));
// Q + T -> T + P
[rt] q>0 & t>0 -> (rt*q*t) : (q'=q-1) &
(p'=min(p+1, pMax));

// X + SY -> SY + B
[rt] x>0 & sy>0 -> (rt*x*sy) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + SY -> SY + Y
[rt] b>0 & sy>0 -> (rt*b*sy) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + SX -> SX + B
[rt] y>0 & sx>0 -> (rt*y*sx) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + SX -> SX + X
[rt] b>0 & sx>0 -> (rt*b*sx) : (b'=b-1) &
(x'=min(x+1, xMax));

endmodule

rewards "x" true : x; endrewards
rewards "y" true : y; endrewards
rewards "b" true : b; endrewards

rewards "w" true : w; endrewards
rewards "z" true : z; endrewards
rewards "u" true : u; endrewards

rewards "p" true : p; endrewards
rewards "r" true : r; endrewards
rewards "q" true : q; endrewards

rewards "s" true : s; endrewards
rewards "t" true : t; endrewards
rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards

rewards "time" true : 1; endrewards
```

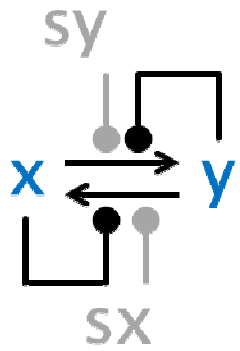
Query file (.csl)

```
const double T;
const int i;
P=? [F[T,T]x=i]
```

Initial Parameters

```
xInit = yInit = bInit = wInit = zInit =
uInit = pInit = rInit = qInit = sInit =
tInit = 5
rt = 1.0
i = 0 .. 15 by 1
T = 0 .. 2 by 0.1
(sxInit = syInit = 0)
```

DC with inputs (Fig 2a)



Chemical Reaction Model

x + y → y + y
y + x → x + x

x + sy → sy + y
y + sx → sx + x

Deterministic Model

Same as in Fig 1a.
xby=1500, sy=300, sx=0..1500

XPPAUT / Oscill8 Source (.ode)

```
p xby=1500
p sy=300, sx=0.1
Rest, same as in Fig 1a.
```

SPiM Source (.spi)

```
directive sample 300.0 2000
directive plot x()
val rt = 1.0
new xcat@rt:chan
new ycat@rt:chan
new sxcat@rt:chan new sxkill:chan
new sycat@rt:chan new sykill:chan
let x() =
   do !xcat; x()
   or ?ycat; y()
   or ?sycat; y()
and y() =
   do !ycat; y()
   or ?xcat; x()
   or ?sxcat; x()
and sy() = do !sycat; sy() or ?sykill; ()
and sx() = do !sxcat; sx() or ?sxkill; ()
run 70 of x()
run 80 of y()
run 30 of sy()
let clock(p:proc(int), t:float) =
(* Produce one p(m) every t sec with
precision dt,
    with m incremented from 0 *)
(val dt= 100.0 run step(p, 0, t, dt, dt))
and step(p:proc(int), m:int, t:float,
n:float, dt:float) =
if n<=0.0 then (p(m)|step(p,m+1,t,dt,dt))
else delay@dt/t; step(p,m,t,n-1.0,dt)
let schedule(n:int) =
  if n < 150 then sx()
```

```
  else if n < 300 then !sxkill;()
  else ()
run clock(schedule,1.0)
```

PRISM Source

Model file (.sm)

Same as in Fig 1a.

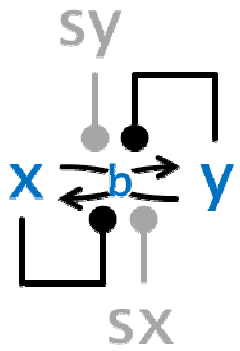
Query file (.csl)

```
const int i;
S=? [ x=i ]
```

Initial Parameters

```
xInit = 7
yInit = 8
sxInit = 0 .. 15 by 1
syInit = 3
rt = 1.0
i = 0 .. 15 by 1
```

AM with inputs (Fig 2b)



Chemical Reaction Model

x + y → y + b
y + x → x + b
b + x → x + x
b + y → y + y

x + sy → sy + b
b + sy → sy + y
y + sx → sx + b
b + sx → sx + x

Deterministic Model

Same as in Fig 1b.
xby=1500, sy=300, sx=0..1500

XPPAUT / Oscill8 Source (.ode)

```
p xby=1500
p sy=300, sx=0.1
Rest, same as in Fig 1b.
```

SPiM Source (.spi)

```
directive sample 300.0 2000
directive plot x()
```

```
val rt = 1.0
new xcat@rt:chan
new ycat@rt:chan
new sxcat@rt:chan new sxkill:chan
new sycat@rt:chan new sykill:chan
let x() =
   do !xcat; x()
   or ?ycat; b()
   or ?sycat; b()
and y() =
   do !ycat; y()
   or ?xcat; b()
   or ?sxcat; b()
and b() =
   do ?xcat; x()
   or ?sxcat; x()
   or ?ycat; y()
   or ?sycat; y()
and sy() = do !sycat; sy() or ?sykill; ()
and sx() = do !sxcat; sx() or ?sxkill; ()
run 50 of x()
run 50 of b()
run 50 of y()
run 30 of sy()
let clock(p:proc(int), t:float) =
(* Produce one p(m) every t sec with
precision dt,
    with m incremented from 0 *)
(val dt= 100.0 run step(p, 0, t, dt, dt))
and step(p:proc(int), m:int, t:float,
n:float, dt:float) =
if n<=0.0 then (p(m)|step(p,m+1,t,dt,dt))
else delay@dt/t; step(p,m,t,n-1.0,dt)
let schedule(n:int) =
  if n < 150 then sx()
  else if n < 300 then !sxkill;()
  else ()
run clock(schedule,1.0)
```

PRISM Source

Model file (.sm)

Same as in Fig 1b.

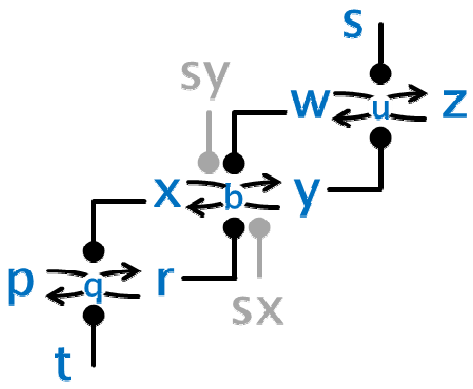
Query file (.csl)

```
const int i;
S=? [ x=i ]
```

Initial Parameters

```
xInit = yInit = bInit = 5
sxInit = 0 .. 15 by 1
syInit = 3
rt = 1.0
i = 0 .. 15 by 1
```

SC with inputs (Fig 2c)



(w and z are swapped w.r.t. figure 2c)

Chemical Reaction Model

```
x + w → w + b
b + w → w + y
y + r → r + b
b + r → r + x

w + s → s + u
u + s → s + z
z + y → y + u
u + y → y + w

p + x → x + q
q + x → x + r
r + t → t + q
q + t → t + p

x + sy → sy + b
b + sy → sy + y
y + sx → sx + b
b + sx → sx + x
```

Deterministic Model

```
Same as in Fig 1c.
wuz=1500, pqr=1500, xby=1500, s=500,
tt=500, sy=300, sx=0..1500
```

XPPAUT / Oscill8 Source (.ode)

```
p wuz=1500, pqr=1500, xby=1500
p s=500, tt=500, sy=300, sx=0.1
Rest, same as in Fig 1c.
```

SPiM Source (.spi)

```
directive sample 300.0 2000
directive plot x()
val rt = 1.0
new xcat@rt:chan
new ycat@rt:chan
new wcat@rt:chan
new rcat@rt:chan
new scat@rt:chan
new tcat@rt:chan
new sxcat@rt:chan new sxkill:chan
new sycat@rt:chan new sykill:chan
let x() =
   do !xcat; x()
   or ?wcat; b()
   or ?sycat; b()
and y() =
   do !ycat; y()
   or ?rcat; b()
   or ?sxcat; b()
and b() =
   do ?rcat; x()
   or ?sxcat; x()
   or ?wcat; y()
   or ?sycat; y()
and z() =
   ?ycat; u()
and r() =
   do !rcat; r()
   or ?tcat; q()
and s() =
   !scat(); s()
and w() =
  do !wcat; w()
  or ?scat; u()
and u() =
   do ?scat; z()
   or ?ycat; w()
and t() =
   !tcat; t()
and p() =
   ?xcat; q()
and q() =
   do ?xcat; r()
   or ?tcat; p()
and sy() = do !sycat; sy() or ?sykill; ()
and sx() = do !sxcat; sx() or ?sxkill; ()
run 50 of x()
run 50 of b()
run 50 of y()
run 50 of z()
run 50 of u()
run 50 of w()
run 50 of p()
run 50 of q()
run 50 of r()
run 50 of s()
run 50 of t()
run 30 of sy()
let clock(ps:proc(int), tm:float) =
(* Produce one ps(m) every tm sec with
precision dt,
    with m incremented from 0 *)
(val dt= 100.0 run step(ps, 0, tm, dt, dt))
and step(ps:proc(int), m:int, tm:float,
n:float, dt:float) =
if n<=0.0 then
(ps(m)|step(ps,m+1,tm,dt,dt))
else delay@dt/tm; step(ps,m,tm,n-1.0,dt)
let schedule(n:int) =
  if n < 150 then sx()
  else if n < 300 then !sxkill;()
  else ()
run clock(schedule,1.0)
```

PRISM Source

Model file (.sm)

```
Same as in Fig 1c.
```

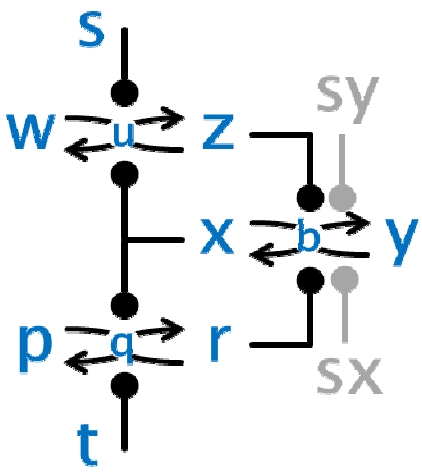
Query file (.csl)

```
const int i;
S=?[x=i]
```

Initial Parameters

```
xInit = yInit = bInit = wInit = zInit =
uInit = pInit = rInit = qInit = sInit =
tInit = 5
sxInit = 0 .. 15 by 1
syInit = 3
rt = 1.0
i = 0 .. 15 by 1
```

CC with inputs (Fig 2d)



Chemical Reaction Model

```
x + z → z + b
b + z → z + y
y + r → r + b
b + r → r + x

w + s → s + u
u + s → s + z
z + x → x + u
u + x → x + w

p + x → x + q
q + x → x + r
r + t → t + q
q + t → t + p

x + sy → sy + b
b + sy → sy + y
y + sx → sx + b
b + sx → sx + x
```

Deterministic Model

```
Same as in Fig 1c.
wuz=1500, pqr=1500, xby=1500, s=500,
tt=500, sy=300, sx=0..1500
```

XPPAUT / Oscill8 Source (.ode)

```
p wuz=1500, pqr=1500, xby=1500
p s=500, tt=500, sy=300, sx=0.1
Rest, same as in Fig 1d.
```

SPiM Source (.spi)

```
directive sample 300.0 2000
directive plot x()
val rt = 1.0
new xcat@rt:chan
new zcat@rt:chan
new scat@rt:chan
new tcat@rt:chan
new rcat@rt:chan
new sxcat@rt:chan new sxkill:chan
new sycat@rt:chan new sykill:chan
let x() =
   do !xcat; x()
   or ?zcat; b()
   or ?sycat; b()
and y() =
   do ?rcat; b()
   or ?sxcat; b()
and b() =
   do ?rcat; x()
   or ?sxcat; x()
   or ?zcat; y()
   or ?sycat; y()
and z() =
   do !zcat; z()
   or ?xcat; u()
and r() =
   do !rcat; r()
   or ?tcat; q()
and w() =
   ?scat; u()
and u() =
   do ?xcat; w()
   or ?scat; z()
and s() =
   !scat; s()
and t() =
   !tcat; t()
and p() =
   ?xcat; q()
and q() =
   do ?xcat; r()
   or ?tcat; p()
and sy() = do !sycat; sy() or ?sykill; ()
and sx() = do !sxcat; sx() or ?sxkill; ()
run 50 of y()
run 50 of x()
run 50 of b()
run 50 of z()
run 50 of w()
run 50 of u()
run 50 of p()
run 50 of r()
run 50 of q()
run 50 of s()
run 50 of t()
run 30 of sy()
let clock(ps:proc(int), tm:float) =
(* Produce one ps(m) every tm sec with
precision dt,
    with m incremented from 0 *)
(val dt= 100.0 run step(ps, 0, tm, dt, dt))
and step(ps:proc(int), m:int, tm:float,
n:float, dt:float) =
if n<=0.0 then
(ps(m)|step(ps,m+1,tm,dt,dt))
else delay@dt/tm; step(ps,m,tm,n-1.0,dt)
let schedule(n:int) =
  if n < 150 then sx()
  else if n < 300 then !sxkill;()
  else ()
run clock(schedule,1.0)
```

PRISM Source

Model file (.sm)

```
Same as in Fig 1d.
```

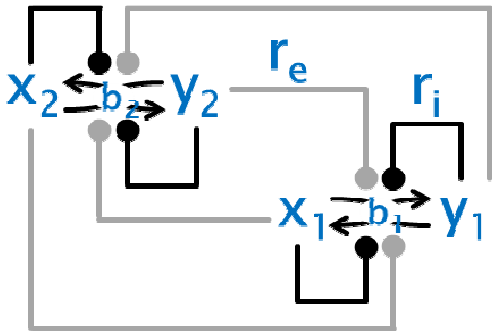
Query file (.csl)

```
const int i;
S=?[x=i]
```

Initial Parameters

```
xInit = yInit = bInit = wInit = zInit =
uInit = pInit = rInit = qInit = sInit =
tInit = 5
sxInit = 0 .. 15 by 1
syInit = 3
rt = 1.0
i = 0 .. 15 by 1
```

2AM Full (Fig 3a)



Chemical Reaction Model

```
x1 + y1 →rᵢ y1 + b1
y1 + x1 →rᵢ x1 + b1
b1 + x1 →rᵢ x1 + x1
b1 + y1 →rᵢ y1 + y1

x2 + y2 →rᵢ y2 + b2
y2 + x2 →rᵢ x2 + b2
b2 + x2 →rᵢ x2 + x2
b2 + y2 →rᵢ y2 + y2

x1 + y2 →rₑ y2 + b1
b1 + y2 →rₑ y2 + y1
y1 + x2 →rₑ x2 + b1
b1 + x2 →rₑ x2 + x1

x2 + x1 →rₑ x1 + b2
b2 + x1 →rₑ x1 + y2
y2 + y1 →rₑ y1 + b2
b2 + y1 →rₑ y1 + x2
```

Deterministic Model

```
(k=rᵢ/rₑ)
dx1/dt = (x1+k*x2+sx)*b1 - (y1+k*y2+sy)*x1
db1/dt = (x1+k*x2+sx)*y1 + (y1+k*y2+sy)*x1
- (y1+k*y2+sy)*b1 - (x1+k*x2+sx)*b1
dx2/dt = (x2+k*y1)*b2 - (y2+k*x1)*x2
db2/dt = (x2+k*y1)*y2 + (y2+k*x1)*x2 -
(y2+k*x1)*b2 - (x2+k*y1)*b2
y1 = xby1-b1-x1
y2 = xby2-b2-x2
```

XPPAUT / Oscill8 Source (.ode)

```
p xby1=1500, xby2=1500
p sy=0, sx=0, k=1
y1 = xby1-b1-x1
dx1/dt = (x1+k*x2+sx)*b1 - (y1+k*y2+sy)*x1
db1/dt = (x1+k*x2+sx)*y1 + (y1+k*y2+sy)*x1
- (y1+k*y2+sy)*b1 - (x1+k*x2+sx)*b1
aux y1=y1
y2 = xby2-b2-x2
dx2/dt = (x2+k*y1)*b2 - (y2+k*x1)*x2
db2/dt = (x2+k*y1)*y2 + (y2+k*x1)*x2 -
(y2+k*x1)*b2 - (x2+k*y1)*b2
aux y2=y2
```

SPiM Source (.spi)

```
(r=rᵢ, s=rₑ)
directive sample 0.005 10000
directive plot x1(); y1(); b1(); x2();
y2(); b2()

val r = 1.0
new x1cat@r:chan
new y1cat@r:chan
new x2cat@r:chan
new y2cat@r:chan

val s = 0.5
new x1cat2@s:chan
new y1cat2@s:chan
new x2cat1@s:chan
new y2cat1@s:chan

let x1() =
   do !x1cat; x1()
   or !x1cat2; x1()
   or ?y1cat; b1()
   or ?y2cat1; b1()
and y1() =
   do !y1cat; y1()
   or !y1cat2; y1()
   or ?x1cat; b1()
   or ?x2cat1; b1()
and b1() =
   do ?x1cat; x1()
   or ?x2cat1; x1()
   or ?y1cat; y1()
   or ?y2cat1; y1()

let x2() =
   do !x2cat; x2()
   or !x2cat1; x2()
   or ?y2cat; b2()
   or ?x1cat2; b2()
and y2() =
   do !y2cat; y2()
   or !y2cat1; y2()
   or ?x2cat; b2()
   or ?y1cat2; b2()
and b2() =
   do ?x2cat; x2()
   or ?y1cat2; x2()
   or ?y2cat; y2()
   or ?x1cat2; y2()

run 10000 of x1()
run 10000 of y1()
run 10000 of b1()

run 10000 of x2()
run 10000 of y2()
run 10000 of b2()
```

PRISM Source

Model file (.sm)

```
(rt1=rᵢ, rt2=rₑ)
ctmc

const int x1Max = x1Init+y1Init+b1Init;
const int y1Max = x1Init+y1Init+b1Init;
const int b1Max = x1Init+y1Init+b1Init;
const int x2Max = x2Init+y2Init+b2Init;
const int y2Max = x2Init+y2Init+b2Init;
const int b2Max = x2Init+y2Init+b2Init;

const int x1Init;
const int y1Init;
const int b1Init;
const int x2Init;
const int y2Init;
const int b2Init;

const double rt1;
const double rt2;

module TwoAM

x1 : [0..x1Max] init x1Init;
y1 : [0..y1Max] init y1Init;
b1 : [0..b1Max] init b1Init;
x2 : [0..x2Max] init x2Init;
y2 : [0..y2Max] init y2Init;
b2 : [0..b2Max] init b2Init;

// X1 + Y1 -> Y1 + B1
[rt1] x1>0 & y1>0 -> (rt1*x1*y1) : (x1'=x1-
1) & (b1'=min(b1+1, b1Max));
// Y1 + X1 -> X1 + B1
[rt1] y1>0 & x1>0 -> (rt1*y1*x1) : (y1'=y1-
1) & (b1'=min(b1+1, b1Max));
// B1 + X1 -> X1 + X1
[rt1] b1>0 & x1>0 -> (rt1*b1*x1) : (b1'=b1-
1) & (x1'=min(x1+1, x1Max));
// B1 + Y1 -> Y1 + Y1
[rt1] b1>0 & y1>0 -> (rt1*b1*y1) : (b1'=b1-
1) & (y1'=min(y1+1, y1Max));

// X2 + Y2 -> Y2 + B2
[rt1] x2>0 & y2>0 -> (rt1*x2*y2) : (x2'=x2-
1) & (b2'=min(b2+1, b2Max));
// Y2 + X2 -> X2 + B2
[rt1] y2>0 & x2>0 -> (rt1*y2*x2) : (y2'=y2-
1) & (b2'=min(b2+1, b2Max));
// B2 + X2 -> X2 + X2
[rt1] b2>0 & x2>0 -> (rt1*b2*x2) : (b2'=b2-
1) & (x2'=min(x2+1, x2Max));
// B2 + Y2 -> Y2 + Y2
[rt1] b2>0 & y2>0 -> (rt1*b2*y2) : (b2'=b2-
1) & (y2'=min(y2+1, y2Max));

// X1 + Y2 -> Y2 + B1
[rt2] x1>0 & y2>0 -> (rt2*x1*y2) : (x1'=x1-
1) & (b1'=min(b1+1, b1Max));
// B1 + Y2 -> Y2 + Y1
[rt2] b1>0 & y2>0 -> (rt2*b1*y2) : (b1'=b1-
1) & (y1'=min(y1+1, y1Max));
// Y1 + X2 -> X2 + B1
[rt2] y1>0 & x2>0 -> (rt2*y1*x2) : (y1'=y1-
1) & (b1'=min(b1+1, b1Max));
// B1 + X2 -> X2 + X1
[rt2] b1>0 & x2>0 -> (rt2*b1*x2) : (b1'=b1-
1) & (x1'=min(x1+1, x1Max));

// X2 + X1 -> X1 + B2
[rt2] x2>0 & x1>0 -> (rt2*x2*x1) : (x2'=x2-
1) & (b2'=min(b2+1, b2Max));
// B2 + X1 -> X1 + Y2
[rt2] b2>0 & x1>0 -> (rt2*b2*x1) : (b2'=b2-
1) & (y2'=min(y2+1, y2Max));
// Y2 + Y1 -> Y1 + B2
[rt2] y2>0 & y1>0 -> (rt2*y2*y1) : (y2'=y2-
1) & (b2'=min(b2+1, b2Max));
// B2 + Y1 -> Y1 + X2
[rt2] b2>0 & y1>0 -> (rt2*b2*y1) : (b2'=b2-
1) & (x2'=min(x2+1, x2Max));

endmodule

rewards "x1" true : x1; endrewards
rewards "y1" true : y1; endrewards
rewards "b1" true : b1; endrewards
rewards "x2" true : x2; endrewards
rewards "y2" true : y2; endrewards
rewards "b2" true : b2; endrewards

rewards "time" true : 1; endrewards
```

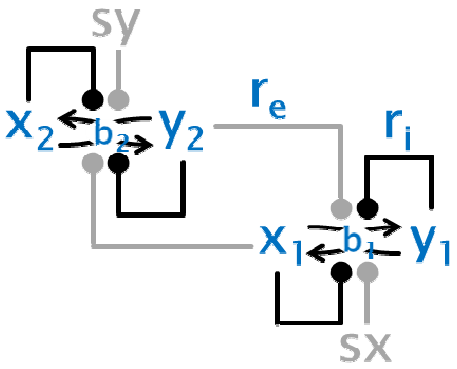
Query file (.csl)

```
const int i;
const int j;
S=? [ x1=i&x2=j ]
```

Initial Parameters

```
x1Init = y1Init = b1Init = x2Init = y2Init
= b2Init = 10
rt1 = 1.0
rt2 = 0.5
i = 0 .. 30 by 1
j = 0 .. 30 by 1
```

2AM Inner (Fig 3b)



Chemical Reaction Model

$x1 + y1 \xrightarrow{r_i} y1 + b1$
$y1 + x1 \xrightarrow{r_i} x1 + b1$
$b1 + x1 \xrightarrow{r_i} x1 + x1$
$b1 + y1 \xrightarrow{r_i} y1 + y1$

$x2 + y2 \xrightarrow{r_i} y2 + b2$
$y2 + x2 \xrightarrow{r_i} x2 + b2$
$b2 + x2 \xrightarrow{r_i} x2 + x2$
$b2 + y2 \xrightarrow{r_i} y2 + y2$

$x1 + y2 \xrightarrow{r_e} y2 + b1$
$b1 + y2 \xrightarrow{r_e} y2 + y1$

```
y1 + sx →re sx + b1
b1 + sx →re sx + x1

x2 + x1 →re x1 + b2
b2 + x1 →re x1 + y2
y2 + sy →re sy + b2
b2 + sy →re sy + x2
```

Deterministic Model

```
(ri = 1.0)
dx1/dt = (x1+re*sx)*b1 - (y1+re*y2)*x1
db1/dt = (x1+re*sx)*y1 + (y1+re*y2)*x1 -
(y1+re*y2)*b1 - (x1+re*sx)*b1
dx2/dt = (x2+re*sy)*b2 - (y2+re*x1)*x2
db2/dt = (x2+re*sy)*y2 + (y2+re*x1)*x2 -
(y2+re*x1)*b2 - (x2+re*sy)*b2
y1 = xby1-b1-x1
y2 = xby2-b2-x2
```

XPPAUT / Oscill8 Source (.ode)

```
p xby1=30, xby2=30
p sy=10, sx=10, re=0.5
y1 = xby1-b1-x1
dx1/dt = (x1+re*sx)*b1 - (y1+re*y2)*x1
db1/dt = (x1+re*sx)*y1 + (y1+re*y2)*x1 -
(y1+re*y2)*b1 - (x1+re*sx)*b1
aux y1=y1
y2 = xby2-b2-x2
dx2/dt = (x2+re*sy)*b2 - (y2+re*x1)*x2
db2/dt = (x2+re*sy)*y2 + (y2+re*x1)*x2 -
(y2+re*x1)*b2 - (x2+re*sy)*b2
aux y2=y2
```

SPiM Source (.spi)

```
directive sample 0.01 10000
directive plot x1(); y1(); b1(); x2();
y2(); b2()

val r = 1.0
new x1cat@r:chan
new y1cat@r:chan
new x2cat@r:chan
new y2cat@r:chan

val s = 0.5
new x1cat2@s:chan
new y2cat1@s:chan
new sxcat@s:chan
new sycat@s:chan

let sx() =
   !sxcat; sx()
and sy() =
   !sycat; sy()

let x1() =
   do !x1cat; x1()
   or !x1cat2; x1()
   or ?y1cat; b1()
   or ?y2cat1; b1()
and y1() =
   do !y1cat; y1()
   or ?x1cat; b1()
   or ?sxcat; b1()
and b1() =
   do ?x1cat; x1()
   or ?sxcat; x1()
   or ?y1cat; y1()
   or ?y2cat1; y1()

let x2() =
   do !x2cat; x2()
   or ?y2cat; b2()
   or ?x1cat2; b2()
and y2() =
   do !y2cat; y2()
   or !y2cat1; y2()
   or ?x2cat; b2()
   or ?sycat; b2()
and b2() =
   do ?x2cat; x2()
   or ?sycat; x2()
   or ?y2cat; y2()
   or ?x1cat2; y2()

run 10000 of x1()
run 10000 of y1()
run 10000 of b1()

run 10000 of x2()
run 10000 of y2()
run 10000 of b2()

run 10000 of sx()
run 10000 of sy()
```

PRISM Source

Model file (.sm)

```
ctmc

const int x1Max = x1Init+y1Init+b1Init;
const int y1Max = x1Init+y1Init+b1Init;
const int b1Max = x1Init+y1Init+b1Init;
const int x2Max = x2Init+y2Init+b2Init;
const int y2Max = x2Init+y2Init+b2Init;
const int b2Max = x2Init+y2Init+b2Init;

const int sxMax = x1Init+y1Init+b1Init;
const int syMax = x2Init+y2Init+b2Init;

const int x1Init;
const int y1Init;
const int b1Init;
const int x2Init;
const int y2Init;
const int b2Init;

const int sxInit;
const int syInit;

const double rt1;
const double rt2;

module TwoAMInner

x1 : [0..x1Max] init x1Init;
y1 : [0..y1Max] init y1Init;
b1 : [0..b1Max] init b1Init;
x2 : [0..x2Max] init x2Init;
y2 : [0..y2Max] init y2Init;
b2 : [0..b2Max] init b2Init;

sx : [0..sxMax] init sxInit;
sy : [0..syMax] init syInit;

// X1 + Y1 -> Y1 + B1
[rt1] x1>0 & y1>0 -> (rt1*x1*y1) : (x1'=x1-
1) & (b1'=min(b1+1, b1Max));
// Y1 + X1 -> X1 + B1
[rt1] y1>0 & x1>0 -> (rt1*y1*x1) : (y1'=y1-
1) & (b1'=min(b1+1, b1Max));
// B1 + X1 -> X1 + X1
[rt1] b1>0 & x1>0 -> (rt1*b1*x1) : (b1'=b1-
1) & (x1'=min(x1+1, x1Max));
// B1 + Y1 -> Y1 + Y1
```

```
[rt1] b1>0 & y1>0 -> (rt1*b1*y1) : (b1'=b1-
1) & (y1'=min(y1+1, y1Max));

// X2 + Y2 -> Y2 + B2
[rt1] x2>0 & y2>0 -> (rt1*x2*y2) : (x2'=x2-
1) & (b2'=min(b2+1, b2Max));
// Y2 + X2 -> X2 + B2
[rt1] y2>0 & x2>0 -> (rt1*y2*x2) : (y2'=y2-
1) & (b2'=min(b2+1, b2Max));
// B2 + X2 -> X2 + X2
[rt1] b2>0 & x2>0 -> (rt1*b2*x2) : (b2'=b2-
1) & (x2'=min(x2+1, x2Max));
// B2 + Y2 -> Y2 + Y2
[rt1] b2>0 & y2>0 -> (rt1*b2*y2) : (b2'=b2-
1) & (y2'=min(y2+1, y2Max));

// X1 + Y2 -> Y2 + B1
[rt2] x1>0 & y2>0 -> (rt2*x1*y2) : (x1'=x1-
1) & (b1'=min(b1+1, b1Max));
// B1 + Y2 -> Y2 + Y1
[rt2] b1>0 & y2>0 -> (rt2*b1*y2) : (b1'=b1-
1) & (y1'=min(y1+1, y1Max));
// Y1 + SX -> SX + B1
[rt2] y1>0 & sx>0 -> (rt2*y1*sx) : (y1'=y1-
1) & (b1'=min(b1+1, b1Max));
// B1 + SX -> SX + X1
[rt2] b1>0 & sx>0 -> (rt2*b1*sx) : (b1'=b1-
1) & (x1'=min(x1+1, x1Max));

// X2 + X1 -> X1 + B2
[rt2] x2>0 & x1>0 -> (rt2*x2*x1) : (x2'=x2-
1) & (b2'=min(b2+1, b2Max));
// B2 + X1 -> X1 + Y2
[rt2] b2>0 & x1>0 -> (rt2*b2*x1) : (b2'=b2-
1) & (y2'=min(y2+1, y2Max));
// Y2 + SY -> SY + B2
[rt2] y2>0 & sy>0 -> (rt2*y2*sy) : (y2'=y2-
1) & (b2'=min(b2+1, b2Max));
// B2 + SY -> SY + X2
[rt2] b2>0 & sy>0 -> (rt2*b2*sy) : (b2'=b2-
1) & (x2'=min(x2+1, x2Max));

endmodule

rewards "x1" true : x1; endrewards
rewards "y1" true : y1; endrewards
rewards "b1" true : b1; endrewards
rewards "x2" true : x2; endrewards
rewards "y2" true : y2; endrewards
rewards "b2" true : b2; endrewards
rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards

rewards "time" true : 1; endrewards
```

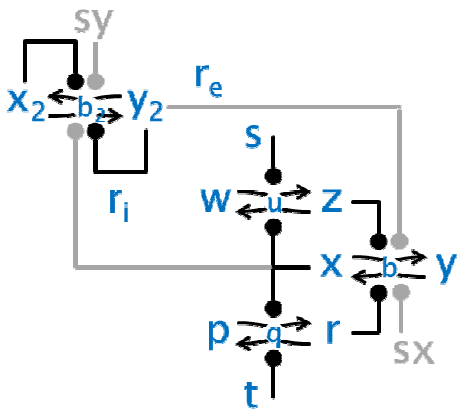
Query file (.csl)

```
const int i;
const int j;
S=? [ x1=i&x2=j ]
```

Initial Parameters

```
x1Init = y1Init = b1Init = x2Init = y2Init
= b2Init = sxInit = syInit = 10
rt1 = 1.0
rt2 = 0.5
i = 0 .. 30 by 1
j = 0 .. 30 by 1
```

CC-AM Inner (Fig 3c)



Chemical Reaction Model

$x + z \xrightarrow{r_i} z + b$
$b + z \xrightarrow{r_i} z + y$
$y + r \xrightarrow{r_i} r + b$
$b + r \xrightarrow{r_i} r + x$

$w + s \xrightarrow{r_i} s + u$
$u + s \xrightarrow{r_i} s + z$
$z + x \xrightarrow{r_i} x + u$
$u + x \xrightarrow{r_i} x + w$

$p + x \xrightarrow{r_i} x + q$
$q + x \xrightarrow{r_i} x + r$
$r + t \xrightarrow{r_i} t + q$
$q + t \xrightarrow{r_i} t + p$

$x2 + y2 \xrightarrow{r_i} y2 + b2$
$y2 + x2 \xrightarrow{r_i} x2 + b2$
$b2 + x2 \xrightarrow{r_i} x2 + x2$
b2 + y2 ->ri y2 + y2

$x + y2 \xrightarrow{r_e} y2 + b$
$b + y2 \xrightarrow{r_e} y2 + y$
$y + sx \xrightarrow{r_e} sx + b$
$b + sx \xrightarrow{r_e} sx + x$

$x2 + x \xrightarrow{r_e} x + b2$
$b2 + x \xrightarrow{r_e} x + y2$
$y2 + sy \xrightarrow{r_e} sy + b2$
$b2 + sy \xrightarrow{r_e} sy + x2$

Deterministic Model

(r_i = 1.0)
(t is time: we use tt for species t)
(p is parameter: we use pp for species p)

```
dw/dt = -s*w + x*u
du/dt = s*w + z*x - s*u - x*u
dr/dt = x*q - tt*r
dq/dt = tt*r + x*pp - x*q - tt*q
dx/dt = (r+re*sx)*b - (z+re*y2)*x
db/dt = (r+re*sx)*y + (z+re*y2)*x -
(z+re*y2)*b - (r+re*sx)*b
dx2/dt = (x2+re*sy)*b2 - (y2+re*x)*x2
```

```
db2/dt = (x2+re*sy)*y2 + (y2+re*x)*x2 -
(y2+re*x)*b2 - (x2+re*sy)*b2
z = wuz-w-u
pp = pqr-q-r
y = xby-b-x
y2 = xby2-b2-x2
```

XPPAUT / Oscill8 Source (.ode)

```
# t is time: we use tt for species t
# p is parameter: we use pp for species p
p wuz=30, pqr=30, xby=30
p s=10, tt=10, sy=10, sx=10
p re=0.675, xby2=30
z = wuz-w-u
pp = pqr-q-r
y = xby-b-x
dw/dt = -s*w + x*u
du/dt = s*w + z*x - s*u - x*u
dr/dt = x*q - tt*r
dq/dt = tt*r + x*pp - x*q - tt*q
dx/dt = (r+re*sx)*b - (z+re*y2)*x
db/dt = (r+re*sx)*y + (z+re*y2)*x -
(z+re*y2)*b - (r+re*sx)*b
aux z=z
aux pp=pp
aux y=y
y2 = xby2-b2-x2
dx2/dt = (x2+re*sy)*b2 - (y2+re*x)*x2
db2/dt = (x2+re*sy)*y2 + (y2+re*x)*x2 -
(y2+re*x)*b2 - (x2+re*sy)*b2
aux y2=y2
```

SPiM Source (.spi)

```
directive sample 0.02 10000
directive plot x(); y(); b(); x2(); y2();
b2(); z(); r()
val ri = 1.0
val re = 0.75
new xcat@ri:chan
new zcat@ri:chan
new rcat@ri:chan
new scat@ri:chan
new tcat@ri:chan
new x2cat@ri:chan
new y2cat@ri:chan
new xcat1@re:chan
new y2cat1@re:chan
new sxcat@re:chan
new sycat@re:chan
let sx() =
   !sxcat; sx()
and sy() =
   !sycat; sy()
let x() =
   do !xcat; x()
   or !xcat1; x()
   or ?zcat; b()
   or ?y2cat1; b()
and y() =
   do ?rcat; b()
   or ?sxcat; b()
and b() =
   do ?rcat; x()
   or ?sxcat; x()
   or ?zcat; y()
   or ?y2cat1; y()
and z() =
   do !zcat; z()
   or ?xcat; u()
and r() =
   do !rcat; r()
   or ?tcat; q()
and s() =
   !scat(); s()
and w() =
   ?scat; u()
and u() =
   do ?scat; z()
   or ?xcat; w()
and t() =
   !tcat; t()
and p() =
   ?xcat; q()
and q() =
   do ?xcat; r()
   or ?tcat; p()

let x2() =
   do !x2cat; x2()
   or ?y2cat; b2()
   or ?xcat1; b2()
and y2() =
   do !y2cat; y2()
   or !y2cat1; y2()
   or ?x2cat; b2()
   or ?sycat; b2()
and b2() =
   do ?x2cat; x2()
   or ?sycat; x2()
   or ?y2cat; y2()
   or ?xcat1; y2()

run 10000 of s()
run 10000 of t()
run 10000 of p()
run 10000 of q()
run 10000 of r()
run 10000 of z()
run 10000 of u()
run 10000 of w()
run 10000 of x()
run 10000 of b()
run 10000 of y()
run 10000 of x2()
run 10000 of y2()
run 10000 of b2()
run 10000 of sx()
run 10000 of sy()
```

PRISM Source

Model file (.sm)

```
ctmc

const int xMax = xInit+yInit+bInit;
const int yMax = xInit+yInit+bInit;
const int bMax = xInit+yInit+bInit;

const int wMax = wInit+zInit+uInit;
const int zMax = wInit+zInit+uInit;
const int uMax = wInit+zInit+uInit;

const int pMax = pInit+rInit+qInit;
const int rMax = pInit+rInit+qInit;
const int qMax = pInit+rInit+qInit;

const int sMax = wInit+zInit+uInit;
const int tMax = pInit+rInit+qInit;

const int x2Max = x2Init+y2Init+b2Init;
const int y2Max = x2Init+y2Init+b2Init;
const int b2Max = x2Init+y2Init+b2Init;

const int sxMax = xInit+yInit+bInit;
const int syMax = xInit+yInit+bInit;

const int xInit;
const int yInit;
```

```
const int bInit;

const int wInit;
const int zInit;
const int uInit;

const int pInit;
const int rInit;
const int qInit;

const int sInit;
const int tInit;

const int x2Init;
const int y2Init;
const int b2Init;

const int sxInit;
const int syInit;

const double rt;
const double rt2;

module CCAMInner

x : [0..xMax] init xInit;
y : [0..yMax] init yInit;
b : [0..bMax] init bInit;

w : [0..wMax] init wInit;
z : [0..zMax] init zInit;
u : [0..uMax] init uInit;

p : [0..pMax] init pInit;
r : [0..rMax] init rInit;
q : [0..qMax] init qInit;

s : [0..sMax] init sInit;
t : [0..tMax] init tInit;

x2 : [0..x2Max] init x2Init;
y2 : [0..y2Max] init y2Init;
b2 : [0..b2Max] init b2Init;

sx : [0..sxMax] init sxInit;
sy : [0..syMax] init syInit;

// X + Z -> Z + B
[rt] x>0 & z>0 -> (rt*x*z) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + Z -> Z + Y
[rt] b>0 & z>0 -> (rt*b*z) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + R -> R + B
[rt] y>0 & r>0 -> (rt*y*r) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + R -> R + X
[rt] b>0 & r>0 -> (rt*b*r) : (b'=b-1) &
(x'=min(x+1, xMax));


// W + S -> S + U
[rt] w>0 & s>0 -> (rt*w*s) : (w'=w-1) &
(u'=min(u+1, uMax));
// U + S -> S + Z
[rt] u>0 & s>0 -> (rt*u*s) : (u'=u-1) &
(z'=min(z+1, zMax));
// Z + X -> X + U
[rt] z>0 & x>0 -> (rt*z*x) : (z'=z-1) &
(u'=min(u+1, uMax));
// U + X -> X + W
[rt] u>0 & x>0 -> (rt*u*x) : (u'=u-1) &
(w'=min(w+1, wMax));

// P + X -> X + Q
[rt] p>0 & x>0 -> (rt*p*x) : (p'=p-1) &
(q'=min(q+1, qMax));
// Q + X -> X + R
[rt] q>0 & x>0 -> (rt*q*x) : (q'=q-1) &
(r'=min(r+1, rMax));
// R + T -> T + Q
[rt] r>0 & t>0 -> (rt*r*t) : (r'=r-1) &
(q'=min(q+1, qMax));
// Q + T -> T + P
[rt] q>0 & t>0 -> (rt*q*t) : (q'=q-1) &
(p'=min(p+1, pMax));

// X2 + Y2 -> Y2 + B2
[rt] x2>0 & y2>0 -> (rt*x2*y2) : (x2'=x2-1)
& (b2'=min(b2+1, b2Max));
// Y2 + X2 -> X2 + B2
[rt] y2>0 & x2>0 -> (rt*y2*x2) : (y2'=y2-1)
& (b2'=min(b2+1, b2Max));
// B2 + X2 -> X2 + X2
[rt] b2>0 & x2>0 -> (rt*b2*x2) : (b2'=b2-1)
& (x2'=min(x2+1, x2Max));
// B2 + Y2 -> Y2 + Y2
[rt] b2>0 & y2>0 -> (rt*b2*y2) : (b2'=b2-1)
& (y2'=min(y2+1, y2Max));

// X + Y2 -> Y2 + B
[rt2] x>0 & y2>0 -> (rt2*x*y2) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + Y2 -> Y2 + Y
[rt2] b>0 & y2>0 -> (rt2*b*y2) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + SX -> SX + B
[rt2] y>0 & sx>0 -> (rt2*y*sx) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + SX -> SX + X
[rt2] b>0 & sx>0 -> (rt2*b*sx) : (b'=b-1) &
(x'=min(x+1, xMax));

// X2 + X -> X + B2
[rt2] x2>0 & x>0 -> (rt2*x2*x) : (x2'=x2-1)
& (b2'=min(b2+1, b2Max));
// B2 + X -> X + Y2
[rt2] b2>0 & x>0 -> (rt2*b2*x) : (b2'=b2-1)
& (y2'=min(y2+1, y2Max));
// Y2 + SY -> SY + B2
[rt2] y2>0 & sy>0 -> (rt2*y2*sy) : (y2'=y2-
1) & (b2'=min(b2+1, b2Max));
// B2 + SY -> SY + X2
[rt2] b2>0 & sy>0 -> (rt2*b2*sy) : (b2'=b2-
1) & (x2'=min(x2+1, x2Max));

endmodule

rewards "x" true : x; endrewards
rewards "y" true : y; endrewards
rewards "b" true : b; endrewards

rewards "w" true : w; endrewards
rewards "z" true : z; endrewards
rewards "u" true : u; endrewards

rewards "p" true : p; endrewards
rewards "r" true : r; endrewards
rewards "q" true : q; endrewards

rewards "s" true : s; endrewards
rewards "t" true : t; endrewards

rewards "x2" true : x2; endrewards
rewards "y2" true : y2; endrewards
rewards "b2" true : b2; endrewards

rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards
```

```
rewards "time" true : 1; endrewards
```

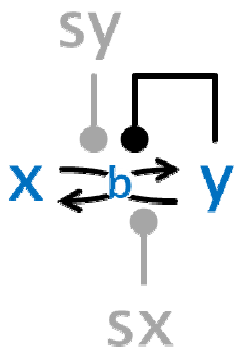
Query file (.csl)

```
const int i;
const int j;
S=? [ x=i&x2=j ]
```

Initial Parameters

```
xInit = yInit = wInit = zInit = pInit =
rInit = sInit = tInit = x2Init = y2Init =
sxInit = syInit =3
bInit = uInit = qInit = b2Init = 2
rt1 = 1.0
rt2 = 0.75
i = 0 .. 8 by 1
j = 0 .. 8 by 1
```

AMup (Fig S3-1a)



Chemical Reaction Model

```
x + y → y + b
b + y → y + y

x + sy → sy + b
b + sy → sy + y
y + sx → sx + b
b + sx → sx + x
```

Deterministic Model

```
dx/dt = (sx)*b - (y+sy)*x
db/dt = (sx)*y + (y+sy)*x - (y+sy)*b -
(sx)*b
y = xby-b-x

xby=90,sy=1, sx=0..60
```

XPPAUT / Oscill8 Source (.ode)

```
p xby=90
p sy=1, sx=0.1
y = xby-b-x
dx/dt = (sx)*b - (y+sy)*x
db/dt = (sx)*y + (y+sy)*x - (y+sy)*b -
(sx)*b
aux y=y
```

SPiM Source (.spi)

```
directive sample 2400.0 2000
val rt = 1.0
new xcat@rt:chan
new ycat@rt:chan
new sxcat@rt:chan new sxkill:chan
new sycat@rt:chan new sykill:chan
let x() =
   do ?ycat; b()
   or ?sycat; b()
and y() =
   do !ycat; y()
   or ?sxcat; b()
and b() =
   do ?sxcat; x()
   or ?ycat; y()
   or ?sycat; y()
and sy() = do !sycat; sy() or ?sykill; ()
and sx() = do !sxcat; sx() or ?sxkill; ()
run 60 of y()
run 60 of b()
run 60 of x()
run 2 of sy()
let clock(p:proc(int), t:float) =
(* Produce one p(m) every t sec with
precision dt,
    with m incremented from 0 *)
(val dt= 100.0 run step(p, 0, t, dt, dt))
and step(p:proc(int), m:int, t:float,
n:float, dt:float) =
if n<=0.0 then (p(m)|step(p,m+1,t,dt,dt))
else delay@dt/t; step(p,m,t,n-1.0,dt)
let schedule(n:int) =
  if n < 120 then sx()
  else if n < 240 then !sxkill;()
  else ()
run clock(schedule,10.0)
```

PRISM Source

Model file (.sm)

```
ctmc

const int xMax = xInit+yInit+bInit;
const int yMax = xInit+yInit+bInit;
const int bMax = xInit+yInit+bInit;
const int sxMax =  xInit+yInit+bInit;
const int syMax = xInit+yInit+bInit;

const int xInit;
const int yInit;
const int bInit;
const int sxInit;
const int syInit;

const double rt;

module AMUpperHalf

x : [0..xMax] init xInit;
y : [0..yMax] init yInit;
b : [0..bMax] init bInit;
sx : [0..sxMax] init sxInit;
sy : [0..syMax] init syInit;


// X + Y -> Y + B
[rt] x>0 & y>0 -> (rt*x*y) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + Y -> Y + Y
[rt] b>0 & y>0 -> (rt*b*y) : (b'=b-1) &
(y'=min(y+1, yMax));

// X + SY -> SY + B
[rt] x>0 & sy>0 -> (rt*x*sy) : (x'=x-1) &
(b'=min(b+1, bMax));
```

```
// B + SY -> SY + Y
[rt] b>0 & sy>0 -> (rt*b*sy) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + SX -> SX + B
[rt] y>0 & sx>0 -> (rt*y*sx) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + SX -> SX + X
[rt] b>0 & sx>0 -> (rt*b*sx) : (b'=b-1) &
(x'=min(x+1, xMax));

endmodule

rewards "x" true : x; endrewards
rewards "y" true : y; endrewards
rewards "b" true : b; endrewards
rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards

// rewards "x_sq" true : x*x; endrewards
// rewards "y_sq" true : y*y; endrewards
// rewards "b_sq" true : b*b; endrewards
// rewards "sx_sq" true : sx*sx; endrewards
// rewards "sy_sq" true : sy*sy; endrewards

rewards "time" true : 1; endrewards
```

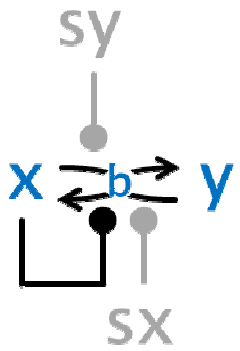
Query file (.csl)

```
const double T;
const int i;
S=? [ x=i ]
```

Initial Parameters

```
xInit = yInit = bInit = 30
sxInit = 0 .. 60 by 2
syInit = 1
rt = 1.0
i = 0 .. 90 by 3
```

AMdn (Fig S3-1b)



Chemical Reaction Model

$y + x \rightarrow x + b$
$b + x \rightarrow x + x$

$x + sy \rightarrow sy + b$
$b + sy \rightarrow sy + y$
$y + sx \rightarrow sx + b$

$b + sx \rightarrow sx + x$

Deterministic Model

```
dx/dt = (x+sx)*b - (sy)*x
db/dt = (x+sx)*y + (sy)*x - (sy)*b -
(x+sx)*b
y = xby-b-x

xby=90, sy=33, sx=0..30
```

XPPAUT / Oscill8 Source (.ode)

```
p xby=90
p sy=33, sx=0.1
y = xby-b-x
dx/dt = (x+sx)*b - (sy)*x
db/dt = (x+sx)*y + (sy)*x - (sy)*b -
(x+sx)*b
aux y=y
```

SPiM Source (.spi)

```
directive sample 1200.0 2000
val rt = 1.0
new xcat@rt:chan
new ycat@rt:chan
new sxcat@rt:chan new sxkill:chan
new sycat@rt:chan new sykill:chan
let x() =
   do !xcat; x()
   or ?sycat; b()
and y() =
   do ?xcat; b()
   or ?sxcat; b()
and b() =
   do ?xcat; x()
   or ?sxcat; x()
   or ?sycat; y()
and sy() = do !sycat; sy() or ?sykill; ()
and sx() = do !sxcat; sx() or ?sxkill; ()
run 60 of x()
run 60 of b()
run 60 of y()
run 66 of sy()
let clock(p:proc(int), t:float) =
(* Produce one p(m) every t sec with
precision dt,
    with m incremented from 0 *)
(val dt= 100.0 run step(p, 0, t, dt, dt))
and step(p:proc(int), m:int, t:float,
n:float, dt:float) =
if n<=0.0 then (p(m)|step(p,m+1,t,dt,dt))
else delay@dt/t; step(p,m,t,n-1.0,dt)
let schedule(n:int) =
  if n < 60 then sx()
  else if n < 120 then !sxkill;()
  else ()
run clock(schedule,10.0)
```

PRISM Source

Model file (.sm)

```
ctmc

const int xMax = xInit+yInit+bInit;
const int yMax = xInit+yInit+bInit;
const int bMax = xInit+yInit+bInit;
const int sxMax =  xInit+yInit+bInit;
const int syMax = xInit+yInit+bInit;

const int xInit;
const int yInit;
const int bInit;
const int sxInit;
const int syInit;
```

```
const double rt;

module AMLowerHalf

x : [0..xMax] init xInit;
y : [0..yMax] init yInit;
b : [0..bMax] init bInit;
sx : [0..sxMax] init sxInit;
sy : [0..syMax] init syInit;


// Y + X -> X + B
[rt] y>0 & x>0 -> (rt*y*x) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + X -> X + X
[rt] b>0 & x>0 -> (rt*b*x) : (b'=b-1) &
(x'=min(x+1, xMax));

// X + SY -> SY + B
[rt] x>0 & sy>0 -> (rt*x*sy) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + SY -> SY + Y
[rt] b>0 & sy>0 -> (rt*b*sy) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + SX -> SX + B
[rt] y>0 & sx>0 -> (rt*y*sx) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + SX -> SX + X
[rt] b>0 & sx>0 -> (rt*b*sx) : (b'=b-1) &
(x'=min(x+1, xMax));

endmodule

rewards "x" true : x; endrewards
rewards "y" true : y; endrewards
rewards "b" true : b; endrewards
rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards

// rewards "x_sq" true : x*x; endrewards
// rewards "y_sq" true : y*y; endrewards
// rewards "b_sq" true : b*b; endrewards
// rewards "sx_sq" true : sx*sx; endrewards
// rewards "sy_sq" true : sy*sy; endrewards

rewards "time" true : 1; endrewards
```

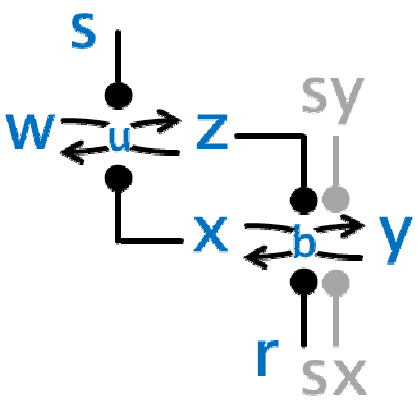
Query file (.csl)

```
const double T;
const int i;
S=? [ x=i ]
```

Initial Parameters

```
xInit = yInit = bInit = 30
sxInit = 0 .. 30 by 1
syInit = 33
rt = 1.0
i = 0 .. 90 by 3
```

CCnn (Fig S3-1c)



Chemical Reaction Model

```
x + z → z + b
b + z→ z + y
y + r → r + b
b + r → r + x

w + s → s + u
u + s → s + z
z + x → x + u
u + x → x + w

x + sy → sy + b
b + sy → sy + y
y + sx → sx + b
b + sx → sx + x
```

Deterministic Model

```
dw/dt = -s*w + x*u
du/dt = s*w + z*x - s*u - x*u
dx/dt = (r+sx)*b - (z+sy)*x
db/dt = (r+sx)*y + (z+sy)*x - (z+sy)*b -
(r+sx)*b
z = wuz-w-u
y = xby-b-x

wuz=30, xby=30, s=2, sy=2, sx=0..10, r= 2
```

XPPAUT / Oscill8 Source (.ode)

```
p wuz=30, xby=30
p s=2, sy=2, sx=0.1, r= 2
z = wuz-w-u
y = xby-b-x
dw/dt = -s*w + x*u
du/dt = s*w + z*x - s*u - x*u
dx/dt = (r+sx)*b - (z+sy)*x
db/dt = (r+sx)*y + (z+sy)*x - (z+sy)*b -
(r+sx)*b
aux z=z
aux y=y
```

SPiM Source (.spi)

```
directive sample 1000.0 2000
directive plot x()
val rt = 1.0
```

```
new xcat@rt:chan
new zcat@rt:chan
new scat@rt:chan
new tcat@rt:chan
new rcat@rt:chan
new sxcat@rt:chan new sxkill:chan
new sycat@rt:chan new sykill:chan
let x() =
   do !xcat; x()
   or ?zcat; b()
   or ?sycat; b()
and y() =
   do ?rcat; b()
   or ?sxcat; b()
and b() =
   do ?rcat; x()
   or ?sxcat; x()
   or ?zcat; y()
   or ?sycat; y()
and z() =
   do !zcat; z()
   or ?xcat; u()
and r() =
   !rcat; r()
and w() =
   ?scat; u()
and u() =
   do ?xcat; w()
   or ?scat; z()
and s() =
   !scat; s()
and sy() = do !sycat; sy() or ?sykill; ()
and sx() = do !sxcat; sx() or ?sxkill; ()
run 50 of y()
run 50 of x()
run 50 of b()
run 50 of z()
run 50 of w()
run 50 of u()
run 10 of s()
run 10 of r()
run 10 of sy()
let clock(ps:proc(int), tm:float) =
(* Produce one ps(m) every tm sec with
precision dt,
    with m incremented from 0 *)
(val dt= 100.0 run step(ps, 0, tm, dt, dt))
and step(ps:proc(int), m:int, tm:float,
n:float, dt:float) =
if n<=0.0 then
(ps(m)|step(ps,m+1,tm,dt,dt))
else delay@dt/tm; step(ps,m,tm,n-1.0,dt)
let schedule(n:int) =
  if n < 50 then sx()
  else if n < 100 then !sxkill;()
  else ()
run clock(schedule,10.0)
```

PRISM Source

Model file (.sm)

```
ctmc

const int xMax = xInit+yInit+bInit;
const int yMax = xInit+yInit+bInit;
const int bMax = xInit+yInit+bInit;

const int wMax = wInit+zInit+uInit;
const int zMax = wInit+zInit+uInit;
const int uMax = wInit+zInit+uInit;

const int rMax = xInit+yInit+bInit;

const int sMax = wInit+zInit+uInit;
const int sxMax = xInit+yInit+bInit;
const int syMax = xInit+yInit+bInit;

const int xInit;
const int yInit;
const int bInit;

const int wInit;
const int zInit;
const int uInit;

const int rInit;

const int sInit;
const int sxInit;
const int syInit;

const double rt;

module CC

x : [0..xMax] init xInit;
y : [0..yMax] init yInit;
b : [0..bMax] init bInit;

w : [0..wMax] init wInit;
z : [0..zMax] init zInit;
u : [0..uMax] init uInit;

r : [0..rMax] init rInit;

s : [0..sMax] init sInit;
sx: [0..sxMax] init sxInit;
sy: [0..syMax] init syInit;


// X + Z -> Z + B
[rt] x>0 & z>0 -> (rt*x*z) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + Z -> Z + Y
[rt] b>0 & z>0 -> (rt*b*z) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + R -> R + B
[rt] y>0 & r>0 -> (rt*y*r) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + R -> R + X
[rt] b>0 & r>0 -> (rt*b*r) : (b'=b-1) &
(x'=min(x+1, xMax));


// W + S -> S + U
[rt] w>0 & s>0 -> (rt*w*s) : (w'=w-1) &
(u'=min(u+1, uMax));
// U + S -> S + Z
[rt] u>0 & s>0 -> (rt*u*s) : (u'=u-1) &
(z'=min(z+1, zMax));
// Z + X -> X + U
[rt] z>0 & x>0 -> (rt*z*x) : (z'=z-1) &
(u'=min(u+1, uMax));
// U + X -> X + W
[rt] u>0 & x>0 -> (rt*u*x) : (u'=u-1) &
(w'=min(w+1, wMax));

// X + SY -> SY + B
[rt] x>0 & sy>0 -> (rt*x*sy) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + SY -> SY + Y
[rt] b>0 & sy>0 -> (rt*b*sy) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + SX -> SX + B
[rt] y>0 & sx>0 -> (rt*y*sx) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + SX -> SX + X
```

```
[rt] b>0 & sx>0 -> (rt*b*sx) : (b'=b-1) &
(x'=min(x+1, xMax));

endmodule

rewards "x" true : x; endrewards
rewards "y" true : y; endrewards
rewards "b" true : b; endrewards

rewards "w" true : w; endrewards
rewards "z" true : z; endrewards
rewards "u" true : u; endrewards

rewards "r" true : r; endrewards

rewards "s" true : s; endrewards

rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards

rewards "time" true : 1; endrewards
```

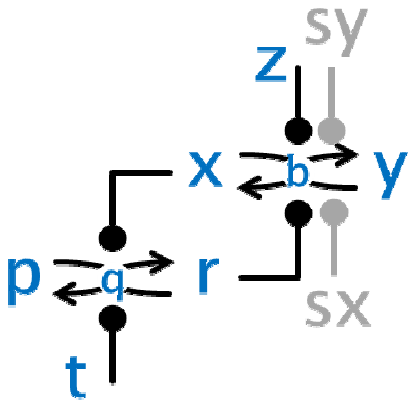
Query file (.csl)

```
const double T;
const int i;
S=?[x=i]
```

Initial Parameters

```
xInit = yInit = bInit = wInit = zInit =
uInit = 10
rInit = sInit = 2
sxInit = 0 .. 10 by 1
syInit = 2
rt = 1.0
i = 0 .. 30 by 1
```

CCpp (Fig S3-1d)



Chemical Reaction Model

```
x + z → z + b
b + z → z + y
y + r → r + b
b + r → r + x

p + x → x + q
q + x → x + r
r + t → t + q

q + t → t + p

x + sy → sy + b
b + sy → sy + y
y + sx → sx + b
b + sx → sx + x
```

Deterministic Model

```
(t is time: we use tt for species t)
dr/dt = x*q - tt*r
dq/dt = tt*r + x*pp - x*q - tt*q
dx/dt = (r+sx)*b - (z+sy)*x
db/dt = (r+sx)*y + (z+sy)*x - (z+sy)*b -
(r+sx)*b
pp = pqr-q-r
y = xby-b-x

z=10, pqr=30, xby=30, tt=10, sy=2, sx=0..10
```

XPPAUT / Oscill8 Source (.ode)

```
p z=10, pqr=30, xby=30
p tt=10, sy=2, sx=0.1
pp = pqr-q-r
y = xby-b-x
dr/dt = x*q - tt*r
dq/dt = tt*r + x*pp - x*q - tt*q
dx/dt = (r+sx)*b - (z+sy)*x
db/dt = (r+sx)*y + (z+sy)*x - (z+sy)*b -
(r+sx)*b
aux pp=pp
aux y=y
```

SPiM Source (.spi)

```
directive sample 1000.0 2000
directive plot x()
val rt = 1.0
new xcat@rt:chan
new zcat@rt:chan
new scat@rt:chan
new tcat@rt:chan
new rcat@rt:chan
new sxcat@rt:chan new sxkill:chan
new sycat@rt:chan new sykill:chan
let x() =
   do !xcat; x()
   or ?zcat; b()
   or ?sycat; b()
and y() =
   do ?rcat; b()
   or ?sxcat; b()
and b() =
   do ?rcat; x()
   or ?sxcat; x()
   or ?zcat; y()
   or ?sycat; y()
and z() =
  !zcat; z()
and r() =
   do !rcat; r()
   or ?tcat; q()
and t() =
   !tcat; t()
and p() =
   ?xcat; q()
and q() =
   do ?xcat; r()
   or ?tcat; p()
and sy() = do !sycat; sy() or ?sykill; ()
and sx() = do !sxcat; sx() or ?sxkill; ()
run 50 of y()
run 50 of x()
```

```
run 50 of b()
run 50 of p()
run 50 of r()
run 50 of q()
run 50 of z()
run 50 of t()
run 10 of sy()
let clock(ps:proc(int), tm:float) =
(* Produce one ps(m) every tm sec with
precision dt,
    with m incremented from 0 *)
(val dt= 100.0 run step(ps, 0, tm, dt, dt))
and step(ps:proc(int), m:int, tm:float,
n:float, dt:float) =
if n<=0.0 then
(ps(m)|step(ps,m+1,tm,dt,dt))
else delay@dt/tm; step(ps,m,tm,n-1.0,dt)
let schedule(n:int) =
  if n < 50 then sx()
  else if n < 100 then !sxkill;()
  else ()
run clock(schedule,10.0)
```

PRISM Source

Model file (.sm)

```
ctmc

const int xMax = xInit+yInit+bInit;
const int yMax = xInit+yInit+bInit;
const int bMax = xInit+yInit+bInit;

const int zMax = xInit+yInit+bInit;

const int pMax = pInit+rInit+qInit;
const int rMax = pInit+rInit+qInit;
const int qMax = pInit+rInit+qInit;

const int tMax = pInit+rInit+qInit;
const int sxMax = xInit+yInit+bInit;
const int syMax = xInit+yInit+bInit;

const int xInit;
const int yInit;
const int bInit;

const int zInit;

const int pInit;
const int rInit;
const int qInit;

const int tInit;
const int sxInit;
const int syInit;

const double rt;

module CC

x : [0..xMax] init xInit;
y : [0..yMax] init yInit;
b : [0..bMax] init bInit;

z : [0..zMax] init zInit;

p : [0..pMax] init pInit;
r : [0..rMax] init rInit;
q : [0..qMax] init qInit;

t : [0..tMax] init tInit;
sx: [0..sxMax] init sxInit;
sy: [0..syMax] init syInit;

// X + Z -> Z + B
[rt] x>0 & z>0 -> (rt*x*z) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + Z -> Z + Y
[rt] b>0 & z>0 -> (rt*b*z) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + R -> R + B
[rt] y>0 & r>0 -> (rt*y*r) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + R -> R + X
[rt] b>0 & r>0 -> (rt*b*r) : (b'=b-1) &
(x'=min(x+1, xMax));

// P + X -> X + Q
[rt] p>0 & x>0 -> (rt*p*x) : (p'=p-1) &
(q'=min(q+1, qMax));
// Q + X -> X + R
[rt] q>0 & x>0 -> (rt*q*x) : (q'=q-1) &
(r'=min(r+1, rMax));
// R + T -> T + Q
[rt] r>0 & t>0 -> (rt*r*t) : (r'=r-1) &
(q'=min(q+1, qMax));
// Q + T -> T + P
[rt] q>0 & t>0 -> (rt*q*t) : (q'=q-1) &
(p'=min(p+1, pMax));

// X + SY -> SY + B
[rt] x>0 & sy>0 -> (rt*x*sy) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + SY -> SY + Y
[rt] b>0 & sy>0 -> (rt*b*sy) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + SX -> SX + B
[rt] y>0 & sx>0 -> (rt*y*sx) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + SX -> SX + X
[rt] b>0 & sx>0 -> (rt*b*sx) : (b'=b-1) &
(x'=min(x+1, xMax));

endmodule

rewards "x" true : x; endrewards
rewards "y" true : y; endrewards
rewards "b" true : b; endrewards

rewards "z" true : z; endrewards

rewards "p" true : p; endrewards
rewards "r" true : r; endrewards
rewards "q" true : q; endrewards

rewards "t" true : t; endrewards
rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards

rewards "time" true : 1; endrewards
```

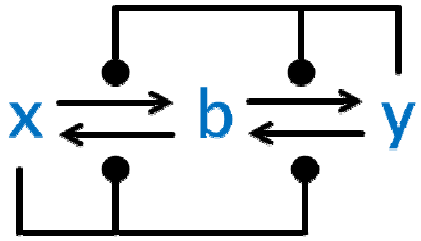
Query file (.csl)

```
const double T;
const int i;
S=?[x=i]
```

Initial Parameters

```
xInit = yInit = bInit = zInit = pInit =
rInit = qInit = tInit = 10
sxInit = 0 .. 10 by 1
syInit = 2
rt = 1.0
i = 0 .. 30 by 1
```

Distributive Modification (Fig 4a)



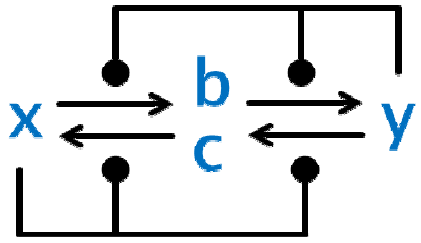
Chemical Reaction Model

x + y → y + b
y + x → x + b
b + x → x + x
b + y → y + y

SPiM Source (.spi)

```
directive sample 2.0 1000
directive plot x()
val r = 0.001
new xcat@r:chan
new ycat@r:chan
let x() =
   do !xcat; x()
   or ?ycat; b()
and y() =
   do !ycat; y()
   or ?xcat; b()
and b() =
   do ?xcat; x()
   or ?ycat; y()
run 7500 of x()
run 7500 of y()
```

Processive Modification (Fig 4b)



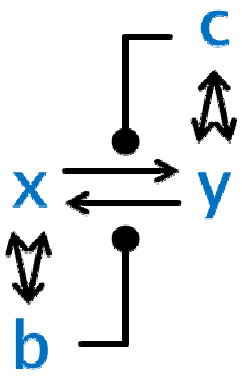
Chemical Reaction Model

x + y → y + b
y + x → x + c
c + x → x + x
b + y → y + y

SPiM Source (.spi)

```
directive sample 20.0 1000
directive plot x(); b()
val r = 0.001
new xcat@r:chan
new ycat@r:chan
let x() =
   do !xcat; x()
   or ?ycat; b()
and y() =
   do !ycat; y()
   or ?xcat; c()
and b() =
   ?ycat; y()
and c() =
   ?xcat; x()
run 7500 of x()
run 7500 of y()
```

Dimerization (Fig 4c)



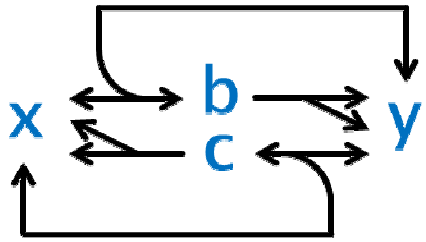
Chemical Reaction Model

x + x → b
y + y → c
y + b → b + x
x + c → c + y

SPiM Source (.spi)

```
directive sample 20.0 1000
directive plot x(); x2()
val r = 0.001
val a = 0.001
val d = 1.0
new x2cat@r:chan
new y2cat@r:chan
new xdim@a:chan
new ydim@a:chan
let x() =
   do !xdim; x2()
   or ?xdim; ()
   or ?y2cat; y()
and x2() =
  do !x2cat; x2()
  or delay@d; (x()|x())
and y() =
   do !ydim; y2()
   or ?ydim; ()
   or ?x2cat; x()
and y2() =
  do !y2cat; y2()
  or delay@d; (y()|y())
run 7500 of x()
run 7500 of y()
```

Direct Enzymatic (Fig 4d)



Chemical Reaction Model

$x + y \rightarrow_a b \rightarrow_d x + y$

$y + x \rightarrow_a c \rightarrow_d y + x$

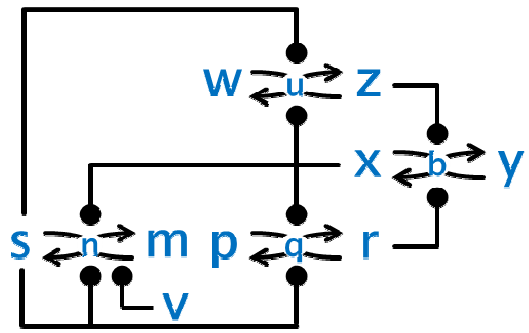
$b \rightarrow_k y + y$

$c \rightarrow_k x + x$

SPiM Source (.spi)

```
directive sample 20.0 1000
directive plot x(); b()
val a = 0.001
val d = 1.0
val k = 1.0
new xcat@a:chan new ycat@a:chan
let x() = do !xcat; c() or ?ycat; ()
and b() = do delay@d; (y()|x()) or delay@k;
(y()|y())
and y() = do !ycat; b() or ?xcat; ()
and c() = do delay@d; (x()|y()) or delay@k;
(x()|x())
run 7500 of x()
run 7500 of y()
```

GW (Fig 5)



Chemical Reaction Model

$x + z \rightarrow z + b$

$b + z \rightarrow z + y$

$y + r \rightarrow r + b$

$b + r \rightarrow r + x$

$w + s \rightarrow s + u$

$u + s \rightarrow s + z$

$z + x \rightarrow x + u$

$u + x \rightarrow x + w$

$p + x \rightarrow x + q$

$q + x \rightarrow x + r$

$r + s \rightarrow s + q$

$q + s \rightarrow s + p$

$s + x \rightarrow x + n$

$n + x \rightarrow x + m$

$m + s \rightarrow s + n$

$n + s \rightarrow s + s$

$m + v \rightarrow v + n$

$n + v \rightarrow v + s$

Deterministic Model

```
dw/dt = -s*w + x*u
du/dt = s*w + z*x - s*u - x*u
dr/dt = x*q - s*r
dq/dt = s*r + x*pp - x*q - s*q
dx/dt = r*b - z*x
db/dt = r*y + z*x - z*b - r*b
dm/dt = x*n - (s+v)*m
dn/dt = x*s + (s+v)*m - (s+v)*n - x*n
z = wuz-w-u
pp = pqr-q-r
y = xby-b-x
s = snm-n-m

wuz=9, pqr=9, xby=9, snm=9, sx=0,v=0
```

XPPAUT / Oscill8 Source (.ode)

```
init w=3, u=3, r=3, q=3, x=3, b=3, m=3, n=3
p wuz=9, pqr=9, xby=9, snm=9
p v=0
z = wuz-w-u
pp = pqr-q-r
y = xby-b-x
dw/dt = s*w + x*u
du/dt = s*w + z*x - s*u - x*u
dr/dt = x*q - s*r
dq/dt = s*r + x*pp - x*q - s*q
dx/dt = r*b - z*x
db/dt = r*y + z*x - z*b - r*b
aux z=z
aux pp=pp
aux y=y
```

SPiM Source (.spi)

```
directive sample 0.002626 2000
directive plot x()
val rt = 1.0
new xcat@rt:chan
new zcat@rt:chan
new rcat@rt:chan
new scat@rt:chan
new vcat@rt:chan
let x() =
   do !xcat; x()
   or ?zcat; b()
and b() =
   do ?rcat; x()
   or ?zcat; y()
and y() =
   ?rcat; b()
and z() =
   do !zcat; z()
   or ?xcat; u()
and u() =
   do ?xcat; w()
   or ?scat; z()
and w() =
   ?scat; u()
and r() =
```

```
   do !rcat; r()
   or ?scat; q()
and q() =
   do ?xcat; r()
   or ?scat; p()
and p() =
   ?xcat; q()
and s() =
   do !scat; s()
   or ?xcat(); n()
and n() =
   do ?xcat; m()
   or ?scat; s()
   or ?vcat; s()
and m() =
   do ?scat; n()
   or ?vcat; n()
and v() =
   !vcat; v()
run 5000 of x()
run 5000 of b()
run 5000 of y()
run 5000 of z()
run 5000 of u()
run 5000 of w()
run 5000 of r()
run 5000 of q()
run 5000 of p()
run 5000 of s()
run 5000 of n()
run 5000 of m()
run 0 of v()
```

PRISM Source

Model file (.sm)

```
ctmc

const int xMax = xInit+yInit+bInit;
const int yMax = xInit+yInit+bInit;
const int bMax = xInit+yInit+bInit;

const int wMax = wInit+zInit+uInit;
const int zMax = wInit+zInit+uInit;
const int uMax = wInit+zInit+uInit;

const int pMax = pInit+rInit+qInit;
const int rMax = pInit+rInit+qInit;
const int qMax = pInit+rInit+qInit;

const int sMax = sInit+mInit+nInit;
const int mMax = sInit+mInit+nInit;
const int nMax = sInit+mInit+nInit;

const int vMax = sInit+mInit+nInit;

const int sxMax = xInit+yInit+bInit;
const int syMax = xInit+yInit+bInit;

const int xInit;
const int yInit;
const int bInit;

const int wInit;
const int zInit;
const int uInit;

const int pInit;
const int rInit;
const int qInit;

const int sInit;
const int mInit;
const int nInit;

const int vInit;

const int sxInit;
const int syInit;

const double rt;

module GW

x : [0..xMax] init xInit;
y : [0..yMax] init yInit;
b : [0..bMax] init bInit;

w : [0..wMax] init wInit;
z : [0..zMax] init zInit;
u : [0..uMax] init uInit;

p : [0..pMax] init pInit;
r : [0..rMax] init rInit;
q : [0..qMax] init qInit;

s : [0..sMax] init sInit;
m : [0..mMax] init mInit;
n : [0..nMax] init nInit;

v : [0..vMax] init vInit;

sx: [0..sxMax] init sxInit;
sy: [0..syMax] init syInit;

// X + Z -> Z + B
[rt] x>0 & z>0 -> (rt*x*z) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + Z -> Z + Y
[rt] b>0 & z>0 -> (rt*b*z) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + R -> R + B
[rt] y>0 & r>0 -> (rt*y*r) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + R -> R + X
[rt] b>0 & r>0 -> (rt*b*r) : (b'=b-1) &
(x'=min(x+1, xMax));

// W + S -> S + U
[rt] w>0 & s>0 -> (rt*w*s) : (w'=w-1) &
(u'=min(u+1, uMax));
// U + S -> S + Z
[rt] u>0 & s>0 -> (rt*u*s) : (u'=u-1) &
(z'=min(z+1, zMax));
// Z + X -> X + U
[rt] z>0 & x>0 -> (rt*z*x) : (z'=z-1) &
(u'=min(u+1, uMax));
// U + X -> X + W
[rt] u>0 & x>0 -> (rt*u*x) : (u'=u-1) &
(w'=min(w+1, wMax));

// P + X -> X + Q
[rt] p>0 & x>0 -> (rt*p*x) : (p'=p-1) &
(q'=min(q+1, qMax));
// Q + X -> X + R
[rt] q>0 & x>0 -> (rt*q*x) : (q'=q-1) &
(r'=min(r+1, rMax));
// R + S -> S + Q
[rt] r>0 & s>0 -> (rt*r*s) : (r'=r-1) &
(q'=min(q+1, qMax));
// Q + S -> S + P
[rt] q>0 & s>0 -> (rt*q*s) : (q'=q-1) &
(p'=min(p+1, pMax));

// S + X -> X + N
[rt] s>0 & x>0 -> (rt*s*x) : (s'=s-1) &
(n'=min(n+1, nMax));
// N + X -> X + M
```

```
[rt] n>0 & x>0 -> (rt*n*x) : (n'=n-1) &
(m'=min(m+1, mMax));
// M + S -> S + N
[rt] m>0 & s>0 -> (rt*m*s) : (m'=m-1) &
(n'=min(n+1, nMax));
// N + S -> S + S
[rt] n>0 & s>0 -> (rt*n*s) : (n'=n-1) &
(s'=min(s+1, sMax));

// M + V -> V + N
[rt] m>0 & v>0 -> (rt*m*v) : (m'=m-1) &
(n'=min(n+1, nMax));
// N + V -> V + S
[rt] n>0 & v>0 -> (rt*n*v) : (n'=n-1) &
(s'=min(s+1, sMax));

// X + SY -> SY + B
[rt] x>0 & sy>0 -> (rt*x*sy) : (x'=x-1) &
(b'=min(b+1, bMax));
// B + SY -> SY + Y
[rt] b>0 & sy>0 -> (rt*b*sy) : (b'=b-1) &
(y'=min(y+1, yMax));
// Y + SX -> SX + B
[rt] y>0 & sx>0 -> (rt*y*sx) : (y'=y-1) &
(b'=min(b+1, bMax));
// B + SX -> SX + X
[rt] b>0 & sx>0 -> (rt*b*sx) : (b'=b-1) &
(x'=min(x+1, xMax));

endmodule

rewards "x" true : x; endrewards
rewards "y" true : y; endrewards
rewards "b" true : b; endrewards

rewards "w" true : w; endrewards
rewards "z" true : z; endrewards
rewards "u" true : u; endrewards

rewards "p" true : p; endrewards
rewards "r" true : r; endrewards
rewards "q" true : q; endrewards

rewards "s" true : s; endrewards
rewards "m" true : m; endrewards
rewards "n" true : n; endrewards

rewards "v" true : v; endrewards

rewards "sx" true : sx; endrewards
rewards "sy" true : sy; endrewards

rewards "time" true : 1; endrewards
```

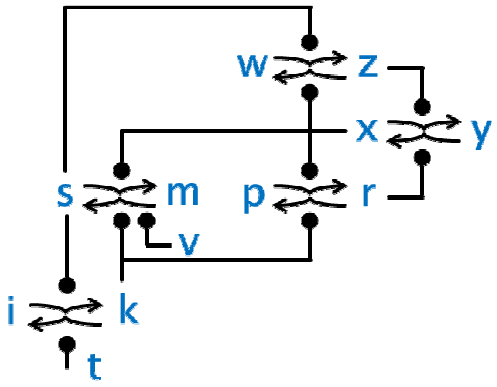
Query file (.csl)

```
const double T;
const int i;
P=? [F[T,T]x=i]
```

Initial Parameters

```
xInit = yInit = bInit = wInit = zInit =
uInit = pInit = rInit = qInit = sInit =
tInit = mInit = nInit = 3
vInit = 0
rt = 1.0
i = 0 .. 9 by 1
T = 0 .. 1 by 0.05
(sxInit = syInit = 0)
```

GW-Indirect (Fig S5-1)

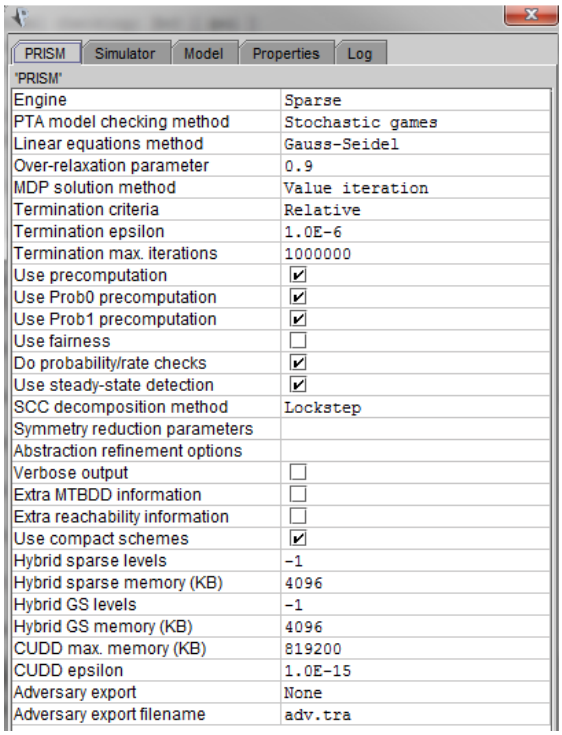


SPiM Source (.spi)

```
directive sample 0.002626  2000
directive plot x()
val rt = 1.0
new xcat@rt:chan
new zcat@rt:chan
new rcat@rt:chan
new scat@rt:chan
new vcat@rt:chan
new kcat@rt:chan
new tcat@rt:chan
let x() =
   do !xcat; x()
   or ?zcat; b()
and b() =
   do ?rcat; x()
   or ?zcat; y()
and y() =
   ?rcat; b()
and z() =
   do !zcat; z()
   or ?xcat; u()
and u() =
   do ?xcat; w()
   or ?scat; z()
and w() =
   ?scat; u()
and r() =
   do !rcat; r()
   or ?kcat; q()
and q() =
   do ?xcat; r()
   or ?kcat; p()
and p() =
   ?xcat; q()
and s() =
   do !scat; s()
   or ?xcat(); n()
and n() =
   do ?xcat; m()
   or ?kcat; s()
   or ?vcat; s()
and m() =
   do ?kcat; n()
   or ?vcat; n()
and i() =
   ?scat; j()
and j() =
   do ?scat; k()
   or ?tcat; i()
and k() =
   do!kcat; k()
   or ?tcat; j()
and t() =
  !tcat; t()
```

```
and v() =
   !vcat; v()
run 5000 of x()
run 5000 of b()
run 5000 of y()
run 5000 of z()
run 5000 of u()
run 5000 of w()
run 5000 of r()
run 5000 of q()
run 5000 of p()
run 5000 of s()
run 5000 of n()
run 5000 of m()
run 5000 of i()
run 5000 of j()
run 5000 of k()
run 5000 of t()
run 0 of v()
```

PRISM v4.0.3 Settings



Engine	Sparse
PTA model checking method	Stochastic games
Linear equations method	Gauss-Seidel
Over-relaxation parameter	0.9
MDP solution method	Value iteration
Termination criteria	Relative
Termination epsilon	1.0E-6
Termination max. iterations	1000000
Use precomputation	☑
Use Prob0 precomputation	☑
Use Prob1 precomputation	☑
Use fairness	☐
Do probability/rate checks	☑
Use steady-state detection	☑
SCC decomposition method	Lockstep
Symmetry reduction parameters	
Abstraction refinement options	
Verbose output	☐
Extra MTBDD information	☐
Extra reachability information	☐
Use compact schemes	☑
Hybrid sparse levels	-1
Hybrid sparse memory (KB)	4096
Hybrid GS levels	-1
Hybrid GS memory (KB)	4096
CUDD max. memory (KB)	819200
CUDD epsilon	1.0E-15
Adversary export	None
Adversary export filename	adv.tra

Supplementary References

1. Novak B, Tyson JJ (1993) Numerical analysis of a comprehensive model of M-phase control in Xenopus oocyte extracts and intact embryos. J Cell Sci 106 (Pt 4): 1153-1168.

2. Angluin D, Aspnes J, Eisenstat D (2008) A simple population protocol for fast robust approximate majority. Distributed Computing 21: 87-102.

3. Domingo-Sananes MR, Novak B (2010) Different effects of redundant feedback loops on a bistable switch. Chaos 20: 045120.

4. Ferrell JE, Jr. (2008) Feedback regulation of opposing enzymes generates robust, all-or-none bistable responses. Curr Biol 18: R244-245.

5. Ciliberto A, Capuani F, Tyson JJ (2007) Modeling networks of coupled enzymatic reactions using the total quasi-steady state approximation. PLoS Comput Biol 3: e45.

6. Sangwin C (2009) The wonky trammel of Archimedes. Teaching Mathematics and its Applications 28: 48-52.

7. Krasinska L, Domingo-Sananes Maria R, Kapuy O, Parisis N, Harker B, et al. (2011) Protein Phosphatase 2A Controls the Order and Dynamics of Cell-Cycle Transitions. Molecular Cell 44: 437-450.

8. Novak B, Vinod PK, Freire P, Kapuy O (2010) Systems-level feedback in cell-cycle control. Biochem Soc Trans 38: 1242-1246.