On Switches and Oscillators

Program Equivalence in Biology?

Luca Cardelli
Microsoft Research

Turin 2011-05-05
http://lucacardelli.name

Outline

- Some questions that nature has (apparently) answered:
  - Building ‘good’ bistable systems
  - Building ‘switches’ (switchable bistable system)
  - Building switches with hysteresys (needed for good oscillators)
  - Building ‘limit cycle’ oscillators that do not dampen or diverge
  - Building robust oscillators that resist parameter variations

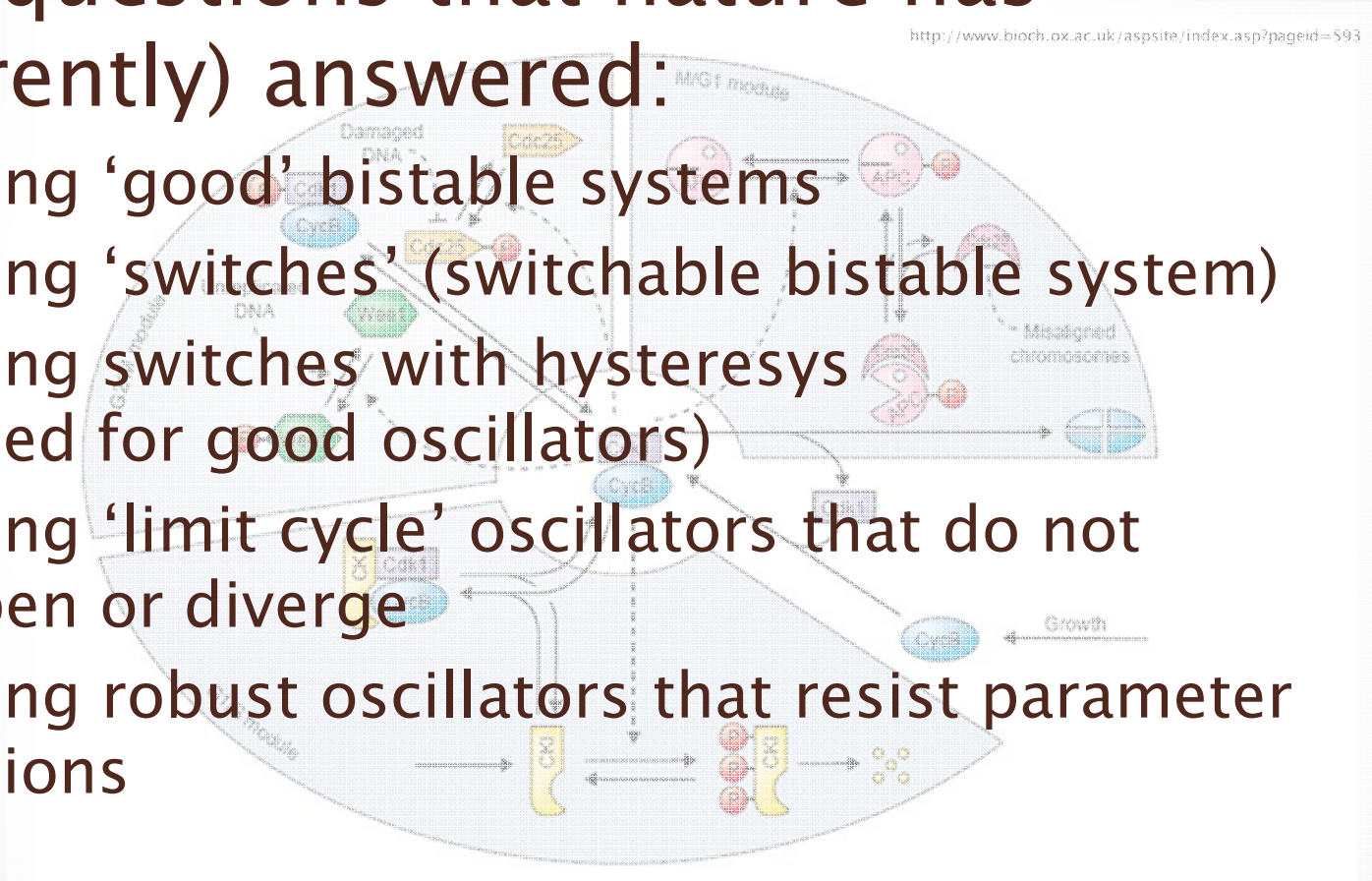


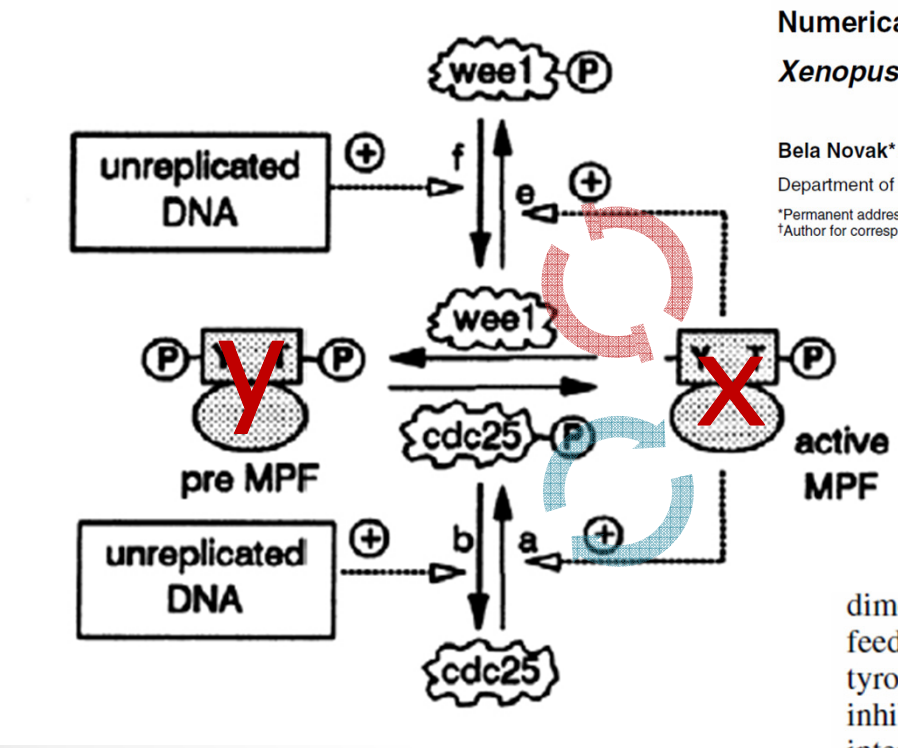
Figure 1: Cell cycle regulation of cyclin dependent kinase (Cdk1) Cyclin-B (CycB) complex. The active Cdk1/CycB dimer can be inactivated by binding to an inhibitor (CKI) or by phosphorylation of the kinase subunit by Wee1. The inhibitory phosphate group is removed by Cdc25 phosphatase (Cdc25). Cdk1/CycB can also be inactivated by proteolysis of its cyclin partner, mediated by the anaphase-promoting complex (APC) in combination with Cdc20

Outline

- Subject to ‘chemical constraints’
  - Not all reactions can be easily implemented
  - Not all molecules can perform all functions
- The ‘logical’ solutions
  - Need to be adapted due to chemical constraints
  - Can we then still recognize them (if they exist)?

Switches

The Cell Cycle Switch



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†

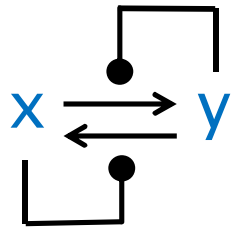
Department of Biology, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24060-0406, USA

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

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

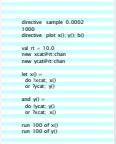
Direct Competition

- x catalyzes the transformation of y into x
- y catalyzes the transformation of x into y

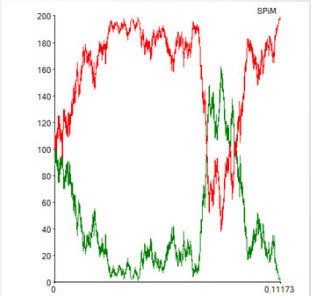


$$x + y \rightarrow x + x$$
$$y + x \rightarrow y + y$$

- 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 (adjusting the rate) you will wait for a long time.



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

val rt = 10.0
new xcatlrt:chan
new ycatlrt: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()
```



Approximate Majority

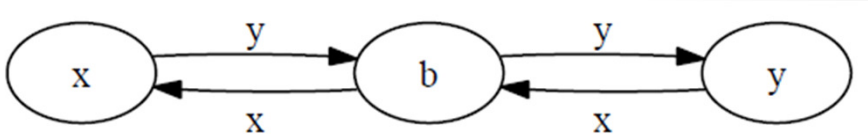
- A fundamental ‘population protocol’
  - Agents in a population start in state x or state y.
  - A pair of agents is chosen randomly at each step, they interact and change state.
  - The whole population must eventually agree on a majority value (x or y) with probability 1.

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).

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

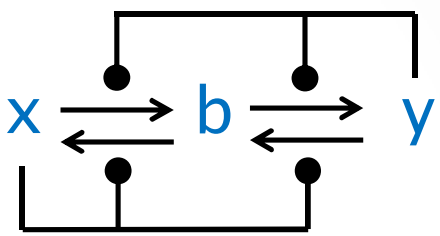
vs. Stochastic Chemistry

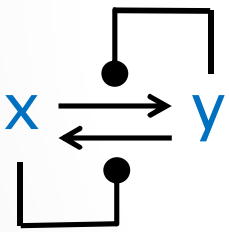
- Chemical systems as distributed systems
  - What can they compute?
  - What can we say about their dynamics?
- Replace "agent" with "molecule" $\Rightarrow$ Chemical Master Equation (modulo details)
- Objection: in real life, some pairs of agents are more likely to interact than others
  - Agents in the same state are interchangeable
  - In a **well-stirred** chemical mixture, reaction *types* occur with the right probabilities (Gillespie, Physica A 1992)

[Angluin et al.]

Chemical Implementation

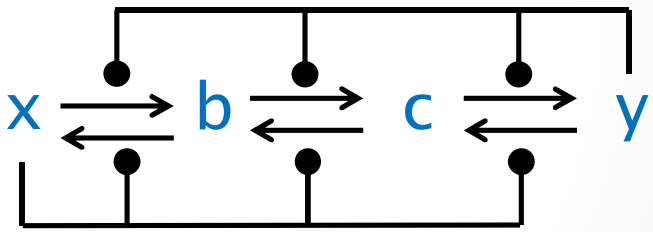
$$x + y \rightarrow y + b$$
$$y + x \rightarrow x + b$$
$$b + x \rightarrow x + x$$
$$b + y \rightarrow y + y$$





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.

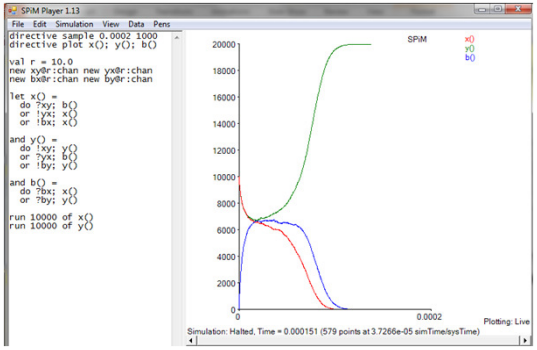


This one gives no significant improvement over the above.

Majority of x=y (CRN)

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

val r = 10.0
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 10000 of x()
run 10000 of y()
```

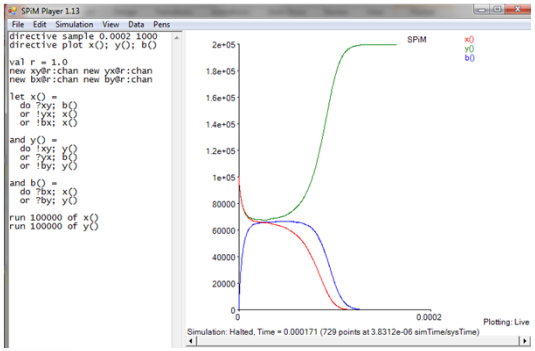
20k molecules, rate 10.0/s



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

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

val r = 1.0
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 y()
```

200k molecules, rate 1.0/s (same concentration)

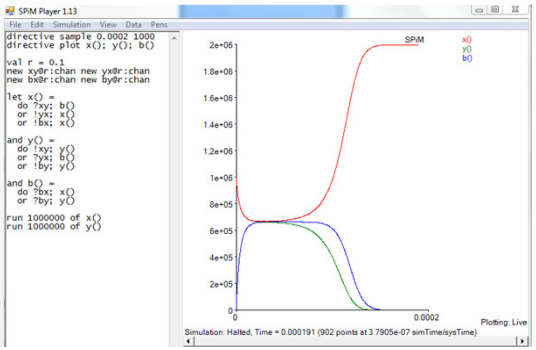


Gillespie simulation of the chemical reactions in SPiM.

- 10x more molecules at same concentration (i.e. lower rate) converge in 'comparable' time.
- Not shown: 10x more molecules in same volume, (i.e. higher concentration) converge 10x faster.

```
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, rate 0.1/s (same concentration)



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

Bistable Element

- 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: it's a switch.

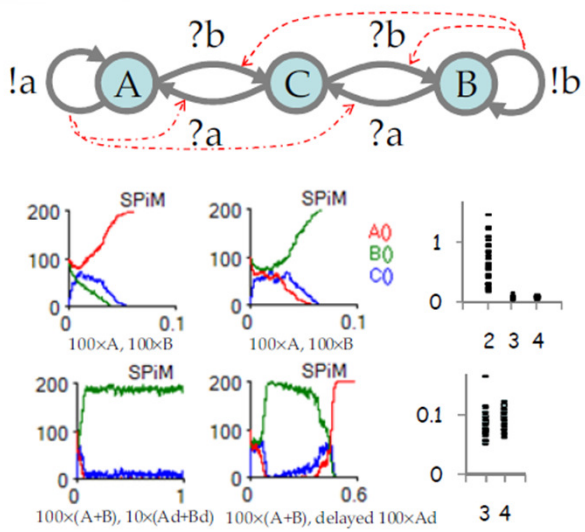


Figure 34 Memory Elements

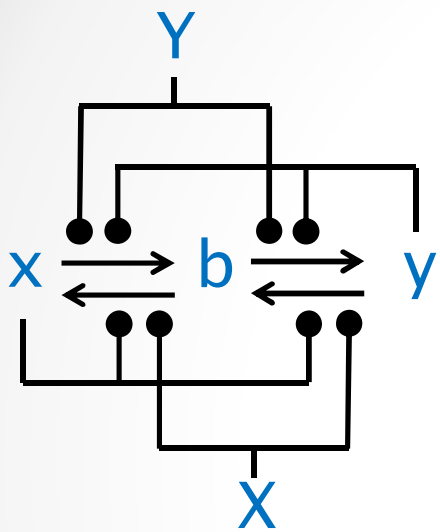
A + B -> B + C
B + A -> A + C
C + A -> A + A
C + B -> B + B

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.

As a Flip-Flop



Outputs x,y:

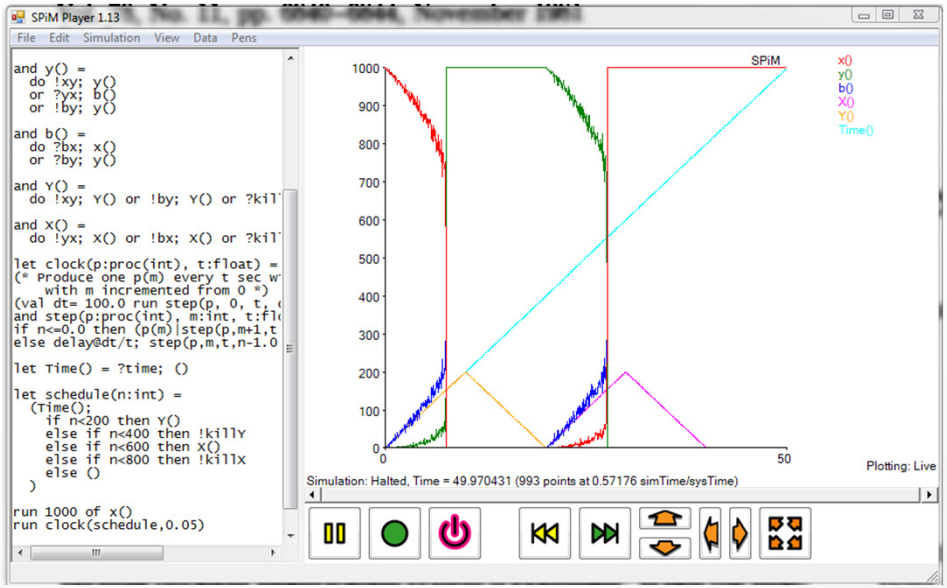
$x + y \rightarrow y + b$
$y + x \rightarrow x + b$
$b + x \rightarrow x + x$
$b + y \rightarrow y + y$

Inputs X,Y:

$x + Y \rightarrow Y + b$
$b + Y \rightarrow Y + y$
$y + X \rightarrow X + b$
$b + X \rightarrow X + y$

```
directive sample 50.0 1000
directive plot x(); y(); b(); X(); Y(); Time()

val r = 1.0
new xy@r:chan new yx@r:chan
new bx@r:chan new by@r:chan
new killX:chan new killY:chan
new time: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()

and Y() =
  do !xy; Y() or !by; Y() or ?killY; ()

and X() =
  do !yx; X() or !bx; X() or ?killX; ()

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<200 then Y()
   else if n<400 then !killY
   else if n<600 then X()
   else if n<800 then !killX
   else ()
  )

run 1000 of x()
run clock(schedule,0.05)
```

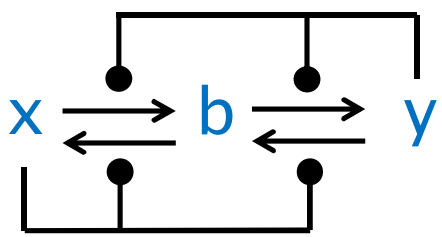
Can be switched by external (catalytic) signals that are only 20% of the x,y levels.



Init: 1000 x
Y growing from 0 (t=0) to 200 (t=200) then back to 0 (t=400)
X growing from 0 (t=400) to 200 (t=600) then back to 0 (t=800).

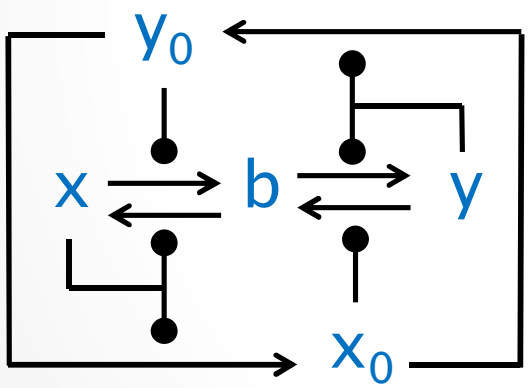
Chemical Constraints

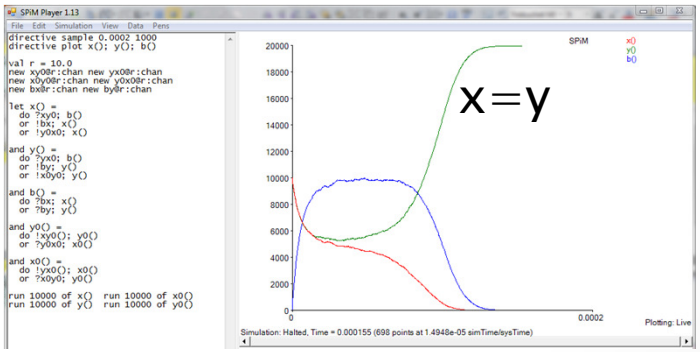
- This circuit is ‘chemically demanding’
  - It requires x molecules to be ‘next’ to y molecules beacause they interact directly
  - It requires both x and y to be catalysts, and in fact autocatalysts, and in fact each-other’s autocatalyst!



Program 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()
```

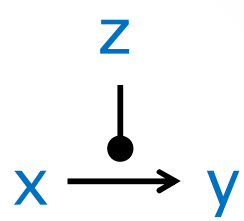
Program Transformations

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

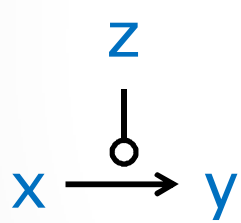
Some Notation

- Catalytic reaction

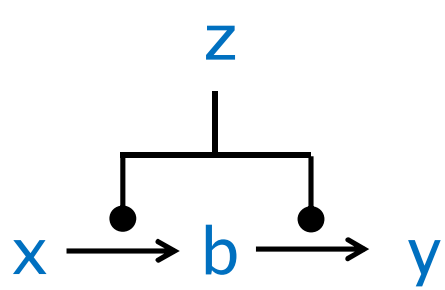
$x + z \rightarrow z + y$ =



- Non-linear reaction



=

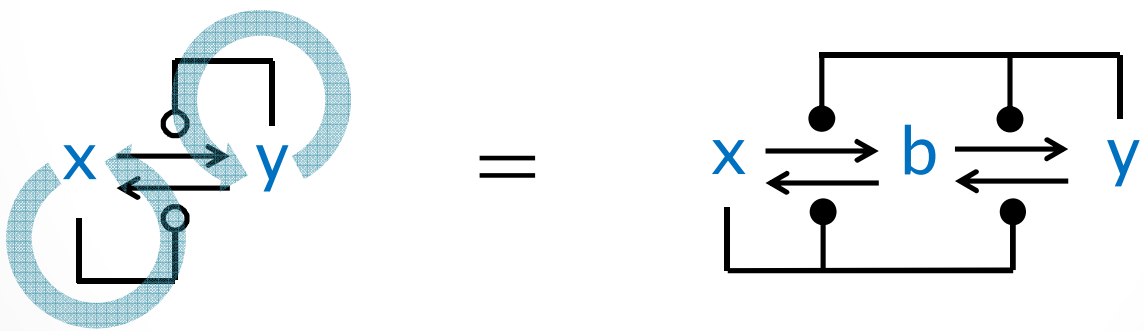


The hollow circle may represent any 'non-linear' reaction (e.g. enzymatic, hill) in addition to this 'double-phosphorylation' network, which however is the standard interpretation here.

Zero-Input Switches

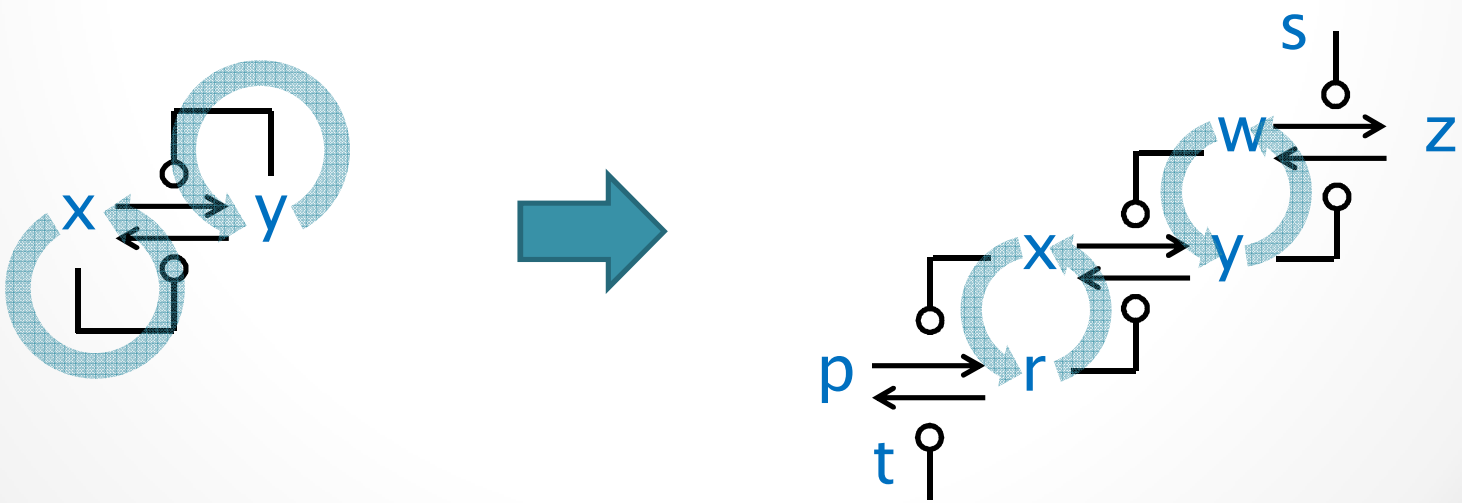
- ‘Zero-input switch’ = majority circuit: just working off the initial conditions.

- Step 1: the original AM Network



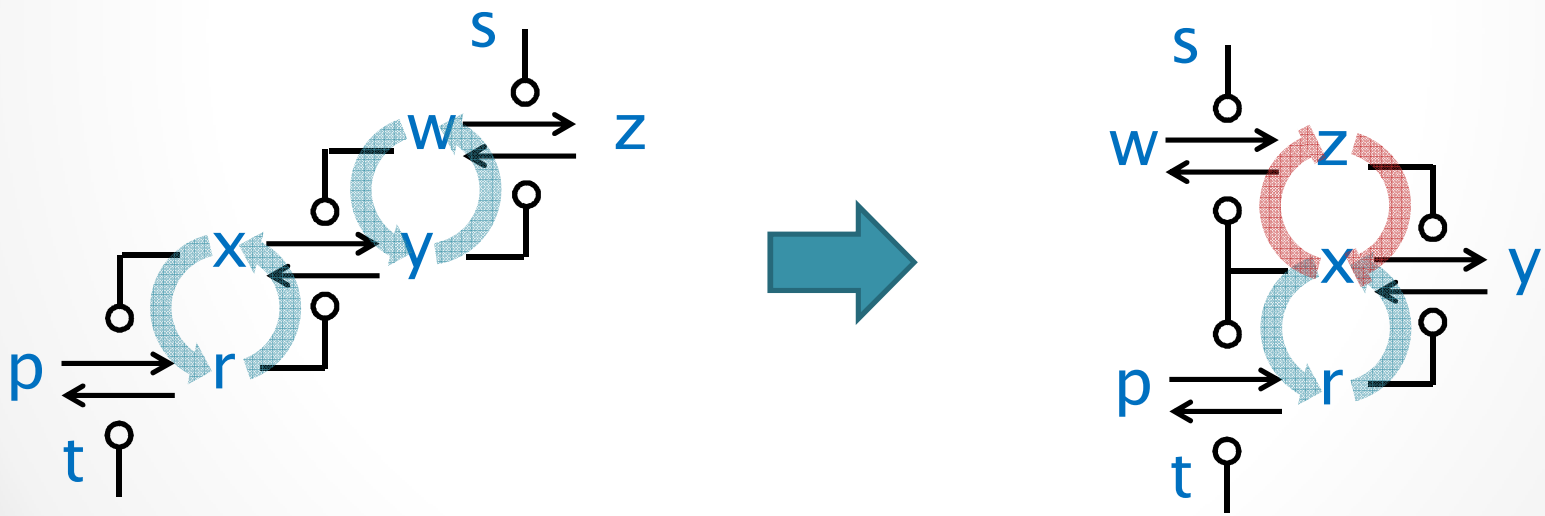
Zero-Input Switches

- 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.



Zero-Input Switches

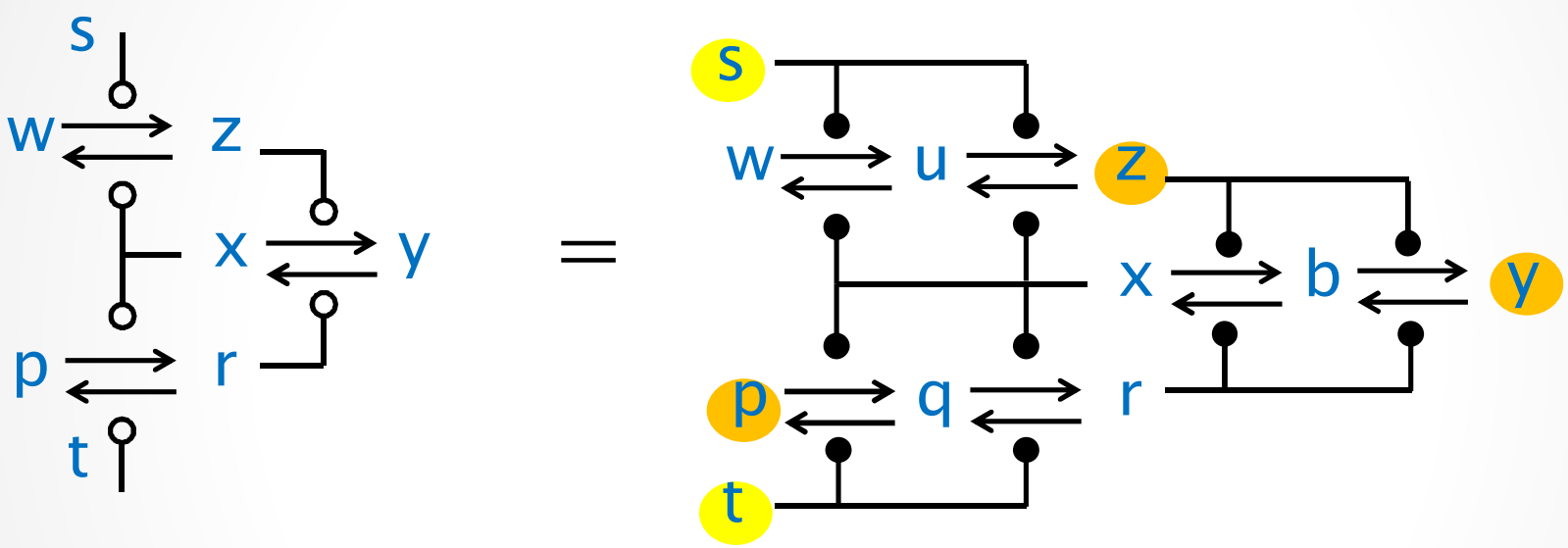
- Step 3: transform a double-positive loop on y into a double-negative loop on x.
  - Instead of y (actively) activating itself through w, we have z activating y (which is passive). To counteract, now x has to swich from inhibiting y to inhibiting z.



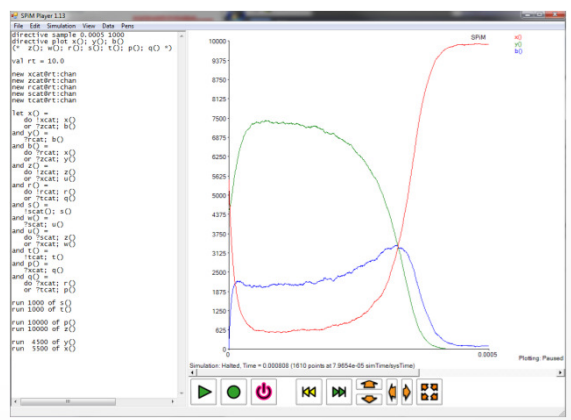
- So that y no longer catalyzes anything.

Zero-Input Switches

- Still an AM circuit

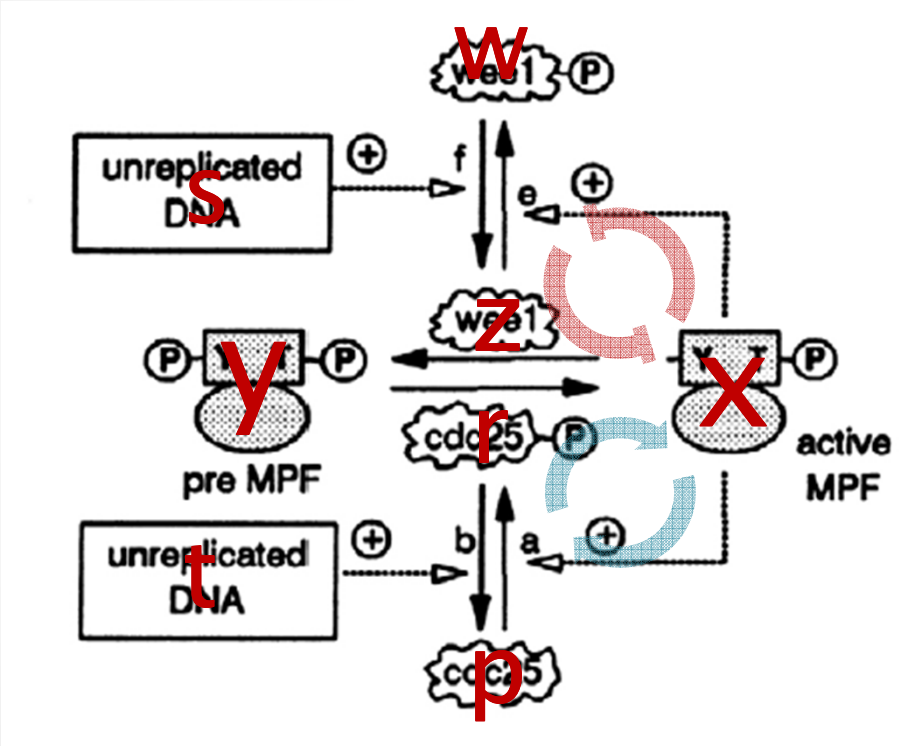


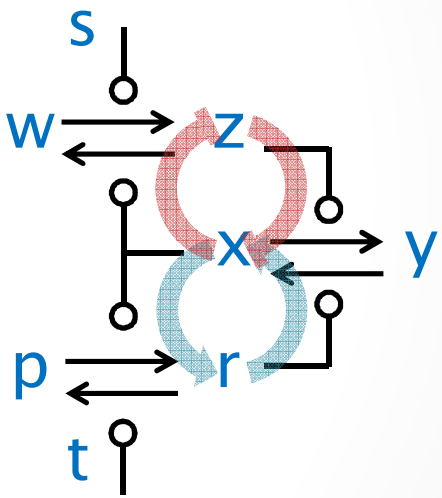

```
directive sample
0.0005 1000
directive plot x(); y();
b()
(* z(); w(); r(); s(); t();
p(); q() *)

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()
```



(Although the equal-likelihood outcome here is around 4500 y vs 5500 x, and there are other paremeters)

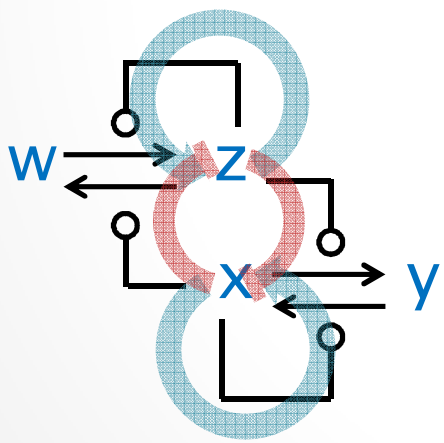
The Cell Cycle Switch

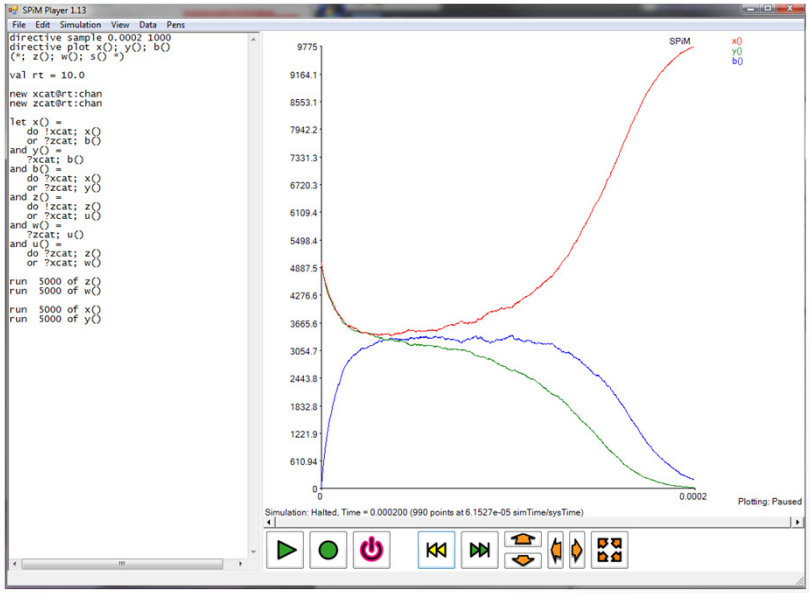




Zero-Input Switches

- 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.

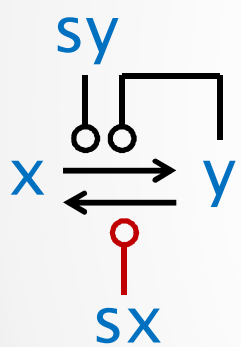


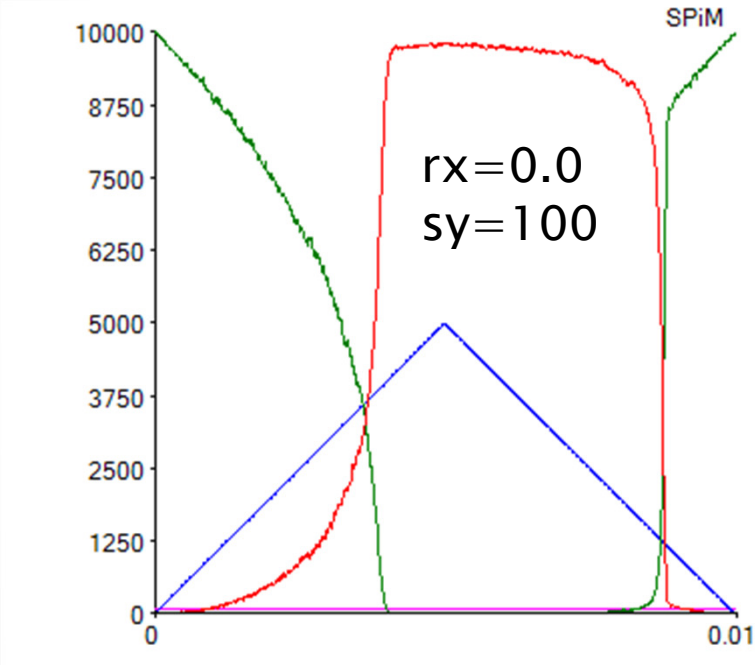


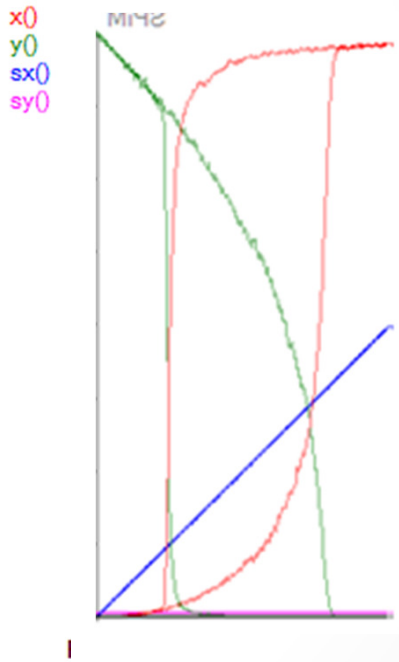

```
directive sample 0.0002 1000
directive plot x(); y(); b()
(*; z(); w(); s() *)

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()
```

One-Input Switches

- Hysteresis in AM-like switches

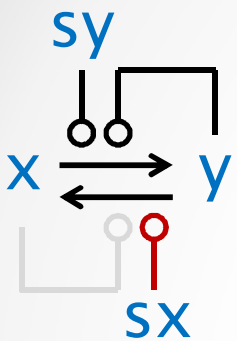




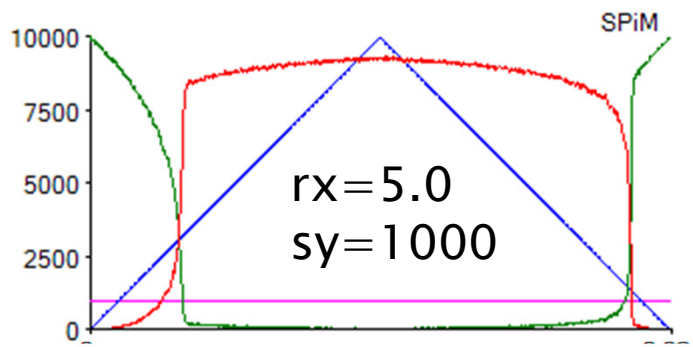


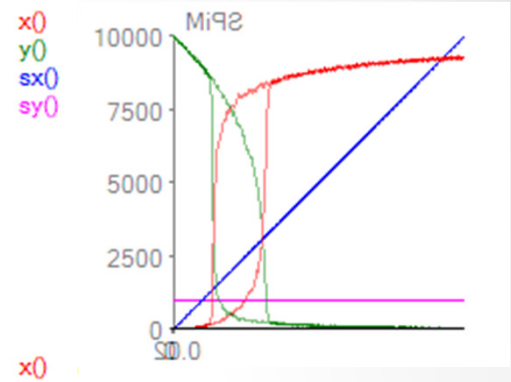

```
directive sample 0.01 1000
directive plot x(); y(); sx(); sy() (* b();  *)

val rt = 10.0
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 10000 of y()
run 100 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 < 5000 then sx()
 else if n < 10000 then !sxkill;()
 else ()

run clock(schedule,0.000001)
```

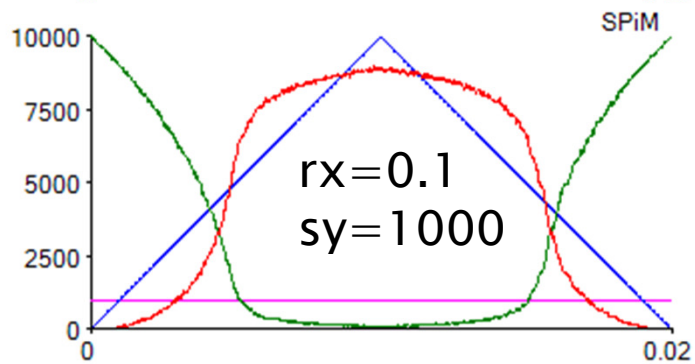
One-Input Switches

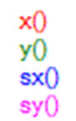


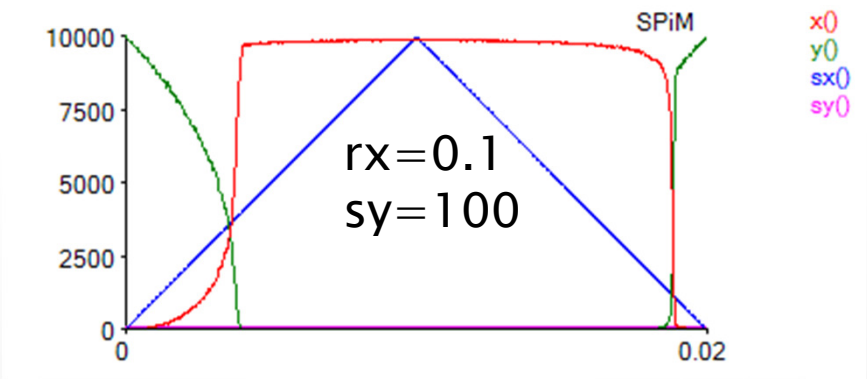

```
directive sample 0.02 1000
directive plot x(); y(); sx(); sy() (* b();  *)

val rt = 10.0
val rx = 1.0
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 < 5000 then sx()
  else if n < 10000 then !sxkill;()
  else ()

run clock(schedule,0.000001)
```

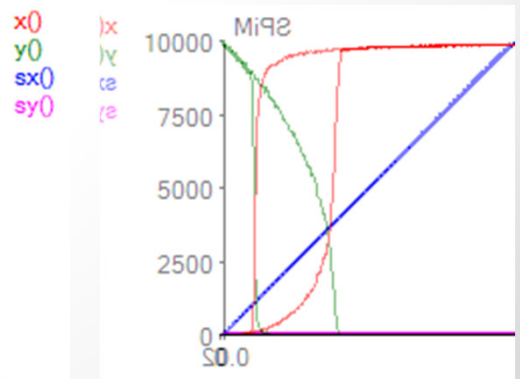






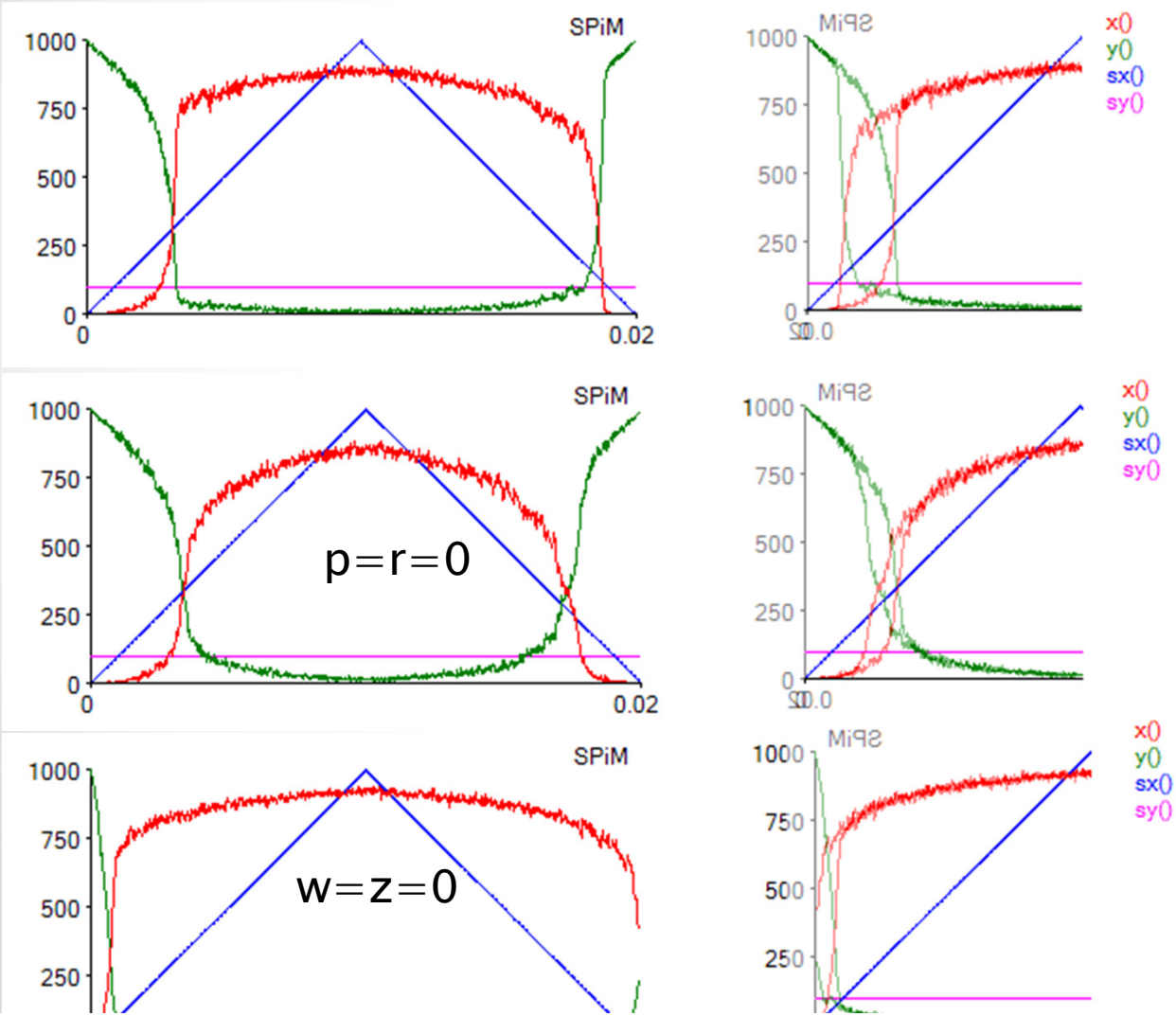


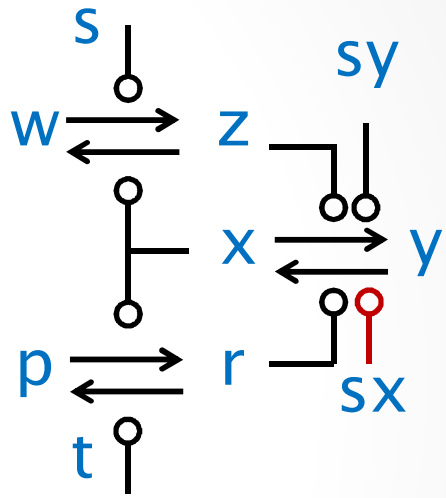


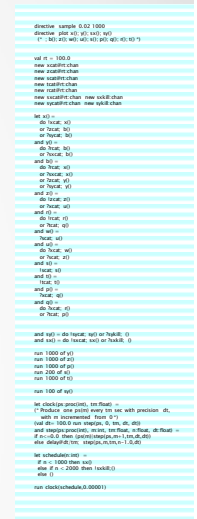


One-Input Switches

- Hysteresis in cell cycle switches

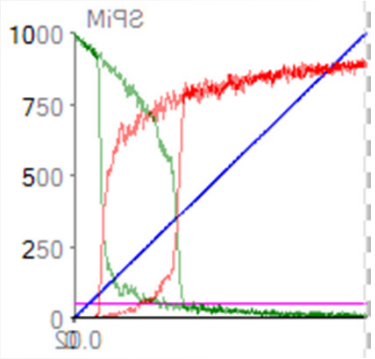




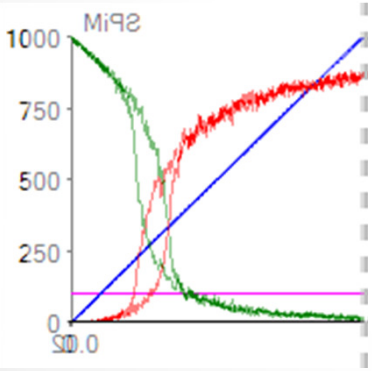


initial conditions:
1000 of y
1000 of z
1000 of p
1000 of t
200 of s
100 of sy

varying sx 0 to 1000



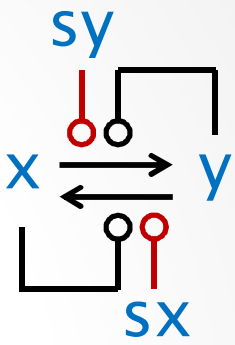
twice s, half t, and half sy



five times t

Two-input Switches

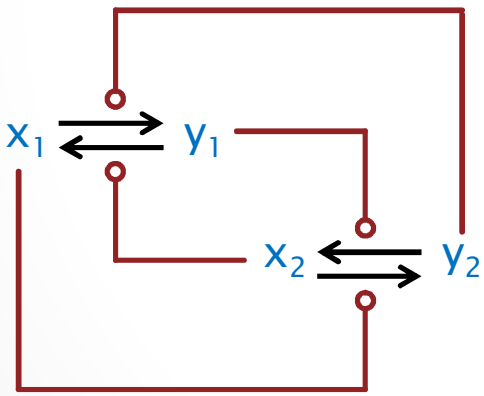
- (not really relevant here)

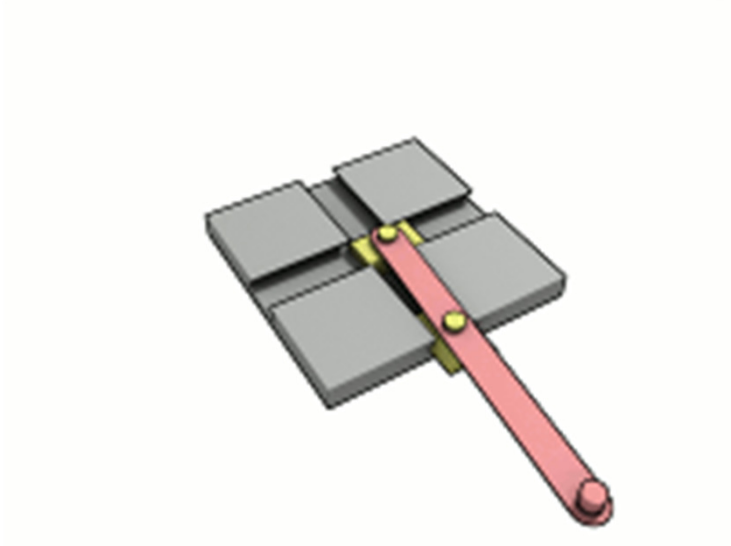


Oscillators

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).

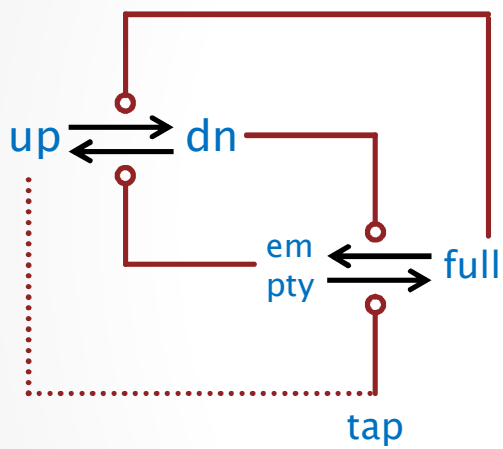




en.wikipedia.org/wiki/Trammel_of_Archimedes

The Shishi Odoshi

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



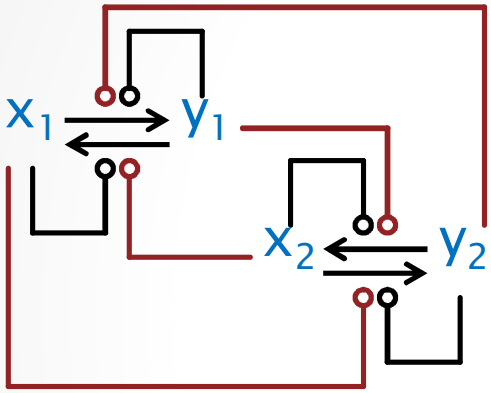
http://www.youtube.com/watch?v=VbvecTIftcE&NR=1&feature=fvwp

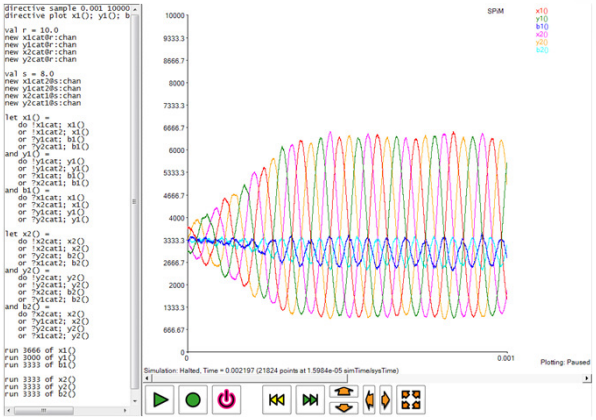
empty + tap → tap + full
up + full → full + dn
full + dn → dn + empty
dn + empty → empty + up

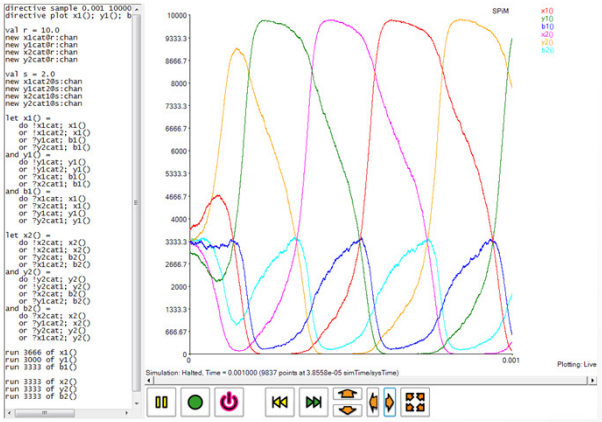
To make it into a full trammel (dotted line), we could make the up position mechanically open the tap (i.e. take up = tap)

The 2AM Limit-Cycle Oscillator

- Two AM switches in a Trammel pattern







```
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()
```

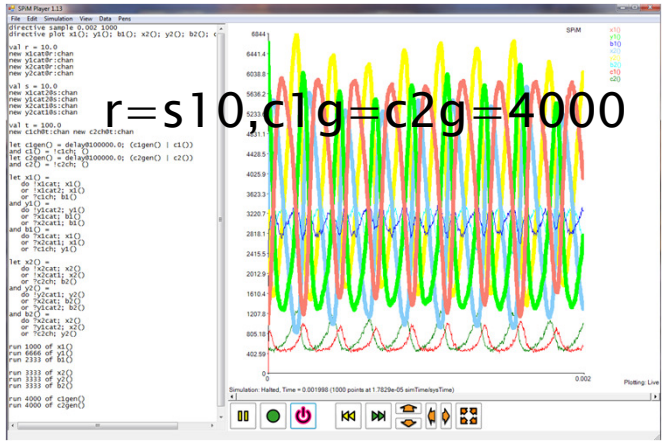
The red reactions need to be slower (even slightly) than the black reactions, but otherwise the oscillation is robust. Oscillation stops at 10 vs. 10 and 1 vs. 10. Here the rates are 8 vs 10.0 top, and 2 vs 10, bottom.

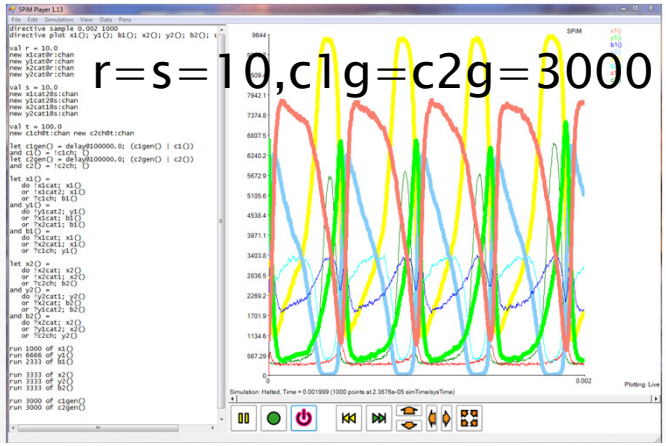
(Simple limit-cycle oscillators in the literature have very critical rate ranges.)

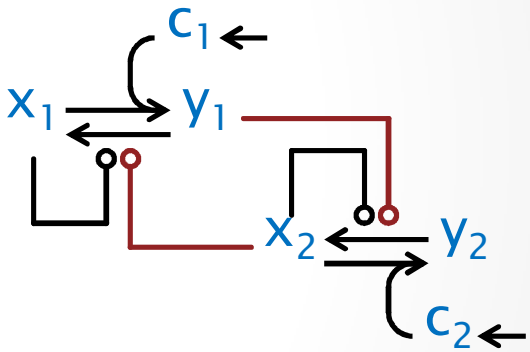
Influx Oscillators

- Similar but:
  - The two-input switches are replaced by one-input switches which are reset by constant influxes.

```
directive sample 0.002 1000
directive plot x1(); y1(); b1(); x2(); y2(); b2(); c1();
c2()

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

val s = 10.0
new x1cat2@s:chan
new y1cat2@s:chan
new x2cat1@s:chan
new y2cat1@s:chan

val t = 100.0
new c1ch@t:chan new c2ch@t:chan

let c1gen() = delay@100000.0; (c1gen() | c1())
and c1() = !c1ch; ()
let c2gen() = delay@100000.0; (c2gen() | c2())
and c2() = !c2ch; ()

let x1() =
  do !x1cat; x1()
  or !x1cat2; x1()
  or ?c1ch; b1()
and y1() =
  do !y1cat2; y1()
  or ?x1cat; b1()
  or ?x2cat1; b1()
and b1() =
  do ?x1cat; x1()
  or ?x2cat1; x1()
  or ?c1ch; y1()

let x2() =
  do !x2cat; x2()
  or !x2cat1; x2()
  or ?c2ch; b2()
and y2() =
  do !y2cat1; y2()
  or ?x2cat; b2()
  or ?y1cat2; b2()
and b2() =
  do ?x2cat; x2()
  or ?y1cat2; x2()
  or ?c2ch; y2()

run 1000 of x1()
run 6666 of y1()
run 2333 of b1()

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

run 4000 of c1gen()
run 4000 of c2gen()
```





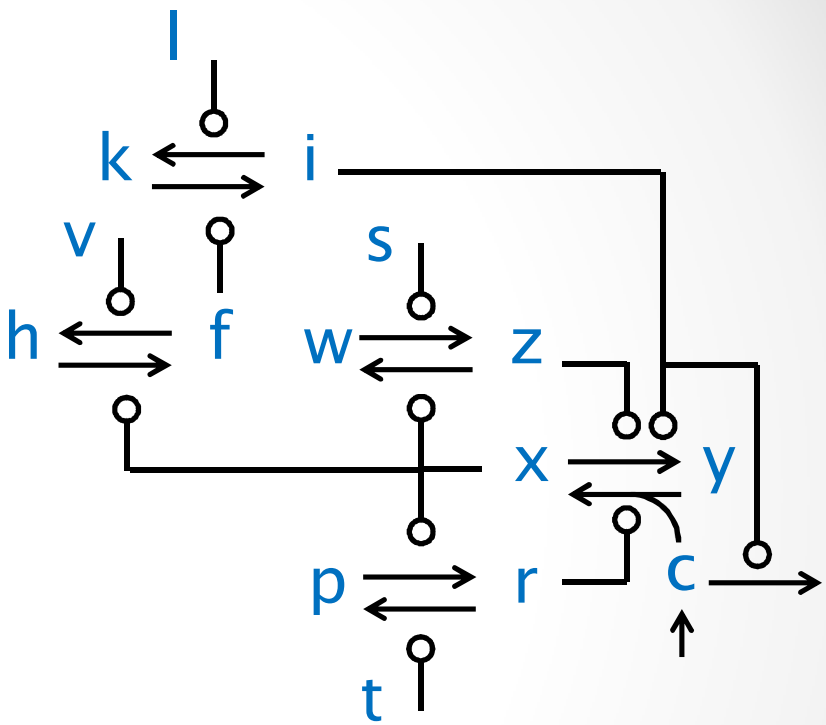


Works best with s=r.

Needs constant influx of c1,c2

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).
  - It can be replaced by a one-stage switch (Ferrell's cell cycle osciallor) but oscillation is a bit harder to obtain.
- Connection
  - Single links, as in the influx oscillator.



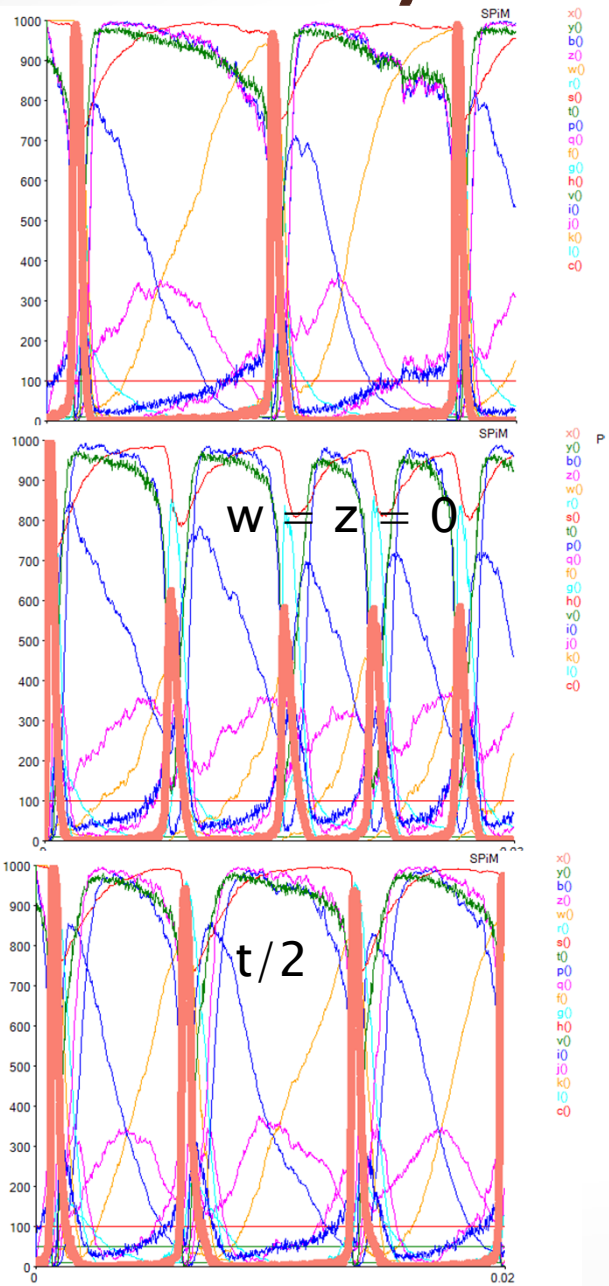

Journal of Cell Science 106, 1153-1168 (1993)
Printed in Great Britain © The Company of Biologists Limited 1993

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

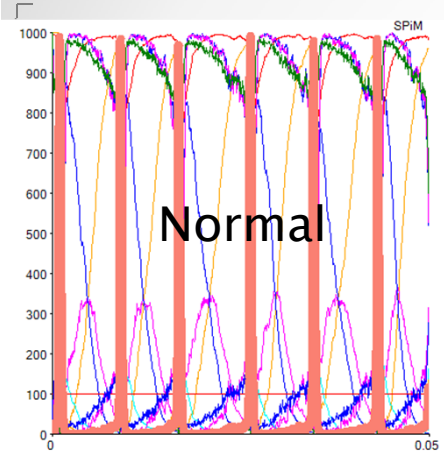
Novak-Tyson Oscillator

```
directive sample 0.02 1000
directive plot x(); y(); b(); z(); w(); r(); s(); t(); p(); q(); f(); g();
h(); v(); i(); j(); k(); l(); c()

val rt = 100.0
val rt2 = 1.0
val rt3 = 200.0
val rc = 10000.0

new c@rt:chan
new xcat@rt:chan new zcat@rt:chan new rcat@rt:chan
new scat@rt:chan new tcat@rt:chan new vcat@rt:chan
new lcat@rt:chan

new fcat@rt3:chan new icat@rt3:chan
new xcat2@rt2:chan

let ci() = delay@rc; (cy()|ci())
and cy() = !c; ()

let x() = do !xcat; x() or !xcat2; x() or ?zcat; b() or ?icat; b()
and y() = do ?rcat; b() or ?c; b()
and b() = do ?rcat; x() or ?c; x() or ?zcat; y() or ?icat; 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()
and f() = do !fcat; f() or ?vcat; g()
and g() = do ?vcat; h() or ?xcat2; f()
and h() =  ?xcat2; g()
and v() = !vcat; v()
and i() = do !icat; i() or ?lcat; j() or ?c; i()
and j() = do ?lcat; k() or ?fcat; i()
and k() = ?fcat; j()
and l() = !lcat; l()

run 100 of s()
run 100 of t()
run 10 of v()
run 10 of l()

run 1000 of y()
run 1000 of z()
run 1000 of p()
run 1000 of h()
run 1000 of k()

run 1000 of ci()
```

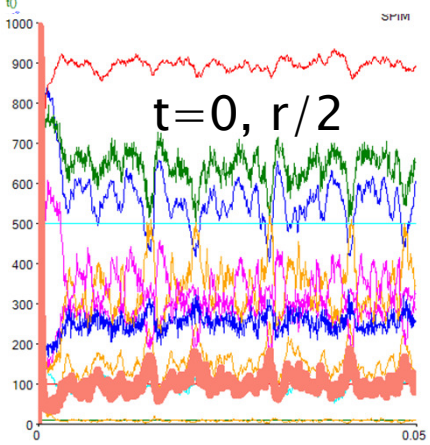


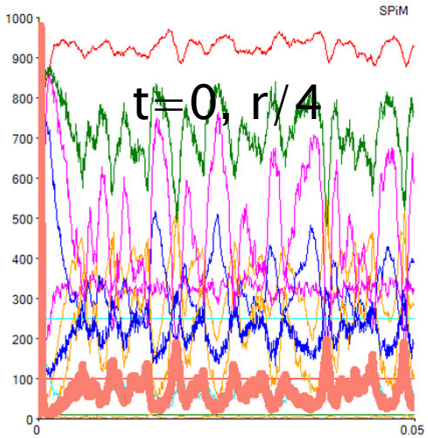
p=r=0, or t/4, or t*2:
no oscillation

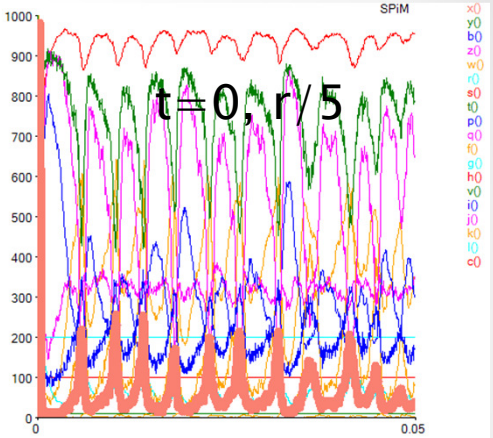
Without double-positive loop

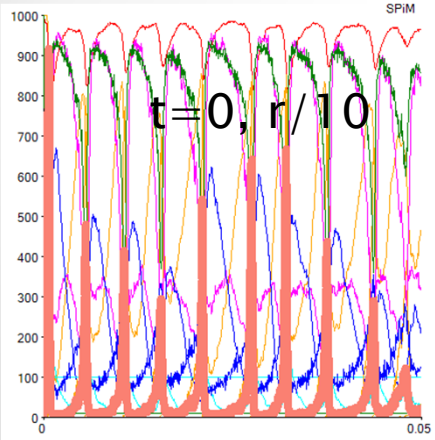


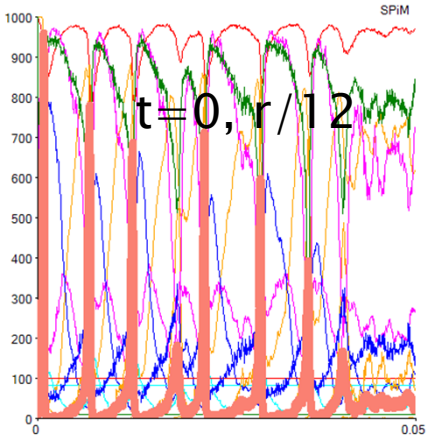
(then, block t and change amount of r)

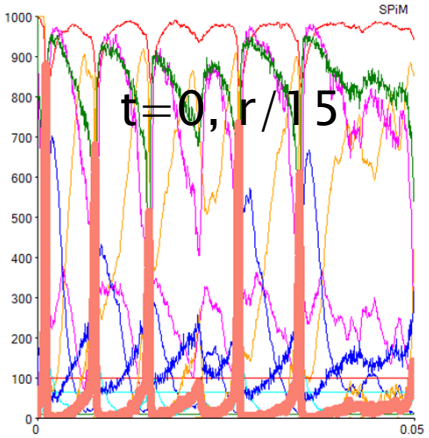


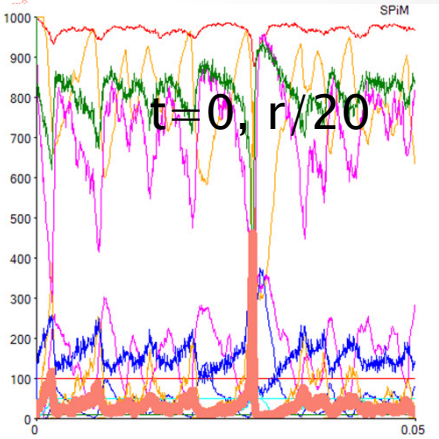




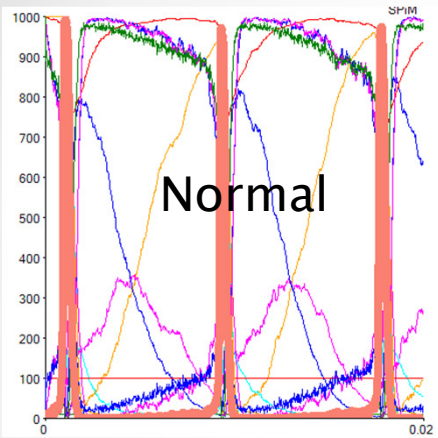




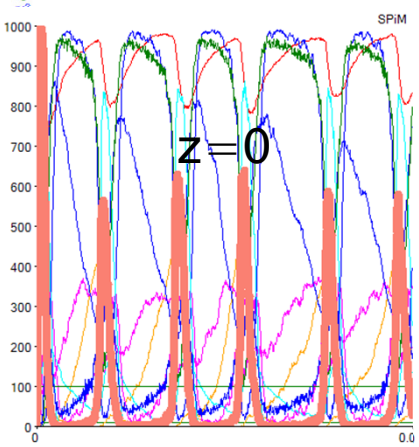


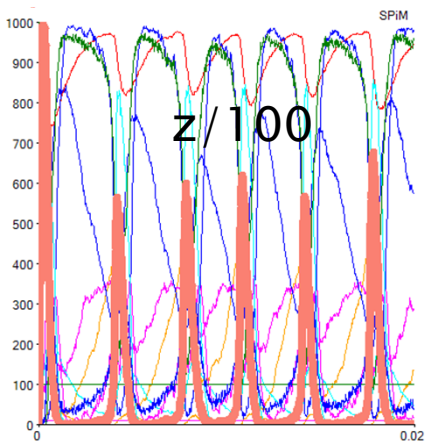


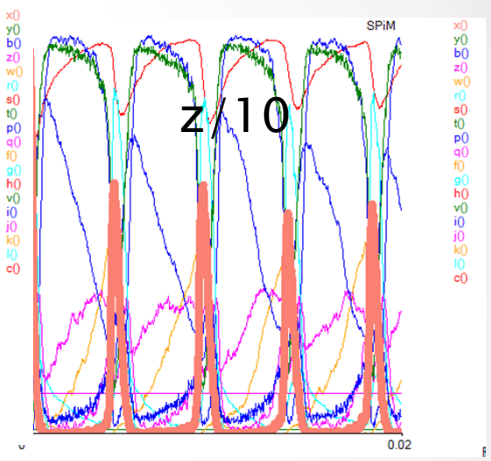
Without double-negative loop

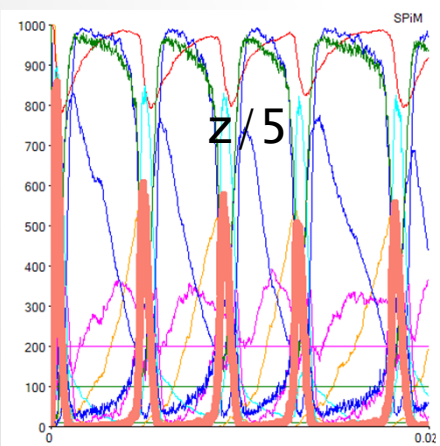


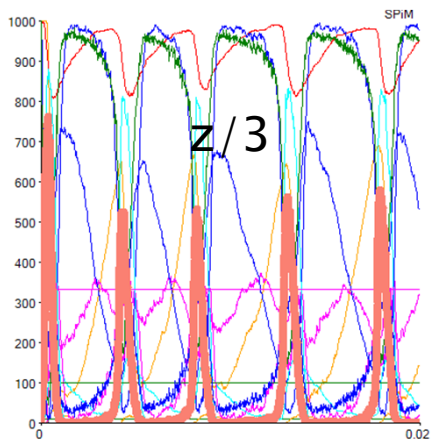
(then, with x no longer acting on z, change amount of z)

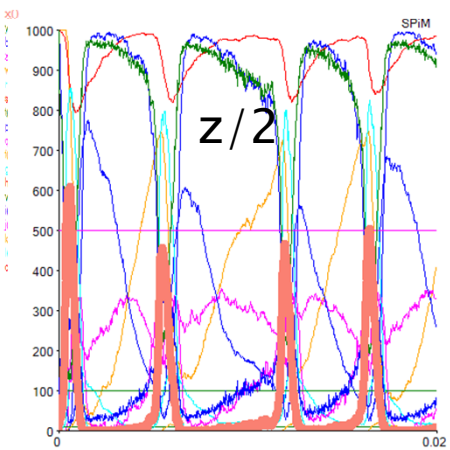


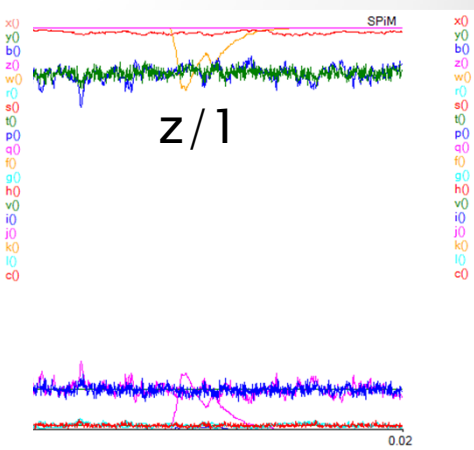


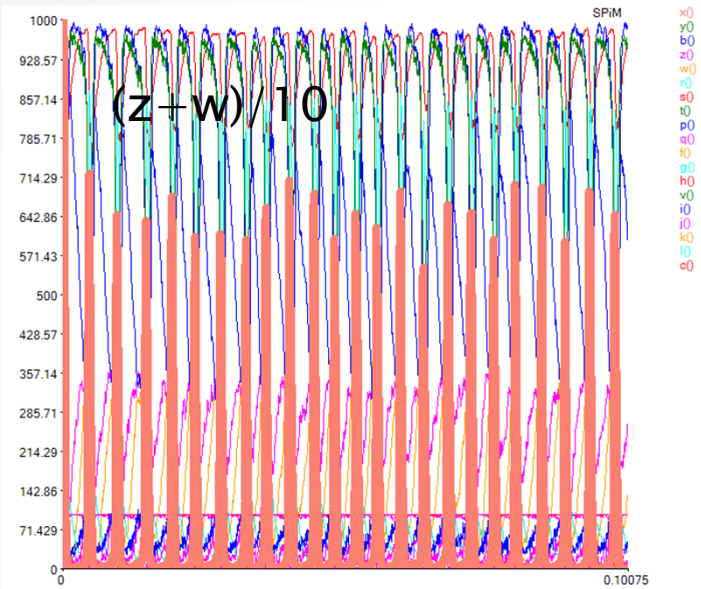


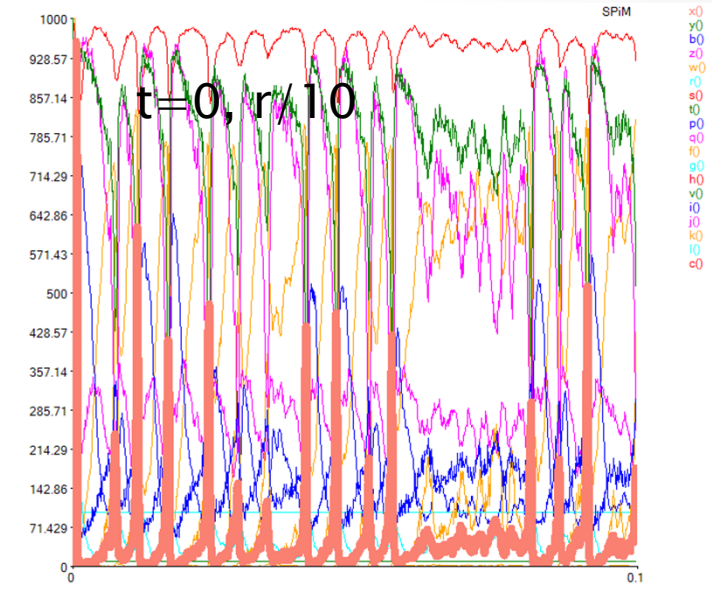


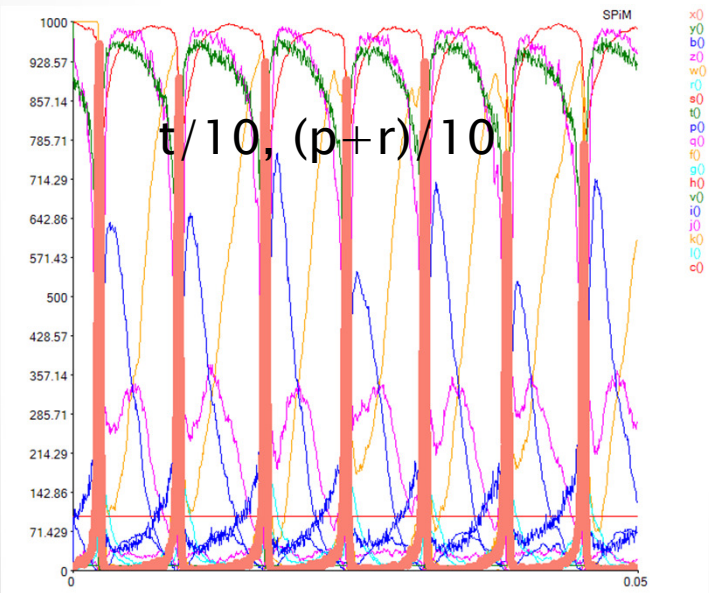


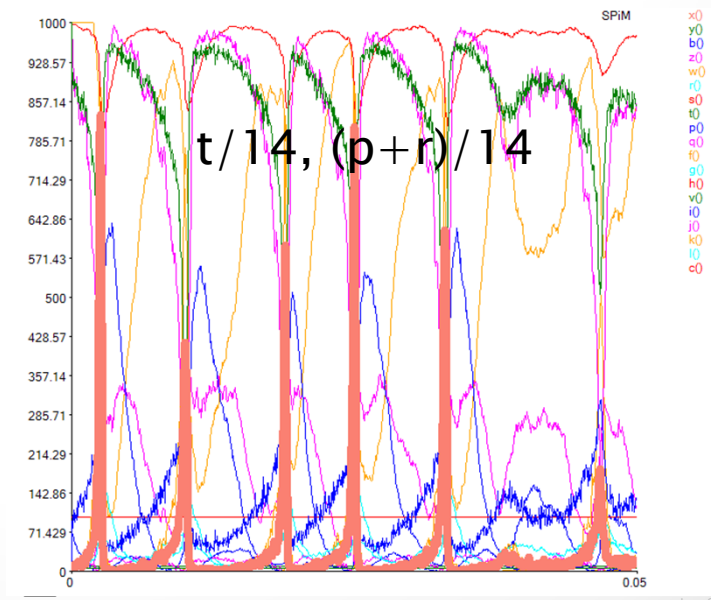





(z+w)/10
SPiM
1000
928.57
857.14
785.71
714.29
642.86
571.43
500
428.57
357.14
285.71
214.29
142.86
71.429
0
0
0.10075
x0 y0 b0 z0 w0 r0 s0 t0 p0 q0 f0 g0 h0 v0 i0 j0 k0 l0 c0


t=0, r/10
SPiM
1000
928.57
857.14
785.71
714.29
642.86
571.43
500
428.57
357.14
285.71
214.29
142.86
71.429
0
0
0.1
x0 y0 b0 z0 w0 r0 s0 t0 p0 q0 f0 g0 h0 v0 i0 j0 k0 l0 c0


t/10, (p+r)/10
SPiM
1000
928.57
857.14
785.71
714.29
642.86
571.43
500
428.57
357.14
285.71
214.29
142.86
71.429
0
0
0.05
x0 y0 b0 z0 w0 r0 s0 t0 p0 q0 f0 g0 h0 v0 i0 j0 k0 l0 c0


t/14, (p+r)/14
SPiM
1000
928.57
857.14
785.71
714.29
642.86
571.43
500
428.57
357.14
285.71
214.29
142.86
71.429
0
0
0.05
x0 y0 b0 z0 w0 r0 s0 t0 p0 q0 f0 g0 h0 v0 i0 j0 k0 l0 c0

Conclusions

Conclusions

- A range of ‘network transformation’
  - Can explain the structure of some natural network
  - 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’?