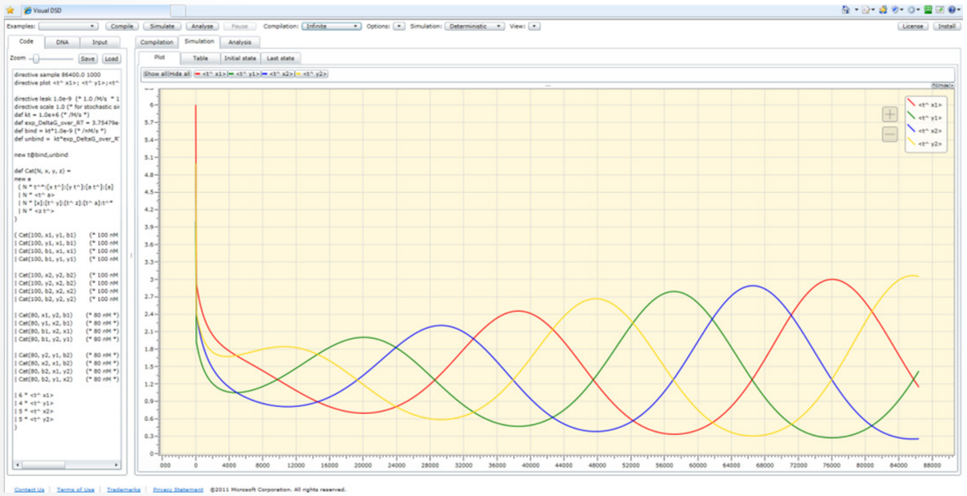
Network Transformations of Switches and Oscillators

Luca Cardelli
Microsoft Research

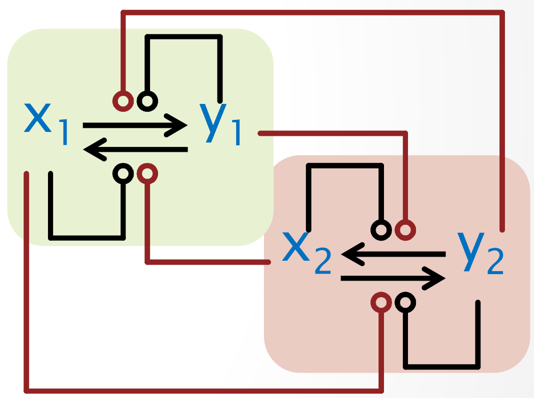
Cambridge 2011-08-02
http://lucacardelli.name

Motivation

- Building synthetic (DNA) oscillators

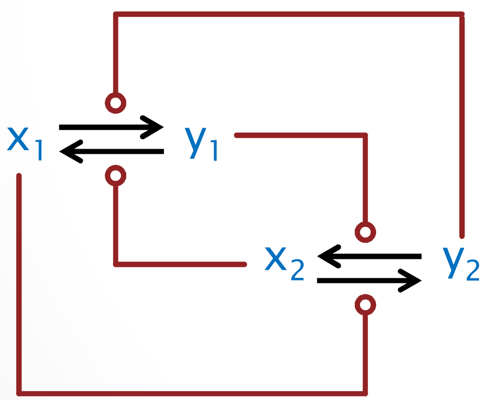


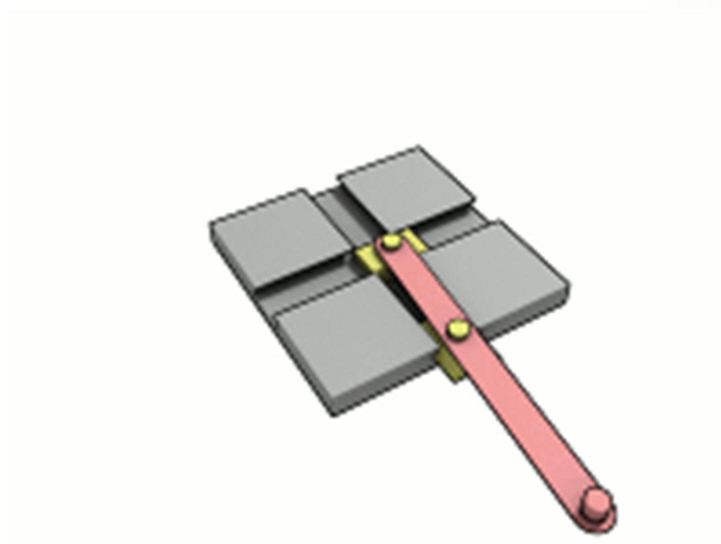
DSD simulation



The Trammel of Archimedes

- A device to draw ellipses
  - Two interconnected switches.
  - Note that amplitude is kept constant by mechanical constraints.
  - When one switch is on (off) it flips the other switch on (off). When the other switch is on (off) it flips the first switch off (on).

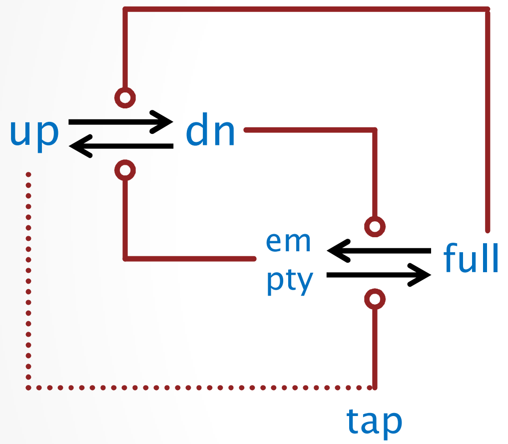




en.wikipedia.org/wiki/Trammel_of_Archimedes

The Shishi Odoshi

- A Japanese scarecrow (scare-deer)
  - Used by Bela Novak to illustrate the cell cycle switch.



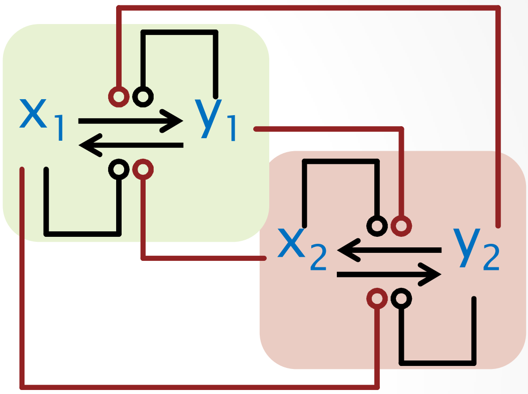
http://www.youtube.com/watch?v=VbvecTIftcE&NR=1&feature=fvwp

empty + tap → tap + full
up + full → full + dn
full + dn → dn + empty
dn + empty → empty + up

To make it into a full trammel (dotted line), we could make the up position mechanically open the tap (i.e. take up = tap)

The Cell Cycle

- Feedback speeds
  - fast (post-translational)
  - slow (transcriptional)
- Some feedbacks may be missing
- Switches are asymmetric
  - One switch is usually simpler than that, just causing a negative feedback
  - One switch is usually more sophisticated than that, because of biochemical constraints



Outline

- Questions that nature has answered
  - Building ‘good’ bistable systems
  - Building ‘switches’ (switchable bistable system)
  - Building switches with hysteresys (needed for good oscillators)
  - Building limit-cycle oscillators
  - Building robust oscillators that resist parameter variations

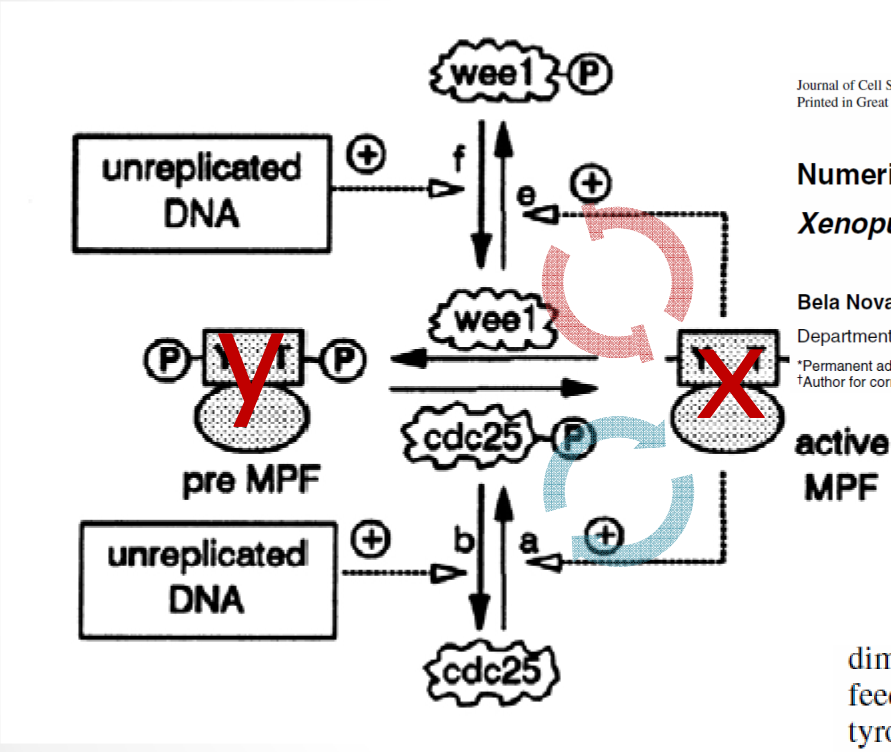
- Engineering solutions to the same problems
  - Are they related?
  - In nature there are chemical constraints
    - Not all reactions can be easily implemented
    - Not all molecules can perform all functions we want them to

- From the point of view of network structure
  - Transforming a network and preserve some function
  - “Program transformations”

Switches

The Cell Cycle Switch

Why this network structure?



Journal of Cell Science 106, 1153-1168 (1993)
Printed in Great Britain © The Company of Biologists Limited 1993

Numerical analysis of a comprehensive model of M-phase control in *Xenopus* oocyte extracts and intact embryos

Bela Novak* and John J. Tyson†

Department of Biology, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24060-0406, USA

*Permanent address: Department of Agricultural Chemical Technology, Technical University of Budapest, 1521 Budapest Gellert Ter 4, Hungary
†Author for correspondence

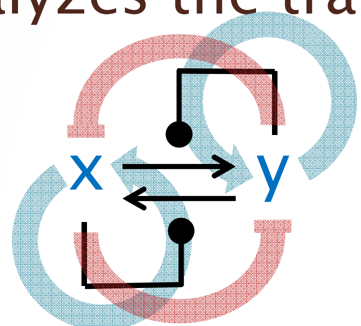
- Double positive feedback on x
- Double negative feedback on x
- No feedback on y

Why on earth .... ??

dimers is left off the diagram to keep it simple.) (B) Positive feedback loops. Active MPF stimulates its own production from tyrosine-phosphorylated dimers by activating Cdc25 and inhibiting Wee1. We suspect that these signals are indirect, but intermediary enzymes are unknown and we ignore them in this paper. The signals from active MPF to Wee1 and Cdc25 generate an autocatalytic instability in the control system. We indicate also an 'external' signal from unreplicated DNA to Wee1 and Cdc25, which can be used to control the efficacy of the positive feedback loops. The letters a, b, e and f are used to label the rate constants for these reactions in Fig. 2. (C) Negative feedback loop. Active

A Bad Algorithm

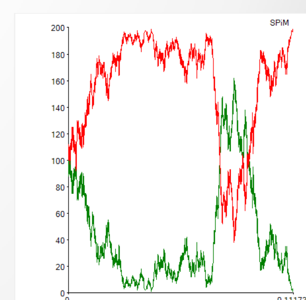
- Direct x–y competition
  - x catalyzes the transformation of y into x
  - y catalyzes the transformation of x into y



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

- This system is bistable, but
  - Convergence to a stable state is slow (a random walk).
  - *Any* perturbation of a stable state can initiate a random walk to the other stable state.
  - With 100 molecules of x and y, convergence is quick, but with 10000 molecules, even at the same concentration, you will wait for a long time.

```
directive sample 0.0002
1000
directive plot x(); y(); b()

val rt = 10.0
new xcat@rt:chan
new ycat@rt:chan

let x() =
  do !xcat; x()
  or ?ycat; y()

and y() =
  do !ycat; y()
  or ?xcat; x()

run 100 of x()
run 100 of y()
```



A Very Good Algorithm

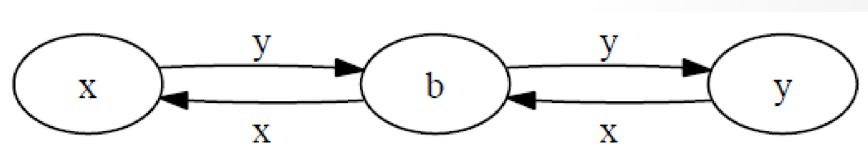
- Approximate Majority
  - Decide which of two populations is in majority
- A fundamental ‘population protocol’
  - Agents in a population start in state x or state y.
  - A pair of agents is chosen randomly at each step, they interact ("collide") and change state.
  - The whole population must eventually agree on a majority value (all x or all y) with probability 1.

Dana Angluin · James Aspnes · David Eisenstat

A Simple Population Protocol for Fast Robust Approximate Majority

We analyze the behavior of the following population protocol with states $Q = \{b, x, y\}$. The state b is the **blank** state. Row labels give the initiator's state and column labels the responder's state.

	x	b	y
x	(x, x)	(x, x)	(x, b)
b	(b, x)	(b, b)	(b, y)
y	(y, b)	(y, y)	(y, y)



Properties

- Using martingales, we show that with high probability,
  - The number of state changes before converging is $O(n \log n)$
  - The total number of interactions before converging is $O(n \log n)$
  - The final outcome is correct if the initial disparity is $\omega(\sqrt{n \log n})$
- This algorithm is the fastest possible
  - Must wait $\Omega(n \log n)$ steps in expectation for all agents to interact

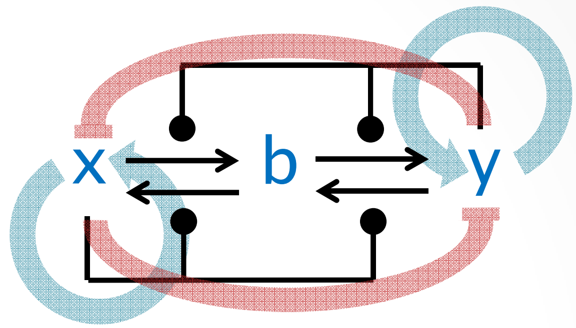
[Angluin et al.]

“Parallel time” is the number of steps divided by the number of agents. Hence the algorithm terminates with high probability in O(log n) steps per agent.

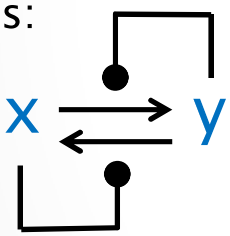
N.B. this bound holds even if the x,y populations are initially of equal size!

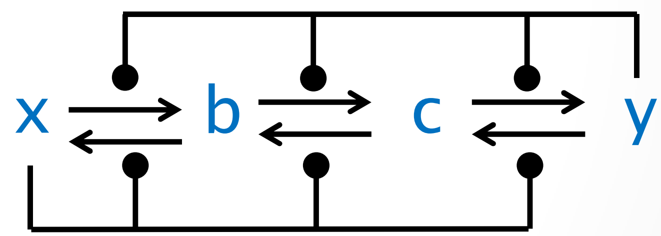
Chemical Implementation

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



Alternatives:





This too is a bistable system, but:

- It converges slowly, by a random walk, hence $O(n^2)$.
- It is unstable: any random fluctuation from an all-x or all-y state can send it (by a random walk) to the other state.

This one gives no significant improvement over the above.

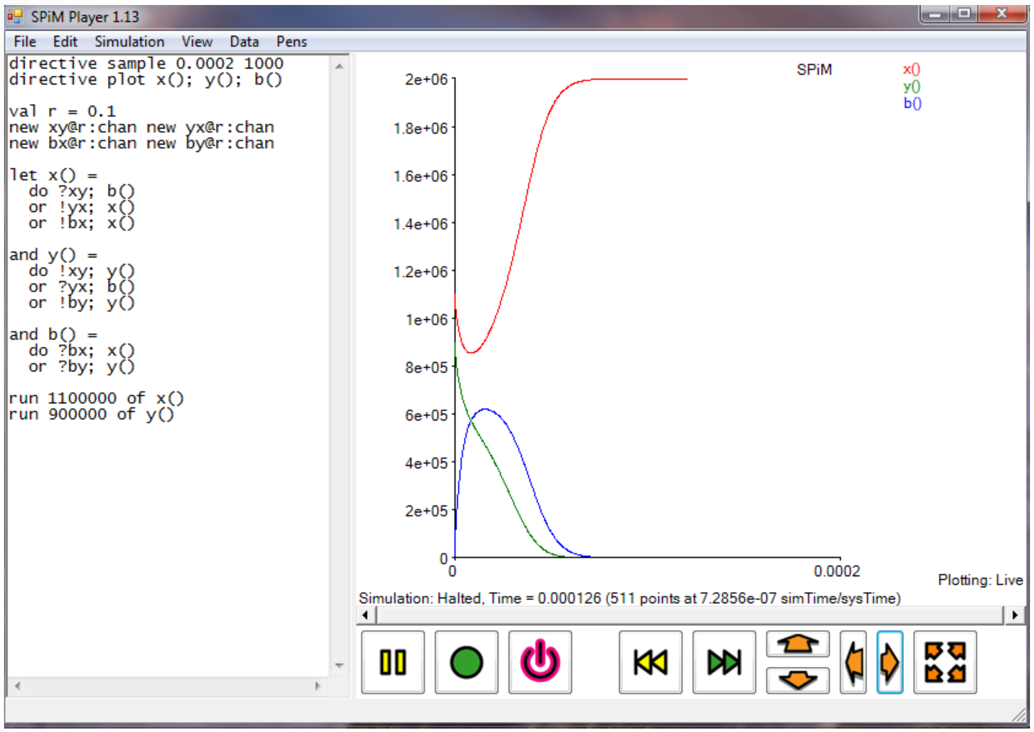
Majority of x>y

```
directive sample 0.0002 1000
directive plot x(); y(); b()

val r = 0.1
new xy@r:chan new yx@r:chan

new bx@r:chan new by@r:chan

let x() =
  do ?xy; b()
  or !yx; x()
  or !bx; x()

and y() =
  do !xy; y()
  or ?yx; b()
  or !by; y()

and b() =
  do ?bx; x()
  or ?by; y()

run 1000000 of x()
run 1000000 of y()
```

2000k molecules
1100k x
900k y

Gillespie simulation of the chemical reactions in SPiM.

All rates are equal.



$x + y \to y + b$
$y + x \to x + b$
$b + x \to x + x$
$b + y \to y + y$

Eventually:
all x
no y
no b

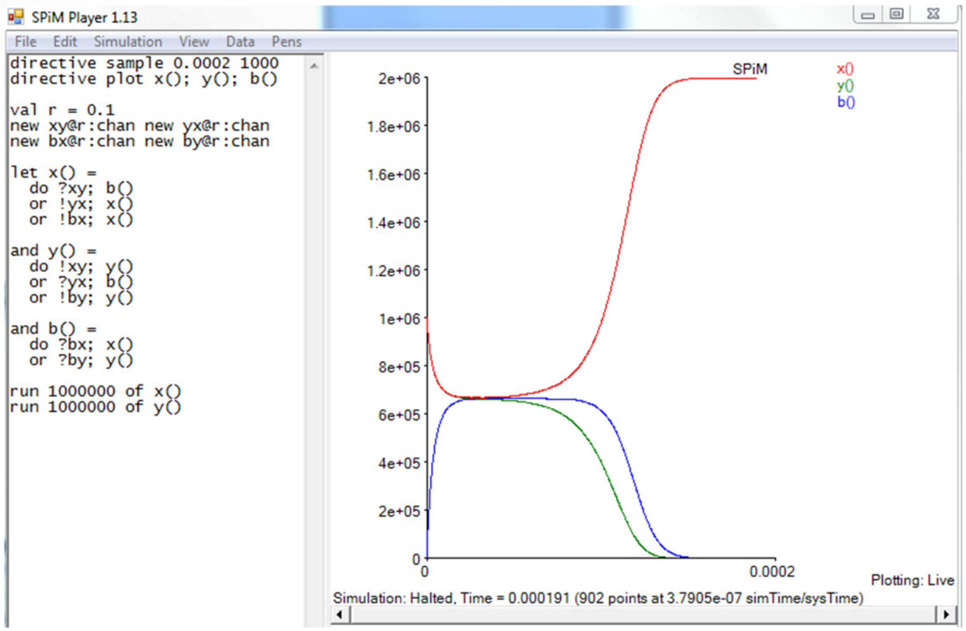
Majority of x=y (!!)

```
directive sample 0.0002 1000
directive plot x(); y(); b()

val r = 0.1
new xy@r:chan new yx@r:chan

new bx@r:chan new by@r:chan

let x() =
  do ?xy; b()
  or !yx; x()
  or !bx; x()

and y() =
  do !xy; y()
  or ?yx; b()
  or !by; y()

and b() =
  do ?bx; x()
  or ?by; y()

run 1000000 of x()
run 1000000 of y()
```

2000k molecules

Gillespie simulation of the chemical reactions in SPiM.

All rates are equal.



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

Eventually either:

all x	all y
no y	no x
no b	no b

The final majority is robust (insensitive to possible noise) because a significant majority always stays a majority:

The final outcome is correct if the initial disparity is $\omega(\sqrt{n}\log n)$

N.B. a deterministic (ODE) simulation with x=y would not converge ever!

A Digression about Other Switches

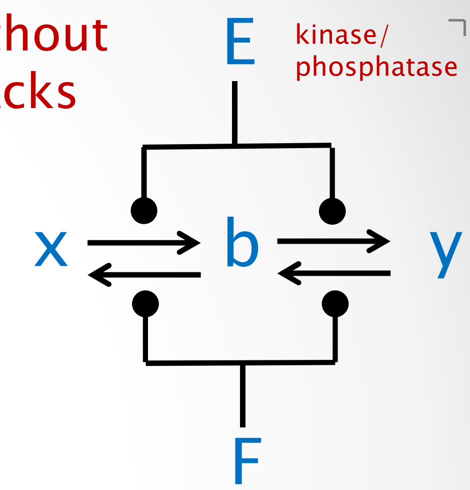
- The AM network is an ‘optimal’ switch in a computational sense. How does it compare with other switches?
- Let us first compare the ‘kernel’ of AM without feedbacks (i.e. ‘double phosphorylation’) with the Goldbeter-Koshland switch
- And then compare the full AM network with GK plus the same feedbacks as AM

Double–Phosphorylation Switch

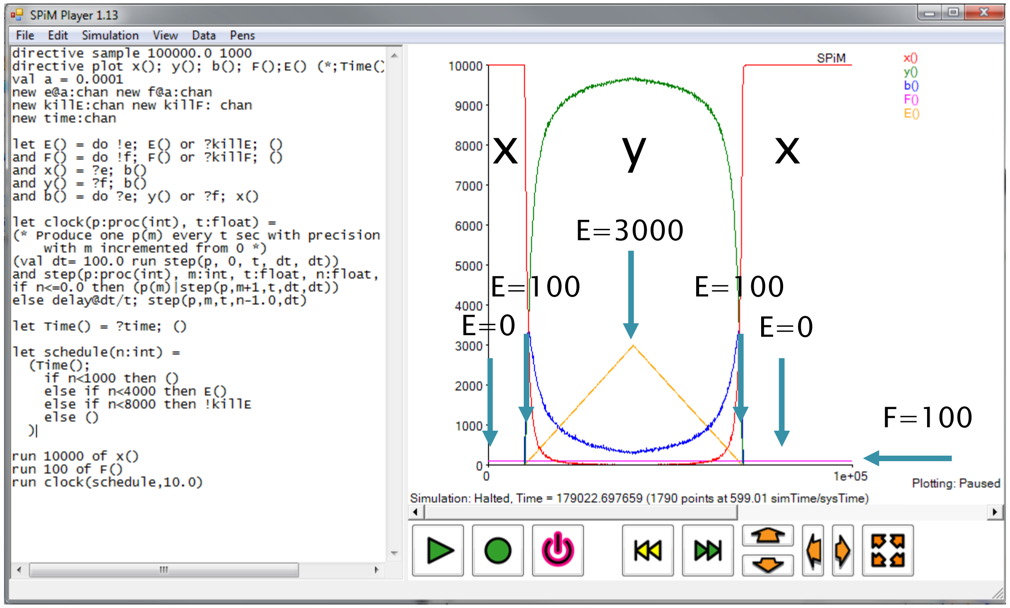
Ultrasensitive
(but no hysteresis)

x + E → E + b
b + E → E + y
y + F → F + b
b + F → F + x

AM without feedbacks




```
directive sample 100000.0 1000
directive plot x(); y(); b(); F();E() (*;Time() *)
val a = 0.0001
new e@a:chan new f@a:chan
new killE:chan new killF: chan
new time:chan

let E() = do !e; E() or ?killE; ()
and F() = do !f; F() or ?killF; ()
and x() = ?e; b()
and y() = ?f; b()
and b() = do ?e; y() or ?f; x()

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 Time() = ?time; ()

let schedule(n:int) =
  (Time();
    if n<1000 then ()
    else if n<4000 then E()
    else if n<8000 then !killE
    else ()
  )

run 10000 of x()
run 100 of F()
run clock(schedule,10.0)
```

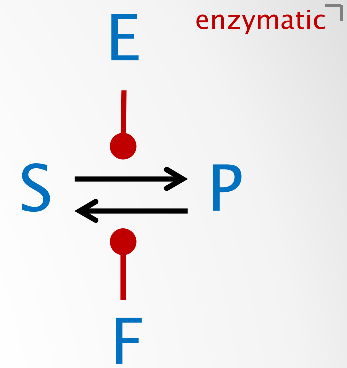


Initially 10000 x, no y, 100 F, no E.
E growing from 0 (t=100) to 3000 (t=400) then back to 0 (t=800)

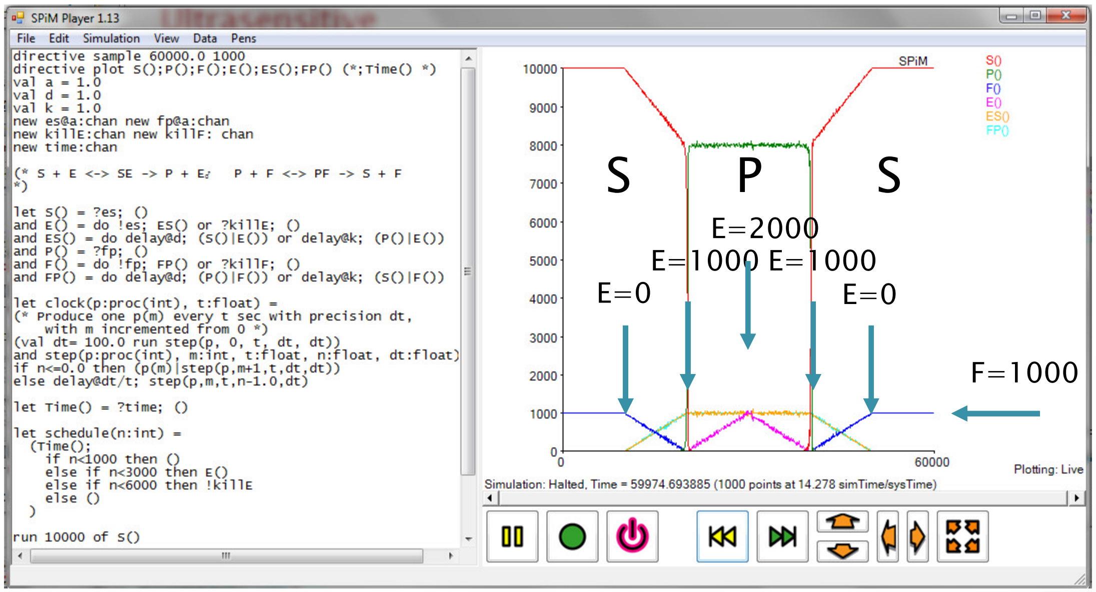
The Goldbeter–Koshland Switch

Ultrasensitive
(but no hysteresis)

$$S + E \ {}_{d}\!\leftrightarrow_{a} SE \rightarrow_{k} P + E$$
$$P + F \ {}_{d}\!\leftrightarrow_{a} PF \rightarrow_{k} S + F$$




```
directive sample 600.0 1000
directive plot S();P();F();E();ES();FP() (*;Time() *)
val a = 1.0
val d = 1.0
val k = 1.0
new es@a:chan new fp@a:chan
new killE:chan new killF: chan
new time:chan

(* S + E <-> SE -> P + E
   P + F <-> PF -> S + F
*)

let S() = ?es; ()
and E() = do !es; ES() or ?killE; ()
and ES() = do delay@d; (S()|E()) or delay@k; (P()|E())
and P() = ?fp; ()
and F() = do !fp; FP() or ?killF; ()
and FP() = do delay@d; (P()|F()) or delay@k; (S()|F())

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 Time() = ?time; ()

let schedule(n:int) =
  (Time();
    if n<1000 then ()
    else if n<3000 then E()
    else if n<6000 then !killE
    else ()
  )

run 10000 of S()
run 1000 of F()
run clock(schedule,0.1)
```



Initially 10000 S, no P, 1000 F, no E.
E growing from 0 (t=100) to 2000 (t=300) then back to 0 (t=500)
The first switch happens at t=200, the second at t=400.

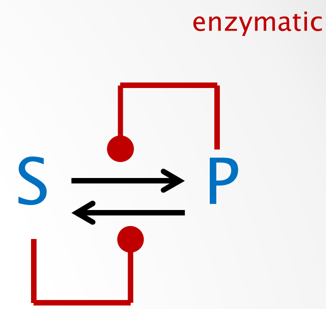
E/F ratio can be lower: GK is a ‘better’ more sensitive switch.

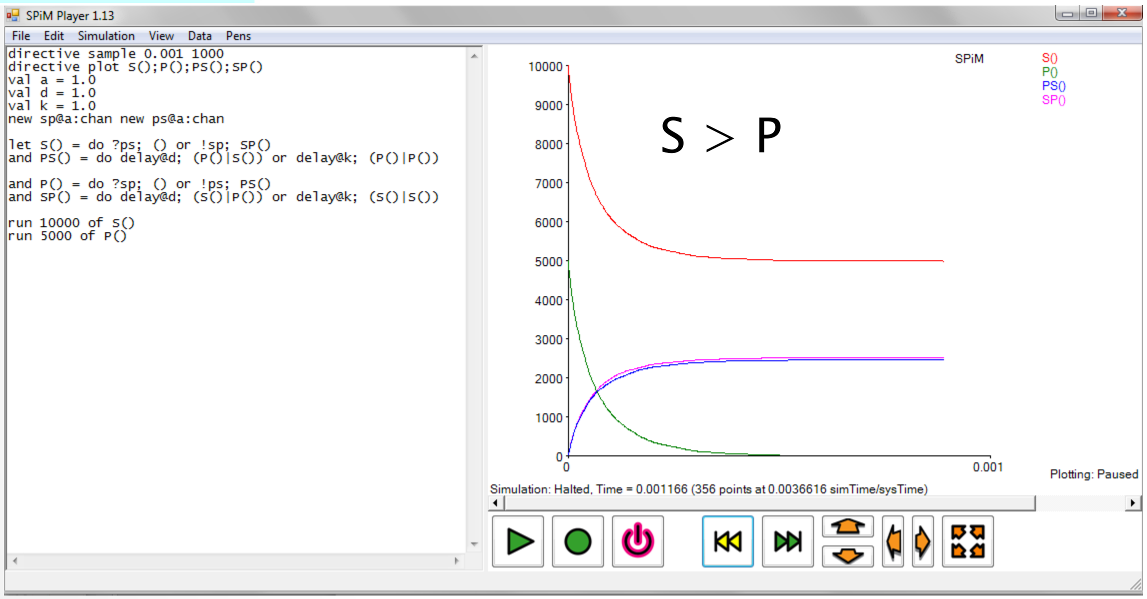
Can GK do majority switching?

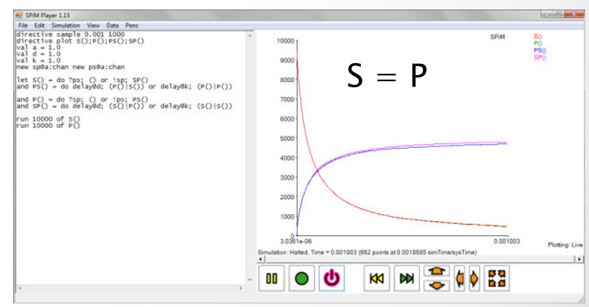
```
directive sample 0.001 1000
directive plot S();P();PS();SP()
val a = 1.0
val d = 1.0
val k = 1.0
new sp@a:chan new ps@a:chan

let S() = do ?ps; () or !sp; SP()
and PS() = do delay@d; (P()|S()) or delay@k; (P()|P())

and P() = do ?sp; () or !ps; PS()
and SP() = do delay@d; (S()|P()) or delay@k; (S()|S())

run 10000 of S()
run 10000 of P()
```

GK in "AM configuration"

$$S + P \ {}_{d}{\leftrightarrow}_{a} \ PS \rightarrow_{k} P + P$$
$$P + S \ {}_{d}{\leftrightarrow}_{a} \ SP \rightarrow_{k} S + S$$







GK in "AM configuration" does not compute a majority.

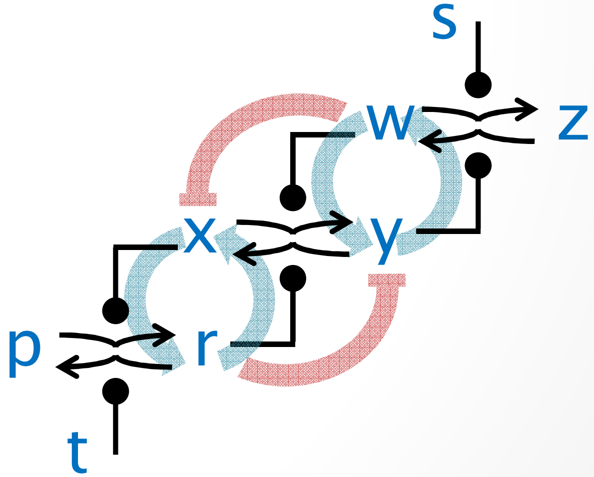
- The initial minority goes down to 0
- The initial majority goes down to $maj_{t=0} - min_{t=0}$
- When $maj_{t=0} \sim min_{t=0}$ the system cannot decide.

- Problem may be that the feedbacks put GK outside of zero-order regime.
- Hence, should check to see if GK works in the case of

$x + w \ {}_d\!\leftrightarrow_a \ xw \rightarrow_k y + w$
$y + r \ {}_d\!\leftrightarrow_a \ yr \rightarrow_k x + r$

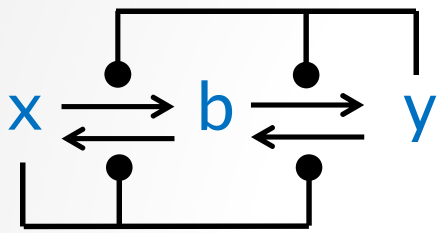
$p + x \ {}_d\!\leftrightarrow_a \ px \rightarrow_k r + x$
$r + t \ {}_d\!\leftrightarrow_a \ rt \rightarrow_k p + t$

$w + s \ {}_d\!\leftrightarrow_a \ ws \rightarrow_k z + s$
$z + y \ {}_d\!\leftrightarrow_a \ zy \rightarrow_k w + y$



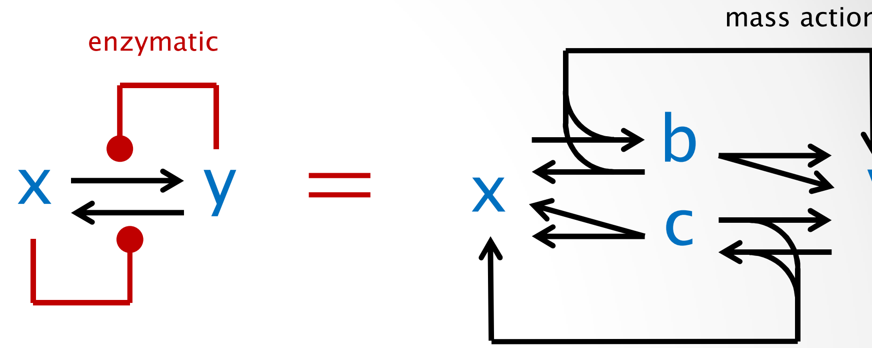
'Double phosphorylation' motif is key

AM



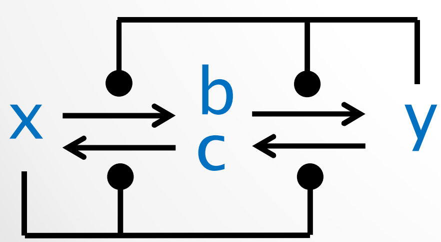
≠

autocatalytic GK



≠

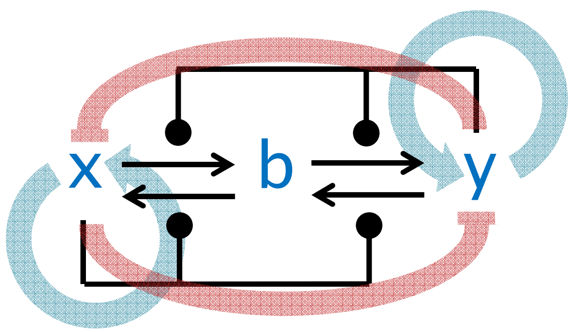
split-AM



It is not just a non-linearity of the x-y transition mechanism that matters:
it is the 'double phosphorylation' network structure of AM, with a *common* 'undecided' state.

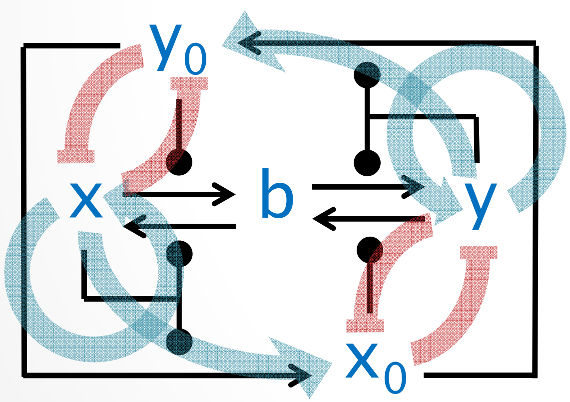
Chemical Constraints

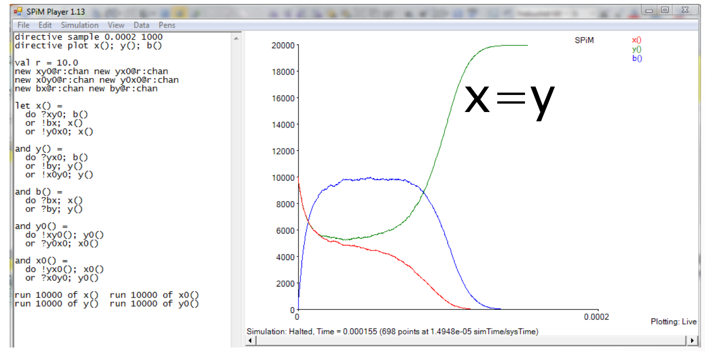
- The AM circuit is ‘chemically demanding’
  - It requires x molecules to be ‘next’ to y molecules beacause they interact directly
  - It requires both x and y to be catalysts, and in fact autocatalysts, and in fact each-other’s autocatalyst!



Network Transformations

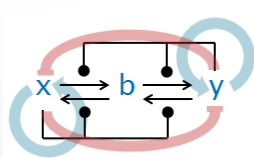
- An example of relaxing those constraints
  - This circuit works just as well as the original, but it no longer requires x to be ‘next’ to y. They no longer interact directly. Instead, they interact through an additional x_0–y_0 equilibrium.






```
directive sample 0.0002 1000
directive plot x(); y(); b()

val r = 10.0
new xy0@r:chan new yx0@r:chan
new x0y0@r:chan new y0x0@r:chan
new bx@r:chan new by@r:chan

let x() =
  do ?xy0; b()
  or !bx; x()
  or !y0x0; x()

and y() =
  do ?yx0; b()
  or !by; y()
  or !x0y0; y()

and b() =
  do ?bx; x()
  or ?by; y()

and y0() =
  do !xy0(); y0()
  or ?y0x0; x0()

and x0() =
  do !yx0(); x0()
  or ?x0y0; y0()

run 5000 of x()  run 5000 of x0()
run 5000 of y()  run 5000 of y0()
```

cf.



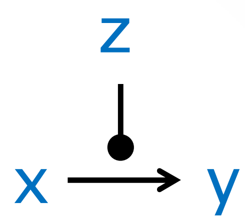
Network Transformations

- Another example of relaxing constraints
  - Build an Approximate Majority network that requires only x to be a catalyst. How?
  - Enter the Cell Cycle switches…

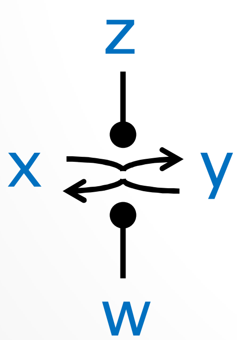
Some Notation

- Catalytic reaction

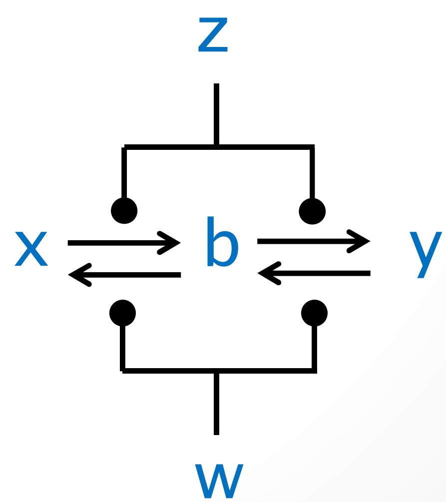
$x + z \rightarrow z + y$ $=$



- Double 'kinase-phosphatase' reactions



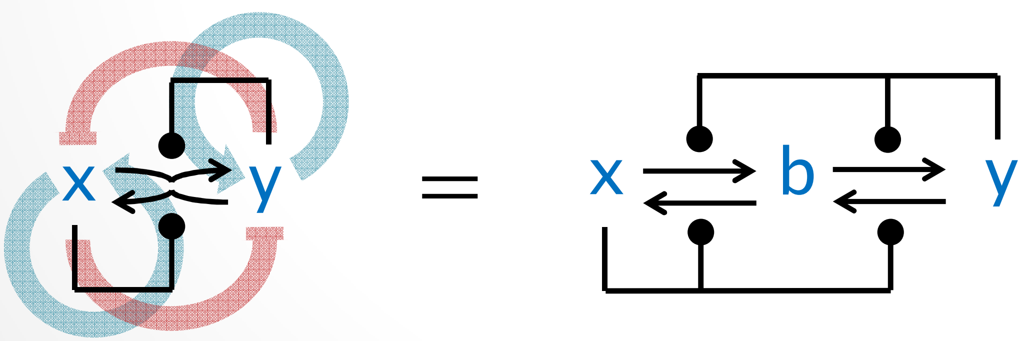
$=$

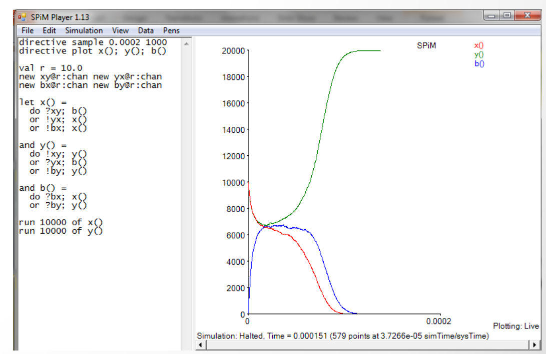


Zero-Input Switches

- 'Zero-input switch' = majority circuit: just working off the initial conditions, with no other inputs.

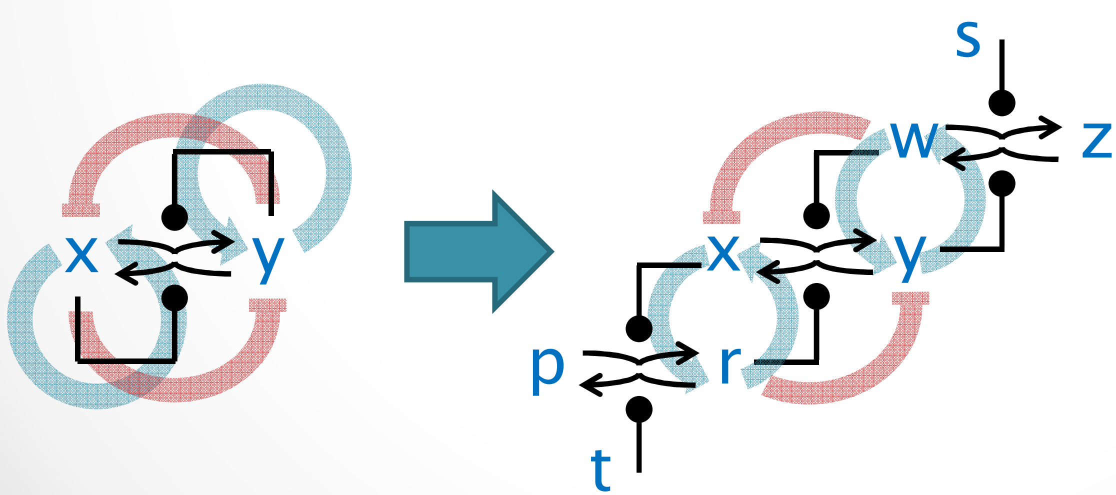
- Step 1: the original AM Network

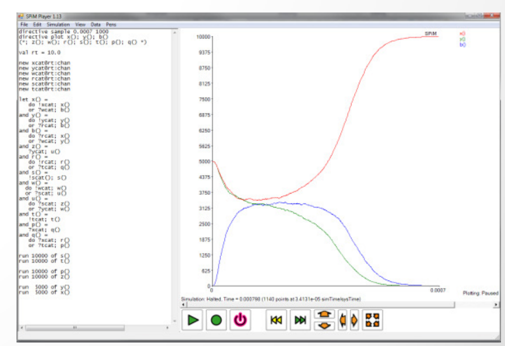




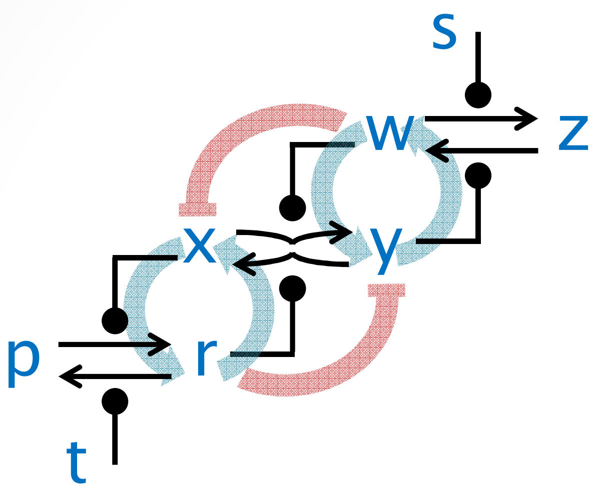
Zero-Input Switches

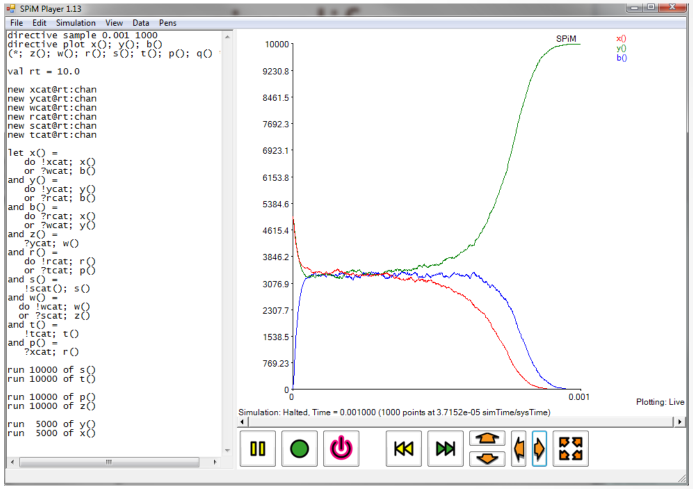
- Step 2: remove auto-catalysis
  - By introducing intermediate species w, r.
  - Here w breaks the y auto-catalysis, and r breaks the x auto-catalysis, while preserving the feedbacks.
  - w and r need to ‘relax back’ (to z and t) when they are not catalyzed: s and t provide the back pressure.





… can simplify?

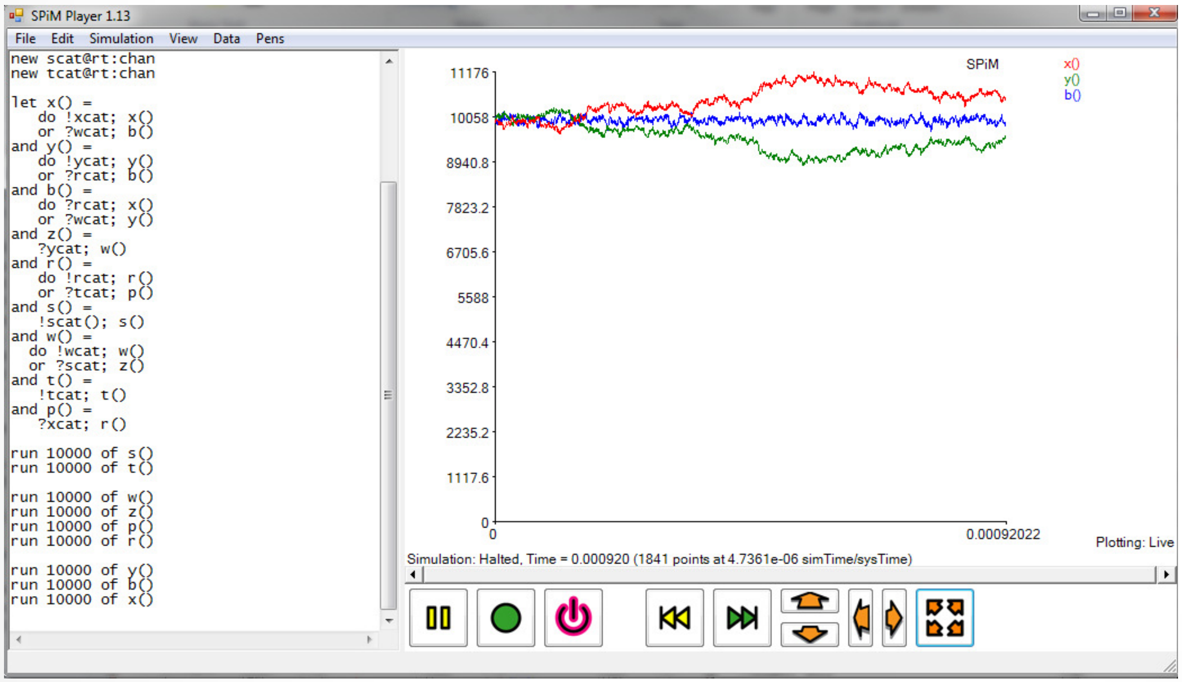




(it appears just sligthly noisier/slower)

… no, it gets stuck!

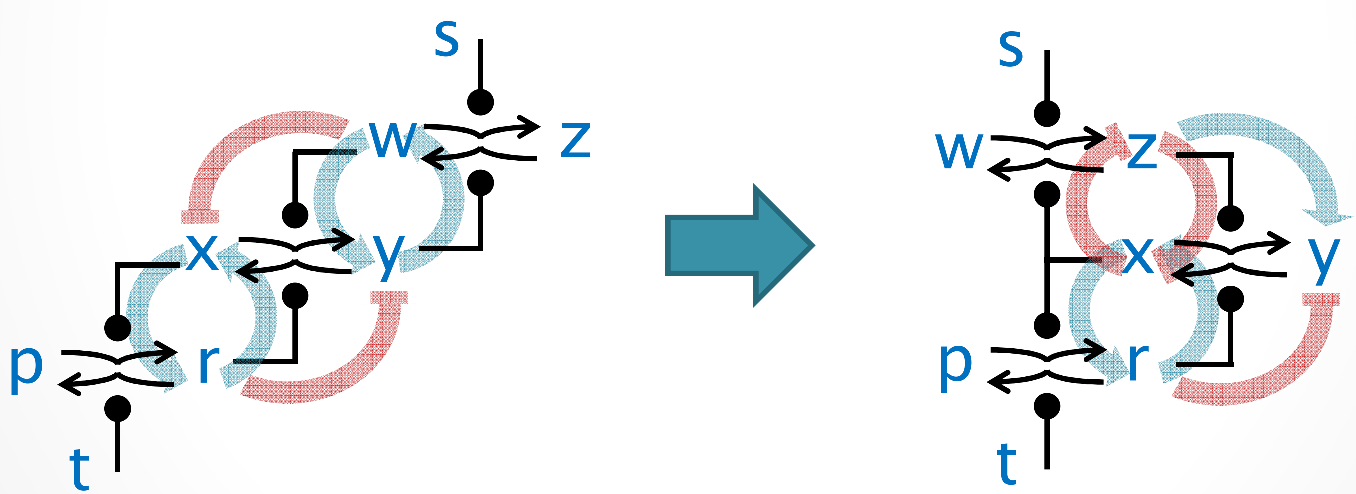
- Equal-size initial conditions




```
directive sample 0.0005
1000
directive plot x(); y(); b()
(*; z(); w(); r(); s(); t(); p();
q() *)

val rt = 10.0

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; w()
and r() =
   do !rcat; r()
   or ?tcat; p()
and s() =
   !scat(); s()
and w() =
  do !wcat; w()
  or ?scat; z()
and t() =
   !tcat; t()
and p() =
   ?xcat; r()

run 10000 of s()
run 10000 of t()

run 10000 of w()
run 10000 of z()
run 10000 of p()
run 10000 of r()

run 10000 of y()
run 10000 of b()
run 10000 of x()
```

Zero-Input Switches

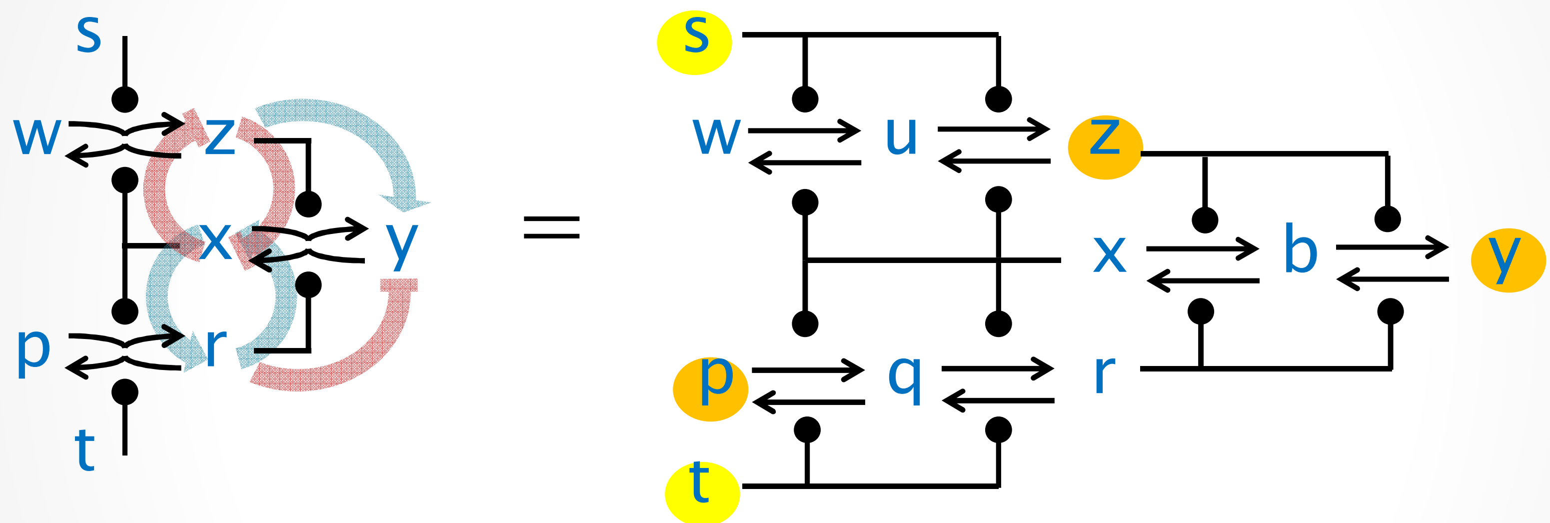
- Step 3: transform a double-positive loop on y into a double-negative loop on x.
  - Instead of y (actively) activating itself through w, we have z activating y (which is passive). To counteract, now x has to switch from inhibiting y to inhibiting z.



- So that y no longer catalyzes anything
  - All species have one active and one inactive form

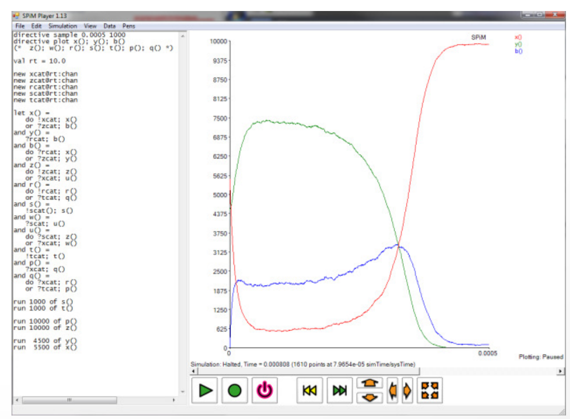
Zero-Input Switches

- Still an AM circuit



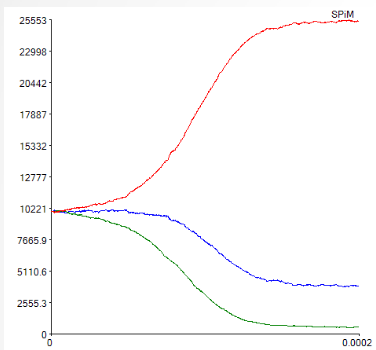
(The equal-likelihood outcome here is around 4500 y vs 5500 x, and can be adjusted by s/t ratio)

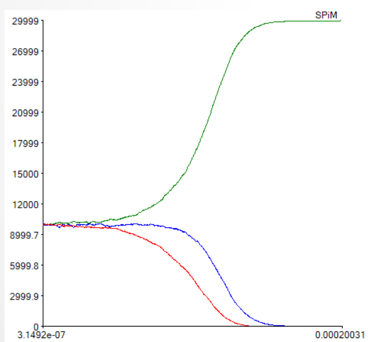
All rates are equal.

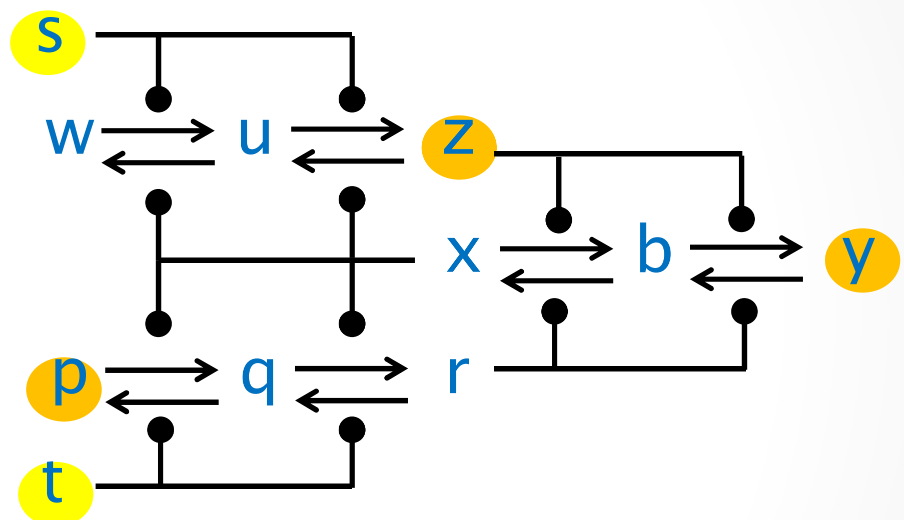


```
directive sample
0.0005 1000
directive plot x(); y();
b()
(* z(); w(); r(); s(); t();
p(); q() *)

val rt = 10.0

new xcat@rt:chan
new zcat@rt:chan
new rcat@rt:chan
new scat@rt:chan
new tcat@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 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()

run 1000 of s()
run 1000 of t()

run 10000 of p()
run 10000 of z()

run  4500 of y()
run  5500 of x()
```

Equal-size initial conditions



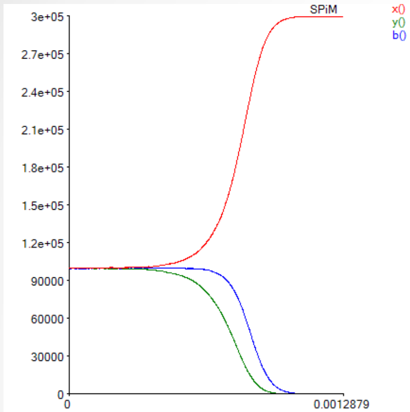


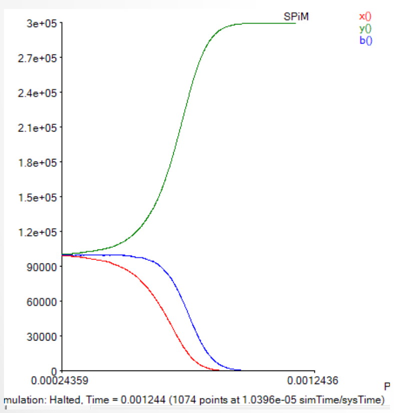



```
directive sample
0.0002 1000
directive plot x(); y();
b()
(* z(); w(); r(); s(); t();
p(); q() *)

val rt = 10.0

new xcat@rt:chan
new zcat@rt:chan
new rcat@rt:chan
new scat@rt:chan
new tcat@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 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()

run 10000 of s()
run 10000 of t()

run 10000 of w()
run 10000 of u()
run 10000 of z()
run 10000 of p()
run 10000 of q()
run 10000 of r()

run 10000 of y()
run 10000 of b()
run 10000 of x()
```

All initial species = 10000. Probability of win seems to be x=y=50%
Note that when x wins, the system does not
terminate because x has active competition from s.

AM Equal-size initial conditions

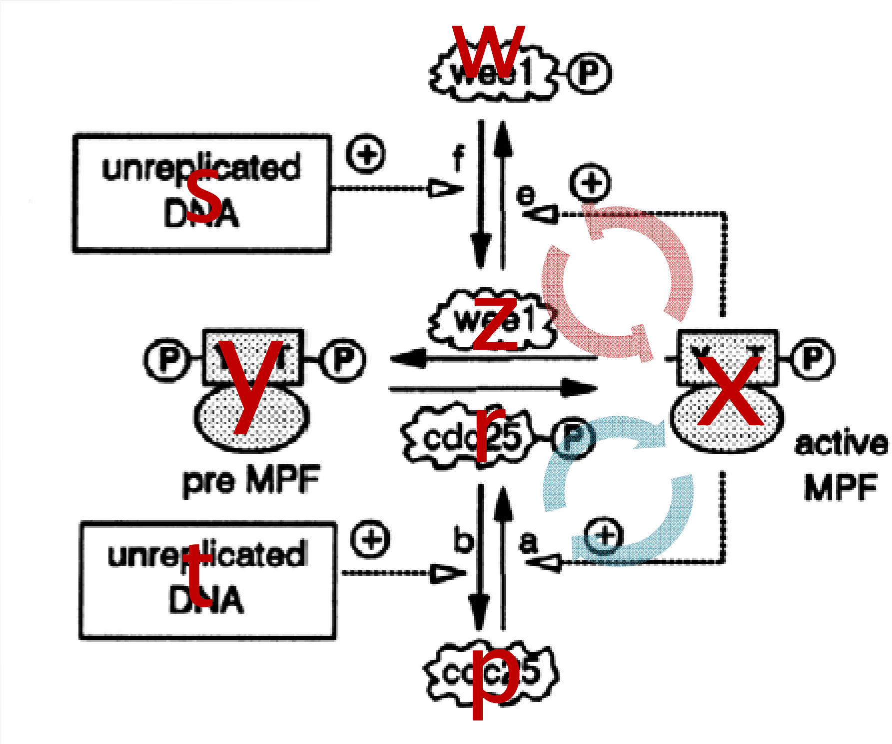


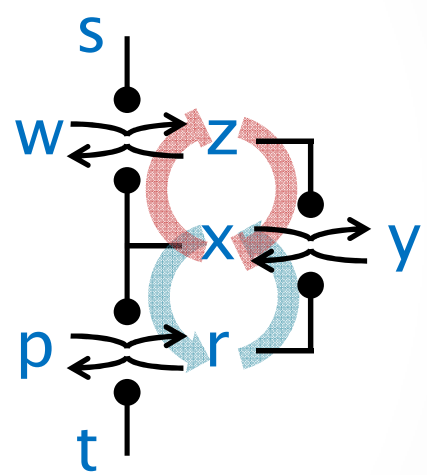



```
directive sample 0.002 1000
directive plot x(); y(); b()

val r = 0.1
new xy@r:chan new yx@r:chan

new bx@r:chan new by@r:chan

let x() =
  do ?xy; b()
  or !yx; x()
  or !bx; x()

and y() =
  do !xy; y()
  or ?yx; b()
  or !by; y()

and b() =
  do ?bx; x()
  or ?by; y()

run 100000 of x()
run 100000 of b()
run 100000 of y()
```

All initial species = 100000..

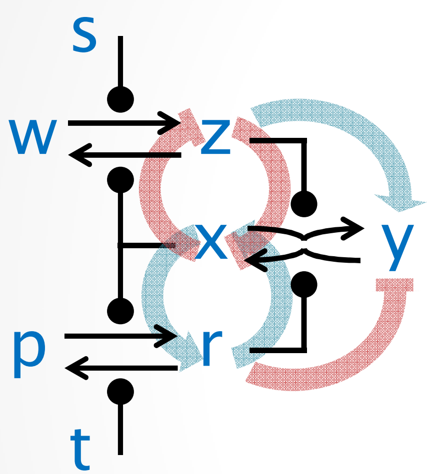
The Cell Cycle Switch

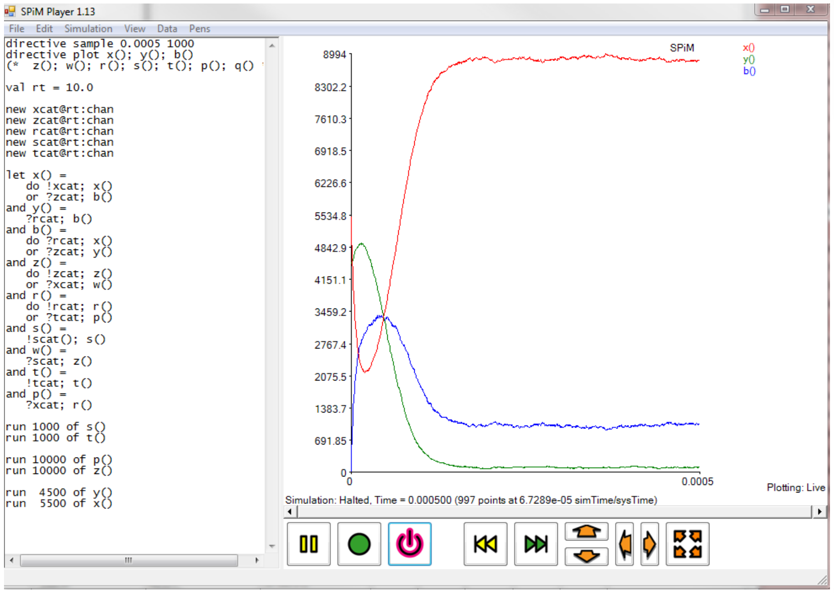




(Some of the bistable states can be enzymatic rather than AM.)

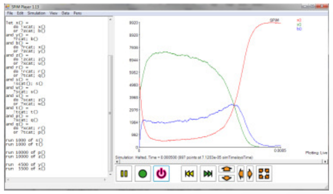
… can simplify?





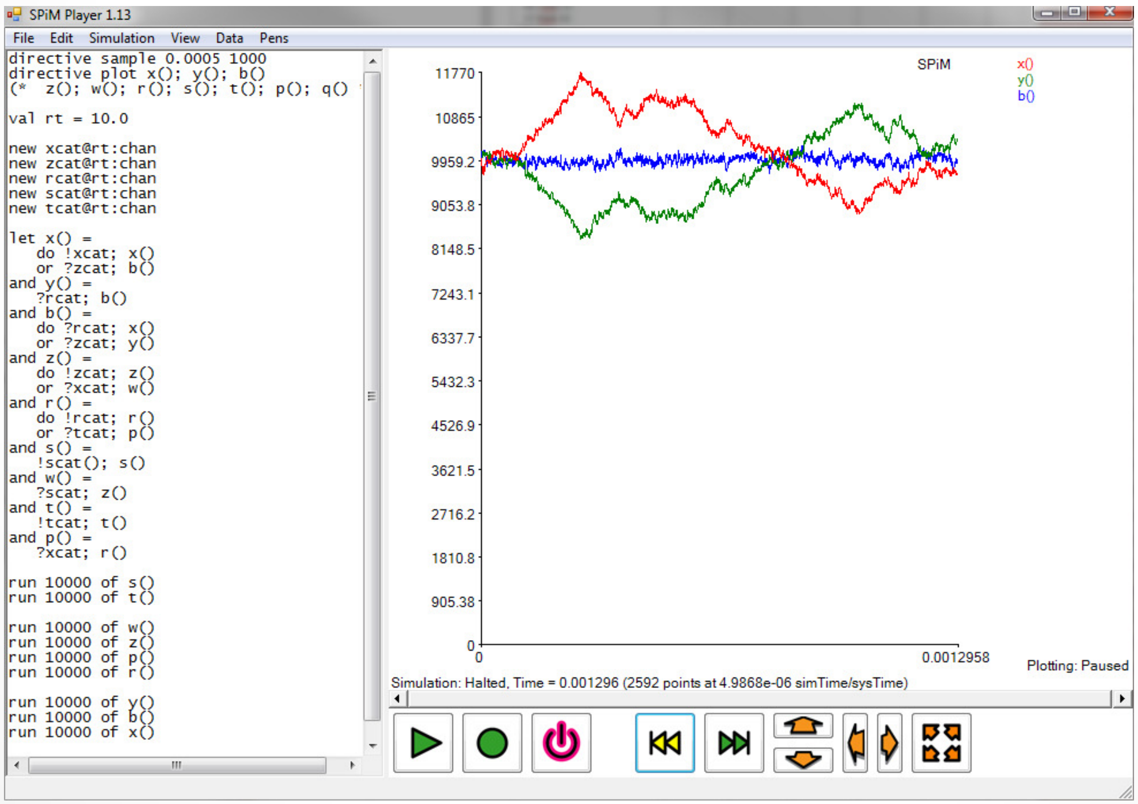

```
directive sample
0.0005 1000
directive plot x(); y();
b()
(* z(); w(); r(); s(); t();
p(); q() *)

val rt = 10.0

new xcat@rt:chan
new zcat@rt:chan
new rcat@rt:chan
new scat@rt:chan
new tcat@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; w()
and r() =
  do !rcat; r()
  or ?tcat; p()
and s() =
  !scat(); s()
and w() =
  ?scat; z()
and t() =
  !tcat; t()
and p() =
  ?xcat; r()

run 1000 of s()
run 1000 of t()

run 10000 of p()
run 10000 of z()

run  4500 of y()
run  5500 of x()
```

It works better
than the original?!?



... no, it gets stuck!

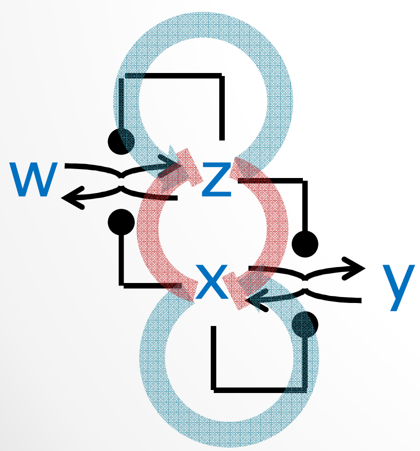
- Equal-size initial conditions

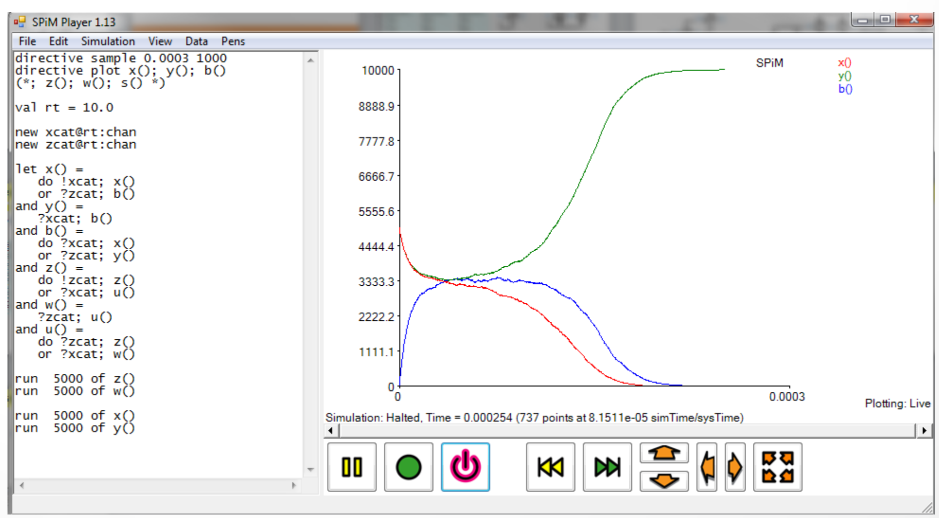



```
directive sample
0.0005 1000
directive plot x(); y();
b()
(* z(); w(); r(); s(); t();
p(); q() *)

val rt = 10.0

new xcat@rt:chan
new zcat@rt:chan
new rcat@rt:chan
new scat@rt:chan
new tcat@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; w()
and r() =
  do !rcat; r()
  or ?tcat; p()
and s() =
  !scat(); s()
and w() =
  ?scat; z()
and t() =
  !tcat; t()
and p() =
  ?xcat; r()

run 10000 of s()
run 10000 of t()

run 10000 of w()
run 10000 of z()
run 10000 of p()
run 10000 of r()

run 10000 of y()
run 10000 of b()
run 10000 of x()
```

More Zero-Input Switches

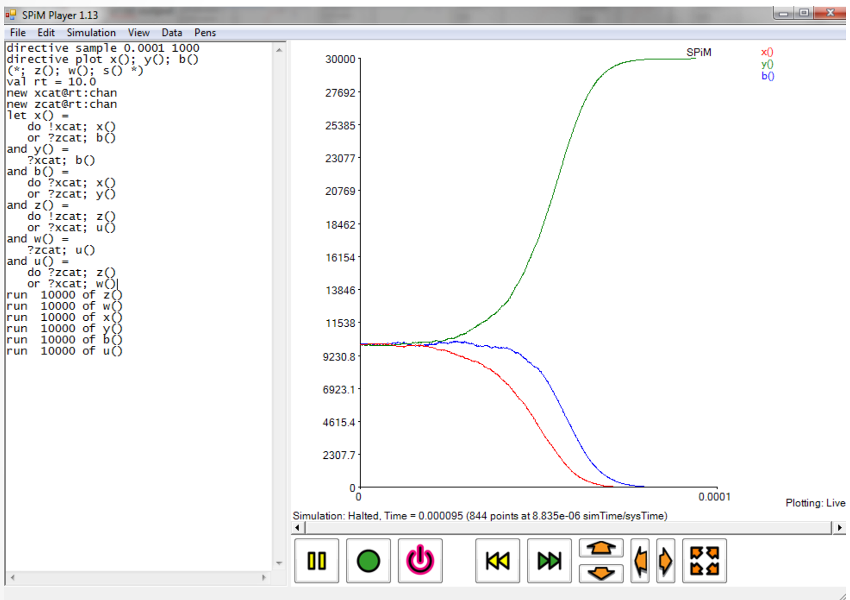
- Other designs
  - A version with no external bias (s,t) where y is still non-catalytic and x and z are self-catalytic.
  - Both x and z have an ‘inactive’ form, y and w, although the both are double catalysts.






```
directive sample 0.0003 1000
directive plot x(); y(); b()
(*; z(); w(); s() *)

val rt = 10.0

new xcat@rt:chan
new zcat@rt:chan

let x() =
    do !xcat; x()
    or ?zcat; b()
and y() =
    ?xcat; b()
and b() =
    do ?xcat; x()
    or ?zcat; y()
and z() =
    do !zcat; z()
    or ?xcat; u()
and w() =
    ?zcat; u()
and u() =
    do ?zcat; z()
    or ?xcat; w()

run  5000 of z()
run  5000 of w()

run  5000 of x()
run  5000 of y()
```

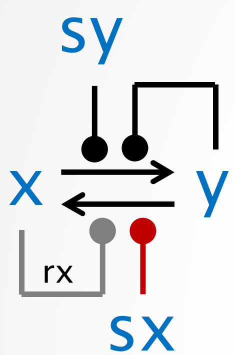
- Equal-size intial conditions

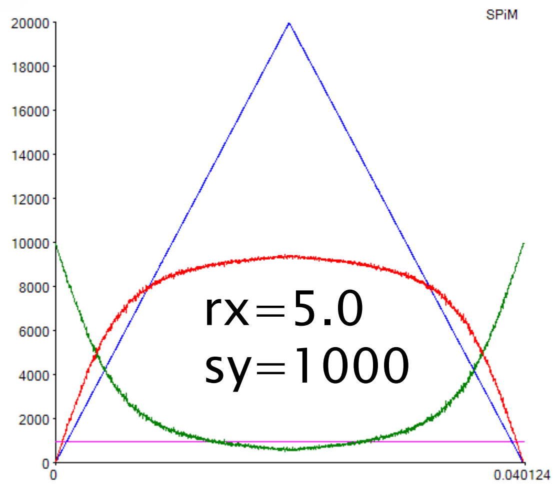



```
directive sample 0.0001 1000
directive plot x(); y(); b()
(*; z(); w(); s() *)
val rt = 10.0
new xcat@rt:chan
new zcat@rt:chan
let x() =
   do !xcat; x()
   or ?zcat; b()
and y() =
   ?xcat; b()
and b() =
   do ?xcat; x()
   or ?zcat; y()
and z() =
   do !zcat; z()
   or ?xcat; u()
and w() =
   ?zcat; u()
and u() =
   do ?zcat; z()
   or ?xcat; w()
run  10000 of z()
run  10000 of w()
run  10000 of x()
run  10000 of y()
run  10000 of b()
run  10000 of u()
```

One-Input Switches

- Ultrasensitivity (none) and hysteresis (none) in trivial majority

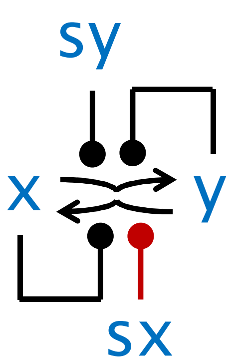


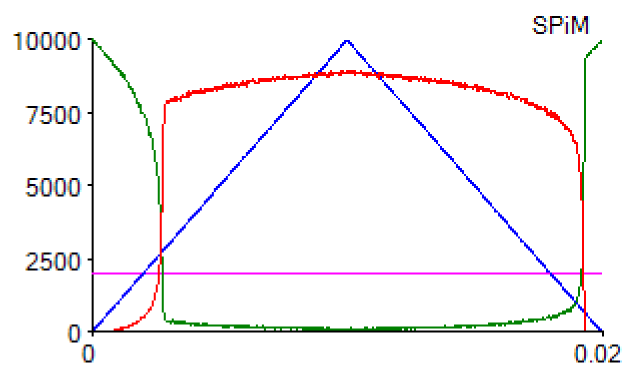


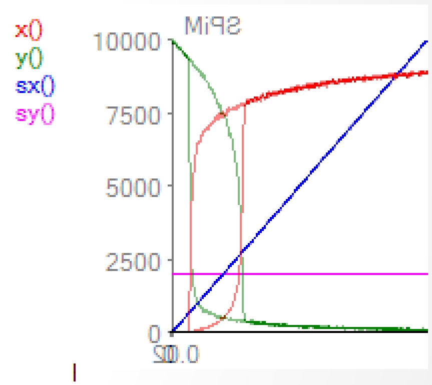

```
directive sample 0.02 1000
directive plot x(); y(); sx(); sy() (* b();  *)

val rt = 10.0
val rx = 5.0
new xcat@rx: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 10000 of y()
run 1000 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 < 20000 then sx()
  else if n < 40000 then !sxkill;()
  else ()

run clock(schedule,0.000001)
```

One-Input Switches

- Hysteresis in unbiased AM-like switches.
  - All rates are equal; constant amount of sy is sufficient for switch-back.



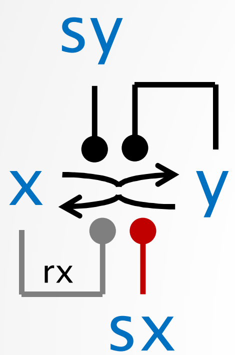


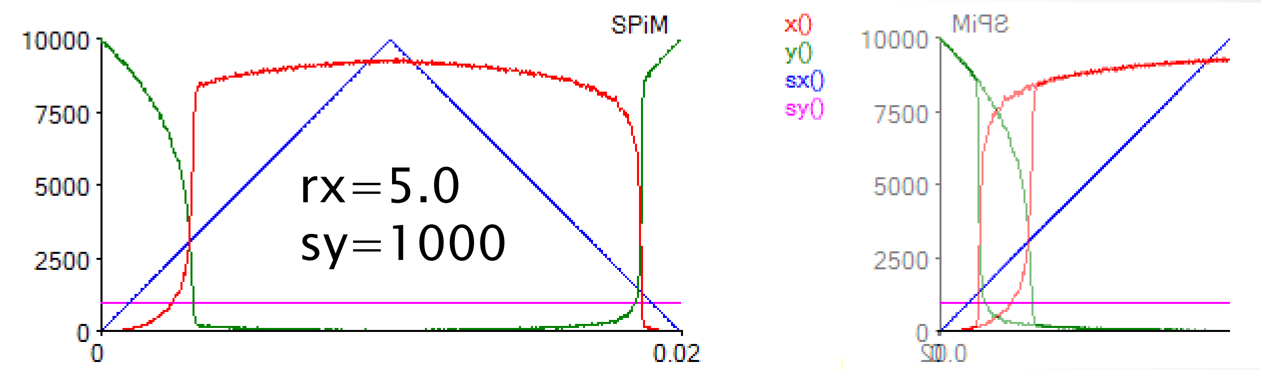



```
directive sample 0.02 1000
directive plot x(); y(); sx(); sy() (* b(); *)

val rt = 10.0
val rx = 10.0
new xcat@rx: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 10000 of y()
run 2000 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 < 10000 then sx()
  else if n < 20000 then !sxkill;()
  else ()

run clock(schedule,0.000001)
```

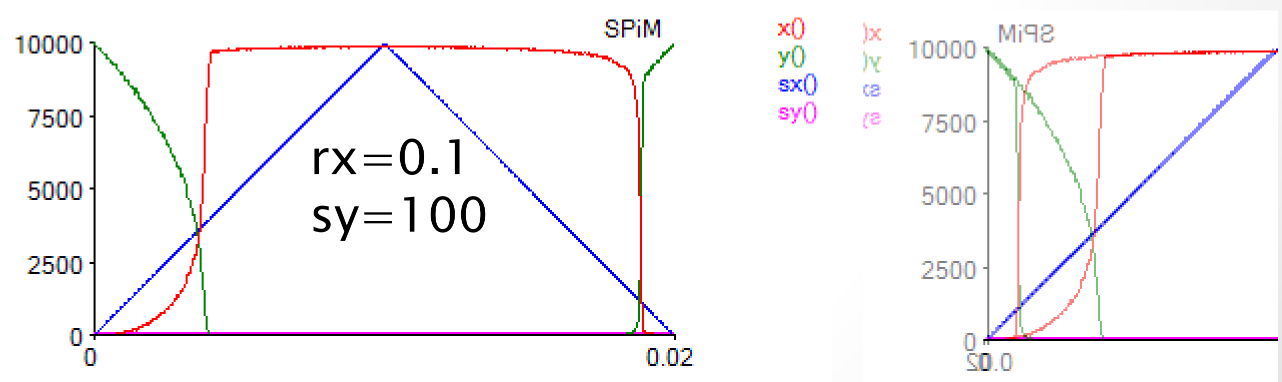
One-Input Switches

- Hysteresis in biased AM-like switches



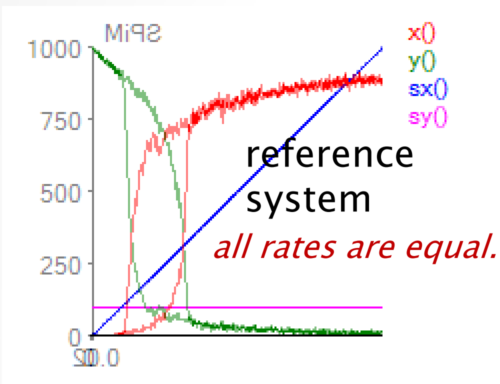


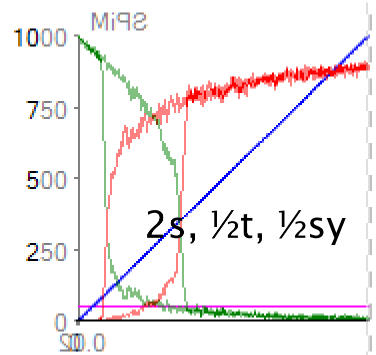

```
directive sample 0.02 1000
directive plot x(); y(); sx(); sy() (* b();  *)

val rt = 10.0
val rx = 5.0
new xcat@rx: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 10000 of y()
run 1000 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 < 10000 then sx()
  else if n < 20000 then !sxkill;()
  else ()

run clock(schedule,0.000001)
```

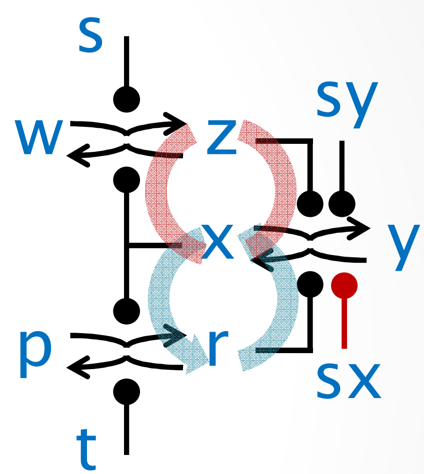


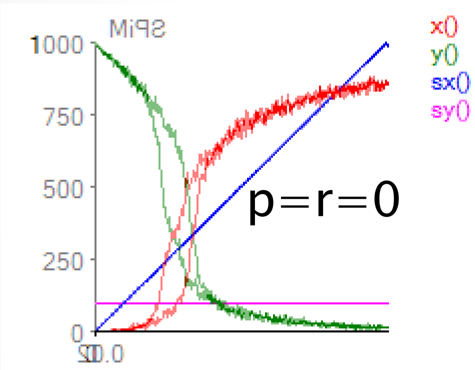
One-Input Switches

- Hysteresis vs. feedbacks in cell cycle switch

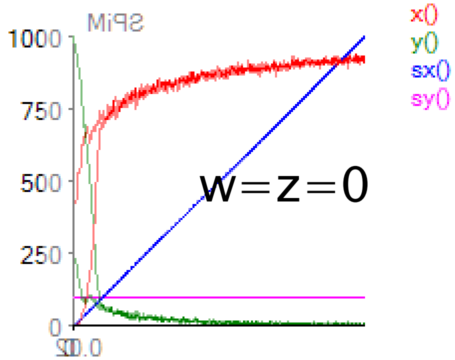








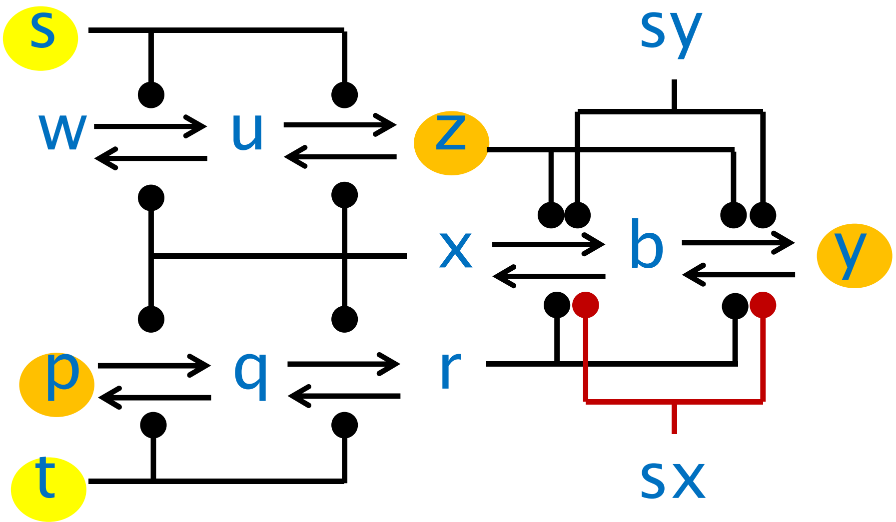
Without pos-pos feedback



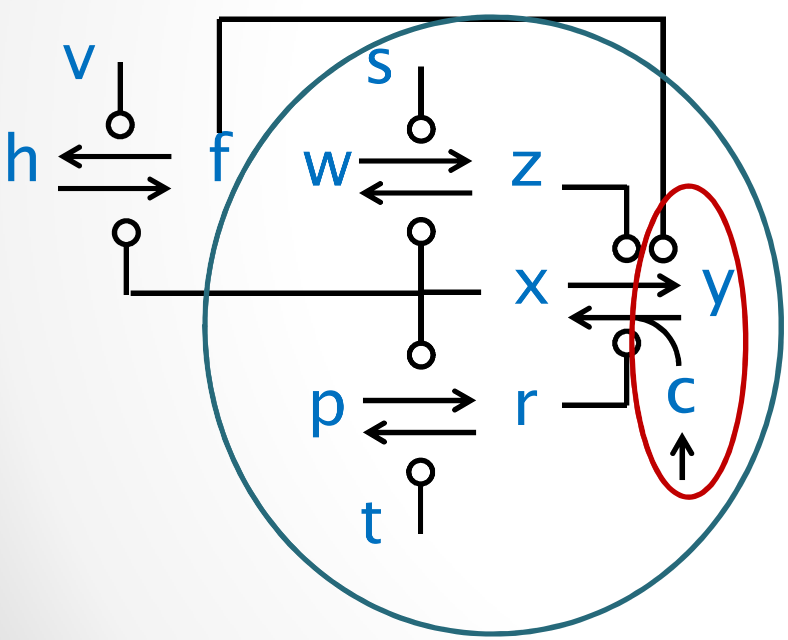
Without neg-neg feedback

initial conditions:
1000 of y
1000 of z
1000 of p
1000 of t
200 of s
100 of sy

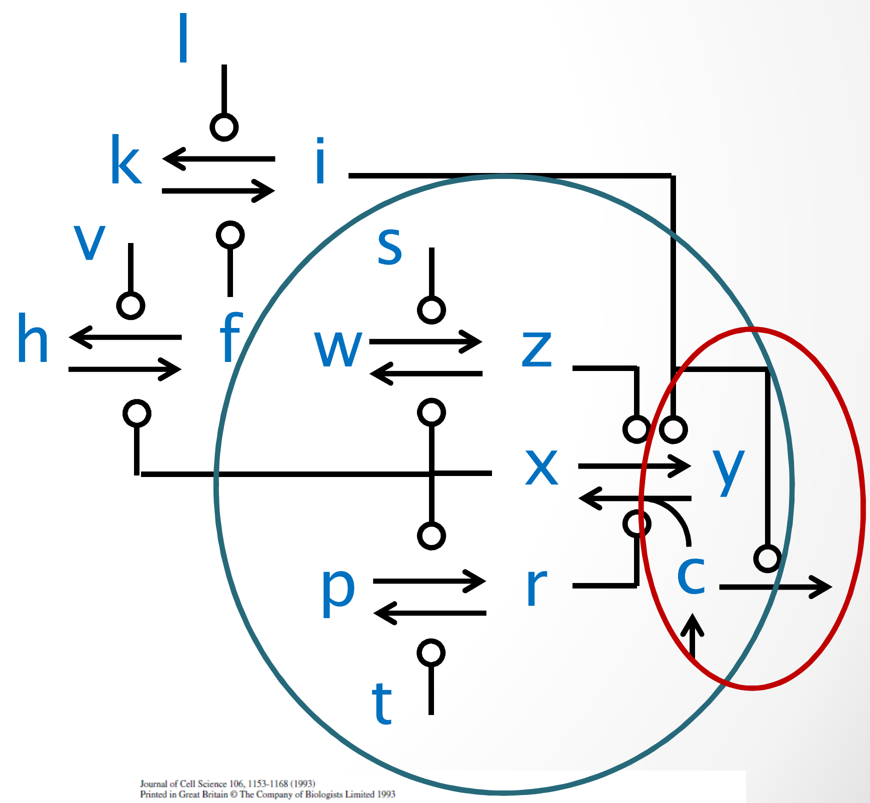
varying sx 0 to 1000 to 0


s
w
u
z
sy
x
b
y
p
q
r
t
sx

Ferrell oscillator



Novak-Tyson oscillator



Journal of Cell Science 106, 1153-1168 (1993)

Numerical analysis of a comprehensive model of M-phase control in *Xenopus* oocyte extracts and intact embryos

Bela Novak* and John J. Tyson†

Department of Biology, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24060-0406, USA

Two-input Switches

- I had rediscovered (but not analyzed so well) the same system, while looking for a memory circuit.
- The point here was not computing majority, but switching easily and quickly and stably.

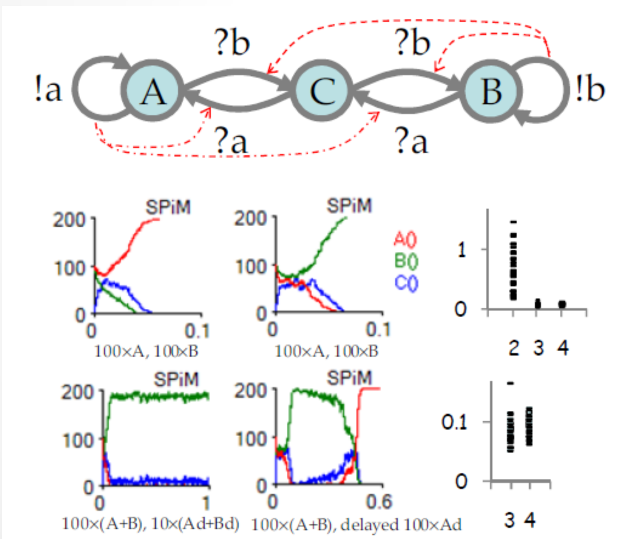


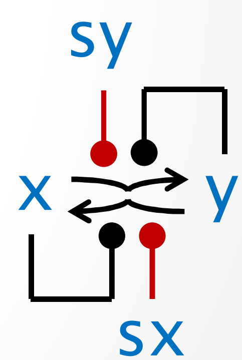
Figure 34 Memory Elements

A + B -> B + C
B + A -> A + C
C + A -> A + A
C + B -> B + B

Artificial Biochemistry. Luca Cardelli

©2009 In A.Condon, D.Harel, J.N.Kok, A.Salomaa, E.Winfree (Eds.) Algorithmic Bioprocesses. Springer 2009. DOI: 10.1007/978-3-540-88869-7_22. ISBN: 978-3-540-88868-0. Auxiliary Materials: Simulations, Figures.

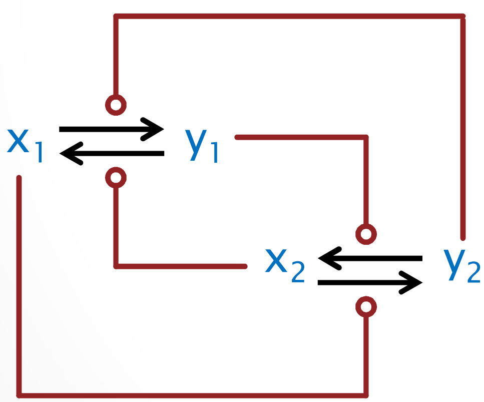
In Figure 34 we show a modified version of the groupies, obtained by adding an intermediate state shared by the two state transitions. This automaton has very good memory properties. The top-left and top-center plots show that it is in fact spontaneously bistable. The bottom-left plot shows that it is stable in presence of sustained 10% fluctuations produced by doping automata. The bottom-center plot shows that, although resistant to perturbations, it can be switched from one state to another by a signal of the same magnitude as the stability level: the switching time is comparable to the stabilization time. In addition, this circuit reaches stability 10 times faster than the original groupies: the top-right plot shows the convergence times of 30 runs each of the original groupies with 2 states, the current automaton with 3 states, and a similar automaton (not shown) with 4 states that has two middle states in series. The bottom-right plot is a detailed view of the same data, showing that the automaton with 4 states is not significantly faster than the one with 3 states. Therefore, we have a stable and fast memory element.

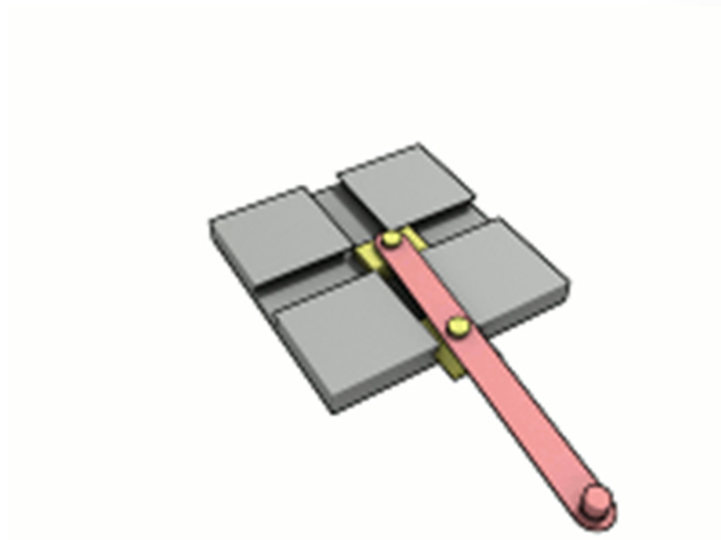


Oscillators

The Trammel of Archimedes

- A device to draw ellipses
  - Two interconnected switches.
  - When one switch is on (off) it flips the other switch on (off). When the other switch is on (off) it flips the first switch off (on).

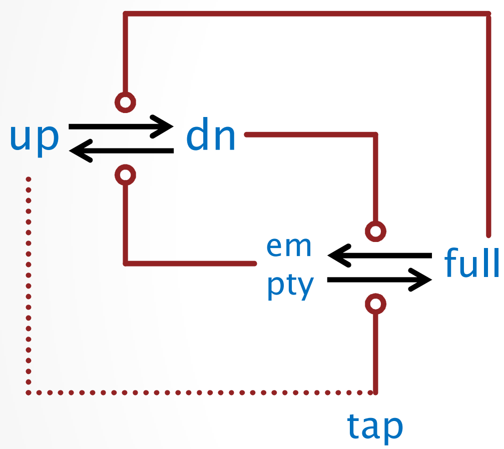




en.wikipedia.org/wiki/Trammel_of_Archimedes

The Shishi Odoshi

- A Japanese scarecrow (scare-deer)
  - Used by Bela Novak to illustrate the cell cycle switch.



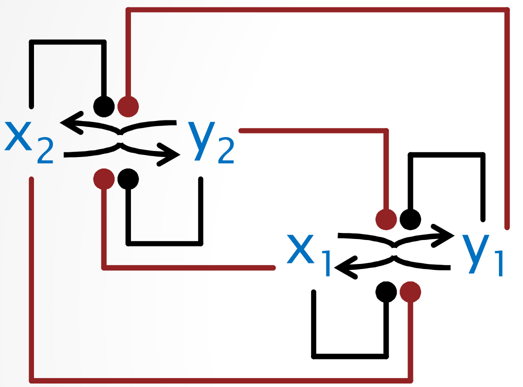
empty + tap → tap + full
up + full → full + dn
full + dn → dn + empty
dn + empty → empty + up

http://www.youtube.com/watch?v=VbvecTIftcE&NR=1&feature=fvwp

To make it into a full trammel (dotted line), we could make the up position mechanically open the tap (i.e. take up = tap)

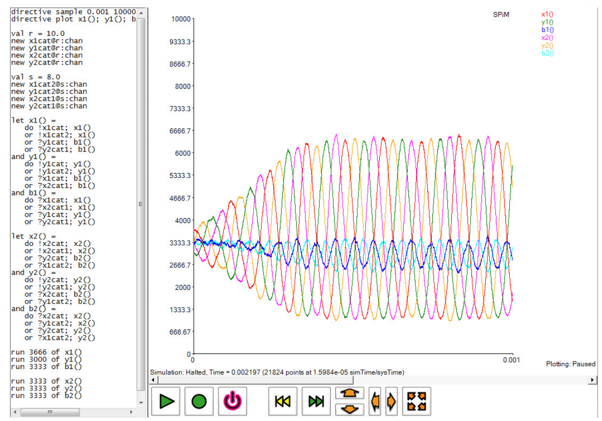
The 2AM Limit-Cycle Oscillator

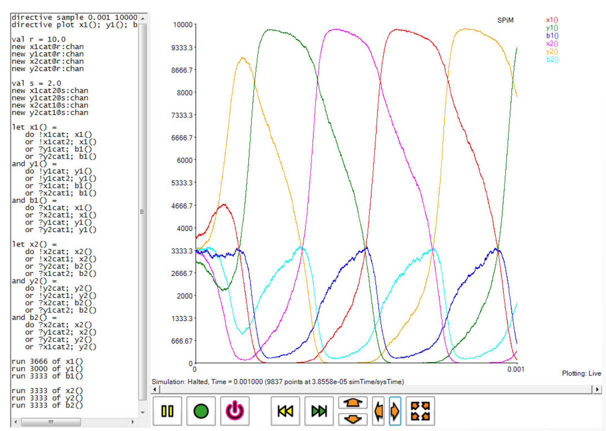
- Two AM switches in a Trammel pattern



The red reactions need to be slower (even slightly) than the black reactions, but otherwise the oscillation is robust. Oscillation stops at 10 vs. 10 and 1 vs. 10. Here the rates are 8(red) vs 10(black) top, and 2 vs 10, bottom.

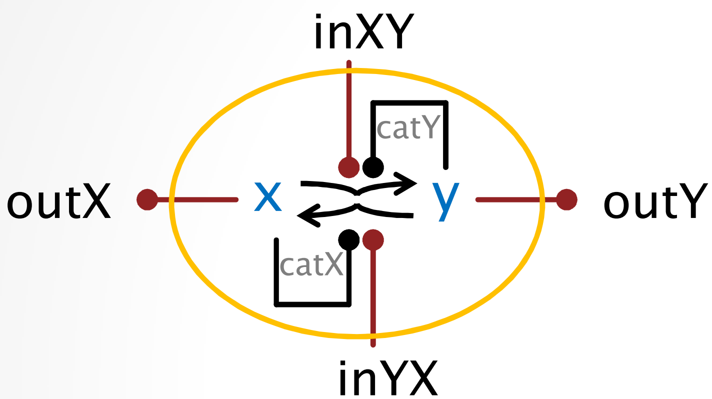
(Simple limit-cycle oscillators in the literature have very critical rate ranges.)

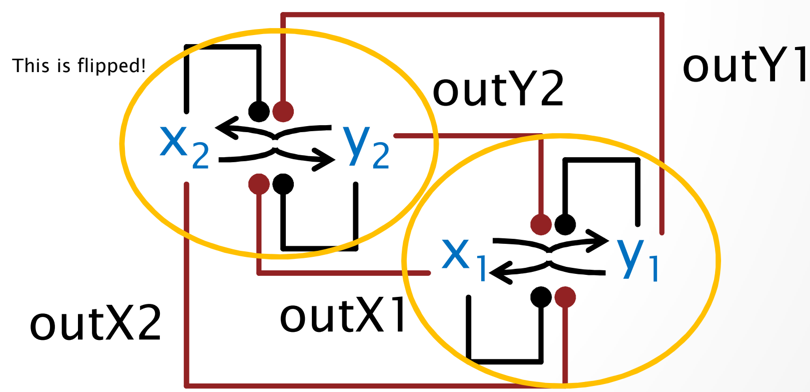




```
directive sample 0.001 10000
directive plot x1(); y1(); b1(); x2();
y2(); b2()

val r = 10.0
new x1cat@r:chan
new y1cat@r:chan
new x2cat@r:chan
new y2cat@r:chan

val s = 8.0
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 3666 of x1()
run 3000 of y1()
run 3333 of b1()

run 3333 of x2()
run 3333 of y2()
run 3333 of b2()
```

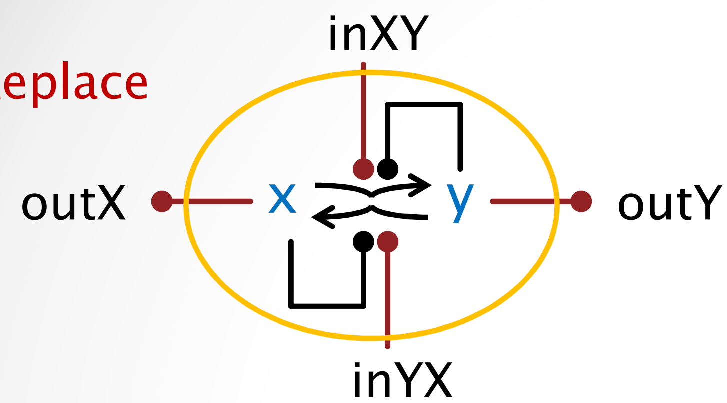
The Switch Module



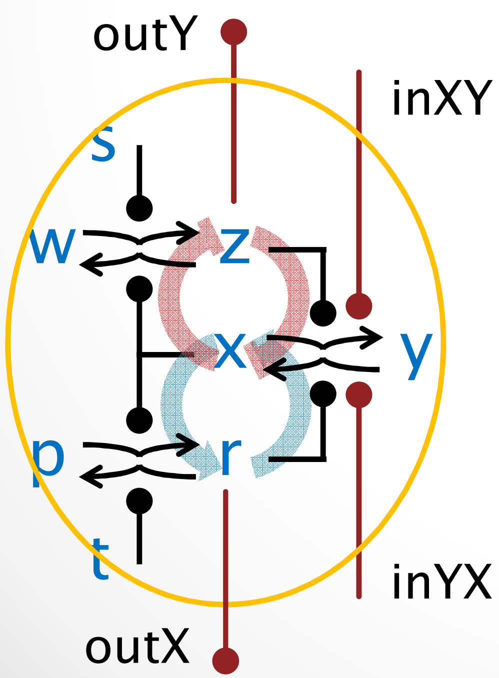


Replacing Switch Modules

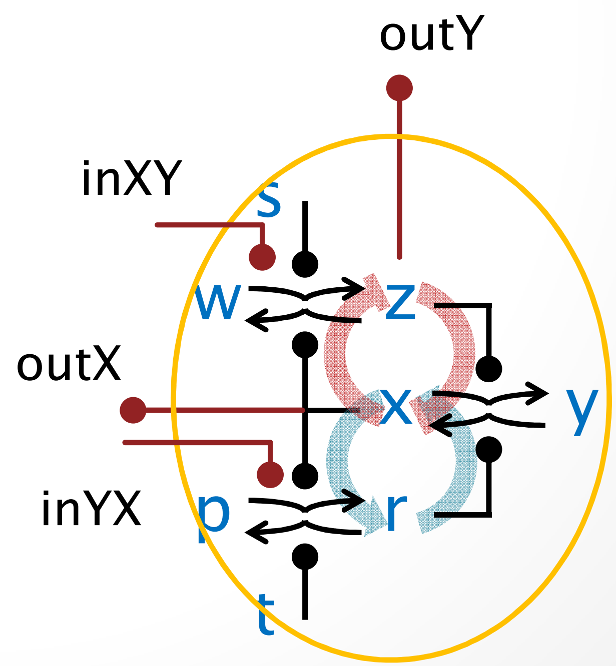
Replace



With

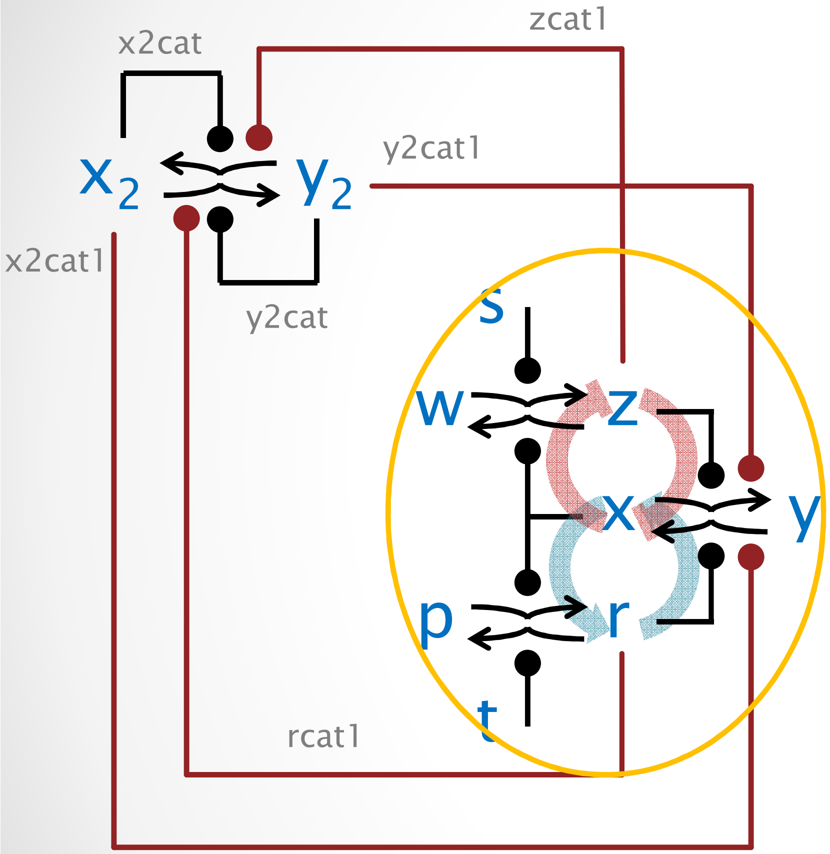


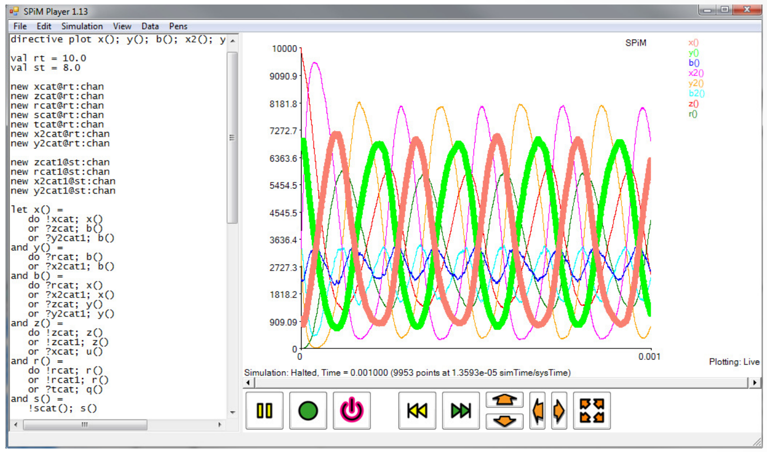
Or

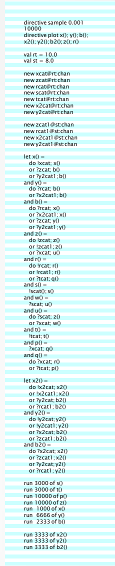


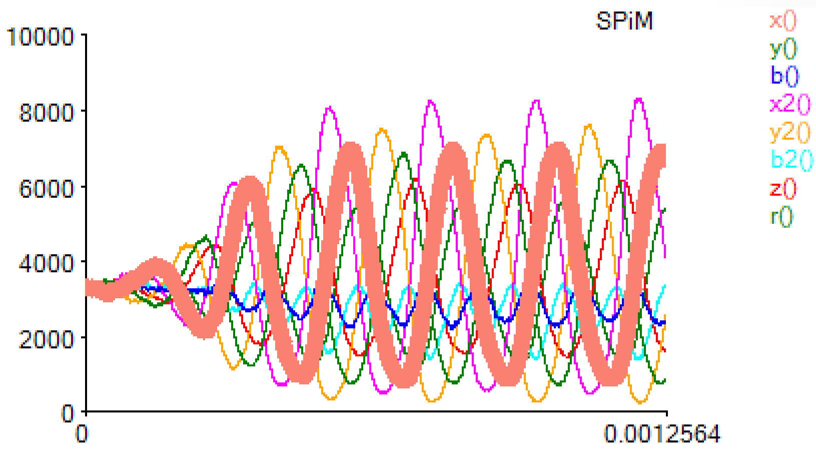
Etc..

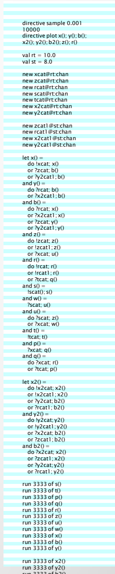
Modified Oscillator 1



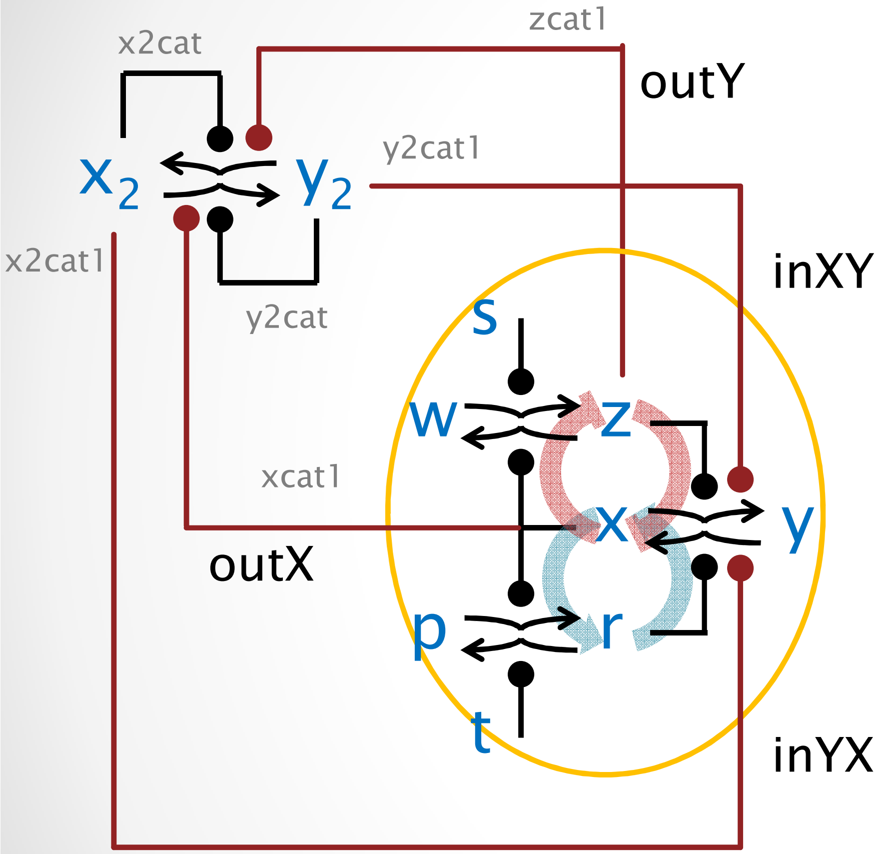


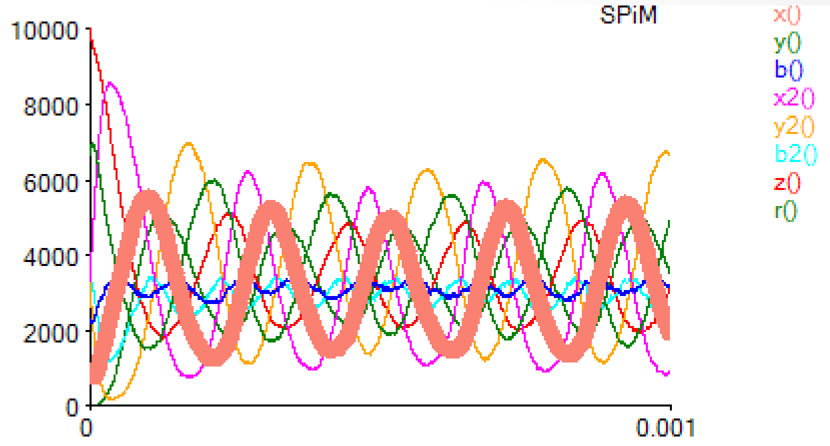


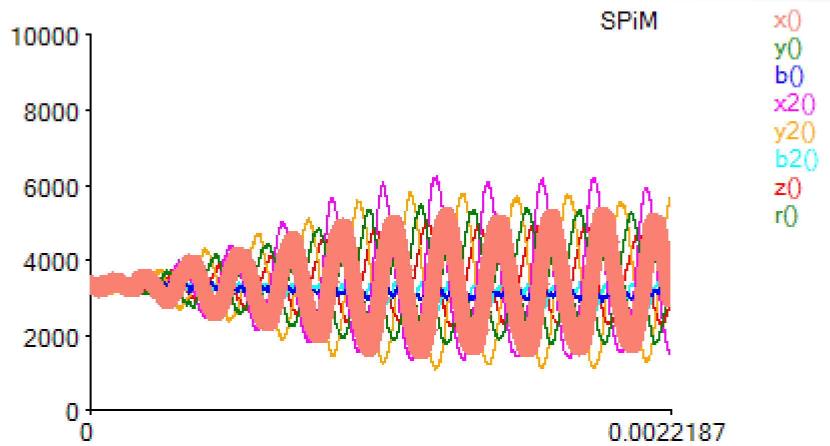




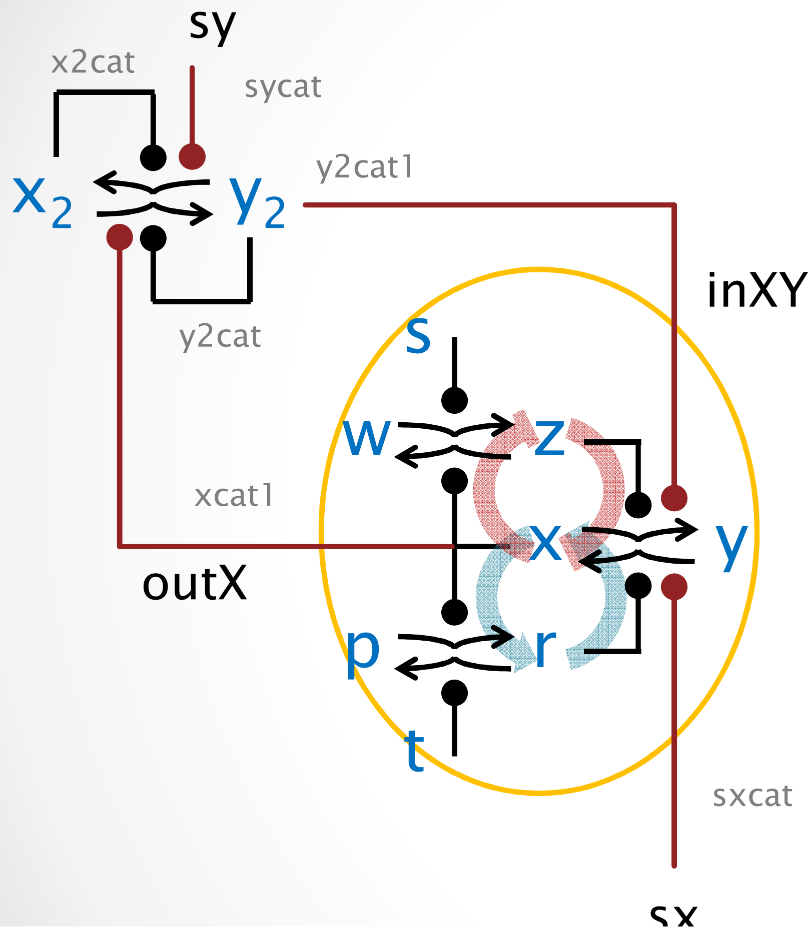
Modified Oscillator 2

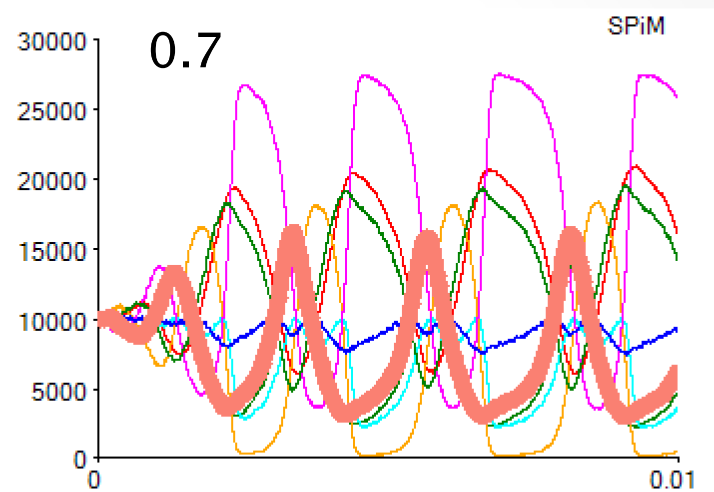


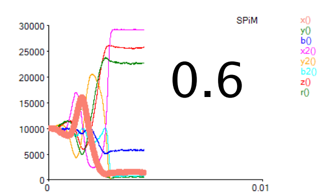


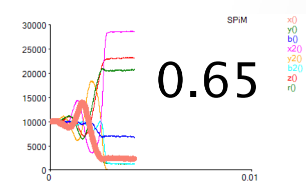


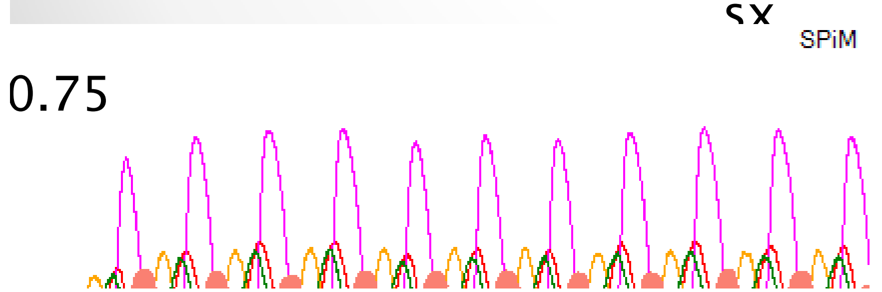
Modified Oscillator 3

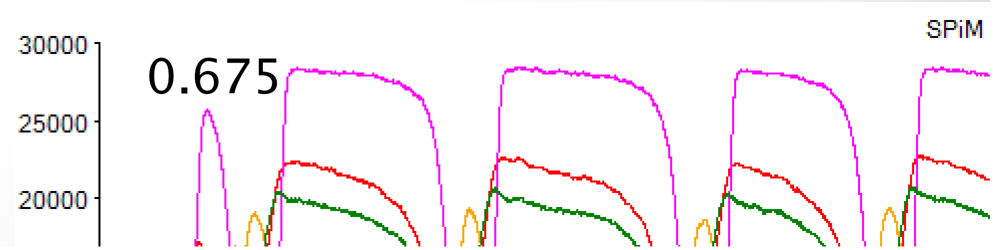




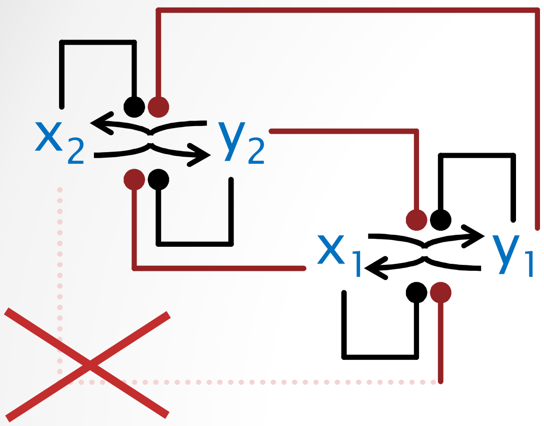




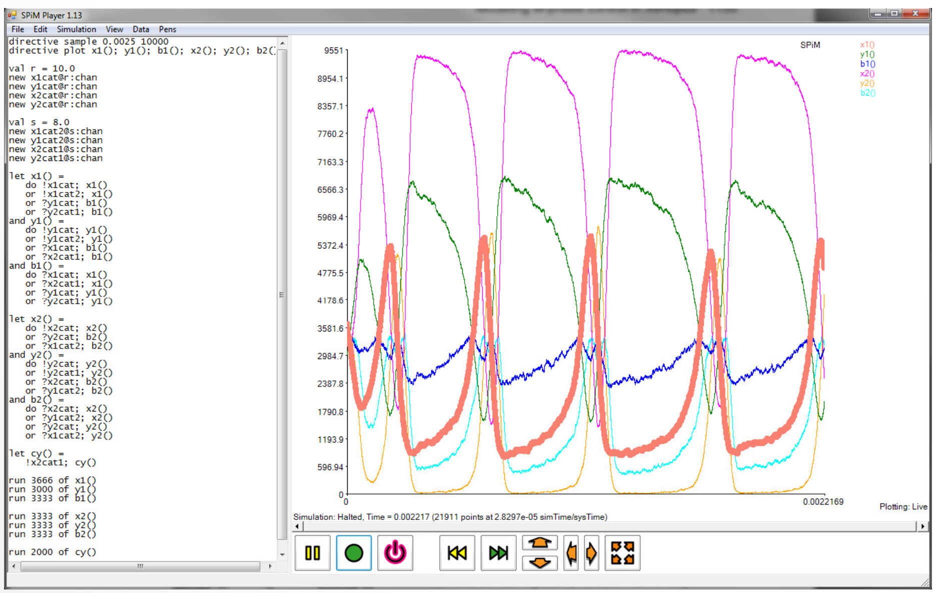




Constant-Influx Oscillator

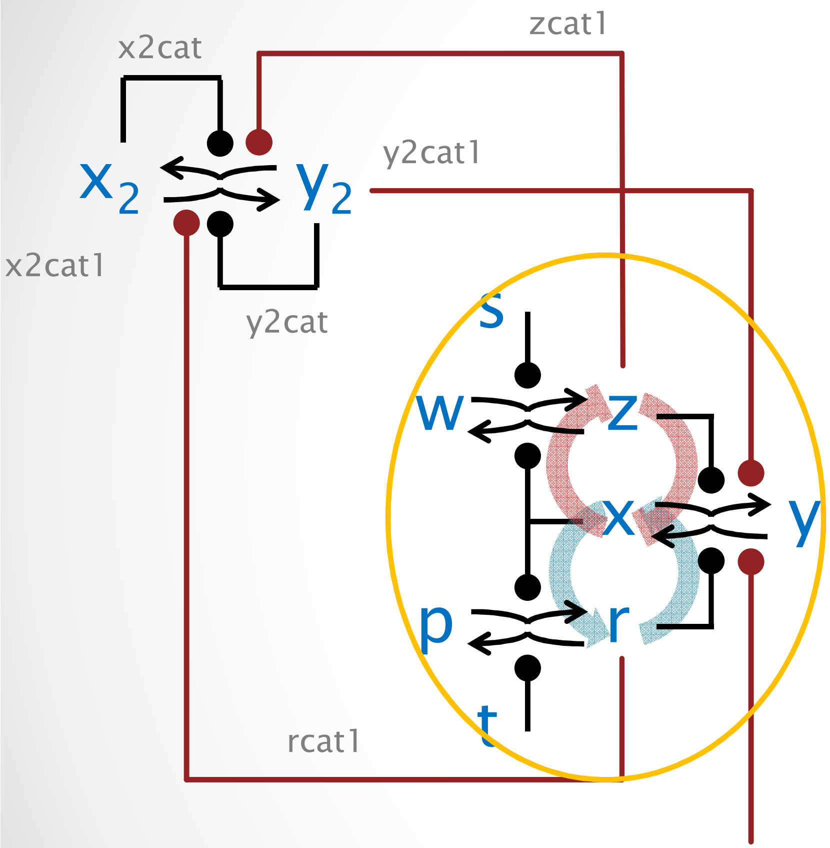


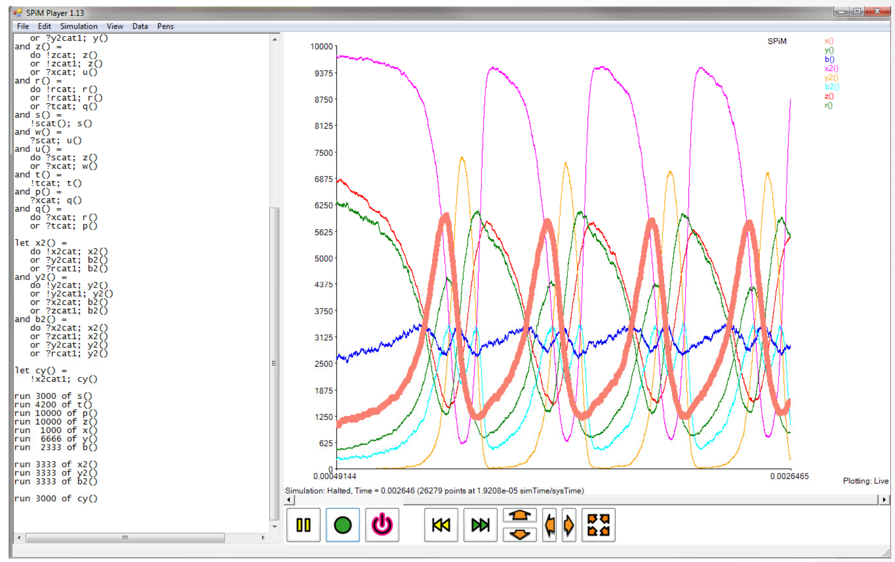
As in the Shishi Odoshi
(and the cell cycle)



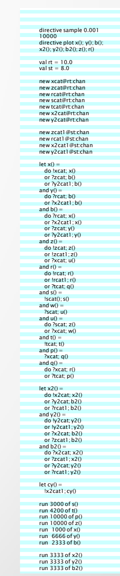

```
directive sample 0.001 10000
directive plot x1(); y1(); b1(); x2();
y2(); b2()

val r = 10.0
new x1cat@r:chan
new y1cat@r:chan
new x2cat@r:chan
new y2cat@r:chan

val s = 8.0
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 ?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()

let cy() =
  !x2cat1; cy()

run 3666 of x1()
run 3000 of y1()
run 3333 of b1()

run 3333 of x2()
run 3333 of y2()
run 3333 of b2()

run 2000 of cy()
```

Constant influx



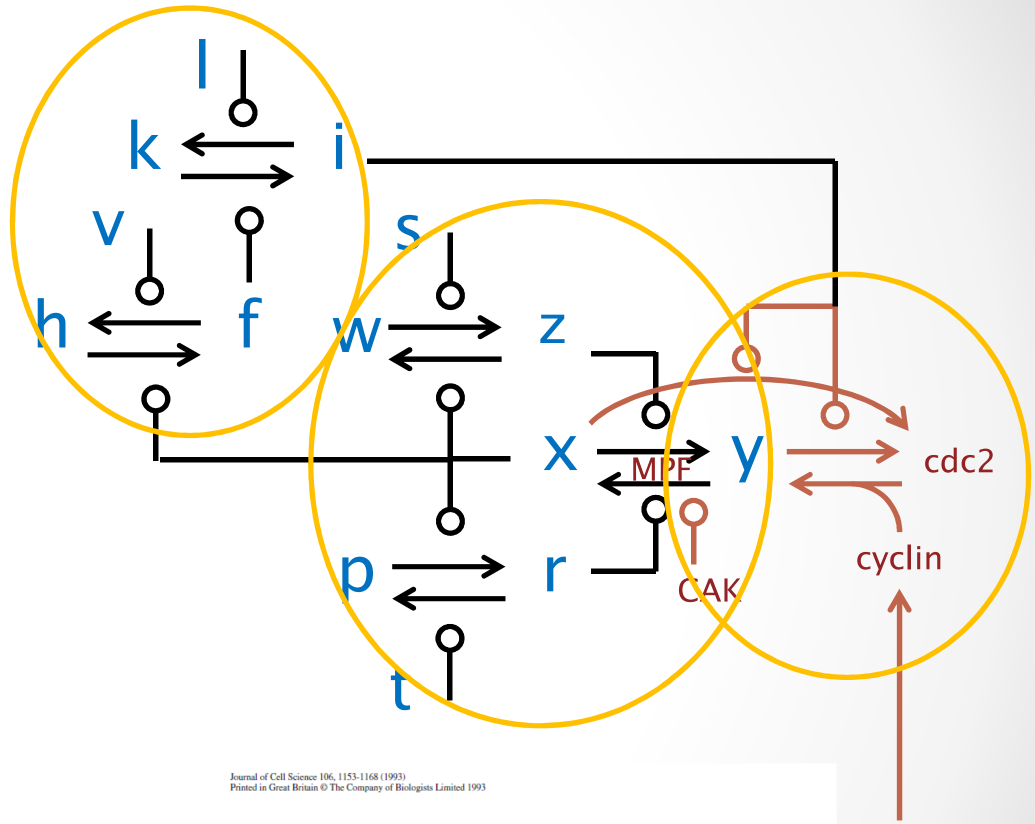


Still working fine with the replaced switch.



The Novak-Tyson Oscillator

- First switch
  - Is the 'transformed' AM switch in one-input configuration (driven by constant influx of cyclin).
- Second switch
  - Is a simple two-stage switch working as a delay (the first switch is so good in terms of hysteresis that the second switch is not very critical for oscillation).
- Connection
  - The feedback from second to first switch is a bit complex, since both x and y are repressed by degrading cyclin. And there are more details still.



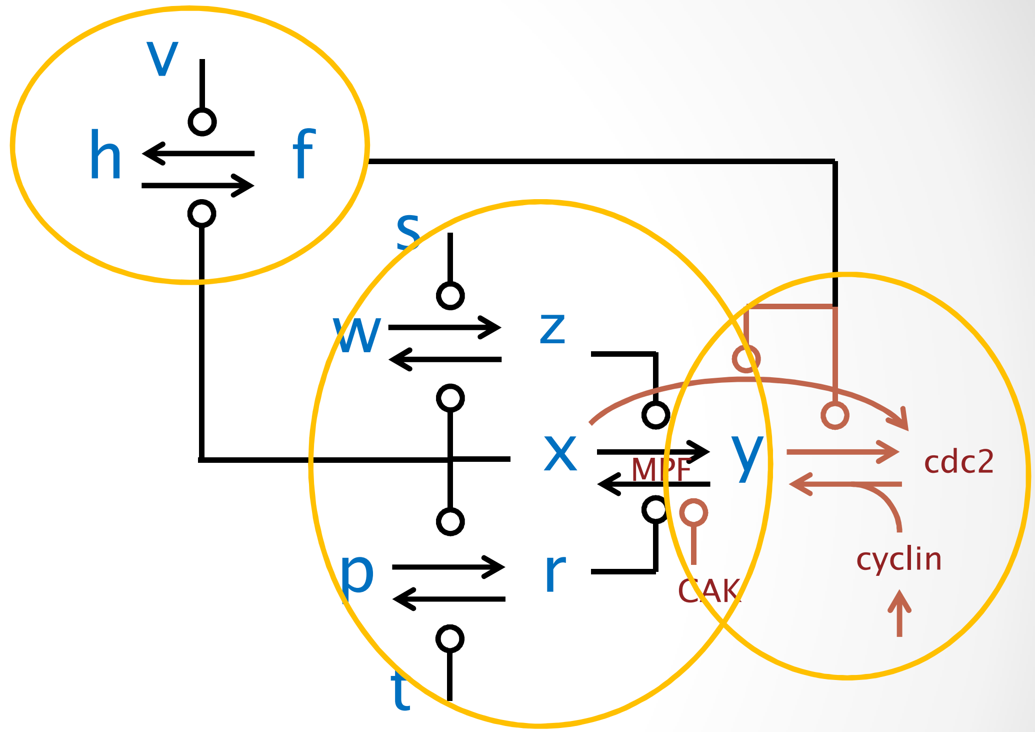
Journal of Cell Science 106, 1153-1168 (1993)
Printed in Great Britain © The Company of Biologists Limited 1993

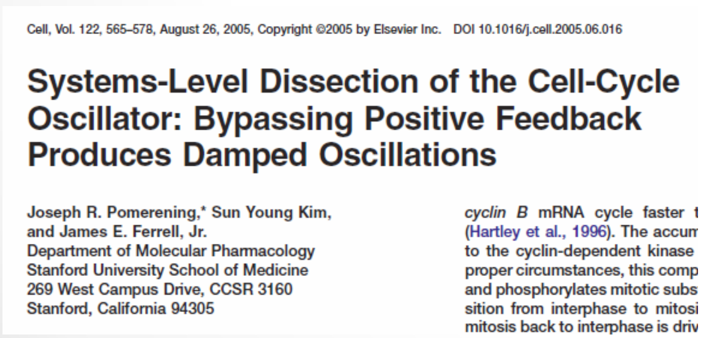
Numerical analysis of a comprehensive model of M-phase control in *Xenopus* oocyte extracts and intact embryos

Bela Novak[*] and John J. Tyson[†]
Department of Biology, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24060-0406, USA

One of Ferrell's Oscillators

- Second switch
  - Replaced by a one-stage switch. The oscillation still works, but is it harder to obtain (parameter tuning).




Cell, Vol. 122, 565–578, August 26, 2005, Copyright ©2005 by Elsevier Inc. DOI 10.1016/j.cell.2005.06.016

Systems-Level Dissection of the Cell-Cycle Oscillator: Bypassing Positive Feedback Produces Damped Oscillations

Joseph R. Pomerening,* Sun Young Kim, and James E. Ferrell, Jr.
Department of Molecular Pharmacology
Stanford University School of Medicine
269 West Campus Drive, CCSR 3160
Stanford, California 94305

cyclin B mRNA cycle faster t
(Hartley et al., 1996). The accum
to the cyclin-dependent kinase
proper circumstances, this comp
and phosphorylates mitotic subs
sition from interphase to mitosi
mitosis back to interphase is driv

Conclusions

Conclusions

- A vast literature on cell cycle switching
  - Ferrell et.al., Novak-Tyson et.al., etc. Mostly ODE based analysis, plus noise
  - Many bistable transitions have different implementations in different cell cycle phases and organisms (phosphorylation, enzymes, synthesis/degradation, etc.)
  - We focused on a mechanism that can only be seen stochastically (quick majority switching with x=y)
- A range of 'network transformation'
  - Can explain the structure of some natural networks
  - From some non-trivial underlying algorithms
  - Discovering the transformation can elucidate the structure and function of the networks
  - But how can we say that these transformations 'preserve (essential) behavior'?

Acknowledgements

- David Soloveichik
- Attila Csikasz-Nagy