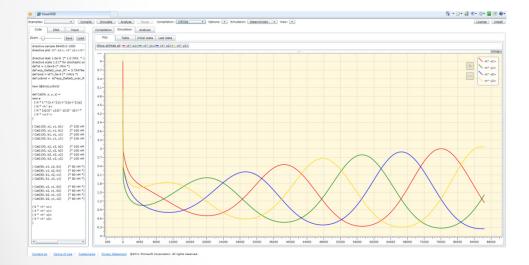
## Network Transformations of Switches and Oscillators

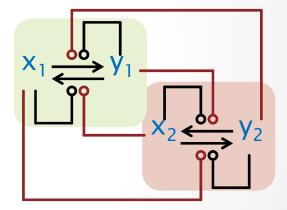
Luca Cardelli Microsoft Research

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

### Motivation

### • Building synthetic (DNA) oscillators





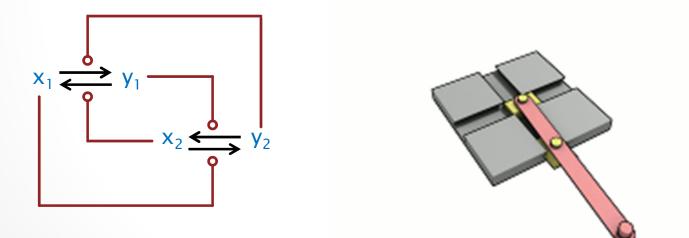
#### **DSD** simulation

L

### The Trammel of Archimedes

### A device to draw ellipses

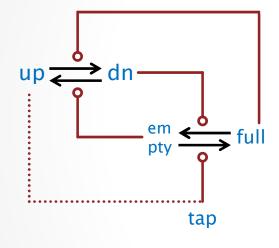
- Two interconnected switches.
- Note that amplitude is kept constant by mechanical constraints.
- When one switch is on (off) it flips the other switch on (off).
   When the other switch is on (off) it flips the first switch off (on).



en.wikipedia.org/wiki/Trammel\_of\_Archimedes

### The Shishi Odoshi

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



 $empty + tap \rightarrow tap + full$  $up + full \rightarrow full + dn$  $full + dn \rightarrow dn + empty$  $dn + empty \rightarrow empty + up$ 

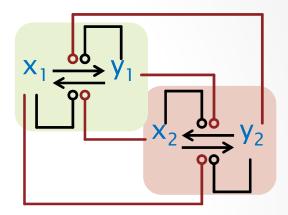


http://www.youtube.com/watch?v=VbvecTIftcE&NR=1&feature=fvwp

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 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

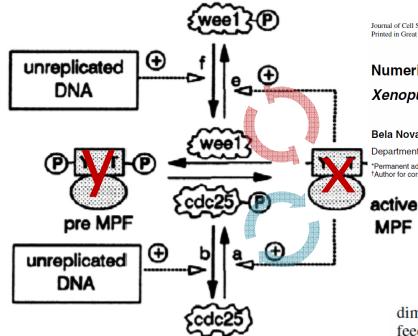
### • From the point of view of network structure

- 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<sup>†</sup>

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 .... ??

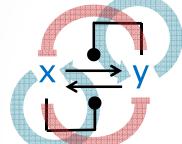
dimers is left off the diagram to keep it simple.) (B) Positive feedback loops. Active MPF stimulates its own production from tyrosine-phosphorylated dimers by activating Cdc25 and inhibiting Wee1. We suspect that these signals are indirect, but intermediary enzymes are unknown and we ignore them in 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

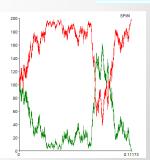
x catalyzes the transformation of y into x

y catalyzes the transformation of x into y



 $\begin{array}{c} x + y \rightarrow x + x \\ y + x \rightarrow y + y \end{array}$ 

- 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.



### A Very Good Algorithm

# Approximate Majority Decide which of two populations is in majority

### A fundamental 'population protocol'

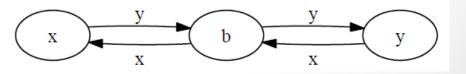
- Agents in a population start in state x or state y.
- A pair of agents is chosen randomly at each step, they interact ("collide") and change state.
- The whole population must eventually agree on a majority value (all x or all y) with probability 1.

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.

 $\begin{array}{ccccc} x & b & y \\ x & (x,x) & (x,x) & (x,b) \\ b & (b,x) & (b,b) & (b,y) \\ y & (y,b) & (y,y) & (y,y) \end{array}$ 



### **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 Ω(n log n) steps in expectation for all agents to interact

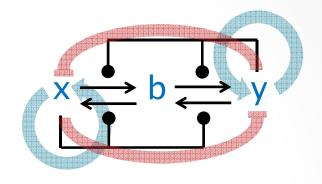
[Angluin et al.]

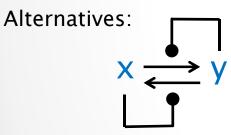
"Parallel time" is the number of steps divided by the number of agents. Hence the algorithm terminates with high probability in O(log n) steps per agent.

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

### **Chemical Implementation**

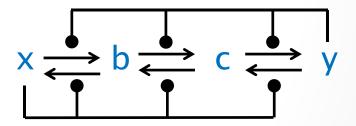
 $x + y \rightarrow y + b$   $y + x \rightarrow x + b$   $b + x \rightarrow x + x$  $b + y \rightarrow y + y$ 





This too is a bistable system, but:

- It converges slowly, by a random walk, hence O(n<sup>2</sup>).
- 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 v() =

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

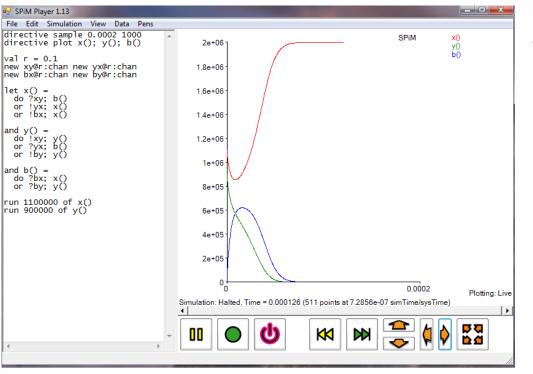
and b() = do ?bx; x()

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.

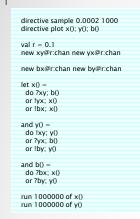


 $\begin{array}{l} x+y\rightarrow y+b\\ y+x\rightarrow x+b\\ b+x\rightarrow x+x\\ b+y\rightarrow y+y\end{array}$ 

Eventually: all x no y no b

All rates are equal.

### Majority of x = y (!!)



#### 2000k molecules

Gillespie simulation of the chemical reactions in SPiM.

#### All rates are equal.

- 0 - 23 File Edit Simulation View Data Pens directive sample 0.0002 1000 SPiM ×0 directive plot x(); y(); b() 2e+06 у0 Ь0 new xy@r:chan new yx@r:chan new bx@r:chan new by@r:chan 1.8e+06 1.6e+06 1.4e+06 1.2e+06 1e+06 8e+05 6e+05 4e+05 2e+05 0.0002 Plotting: Live Simulation: Halted, Time = 0.000191 (902 points at 3.7905e-07 simTime/sysTime) 4

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

r:

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

 $\omega(\sqrt{n \log n})$ 

SPiM Player 1.13

val r = 0.1

let x() =

and y() =

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

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()

#### N.B. a deterministic (ODE) simulation with x=ywould 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**

v

Ultrasensitive (but no hysteresis)

e	$x + E \rightarrow E + b$ $b + E \rightarrow E + y$	AM without feedbacks	kinase/ phosphata
eresis)	$y + F \rightarrow F + b$	٦	
∎ SPIM Player 1.13	$b + F \rightarrow F + x$	X _	$\rightarrow b \xrightarrow{\bullet}$
File Edit Simulation View Data Pens			
directive sample 100000.0 1000 directive plot x(); y(); b(); F();E() val a = 0.0001 new e@a:chan new f@a:chan new killE:chan new killF: chan new time:chan	9000	SPiM         x0           y0         b0           F0         E0	
<pre>let E() = do !e; E() or ?killE; () and F() = do !f; F() or ?killF; () and x() = ?e; b() and y() = ?f; b() and b() = ?f; b() and b() = do ?e; y() or ?f; x()</pre>		{	F
let clock(p:proc(int), t:float) = (" produce one p(m) every t sec with (w with m incremented from 0 ") (val dt= 100.0 run step(p, 0, t, dt, and step(p:proc(int)) m:int, t:float if n<-0.0 then (p(m) step(p,m+1.t, dt) = lse delayddt/t; step(p,m,t,1-1.0, dt) = (t) =			
<pre>let Time() = ?time; () let schedule(n:int) =    (Time();    if n&lt;1000 then ()    else if n&lt;2000 then E()    else if n&lt;2000 then !killE</pre>	4000 E= 3000 2000		
else () )  run 10000 of x() run 100 of F() run clock(schedule,10.0)	Simulation: Halted. Time = 179022.697659 (1790 points at 599.01 sim	F=100	
٠			

directive sample 100000.0 1000 directive plot x(); y(); b(); F(); F(); F(); f(); val a = 0.0001 new e@a.chan new f@a.chan new killE:chan new killF: chan new time:chan let F() = do le: F() or %killF: ()

and F(0) = d0 | f; F(0) or ?killF; ()and <math>x(0) = ?e; b()and y(0) = ?f; b()and y(0) = ?f; b()and b(0) = d0 ?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 2000 (t - 400) then had

E growing from 0 (t=100) to 3000 (t=400) then back to 0 (t=800)

## The Goldbeter-Koshland Switch

#### Ultrasensitive (but no hysteresis)

directive sample 600.0 1000 directive plot S();P();E();ES();FP() (\*;Time() \*)

val k = 1.0 new es@ac.han new fp@ac.han new killE:chan new killE: chan new time:chan (' S + E <>> SE -> P + E P + F <>> PF -> S + F \*) let S0 = ?es; 0

and P() = ?fp; () and F() = do !fp; FP() or ?killF; ()

let clock(p:proc(int), t:float)

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

else () ) run 10000 of S() run 1000 of F() run clock(schedule.0.1)

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

and FP() = do delay@d; (P()|F()) or delay@k; (S()|F())

(\* 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; ()

val a = 1.0 val d = 1.0

esis)	$S + E_d \leftrightarrow$ $P + F_d \leftrightarrow$					$\varsigma \xrightarrow{\bullet} \varsigma$
						•
- SPiM Player 1.13	racensitive					
File Edit Simulation View D	ita Pens					-
val a = 1.0 val d = 1.0 val k = 1.0	();E();ES();FP() (*;Time() *)	10000		SPiM	S0 P0 F0 E0	F
new es@a:chan new fp@a:c new killE:chan new killF new time:chan	han : chan	8000 -	Internet and an and	-	ES0 FP()	
1 °2	2 P + F <-> PF -> S + F	7000 · S	•   P	S		
<pre>let S() = ?es; () and E() = do !es; ES() c and ES() = do delay@d; (</pre>	$S() E()\rangle$ or delay( $k$ ( $P() E()\rangle$ )	6000	E=200			
	r ?killF; () P() F()) or delay@k; (5() F())		E=1000 E=	=1000 E=0		
<pre>let clock(p:proc(int), t (* Produce one p(m) ever with m incremented f</pre>	:float) = y t sec with precision dt, rom 0 *)	4000 -		L=U		
and step(p:proc(int), m: if n<=0.0 then (p(m) step)	<pre>p, 0, t, dt, dt)) int, t:float, n:float, dt:float p(p,m+1,t,dt,dt)) m,t,n-1.0,dt)</pre>	2000 ·			F=1000	
<pre>let Time() = ?time; ()</pre>	,_,,	1000			→	
<pre>let schedule(n:int) =     (Time();     if n&lt;1000 then ()</pre>		0		600	00 Plotting: Live	
else if n<3000 then else if n<6000 then else ()	E() !killE	Simulation: Halted, Time =	59974.693885 (1000 points at	14.278 simTime/sysTime		
י run 10000 of ג()		- 00 🔘 🌘	<b>b</b> ka ma		54 54	

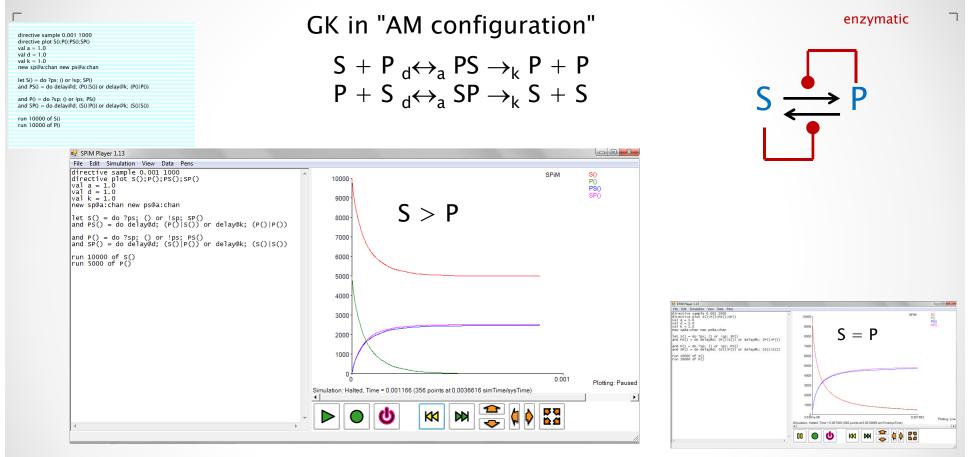
enzymatic

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?



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

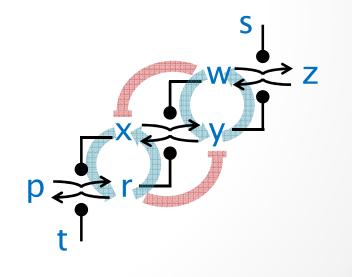
- The initial minority goes down to 0
- The initial majority goes down to  $maj_{t=0}$   $min_{t=0}$
- When  $maj_{t=0} \sim min_{t=0}$  the system cannot decide.

- 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

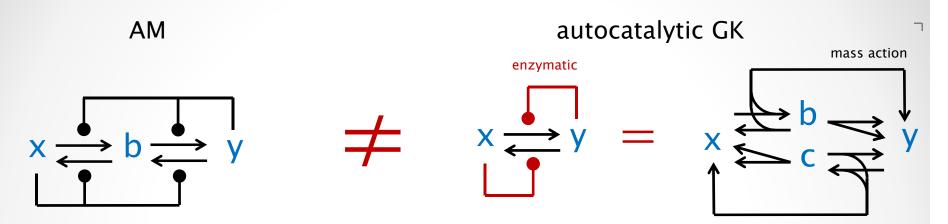
$$x + w _{d} \leftrightarrow_{a} xw \rightarrow_{k} y + w _{d} \leftrightarrow_{a} yr \rightarrow_{k} x + r$$

$$p + x _{d} \leftrightarrow_{a} px \rightarrow_{k} r + x _{r} + t _{d} \leftrightarrow_{a} rt \rightarrow_{k} p + t$$

$$w + s _{d} \leftrightarrow_{a} ws \rightarrow_{k} z + s _{z} + y _{d} \leftrightarrow_{a} zy \rightarrow_{k} w + y$$



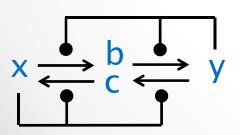
### 'Double phosphorylation' motif is key





Г

split-AM

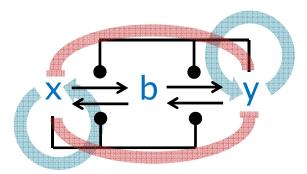


It is not just a non-linearity of the x-y transition mechanism that matters:

it is the 'double phosphorylation' network structure of AM, with a *common* 'undecided' state.

### **Chemical Constraints**

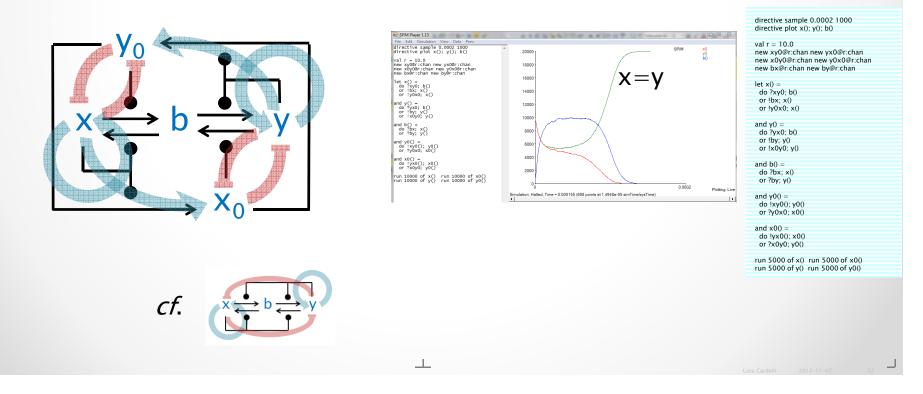
- The AM circuit is 'chemically demanding'
  - It requires x molecules to be 'next' to y molecules beacause they interact directly
  - It requires both x and y to be catalysts, and in fact autocatalysts, and in fact each-other's autocatalyst!



### **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.

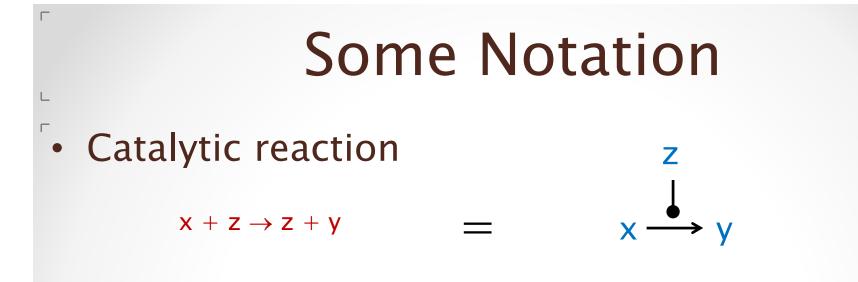


### **Network Transformations**

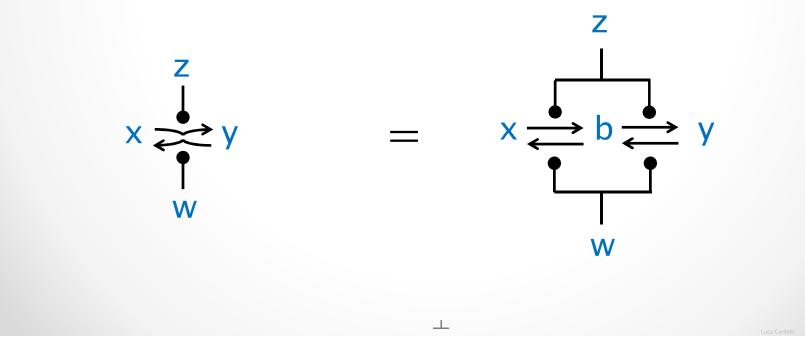
Another example of relaxing constraints

 Build an Approximate Majority network that requires
 only x to be a catalyst. How?

• Enter the Cell Cycle switches...

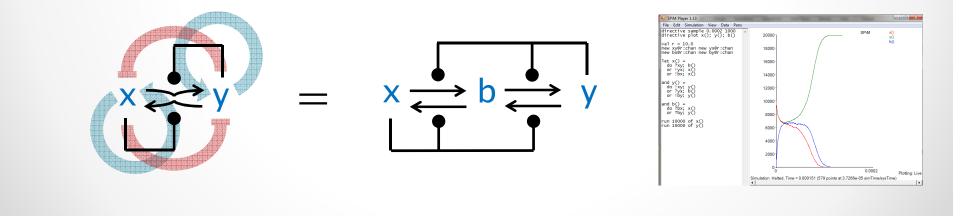


Double 'kinase-phosphatase' reactions



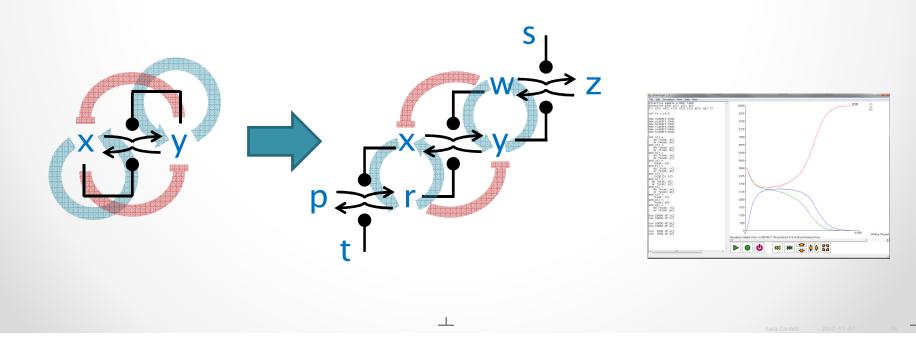
### Zero-Input Switches

- 'Zero-input switch' = majority circuit: just working off the initial conditions, with no other inputs.
- Step 1: the original AM Network

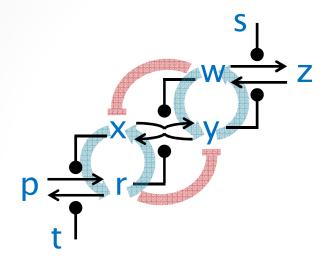


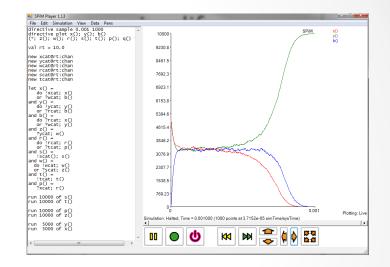
### Zero-Input Switches

- 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?



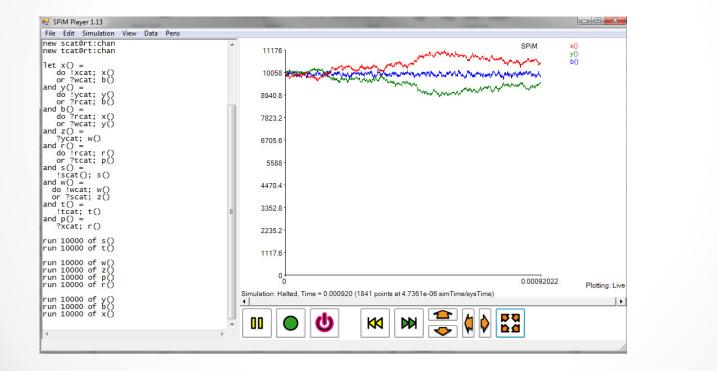


#### (it appears just sligthly noisier/slower)

## ... no, it gets stuck! Equal-size initial conditions

 $\square$ 

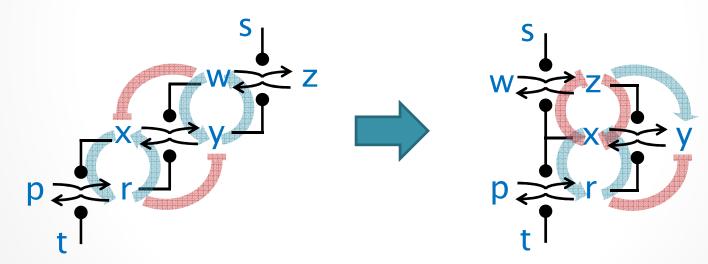
•



directive sample 0.000	5
1000	
directive plot x(); y(); b	0
(*; z(); w(); r(); s(); t(); p	0;
q() *)	
undunt 10.0	
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()	
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() =	
ltcat; 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 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.

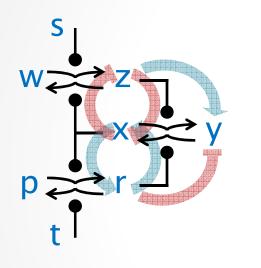


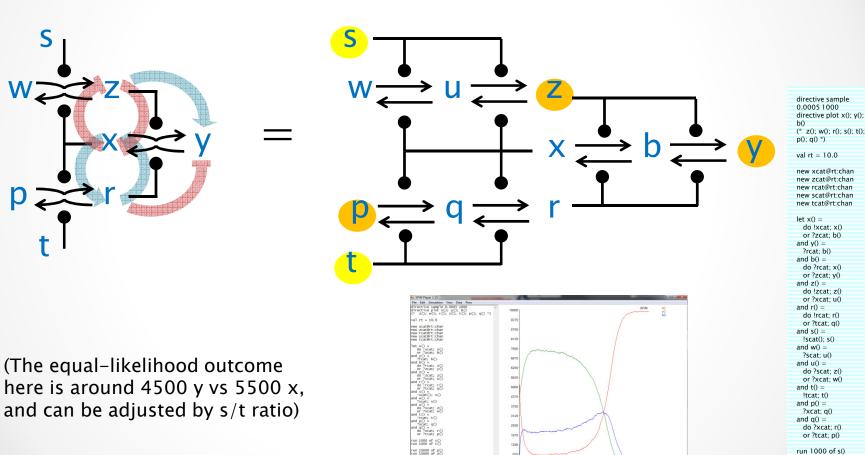
So that y no longer catalyzes anything
 All species have one active and one inactive form

### **Zero-Input Switches**

un 4500 of yO

#### $\square$ Still an AM circuit





40 23

ወ M DD run 1000 of t()

run 10000 of p()

run 10000 of z()

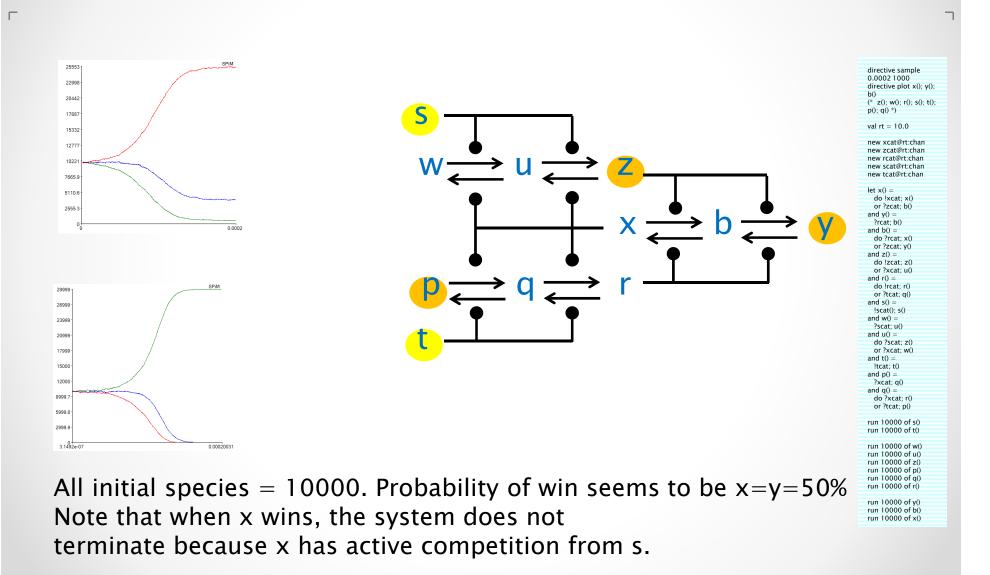
run 4500 of y() run 5500 of x()

and can be adjusted by s/t ratio)

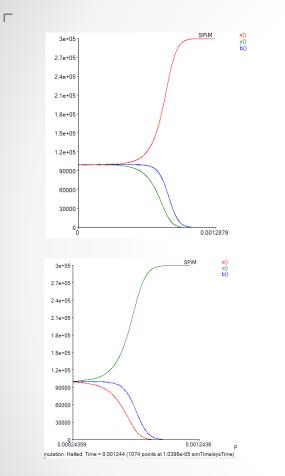
(The equal-likelihood outcome

All rates are equal.

### Equal-size initial conditions



## AM Equal-size initial conditions

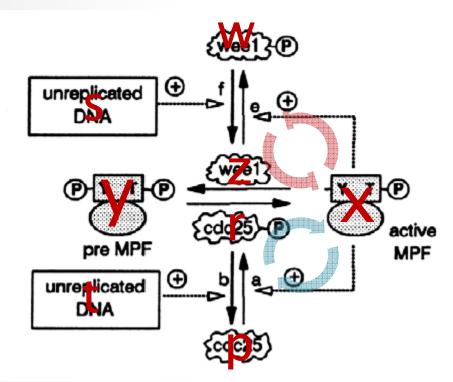


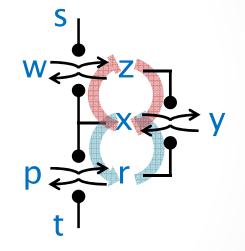
All initial species = 100000.

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; y0
and b() =
do ?bx; x()
or ?by; y()
run 100000 of x()
run 100000 of b()
run 100000 of y()

directive cample 0.002 1000

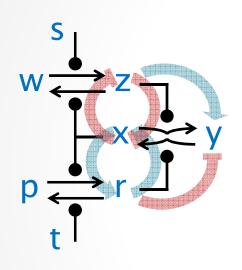
### The Cell Cycle Switch

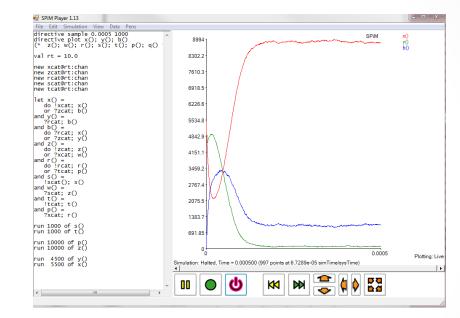




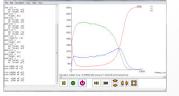
(Some of the bistable states can be enzymatic rather than AM.)

### ... can simplify?

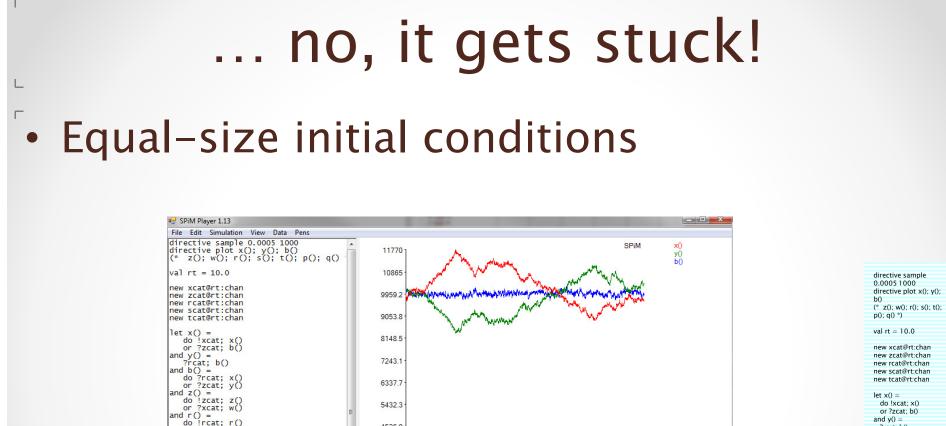




It works better than the original?!?



0.0005 1000 directive plot x(); y(); b() (* 20; w(); r(); s0; t0; p(); q0 *) val rt = 10.0 new xcat@rt:chan new rcat@rt:chan new rcat@rt:chan new tcat@rt:chan new	$0.0005 1000^{-1}$ directive plot x(); y(); b() (* 2(); w(); r(); s(); t(); p(); q() *) val rt = 10.0 new xcat@rt:chan new zcat@rt:chan new zcat@rt:chan new scat@rt:chan new scat@rt:chan new tcat@rt:chan new t	directive sample	
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 cat@rt:chan new tcat@rt:chan new tcat@rt:chan nd b() = do lzcat; x() or ?zcat; w() and z() = do lzcat; r() or ?zcat; w() and s() = lscat(); s() and w() = ?scat; r() new tcat new	directive plot x(); y(); b() ( $c^*$ z(); w(); r(); s(); t(); p(); q() *) val rt = 10.0 new xcat@rt:chan new zcat@rt:chan new zcat@rt:chan new tcat@rt:chan new tcat@rt:chan nd y() = do lzcat; x() or ?zcat; y() and z() = do lzcat; z() or ?zcat; w() and r() = do lzcat; r() or ?tcat; p() and w() = ?scat; z() and w() = ?scat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 1000 of y() run 4500 of y()		
b) (* z0; w0; r0; s0; t0; p0; q0 *) val rt = 10.0 new xcat@rt:chan new zcat@rt:chan new cat@rt:chan new tcat@rt:chan new tcat@rt:chan let x() = do zcat; z() or ?zcat; z() and y() = ?rcat; b() and y() = ?rcat; v() and y() = do ?rcat; x() or ?zcat; v() and z() = do lzcat; z() or ?zcat; w() and z() = do lzcat; z() or ?zcat; w() and x() = lscat(); s() and y() = ?scat; z() and y() = ?scat; r() run 1000 of p() run 1000 of y() run 4500 of y()	b) (* z0; w0; r0; s0; t0; p0; q0 *) val rt = 10.0 new xcat@rt:chan new zcat@rt:chan new cat@rt:chan new tcat@rt:chan new tcat@rt:chan let x() = do zcat; z0) or ?zcat; x0) and y0 = ?rcat; b0) and y0 = ?rcat; b0) and y0 = ?rcat; v0) and y0 = do ?rcat; x0) or ?zcat; y0) and z0 = do !rcat; r0) or ?zcat; w0 and r0 = do !rcat; r0) or ?tcat; y0 and x0 = !scat0; s0) and w0 = ?scat; z0) and w0 = ?scat; z0) and y0 = !scat0; s0) and y0 = ?scat; z0) and y0 = ?scat; z0) and y0 = ?scat; r0) run 1000 of s0) run 1000 of p0) run 4500 of y()		
(* $z()$ ; w0; r0; s0; t0; p0; q0 *) val rt = 10.0 new xcat@rt:chan new zcat@rt:chan new rcat@rt:chan new tcat@rt:chan new tcat@rt:chan nd y0 = do lrcat; x0 or ?zcat; w0 and z() = do lrcat; r0 or ?zcat; w0 and s() = lscat(); s0 and w0 = ?scat; r0 and p0 = ?xcat; r0 nun 1000 of p0 run 1000 of p0 run 1000 of y0 run 4500 of y0	(* $z()$ ; w0; r0; s0; t0; p0; q0 *) val rt = 10.0 new xcat@rt:chan new zcat@rt:chan new rcat@rt:chan new tcat@rt:chan new tcat@rt:chan nd y0 = do lrcat; x0 or ?zcat; w0 and z() = do lrcat; r0 or ?zcat; w1 and r0 = do lrcat; r0 or ?zcat; w1 and s() = lscat(); s0 and w0 = ?scat; r0 and p0 = ?xcat; r0 run 1000 of s0 run 1000 of p0 run 1000 of y0 run 4500 of y()		
p(); q() *) val rt = 10.0 new xcat@rt:chan new zcat@rt:chan new cat@rt:chan new scat@rt:chan new tcat@rt:chan new tcat@rt:new tcat	p(); q() *) val rt = 10.0 new xcat@rt:chan new zcat@rt:chan new cat@rt:chan new scat@rt:chan new scat@rt:chan new tcat@rt:chan let x() = do !xcat; x() or ?zcat; b() and y() = ?rcat; t() and y() = do ?rcat; x() or ?zcat; y() and z() = do !zcat; z() or ?zcat; y() and z() = do !zcat; z() or ?zcat; y() and z() = lotat; r() or ?zcat; y() and y() = lotat; z() and y() = !lscat(); s() and y() = !lscat(); s() and y() = ?scat; z() and y() = ?scat; r() run 1000 of s() run 1000 of p() run 1000 of y() run 4500 of y()		
val rt = 10.0 new xcat@rt:chan new zcat@rt:chan new rcat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan let x() = do lxcat; x() or ?zcat; b() and y() = ?rcat; b() and b() = do lzcat; z() or ?zcat; w() and z() = do lzcat; z() or ?zcat; w() and s() = lscat(); s() and w() = ?scat; z() and y() = lscat(); s() and y() = ?xcat; t() and y() = ?xcat; t() and y() = ?xcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 1000 of y() run 4500 of y()	val rt = 10.0 new xcat@rt:chan new zcat@rt:chan new rcat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan let x() = do lxcat; x() or ?zcat; b() and b() = ?rcat; b() and b() = do lzcat; z() or ?zcat; w() and z() = do lzcat; z() or ?zcat; w() and s() = lscat(); s() and w() = ?scat; z() and s() = ltcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of p() run 4500 of y()		
new xcat@rt:chan new xcat@rt:chan new rcat@rt:chan new rcat@rt:chan new scat@rt:chan new tcat@rt:chan new tcat@rt:chan let x() = do lxcat; x() or ?zcat; b() and y() = ?rcat; b() and y() = do ?rcat; x() or ?zcat; y() and z() = do lzcat; z() or ?zcat; v() and s() = lscat(); s() and y() = ?scat; z() and y() = ?scat; z() and y() = ?scat; t() and y() = ?scat; t() and y() = ?tcat; t() ?tcat; t() ?tc	new xcat@rt:chan new zcat@rt:chan new zcat@rt:chan new zcat@rt:chan new scat@rt:chan new scat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan no r?zcat; b() and b() = do ?rcat; v() and z() = do !zcat; z() or ?xcat; v() and s() = !scat(); s() and w() = ?scat; z() and w() = ?scat; z() and p() = ?tcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of p() run 4500 of y()		
new zcat@rt:chan new rcat@rt:chan new scat@rt:chan new scat@rt:chan new scat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan new tcat@rt:chan do !zcat; b() and y() = do ?rcat; b() and y() = do ?rcat; v() and z() = do !zcat; z() or ?zcat; v() and r() = do !zcat; r() or ?tcat; p() and s() = !scat; z() and y() = ?scat; z() and y() = ?scat; z() and y() = ?scat; t() and y() = ?scat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 4500 of y()	new zcat@rt:chan new rcat@rt:chan new scat@rt:chan new scat@rt:chan new tcat@rt:chan new tcat@rt:chan let x() = do lxcat; x() or ?zcat; b() and y() = ?rcat; b() and y() = do ?rcat; x() or ?zcat; y() and z() = do lzcat; z() or ?zcat; w() and r() = do lzcat; r() or ?tcat; p() and s() = lscat(); s() and w() = ?scat; z() and y() = ?scat; z() and y() = ?scat; t() and y() = ?scat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 4500 of y()	val rt = 10.0	
new rcat@rt:chan new scat@rt:chan new tcat@rt:chan let x() = do lxcat; x() or ?zcat; b() and y() = ?rcat; b() and b() = do ?rcat; x() or ?zcat; x() or ?zcat; v() and z() = do lzcat; z() or ?xcat; w() and r() = do lzcat; r() or ?xcat; v() and s() = lscat(); s() and w() = ?scat; z() and y() = ?kcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of p() run 4500 of y()	new rcat@rt:chan new scat@rt:chan new tcat@rt:chan let x() = do lxcat; x() or ?zcat; b() and y() = ?rcat; b() and b() = do ?rcat; x() or ?zcat; x() or ?zcat; v() and z() = do lzcat; z() or ?xcat; w() and r() = do lzcat; r() or ?xcat; p() and s() = lscat(); s() and w() = ?scat; z() and y() = ?kcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of p() run 4500 of y()		
new scat@rt:chan new tcat@rt:chan let x() = do !xcat; x() or ?zcat; b() and y() = ?rcat; b() and b() = do ?rcat; v() and z() = do !zcat; z() or ?zcat; w() and r() = do !zcat; z() or ?zcat; w() and r() = locat; r() or ?tcat; p() and s() = !scat(): s() and w() = ?scat; z() and w() = ?scat; z() and p() = ?tcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 4500 of y()	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 ?zcat; w() and r() = do !rcat; r() or ?tcat; p() and s() = !scat(): s() and w() = ?scat; z() and w() = ?scat; z() and y() = !tcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 4500 of y()	new zcat@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() = !lscat(); s() and y() = ?scat; z() and y() = ?tcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of p() run 4500 of y()	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; v() and r() = do !rcat; r() or ?tcat; p() and s() = !lscat(); s() and w() = ?scat; z() and y() = !lscat(); s() and y() = ?tcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of p() run 4500 of y()	new rcat@rt:chan	
let $x() =$ do 1xcat; $x()$ or 7zcat; $b()$ and $y() =$ 7rcat; $b()$ and $b() =$ do 7rcat; $x()$ or 7zcat; $y()$ and $z() =$ do 1zcat; $z()$ or 7xcat; $w()$ and $r() =$ do 1rcat; $r()$ or 7xcat; $p()$ and $s() =$ 1scat(); $s()$ and $w() =$ 7scat; $z()$ and $y() =$ 1scat(); $s()$ and $y() =$ r(z) r(z	let $x() =$ do 1xcat; $x()$ or 7zcat; $b()$ and $y() =$ 7rcat; $b()$ and $b() =$ do 7rcat; $x()$ or 7zcat; $y()$ and $z() =$ do 1zcat; $z()$ or 7zcat; $w()$ and $r() =$ do 1rcat; $r()$ or 7zcat; $p()$ and $s() =$ 1scat(); $s()$ and $w() =$ 7scat; $z()$ and $y() =$ 1scat(); $s()$ and $y() =$ r(z) r(z	new scat@rt:chan	
do ixcat; x() or ?zcat; b() and y() = ?rcat; b() and b() = do ?rcat; x() or ?zcat; y() and z() = do !zcat; z() or ?zcat; w() and r() = do !zcat; z() or ?zcat; w() and s() = !scat(); s() and w() = ?scat; z() and w() = ?scat; z() and p() = ?kcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of p() run 4500 of y()	do ixcat; x() or ?zcat; b() and y() = ?rcat; b() and b() = do ?rcat; x() or ?zcat; y() and z() = do !zcat; z() or ?zcat; w() and r() = do !zcat; r() or ?zcat; w() and s() = !scat(); s() and w() = ?scat; z() and y() = ?kcat; r() itcat; t() and p() = ?kcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 4500 of y()	new tcat@rt:chan	
do ixcat; x() or ?zcat; b() and y() = ?rcat; b() and b() = do ?rcat; x() or ?zcat; y() and z() = do izcat; z() or ?zcat; w() and r() = do izcat; r() or ?xcat; r() or ?tcat; p() and s() = iscat(); s() and w() = ?scat; z() and t() = itcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of p() run 4500 of y()	do ixcat; x() or ?zcat; b() and y() = ?rcat; b() and b() = do ?rcat; x() or ?zcat; y() and z() = do izcat; z() or ?zcat; y() and r() = do izcat; r() or ?xcat; y() and s() = iscat(); s() and s() = iscat(); s() and y() = ?zcat; z() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of p() run 4500 of y()	lat()	
or 72cat; b0 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 p() run 10000 of y() run 4500 of y()	or 72cat; b() and y() = ?rcat; b() and b() = do ?rcat; x() or ?zcat; y() and z() = do !rcat; 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 t() = !tcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 4500 of y()		
and y() = 7rcat; b() and b() = do ?rcat; x() or ?zcat; y() and z() = do !zcat; z() or ?xcat; w() and r() = do !rcat; r() or ?rcat; p() and s() = !scat(); s() and w() = ?scat; z() and y() = !tcat; t() and p() = ?xcat; r() Itcat; r() Itcat; r() Itcat; r() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of p() run 4500 of y()	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 y() = !tcat; r() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 10000 of p() run 10000 of p() run 10000 of p() run 4500 of y()	or 27cat: b()	
?rcat; b()         and b() =         do ?rcat; x()         or ?zcat; y()         and z() =         do !zcat; z()         or ?xcat; w()         and r() =         do !zcat; z()         or ?xcat; w()         and r() =         ido !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 10000 of p()         run 10000 of z()         run 4500 of y()	?rcat; b()         and b() =         do ?rcat; x()         or ?zcat; y()         and z() =         do !zcat; z()         or ?xcat; w()         and r() =         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 10000 of p()         run 10000 of z()         run 4500 of y()		
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; r() and w() = ?scat; z() and w() = ?scat; z() and t() = !tcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 10000 of p() run 10000 of p() run 4500 of y()	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; r() and w() = ?scat; z() and w() = ?scat; z() and t() = !tcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 10000 of p() run 10000 of p() run 4500 of y()		
do 7rcat; x() or 7zcat; y() and z() = do 1zcat; z() or 7xcat; w() and r() = do 1rcat; r() or 7tcat; p() and s() = 1scat(); s() and w() = 7scat; z() and t() = 1tcat; t() and p() = 7xcat; r() run 1000 of s() run 1000 of p() run 1000 of p() run 1000 of z() run 4500 of y()	do 7rcat; x0 or 7zcat; y0 and z0 = do 1zcat; z0 or 7xcat; w0 and r0 = do 1rcat; r0 or 7tcat; p0 and s0 = 1scat(); s0 and w0 = 7scat; z0 and t0 = 1tcat; t0 and p0 = 7xcat; r0 run 1000 of s0 run 1000 of p0 run 1000 of z0 run 4500 of y0		
or ?zcat; y0 and z() = do !zcat; z0 or ?xcat; w() and r0 = do !ncat; r0 or ?tcat; p0 and s0 = !scat(); s0 and w0 = ?scat; z() and t0 = !tcat; t0 and p() = ?xcat; r() run 1000 of s0 run 1000 of p() run 10000 of p() run 10000 of p() run 10000 of y() run 4500 of y()	or ?zcat; y0 and z() = do !zcat; z0 or ?xcat; w() and r() = do !ncat; 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 s() run 10000 of p() run 10000 of p() run 10000 of p() run 10000 of y()		
and z() = do lzcat; 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 p() run 10000 of p() run 10000 of z() run 4500 of y()	and z() = do lzcat; 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 p() run 10000 of z() run 4500 of y()		
do Izcat; z0 or ?xcat; w() and r() = do Ircat; r() or ?tcat; p0 and s() = Iscat(); s() and w() = ?scat; z() and t() = Itcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of p() run 4500 of y()	do Izcat; z0 or ?xcat; w() and r() = do Ircat; r() or ?tcat; p0 and s() = Iscat(); s() and w() = ?scat; z() and t() = Itcat; t() and p() = ?xcat; r() run 1000 of s() run 10000 of p() run 10000 of p() run 10000 of p() run 4500 of y()		
or ?xcat; w() and r() = do !rcat; r() or ?tcat; p() and s() = !scat; c() and w() = ?ccat; z() and t() = !tcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 10000 of p() run 10000 of p() run 4500 of y()	or ?xcat; w() and r() = do !rcat; r() or ?tcat; p() and s() = !scat; s() and w() = ?ccat; z() and t() = !tcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 10000 of p() run 10000 of p() run 4500 of y()		
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 p() run 10000 of p() run 10000 of z() run 4500 of y()	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 p() run 10000 of p() run 10000 of p() run 4500 of y()	00 12Cat; 2()	
do ircat; r() or ?tcat; p() and s() = Iscat(); s() and w() = ?scat; z() and t() = Itcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of p() run 10000 of p() run 10000 of p() run 4500 of y()	do ircat; r() or ?tcat; p() and s() = Iscat(); s() and w() = ?scat; z() and t() = Itcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of p() run 10000 of p() run 10000 of p() run 4500 of y()	OF (XCal, W()	
or ?tcat; p0 and s() =  scat(); s() and w() = ?scat; z() and t() =  tcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 10000 of p() run 10000 of z() run 4500 of y()	or ?tcat; p0 and s() = lscat(); s() and w() = ?scat; z() and t() = ltcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 10000 of p() run 10000 of z() run 4500 of y()		
and s() = !scat(); s() and w() = ?scat; z() and t() = !tcat; t() and p() = ?xcat; r() run 1000 of s() run 10000 of t() run 10000 of p() run 10000 of p() run 10000 of y() run 4500 of y()	and s() = !scat(); s() and w() = ?scat; z() and t() = !tcat; t() and p() = ?xcat; r() run 1000 of s() run 10000 of t() run 10000 of p() run 10000 of p() run 10000 of y() run 4500 of y()		
Iscat(); s() and w() = ?scat; z() and t() = Itcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 10000 of p() run 10000 of p() run 4500 of y()	Iscat(); s() and w() = ?scat; z() and t() = Itcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of p() run 4500 of y()		
and w() = ?scat; z() and t() = !tcat; t() and p() = ?xcat; r() run 1000 of s() run 10000 of t() run 10000 of p() run 10000 of p() run 10000 of y() run 4500 of y()	and w() = ?scat; z() and t() = ltcat; t() and p() = ?xcat; r() run 1000 of s() run 10000 of p() run 10000 of p() run 10000 of p() run 4500 of y()		
?scat; z() and t() = ltcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of p() run 10000 of z() run 4500 of y()	?scat; z() and t() = ltcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of p() run 10000 of z() run 4500 of y()		
and t() = ltcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 10000 of p() run 10000 of z() run 4500 of y()	and t() = ltcat; t() and p() = ?xcat; r() run 1000 of s() run 1000 of p() run 10000 of p() run 10000 of z() run 4500 of y()		
Itcat; 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()	Itcat; 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()		
and p() = ?xcat; r() run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of z() run 4500 of y()	and p() = ?xcat; r() run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of z() run 4500 of y()		
7xcat; r() run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of z() run 4500 of y()	7xcat; r() run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of z() run 4500 of y()		
run 1000 of s() run 1000 of t() run 10000 of p() run 10000 of z() run 4500 of y()	run 1000 of s0 run 1000 of t0 run 10000 of p0 run 10000 of z0 run 4500 of y()		
run 1000 of t() run 10000 of p() run 10000 of z() run 4500 of y()	run 1000 of t() run 10000 of p() run 10000 of z() run 4500 of y()	?xcat; r()	
run 10000 of p() run 10000 of z() run 4500 of y()	run 10000 of p() run 10000 of z() run 4500 of y()	run 1000 of s()	
run 10000 of z() run 4500 of y()	run 10000 of z() run 4500 of y()	run 1000 of t()	
run 10000 of z() run 4500 of y()	run 10000 of z() run 4500 of y()	run 10000 of n()	
run 4500 of y()	run 4500 of y()		
run 4500 of y() run 5500 of x()	run 4500 of y() run 5500 of x()		
run 5500 of x()	run 5500 of x()	run 4500 of y()	
		run 5500 of x()	



5432.3

4526.9 3621.5

2716.2

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

!tcat; t()

and t() =

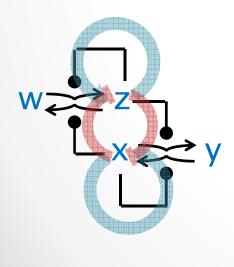
and $p() =$						
?xcat; r()	1810.8					
run 10000 of s() run 10000 of t()	905.38					
run 10000 of w() run 10000 of z() run 10000 of p()	0;				0.0012958	
run 10000 of r()	Circulations Us	Hed Time = 0.0012	96 (2592 points at 4.9868)	- 06 -:		Plotting: Paused
run 10000 of v()		inted, 11me - 0.00128	90 (2592 points at 4.9000	e-oo sim rime/sys rime)		۱.
run 10000 of y() run 10000 of b() run 10000 of x()					57	
		• •				
P						
						11.

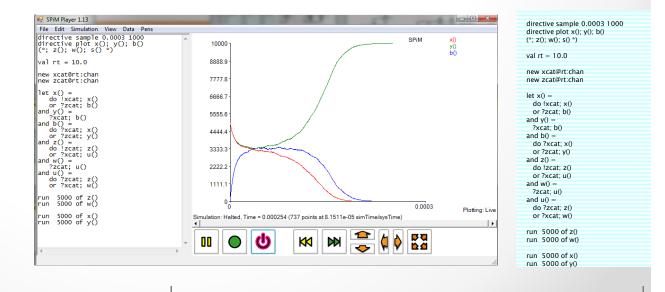
uncenve sumpre
0.0005 1000
directive plot x0; y0;
b0
(* z(); w(); r(); s(); t();
p(); q() *)
val rt = 10.0
funct fore
new xcat@rt:chan
new zcat@rt:chan
new rcat@rt:chan
new scat@rt:chan
new tcat@rt:chan
new (cat@rt.chan
et x() =
do !xcat; x()
or ?zcat; b()
and y() =
?rcat; b()
and b() =
do ?rcat; x()
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 r()
run 10000 of y()
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.

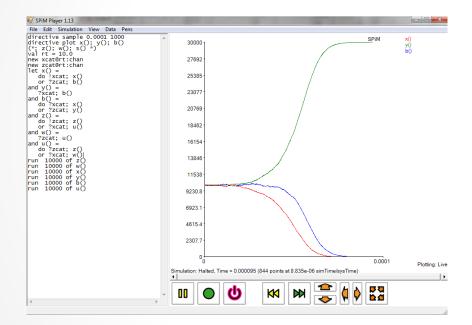




#### • Equal-size intial conditions

Г

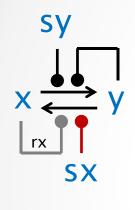
L



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 |et x0| =do !xcat: x() or ?zcat; b() and y() =?xcat; b() and b0 =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()
- do !xcat; x() or ?ycat; y() or ?sycat; 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()

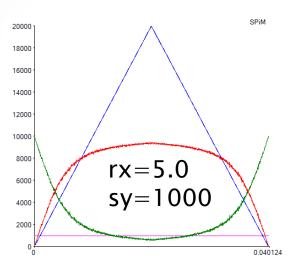
- let clock(p:proc(int), t:float) =
   (\* Produce one p(m) every t sec with precision dt,
   with m incremented from 0 \*)
   (val dt= 100.0 run step(p, 0, t, dt, dt))
   and step(o:proc(int), mint. t:float, n:float, dt:float) =
- and step(p:proc(int), m:int, t:float, n:float, dt:float) if n<=0.0 then (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 ()

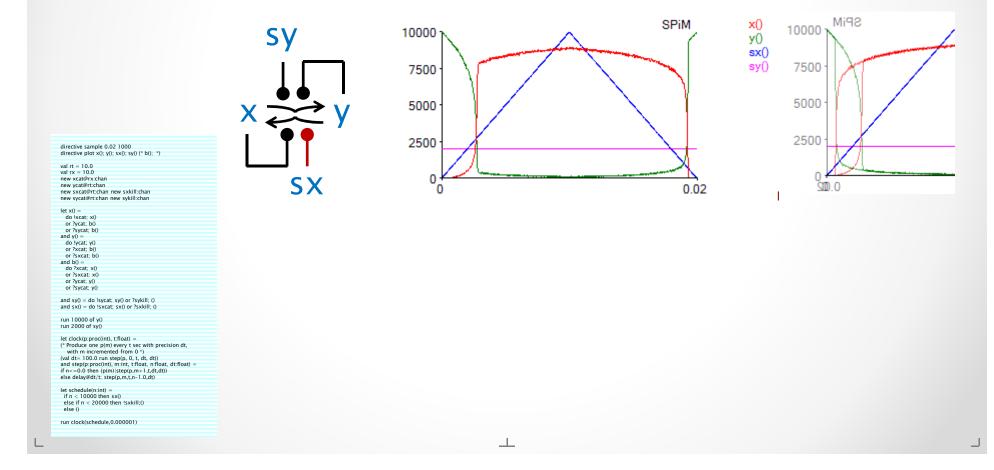
run clock(schedule,0.000001)



#### **One-Input Switches**

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

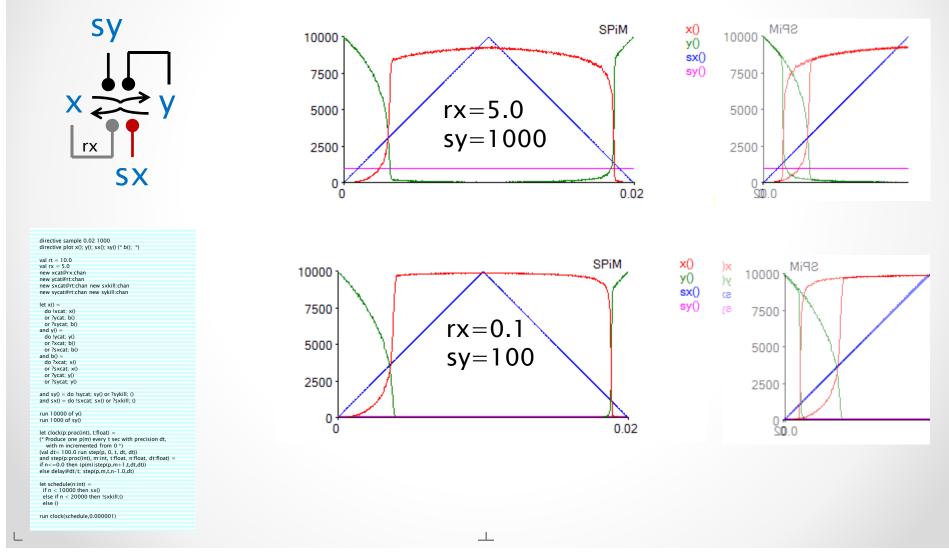
for switch-back.

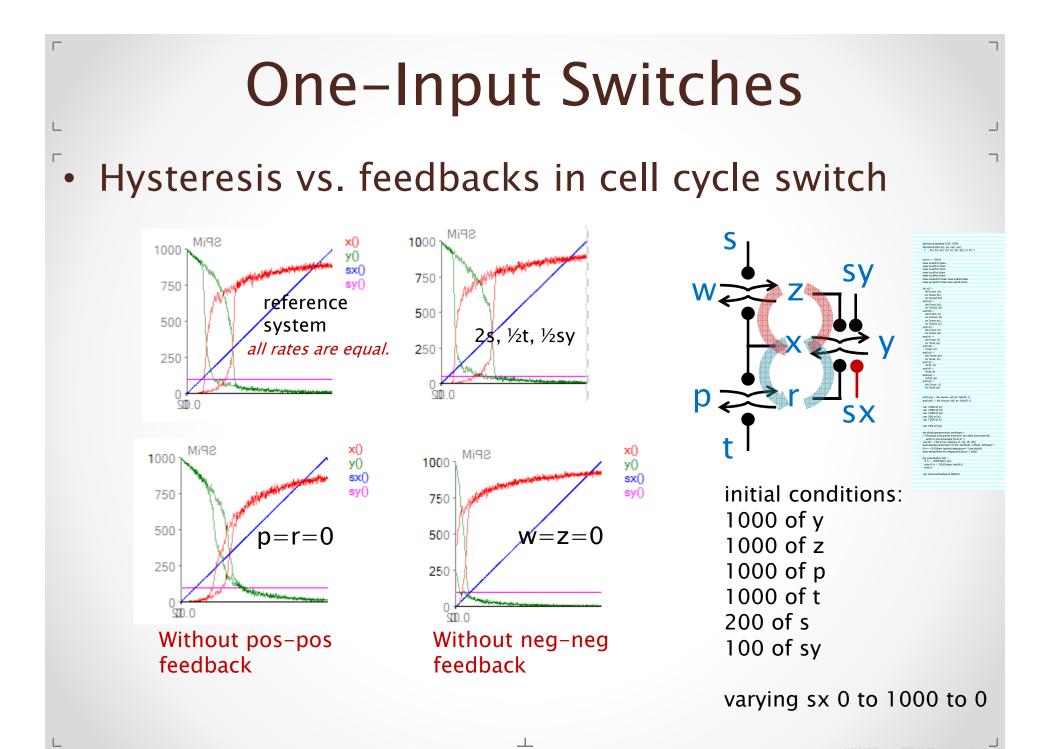


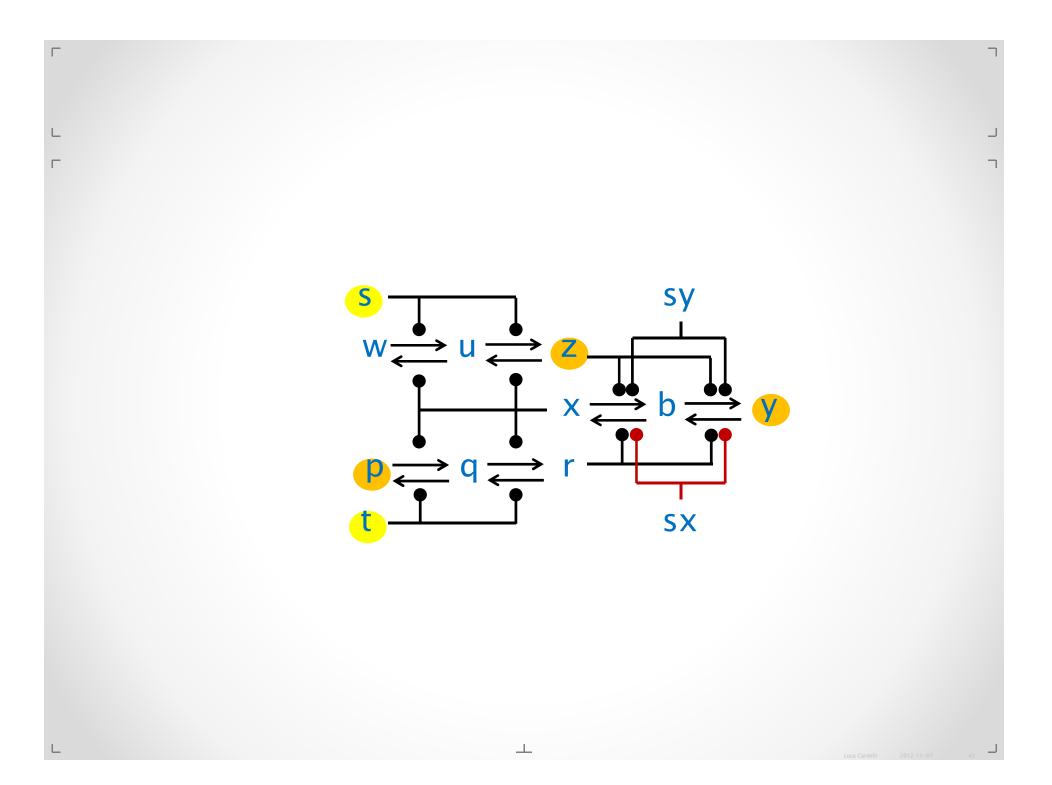
#### **One-Input Switches**

#### Hysteresis in biased AM-like switches

 $\square$ 

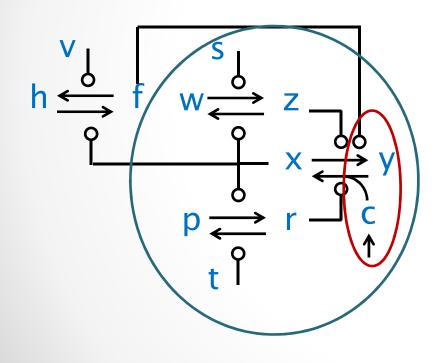


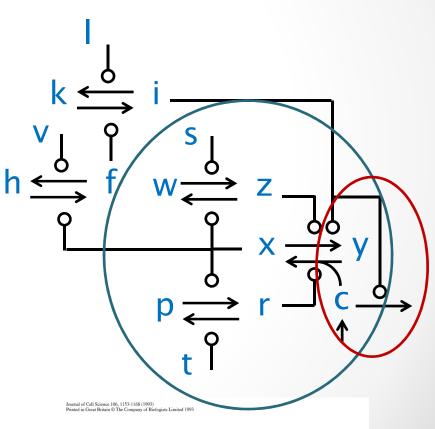




#### Ferrell oscillator

#### Novak-Tyson oscillator



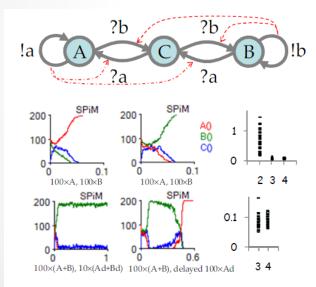


Numerical analysis of a comprehensive model of M-phase control in Xenopus oocyte extracts and intact embryos

Bela Novak\* and John J. Tyson<sup>†</sup> 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 + AC + B -> B + B

#### Artificial Biochemistry. Luca Cardelli

exception 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.

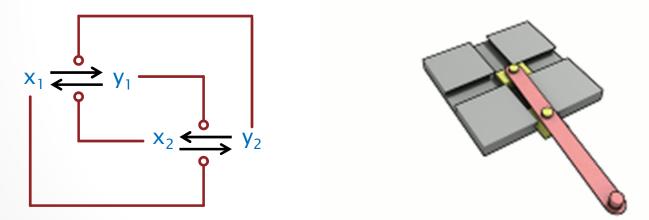
SY X Y SX

## Oscillators

### The Trammel of Archimedes

#### A device to draw ellipses

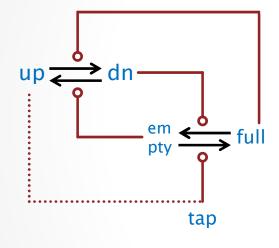
- Two interconnected switches.
- When one switch is on (off) it flips the other switch on (off). When the other switch is on (off) it flips the first switch off (on).



en.wikipedia.org/wiki/Trammel\_of\_Archimedes

### The Shishi Odoshi

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



empty + tap  $\rightarrow$  tap + full up + full  $\rightarrow$  full + dn full + dn  $\rightarrow$  dn + empty dn + empty  $\rightarrow$  empty + up

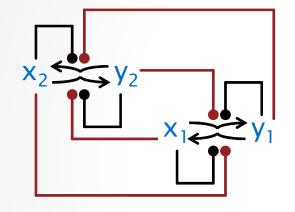


http://www.youtube.com/watch?v=VbvecTIftcE&NR=1&feature=fvwp

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

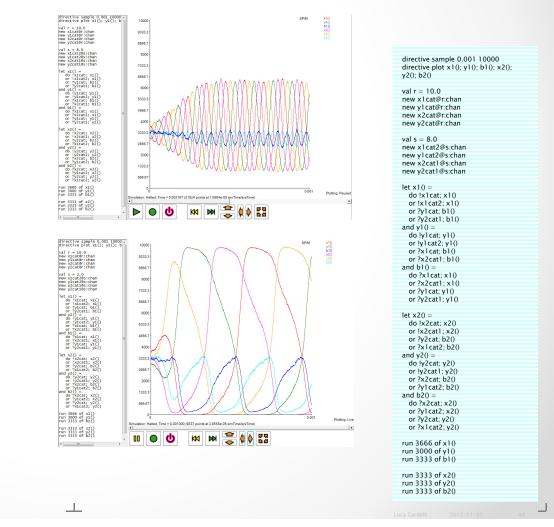
### The 2AM Limit-Cycle Oscillator

#### Two AM switches in a Trammel pattern

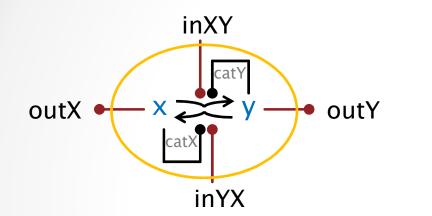


The red reactions need to be slower (even slightly) than the black reactions, but otherwise the oscillation is robust. Oscillation stops at 10 vs. 10 and 1 vs. 10. Here the rates are 8(red) vs 10(black) top, and 2 vs 10, bottom.

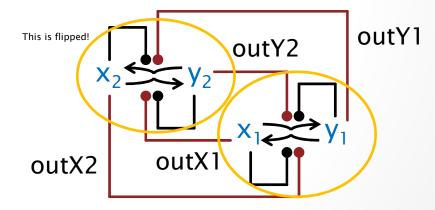
(Simple limit-cycle oscillators in the literature have very critical rate ranges.)



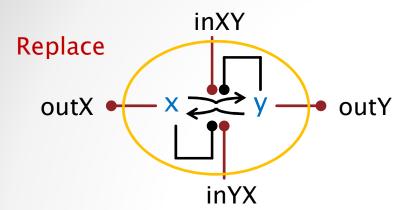
### The Switch Module



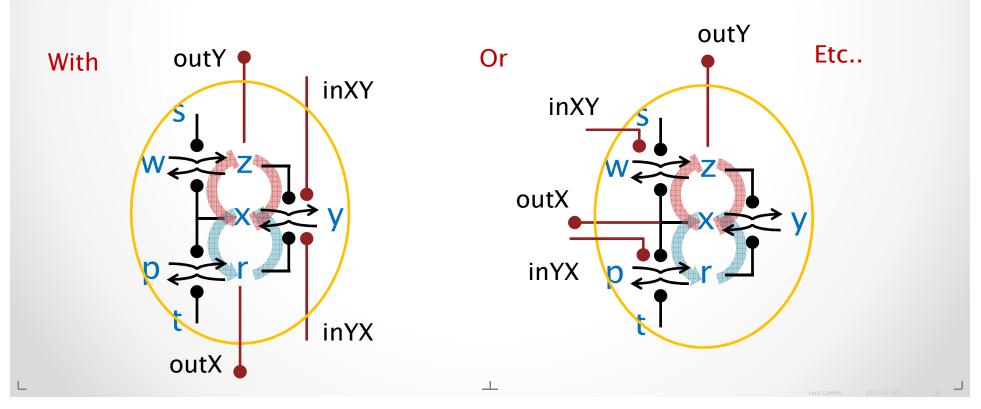
Г



### **Replacing Switch Modules**

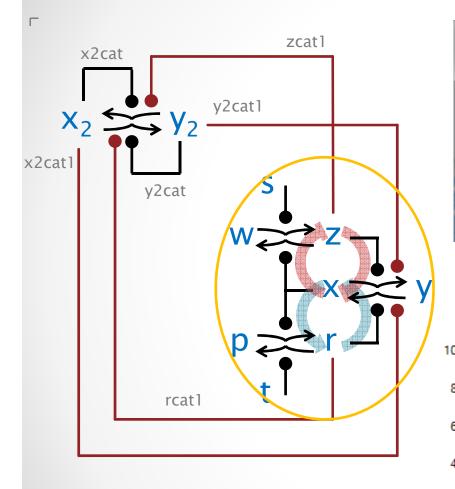


 $\square$ 

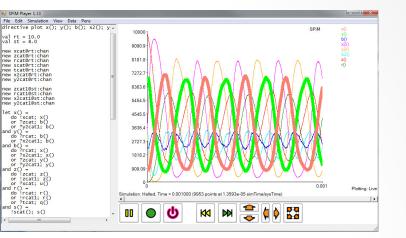


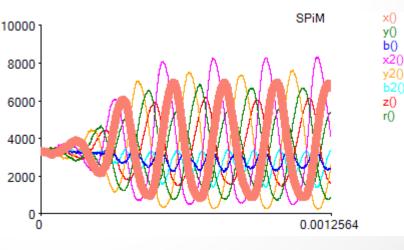
### **Modified Oscillator 1**

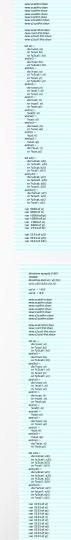
do or



Г





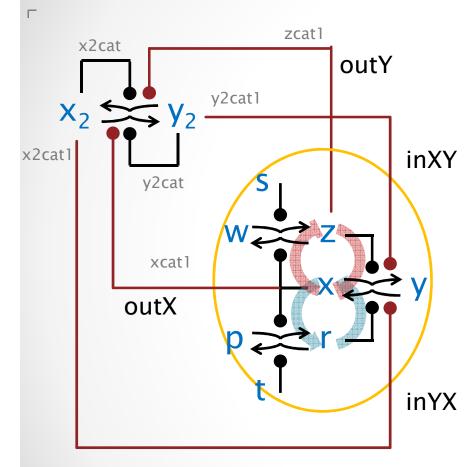


run 3333 of x20 run 3333 of y20 run 3333 of b20

directive sample 0.001 10000 directive plot x(); y(); b() x2(); y2(); b2(); z(); r()

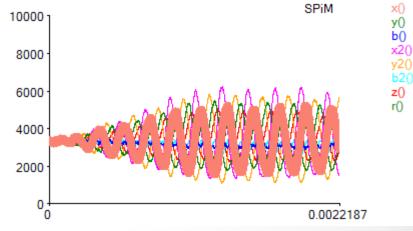
val nt = 10.0 val st = 8.0

### **Modified Oscillator 2**



Г





 $\label{eq:constraint} \begin{array}{l} et x_i l \rangle = \\ do to a \\ do$ let x20 = do b2zat x20 er k7zat ; x2 er y7zat ; x2 er y7zat ; b2 and y20 er y7zat ; b2 and y20 er y7zat ; b2 and b20 er y7zat ; b2 and b20 er y7zat ; b2 and b20 er y7zat ; b2 er y7z

F

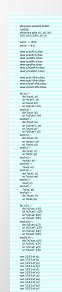
F

X()

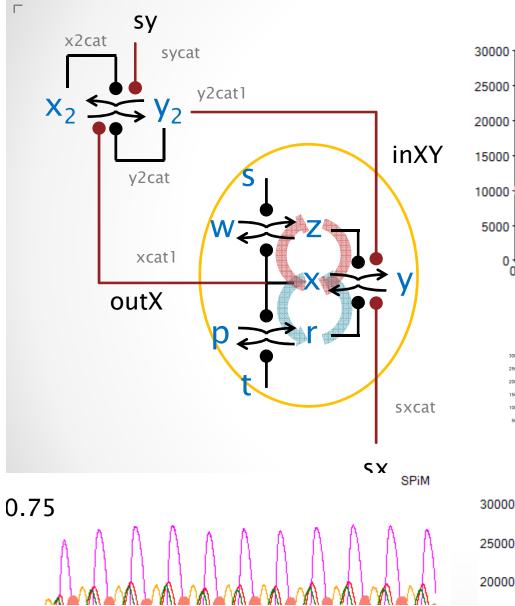
directive plot x(); y(); x2(); y2(); b2(); z(); r()

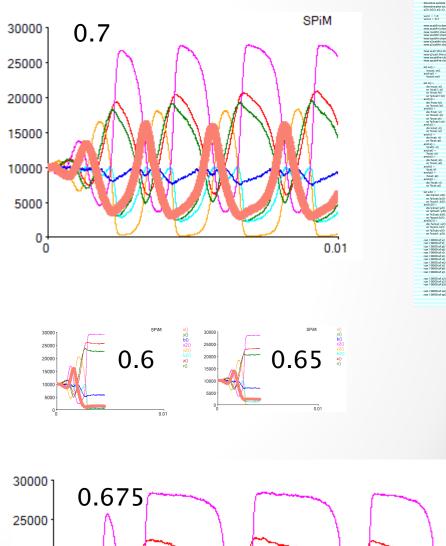
val rt = 10.0 val st = 8.0

new xcatiërt cha new zcatiërt cha new rcatiërt cha new scatiërt cha new tcatiërt cha new x2 catiërt cha



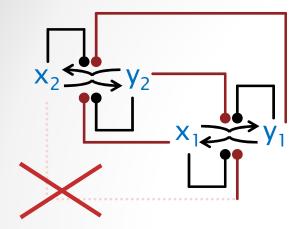
### Modified Oscillator 3





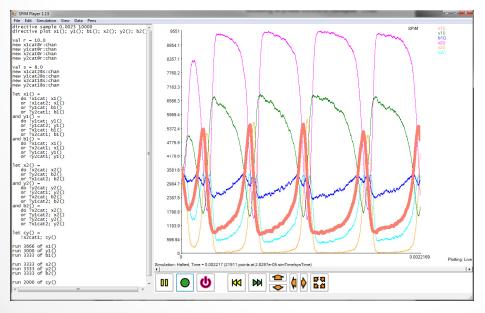
SPiM

#### **Constant-Influx Oscillator**



Г

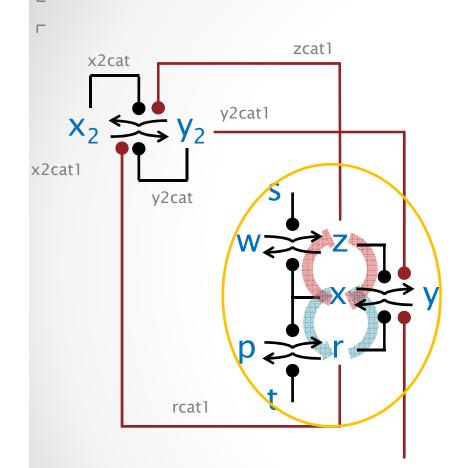
As in the Shishi Odoshi (and the cell cycle)

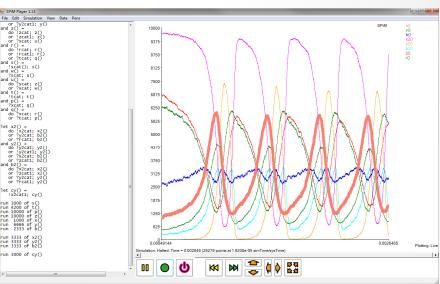


dimention and a 0.001 1.0000
directive sample 0.001 10000
directive plot x1(); y1(); b1(); x2();
y20; b20
und n 100
val r = 10.0
new x1cat@r:chan
new y1cat@r:chan
new x2cat@r:chan
new y2cat@r:chan
und n 0.0
val s = 8.0
new x1cat2@s:chan
new y1cat2@s:chan
new x2cat1@s:chan
new y2cat1@s:chan
new yzcaci @s.chan
et x1()  =
do !x1cat; x1()
or !x1cat2; x1()
or ?y1cat; b1()
or ?y2cat1; b1()
51. j = cut1, b1()
and y1() =
do !y1cat; y1()
or !y1cat2; y1()
or ?x1cat; b1()
or ?x2cat1; b1()
and b1() =
de Del este el O
00 (XTCal; XT()
do ?x1cat; x1() or ?x2cat1; x1()
an Dulentuul()
or ?y1cat; y1()
or ?y2cat1; y1()
let x2() =
do !x2cat; x2()
u0 1x2cat, x2()
or ?y2cat; b2()
or ?x1cat2; b2()
and y2() =
do !y2cat; y2()
40., j 2 cut, j 20
or !y2cat1; y2()
or !y2cat I ; y2() or ?x2cat: b2()
or !y2cat1; y2() or ?x2cat; b2()
or ?y1cat2; b2()
or ?y1cat2; b2()
or ?y1cat2; b2() and b2() =
or ?y1cat2; b2() and b2() =
or ?y1cat2; b2() and b2() =
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y1cat2; x2()
or ?y1 cat2; b2() and b2() = do ?x2 cat; x2() or ?y1 cat2; x2() or ?y2 cat; y2()
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y1cat2; x2()
or ?y1 cat2; b2() and b2() = do ?x2 cat; x2() or ?y1 cat2; x2() or ?y2 cat; y2()
or ?y1 cat2; b2() and b2() = do ?x2cat; x2() or ?y1 cat2; x2() or ?y2cat; y2() or ?x1 cat2; y2()
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y1cat2; x2() or ?y1cat2; x2() or ?x1cat2; y2() let cy() =
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y1cat2; x2() or ?y1cat2; x2() or ?x1cat2; y2() let cy() =
or ?y1 cat2; b2() and b2() = do ?x2cat; x2() or ?y1 cat2; x2() or ?y2cat; y2() or ?x1 cat2; y2()
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y1cat2; x2() or ?y1cat2; y2() or ?x1cat2; y2() let cy() = !x2cat1; cy()
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y1cat2; x2() or ?y1cat2; y2() or ?x1cat2; y2() let cy() = !x2cat1; cy()
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y2cat2; x2() or ?y2cat2; x2() or ?x1cat2; y2() let cy() = lx2cat1; cy() run 3666 of x1()
or 7y1cat2; b2() and b2() = do 7x2cat; x2() or 7y1cat2; x2() or 7x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3000 of y1()
or 7y1cat2; b2() and b2() = do 7x2cat; x2() or 7y1cat2; x2() or 7x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3000 of y1()
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y2cat2; x2() or ?y2cat2; x2() or ?x1cat2; y2() let cy() = lx2cat1; cy() run 3666 of x1()
or ?ylcat2; b2() and b2() = do ?x2cat; x2() or ?ylcat2; x2() or ?ylcat2; x2() or ?xlcat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3060 of y1() run 3333 of b1()
or 7y1cat2; b2() and b2() = do 7x2cat; x2() or 7y1cat2; x2() or 7x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3000 of y1()
or 7y1cat2; b2() and b2() = do 7x2cat; x2() or 7y1cat2; x2() or 7y2cat; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3366 of y1() run 3333 of b1() run 3333 of x2()
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y1cat2; x2() or ?y1cat2; x2() or ?x1cat2; y2() let cy() = lx2cat1; cy() run 3666 of x1() run 3333 of x2() run 3333 of x2() run 3333 of y2()
or 7y1cat2; b2() and b2() = do 7x2cat; x2() or 7y1cat2; x2() or 7y2cat; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3366 of y1() run 3333 of b1() run 3333 of x2()
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y1cat2; x2() or ?y1cat2; x2() or ?x1cat2; y2() let cy() = lx2cat1; cy() run 3666 of x1() run 3333 of x2() run 3333 of x2() run 3333 of y2()
or ?ylcat2; b2() and b2() = do ?x2cat; x2() or ?ylcat2; x2() or ?ylcat2; x2() or ?x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3333 of b1() run 3333 of b2() run 3333 of b2()
or ?y1cat2; b2() and b2() = do ?x2cat; x2() or ?y1cat2; x2() or ?y1cat2; x2() or ?x1cat2; y2() let cy() = lx2cat1; cy() run 3666 of x1() run 3333 of x2() run 3333 of x2() run 3333 of y2()
or ?ylcat2; b2() and b2() = do ?x2cat; x2() or ?ylcat2; x2() or ?ylcat2; x2() or ?x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3333 of b1() run 3333 of b2() run 3333 of b2()
or ?ylcat2; b2() and b2() = do ?x2cat; x2() or ?ylcat2; x2() or ?ylcat2; x2() or ?x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3333 of b1() run 3333 of b2() run 3333 of b2()
or ?ylcat2; b2() and b2() = do ?x2cat; x2() or ?ylcat2; x2() or ?ylcat2; x2() or ?x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3333 of b1() run 3333 of b2() run 3333 of b2()
or ?ylcat2; b2() and b2() = do ?x2cat; x2() or ?ylcat2; x2() or ?ylcat2; x2() or ?x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3333 of b1() run 3333 of b2() run 3333 of b2()
or ?ylcat2; b2() and b2() = do ?x2cat; x2() or ?ylcat2; x2() or ?ylcat2; x2() or ?x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3333 of b1() run 3333 of b2() run 3333 of b2()
or ?ylcat2; b2() and b2() = do ?x2cat; x2() or ?ylcat2; x2() or ?ylcat2; x2() or ?x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3333 of b1() run 3333 of b2() run 3333 of b2()
or ?ylcat2; b2() and b2() = do ?x2cat; x2() or ?ylcat2; x2() or ?ylcat2; x2() or ?x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3333 of b1() run 3333 of b2() run 3333 of b2()
or ?ylcat2; b2() and b2() = do ?x2cat; x2() or ?ylcat2; x2() or ?ylcat2; x2() or ?x1cat2; y2() let cy() = !x2cat1; cy() run 3666 of x1() run 3333 of b1() run 3333 of b2() run 3333 of b2()

### **Constant** influx

cy() = !x2cat1:





Still working fine with the replaced switch.

let cy0 = b2cat1; cy0 nun 3000 of s0 nun 10000 of p0 nun 10000 of p0 nun 10000 of x0 nun 6666 of y0 nun 6666 of y0 run 3333 of x2() run 3333 of y2() run 3333 of b2()

### The Novak-Tyson Oscillator

#### • First switch

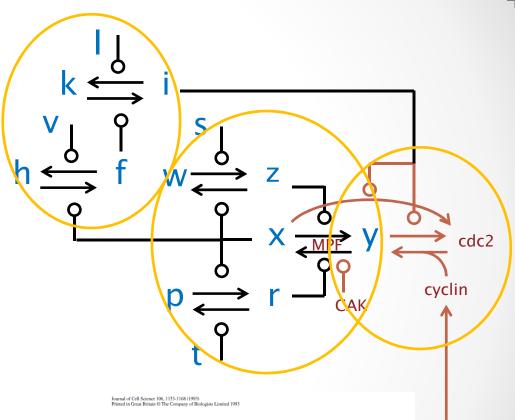
 Is the 'transformed' AM switch in one-input configuration (driven by constant influx of cyclin).

#### 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.



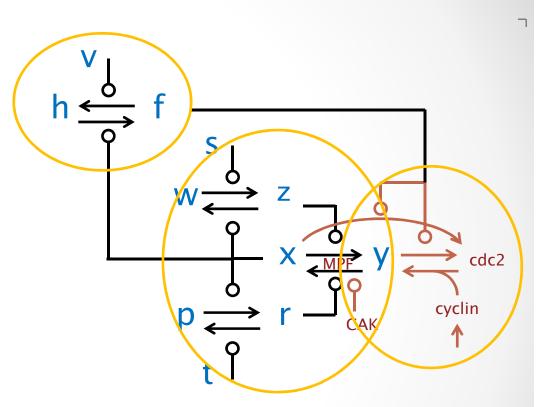
Numerical analysis of a comprehensive model of M-phase control in *Xenopus* oocyte extracts and intact embryos

Bela Novak\* and John J. Tyson<sup>†</sup> 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